Data Science Tools and Software

Dr. Mohamed Abdelhafeez

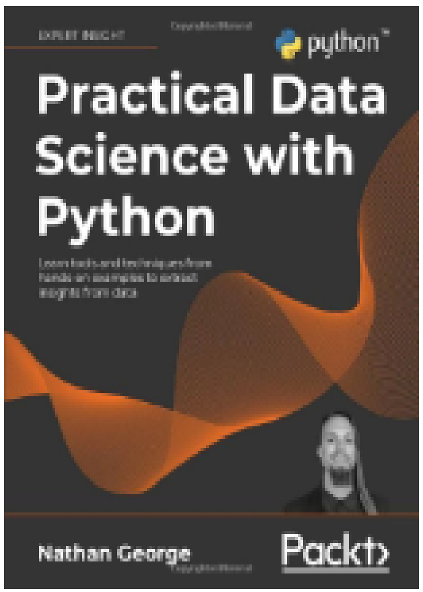
Lecture 1

Course Description

Practical data handling and statistical tools, Applications using Python, Predictive data analysis software as SAS and Apache Spark, Other data analytics software tools, Use cases in Finance, Media, and Health.

Text Book

Practical Data Science with Python



Author: Nathan George

Publisher: Packt Publishing (2021)

Pages: 620

Formats: Paperback, Kindle

Key Topics: Python programming methods, pandas, SciPy, scikit-learn, data cleaning, machine learning, evaluation methods.

Check Price

Course Overview

- Data scientist is one of the best job in terms of job opening, salary, career opportunities rating
- Data science is the application of computational and statistical techniques to address or gain insight into some problem in the real world and to answer scientific inquiries
- Data science is the iterative cycle of designing a concrete problem, building an algorithm to solve it (or determining that this is not possible), and evaluating what insights this provides for the real question
- Data science = statistics +data processing + machine learning + scientific inquiry + visualization + business analytics + big data+…….
- Machine learning may or may not be a step in solving data science problem
- Data science is concerned with both big and small data

Course Overview

- There is no heavy focus on fancy algorithms like in machine learning
- Several interesting topics in data science will be given along with case studies in gene expression analysis, time series, recommender systems, network analysis, and, natural language processing
- The course has a focus on the whole development life cycle of the application including collecting data from unstructured sources and store it using appropriate structure such as relational databases, graphs, matrices, etc. , explore and visualize your data and analyze your data rigorously using a variety of statistical and machine learning approaches in Python.
- We strongly recommend that students have experience with Python, ideally some background in probability and statistics, and linear algebra. However, crash courses will be given.

Data Science Tools

- Data format and Representation (XML/HTML/JSON/CSV,…)
- SQL Database (SQLite and MySQL)
- NOSQL (MongoDb,Neo4j)
- Python (Jupyter Notebook locally or Collab to work across multiple devices): data processing (Panda and Numpy)
- Python: Web scraping (Scrapy)
- Python: Machine learning (Scikit Learn)
- Python: Data Visualization (Matplotlib)
- Python: Deep Learning (TensorFlow): It is a Google-owned open-source machine learning tool widely famous for creating deep learning neural networks.
- SAS, Weka and RapidMiner : includes tools for data processing, machine learning algorithm implementation, and visualization.
- Distributed Data (Apache Hadoob) is an open-source framework that helps create programming models for massive data volumes across multiple clusters of machines.
- Distributed Data Apache Spark: Complex data streams can be analyzed and visualized dynamically using Spark with visualization tools
- Data visualization(Tableau): Creates interactive charts, graphs, and able to OLAP cubes, databases, spreadsheets, and other data sources. It also includes an analytics tool for observing trends and patterns

Grading

- 20 for mid term exam
- 40 for final exam
- 15 for project
- 15 lab assignment
- 10 for assignments, Quizzes & participation

Learning objectives of this course

After taking this course, you should…

… understand the full data science pipeline, and be familiar with programming tools to accomplish the different portions

... be able to collect data from unstructured sources and store it using appropriate structure such as relational databases, graphs, matrices, etc

... know to explore and visualize your data

... be able to analyze your data rigorously using a variety of statistical and machine learning approaches

Topics covered (subject to change)

Data collection and management: relational data, matrices and vectors, graphs and networks, free text processing, geographical data

Statistical modeling and machine learning: linear and nonlinear classification and regression, regularization, data cleaning, hypothesis testing, kernel methods and SVMs, boosting, clustering, dimensionality reduction, recommender systems, deep learning, probabilistic models, scalable ML

Visualization: basic visualization and data exploration, data presentation and interactivity

Philosophy: tools and deeper understand

Most of the techniques we will teach in this course have mature tools that you will likely use in practice

But, the philosophy of this course is that you will use these tools most effectively when you understand what is going on under the hood

This course will teach you some of the more common tools, you will also need to implement some of the underlying methods

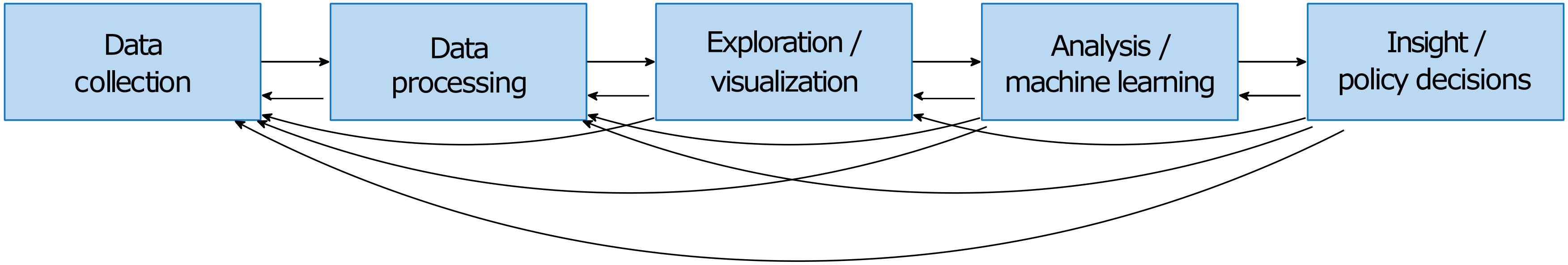
Example: we'll teach you how to run machine learning algorithms using scikit-learn library, but you'll also need to implement some of the algorithms yourself

Distinguishing The Course

In general, this course puts a high emphasis on exploring and analyzing real (unprepared) data, managing the entire data science pipeline

Compared to other machine learning or statistics courses, there is relatively little theory, higher emphasis on implementation and use on practical data sets

Back to what data science is



Data Collection

What is Data?

- Data: noun, *plural (singular: datum)* (dā-tə; dä-)
  - Collection of entities and attributes
- Object:
  - Also known as sample, instance, data point, record, etc.
  - “Row” of the table
  - e.g. a person, a school, a tweet
- Attribute:
  - Also known as field, feature, parameter, variable, code, encoding, etc.
  - “Column” of the table
  - e.g. BMI of a person, student enrollment of a school, number of words in a tweet

avocado.csv (1.9 MB)

	#	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
	0 52	3Jan15 24Mar18	0.44 3.25	84.6 62.5m	0 22.7m	0 20.5m	0 2.55m	0 19.4m	0 13.4m	0 5.72m	0 552k	conventional 50% organic 50% Other (1) 0%	2.02k 2.02k	Albany 2% Atlanta 2% Other (53) 96%
1	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
2	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
3	2	2015-12-13	0.93	118220.22	794.7	109149.67	130.5	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
4	3	2015-12-06	1.08	78992.15	1132.0	71976.41	72.58	5811.16	5677.4	133.76	0.0	conventional	2015	Albany
5	4	2015-11-29	1.28	51039.6	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany
6	5	2015-11-22	1.26	55979.78	1184.27	48067.99	43.61	6683.91	6556.47	127.44	0.0	conventional	2015	Albany
7	6	2015-11-15	0.99	83453.76	1368.92	73672.72	93.26	8318.86	8196.81	122.05	0.0	conventional	2015	Albany
8	7	2015-11-08	0.98	109428.33	703.75	101815.36	80.0	6829.22	6266.85	562.37	0.0	conventional	2015	Albany
9	8	2015-11-01	1.02	99811.42	1022.15	87315.57	85.34	11388.36	11104.53	283.83	0.0	conventional	2015	Albany
10	9	2015-10-25	1.07	74338.76	842.4	64757.44	113.0	8625.92	8061.47	564.45	0.0	conventional	2015	Albany
11	10	2015-10-18	1.12	84843.44	924.86	75595.85	117.07	8205.66	7877.86	327.8	0.0	conventional	2015	Albany
12	11	2015-10-11	1.28	64489.17	1582.03	52677.92	105.32	10123.9	9866.27	257.63	0.0	conventional	2015	Albany
13	12	2015-10-04	1.31	61007.1	2268.32	49880.67	101.36	8756.75	8379.98	376.77	0.0	conventional	2015	Albany
14	13	2015-09-27	0.99	106803.39	1204.88	99409.21	154.84	6034.46	5888.87	145.59	0.0	conventional	2015	Albany
15	14	2015-09-20	1.33	69759.01	1028.03	59313.12	150.5	9267.36	8489.1	778.26	0.0	conventional	2015	Albany
16	15	2015-09-13	1.28	76111.27	985.73	65696.86	142.0	9286.68	8665.19	621.49	0.0	conventional	2015	Albany
17	16	2015-09-06	1.11	99172.96	879.45	90062.62	240.79	7990.1	7762.87	227.23	0.0	conventional	2015	Albany
18	17	2015-08-30	1.07	105693.84	689.01	94362.67	335.43	10306.73	10218.93	87.8	0.0	conventional	2015	Albany
19	18	2015-08-23	1.34	79992.09	733.16	67933.79	444.78	10880.36	10745.79	134.57	0.0	conventional	2015	Albany
20	19	2015-08-16	1.33	80043.78	539.65	68666.01	394.9	10443.22	10297.68	145.54	0.0	conventional	2015	Albany
21	20	2015-08-09	1.12	111140.93	584.63	100961.46	368.95	9225.89	9116.34	109.55	0.0	conventional	2015	Albany
22	21	2015-08-02	1.45	75133.1	509.94	62035.06	741.08	11847.02	11768.52	78.5	0.0	conventional	2015	Albany
23	22	2015-07-26	1.11	106757.1	648.75	91949.05	966.61	13192.69	13061.53	131.16	0.0	conventional	2015	Albany
24	23	2015-07-19	1.26	96617.0	1042.1	82049.4	2238.02	11287.48	11103.49	183.99	0.0	conventional	2015	Albany
25	24	2015-07-12	1.05	124055.31	672.25	94693.52	4257.64	24431.9	24290.08	108.49	33.33	conventional	2015	Albany
26	25	2015-07-05	1.35	109252.12	869.45	72600.55	5883.16	29898.96	29663.19	235.77	0.0	conventional	2015	Albany
27	26	2015-06-28	1.37	89534.81	664.23	57545.79	4662.71	26662.08	26311.76	350.32	0.0	conventional	2015	Albany
28	27	2015-06-21	1.27	104849.39	804.01	76688.55	5481.18	21875.65	21662.0	213.65	0.0	conventional	2015	Albany
29	28	2015-06-14	1.32	89631.3	850.58	55400.94	4377.19	29002.59	28343.14	659.45	0.0	conventional	2015	Albany
30	29	2015-06-07	1.07	122743.06	656.71	99220.82	90.32	22775.21	22314.99	460.22	0.0	conventional	2015	Albany
31	30	2015-05-31	1.23	95123.62	922.37	70469.69	50.55	23681.01	23222.49	458.52	0.0	conventional	2015	Albany
32	31	2015-05-24	1.19	101470.91	680.27	71376.81	58.7	29355.13	28761.81	593.32	0.0	conventional	2015	Albany
33	32	2015-05-17	1.43	109857.47	1150.55	81955.16	94.32	26657.44	26285.43	372.01	0.0	conventional	2015	Albany
34	33	2015-05-10	1.26	120427.91	1420.43	102000.52	185.66	16821.3	16535.55	285.75	0.0	conventional	2015	Albany
35	34	2015-05-03	1.2	59197.67	919.87	45490.05	217.24	12570.51	12201.95	368.56	0.0	conventional	2015	Albany
36	35	2015-04-26	1.22	49585.46	875.65	35841.75	89.62	12778.44	12076.83	701.61	0.0	conventional	2015	Albany
37	36	2015-04-19	1.19	49064.73	774.15	33941.51	47.15	14301.92	13602.97	698.95	0.0	conventional	2015	Albany
38	37	2015-04-12	1.13	48364.29	864.27	30374.15	21.5	17104.37	16438.49	665.88	0.0	conventional	2015	Albany
39	38	2015-04-05	1.16	47362.13	961.77	35577.66	93.76	10728.94	9869.16	755.61	104.17	conventional	2015	Albany
40	39	2015-03-29	1.02	67799.08	1402.28	58623.22	89.5	7684.08	7208.49	475.59	0.0	conventional	2015	Albany
41	40	2015-03-22	1.12	46346.85	2141.83	34313.56	141.8	9749.66	9252.6	497.06	0.0	conventional	2015	Albany
42	41	2015-03-15	1.11	43045.79	2128.26	30447.17	99.67	10370.69	9989.59	381.1	0.0	conventional	2015	Albany

Attribute
s

avocado.csv (1.9 MB)

	#	Date	AveragePrice	Total Volume	4046	4225	4770	Total Bags	Small Bags	Large Bags	XLarge Bags	type	year	region
	0 – 52	3Jan15 – 24Mar18	0.44 – 3.25	84.6 – 62.5m	0 – 22.7m	0 – 20.5m	0 – 2.55m	0 – 19.4m	0 – 13.4m	0 – 5.72m	0 – 552k	conventional 50%, organic 50%, Other (1) 0%	2.02k – 2.02k	Albany 2%, Atlanta 2%, Other (53) 96%
1	0	2015-12-27	1.33	64236.62	1036.74	54454.85	48.16	8696.87	8603.62	93.25	0.0	conventional	2015	Albany
2	1	2015-12-20	1.35	54876.98	674.28	44638.81	58.33	9505.56	9408.07	97.49	0.0	conventional	2015	Albany
3	2	2015-12-13	0.93	118220.22	794.7	109149.67	130.5	8145.35	8042.21	103.14	0.0	conventional	2015	Albany
4	3	2015-12-06	1.08	78992.15	1132.0	71976.41	72.58	5811.16	5677.4	133.76	0.0	conventional	2015	Albany
5	4	2015-11-29	1.28	51039.6	941.48	43838.39	75.78	6183.95	5986.26	197.69	0.0	conventional	2015	Albany
6	5	2015-11-22	1.26	55979.78	1184.27	48067.99	43.61	6683.91	6556.47	127.44	0.0	conventional	2015	Albany
7	6	2015-11-15	0.99	83453.76	1368.92	73672.72	93.26	8318.86	8196.81	122.05	0.0	conventional	2015	Albany
8	7	2015-11-08	0.98	109428.33	703.75	101815.36	80.0	6829.22	6266.85	562.37	0.0	conventional	2015	Albany
9	8	2015-11-01	1.02	99811.42	1022.15	87315.57	85.34	11388.36	11104.53	283.83	0.0	conventional	2015	Albany
10	9	2015-10-25	1.07	74338.76	842.4	64757.44	113.0	8625.92	8061.47	564.45	0.0	conventional	2015	Albany
11	10	2015-10-18	1.12	84843.44	924.86	75595.85	117.07	8205.66	7877.86	327.8	0.0	conventional	2015	Albany
12	11	2015-10-11	1.28	64489.17	1582.03	52677.92	105.32	10123.9	9866.27	257.63	0.0	conventional	2015	Albany
13	12	2015-10-04	1.31	61007.1	2268.32	49880.67	101.36	8756.75	8379.98	376.77	0.0	conventional	2015	Albany
14	13	2015-09-27	0.99	106803.39	1204.88	99409.21	154.84	6034.46	5888.87	145.59	0.0	conventional	2015	Albany
15	14	2015-09-20	1.33	69759.01	1028.03	59313.12	150.5	9267.36	8489.1	778.26	0.0	conventional	2015	Albany
16	15	2015-09-13	1.28	76111.27	985.73	65696.86	142.0	9286.68	8665.19	621.49	0.0	conventional	2015	Albany
17	16	2015-09-06	1.11	99172.96	879.45	90062.62	240.79	7990.1	7762.87	227.23	0.0	conventional	2015	Albany
18	17	2015-08-30	1.07	105693.84	689.01	94362.67	335.43	10306.73	10218.93	87.8	0.0	conventional	2015	Albany
19	18	2015-08-23	1.34	79992.09	733.16	67933.79	444.78	10880.36	10745.79	134.57	0.0	conventional	2015	Albany
20	19	2015-08-16	1.33	80043.78	539.65	68666.01	394.9	10443.22	10297.68	145.54	0.0	conventional	2015	Albany
21	20	2015-08-09	1.12	111140.93	584.63	100961.46	368.95	9225.89	9116.34	109.55	0.0	conventional	2015	Albany
22	21	2015-08-02	1.45	75133.1	509.94	62035.06	741.08	11847.02	11768.52	78.5	0.0	conventional	2015	Albany
23	22	2015-07-26	1.11	106757.1	648.75	91949.05	966.61	13192.69	13061.53	131.16	0.0	conventional	2015	Albany
24	23	2015-07-19	1.26	96617.0	1042.1	82049.4	2238.02	11287.48	11103.49	183.99	0.0	conventional	2015	Albany
25	24	2015-07-12	1.05	124055.31	672.25	94693.52	4257.64	24431.9	24290.08	108.49	33.33	conventional	2015	Albany
26	25	2015-07-05	1.35	109252.12	869.45	72600.55	5883.16	29898.96	29663.19	235.77	0.0	conventional	2015	Albany
27	26	2015-06-28	1.37	89534.81	664.23	57545.79	4662.71	26662.08	26311.76	350.32	0.0	conventional	2015	Albany
28	27	2015-06-21	1.27	104849.39	804.01	76688.55	5481.18	21875.65	21662.0	213.65	0.0	conventional	2015	Albany
29	28	2015-06-14	1.32	89631.3	850.58	55400.94	4377.19	29002.59	28343.14	659.45	0.0	conventional	2015	Albany
30	29	2015-06-07	1.07	122743.06	656.71	99220.82	90.32	22775.21	22314.99	460.22	0.0	conventional	2015	Albany
31	30	2015-05-31	1.23	95123.62	922.37	70469.69	50.55	23681.01	23222.49	458.52	0.0	conventional	2015	Albany
32	31	2015-05-24	1.19	101470.91	680.27	71376.81	58.7	29355.13	28761.81	593.32	0.0	conventional	2015	Albany
33	32	2015-05-17	1.43	109857.47	1150.55	81955.16	94.32	26657.44	26285.43	372.01	0.0	conventional	2015	Albany
34	33	2015-05-10	1.26	120427.91	1420.43	102000.52	185.66	16821.3	16535.55	285.75	0.0	conventional	2015	Albany
35	34	2015-05-03	1.2	59197.67	919.87	45490.05	217.24	12570.51	12201.95	368.56	0.0	conventional	2015	Albany
36	35	2015-04-26	1.22	49585.46	875.65	35841.75	89.62	12778.44	12076.83	701.61	0.0	conventional	2015	Albany
37	36	2015-04-19	1.19	49064.73	774.15	33941.51	47.15	14301.92	13602.97	698.95	0.0	conventional	2015	Albany
38	37	2015-04-12	1.13	48364.29	864.27	30374.15	21.5	17104.37	16438.49	665.88	0.0	conventional	2015	Albany
39	38	2015-04-05	1.16	47362.13	961.77	35577.66	93.76	10728.94	9869.16	755.61	104.17	conventional	2015	Albany
40	39	2015-03-29	1.02	67799.08	1402.28	58623.22	89.5	7684.08	7208.49	475.59	0.0	conventional	2015	Albany
41	40	2015-03-22	1.12	46346.85	2141.83	34313.56	141.8	9749.66	9252.6	497.06	0.0	conventional	2015	Albany
42	41	2015-03-15	1.11	43045.79	2128.26	30447.17	99.67	10370.69	9989.59	381.1	0.0	conventional	2015	Albany

Objects

Set, Sequence, & Space

Ordered

Set, Sequence, & Space

Unorder
ed

Geometr
ic-
Structure
d

Unordered Data

- A set of attributes
- Ordering of attributes doesn’t really matter {a, b} = {b, a}

- Examples
  - Documents, tweets, web contents
  - Demographic data
  - Employee records
  - Student records
  - Bank transaction records
  - Product inventory
  - ...

Unordered Data

- e.g. Student record table

qryStudentInformation : Select Query

strStudentID	strFirstName	strLastName	strAddress1	strPostCode
BJ25555	LEE	FORD	28 SIMPSON CLOSE	L21 5ER
BJ25567	JANE	PIPER	8 LUBERNUM ROAD	CH45 5RT
BJ26667	ADELE	CUSHING	12 SEDDEN ROAD	L19 2LJ
BJ26887	MATTHEW	FOGGERTY	15 WOODMEDOW COURT	OL5 0BQ
BJ29991	DARREN	PHILLIPS	101 HERSCHELL STREET	L5 1XE
BJ34684	IAN	FORSTER	20 ALT ROAD	L20 5ES
BJ45126	PAUL	LAWSON	29 BROAD LANE	L11 8LY
CY21147	PENNY	HARES	75 WIDCOMBE RD	SE9 4HY

Record: 12 of 52

Unordered Data

- e.g. Documents (bag of words)

Article ID	biolog	biopsi	biolab	biotin	almost	cancer-surviv	cancer-stage	Article Class
00001	12	1	2	10	0	1	4	breast-cancer
00002	10	1	0	3	0	6	1	breast-cancer
00014	4	1	1	1	0	28	0	breast-cancer
00063	4	0	0	0	0	18	7	breast-cancer
00319	0	1	0	9	0	20	1	breast-cancer
00847	7	2	0	14	0	11	5	breast-cancer
03042	3	1	3	1	0	19	8	lung-cancer
05267	4	4	2	6	0	14	11	lung-cancer
05970	8	0	4	9	0	9	17	lung-cancer
30261	1	0	0	11	0	21	1	prostate-cancer
41191	9	0	5	14	0	11	1	prostate-cancer
52038	6	1	1	17	0	19	0	prostate-cancer
73851	1	1	8	17	0	17	3	prostate-cancer

Ordered Data

- Ordered set of attributes
- Ordering matters! (a, b) != (b, a)

- Examples
  - Time series
  - Sequence
  - ...

Ordered Data

- e.g. Genetic Sequence

```
12854400 tcaaagtaagttagataaacatgatcattcacaggtcagatgttttaaaaaaaaatcattatggtgtacatcacatgtagacaatacttcagaattcatc
         tggactaccagaattgagttacctagtacttctcaattctattttaccctaacgtctaataaataacaagtactctagcctcttcgttttatgattcctc
12854200 taggaaaagttaatgttacggcccaatcactttttttaacagcccaaacaacatatattagctccaaatatcatttttttcccctagaatattctcaacct
         attgtccactcaaaacgtgacaaatggaggtctaaagggagaccatacttgactcattttagagctaggatcagacagagtagatttttttgccataactc
12854000 cttgtaaatgtattcacatttcattcccaagaaaaatagactgatgaagaaatatatcagatatgacaaggccgtgtcgtttaggttacgtaactctaca
         aggtttagggtctcaatataaacacacaaagcagatagaagaagcaaaccattcacaatcagacaATGACATCTCTCCATACGTTACTCTTCTCTTCTCT
12853800 TCTTTTCTTCATCGTCTTTCCAACCTTCACGTTTTCCTCCACCTTATTGTTTCAGgttcgtctttagttttgcttctttacatacacagactctacacac
         tcacttattgggtttctttcaattgtgaaacagAGTTTCAATTGGGAGTCATGGAAGAAAGAAGGAGGATTCTACAATTCTCTCCACAACTCCATTGACG
12853600 ACATAGCCAACGCTGGAATCACTCATCTTTGGCTTCCTCCTCCTTCTCAATCCGTTGCTCCTGAAGgttccatttctgctttactctttacacattcaca
         taccaatcttgttactcacgcaatcttcattcctcagGTTACTTACCGGGAAAGCTATACGATCTAAACAGCTCCAAATACGGTTCAGAGGCGGAACTGA
12853400 AATCGTTAATCAAAGCGTTGAATCAAAAAGGAATAAAAGCTTTGGCTGATATAGTGATTAACCACAGAACAGCTGAGAGGAAAGACGATAAATGTGGATA
         CTGTTATTTCGAAGGTGGACTTCCGATGATCGTCTTGATTGGGATCCTTCCTTTGTCTGCCGCAATGACCCTAAATTTCCCGGTACCGGAAACCTCGAC
12853200 ACCGGAGGAGATTTTGATGGAGCGCCCGACATCGACCACCTTAACCCTAGAGTTCAGAAAGAGTTGTCCGAATGGATGAATTGGCTTAAAACTGAAATCG
         GATTCCATGGTTGGAGATTTGATTATGTTCGAGGTTATGCATCTTCCATCACCAAATTATACGTTCAGgtaaatcacatatgaattctcaaatatcagac
12853000 aacagtattagtatataagaaacataggttgagataattatttactattagtatatataagtatcataggttgatagggttatttactactatttagtat
         ataagaaacataagtcaatgcaatcaataagaaatatataagaaagttcactactgattatgtgataaattcctctgtttttggatacacagAATACATC
12852800 ACCGGATTTTGCGGTGGGTGAGAAATGGGACGATATGAAGTACGGAGGAGACGGGAAACTAGACTATGATCAGAACGAGCATCGGTCGGGTCTCAAACAG
         TGGATCGAGGAAGCGGGTGGTGGTGTGTTGACAGCTTTTGATTTCACCACCAAAGGGATCTTACAGTCTGCTGTCAAAGGTGAGCTTTGGAGACTAAAGG
12852600 ACTCGCAGGGAAAACCGCCTGGTATGATAGGAATCATGCCCGGAAACGCTGTCACATTCATAGATAACCATGATACATTCAGAACGTGGGTTTTCCCTTC
         TGATAAAGTCTTGCTTGGATACGTTTATATACTTACTCATCCAGGAACTCCTTGCATTgtaagtatcattttagtatgtagctatactatttacaactac
12852400 aatcttgttgatatgttattttttgttgcagTTTTATAATCATTACATAGAATGGGGACTAAAAGAGAGCATCTCAAAGCTGGTGGCTATCAGGAACAAAA
         ATGGGATTGGTAGCACAAGCTCTGTAACGATAAAAGCGGCAGAGGCGGATCTCTACTTGGCTATGATTGATGATAAAGTTATCATGAAGATTGGACCAAA
12852200 GCAAGATGTGGGAACACTTGTTCCTTCTAATTTTGCTTTAGCTTATTCAGGCCTTGACTTTGCTGTCTGGGAGAAGAAGTAAcgcataactcgaatcata
         agaaaagtaatcgaatgtatcttcttccttttttaataaaacatttggcagtatctaaagatatgtataatgaaatataaaatgataaagaatacctaaa
12852000 taaaaagagcactagtggtgttaaaggatacaactccagtgaaagaaaagagttcaagtgaagaagtgtcaacttgtagaaataagtattggaaagtttc
         catcgttttgttttgttgcatacaactaatatattatatattggccgactcgtataagatttggagccctactaaaatcagaattatgatgtcttaacca
12851800 cacaatactgccaaaatcagaacgaattatattattgtagaagaagaaaaaaaaagtatggtgggaagtgggaacagttagacaggtaaattcgaataaa
```

AT4G25000.1

Ordered Data

- e.g. Stock price (candlestick chart)

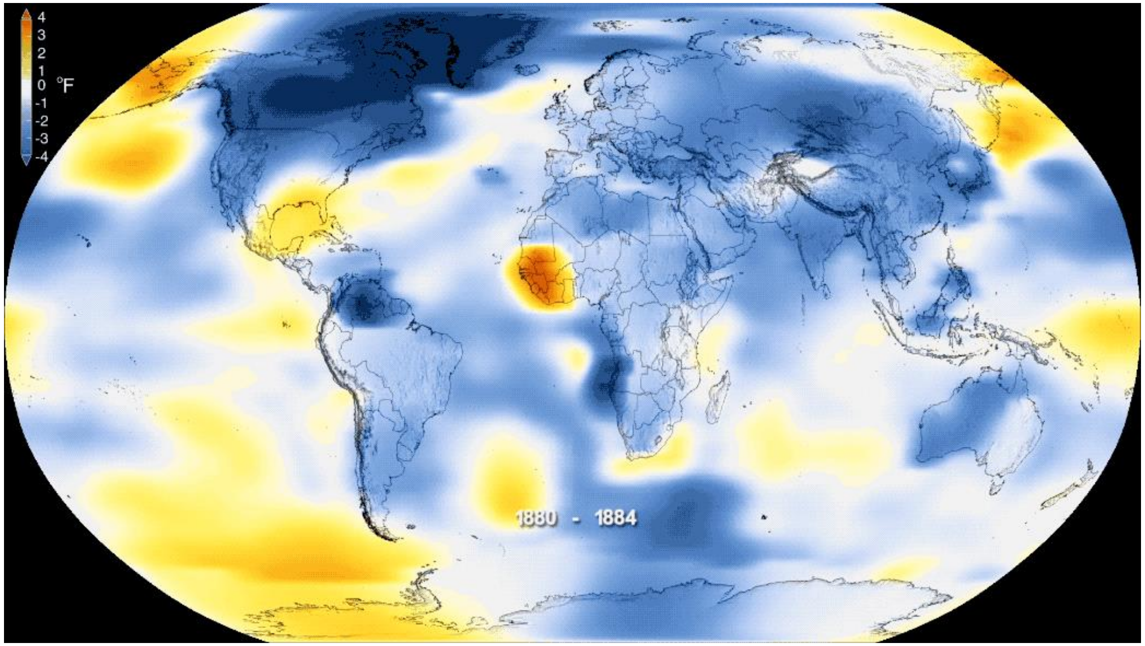
Geometric/Structured Data

- Data sets that have geometric/topological/geographical structures.
- Spatial location of an object comes into play.

- Example
  - Spatio-temporal data
  - Image pixels, points in LiDAR, computer graphics models
  - Graph data

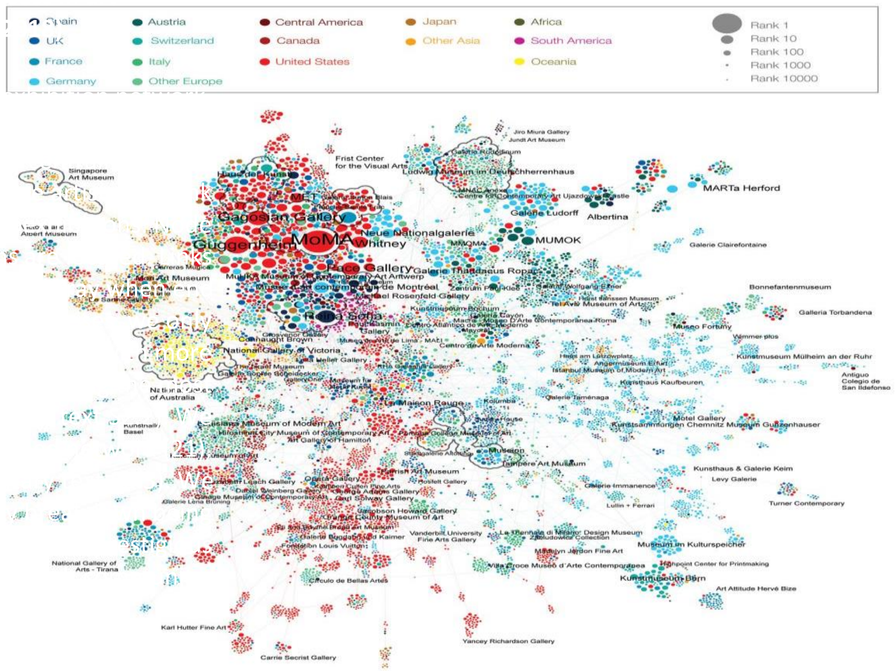
Geometric/Structured Data

- e.g. Spatio-temporal data



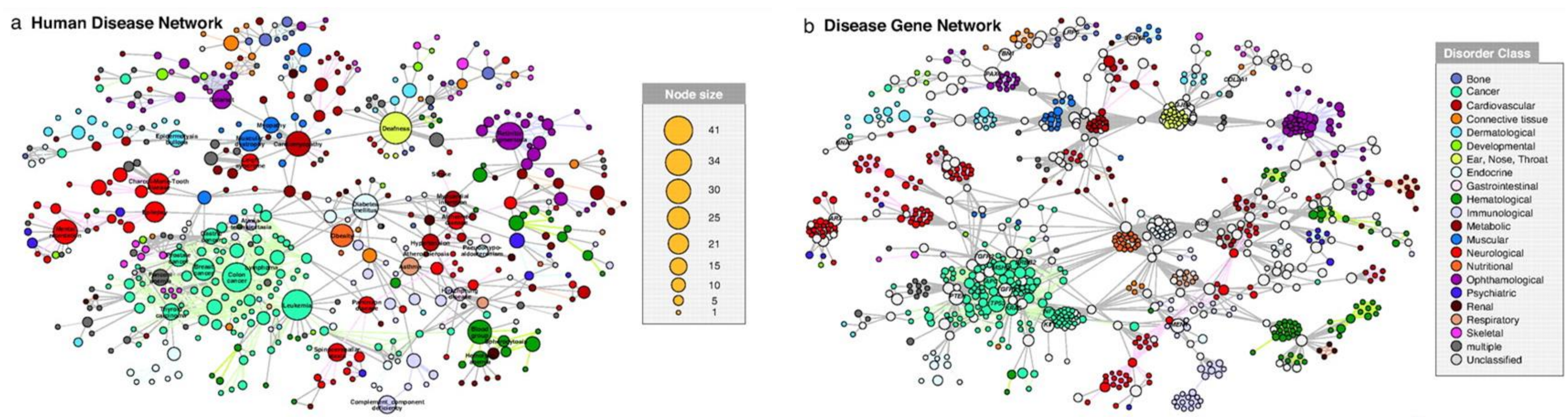
Geometric/Structured Data

- e.g. Graph data (social network)



Geometric/Structured Data

- e.g. Graph data (disease network)



Geometric/Structured Data

- e.g. Graph data (Stanford Large Network Dataset Collection)
  - http://snap.stanford.edu/
  - Social Networks
  - Communication Networks
  - Citation Networks
  - Collaboration Networks
  - Road Networks
  - Temporal Networks
  - ...

The first step of data science

- The first step in data science ...
- ... is to get some data
- You will typically get data in one of four ways:

1. Directly download a data file (or files) manually – not much to say
2. Query data from a database – to be covered in later lecture

3. Query an API (usually web-based, these days)
4. Scrap data from a webpage

covered today

Issuing HTTP queries

The vast majority of automated data queries you will run will use HTTP requests (it's become the dominant protocol for much more than just querying web pages)

I know we promised to teach you know things work under the hood … but we are *not* going to make you implement an HTTP client

Do this instead (requests library, http://docs.python-requests.org/):

```
import requests
response = requests.get("http://www.datasciencecourse.org")
# some relevant fields
response.status_code
response.content # or response.text
response.headers
response.headers['Content-Type']
```

HTTP Request Basics

You've seen URLs like these:
https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=9&cad=rja&uact=8…

The weird statements after the url are *parameters,* you would provide them using the requests library like this:

```
params = {"sa":"t", "rct":"j", "q":"", "esrc":"s",
"source":"web", "cd":"9", "cad":"rja",
"uact":"8"}
response =
requests.get("http://www.google.com/url",
params=params)
```

HTTP GET is the most common method, but there are also PUT, POST, DELETE methods that *change* some state on the server

```
response = requests.put(...)
response = requests.post(...)
response = requests.delete(...)
```

RESTful APIs

If you move beyond just querying web pages to web APIs, you'll most likely encounter REST APIs (Representational State Transfer)

REST is more a design architecture, but a few key points:

1. Uses standard HTTP interface and methods (GET, PUT, POST, DELETE)
2. Stateless – the server doesn't remember what you were doing

Rule of thumb: if you're sending the your account key along with each API call, you're probably using a REST API

Querying a RESTful API

You query a REST API similar to standard HTTP requests, but will almost always need to include parameters

```
token = "" # not going to tell you mine
response = requests.get("https://api.github.com/user",
params={"access_token":token})
print(response.content)
#{"login":"zkolter","id":2465474,"avatar_url":"https://avatars.githubu…
```

Get your own access token at https://github.com/settings/tokens/new

GitHub API uses GET/PUT/DELETE to let you query or update elements in your GitHub account automatically

Example of REST: server doesn't remember your last queries, for instance you always need to include your access token if using it this way

Authentication

Basic authentication has traditionally been the most common approach to access control for web pages

```
# this won't work anymore
response = requests.get("https://api.github.com/user",
auth=('zkolter', 'passwd'))
```

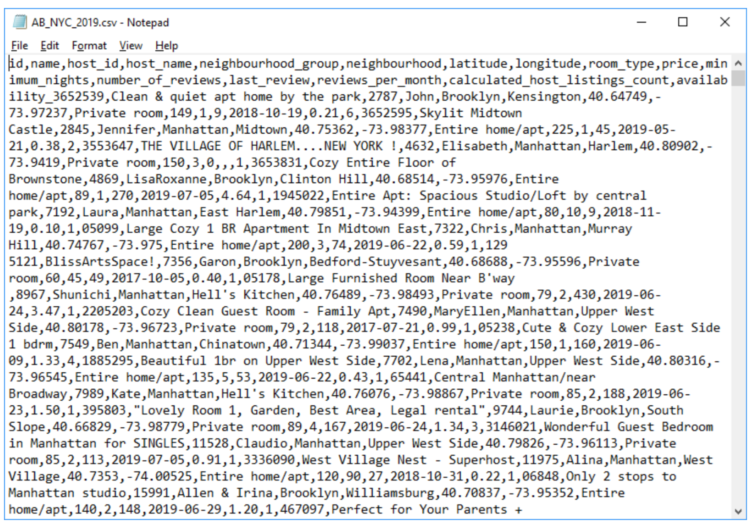
Most APIs have replaced this with some form of OAuth (you'll get familiar with OAuth in the homework)

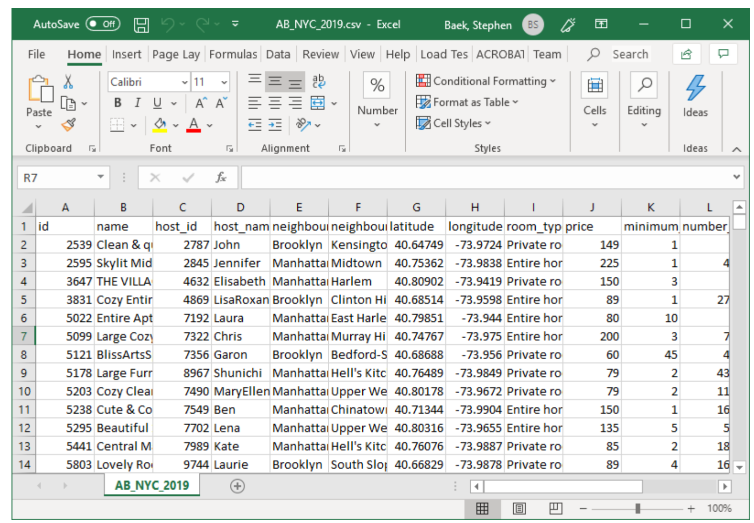
Data Formats

- There are MANY data representations in computers!
  - Comma Separated Values (CSV)
  - JavaScript Object Notation (JSON)
  - (hypertext markup language / extensible markup language) (HTM/XML)...

Data Formats

- Comma Separated Values (CSV)
  - Delimited text file that uses comma (,) to separate values
  - Some ~~weird~~ people use something else, like semicolon (;), instead
  - Tabular data is stored in plain text → large file size





https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

CSV Files

Refers to any delimited text file (not always separated by commas)

```
"Semester","Course","Section","Lecture","Mini","Last Name","Preferred/First
Name","MI","Andrew ID","Email","College","Department","Class","Units","Grade
Option","QPA Scale","Mid-Semester Grade","Final Grade","Default Grade","Added
By","Added On","Confirmed","Waitlist Position","Waitlist Rank","Waitlisted
By","Waitlisted On","Dropped By","Dropped On","Roster As Of Date”
"F16","15688","B","Y","N",”Kolter","Zico","","zkolter","zkolter@andrew.cmu.edu","S
CS","CS","50","12.0","L","4+"," "," ","","reg","1 Jun
2016","Y","","","","","","","30 Aug 2016 4:34"
```

If values themselves contain commas, you can enclose them in quotes (our registrar apparently always does this, just to be safe)

We’ll talk about the pandas library a lot more in later lectures

```
import pandas as pd
dataframe = pd.read_csv("CourseRoster_F16_15688_B_08.30.2016.csv",
delimiter=',', quotechar='"')
```

Data Formats

- JavaScript Object Notation (JSON)
  - Easy for humans to read and write, easy for machines to parse and generate
  - Attribute-value pairs + arrays
  - Good for structured data


```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```


Wikipedia

JSON files / string

JSON originated as a way of encapsulating Javascript objects

A number of different data types can be represented

- Number: `1.0` (always assumed to be floating point)
- String: `"string"`
- Boolean: `true` or `false`
- List (Array): `[item1, item2, item3,…]`
- Dictionary (Object in Javascript): `{"key":value}`

Lists and Dictionaries can be embedded within each other:

[{"key":[value1, [value2, value3]]}]

Example JSON data

JSON from Github API


```
{
 "login":"zkolter",
 "id":2465474,
 "avatar_url":"https://avatars.githubusercontent.com/u/2465474?v=3",
 "gravatar_id":"",
 "url":"https://api.github.com/users/zkolter",
 "html_url":"https://github.com/zkolter",
 "followers_url":"https://api.github.com/users/zkolter/followers",
 "following_url":"https://api.github.com/users/zkolter/following{/other_user}",
 "gists_url":"https://api.github.com/users/zkolter/gists{/gist_id}",
 "starred_url":"https://api.github.com/users/zkolter/starred{/owner}{/repo}",
 "subscriptions_url":"https://api.github.com/users/zkolter/subscriptions",
 "organizations_url":"https://api.github.com/users/zkolter/orgs",
 "repos_url":"https://api.github.com/users/zkolter/repos",
 "events_url":"https://api.github.com/users/zkolter/events{/privacy}",
 "received_events_url":"https://api.github.com/users/zkolter/received_events",
 "type":"User",
 "site_admin":false,
 "name":"Zico Kolter"
 ...
```

Parsing JSON in Python

Built-in library to read/write Python objects from/to JSON files

```
import json

# load json from a REST API call

response = requests.get("https://api.github.com/user", params={"access_token":token})

data = json.loads(response.content)

json.load(file) # load json from file

json.dumps(obj) # return json string

json.dump(obj, file) # write json to file
```

Data Formats

- eXtensible Markup Language (XML)
  - Easy for humans, easy for machines
  - Made of tags
    - Start-tag: <tagname>
    - End-tag: </tagname>
    - Empty-element-tag: <tagname />
  - Attributes
    - Name-value pair that exists in a tag.
    - For example,

      - Tag: img (an empty tag)
      - Attributes: src, alt


```
<root>
  <listing>
    <seller_info>
      <seller_name> cubsfantony</seller_name>
      <seller_rating> 848</seller_rating>
    </seller_info>
    <payment_types>
      Visa/MasterCard, Money Order/Cashiers Checks, Personal Checks, See item description fo
    </payment_types>
    <shipping_info>
      Buyer pays fixed shipping charges, Will ship to United States only
    </shipping_info>
    <buyer_protection_info> </buyer_protection_info>
    <auction_info>
      <current_bid>$620.00 </current_bid>
      <time_left> 4 days, 14 hours + </time_left>
      <high_bidder>
        <bidder_name> gosha555@excite.com </bidder_name>
        <bidder_rating>-2 </bidder_rating>
      </high_bidder>
      <num_items>1 </num_items>
      <num_bids> 12</num_bids>
      <started_at>$1.00 </started_at>
      <bid_increment> </bid_increment>
      <location> USA/Chicago</location>
      <opened> Nov-27-00 04:57:50 PST</opened>
      <closed> Dec-02-00 04:57:50 PST</closed>
      <id_num> 511601118</id_num>
      <notes> </notes>
    </auction_info>
    <bid_history>
      <highest_bid_amount>$620.00 </highest_bid_amount>
      <quantity> 1</quantity>
    </bid_history>
    <item_info>
      <memory> 256MB PC133 SDram</memory>
      <hard_drive> 30 GB 7200 RPM IDE Hard Drive</hard_drive>
      <cpu>Pentium III 933 System </cpu>
      <brand> </brand>
      <description>
        NEW Pentium III 933 System - 133 MHz BUS Speed Pentium Motherboard, Intel Pentium II
        Panasonic CD-RW 8x4x32 - ATI All-In-Wonder 128 PRO 32MB AGP Video Card with TV tuner
        V90 US Robotics Fax/Modem, 10/100 Network Card, Microsoft Internet Keyboard, Microso
        Windows 98 2nd Edition is installed for configuration purposes only and then removed
        options. 1 Year warranty on parts and labor (3 years on monitor from mfg). Buyer agr
        through eBay's Billpointand PayPal. SHIPMENT GUARANTEED WITHIN 10 BUSINESS DAYS FROM
        insurance. NO RESERVE PRICE.. Bid with confidence with one of eBay's ID VERIFIED Pow
        cost of the system and possible upgrades the system will be built once payment is re
        Please allow 3 to 5 additional business days for shipping VIA UPS Ground Service. DO
      </description>
    </item_info>
```

XML/ HTML files

The main format for the web (though XML seems to be loosing a bit of popularity to JSON for use in APIs / file formats)

XML files contain hiearchical content delineated by tags

```
<tag attribute="value">
<subtag>
Some content for the subtag
</subtag>
<openclosetag attribute="value2”/>
</tag>
```

HTML is syntactically like XML but horrible (e.g., open tags are not always closed), more fundamentally, HTML is mean to describe appearance

Parsing XML/HTML in Python

There are a number of XML/HTML parsers for Python, but a nice one for data science is the BeautifulSoup library (specifically focused on getting data out of XML/HTML files

```
# get all the links within the data science course schedule
from bs4 import BeautifulSoup
import requests
response = requests.get("http://www.datasciencecourse.org/2016")
root = BeautifulSoup(response.content)
root.find("section",id="schedule").find("table").find("tbody").findAll("a")
```

You'll play some with BeautifulSoup in the first homework

SQL database

- Relational DataBase Management System
- Main Focus is ACID
  - Atomicity – Each transaction is atomic. If one part of it fails, the entire transaction fails (and is rolled back)
  - Consistency – Every transaction is subject to a consistent set of rules (constraints, triggers, cascades)
  - Isolation – No transaction should interfere with another transaction
  - Durability – Once a transaction is committed, it remains committed

Regular expressions

Once you have loaded data (or if you need to build a parser to load some other data format), you will often need to search for specific elements within the data

E.g., find the first occurrence of the string “data science”

```
import re
text = "This course will introduce the
basics of data science"
match = re.search(r"data science", text)
print(match.start())
```

Regular expressions in Python

A few common methods to call regular expressions in Python:

```
match = re.match(r"data science", text)
match = re.search(r"data science", text)
for match in re.finditer("data science", text) :

    ...
all_matches = re.findall(r"data science", text)
```

You can also use “compiled” version of regular expressions

```
regex = re.compile(r"data science")
regex.match(text, [startpos, [endpos]])
regex.search(...)
regex.finditer(...)
regex.findall(...)
```

Matching multiple potential characters

The real power of regular expressions comes in the ability to match multiple possible sequence of characters
Special characters in regular expressions: .^$*+?{}\[]|() (if you want to match these characters exactly, you need to escape them: \$)

Match sets of characters:

- Match the character ‘a’: a
- Match the character ‘a’, ‘b’, or ‘c’: [abc]
- Many any character except ‘a’, ‘b’, or ‘c’: [^abc]
- Match any digit: \d (= [0-9])
- Match any alpha-numeric: \w (= [a-zA-z0-9_])
- Match whitespace: \s (= [\t\n\r\f\v])
- Match any character:. (including newline with re.DOTALL)

Matching repeated characters

Can match one or more instances of a character (or set of characters)

Some common modifiers:

- Match character ‘a’ exactly once: a
- Match character ‘a’ zero or one time: a?
- Match character ‘a’ zero or more times: a*
- Match character ‘a’ one or more times: a+
- Match character ‘a’ exactly n times: a{n}

Can combine these with multiple character matching:

- Match all instances of “<something> science” where <something> is an alphanumeric string with at least one character
- \w+\s+science

Poll: regular expressions

Which strings would be matched (i.e, calling re.match()) by the regular expression?

`\w+\s+science`

1. “life science”
2. “life sciences”
3. “life. Science”
4. “this data science problem”

Grouping

We often want to obtain more information that just whether we found a match or not (for instance, we may want to know what text matched)
Grouping: enclose portions of the regular expression in quotes to “remember” these portions of the match

```
(\w+)\s([Ss]cience)
```

```
match = re.search(r"(\w+)\s([Ss]cience)", text)
print(match.start(), match.groups())
# 49 ('data', 'science')
```

Why the ‘r’ before the string? Avoids need to double escape strings

Substitutions

Regular expressions provide a power mechanism for replacing some text with outer text

```
better_text = re.sub(r"data science", r "schmada science", text)
```

To include text that was remembered in the matching using groups, use the escaped sequences \1, \2, … in the substitution text

```
better_text = re.sub(r"(\w+)\s([Ss])cience", r"\1 \2hmience", text)
```

(You can also use backreferences within a single regular expression)

Ordering and greedy matching

There is an order of operations in regular expressions

- abc|def matches the strings “abc” or “def”, not “ab(c or d)ef”
- You can get around this using parenthesis e.g. a(bc|de)f
- This also creates a group, use a(?:bc|de)f if you don’t want to capture it

By default, regular expressions try to capture as much text as possible (greedy matching)

- <(.*)> applied to <a>text will match the entire expression
- If you want to capture the *least* amount of text possible use <(.*?)> this will just match the <a> term

Additional features

We left out a lot of elements here to keep this brief: start/end lines, lookaheads, named groups, etc

Don't worry if you can't remember all this notation (I had to look some things up while preparing this lecture too)

Use the docs: https://docs.python.org/3/howto/regex.html, https://docs.python.org/3/library/re.html

Try out test expressions to see what happens