Introduction to Data Analysis

Contents page

I II III IV V

21/12/2015

Introduction to Data Analysis

L1 Data Analysis Process

1.1 Introduction

Using data to answer questions like...
- → What makes students more likely to submit their projects?
- → What countries have the highest / lowest employment rates?
- → How does subway ridership vary by weather, time of day and location?

1.2 Problems solved by Data Analysis

- → Netflix - Provide movie recommendations
- → Facebook - Newsfeed ranking algorithm
- → OKCupid - Find good romantic matches

Facebook blog post: Exposure to diverse information on Facebook

OKCupid blog post: Best questions for a first date

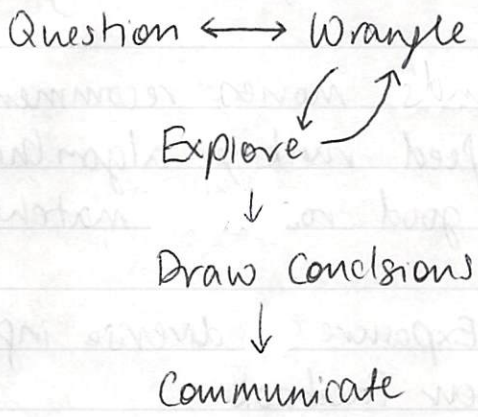
Walmart uses customer's purchase orders and media posts in order to know what to stock

Bill James used data to find current and predict future best performers in baseball

Pharmaceutical companies predict which compounds are likely to make effective drugs

1.3 Data Analysis Process

* Question Phase
  → question you want to answer
* Data Wrangling Phase
  1. Data aquisition
  2. Data cleaning
* Explore Phase
  → Build intuition
  → Find patterns
* Drawing Conclusions Phase (or make predictions)
  → Using stats and or machine learning
* Communication Phase



1.4 Downloading Anaconda, IPython Notebook & Downloading data files

To run: ipython notebook notebook_name.ipynb

1.5 Intro to CSVs

Data Acquisition
→ downloading files
→ accessing an API
→ scraping a webpage
→ combine data in different formats

Data format : CSV - Comma Separated Values

1.6 CSVs in Python

In Python CSV can be represented in a couple of ways :

① Each row is a list
⇒ CSV is a list of lists
⇒ CSV = [['A1', 'A2', 'A3'],
['B1', 'B2', 'B3']]

② Each row is a dictionary
⇒ CSV is a list of dictionaries
⇒ CSV = [{'name1': 'A1', 'name2': 'A2', 'name3': 'A3'},
{'name1': 'B1', 'name2': 'B2', 'name3': 'B3'}]

You can write the code to get the data in the csv files but there are libraries for doing that. This course uses the unicodecsv library.

```
import unicodecsv
enrollments = []
f = open('enrollments.csv', 'rb')
reader = unicodecsv.DictReader(f)
for row in reader:
    enrollments.append(row)

f.close

enrollments[0]
```

file is opened for reading

changes format for reading used for unicodecsv library

↑ An iterator rather than a list

iterator allows us to write a for loop to access each element but we can only do this once - subsequent for loops on elements will not be executed

Can shorten the code as follows

```
import unicodecs
# don't need to close the file
with open('enrollments.csv', 'rb') as f:
    reader = unicodecsv.DictReader(f)
    enrollments = list(reader)  <- don't need to loop over rows

enrollments[0]
```

Since we have to do this for several files we can write a function:

```
def read_csv(filename):
    with open(filename, 'rb') as f:
    reader = unicodecsv.DictReader(f)
    return list(reader)

enrollments = read_csv('enrollments.csv')
daily_engagement = read_csv('daily_engagement.csv')
project_submissions = read_csv('project_submissions.csv')
```

1.7 Fixing data types

All data types are taken as strings, these need to be fixed

1.8 Questions about student data

- → How long to submit projects?
- → How do students who pass differ from those who don't?

1.9 Investigating the data

→ number of rows in
enrollments
daily_engagement
project_submissions
→ number of unique account keys

1.10 Problems with data

→ More unique students in enrollments than the daily_engagement table: engagement table should include a row for each day that each student is enrolled, even if the student didn't visit the site at all that day. So there should be the same number of unique students in both tables

→ variables are not named consistently in the three tables i.e. 'account_key' in two tables and 'acct' in the other table

1.11 Missing engagement records

Why? We find some students who enrolled and cancelled the same day

① Identify surprising data points
in this case any enrollment record with no corresponding engagement records
② Print out one or a few of them

1.12 Checking for more problem records

③ Fix any problems you find

→ More investigation maybe necessary
→ Or there might not be a problem as in our case

1.13 Tracking down the remaining problems

After removing the records where the students joined and cancelled the same day we are left with 3 records which turnout to be test records

1.14 Refining the Question

Explore Phase o—o—●—o—o

Question: How do numbers in the daily_engagement table differ for students who pass the first project?

Revision: Only look at engagement from the first week, and exclude students who cancel within a week

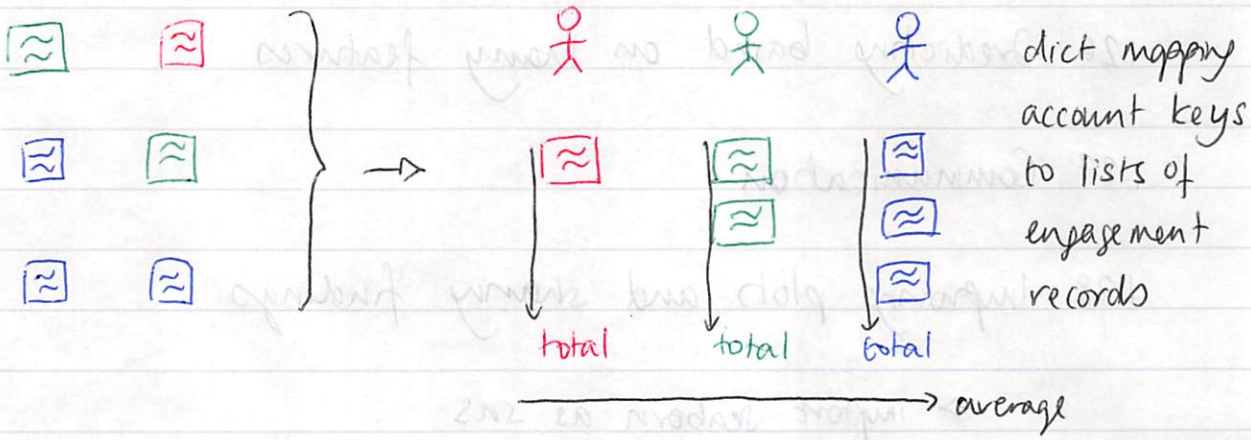
Problems:
1. Includes data from after the first project submission
2. Compares data from different lengths of time
3. Includes engagement data from courses not related to the first project.

1.15 Getting data from the first week

1.16 Indulge curiosity

1.17 Exploring Student Engagement

Average minutes spent in classroom



1.18 Debugging Data Analysis Code

We find that the maximum number of minutes spent in the classroom for any of the account keys is 10568.1008637 > total number of minutes in one week ⇒ something wrong

① Identify surprising data points
② Print out one or a few surprising points
③ Fix any problems you find

1.19 Lessons completed in first week

1.20 Number of visits in the first week

1.21 Splitting out passing students

1.22 Comparing the two student groups

1.23 Making histograms

1.24 Are our results just noise?

1.25 Correlation does not imply causation

1.26 Predicting based on many features

1.27 Communication

1.28 Improving plots and sharing findings

→ import seaborn as sns

makes all plots look nicer

1.29 Data analysis and related terms

1.30 Conclusion

I

II

III

IV

V

L2 NumPy and Pandas for 1D Data

2.1 Introduction

Data analysis libraries NumPy and Pandas
Code runs faster using these because implementations are in C

2.2 Gapminder Data

- Employment levels
- Life expectancy
- GDP
- School completion rates

Questions:
- How has employment in US varied over time?
- What are the highest and lowest employment levels?
  - which countries have them?
  - where is the US on the spectrum?
- Same questions for other variables
- How do these variables relate to each other?
- Are there consistent trends across countries?

2.3 1D Data in NumPy & Pandas

Using Pandas to load large datasets

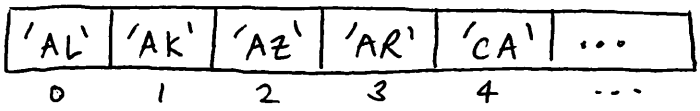
2.4 NumPy Arrays

One-dimensional data structures

Pandas Series → built on → NumPy (Numerical Python) Array
↳ more features
↳ simpler

Numpy Arrays and Python Lists:

Numpy Array:



Similarities	Differences
• Access elts by posn a[0] = 'AL' • Access a range of elts a[1:3] = ['AK' | 'AZ'] • Use loops for x in a	• Each elt should have same type (string, int, boolean, etc.) • Convenient functions mean(), std() • Can be multidimensional

2.5 Vectorized Operations

Vector Addition [1|2|3] + [1|2|3]

In NumPy '+' is used for vector addition as in linear algebra i.e. [2|4|6]

In Python '+' is used for list concatenation i.e. [1|2|3|1|2|3]

2.6 Scalar Multiplication [1|2|3] * 3

In NumPy '*' is used for scalar multiplication as in linear algebra i.e. [3|6|9]

In Python '*' is used to repeat or concatenate i.e. [1|2|3|1|2|3|1|2|3]

2.7 More vectorised operations:

Math operations		Logical operations		Comparison operations	
Add:	+	And:	&	Greater:	>
Subract:	-	Or:	|	Greater or equal:	>=
Multiply:	*	Not:	~	Less:	<
Divide:	/			Less or equal:	<=
Exponentiate:	**	(make sure the arrays contain booleans)		Equal:	==
				Not equal:	!=

2.8 Standardising data

How does one compare one data point to the rest?

Convert each data point to the number of standard deviations away from the mean

2.9 Numpy Index Arrays

a = | 1 | 2 | 3 | 4 | 5 | a[b] = | 3 | 4 | 5 |

b = | F | F | T | T | T | = a > 2 a[a>2]

2.10 + vs. +=

Code Snippet 1

```
import numpy as np
a= np.array([1,2,3,4])
b=a
a+= np.array([1,1,1,1])
print b
```

Output: array([2,3,4,5])

Code Snippet 2

```
import numpy as np
a= p.array([1,2,3,4])
b=a
a= a+ np.array([1,1,1,1])
print b
```

Output: array([1,2,3,4])

2.11 In-Place vs. Not In Place

+= operates in-place while + does not

Code Snippet

```
import numpy as np
a = np.array([1,2,3,4,5])
slice = a[:3]
slice[0] = 100
print a
```

} Output array([100,2,3,4,5])

2.12 Pandas Series

A series is similar to NumPy array, but with extra functionality e.g. s.describe() will print out the mean, standard deviation, median and other statistics of the series s.

Similarities:

- → Accessing elements — s[0], s[3:7]
- → Looping — for x in s
- → Convenient functions — s.mean(), s.max()
- → Vectorised operations — s1 + s2

2.13 Series Indices (like a dictionary in python)

2.14 Vectorized operations and series indices

for series vectorized operations act on indices rather than positions

2.15 Filling missing values (.dropna() and .add(s, fill_value=?))

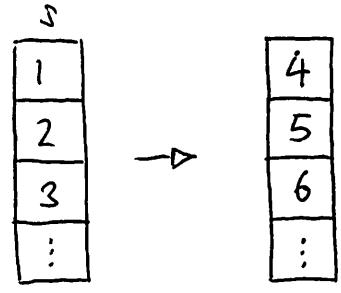
2.16 Pandas Series apply()

Non Built-In Calculations:

apply() takes a series and a function and returns a new series e.g.

```
def add3(x):
    return x+3

s.apply(add3)
```



same as s+3

s.apply(f) f: element ↦ element

2.17 Plotting in Pandas

I II III IV V

L3 Numpy & Pandas for 2D Data

3.1 Introduction

We'll be able to rewrite Lesson 1 code in Pandas

3.2 Subway data

Questions:

- What variables are related to subway ridership?
  - Which station have the most riders?
  - What are the ridership patters over time?
  - How does the weather affect ridership?
- What patterns can I find in the weather?
  - Is the temperature rising throughout the month?
  - How does the weather vary accross the city?

3.3 2D Numpy Arrays

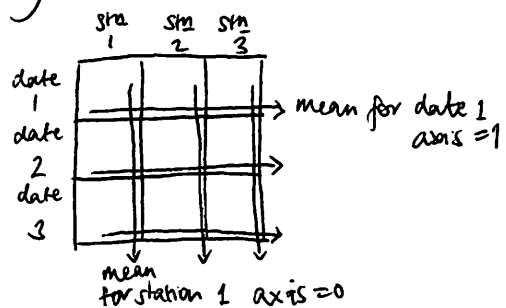
Python : List of list
NumPy : 2Darray ← start here
Pandas : DataFrame

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

2Darrays as oppose to array of arrays:

- More memory efficient
- Accessing elements: a[1, 3] (row, column) rather than a[1][3]
- mean(), std() etc. opperate on entire array

3.4 NumPy Axis



ridership.mean(axis = 0)

or

ridership.mean(axis = 1)

3.5 Numpy & Pandas Data Types

All elements in Numpy arrays must have the same type.

Pandas DataFrame is a 2D data structure but each column is assumed to be a different type. Pandas DataFrames are displayed in a nice table When you take the mean of a DataFrame you get the mean for each column which contains numerical data

3.6 Accessing Elements of a DataFrame

Can use .loc(-)and .iloc(-) as with 1D Series
.values() spits out a numpy array

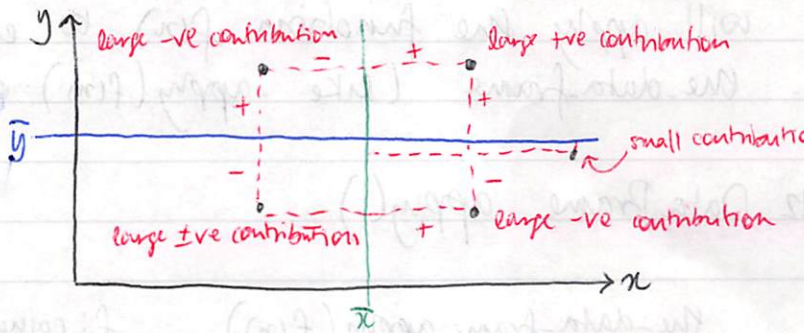
3.7 Loading Data into a Data Frame

.read_csv(filename.csv)
.head() returns a subset of the table

3.8 Calculating Correlation (Pearson's)

N.B. Numpy has a function to calculate this

x_1 y_1 ← both above mean? both below? one above and one below?
x_2 y_2
x_3 y_3
⋮ ⋮
x_n y_n



Pearson's r:
- First standardise each variable
- Multiply each pair of values, and take the average

r = average((x in std units)(y in std units))

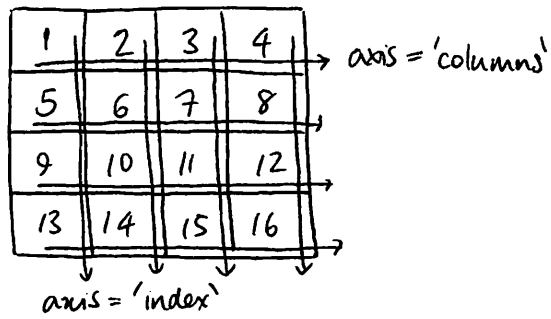
Use var.std(ddof=0)

ensures Bessel's correction is not used

3.9 Pandas Axis Names

Instead of axis = 0 and axis = 1 you can use axis = 'index' and axis = 'columns'

- .mean(axis = 'columns') takes mean along the columns
- .mean(axis = 'index') takes mean along the index or rows



3.10 DataFrame Vectorised Operations

Similar to vectorised operations for 2D NumPy arrays.
Match up elements by index and column rather than pos^{n}

3.11 DataFrame applymap()

Non-built-in functions for DataFrames

the_data_frame.applymap(f(x)) f : element ⟼ element

will apply the function f(x) to each element of the_data_frame (like .apply(f(x)) on a Pandas Series?)

3.12 DataFrame apply()

the_data_frame.apply(f(x)) f : column ⟼ column

will apply the func^{n} f(x) to each column of the_data_frame.

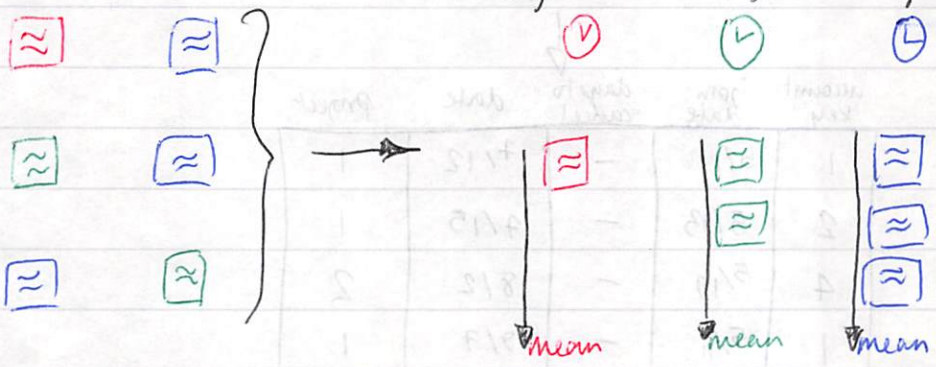
3.13 DataFrame apply() Use Case 2

data_frame.apply(f(x)) f: column $\longmapsto$ element

3.14 Adding a DataFrame to a Series
3.15 Standardising each Column
} See ipython notebook

3.16 Pandas groupby()

e.g. ridership by hour of the day:



data_frame.groupby(column_name) creates a DataFrameGroupBy object. This is like a dict where the keys are column_name entries and values are smaller DataFrames which can be described by a list of rows

data_frame_group_by.groups shows us which rows were mapped to which column_name entry key

- .sum()
- .mean()
- .describe()
- .apply(f)

} all work on DataFrameGroupBy objects

3.17 Combining Pandas DataFrames

enrollments

account key	join date	days to cancel
1	5/11	–
2	5/13	–
3	5/16	3
4	5/19	–
2	6/26	–

duplicate

submissions

account key	date	project
1	7/12	1
2	7/15	1
4	8/2	1
1	9/7	2

merge

account key	join date	days to cancel	date	project	
1	5/11	–	7/12	1	
2	5/13	–	7/15	1	
4	5/19	–	8/2	2	
1	5/11	–	9/7	1	
2	6/26	–	7/15	1	duplicate
3	5/16	3	NaN	Nan	

```
enrollments.merge(submissions, on='account_key', how='inner')
```

on can also be a list

how determines what to do with missing rows

how options: 'inner', 'outer', 'left', 'right'

- 'inner' → only rows with account_keys in both tables are kept
- 'outer' → rows from both tables are kept
- 'left' → rows from left table are kept
- 'right' → rows from right table are kept

merge in Pandas is similar to join in sql

3.18 Plotting with DataFrames

df.plot() produces a line graph with a different coloured line for each column

as_index = False argument in
df.groupby(column_name, as_index = False)
ensures column_name remains in the output.

3.19 Three Dimensional Data

NumPy:
```
a = np.array([
  [['A1a', 'A1b', 'A1c'], ['A2a', 'A2b', 'A2c']],
  [['B1a', 'B1b', 'B1c'], ['B2a', 'B2b', 'B2c']]
  ])
```

Pandas: wp = pd.Panel(...)

I II III IV V

I II III IV V