

Course 1264

Introduction to Data Science, Machine Learning, and AI Using Python

by

Mary Flynn



LEARNING TREE™
INTERNATIONAL

Copyright

© LEARNING TREE INTERNATIONAL, INC.
All rights reserved.

All trademarked product and company names are the property of their
respective trademark holders.

No part of this publication may be reproduced, stored in a retrieval system, or
transmitted in any form or by any means, electronic, mechanical, photocopying,
recording or otherwise, or translated into any language, without the prior written
permission of the publisher.

Copying software used in this course is prohibited without the express
permission of Learning Tree International, Inc. Making unauthorized copies of
such software violates federal copyright law, which includes both civil and
criminal penalties.

Introduction and Overview



LEARNING TREE
INTERNATIONAL

Course Objectives

- ▶ Work with Python to analyze structured and unstructured data
- ▶ Harness data mining methods to answer crucial business questions from internal and external data sources
- ▶ Employ supervised machine learning techniques to estimate future outcomes
- ▶ Unearth patterns in data with unsupervised techniques
- ▶ Preprocess unstructured data for deeper analysis



Course Objectives

- ▶ **Apply clustering, classification, and regression to datasets**
- ▶ **Generate association rules from transaction data**
- ▶ **Mine relational patterns from network data**
- ▶ **Communicate data-driven narratives to a wider audience**



Course Contents

Introduction and Overview

- Chapter 1 Introduction to Data Science, Machine Learning, and AI**
- Chapter 2 Exploratory Data Analysis (EDA)**
- Chapter 3 Working With Unstructured Data**
- Chapter 4 Predicting Outcomes With Regression Analysis**
- Chapter 5 Categorizing Data With Classification Techniques**
- Chapter 6 Additional Classification Methods**
- Chapter 7 Detecting Patterns in Data With Clustering Analysis**
- Chapter 8 Association Rules**
- Chapter 9 Additional Techniques: Social Network Analysis**

Course Contents

- Chapter 10 Data Science and Big Data Analytics: Final Thoughts**
- Chapter 11 Course Summary**
- Next Steps**

Course Content and Audience

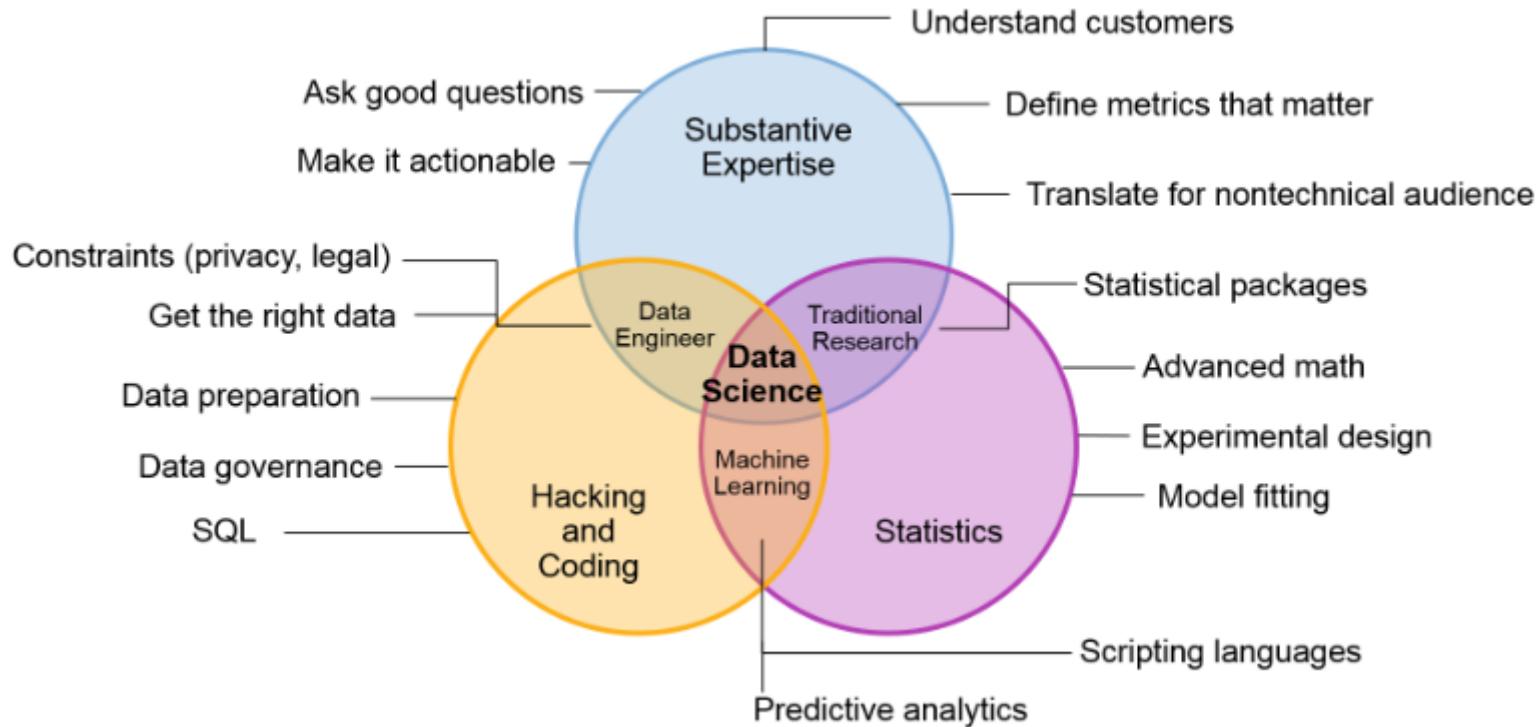
- ▶ **The content is important to both technical and business specialists in**
 - Becoming comfortable exploring different types of data (structured, unstructured, Big Data)
 - Understanding data-driven decision-making
 - Exploring modeling techniques that go beyond standard data reporting
 - Examining some of the current approaches to handling Data Analytics
 - Planning and implementing a Data Analytics project in the workplace
 - Learning Python for data science tasks
- ▶ **Attendee background**
 - Background in basic statistics is helpful, but not required
 - Programming background is helpful in understanding the sample code
- ▶ **This course is not designed for**
 - Attendees already proficient in machine learning techniques (e.g., clustering, classification, association rules analysis, regression)
 - Learning Java-based Hadoop development and administration

Technical vs. Non-Technical Attendees

- ▶ Data Science requires a mixture of technical and non-technical skills, and as such, attracts audiences from very different backgrounds
- ▶ Attendees from a technical background may wish to use this course as an opportunity to learn Python
 - If so, follow the exercises as instructed
- ▶ On the other hand, if you are from a non-technical background, solution files are provided on your Hands-On machines
 - Run the solutions, step by step, ensuring you understand what each step is doing, and what the output is telling you

Data Scientist Skill Set

The Data Science Venn Diagram*

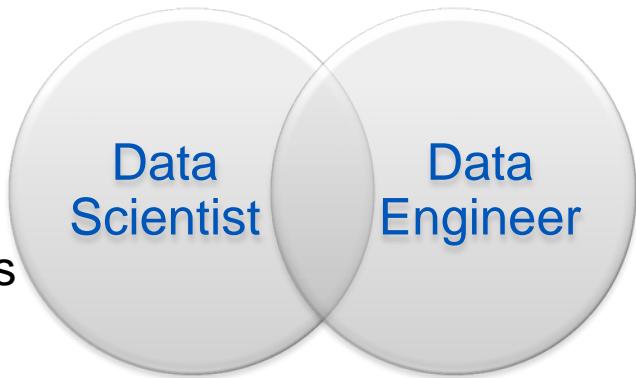


- ***Data scientists* are skilled professionals whose expertise allows them to quickly switch roles at any point in the lifecycle of data science projects**

*Gartner, 2016.

Data Scientist vs. Data Engineer

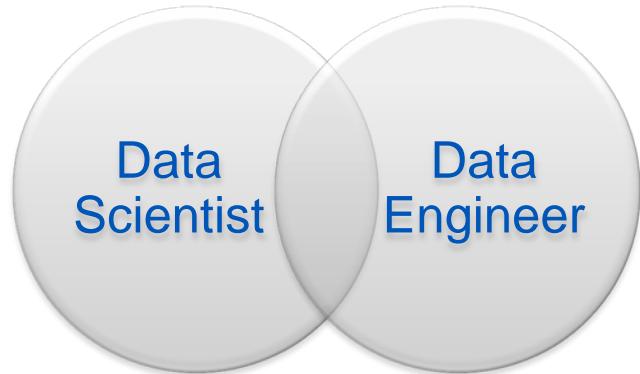
- ▶ **The roles can overlap quite substantially**
 - Usually dependent on DS/DE background
- ▶ **A data scientist typically:**
 - Conducts research to answer industry questions
 - Identifies internal and external data sources
 - Examines data to find hidden patterns
 - Uses analytics, statistical methods and machine learning in building ML models



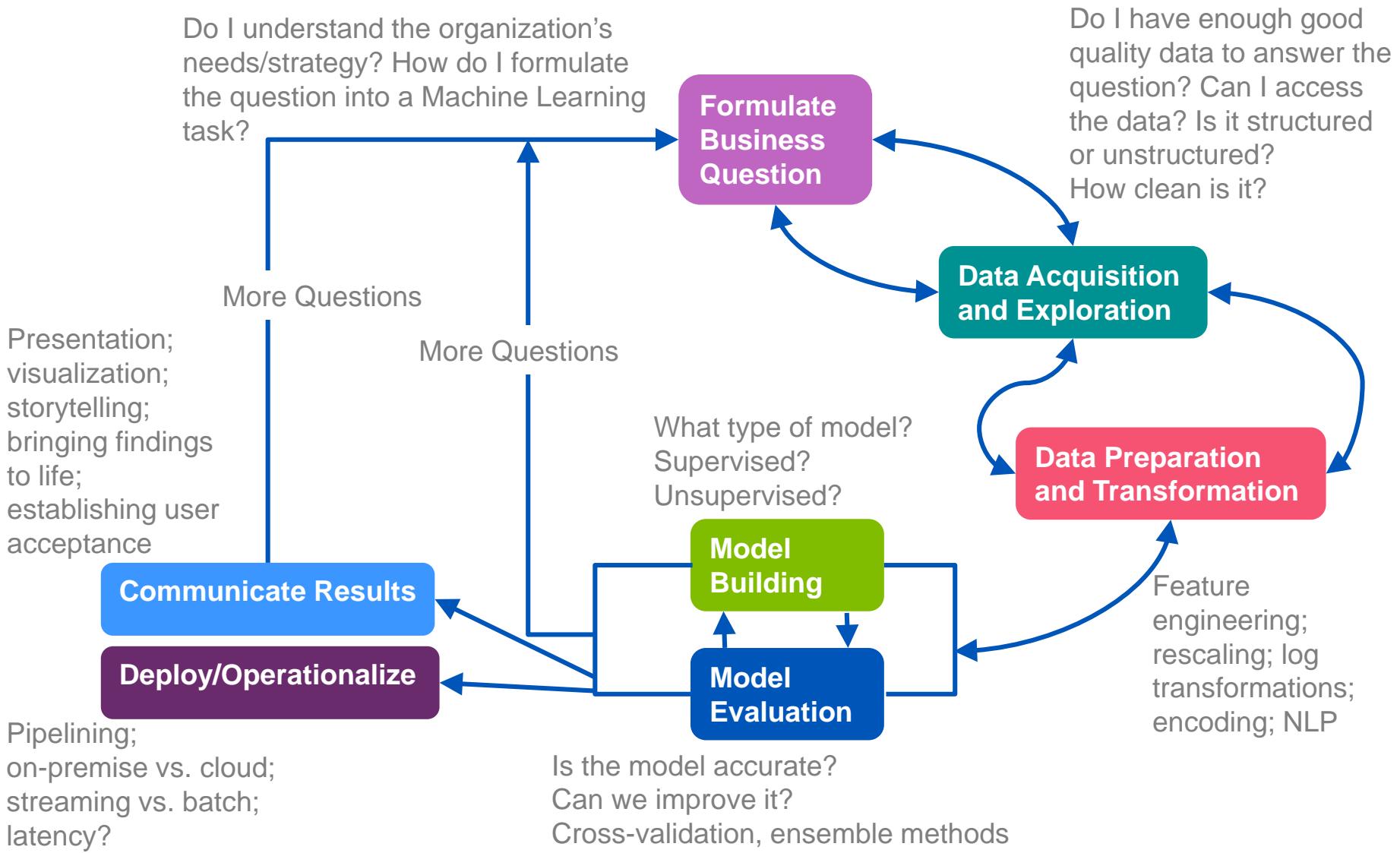
DS/DE = Data Scientist/Data Engineer
ML = Machine Learning

Data Scientist vs. Data Engineer

- ▶ **A data engineer typically:**
 - Develops, constructs, tests and maintains architectures, such as databases and large-scale processing systems
 - Ensures architecture will support the requirements of the business
 - Develops data processes for putting ML models into production
 - Employs a variety of tools to connect different data entities together
- ▶ **The output of a data scientist's efforts is typically a report to key stakeholders where narrative is built around the data findings to convey meaning in an audience appropriate language**
 - Sometimes outputs are built into data engineering pipelines, but not always



Data Science Life Cycle



Datasets Used in the Course

- ▶ A mixture of datasets will be used throughout the course to demonstrate machine learning techniques
- ▶ Datasets used within the course are publicly available datasets that were provided by third parties for educational purposes within the Machine Learning community
 - We wish to thank and acknowledge the contributors for those datasets; and they are provided for your convenience
- ▶ Since some datasets are compiled by third parties, Learning Tree International makes no representation or warranty on data contained in such files
 - The datasets are made available on an “as is” and “as available” basis without any warranties of any kind

Chapter 1

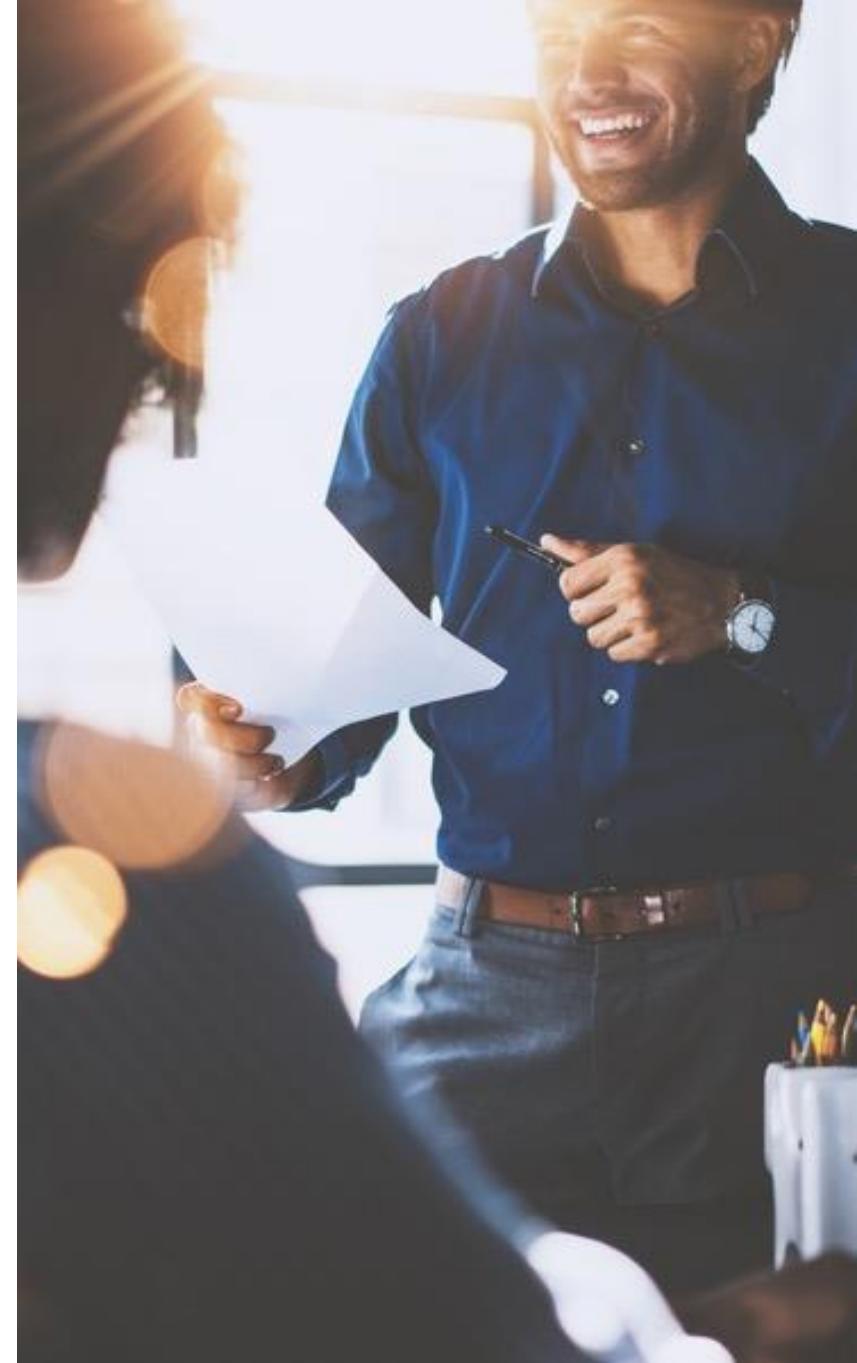
Introduction to Data Science, Machine Learning and AI



LEARNING TREE™
INTERNATIONAL

Objectives

- ▶ **Discuss how to turn business questions into data mining problems**
- ▶ **Explore diverse and wide-ranging data sources that can be used to answer business questions**
- ▶ **Explore the use of Python for Data Analytics and Machine Learning**



Contents

Defining Data Science

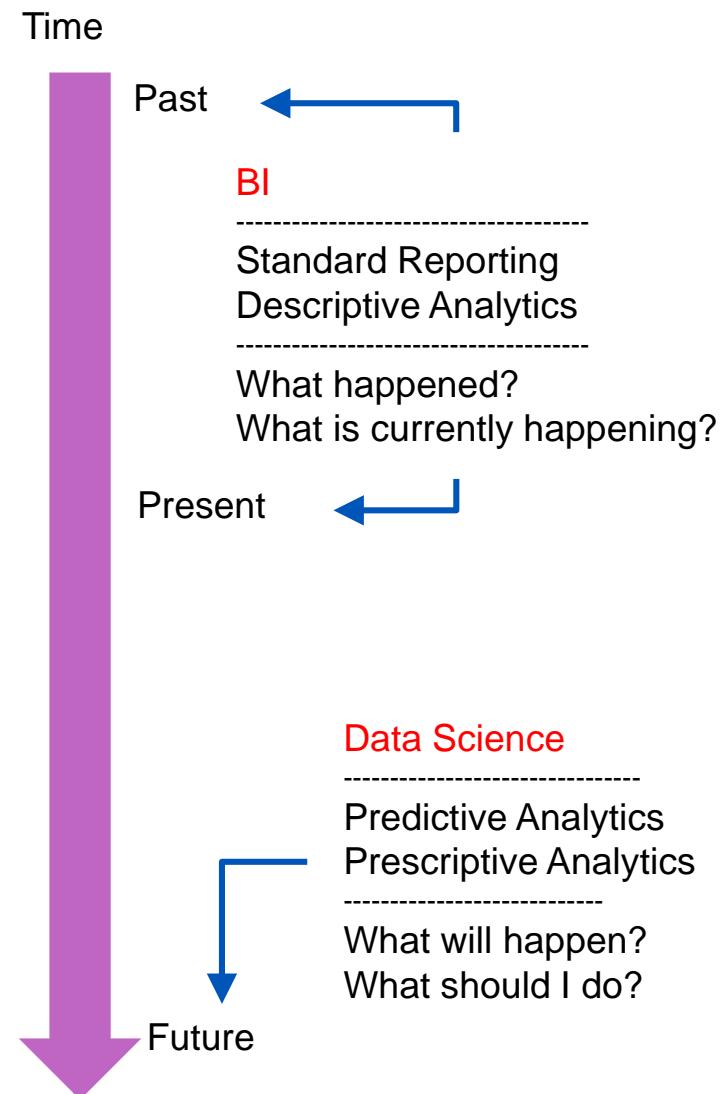
- ▶ **Formulating Business Questions**
- ▶ **Sourcing Data to Understand and Answer Business Questions**
- ▶ **Introduction to Python**
- ▶ **Viewing Datasets in Python**
- ▶ **Hands-On Exercise 1.1**



How Does Data Science Differ From BI and Data Analytics?

► **BI helps us interpret current and past data, typically through summaries and aggregations**

- Mainly used for reporting or descriptive analytics
 - What pages did visitors view most often?
 - What are our sales figures by city?
 - What products are most popular?

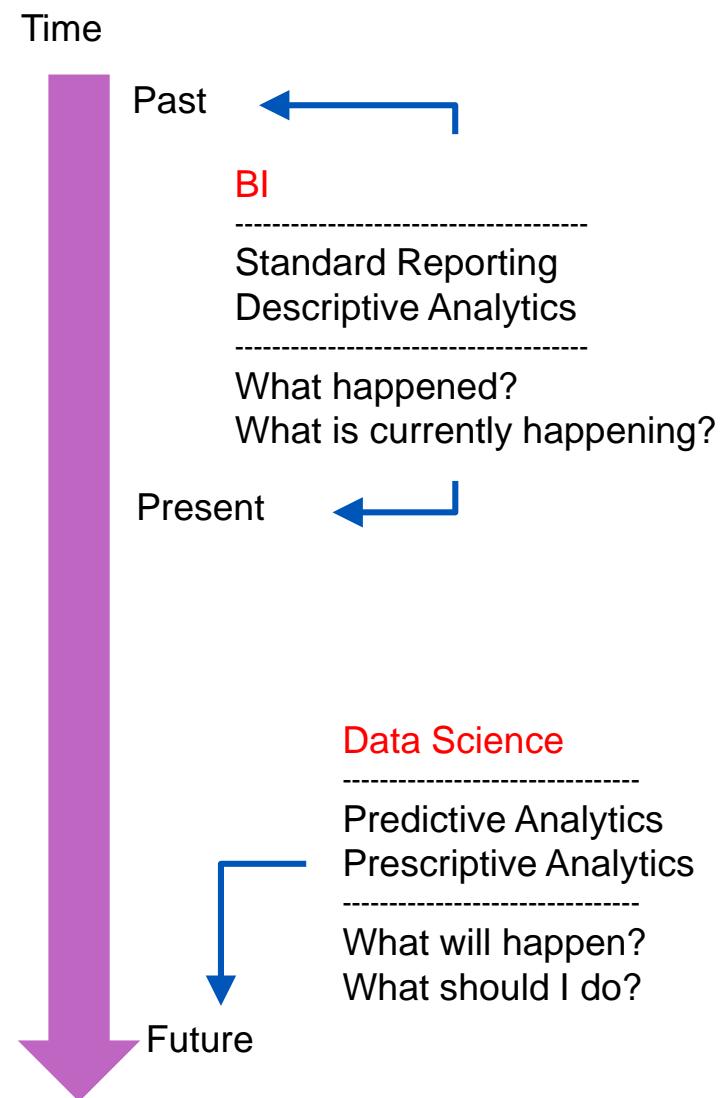


BI = Business Intelligence

How Does Data Science Differ From BI and Data Analytics?

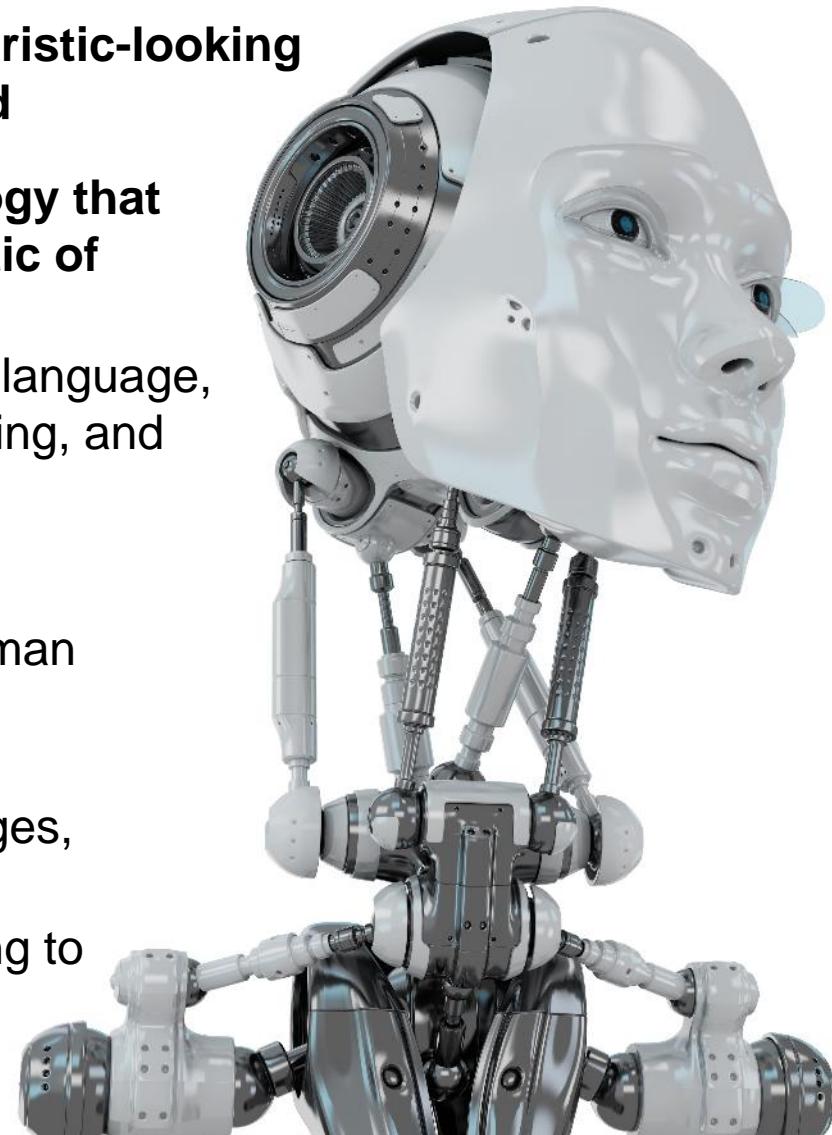
► **Data Science is used more for Predictive Analytics or Prescriptive Analytics**

- Data Science can analyze the past data (trends or patterns) to make future predictions
 - What page will a visitor next view, given the visitor's...
 - Browsing history and demographics
 - Should a credit card company approve a transaction that's waiting, given the...
 - User's usage history
 - Item being purchased
 - Location of the merchant



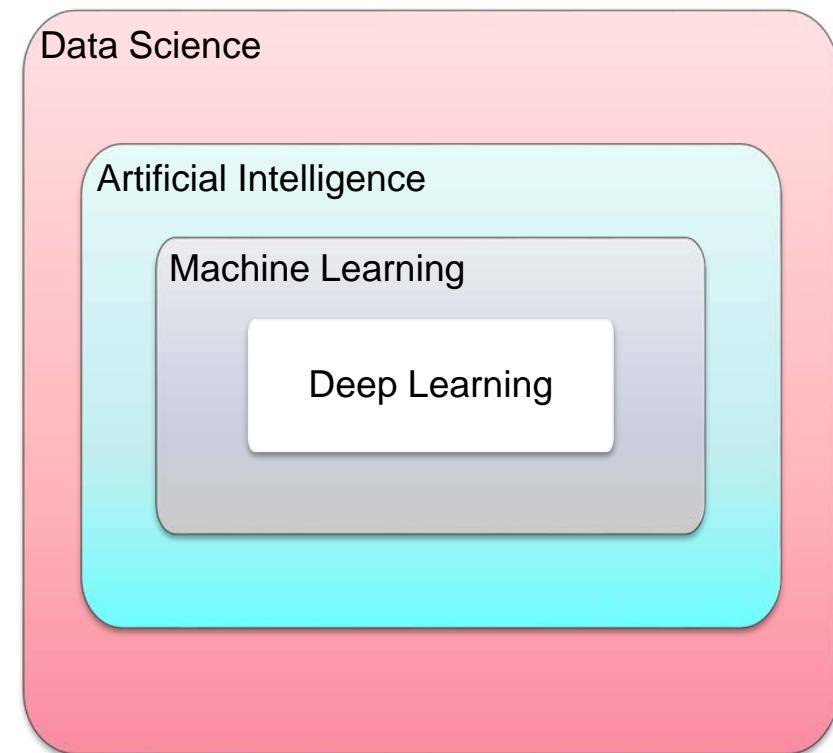
How Does Data Science Overlap With AI and Machine Learning?

- ▶ AI has come to be associated with futuristic-looking robots and a machine-dominated world
- ▶ AI is a broad term to describe technology that can perform tasks that are characteristic of human intelligence
 - May include things like understanding language, recognizing objects and sounds, learning, and problem solving
- ▶ AI can be narrow or general
 - Narrow AI exhibits some aspect of human intelligence that it can do quite well, but nothing else
 - A machine that can recognize images, but nothing else
 - Research into General AI, while leading to exciting advancements, is still in its infancy



Data Science vs. Machine Learning and AI

- **Machine Learning is a subsection of AI where models are built that can learn and improve from data rather than being programmed to do so and includes:**
 - Supervised machine learning
 - Uses labelled data to uncover patterns that might be used for future forecasting
 - Unsupervised machine learning
 - Uses unlabelled data and focuses on discovering hidden structures and patterns in data
 - Reinforcement machine learning
 - Learns from trial and error or punishment and reward to achieve an outcome



Data Science vs. Machine Learning and AI

- ▶ Deep Learning is a subset of Machine Learning and focuses on using neural networks to learn iteratively from enormous datasets
 - Repeated data exposure helps in understanding differences and eradicating errors
- ▶ Contributes heavily towards making our daily lives more convenient
 - Parking assistance
 - Face recognition at the airport
 - Self-driving cars
 - Voice assistants
- ▶ Deep learning is fuelling a lot of automation in today's world



Contents

- ▶ Defining Data Science

Formulating Business Questions

- ▶ Sourcing Data to Understand and Answer Business Questions
- ▶ Introduction to Python
- ▶ Viewing Datasets in Python
- ▶ Hands-On Exercise 1.1



Thinking Skills Required of a Data Scientist

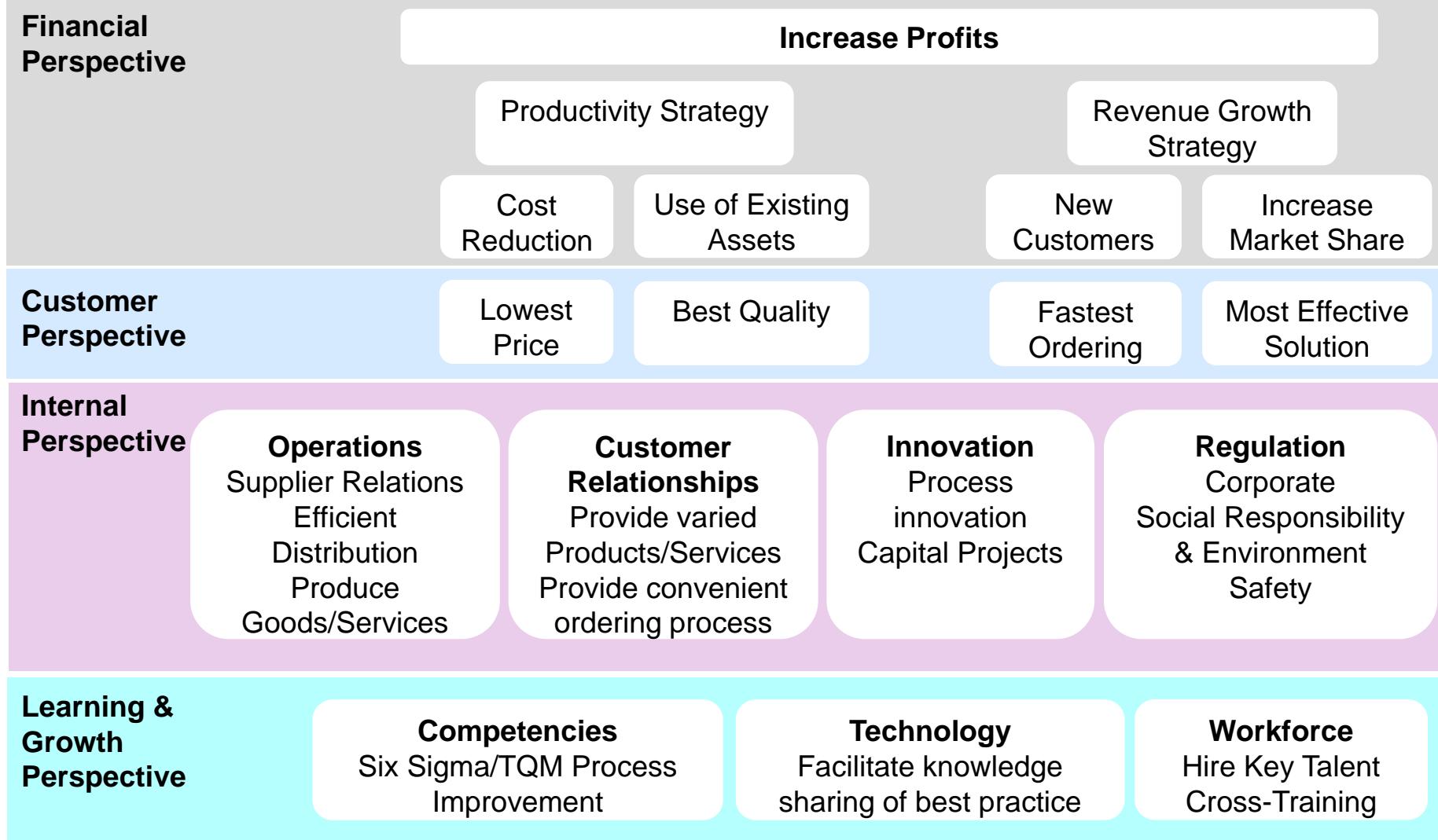
- ▶ According to Stephen Few*, a data scientist requires many types of thinking skills
 - Domain Knowledge
 - An understanding of the subject matter context including objectives and processes of that context
 - Critical thinking
 - Looking for data to both validate and disprove our assumptions in equal measure; dealing with our own cognitive biases
 - Scientific thinking
 - Formulating and testing hypotheses
 - Statistical thinking
 - Applying statistical tests in order to prove/disprove our hypotheses
 - Systems thinking
 - Seeing how aspects of the system interact and affect each other
 - Visual thinking
 - Knowing how to visualize in order to clarify patterns in data
 - Ethical thinking
 - How might our findings be received and used, once presented

*Stephen Few is the author of many books on Data Analysis such as: "Now You See It", "Information Dashboard Design", "The Data Loom", and many others.

Finding Data Science Questions: Where to Begin?

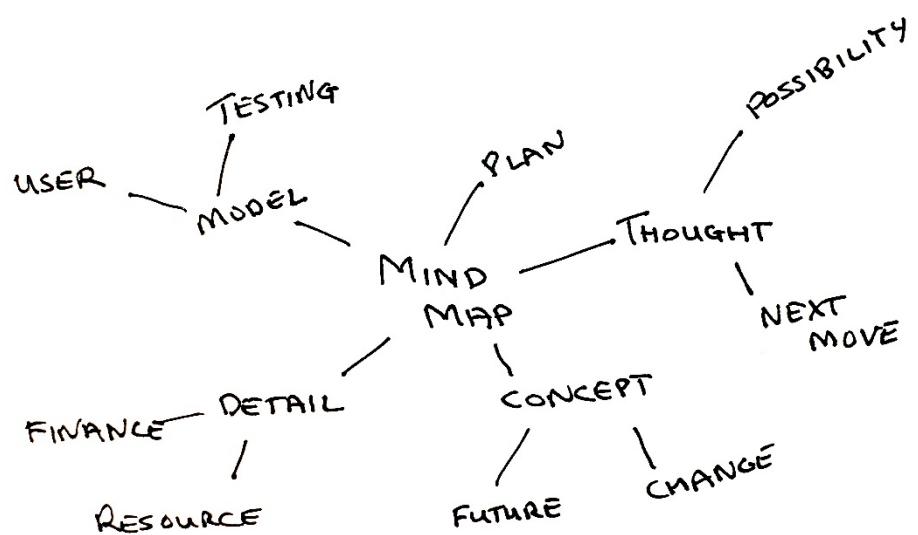
- ▶ **Data science questions typically focus on real-world problems rather than simplified definitions/categorizations/rules and aim to explain them in a more nuanced way**
- ▶ **Questions often start with an observation**
 - Don't worry initially about what data is available
 - Write down anything interesting you noticed in or around the organization
 - The data scientist may make a hypothesis about an observed phenomena
 - It is essential that the hypothesis is testable and is based on reason and prior information
 - The answer to one question typically leads to new questions
- ▶ **Start by thinking broadly about the organization's strategy**
 - Most organizations will operate under a strategy map
 - The more familiar you are with the goals of the organization, the more focused your data science projects will be
- ▶ **An observation can turn into a data science question**
 - Talk about it with your colleagues!

Aligning Data Science Questions to Organizational Strategy



Concept Maps

- ▶ Concept maps can be useful when brainstorming business questions with colleagues
- ▶ Ask questions that require:
 - Observation: What do you notice?
 - Recall: What memories do you have of seeing it in the past?
 - Grouping: Which of our observations can be grouped together for some reason?
 - Labelling: What might be a name for this group based on its attributes?
 - Classifying: What observation doesn't fit in any of our groups and why?
- ▶ Draw lines between entities you think might have interesting relationships that would be worth investigating



Translating Business Questions Into Machine Learning Problems

- ▶ Recognizing the structure of a Machine Learning task in a business question is a key skill for a Data Scientist

Question	Machine Learning Task	Machine Learning Technique
<ul style="list-style-type: none">• How do I estimate the demand for our products to optimize our logistics?• What are my sales figures likely to be next month?	Predict numeric values by estimating the relationship between variables	Regression
<ul style="list-style-type: none">• How can we better meet our customers needs?• How can we make good recommendations to customers that won't alienate them?• Is there a way of predicting what plant/machinery might fail so that we can service it before it happens?	Discover structure by separating similar data points into natural groupings	Clustering
<ul style="list-style-type: none">• Is this tweet positive?• Will this customer churn?• Which website layout retains customers?• What service will this customer choose?	Predict between categories	Classification

Formulating Hypotheses

- In the scientific method, a question is formulated into a “hypothesis” which is composed of two hypotheses

- A null hypothesis (H_0)
- An alternate hypothesis (H_a)

- A business question such as “Does our marketing campaign lead to increased product awareness?” would be phrased as:

- “There is no relationship between the marketing campaign and increased product awareness” (H_0)
- “There is a positive relationship between the marketing campaign and increased product awareness” (H_a)

- Formulating a business question into a hypothesis

- Focuses the data mining effort by defining the variables of interest
- Helps determine what data will and will not be needed in the study
- Clarifies what statistical tests are required to answer the question

The Scientific Method as an Ongoing Process



Disproving the Null Hypothesis

- ▶ **A data scientist needs to look for data to disprove the null hypothesis**
 - For example, data needs to show that a recommender engine achieves a conversion rate of 3 percent or more, and is statistically inconsistent with the organization's current conversion rate of 1.5 percent
- ▶ **These quantitative requirements impose constraints on the quality and quantity of the data needed to achieve the goal**
- ▶ **Our initial focus will be to identify appropriate datasets by asking**
 - Is the data we already have sufficient to answer the question asked?
 - If not, where and how can we get the data we really need?
 - This may entail proactive data collection
 - Are there variances and biases within the data?
 - What is a reasonable and realistic baseline against which to test our hypothesis?
 - How much data do we need to reject our hypothesis?

Type I and Type II Errors

- ▶ A Type I error (a false positive) occurs if the H_0 is rejected when it is actually true
 - A medical test comes back positive, but you don't have the disease
 - Virus software on your computer incorrectly identifies a harmless program as malicious
- ▶ A Type II error (a false negative) occurs when the H_0 is not rejected and it is false
 - A defective item passes quality control tests
 - In the Justice System, a false negative occurs when a guilty suspect is found "Not Guilty" and allowed to walk free
- ▶ Type I and Type II errors cannot be completely eliminated
 - The likelihood of either can be minimized, but typically the probability of the other will be increased
 - The particular use case will often determine which error you are more accepting of

Null Hypothesis vs. Alternate Hypothesis

Do Now

- The owner of a retail store assumes his employees are honest. However, the data from the daily takings is showing a number of discrepancies. The owner has the choice of: a) assuming it is due to human error and ignoring the situation, or b) making an allegation of theft.
1. Which is the null hypothesis, and which is the alternate hypothesis? Explain your answer.

2. What would amount to a Type I error in this problem?

3. What would amount to a Type II error in this problem?

4. Which is more serious? Explain your answer.

Overview of the Data Scientist's Approach to Business Questions

- ▶ A large number of visualizations and statistical tests are typically generated to understand the nature and quality of the datasets
- ▶ Insights derived from the data should be communicated back to stakeholders in a rapid cycle typically every few days to every week
- ▶ Questions/hypotheses are further honed, and typically new data sources are examined with each cycle
- ▶ Through this iterative process, the data scientist reaches deeper insight into how exactly to answer valuable business questions with real data
- ▶ The data scientist also reaches conclusions about viable tools that can be used to answer those questions with the data

Contents

- ▶ Defining Data Science
- ▶ Formulating Business Questions

Sourcing Data to Understand and Answer Business Questions

- ▶ Introduction to Python
- ▶ Viewing Datasets in Python
- ▶ Hands-On Exercise 1.1



Data Sourcing

► Data can come from both internal and external sources

- Wireless sensor networks
- NoSQL data stores
- The Internet
- Spatial data
- Data warehouse
- Transactional systems

Text Files and Documents

Relational Databases, Data Warehouses

Web and Application Log Files

Social Media

Video Files

Audio Files

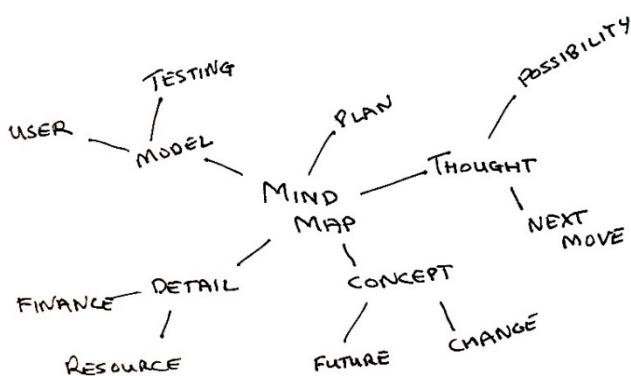
Images

Emails

IoT Sensor Data

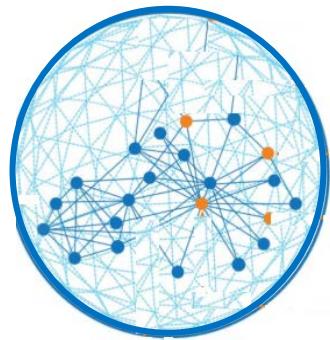
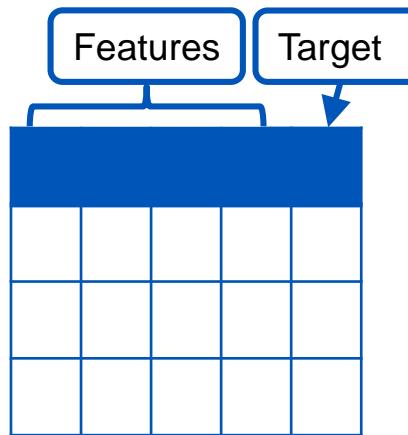
Evaluating What Data Is Available for Your Specific Question

- ▶ Look at each of the related ideas on your concept map and evaluate which you have data available for
- ▶ How will you use the available data to answer your question?



Machine Learning Workflow

- Data sources will typically require integration, cleaning and transformation before they can be used in a Machine Learning model



Data Sourcing and Integration

Machine Learning Workflow

IoT = Internet of Things
ML = Machine Learning

Acquiring the Data

- ▶ **Is the data you need available to you?**
 - How will you gain access?
 - Are data owners willing to share?
- ▶ **How is it stored?**
 - Relational database
 - Data Warehouse
 - NoSQL
 - Images
 - Audio
 - Flat files
- ▶ **What tools do you need to work with so that you can import and manage it?**

Review Questions

Review

- 1. What is the difference between Data Science and Business Intelligence?**

- 2. What is the difference between Data Science, Machine Learning, and AI?**

- 3. Where is a good place to start finding questions that can be answered with Data Science and Machine Learning?**

- 4. How might you formulate these questions in a way that can be answered with data?**

- 5. What might be a good Data Science question in your own workplace?**

Contents

- ▶ Defining Data Science
- ▶ Formulating Business Questions
- ▶ Sourcing Data to Understand and Answer Business Questions

Introduction to Python

- ▶ Viewing Datasets in Python
- ▶ Hands-On Exercise 1.1



Introduction to Python

- ▶ **Free, open-source software, with a vibrant community**
- ▶ **Python is a general-purpose language**
- ▶ **Uses external libraries, usually written in faster languages (e.g., C, C++, etc.) for analytic computing**
 - A *library* is a collection of functions and methods that allows you to perform lots of actions without writing your own code
 - Some of the libraries used in this course are pandas, Scikit-learn and Matplotlib

Why Use Python for Data Science?

- ▶ **Data scientists typically depend on either R or Python as their language of choice for data science activities**
 - Both enjoy strong open-source communities
- ▶ **Python:**
 - Can be used for both data science and deployment, which enables rapid deployment where required
 - Python, however, is a general-purpose language and requires a higher level of programming skills from the data scientist
- ▶ **R:**
 - More purpose-built for data science with very sophisticated capabilities for machine learning and statistics
 - Not a general programming language
 - If integration into data engineering pipelines is required, R is more of a prototyping environment
- ▶ **Ultimately, context will drive the decision**
 - The majority of data scientists will be familiar with both

Python Stacks

- The easiest way to install a Python stack is to download one of these Python distributions, which include all the key packages

Distribution	Description	Supports
Anaconda	Free distribution	Linux, Windows, and Mac
Enthought Canopy	Free and commercial versions	Linux, Windows, and Mac
Python(x,y)	Free distribution based around the Spyder IDE	Windows only
WinPython	Free distribution	Windows only
Pyzo	Free distribution based on Anaconda and the IEP interactive development environment	Linux, Windows, and Mac

IDE = Integrated Development Environment
IEP = Individualized Education Program

External Libraries

- ▶ NumPy: provides numerical arrays and routines for manipulation
<http://www.numpy.org/>
- ▶ SciPy: provides high-level data processing routines, such as regression
<https://www.scipy.org/>
- ▶ Matplotlib: provides comprehensive 2-D visualizations
<http://matplotlib.org/>
- ▶ pandas: provides high-performance, easy-to-use data structures and data analysis tools
<http://pandas.pydata.org/>
- ▶ Scikit-learn: provides tools for data mining and data analysis
<http://scikit-learn.org/stable/>
- ▶ IPython: an enhanced interactive console
<http://ipython.org/>

Importing Libraries

- Once libraries are installed, they can be used by importing them at the beginning of the code
 - Must be done for every new notebook used
 - Functions within a module can then be referenced using dot notation
 - Module names may also be changed using aliases

```
In [1]: import math  
math.sqrt(9)  
  
Out[1]: 3.0
```

```
In [4]: import math as m  
  
m.sqrt(9)  
  
Out[4]: 3.0
```

- Functions from a module may be referred to directly when importing
 - This eliminates the need for dot notation

```
In [3]: from math import sqrt  
  
sqrt(25)  
  
Out[3]: 5.0
```

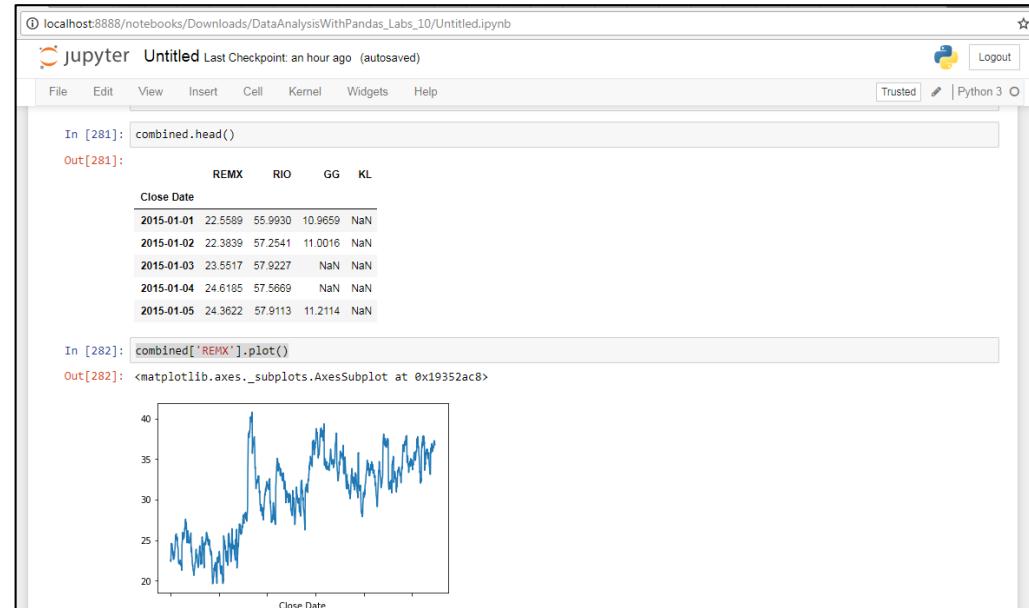
Python IDEs: Jupyter Notebooks

► An open-source web application

- Provides you with an easy-to-use, interactive data science environment across many programming languages

► Not a traditional IDE

- Can also act as a presentation or education tool
- Allows creation and sharing of documents that contain live code, equations, visualizations and narrative text
 - Graphs can be shown in the same document as the code
 - Allows adding of HTML components (e.g., images, videos)



► Final work can be exported to PDF and HTML files

Jupyter Notebook Document

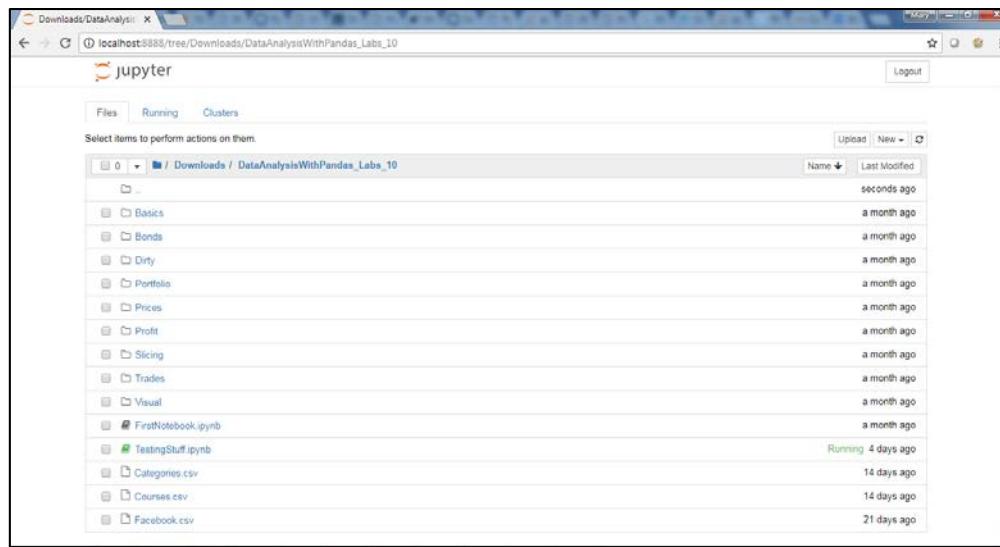
- ▶ **Notebooks are documents produced by the Jupyter Notebook App**
 - Contains both computer code (e.g., Python) and rich text elements (paragraph, equations, figures, links, etc.)
 - Human-readable
 - Can contain a description of the analysis and the results (e.g., narrative, images, etc.)

Jupyter Notebook App

- ▶ **Allows editing and execution of notebooks via a web browser**
 - Can be executed on local desktop, on remote server, or through the internet
- ▶ **Has a “Dashboard” showing local files, which allows opening of Notebooks or shutting down of their kernels**
 - A *kernel* is a “computational engine” that executes the code contained in a Notebook document
 - Here the *kernel* executes python code, but *kernel*s for other languages exist
- ▶ **When Notebook is launched, the associated *kernel* is launched**
 - When the Notebook is executed (either cell-by-cell or all cells), the *kernel* performs the computation and produces the results
 - Depending on the type of computations, the *kernel* may consume significant CPU and RAM
 - RAM is released when the *kernel* is shut down

The Notebook Dashboard

- ▶ Shown when you first launch Jupyter Notebook App
- ▶ Used to open notebook documents
- ▶ Also used to manage the running kernels
 - View or shutdown
- ▶ Similar to a file manager
 - Use to navigate folders
 - Use to rename or delete files



Jupyter Notebook App

- ▶ The Jupyter Notebook App can access only files within its start-up folder (including any sub-folder)
- ▶ The Jupyter Notebook App is a server that appears in your browser at a default address (<http://localhost:8888>)
- ▶ To shut it down, you need to close the associated terminal
 - Closing the browser (or the tab) will not close the Jupyter Notebook App
- ▶ When a notebook is opened, its “computational engine” (*kernel*) is automatically started
 - Closing the notebook browser tab, will not shut down the kernel
 - It will keep running until is explicitly shut down
- ▶ To shut down a kernel, go to the associated notebook and click File/Close and Halt
 - Alternatively, the Notebook Dashboard has a tab named *Running* that shows all the running notebooks (i.e., kernels)
 - Allows shutting them down by clicking a **Shutdown** button

Executing a Notebook

- ▶ Click the notebook name, and it will open in a new browser tab
- ▶ A notebook document can be executed one cell at a time by:
 - <Ctrl><Enter>: execute cell
 - <Shift><Enter>: execute cell and highlight/create cell below
- ▶ The entire notebook can be executed by clicking Cell -> Run All
- ▶ Changes to notebooks are automatically saved every few minutes
- ▶ Care should be taken not to open the same notebook document on many tabs
 - Edits on different tabs can overwrite each other

Saving/Loading Notebooks

Saving/Loading Notebooks

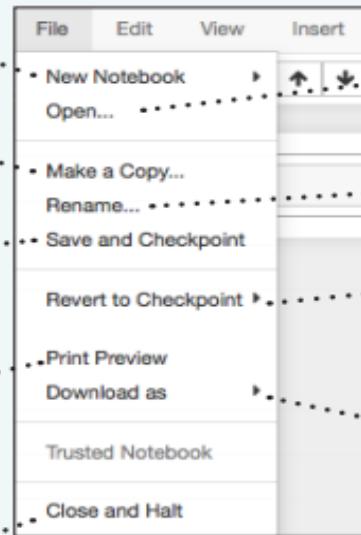
Create new notebook

Make a copy of the current notebook

Save current notebook and record checkpoint

Preview of the printed notebook

Close notebook & stop running any scripts



Open an existing notebook

Rename notebook

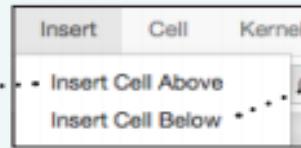
Revert notebook to a previous checkpoint

Download notebook as

- IPython notebook
- Python
- HTML
- Markdown
- reST
- LaTeX
- PDF

Insert Cells

Add new cell above the current one



Add new cell below the current one

Notebook Cells

Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells.

Executing Cells

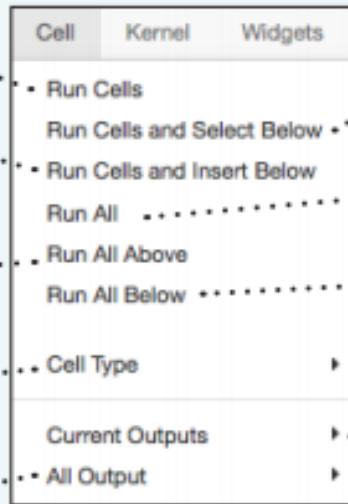
Run selected cell(s)

Run current cells down
and create a new one
above

Run all cells above the
current cell

Change the cell type of
current cell

toggle, toggle
scrolling and clear
all output



Run current cells down
and create a new one
below

Run all cells

Run all cells below
the current cell

toggle, toggle
scrolling and clear
current outputs

Accessing Help

Asking For Help

Walk through a UI tour

Edit the built-in keyboard shortcuts

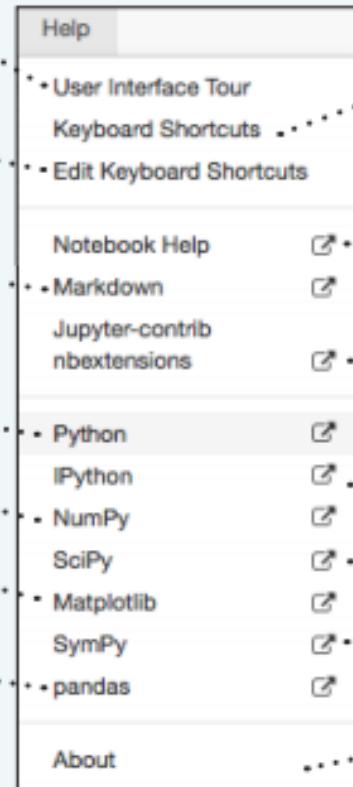
Description of markdown available in notebook

Python help topics

NumPy help topics

Matplotlib help topics

Pandas help topics



List of built-in keyboard shortcuts

Notebook help topics

Information on unofficial Jupyter Notebook extensions

IPython help topics

SciPy help topics

Sympy help topics

About Jupyter Notebook

Contents

- ▶ Defining Data Science
- ▶ Formulating Business Questions
- ▶ Sourcing Data to Understand and Answer Business Questions
- ▶ Introduction to Python

Viewing Datasets in Python

- ▶ Hands-On Exercise 1.1



Importing Data Into Python

- ▶ **Data can be imported from many different sources into Python**
 - CSV files/Excel files
 - Websites
 - Relational databases and data warehouses using ODBC
 - NoSQL data stores (e.g., MongoDB)
 - XML documents
 - JSON files
 - Library option also allows many other types to be processed

CSV = comma-separated value

JSON = JavaScript Object Notation

NoSQL = Not Only Structured Query Language

ODBC = Open Database Connectivity

XML = extensible markup language

Loading Data Into Python: CSV files

- The pandas library provides useful abstractions for dealing with datasets
- To load data from an external (.csv) file into a pandas DataFrame:

```
import os
os.getcwd()      # To view the Current Working Directory
'C:\\\\1264'
```

```
import pandas as pd
iris = pd.read_csv('iris.csv')
iris
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...

- The pandas library also comes with a lot of functionality for ETL
 - Enables succinct expression of data transformations and provides functions to load, unify, and store data from different sources and formats

Importing Data From a Relational Database

- The SQLAlchemy package provides a way of interacting with databases

```
import sqlalchemy as sql

connect_string = 'mysql://mary:password@localhost/classicmodels'

sql_engine = sql.create_engine(connect_string)

query = "select * from customers"
df = pd.read_sql_query(query, sql_engine)

df
```

- Connecting to an Oracle Database—(cx_oracle is a driver) (default drivers are used where this is not specified)

```
sql_engine =
create_engine('oracle+cx_oracle://scott:tiger@tnsname' )
```

- Connecting to a SQLServer Database—(pyodbc is a driver)

```
sql_engine = create_engine('mssql+pyodbc://scott:tiger@mydsn' )
```

Loading Data Into Python: Web Scraping

- The Pandas library has a built-in method to scrape tabular data from html pages called `read_html()`:

Interpret the 1st row
as column headers

```
In [55]: tables = pd.read_html("https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_population", header=0)  
  
tables[1]
```

Out[55]:

Rank	Country(or dependent territory)	Population	Date	% of worldpopulation	Source
0	1	China[Note 2]	1393570000	August 18, 2018	18.2%
1	2	India[Note 3]	1335800000	August 18, 2018	17.5%
2	3	United States[Note 4]	327675000	August 18, 2018	4.29%
3	4	Indonesia	265015300	July 1, 2018	3.47%
4	5	Brazil	209463000	August 18, 2018	2.74%

Display the 2nd
table retrieved from
the webpage

Loading External Data Into Python: Web Scraping

- ▶ Many websites may have data you would like to read programmatically
 - Use caution—attempting to read pages too quickly can get your IP address blocked
- ▶ HTML parsing in Python is simplified with use of a library called *Beautiful Soup*
 - Allows navigating and searching of the parse tree
- ▶ Example: To display all the URLs on a web page:

```
from bs4 import BeautifulSoup
import requests

r = requests.get("http://en.wikipedia.org/wiki/List_of_countries_by_population ")

soup = BeautifulSoup(r.content, "html.parser")

for link in soup.find_all('a'):
    print(link.get('href'))
```

HTML = hypertext markup language

Contents

- ▶ Defining Data Science
- ▶ Formulating Business Questions
- ▶ Sourcing Data to Understand and Answer Business Questions
- ▶ Introduction to Python
- ▶ Viewing Datasets in Python

Hands-On Exercise 1.1



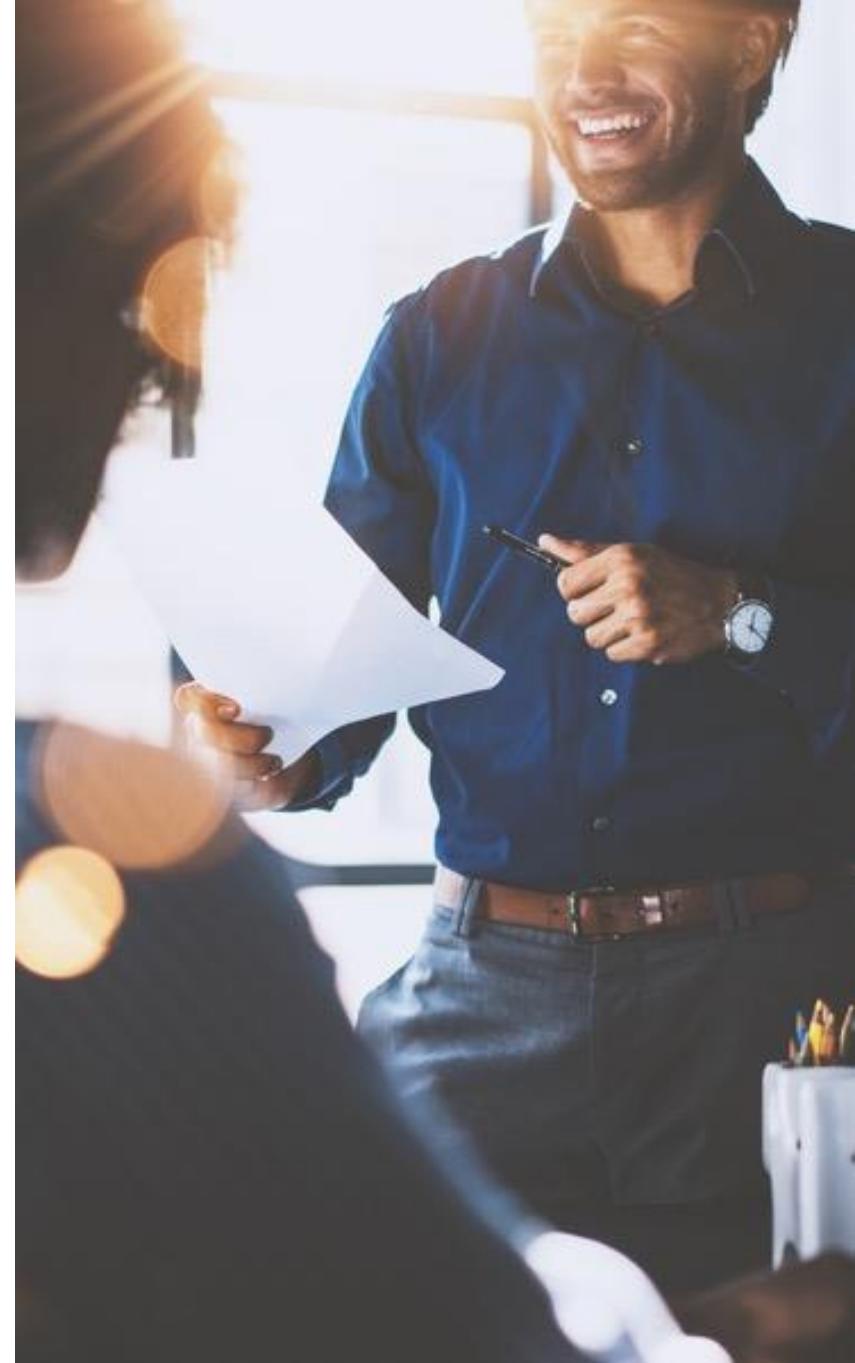


Hands-On Exercise 1.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 1.1: Exploring Jupyter and Loading Data in Python

Objectives

- ▶ **Discuss how to turn business questions into data mining problems**
- ▶ **Explore diverse and wide-ranging data sources that can be used to answer business questions**
- ▶ **Explore the use of Python for Data Analytics and Machine Learning**

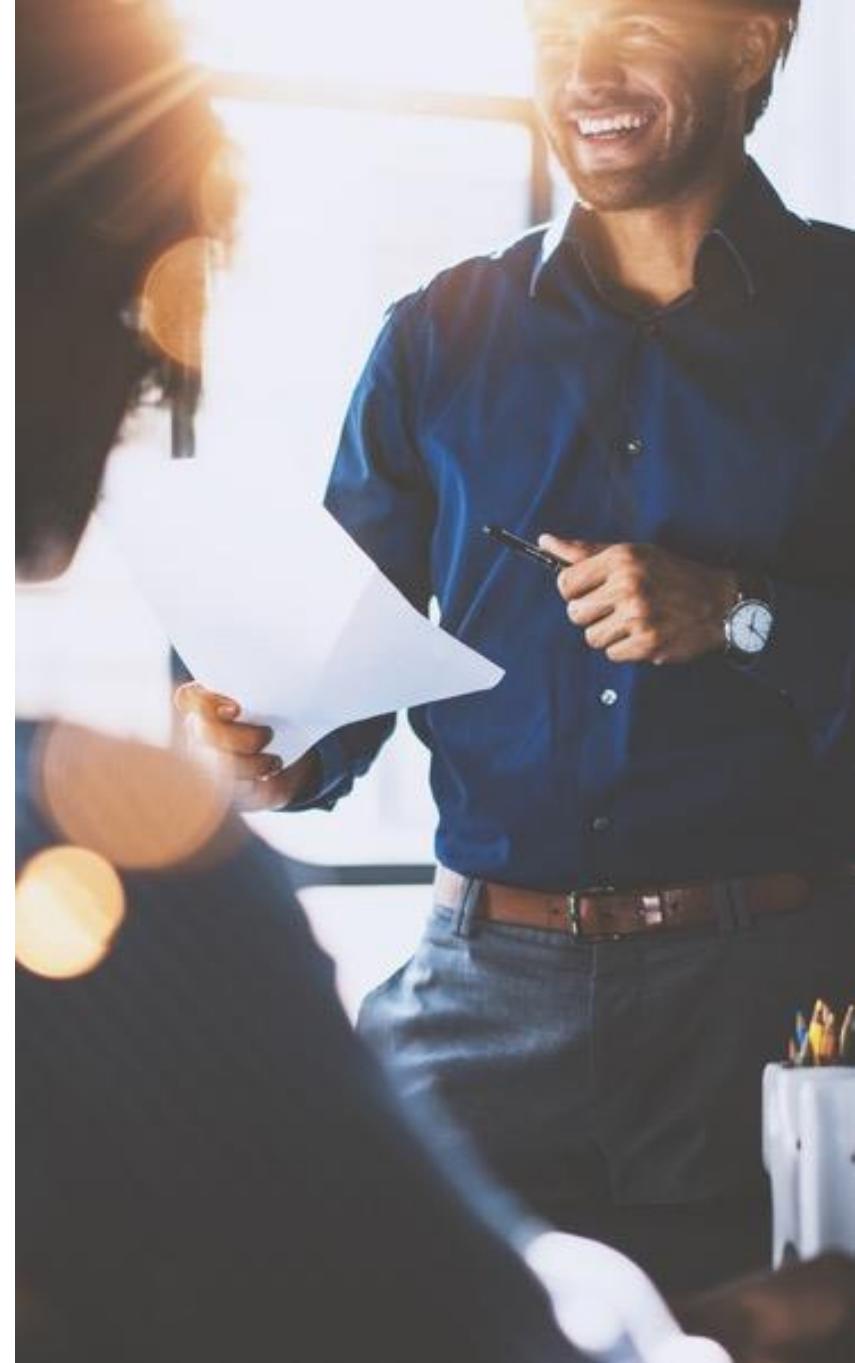


Chapter 2

Exploratory Data Analysis (EDA)

Objectives

- ▶ Explore datasets with Python and pandas
- ▶ Employ data visualization as an aid to understanding datasets
- ▶ Prepare data for analysis



Contents

Exploring Data With pandas

- ▶ Hands-On Exercise 2.1
- ▶ Exploratory Data Analysis
- ▶ Visualizing Data
- ▶ Hands-On Exercise 2.2
- ▶ Transforming Data
- ▶ Hands-On Exercise 2.3



Exploratory Analysis Using the pandas Library

- ▶ **pandas is the most widely used library in Python for data munging/analysis**
 - Provides flexible data structures that make working with data easier and more intuitive
- ▶ **Allows for practical, real-world data analysis in Python**
- ▶ **There are two primary data structures in pandas**
 - Easier to work with than other data structures in Python (lists, dictionaries, etc., requiring For... Loops)
 - Series (one-dimensional)
 - DataFrame (two-dimensional)
- ▶ **There are other data structures in Python for storing data values in memory, but we will focus mainly on the above two in this course**
 - Lists (ordered sequences of values)
 - Tuples (ordered, immutable sequences of values)
 - Sets (unordered bags of values)
 - Dictionaries (unordered bags of key-value pairs)

Series

- **A one-dimensional array capable of holding any data type**
 - All values in a single series must be of the same data type

```
Out[26]: 0    1.331587  
          1    0.715279  
          2   -1.545400  
          3   -0.008384  
          4    0.621336  
          5   -0.720086  
          6    0.265512  
          7    0.108549  
          8    0.004291  
          9   -0.174600  
         10    0.433026
```

Index



DataFrame

- ▶ A two-dimensional array stored as a collection of series
- ▶ A tabular data structure comprised of rows and columns
 - Similar to a spreadsheet or database table
- ▶ A DataFrame can be used to represent an entire dataset
 - The most commonly used structure in data exploration

The diagram illustrates the structure of a DataFrame. At the top, a code snippet shows 'iris = pd.read_csv('iris.csv')' followed by 'iris'. Below this, a table represents the data. The table has a header row with column names: 'sepal_length', 'sepal_width', 'petal_length', 'petal_width', and 'species'. The data rows are indexed from 0 to 7. An arrow labeled 'Index' points to the first column of the data. Another arrow points from a box labeled 'Column names' to the 'species' column header. The data rows are as follows:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa

DataFrame Indexes

- By default the DataFrame will automatically create a zero-based index for the dataset
 - Here the weather dataset has been loaded with the default zero-based index
 - To make a column from the dataset, the index, use the `.set_index()` method

Zero-based Index

The diagram shows a DataFrame with three rows and nine columns. The columns are labeled: Date, Location, MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, WindGustDir. The rows are indexed 0, 1, and 2. Row 0 has Date 2007-11-01, Location Canberra, MinTemp 8.0, MaxTemp 24.3, Rainfall 0.0, Evaporation 3.4, Sunshine 6.3, and WindGustDir NW. Row 1 has Date 2007-11-02, Location Canberra, MinTemp 14.0, MaxTemp 26.9, Rainfall 3.6, Evaporation 4.4, Sunshine 9.7, and WindGustDir ENE. Row 2 has Date 2007-11-03, Location Canberra, MinTemp 13.7, MaxTemp 23.4, Rainfall 3.6, Evaporation 5.8, Sunshine 3.3, and WindGustDir NW.

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
0	2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW
1	2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE
2	2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW

```
weather.set_index('Date', inplace=True)
```

Avoids copy being created

New Index

The diagram shows the same DataFrame as above, but with the columns rearranged. The Date column is now at the top, followed by Location, MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine, and WindGustDir. The rows are indexed 0, 1, and 2. Row 0 has Date 2007-11-01, Location Canberra, MinTemp 8.0, MaxTemp 24.3, Rainfall 0.0, Evaporation 3.4, Sunshine 6.3, and WindGustDir NW. Row 1 has Date 2007-11-02, Location Canberra, MinTemp 14.0, MaxTemp 26.9, Rainfall 3.6, Evaporation 4.4, Sunshine 9.7, and WindGustDir ENE. Row 2 has Date 2007-11-03, Location Canberra, MinTemp 13.7, MaxTemp 23.4, Rainfall 3.6, Evaporation 5.8, Sunshine 3.3, and WindGustDir NW.

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
Date							
2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW
2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE
2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW

Setting an Index on Data Load

- The index may also be set when the data is being loaded using the `index_col` parameter

```
weather = pd.read_csv('weather.csv', index_col=0)  
weather
```

Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir
2007-11-01	Canberra	8.0	24.3	0.0	3.4	6.3	NW
2007-11-02	Canberra	14.0	26.9	3.6	4.4	9.7	ENE
2007-11-03	Canberra	13.7	23.4	3.6	5.8	3.3	NW

This indicates that the 1st column in the .csv file should be used as the index

(Index column positions are zero-based)

Examining the Data's Structure

- ▶ A DataFrame has methods and attributes which can be called on, to examine the data's structure
 - Methods are called with parentheses
 - Attributes are called without parentheses
- ▶ To examine the columns of the iris dataset, use the .columns attribute

```
iris.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

- ▶ The dimensions of the dataset are given by the .shape attribute
 - This example shows that the iris dataset has 150 rows and 5 columns

```
iris.shape  
(150, 5)
```

Examining the Data's Structure

- To examine the index of a DataFrame, use the `index` attribute

```
iris.index
```

```
RangeIndex(start=0, stop=150, step=1)
```

- To examine the data types of the DataFrame, use the `.dtypes` attribute or for a summary of the DataFrame's structure, use the `.info()` method

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

- When a dataset is loaded into a pandas DataFrame, it will infer the data types
 - Numbers will be stored in a numeric datatype (i.e., `int64` or `float64`)
 - Text will be stored in an object datatype, etc.

Data Types

- There are many basic data types in Python

Data Type	Usage
Object	Text
int64	Integer numbers
float64	Floating point numbers
datetime64	Date and time values
timedelta[]	Differences between two date times
bool	True/False values

- Data can be assigned to variables using the assignment operator =
 - For example, varA = 2
 - varB = 'Hello'
 - Strings
 - Can be defined with either single quotes (') or double quotes (")
 - Can concatenate strings using the + operator

Lists

- There are other data structures in Python for storing data beyond pandas DataFrames, i.e., tuples, sets, dictionaries
- We will use lists (one such structure) quite frequently in this course
 - Lists are quite flexible data structures
 - Can be used to store any combination of data values together

```
new_list = ['a', 'b', 'python', 'c', 'data science']  
new_list
```

```
['a', 'b', 'python', 'c', 'data science']
```

```
new_list[0]
```

```
'a'
```

```
new_list[-3]
```

```
'python'
```

```
new_list[4]
```

```
'data science'
```

```
new_list[1:3]
```

```
['b', 'python']
```

```
new_list[-1]
```

```
'data science'
```

Dictionaries

► A **dictionary** is an unordered set of key-value pairs

- When a key is added to a dictionary, a value must also be added for that key
 - Can be changed later
 - Optimized for retrieving the value when the key is known, but not the other way around
- Cannot have duplicate keys
 - Assigning a value to an existing dictionary key simply replaces the old value with the new one

```
new_dict = {'language': 'python', 'database': 'mysql'}  
new_dict
```

```
{'database': 'mysql', 'language': 'python'}
```

```
new_dict['language']
```

```
'python'
```

```
new_dict['database']
```

```
'mysql'
```

Displaying the Data in a DataFrame

- To display the contents of the DataFrame, use the `print()` function:

```
print(iris)
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa

- In Jupyter notebooks, simply typing the DataFrame name results in nicely formatted outputs

- Displays only 20 columns by default for wide data DataFrames, and only 60 or so rows, truncating the middle section

```
iris
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa

Previewing the Data

- A small selection of rows from the DataFrame can be previewed with the `head()` and `tail()` methods

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

Retrieving Data Subsets Using the [] Indexer

- ▶ A column may be retrieved by specifying the name within single square parentheses
 - Single brackets return a series

```
iris['sepal_length']
```

```
0    5.1
1    4.9
2    4.7
3    4.6
4    5.0
5    5.4
```

- ▶ To retrieve a subset of the column rows, a range may be specified as follows:

```
iris['sepal_length'][0:3]
```

```
0    5.1
1    4.9
2    4.7
Name: sepal_length, dtype: float64
```

Retrieving Data Subsets Using the [] Indexer

- To retrieve multiple columns from the DataFrame, a list of values should be passed to the indexer which results in double square brackets being used
 - This will return a DataFrame
 - Single brackets would result in an error

```
iris[['sepal_length', 'sepal_width']]
```

	sepal_length	sepal_width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

Using the [] Indexer with a Filtering Condition

- To filter rows, specify a Boolean condition that must be met before rows are returned
 - Returns a series of True and False data with which to filter rows
 - Multiple conditions may be specified with an & symbol

```
iris[iris['sepal_length']>7.6]
```

	sepal_length	sepal_width	petal_length	petal_width	species
117	7.7	3.8	6.7	2.2	Iris-virginica
118	7.7	2.6	6.9	2.3	Iris-virginica
122	7.7	2.8	6.7	2.0	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica

- To restrict columns being retrieved, specify columns required as follows:

```
iris[iris['sepal_length']>7.6][['species','sepal_length']]
```

	species	sepal_length
117	Iris-virginica	7.7
118	Iris-virginica	7.7
122	Iris-virginica	7.7
131	Iris-virginica	7.9
135	Iris-virginica	7.7

Retrieving Data Subsets Using the .iloc[] Indexer

- ▶ Selections using the .iloc[] indexer are based on the numeric position of the row in the DataFrame
- ▶ Syntax:

```
df_name.iloc[<row selection>, <column selection>]
```

- | | |
|--|--|
| <ul style="list-style-type: none">• Single row position
e.g., 2 (i.e., 3rd row)• List of row positions
e.g., [2,3,4]• Range/slice of rows
e.g., [2:5] | <ul style="list-style-type: none">• Optional• Single column position
e.g., 1• List of column positions
e.g., [0,1,2]• Range/Slice of column positions
e.g., [1:4] |
|--|--|

- ▶ Examples:

```
iris.iloc[3]      # Single row
iris.iloc[[3,4]] # Multiple rows
iris.iloc[3,1]    # Single row, single column
iris.iloc[3,[1,2]] # Single row, multiple columns
iris.iloc[[3,4],[1,2]] # Multiple row, multiple columns
iris.iloc[3,1:3]  # Single row, range of columns
iris.iloc[:,[1,2]] # All rows, multiple columns
```

Examples Using the .loc[] Indexer

Rank	Population	Date	% of worldpopulation	Source
Country				
Indonesia	4 265015300	July 1, 2018	3.47%	Official annual projection
Brazil	5 208721442	August 21, 2018	2.73%	Official population clock
Pakistan	6 201174754	August 21, 2018	2.63%	Official population clock
Nigeria	7 197227340	August 21, 2018	2.58%	Official population clock
Bangladesh	8 167008838	August 21, 2018	2.18%	Official population clock

```
Countries.loc['Brazil']          # Single row
Countries.loc[['Indonesia', 'Brazil']] # Multiple rows
Countries.loc[Countries['Rank'] == '3'] # Row based on filter condition
Countries.loc['Brazil','Rank']      # Single row and single column
Countries.loc['Brazil',['Rank','Population']] # Single row, multiple columns
Countries.loc[['Indonesia','Brazil'],['Rank','Population']] # Multiple rows,
                                                               # multiple columns
Countries.loc['Brazil','Rank':'Source'] # Single row, range of columns
```

Dropping Columns from a DataFrame

- Columns are dropped from a DataFrame by specifying the axis parameter as 1

```
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.drop('species',axis=1).head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Contents

- ▶ Exploring Data With pandas

Hands-On Exercise 2.1

- ▶ Exploratory Data Analysis
- ▶ Visualizing Data
- ▶ Hands-On Exercise 2.2
- ▶ Transforming Data
- ▶ Hands-On Exercise 2.3





Hands-On Exercise 2.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 2.1: Working With Data in Python

Contents

- ▶ Exploring Data With pandas
- ▶ Hands-On Exercise 2.1

Exploratory Data Analysis

- ▶ Visualizing Data
- ▶ Hands-On Exercise 2.2
- ▶ Transforming Data
- ▶ Hands-On Exercise 2.3

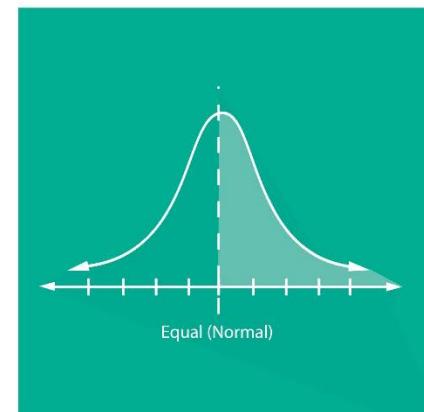
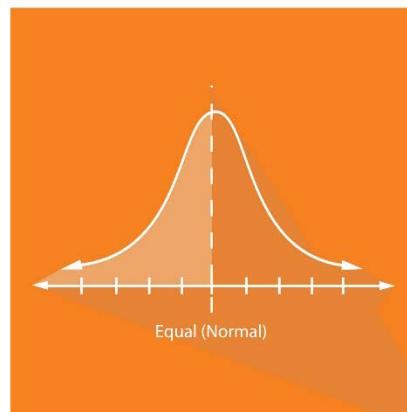
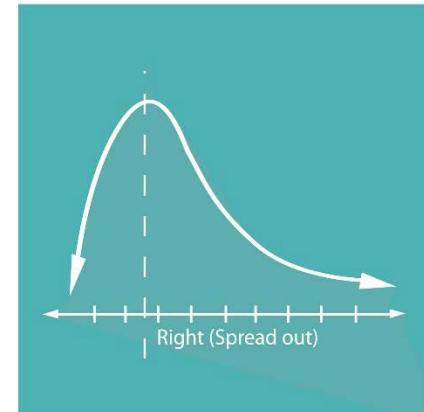
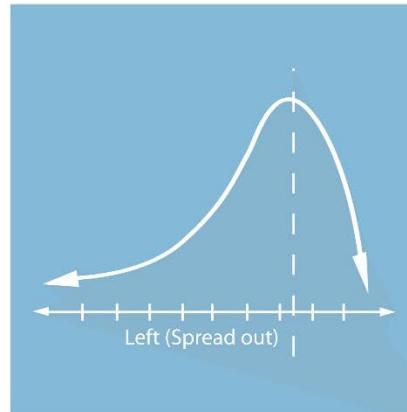


Exploratory Data Analysis

- ▶ Before we begin to model the data, we need to understand the nature of it
- ▶ This is typically done through a mixture of statistical and graphical techniques
- ▶ It helps to:
 - See the underlying structure of the data
 - Detect outliers
 - Spot data quality issues and missing data
 - Help find patterns in the data
 - Help in developing hypotheses about the real-world event that the data is capturing

Exploratory Data Analysis in Summary

- Descriptive statistics summarize a given dataset
- Include measures of central tendency and measures of variability (spread)
- Measures of central tendency include the mean, median, and mode
- Measures of variability include the standard deviation and variance
 - May also include the minimum and maximum, as well as the kurtosis and skewness



Central Tendency

- **The following are three measurements of central tendency**
 - *Mean*: The sum of observed data points divided by the number of data records

```
iris['sepal_length'].mean()  
5.843333333333335
```
 - *Median*: The middle data point (or average of the two middle data points if there is an even number of observations) when all data points are lined up in either ascending or descending order

```
iris['sepal_length'].median()  
5.8
```
 - *Mode*: The most frequently occurring data point value in a dataset

```
iris['sepal_length'].mode()  
0    5.0  
dtype: float64
```

Measuring Dispersion

- **Dispersion refers to the spread of data about the data's center and can be measured by:**

- *Variance*: A cumulative measure of individual data points' distance from the mean

```
iris['sepal_length'].var()
```

```
0.6856935123042505
```

- *Standard deviation* is an average deviation of the data from the mean
 - Calculated as the square root of the variance:

```
iris['sepal_length'].std()
```

```
0.8280661279778629
```

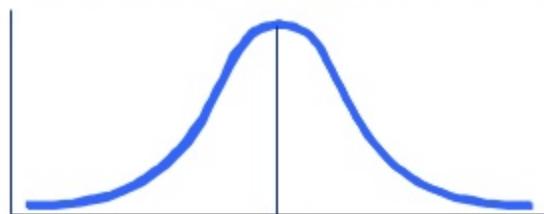
describe() Function

- The describe method displays a summary of some distribution measures

```
iris.describe()
```

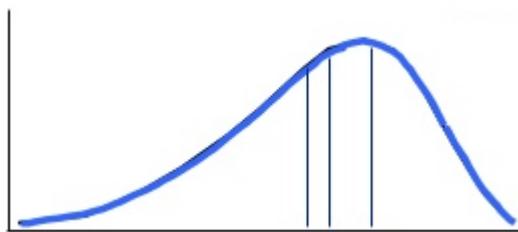
	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Skewness



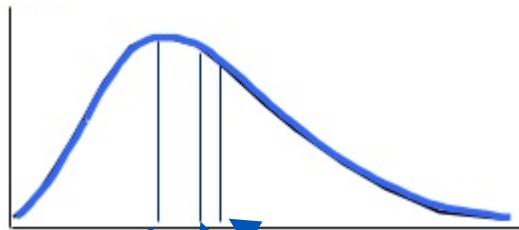
Normal Distribution

Mode = Mean = Median



Skewed Left

Mean Median Mode



Skewed Right

Mode Median Mean

Distributions

► Other important distributions

- *Poisson*: The distribution of counts that occur at a certain “rate”
 - Observed frequency of a given term in a corpus
 - Number of visits to a website in a fixed-time interval
 - Number of website clicks in an hour
- *Binomial/multinomial*: The number of counts of events (e.g., die tosses = 6) out of n trials

Contents

- ▶ Exploring Data With pandas
- ▶ Hands-On Exercise 2.1
- ▶ Exploratory Data Analysis

Visualizing Data

- ▶ Hands-On Exercise 2.2
- ▶ Transforming Data
- ▶ Hands-On Exercise 2.3



Visualizing Data

- ▶ When interpreting or presenting findings on data distribution, it is helpful to provide a visual of the data's distribution in addition to the numbers
- ▶ For example, a histogram shows the frequency (count) of values in the dataset over a series of intervals that covers the range of the data
 - A histogram is a great way of initially visualizing the data's distribution
 - Gives a sense of the central tendency and dispersion of data around that center before calculating any statistics
- ▶ The `matplotlib` package is the most common package used for visualizations in Python

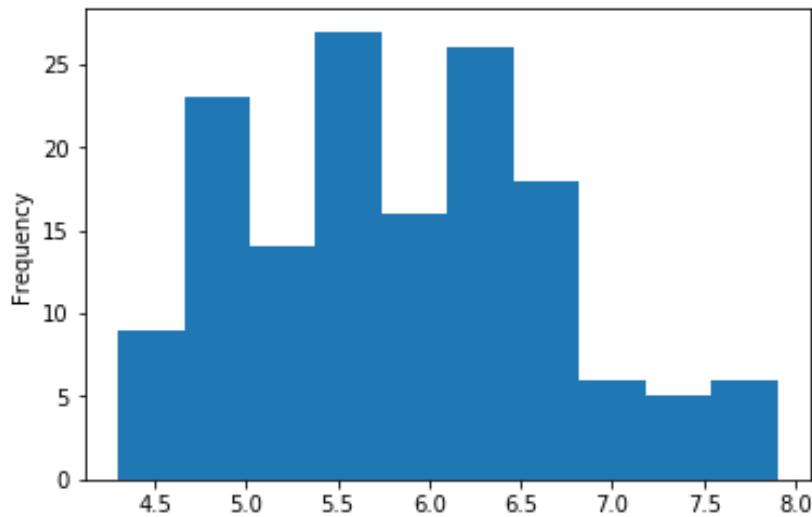
pandas Plotting Methods

- ▶ **pandas plotting methods are convenient wrappers around matplotlib calls**
 - For example, the plot method on a Series or DataFrame is a simple wrapper around plt.plot()
 - Uses defaults and auto-formats in order to simplify plotting requirements
 - Will need to use matplotlib.pyplot directly to add figure labels and axis labels to your diagrams
- ▶ **Line plots are the default**
 - Other plot styles can be generated by specifying the type in the kind keyword argument to plot():
 - 'bar' or 'barh' for bar plots
 - 'hist' for histogram
 - 'box' for boxplot
 - 'kde' or 'density' for density plots
 - 'area' for area plots
 - 'scatter' for scatter plots
 - 'pie' for pie plots

Visualizing Numeric Data: Histograms

- A histogram is a visual display of data using rectangles whose area is proportional to the frequency of a variable and whose width is equal to the class interval
 - Note: In Jupyter, the command sometimes needs to be executed a second time, before the plot will display

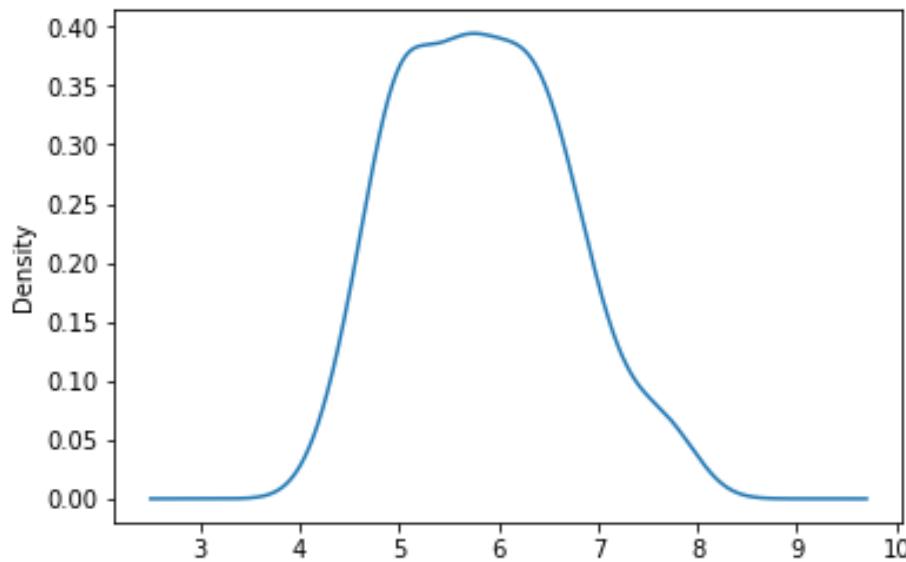
```
iris['sepal_length'].plot(kind='hist')  
<matplotlib.axes._subplots.AxesSubplot at 0x16cdfc18>
```



Visualizing Numeric Data: Density Plots

- ▶ Kernel density plots are a much more effective way to view the distribution of a variable

```
iris['sepal_length'].plot(kind='kde')  
<matplotlib.axes._subplots.AxesSubplot at 0x101d5240>
```



Visualizing Numeric Data: Box Plots

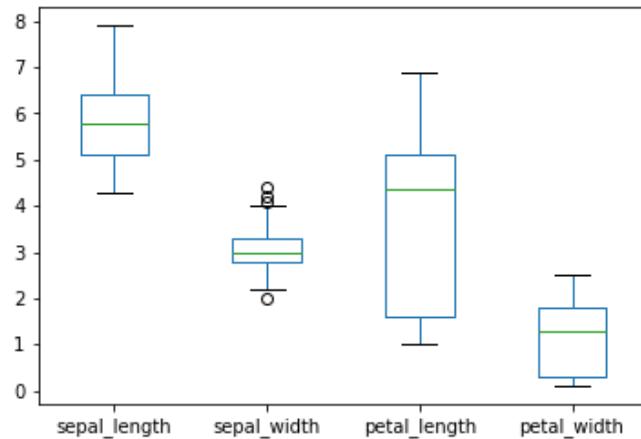
► A box plot illustrates the distribution of the data

- Made up of: median, minimum value, maximum value, Quartile 1 value, and Quartile 3 value
- Lower 25 percent (Quartile 1) of the data occurs between the bottom edge of the box and the bottom edge of the lower whisker
- Upper 25 percent (Quartile 3) of the data occurs between the top edge of the box and the top edge of the upper whisker

► To display a box plot:

```
iris.plot(kind='box')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x16ca32b0>
```



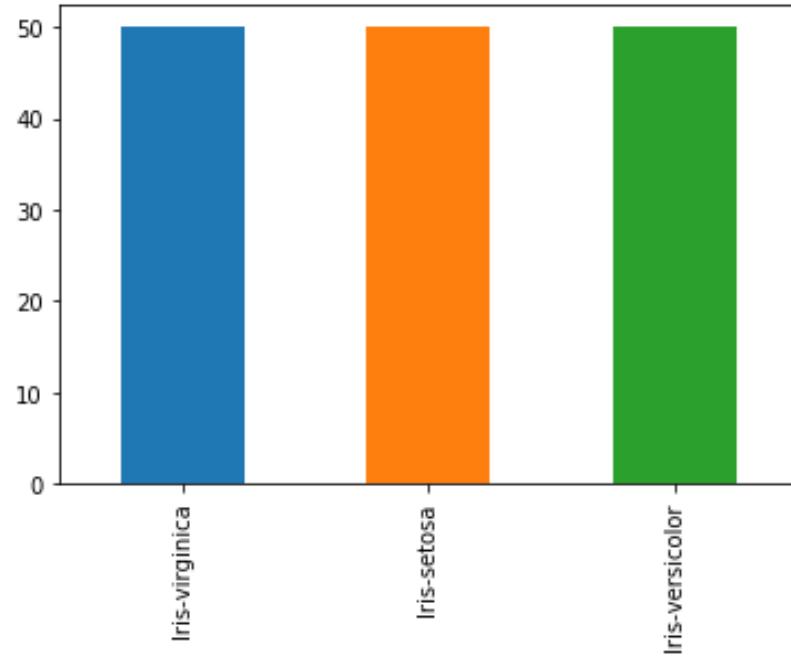
A side-by-side box plot can be used to visually compare the central tendencies of multiple datasets using the middle line of the box (representing the median value of the dataset)

Visualizing Categorical Data: Bar Charts

- Bar Charts display counts of the number of observations using vertical bars for each categorical variable

```
iris['species'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x17893160>
```



Refining Your Business Question

- ▶ After exploring your data with descriptive statistics, you may want to modify or refine your central question



Contents

- ▶ Exploring Data With pandas
- ▶ Hands-On Exercise 2.1
- ▶ Exploratory Data Analysis
- ▶ Visualizing Data

Hands-On Exercise 2.2

- ▶ Transforming Data
- ▶ Hands-On Exercise 2.3



Hands-On Exercise 2.2

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 2.2: Exploring and Visualizing Data in Python



Contents

- ▶ Exploring Data With pandas
- ▶ Hands-On Exercise 2.1
- ▶ Exploratory Data Analysis
- ▶ Visualizing Data
- ▶ Hands-On Exercise 2.2

Transforming Data

- ▶ Hands-On Exercise 2.3



Cleaning and Transforming the Data

- ▶ **We will need to clean and transform raw data in different ways in order to model it**
 - This is particularly true when dealing with unstructured data
- ▶ **Raw data may come in varied formats**
 - Machine-generated log files
 - Excel spreadsheets from the HR department
 - JSON data retrieved from the LinkedIn Application Programming Interface (API)
 - Downloaded academic papers
- ▶ **Regardless of its raw form, data needs to be cleaned and transformed to a structured state (similar to relational databases)**
 - Variables should be in separate columns
 - Observations should be in separate rows

Cleaning and Transforming the Data

- ▶ **Typical transformation tasks are**
 - Rescaling the data
 - Variables need to be on a similar scale to be compared
 - Dealing with missing values
 - Replacing with mean, median, mode, constants or special symbols
 - Remapping
 - Allows various types of binning, log transformation, etc.
- ▶ **It is advisable to script required data changes as evidence of results**

Rescaling

► Normalization

- Rescaling numeric attributes into the range 0 and 1

```
from sklearn import preprocessing  
  
preprocessing.normalize(iris.iloc[:,[0,1,2,3]])  
  
Out[44]: array([[0.80377277, 0.55160877, 0.22064351, 0.0315205 ],  
                 [0.82813287, 0.50702013, 0.23660939, 0.03380134],  
                 [0.80533308, 0.54831188, 0.2227517 , 0.03426949],  
                 [0.80003025, 0.53915082, 0.26087943, 0.03478392],  
                 [0.790965 , 0.5694948 , 0.2214702 , 0.0316386 ],  
                 [0.78417499, 0.5663486 , 0.2468699 , 0.05808704],  
                 [0.78010936, 0.57660257, 0.23742459, 0.0508767 ],  
                 [0.80218492, 0.54548574, 0.24065548, 0.0320874 ],  
                 [0.80642366, 0.5315065 , 0.25658935, 0.03665562],  
                 [0.81803119, 0.51752994, 0.25041771, 0.01669451],  
                 [0.80373519, 0.55070744, 0.22325977, 0.02976797],  
                 [0.786991 , 0.55745196, 0.26233033, 0.03279129],  
                 [0.82307218, 0.51442011, 0.24006272, 0.01714734],  
                 [0.8025126 , 0.55989251, 0.20529392, 0.01866308],  
                 [0.81120865, 0.55945424, 0.16783627, 0.02797271],  
                 [0.77381111, 0.59732787, 0.2036345 , 0.05430253],  
                 [0.79428944, 0.57365349, 0.19121783, 0.05883625],  
                 [0.80327412, 0.55126656, 0.22050662, 0.04725142],  
                 [0.8068282 , 0.53788547, 0.24063297, 0.04246464],  
                 [0.77064222, 0.50001402, 0.22020012, 0.0450617 ]])
```

Missing Values

- ▶ Missing values need to be treated with care as they can affect or skew analysis if not treated correctly
- ▶ Missing data in Python appears as NaN
 - NaN is not a string or a numeric value

```
titanic = pd.read_csv('titanic.csv')
titanic['Age'][25:30]
```

```
25    38.0
26    NaN
27    19.0
28    NaN
29    NaN
Name: Age, dtype: float64
```

Missing Values

- Use `.isnull()` to detect missing values

```
titanic['Age'][25:30].isnull()
```

```
25    False
26    True
27    False
28    True
29    True
Name: Age, dtype: bool
```

Imputation of Missing Values

- ▶ Allows replacement of missing data with substitute values
- ▶ Zero/mean/median/mode or a constant can be imputed for missing values
- ▶ The `fillna()` method can “fill in” NA values with non-null data
- ▶ Example:

```
titanic['Age'][25:30]
```

```
25    38.0
26    NaN
27    19.0
28    NaN
29    NaN
Name: Age, dtype: float64
```

```
titanic['Age'][25:30].fillna(0)
```

```
25    38.0
26     0.0
27    19.0
28     0.0
29     0.0
Name: Age, dtype: float64
```

- ▶ Other “fill” methods can be used in a similar way:
 - `ffillna()`
 - `bfillna()`
 - `mean()`
 - `dropna()`

Remapping/Recode Operations

- Categorical data may be encoded as follows:

```
titanic.insert(5, 'GenderCode', pd.Categorical(titanic['Sex']).codes)
```

```
titanic[['Sex', 'GenderCode']]
```

	Sex	GenderCode
0	male	1
1	female	0
2	female	0
3	female	0
4	male	1
5	male	1

pandas Methods

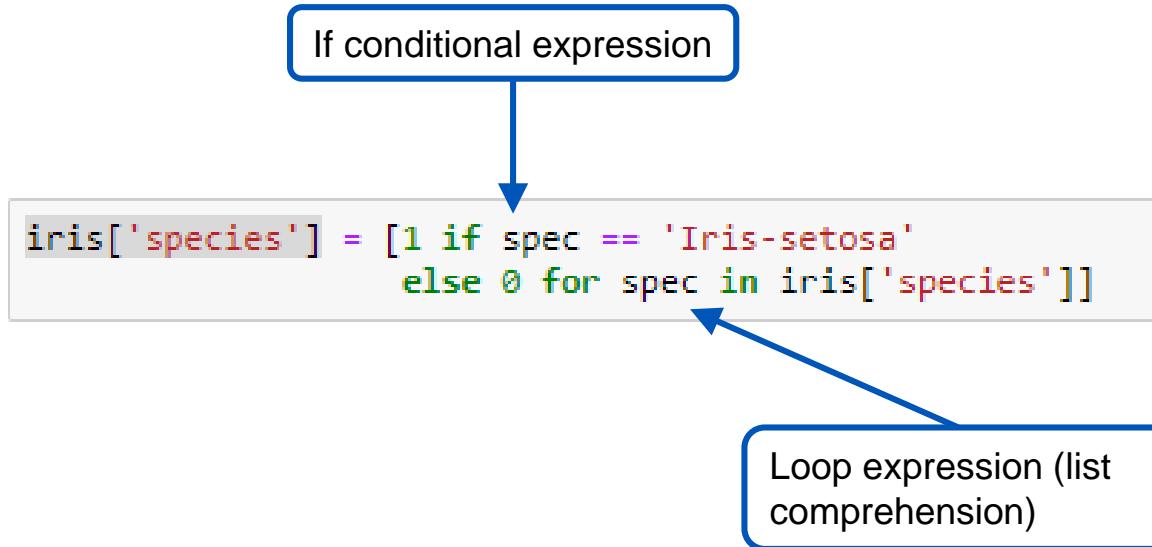
- The pandas libraries have many functions and methods that can be used for data transformation
 - Some additional pandas methods we will use in this course:

Conversion	
.as_type(dtype)	.to_datetime()
.to_numeric()	.to_timestamp()
Numeric	
.round()	.diff()
.div()	
String	
.str.upper()	.str.replace()
.str.lower()	.str.format()

- For an extensive list of pandas functions and methods:
 - <https://pandas.pydata.org/pandas-docs/stable/api.html>

Looping Over Elements in a Column: List Comprehensions

- ▶ List comprehensions can also be used to carry out replacements



Data Preparation: Data Type Conversion

- Data types of the DataFrame columns may be converted .astype()

```
a = [['x', '1.5', '6.3'], ['y', '20', '0.15'], ['z', '8', '0.4']]  
df = pd.DataFrame(a, columns = ['A', 'B', 'C'])  
df
```

	A	B	C
0	x	1.5	6.3
1	y	20	0.15
2	z	8	0.4

```
df.dtypes
```

```
A    object  
B    object  
C    object  
dtype: object
```

Converts columns B and C to a float datatype

```
df[['B', 'C']] = df[['B', 'C']].astype(float)  
df.dtypes
```

```
A    object  
B    float64  
C    float64  
dtype: object
```

Grouping Rows By a Shared Value

- The `groupby()` operation must always be paired with the function (i.e., `mean()`, `sum()`, `count()`, etc.) that is to be applied to the resulting group
 - Otherwise summary info only will be displayed

```
iris.groupby('species').count()
```

species	sepal_length	sepal_width	petal_length	petal_width
Iris-setosa	50	50	50	50
Iris-versicolor	50	50	50	50
Iris-virginica	50	50	50	50

Data Preparation: Matrix Manipulation—concat()

- ▶ You will frequently need to combine datasets in different ways
- ▶ The concat() function combines DataFrames by rows
`concat([dataframeA, dataframeB])`

df1

	Subtype	Gender	Expression
0	A	M	-0.54
1	A	F	-0.80
2	B	M	-1.03
3	C	M	-0.41

`pd.concat([df1, df2])`

	Subtype	Gender	Expression
0	A	M	-0.54
1	A	F	-0.80
2	B	M	-1.03
3	C	M	-0.41
0	D	F	2.50
1	D	F	-0.15
2	D	M	-0.20
3	D	F	-0.04

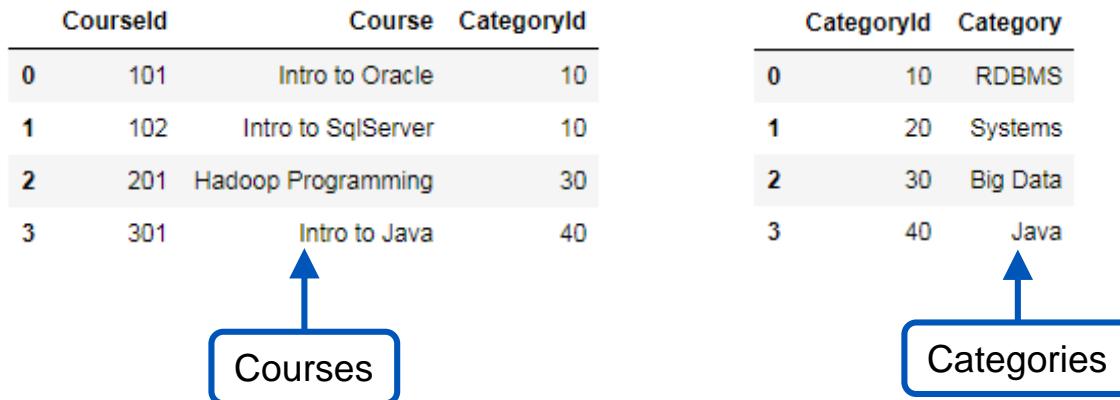
df2

	Subtype	Gender	Expression
0	D	F	2.50
1	D	F	-0.15
2	D	M	-0.20
3	D	F	-0.04



Merging DataFrames

- The `merge()` function joins two DataFrames by common column(s)
 - Can also be called as a method (e.g., `df1.merge(df2)`)



```
pd.merge(Courses, Categories, on='CategoryId', how='inner')
```

	CourseId	Course	CategoryId	Category
0	101	Intro to Oracle	10	RDBMS
1	102	Intro to SqlServer	10	RDBMS
2	201	Hadoop Programming	30	Big Data
3	301	Intro to Java	40	Java

Functions

- ▶ User-defined functions can also be easily added in Python
- ▶ To master some of the more advanced analytics examples in this course, you need a solid foundation in how Python functions work
- ▶ The structure of a function is:

```
def functionname( parameters ):  
    executable code  
    return [expression]
```

- Example:

```
def firstfunction():  
    str = 'Hello World'  
    return str
```

```
firstfunction()
```

```
'Hello World'
```

Lambda Functions

- **Used for creating one-time anonymous functions**
 - Allow creation of a function when it is needed only and then applied to the relevant data set.

- **Syntax:**

```
lambda arguments : expression
```

- **Example:**

```
arguments      expression
add = lambda a, b : a + b
print (add(2, 3))
```

5

- Can have any number of arguments but only one expression
- Cannot contain any statements
- Returns a function object
- Often used as a parameter to a method such as .apply()

Looping Over Elements in a Column: The .apply() Method

- ▶ Used to execute either a Python built-in function or a custom-defined function across every element of a dataset
- ▶ What if the Fare price paid on the Titanic had been \$1 more than they were?
 - In this simplistic example, the lambda function takes as input the Fare price from the DataFrame, adds 1 to it, returning the new price

```
titanic = pd.read_csv('titanic.csv')
titanic['Fare'].head()
```

```
0    7.2500
1   71.2833
2    7.9250
3   53.1000
4    8.0500
Name: Fare, dtype: float64
```

```
titanic['Fare'].apply(lambda price: 1 + price).head()
```

```
0    8.2500
1   72.2833
2    8.9250
3   54.1000
4    9.0500
Name: Fare, dtype: float64
```

Principal Component Analysis

- ▶ To reduce the size of a dataset, the technique of Principal Component Analysis (PCA) is sometimes used
 - Improves the efficiency of many data modeling techniques (e.g., regression, clustering, classification) if less data needs to be dealt with
 - Relationships between numeric variables are used to replace large numbers of variables with fewer variables (called *principal components*)

PCA Example

► We will use the `iris` dataset as an example

- The `Species` variable has been removed
 - PCA works on numeric data

```
from sklearn.decomposition import PCA  
  
irisCopy = iris.drop('species', axis=1)  
  
pca = PCA()  
pca.fit(irisCopy)  
pca.explained_variance_ratio_  
  
array([0.92461872, 0.05306648, 0.01710261, 0.00521218])  
  
pd.DataFrame(pca.components_, columns=irisCopy.columns)
```

	sepal_length	sepal_width	petal_length	petal_width
0	0.361387	-0.084523	0.856671	0.358289
1	0.656589	0.730161	-0.173373	-0.075481
2	-0.582030	0.597911	0.076236	0.545831
3	-0.315487	0.319723	0.479839	-0.753657

PCA Example

- ▶ **The explained_variance_ratio function can be used to display the proportions of the total variance explained by each component**
 - 92.4% of the variation in the dataset is explained by the first component
 - The first two components cumulatively explain 97.7 percent of the variability in data
 - We can ignore the second two components
- ▶ **The components_ attribute of the principal components object display the coefficients of the new variables**
 - PCA found four new variables which can explain the same information as the original four variables (Sepal.Length, Sepal.Width, Petal.Length, Petal.Width), which are 0 to 3

```
Comp.0 = 0.361 * Sepal.Length - 0.0845 * Sepal.Width + 0.857 *  
Petal.Length + 0.358 * Petal.Width
```

PCA in Use

- ▶ **These new principal components are linear combinations of the initial variables**
 - They attempt to represent a larger dataset with a smaller dataset (without loss of meaning)
 - This will reduce the complexity of some of the data mining processes

Contents

- ▶ Exploring Data With pandas
- ▶ Hands-On Exercise 2.1
- ▶ Exploratory Data Analysis
- ▶ Visualizing Data
- ▶ Hands-On Exercise 2.2
- ▶ Transforming Data

Hands-On Exercise 2.3



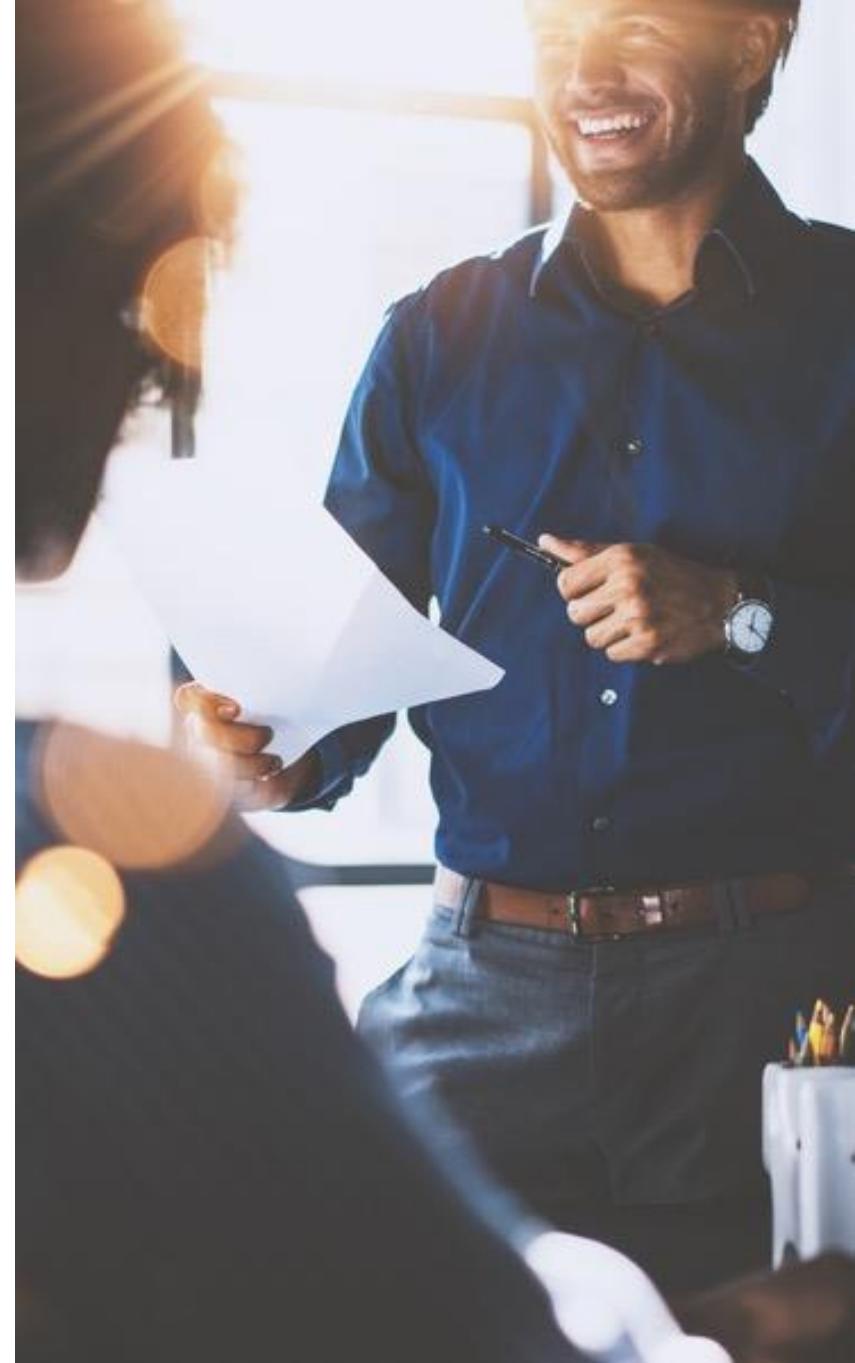
Hands-On Exercise 2.3

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 2.3: Transforming Data in Python in Preparation for Analysis



Objectives

- ▶ Explore datasets with Python and pandas
- ▶ Employ data visualization as an aid to understanding datasets
- ▶ Prepare data for analysis



Chapter 3

Working With

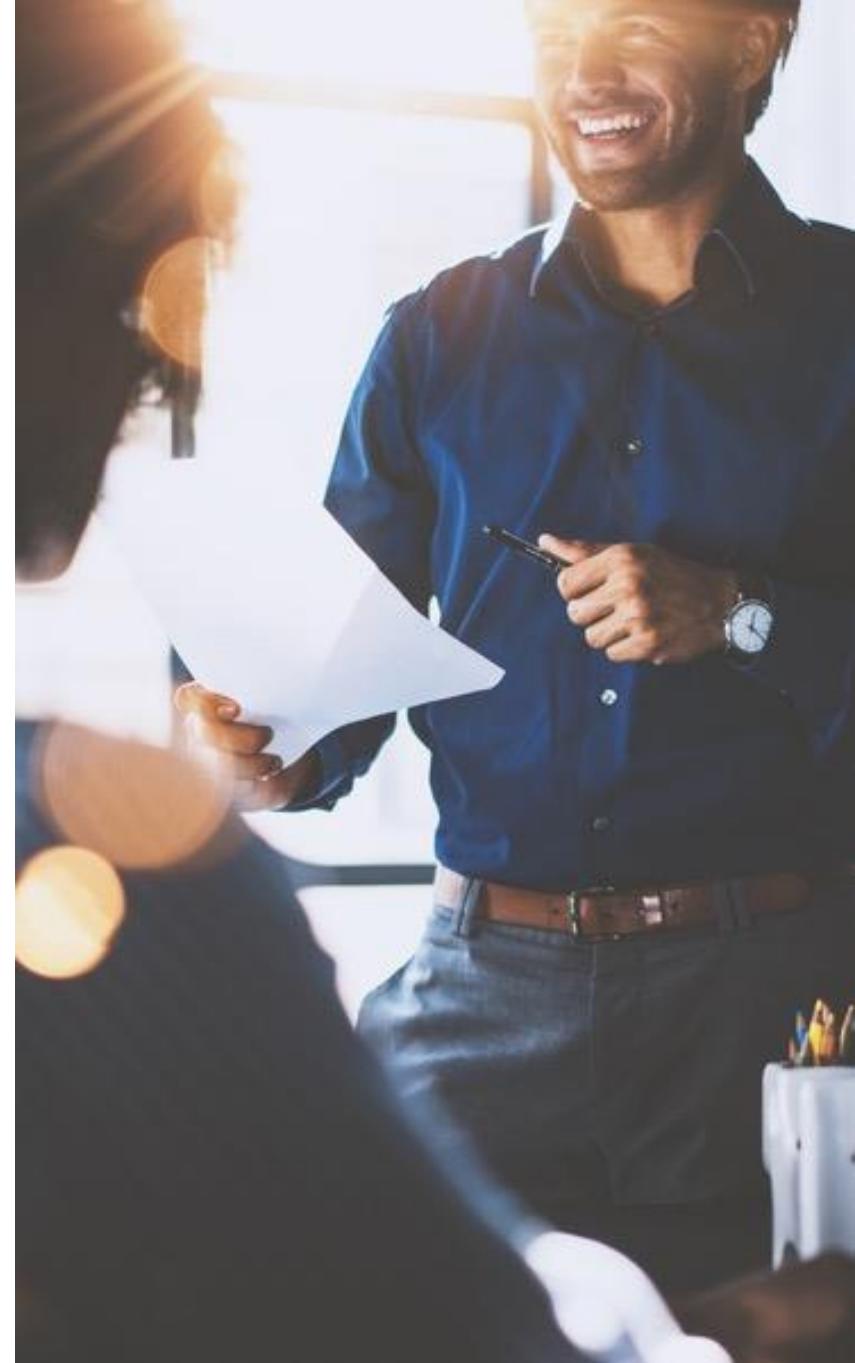
Unstructured Data



LEARNING TREE
INTERNATIONAL

Objectives

- ▶ Preprocess unstructured data in preparation for text mining
- ▶ Explore the concept of stemming
- ▶ Investigate the meaning of stop words
- ▶ Prepare a term-document matrix of unstructured documents in preparation for analysis



Contents

Introduction to Unstructured Data

- ▶ **Text Splitting and Preprocessing**
- ▶ **Term Frequency-Inverse Document Frequency**
- ▶ **Example**
- ▶ **Hands-On Exercise 3.1**

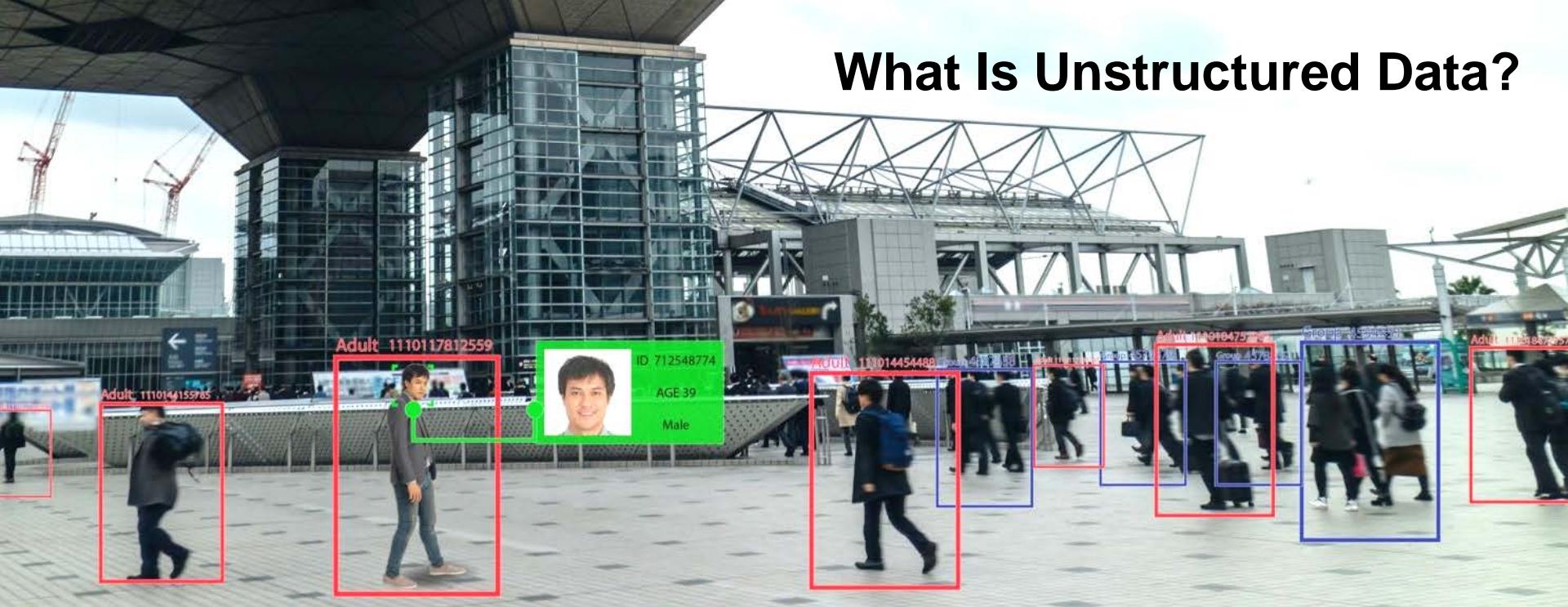


What Is Unstructured Data?

- There are many types of unstructured data that can be incorporated into Machine Learning (ML) models
 - Textual documents
 - E-mails
 - Tweets
 - PDF documents, etc.
 - Audio
 - Human voice
 - Music
 - Images and video



What Is Unstructured Data?



- ▶ Once preprocessed, unstructured data can be used in a similar way to structured data in building ML models
 - Discover patterns and trends
 - Predicting events

Examples of AI Built Around Unstructured Data

- ▶ Plagiarism detection
 - ▶ Customer relations management and sentiment analysis
 - ▶ Spam/phishing detection
 - ▶ Organizing web search results
 - ▶ Human detection in security video imagery
 - ▶ Home assistants such as Amazon Echo
 - ▶ Can you give examples of potentially valuable unstructured data from your workplace?
-
-
-



Contents

- ▶ Introduction to Unstructured Data

Text Splitting and Preprocessing

- ▶ Term Frequency-Inverse Document Frequency
- ▶ Example
- ▶ Hands-On Exercise 3.1



Representing Unstructured Data for ML Tasks

- The aim in preprocessing unstructured data is to represent the document/video/image, etc., in terms of its measurable features
 - This will typically be a vector or matrix structure

Twitter Screen

Tweets

Twitter API @twitterapi 5 Feb
Twitter Data Grants: A pilot program to give researchers access to public and historical data. Learn more blog.twitter.com/2014/introduct...
[Show Summary](#)

Twitter API @twitterapi 4 Feb
How eBay uses Cards and Twitter Card analytics: blog.twitter.com/2014/how-ebay-...
[Show Summary](#)

Twitter API @twitterapi 23 Jan
Introducing analytics for Twitter Cards: blog.twitter.com/2014/introduci...
[Show Summary](#)

Twitter API @twitterapi 13 Jan
Tomorrow (Jan 14, 2014) all requests to api.twitter.com will be restricted to SSL only. Read more: dev.twitter.com/discussions/24...
[Show Summary](#)

Twitter API @twitterapi 10 Jan
Important: On Jan 14, 2014 all requests to api.twitter.com will require SSL & disallow plaintext connections dev.twitter.com/discussions/24...
[Show Summary](#)

Tweet to @twitterapi

Documents (Tweets)

The word *tomorrow* appears 1 time in Tweet 4

	Tweet1	Tweet2	Tweet3	Tweet4
tomorrow				1
analytics		1	1	
learn	1			
historical	1			
...				

Term-Document Matrix

How to Prepare Text Data for ML

- ▶ Before the data can be vectorized, textual data typically needs to be preprocessed
- ▶ Preparing unstructured data for ML typically includes:
 1. Cleaning and feature extraction
 - Lower casing
 - Removing punctuations, HTML tags, special characters, alphanumerics
 - Removing most frequent or rare words
 - Correcting spelling mistakes
 - Removal of stop words
 - Stemming or lemmatization
 2. Tokenization and encoding the text data into integers values that can be used by machine learning algorithms

Splitting Text Into N-grams

- ▶ **An N-gram is a sequence of N words**
- ▶ **In these examples, the first two have more frequency in language than the last**
 - Thank you (is a 2-gram)
 - The Big Lebowski (is a 3-gram)
 - He reached over quickly (is a 4-gram)
- ▶ **Probabilities are used to calculate n-grams across a corpus of documents**
 - Example: No. of times “Thank You” occurs/No. of times “Thank” occurs
- ▶ **Assigning a probability to the occurrence of an N-gram is helpful for NLP**
 - In deciding which N-grams can be chunked together to form single entities for analysis
 - Making next word predictions
 - Making spelling error corrections
 - Example: “eat choclate” could be corrected to “eat chocolate” if you knew that the word “chocolate” had a high probability of occurrence after the word “eat”, and also the overlap of letters between “choclate” and “chocolate” is high

Stop Word Removal

- ▶ **Text documents contain a large number of words that do not help in a machine algorithm**
 - The algorithm could be programmed to ignore them, or we can just remove them to avoid storage and processing costs
- ▶ ***Stop words* are commonly used words in spoken and written text**
 - Articles—a, an, the
 - Prepositions—as, by, of
 - Pronouns—you, she, he, it
 - Conjunctions—and, or
- ▶ **They can be removed by storing a list of words that you consider to be stop words**
- ▶ **Alternatively, Python's NLTK has a list of stop words stored in 16 different languages that simplify the task**

Stemming

- The process of reducing derived words to their root forms even if the stem itself is not a valid word

Derived word	Root word
Playing	Play
Played	Play
Plays	Play

- Typically achieved by removing -ing, -s, -er, -ed, etc.
- English and non-English stemmers are available in the NLTK package
 - Porter Stemmer or Lancaster Stemmer can be used for the English language

Stemmers

- ▶ **The Porter Stemmer algorithm follows a set of five generic rules to generate stems**
 - Often generate stems that are not actual English words
 - Simple and fast
 - For example, the word, “destabilized”, is stemmed to “dest”
- ▶ **Lancaster Stemmer has many rules indexed by the last letter of a suffix**
 - A rule specifies either a deletion or replacement of an ending
 - No stemming occurs if a word starts with a vowel and has only two letters left, or if a word starts with a consonant and has only three characters left
 - Simple but not as fast
 - Over-stemming may cause stems to have no meaning
 - For example, the word, “destabilized”, is stemmed to “destabl”
- ▶ **Snowball Stemmers can be used to create non-English stemmers, or one's own language stemmer**
 - Danish, Dutch, English, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish

Lemmatization

- ▶ **Lemmatization reduces derived words ensuring that the root word belongs to the language**
 - Use where it is necessary to retrieve valid words, and speed is not as important
- ▶ **A root form is not produced where a word is given without context**
 - This is given as a parts-of-speech (POS) parameter

Contents

- ▶ Introduction to Unstructured Data
- ▶ Text Splitting and Preprocessing

Term Frequency-Inverse Document Frequency

- ▶ Example
- ▶ Hands-On Exercise 3.1



Term Frequency-Inverse Document Frequency

- The assumption behind Term Frequency-Inverse Document Frequency (TF-IDF) is that terms that appear more frequently in a document should be given a higher weight, unless they also appear in a lot of documents
- To prepare the corpus for analysis, a term-document matrix (TDM) is created
 - Shows the terms and frequency of each word in the corpus



Documents (Tweets)

	Tweet1	Tweet2	Tweet3	Tweet4
tomorrow				1
analytics		1	1	
learn	1			
historical	1			
...				

The word *tomorrow* appears 1 time in Tweet 4

TDM

Term Frequency-Inverse Document Frequency

- ▶ **Some words are more important than others in determining meaning**
 - Each text word represents a dimension
- ▶ **To reduce the number of dimensions, TF-IDF is often used to prepare text for ML algorithms**
 - Text is processed into weighted vectors
- ▶ **Term Frequency (TF)**
 - Number of times a term occurs in a document
 - Used to find out the relevance of a word in a document
 - The more frequent a word is, the more relevance the word holds in the context
- ▶ **Document Frequency (DF)**
 - Number of documents that contain each word
 - Inverse Document Frequency (IDF) = $\log(\text{Total No. of docs}/\text{DF})$
 - $\text{TF-IDF} = \text{TF} * \text{IDF}$

Contents

- ▶ Introduction to Unstructured Data
- ▶ Text Splitting and Preprocessing
- ▶ Term Frequency-Inverse Document Frequency

Example

- ▶ Hands-On Exercise 3.1



Reading in the Data File

- ▶ **To open a text-base file use the open() function**
 - Requires two arguments
 - The file path or file name and the mode it should be opened in
 - “r” (default): read only, “w”: if the file exists then content is deleted and opened it to write, and “a”: append
- ▶ **To read the contents of the file, use the read() function**

```
shakes = open('shakes.txt')
shakes = shakes.read(500)
```

```
shakes
'The Project Gutenberg EBook of The Complete Works of
William Shakespeare, by\nWilliam Shakespeare\n\nThis eBook is
for the use of anyone anywhere at no cost and with\nalmost no
restrictions whatsoever. You may copy it, give it away or\nre-
use it under the terms of the Project Gutenberg License
included\nwith this eBook or online at www.gutenberg.org\n\n**
This is a COPYRIGHTED Project Gutenberg eBook, Details Below
**\n**      Please follow the copyright guidelines in this file.
**\n\nTitle: The Comple'
```

Text Preprocessing

- ▶ **Typically, a number of transformations are required**
 - Text case conversion
 - Remove numbers
 - Remove punctuation
 - Stop word removal
 - Stemming

Converting Text to Lowercase

- ▶ Use the `lower()` method to return a copy of the string in which all case-based characters have been lowercased

```
shakes.lower()
```

```
the project gutenberg ebook of the complete works  
of william shakespeare, by\nwilliam shakespeare\n\nthis ebook is  
for the use of anyone anywhere at no cost and with\nalmost no  
restrictions whatsoever. you may copy it, give it away or\nre-  
use it under the terms of the project gutenberg license  
included\nwith this ebook or online at www.gutenberg.org\n\n**  
this is a copyrighted project gutenberg ebook, details below  
**\n**      please follow the copyright guidelines in this file.  
**\n\n[title: the comple'2]:
```

Removing Numbers and Punctuation Marks

- To remove numbers and punctuation marks, a built-in package known as `re` may be used

```
import re
```

shakes

```
'te Works of William Shakespeare\n\nAuthor: William  
Shakespeare\n\nPosting Date: September 1, 2011 [EBook  
#100]\nRelease Date: January, 1994\n\nLanguage:  
English\n\nCharacter set encoding: ASCII\n\n*** START OF THIS  
PROJECT GUTENBERG EBOOK COMPLETE WORKS--WILLIAM SHAKESPEARE  
***\n\nProduced by World Library, Inc., from their Library  
of the Future\n\nThis is the 100th Etext file presented by  
Project Gutenberg, and\nis presented in cooperation with World  
Library, Inc., from their\nLibrary of the Future and Sh'
```

```
re.sub( "[^a-zA-Z]", " ", shakes )
```

The code above uses the `re.sub()` function to replace all non-alphabetic characters in the `shakes` string with a space. A blue arrow points from the regular expression pattern `[^a-zA-Z]` in the code to the corresponding non-alphabetic characters in the original text.

```
'te Works of William Shakespeare Author William  
Shakespeare Posting Date September EBook  
Release Date January Language English Character set  
encoding ASCII START OF THIS PROJECT GUTENBERG EBOOK  
COMPLETE WORKS WILLIAM SHAKESPEARE Produced by World  
Library Inc from their Library of the Future This is the  
th Etext file presented by Project Gutenberg and is presented  
in cooperation with World Library Inc from their Library of  
the Future and Sh'
```

Replace anything that is *not* a lowercase letter (a-z) or an uppercase letter (A-Z), and replace it with a space

[] indicates group membership
^ indicates “not”

Tokenization

- The corpus* is then normally vectorized through a process called ***tokenization***
 - The text is split into tokens
 - A token may refer to a word, punctuation mark, number, or alphanumeric
 - Tokenization may result in loss of meaning
 - In English, white space or punctuation marks are often used to tokenize
 - More difficult in languages such as Chinese or Arabic, as there are no explicit boundaries

```
nltk.tokenize.word_tokenize(shakes)
['the',
 'project',
 'gutenberg',
 'ebook',
 'of',
 'the',
 'complete',
 'works',
 'of',
 'william',
 'shakespeare',
 'by',
 'william',
 'shakespeare',
```

*A corpus is a body of text containing a large number of sentences.

Removing Stop Words With NLTK

- The `nltk` library greatly simplifies the task of stop word identification and removal
- Load the `nltk` library for text processing

```
import nltk
```

- To download text datasets including stop words

```
nltk.download('stopwords')
```

Import and View Stop Words

► Import and print a list of English language stop words

```
from nltk.corpus import stopwords  
  
stopwords.words("english")  
  
['i',  
 'me',  
 'my',  
 'myself',  
 'we',  
 'our',  
 'ours',  
 'ourselves',  
 'you',  
 'your',  
 'yours',  
 'yourself',  
 'yourselves',  
 'he',  
 'him',  
 'his',  
 'himself',  
 'she',
```

Remove Stop Words

- To remove stop words from text

```
sentence = "the quick brown fox jumped over the lazy dogs"
```

```
[i for i in sentence.split() if i not in stopwords.words('english')]  
['quick', 'brown', 'fox', 'jumped', 'lazy', 'dogs']
```

Text Stemming

► Use NLTK to do a Porter Stemmer

```
from nltk.stem import PorterStemmer  
  
stemmer=PorterStemmer()  
  
stemmed = []  
  
sentence = "running jumping played presumably multiply"  
  
for item in sentence.split():  
    stemmed.append(stemmer.stem(item))  
  
stemmed  
['run', 'jump', 'play', 'presum', 'multipli']
```

Building a TDM

- ▶ To create the TDM, a number of functions from the Scikit-learn package are used
- ▶ The `TfidfVectorizer()` function is made up of a:
 - `CountVectorizer()` function: Converts documents to a sparse matrix of token counts, and a
 - `TfidfTransformer()` function: Transforms a count matrix to a normalized *tf-idf* representation
- ▶ The `fit_transform()` function learns the vocabulary and inverse document frequencies, and returns a TDM

```
from sklearn.feature_extraction.text import TfidfVectorizer  
  
tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words='english')  
  
tfs = tfidf.fit_transform(token_dict.values())
```

Exploring the TDM

- ▶ The `get_feature_names()` method can be used to see the words/features
- ▶ The `idf_` attribute can be used to view the inverse document frequencies
- ▶ The creation of the TDM is a pre-processing step when using unstructured textual data in Machine Learning models
 - Example: A Classification model for detecting Spam email

Contents

- ▶ Introduction to Unstructured Data
 - ▶ Text Splitting and Preprocessing
 - ▶ Term Frequency-Inverse Document Frequency
 - ▶ Example

Hands-On Exercise 3.1



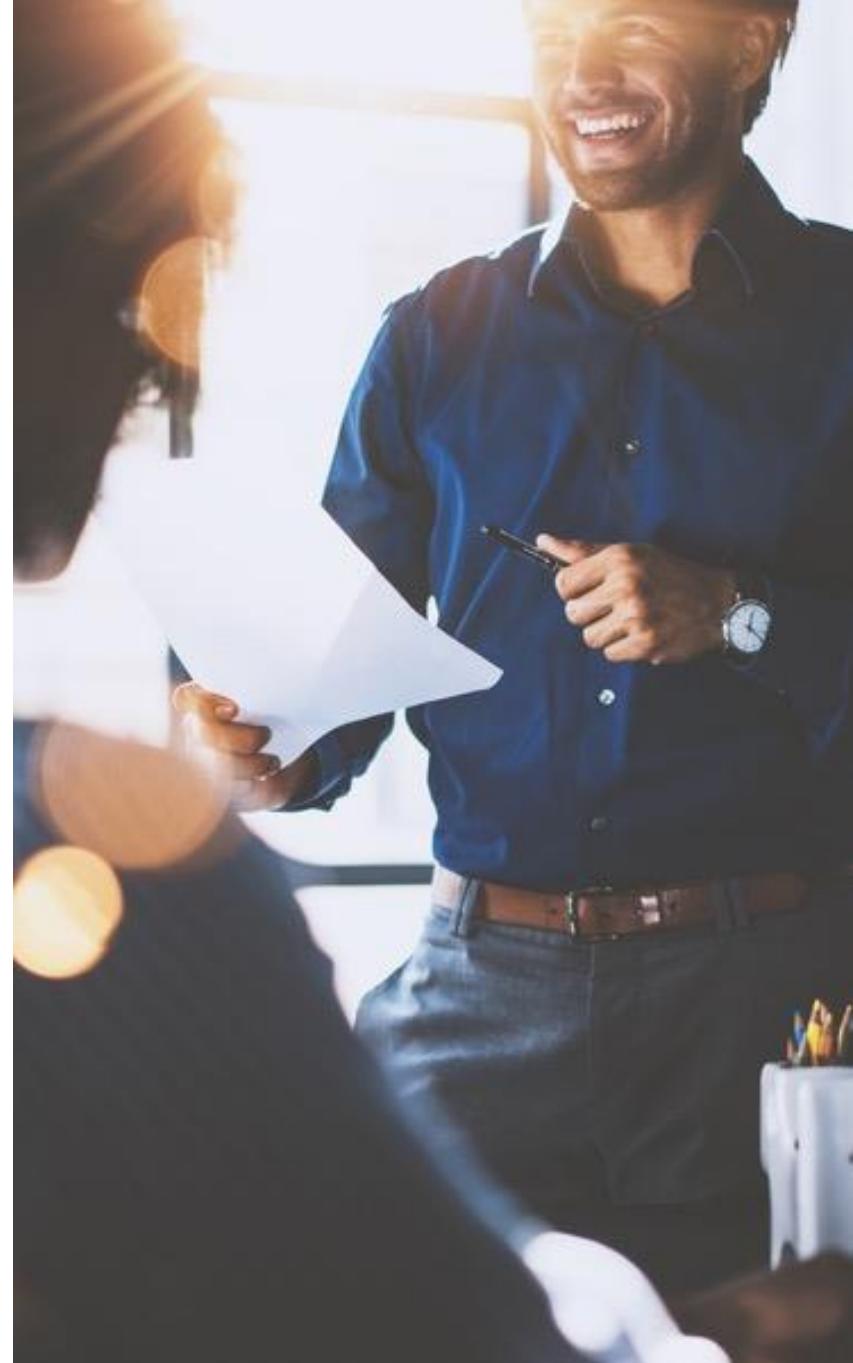


Hands-On Exercise 3.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 3.1: Text Mining

Objectives

- ▶ Preprocess unstructured data in preparation for text mining
- ▶ Explore the concept of stemming
- ▶ Investigate the meaning of stop words
- ▶ Prepare a term-document matrix of unstructured documents in preparation for analysis



Chapter 4

Predicting Outcomes

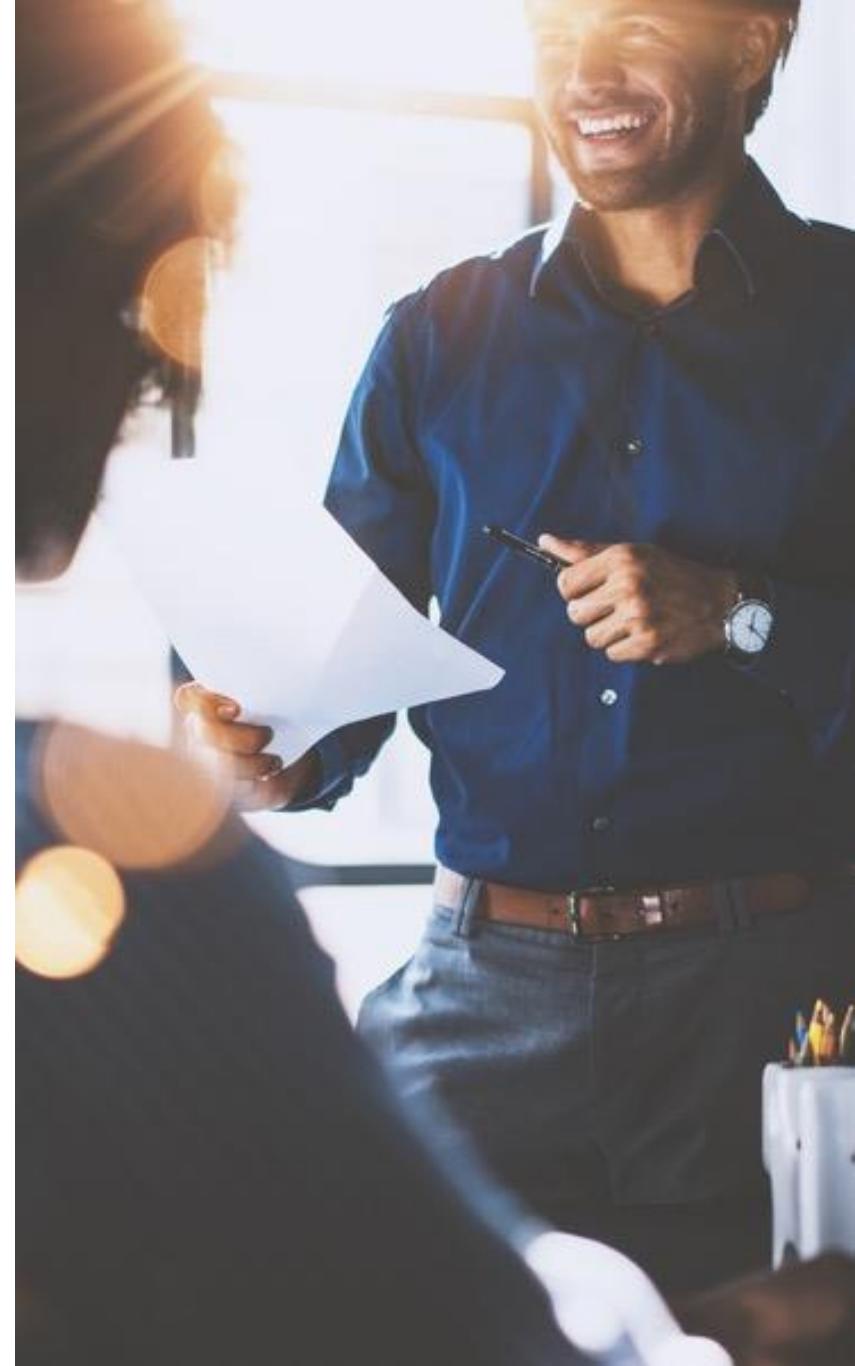
With Regression Analysis



LEARNING TREE
INTERNATIONAL

Objectives

- ▶ Express business problems as linear regression tasks
- ▶ Produce a linear regression model in Python
- ▶ Interpret, evaluate, and improve the linear regression model



Contents

Introduction to Linear Regression

- ▶ A Regression Example
- ▶ Linear Regression Assumptions
- ▶ Fitting the Model
- ▶ Interpreting and Evaluating the Model
- ▶ Hands-On Exercise 4.1



Introduction to Regression

- ▶ **Linear regression is the most common statistical method**
 - Used across fields as diverse as economics, sociology, psychology, physics, and ecology
- ▶ **Used to model real-world relationships between variables and measure how one (or more) of these variables might impact another; for example:**
 - Car insurance premiums might be based on attributes of the car, driver information or demographics
 - Credit card default might be related to salary, unhealthy spending habits, employment security
 - One's income might be related to years of education and job experience



Other Sample Use Cases

- ▶ **Some sample use cases are**
 - Quantifying the causal relationship between an event and the response
 - Such as those in engineering safety tests, clinical drug trials, or marketing research
 - Sales forecasting
 - Predicting optimum product price
 - Identifying patterns that can be used to forecast future behavior given known criteria
 - Such as for predicting insurance claims, natural disaster damage, election results, and crime rates



When Can Linear Regression Be Used?

- ▶ **Linear regression can be used when the dependent/target variable is of a continuous nature**
- ▶ **The input (independent) and the target (dependent) variables are numeric**
- ▶ **It also makes a number of assumptions of the data**

Simple Linear Regression

- A simple regression is used to show the extent to which changes in the variable to be predicted (shown on the y-axis) can be attributed to changes in an explanatory variable (shown on the x-axis)

- It is described by the following equation:

$$y = \alpha + \beta x$$

y: the target variable

x: the predictor variable

β : the slope (also called the coefficient) indicates how much the target value increases for each unit increase in *the predictor variable*

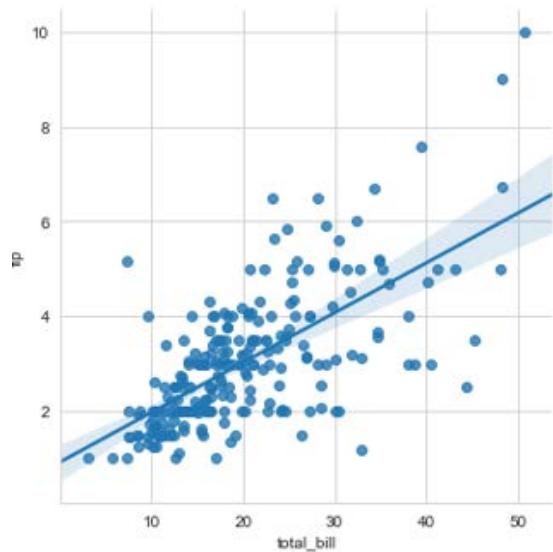
α : the intercept—the value of *the target* when *the predictor is valued at 0* (where the line crosses the vertical axis)

- Trend lines might then be used to predict a target variable's value for future events

Visualizing Regression to Identify Trends

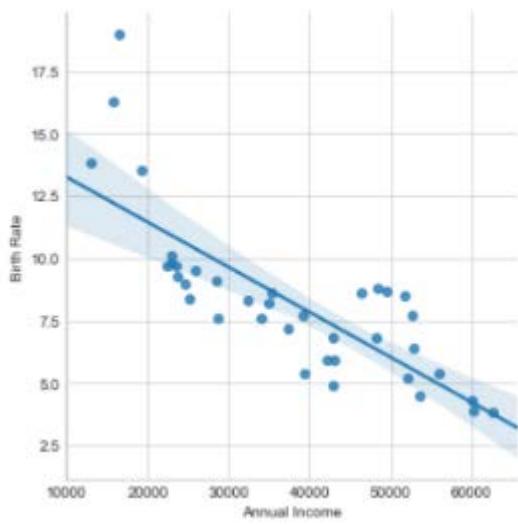
► Example Positive Trend

- If the amount of a tip increases along with the total amount of the bill, then plotting the tip amount data on the y-axis, and total bill amount data on the x-axis, produces a line depicting an upward trend
- The trend line can then be used to estimate future tip amounts from different group numbers in a restaurant in future months



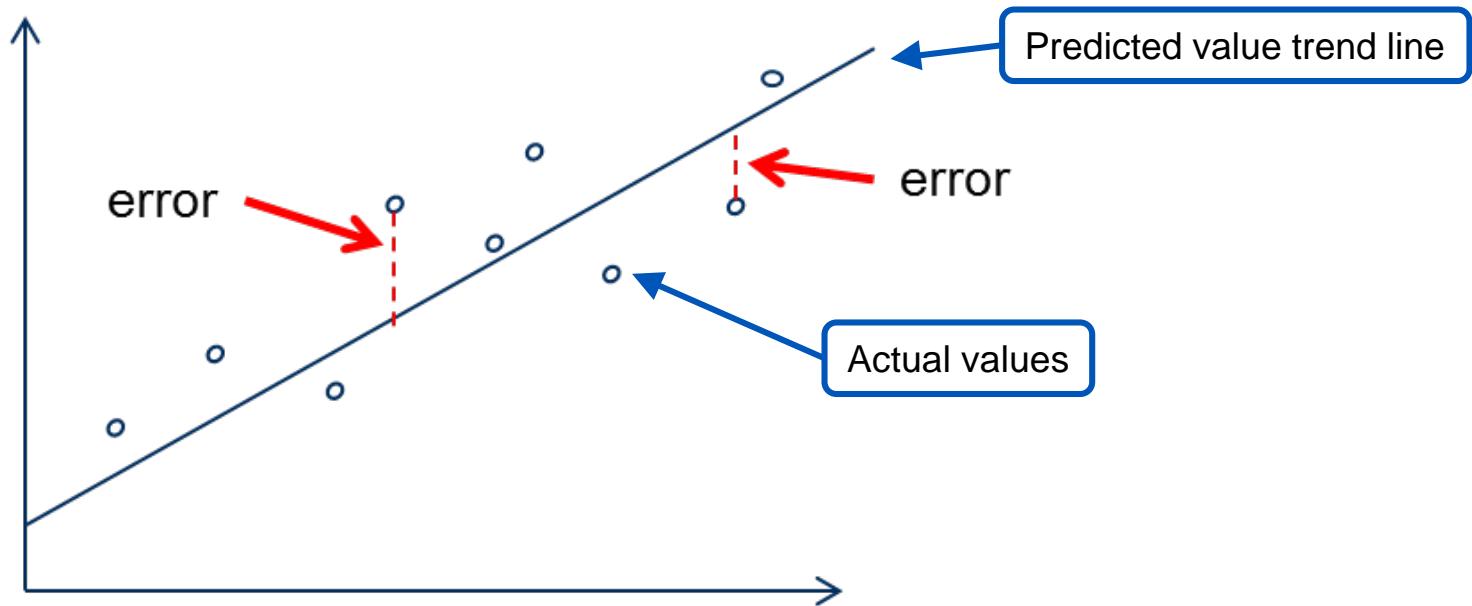
► Example Negative Trend

- The decision to have a child can be a costly decision
- There are reasons to believe that economic considerations play a role in deciding to have children
 - The figure here shows the relationship between birth rate and Annual Income



Training a Linear Regression Model

- To train a regression model means to find the parameters that best fit the data
- Here a line is fitted to the data such that the difference between the predicted values and the observed values are minimized
 - This is called the *best fit line* or the *regression line*
 - Errors are also called *residuals*



Multiple Linear Regression

- ▶ **Most real-world analyses have more than one independent variable**
- ▶ ***Multiple regression* is an extension of simple linear regression**
 - The key difference is that there will be additional beta coefficients for the additional predictor variables
 - The goal in both cases is to find beta coefficients that minimize the errors of the linear equation
- ▶ **Weaknesses**
 - Only works with numeric features
 - Categorical data needs to be pre-processed
 - Does not cope well with missing data
 - Makes assumptions about the data
 - More difficult to draw

Equation for Multiple Linear Regression

- ▶ Similar to a simple linear equation, the dependent variable y is quantified as the sum of an intercept term plus the product of the β coefficients multiplied by the x value for each of the i features

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_i x_i + \varepsilon$$

- ▶ The error or residual term (ε) indicates that the predictions are not perfect
- ▶ A coefficient is estimated for each predictor variable
 - Allows each predictor to have a separate estimated effect on the value of y
 - y changes by the amount β_i for each unit increase in x_i
 - The intercept (α) indicates the expected value of y when the independent variables are all zero
- ▶ The target and predictor variables are now vectors with a row for every example
 - The regression coefficients β and errors ε are also vectors

Python's Built-In Function for Linear Regression

- To carry out a linear regression analysis in Python, the `ols()` function from `statsmodel` can be used:

```
ols(formula = 'target ~ predictors', data).fit()
```

- target: the variable to be predicted
- predictors: *the variables being used to predict the target variable*
- data: typically a DataFrame holding the target and predictor variables
- The tilde character ~ separates the target and predictor variables
 - target variable is on the left of the tilde
 - predictor variables are on the right separated by the + sign
- The regression model's intercept term is assumed by default

- For example:

```
est = ols(formula = 'prestige ~ education + income + women +  
type', data = Prestige).fit()
```

Contents

- ▶ Introduction to Linear Regression

A Regression Example

- ▶ Linear Regression Assumptions
- ▶ Fitting the Model
- ▶ Interpreting and Evaluating the Model
- ▶ Hands-On Exercise 4.1



Example—Understanding a Variable: Prestige

- ▶ Regression analysis can be used to make predictions, but can also be used to understand latent variables (variables that cannot be directly observed)
- ▶ The goal of this analysis is to use census data to study a target variable prestige from a Pineo-Porter prestige score for occupation status from a social survey conducted in Canada in the mid-1960s
 - Refers to the consensual rate of “general standing” people give to an occupation (e.g., a doctor might be generally held in high standing, and therefore have a higher prestige score)
 - Predictor variables such as education and income will be used to attempt to estimate the so-called “prestige” level
 - These estimates could be used in policy-making decisions for education
- ▶ In regression analysis explanatory variables are specified by the user rather than detected automatically (e.g., we might specify that education and income level are indicators explaining “prestige”)

The Dataset

- ▶ **The observations in the Prestige dataset are of Canadian occupations**
 - Comes from the “car” library in R
- ▶ **This dataset has 102 observations and six attributes**
 - education: Average education of occupational incumbents
 - income: Average income of incumbents
 - women: Percentage of incumbents who are women
 - prestige: Pineo-Porter prestige score for occupation
 - From a social survey conducted in the mid-1960s
 - census: Canadian census occupational code
 - type: Category of occupation
 - A factor with three levels:
 - bc: blue collar
 - prof: professional, managerial, and technical
 - wc: white collar

Exploring the Data

- ▶ Import the library and read the data into Python:

```
import statsmodels.api as sm  
  
Prestige = sm.datasets.get_rdataset("Prestige", "carData").data
```

- ▶ Explore the data:

```
Prestige.shape
```

```
(102, 6)
```

```
Prestige.columns
```

```
Index(['education', 'income', 'women', 'prestige', 'census', 'type'], dtype='object')
```

```
Prestige.head()
```

	education	income	women	prestige	census	type
gov.administrators	13.11	12351	11.16	68.8	1113	prof
general.managers	12.26	25879	4.02	69.1	1130	prof
accountants	12.77	9271	15.70	63.4	1171	prof
purchasing.officers	11.42	8865	9.11	56.8	1175	prof
chemists	14.62	8403	11.68	73.5	2111	prof

Contents

- ▶ Introduction to Linear Regression
- ▶ A Regression Example

Linear Regression Assumptions

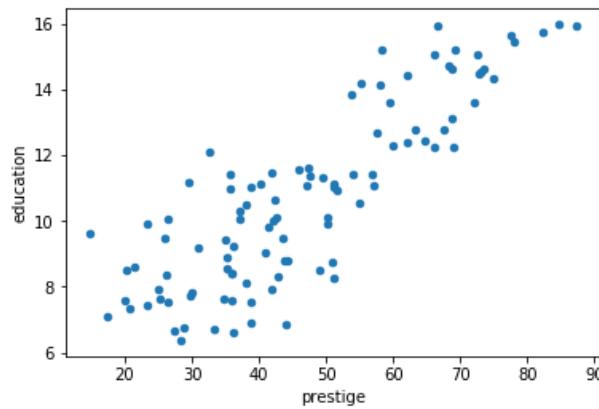
- ▶ Fitting the Model
- ▶ Interpreting and Evaluating the Model
- ▶ Hands-On Exercise 4.1



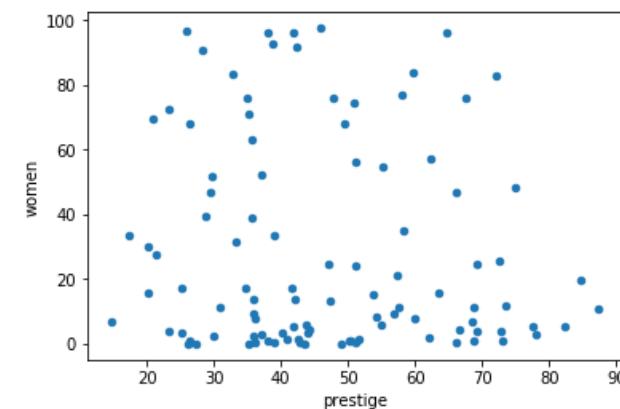
First Assumption of Linear Regression: Linear Relationship

- ▶ Linear Regression calculations make a number of assumptions
- ▶ There should be a linear relationship between the independent and dependent variables

```
Prestige.plot.scatter(x='prestige',y='education')  
<matplotlib.axes._subplots.AxesSubplot at 0x1444e438>
```



```
Prestige.plot.scatter(x='prestige',y='women')  
<matplotlib.axes._subplots.AxesSubplot at 0x14405940>
```

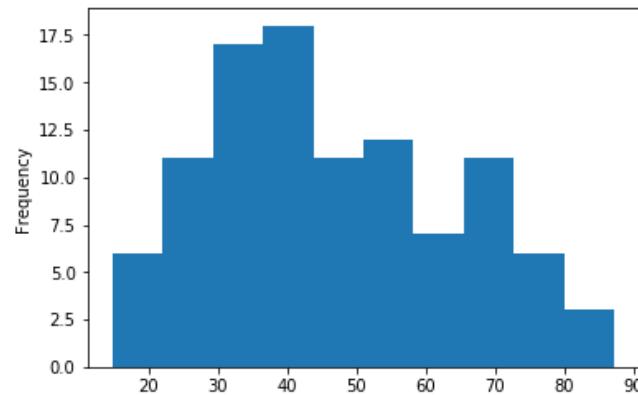


- This can be checked using scatterplots
 - This should also be checked for outliers since linear regression is sensitive to outlier effects
- Here the variable education shows a strong linear relationship while the variable women does not

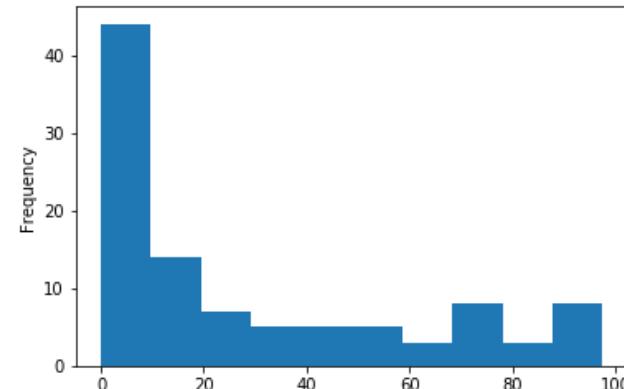
Second Assumption of Linear Regression: Multivariate Normality

- All variables should be normally distributed
 - This can be checked using a histogram

```
Prestige['prestige'].plot(kind='hist')  
<matplotlib.axes._subplots.AxesSubplot at 0x11af3438>
```



```
Prestige['women'].plot(kind='hist')  
<matplotlib.axes._subplots.AxesSubplot at 0x11b9e940>
```

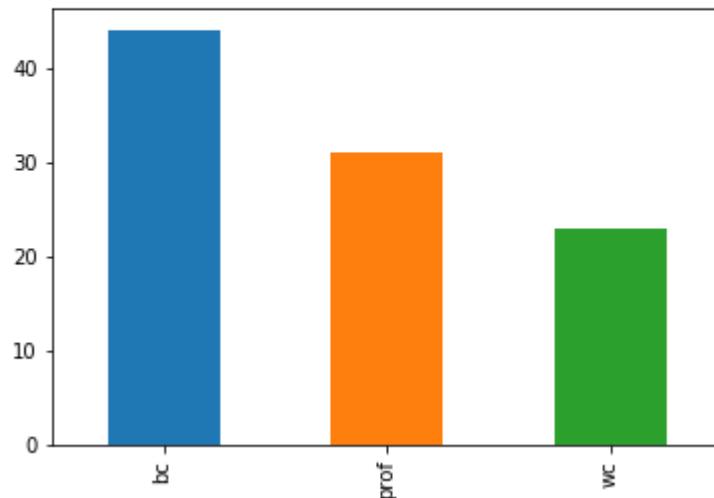


- When the data is not normally distributed a non-linear transformation (e.g., log-transformation) may fix the issue

Second Assumption of Linear Regression: Multivariate Normality

- ▶ Though regression models require that every feature is numeric, we can encode non-numeric variables in order to use them in the model
 - The type variable in this dataset is non-numeric
 - It is divided into “blue collar,” “professional,” and “white collar”
- ▶ Use a plot to show how the data is distributed across the three non-numeric variables

```
Prestige['type'].value_counts().plot(kind='bar')  
<matplotlib.axes._subplots.AxesSubplot at 0x11b752e8>
```



Third Assumption of Linear Regression: No Multicollinearity

- ▶ **All input variables should be independent of each other**
 - Can be difficult to completely achieve
 - At a minimum eliminate highly correlated variables
 - Where correlated variables have opposing correlation (i.e., one positive and one negative), they will end up canceling each other out, and the effect of neither will be seen in the model
- ▶ **The correlation between two variables is a number that indicates how closely their relationship follows a straight line**
 - The correlation ranges between -1 and +1
 - Extreme values indicate a perfectly linear relationship, whereas a correlation close to zero indicates the absence of a linear relationship
- ▶ **Correlation does not imply causation**
 - Only describes the association between a pair of variables
 - There could be other explanations

Third Assumption of Linear Regression: No Multicollinearity

► A correlation matrix provides a quick overview of these relationships

- Correlates each pairwise relationship
- The diagonal is always 1 since there is always a perfect correlation between a variable and itself

► Thresholds:

- Between 0.1 and 0.3: Weakly correlated
- Between 0.3 to 0.5: Moderately correlated
- Above 0.5: Strong correlation
 - Also applies to negative values

► Correlation must be interpreted in context

- Thresholds may be too lax for some purposes
- In data involving human beings, a correlation of 0.5 may be considered extremely high
- In data generated by mechanical processes, a correlation of 0.5 may be weak

Prestige.corr()					
	education	income	women	prestige	census
education	1.000000	0.577580	0.061853	0.850177	-0.823088
income	0.577580	1.000000	-0.441059	0.714906	-0.361002
women	0.061853	-0.441059	1.000000	-0.118334	-0.227003
prestige	0.850177	0.714906	-0.118334	1.000000	-0.634510
census	-0.823088	-0.361002	-0.227003	-0.634510	1.000000

Third Assumption of Linear Regression: No Multicollinearity

- It can also be helpful to visualize the relationships among features using a scatterplot matrix
 - Only two features are examined at a time, but it provides a general sense of how the data may be interrelated



Third Assumption of Linear Regression: No Multicollinearity

- ▶ In both the correlation matrix and pairplot, the intersection of each row and column holds the scatterplot of the variables indicated by the row and column pair
- ▶ The values above and below the diagonal in the correlation matrix are identical because correlations are symmetrical
 - In other words, $\text{corr}(x, y)$ is equal to $\text{corr}(y, x)$
 - In the pairplot the diagrams above and below the diagonal are transpositions since the x-axis and y-axis have been swapped
- ▶ Most of the correlations in the matrix are not considered strong, but there are some notable associations
 - *Education* and *income* appear to have a strong correlation, meaning that as *education* increases, so does *income*
 - There is also a moderate (negative) correlation between *income* and *women*
 - These relationships need to be teased out when the final regression model is built

Other Assumptions of Linear Regression

- ▶ **Linear regression models also assume little or no autocorrelation in the data**
 - Autocorrelation occurs when the residuals are not independent from each other
 - Often occurs in stock price analysis where one price is not independent from the previous price
- ▶ **Linear regression models assume *homoscedasticity*, which is where the error term is the same across all values of the independent variables**
 - *Heteroscedasticity* (the violation of homoscedasticity) is present when the size of the error term differs across values of an independent variable
 - The impact of violating the assumption of homoscedasticity is a matter of degree, increasing as heteroscedasticity increases

Contents

- ▶ Introduction to Linear Regression
- ▶ A Regression Example
- ▶ Linear Regression Assumptions

Fitting the Model

- ▶ Interpreting and Evaluating the Model
- ▶ Hands-On Exercise 4.1



Fitting a Linear Model to the Data

- The `ols()` function comes from the `statsmodel` package:

```
from statsmodels.formula.api import ols  
  
est <- ols(formula = dv ~ iv, data = Prestige).fit()
```

`dv`: target variable

`iv`: formula for predictor variables

`data`: `DataFrame` where the variables can be found

Making Predictions From the Model

► **To make predictions from the derived model:**

```
est.predict(test)
```

- est: The model trained by `ols().fit`
- test: A DataFrame containing the same attributes as the training dataset

► **A vector of predicted values will be returned**

► **Example:**

```
est = ols(formula = 'prestige ~ education + income + women +  
type', data = Prestige).fit()
```

```
pred = est.predict(Prestige)
```

- Creates a model relating four independent variables to the target variable
- It fits the linear regression model (est) to new data

Contents

- ▶ Introduction to Linear Regression
- ▶ A Regression Example
- ▶ Linear Regression Assumptions
- ▶ Fitting the Model

Interpreting and Evaluating the Model

- ▶ Hands-On Exercise 4.1



Interpreting the Model

- After building a model, view its results using the `summary()` function:

```
In [714]: est.summary()
Out[714]:
<class 'statsmodels.iolib.summary.Summary'>
"""
=====
OLS Regression Results
=====
Dep. Variable: prestige   R-squared:      0.835
Model:          OLS         Adj. R-squared:  0.826
Method:        Least Squares   F-statistic:    93.07
Date: Fri, 27 Apr 2018   Prob (F-statistic): 1.93e-34
Time: 22:41:51           Log-Likelihood: -328.48
No. Observations: 98          AIC:             669.0
Df Residuals:     92          BIC:             684.5
Df Model:        5
Covariance Type: nonrobust
=====
            coef  std err      t      P>|t|      [0.025      0.975]
-----
Intercept    -0.8139    5.331    -0.153    0.879    -11.402     9.774
type[T.prof]  5.9052    3.938     1.500    0.137    -1.915    13.726
type[T.wc]    -2.9171    2.665    -1.094    0.277    -8.211     2.377
education     3.6624    0.646     5.671    0.000     2.380     4.945
income        0.0010    0.000     3.976    0.000     0.001     0.002
women         0.0064    0.030     0.212    0.832    -0.054     0.067
=====
Omnibus:            0.681 Durbin-Watson:       1.779
Prob(Omnibus):      0.711 Jarque-Bera (JB):  0.798
Skew:              -0.112 Prob(JB):          0.671
Kurtosis:           2.619 Cond. No.        7.48e+04
=====
```

Interpreting the Model

- ▶ We only specified four features in our model formula, but there are five reported coefficients in addition to the intercept
- ▶ The `ols()` function automatically applied dummy coding to each of the type variables we included in the model
 - Dummy coding allows a nominal feature to be treated as numeric by creating a binary variable for each category of the feature, which is set to 1 if the observation falls into that category or 0 otherwise
 - For example, the type variable has categories, bc, wc, and prof
 - This will be split into binary values, which are named `type[T.prof]` and `type[T.wc]`
 - For observations where `type = wc`, then `type[T.wc] = 1` and `type[T.prof] = 0`; if `type = prof`, then `type[T.wc] = 0` and `type[T.prof] = 1`

Interpreting the Model

- ▶ When adding a dummy coded variable to a regression model, one category is left
 - Acts as the reference category
 - Estimates are then interpreted relative to the reference
 - In our model ols automatically held out the type[T.bc] variable, making blue collar workers the reference group
- ▶ By default ols uses the first level of the factor variable as the reference

Interpreting the Model

- ▶ **The intercept tells us the prestige value when the independent variables are equal to zero**
 - Intercept is often difficult to interpret because it is impossible to have values of zero for all features (e.g., since no person exists with education zero and income zero, the intercept has no inherent meaning)
 - For this reason, in practice, the intercept is often ignored
- ▶ **The estimated beta coefficients indicate the increase in prestige for an increase of one in each of the features when the other features are held constant**
 - For example, for each year that education increases, we would expect 3.66 extra points on the prestige score, assuming everything else is equal

Evaluating the Model

- ▶ **The p-value ($P>|t|$) tells you the significance of the regression coefficient for the variable**
 - The p-value for each term tests the null hypothesis that the coefficient is equal to zero (no effect)
 - A low p-value (< 0.05) indicates that you can reject the null hypothesis—there is a significant relationship between variables in the model
 - The significance level measures how likely the true coefficient is zero given the value of the estimate

Evaluating the Model

- ▶ **R-squared is the value of the coefficient of determination**
 - Provides a measure of how well our model explains the values of the target variable
 - The closer the value is to 1.0, the better the model explains the target
 - In our model the R-squared value of 0.835 means 83 percent of the variation in the target variable is explained by the model
 - This is quite good
 - Models with more features explain more variation
 - Adjusted R-squared value penalizes models with a large number of independent variables
 - Useful when carrying out a comparison between models with different numbers of predictor variables

Contents

- ▶ Introduction to Linear Regression
- ▶ A Regression Example
- ▶ Linear Regression Assumptions
- ▶ Fitting the Model
- ▶ Interpreting and Evaluating the Model

Hands-On Exercise 4.1



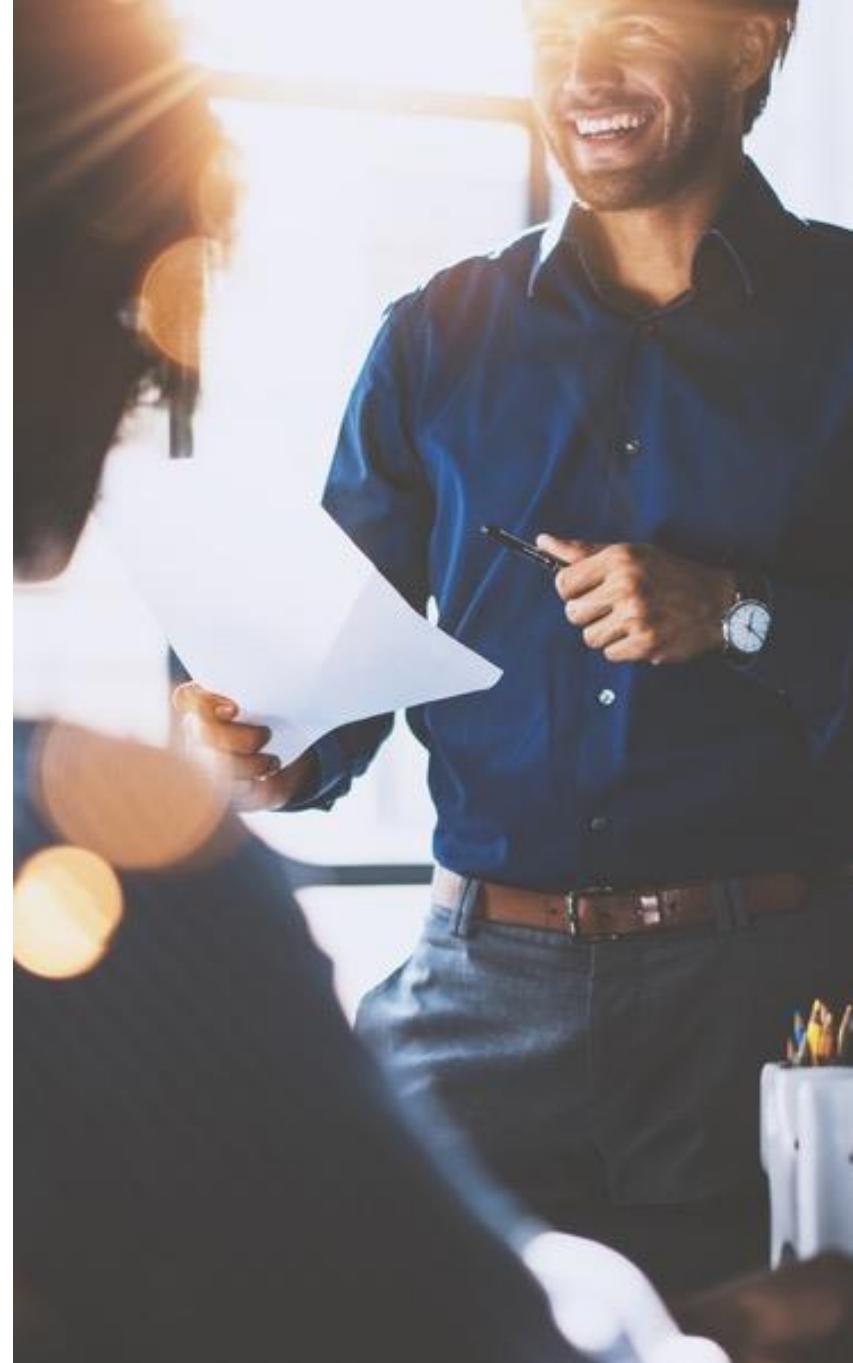


Hands-On Exercise 4.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 4.1: Performing Regression Analysis of Structured Data With Python

Objectives

- ▶ Express business problems as linear regression tasks
- ▶ Produce a linear regression model in Python
- ▶ Interpret, evaluate, and improve the linear regression model

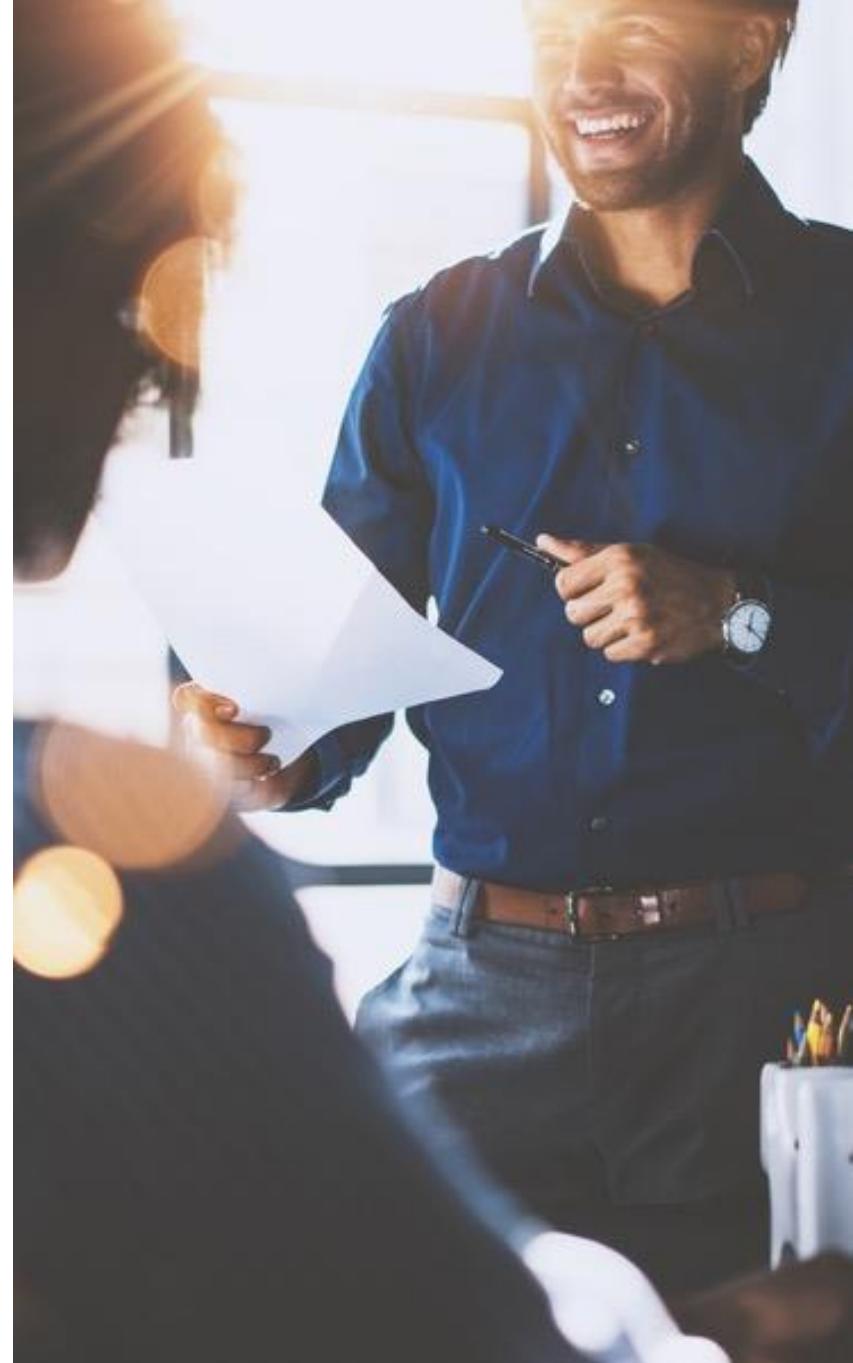


Chapter 5

Categorizing Data With Classification Techniques

Objectives

- ▶ Learn how classifiers are used to predict data categorization
- ▶ Explore how decision trees are built
- ▶ Build and apply a decision tree classifier



Contents

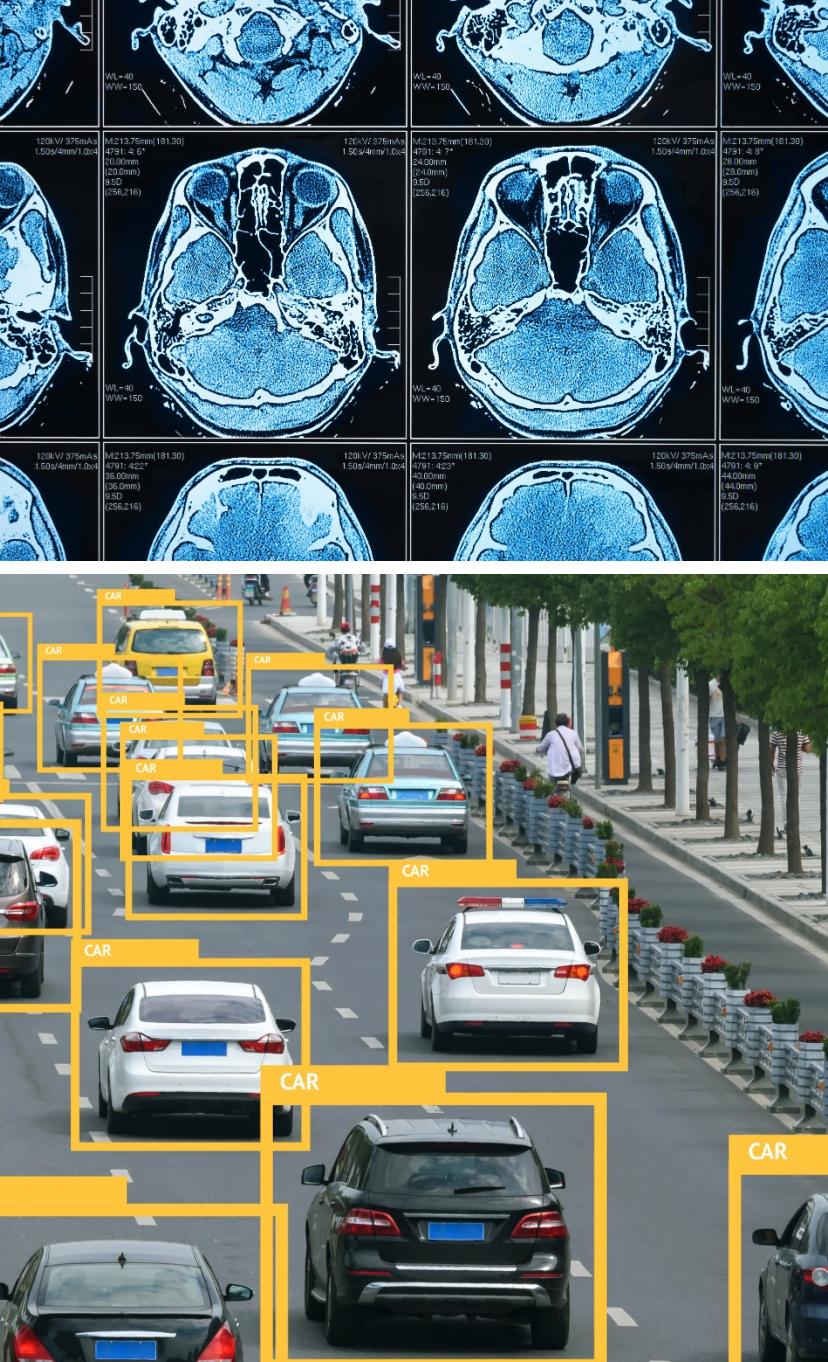
Introduction to Classification

- ▶ Decision Trees
- ▶ A Decision Tree Example
- ▶ Hands-On Exercise 5.1



Applications of Classification

- ▶ **Medical diagnosis**
 - Benign or malignant growth
- ▶ **Treatment-effectiveness analysis**
 - Responsive to treatment or not
- ▶ **Mortgage application**
 - Applicant likely to default or not
- ▶ **Targeted marketing**
 - Customer likely to respond to an offer or not
- ▶ **Object Recognition**
 - Identifying people or objects in an image



Classification: Introduction

- ▶ The goal of a classifier is to predict an outcome based on input data
- ▶ A number of predictor variables are combined to best decide the output for a target variable
 - Known as a classification model (a set of predictor rules)
- ▶ The datasets need to be representative samples of the data that the model will be applied to
 - Sometimes, a validation set is also used to tune the model
- ▶ Classification uses two datasets
 - The *training set* (seen data) builds the model
 - The observations have already been classified
 - The *test set* (unseen data) measures its performance in terms of its error rate
 - The percentage of incorrectly classified instances in the dataset

Census Dataset

- This census dataset might be used to predict future earnings

AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATIONNUM	MARITALSTATUS	OCCUPATION	RELATIONSHIP	RACE	SEX
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White Female
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspt	Husband	White Male
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White Female
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White Male
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White Female

Predictor variables/features to be used in training the model

CAPITALGAIN	CAPITALLOSS	HOURSPERWEEK	NATIVECOUNTRY	ABOVE50K
0	0	38	United-States	0
0	0	40	United-States	1
0	0	40	United-States	0
0	0	20	United-States	0
15024	0	40	United-States	1

Target value (earnings) to be predicted

Weather Dataset

- Weather attributes can be used to predict whether it will rain tomorrow or not

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindSpeed3pm
1	Canberra	8.0	24.3	0.0	3.4	6.3	NW	30	SW	NW	6	20
2	Canberra	14.0	26.9	3.6	4.4	9.7	ENE	39	E	W	4	17
3	Canberra	13.7	23.4	3.6	5.8	3.3	NW	85	N	NNE	6	6
4	Canberra	13.3	15.5	39.8	7.2	9.1	NW	54	NNW	W	30	24
5	Canberra	7.6	16.1	2.8	5.6	10.6	SSE	50	SSE	ESE	20	28
6	Canberra	6.2	16.9	0.0	5.8	8.2	SE	44	SE	E	20	24
7	Canberra	6.1	18.2	0.2	4.2	8.4	SE	43	SE	ESE	19	26

Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp9am	Temp3pm	RainToday	RISK_MM	RainTomorrow
68	29	1019.7	1015.0	7	7	14.4	23.6	No	3.6	Yes
80	36	1012.4	1008.4	5	3	17.5	25.7	Yes	3.6	Yes
82	69	1009.5	1007.2	8	7	15.4	20.2	Yes	39.8	Yes
62	56	1005.5	1007.0	2	7	13.5	14.1	Yes	2.8	Yes
68	49	1018.3	1018.5	7	7	11.1	15.4	Yes	0.0	No
70	57	1023.8	1021.7	7	5	10.9	14.8	No	0.2	No
63	47	1024.6	1022.2	4	6	12.4	17.3	No	0.0	No

Predictor variables/features used in training the model

Target value (Rain) to be predicted

Types of Classification Algorithms in Machine Learning

- ▶ There are many different classification algorithms
- ▶ For example:
 - Decision Trees
 - Naïve Bayes Classifier Algorithm
 - Support Vector Machine Algorithm
 - Logistic Regression
 - Artificial Neural Networks
 - Random Forests
 - Nearest Neighbors
- ▶ Choice of which algorithm to use often comes down to the modelling purpose, your own style, and the required model interpretability
- ▶ Several algorithms will often be tried to see which gives the best results

Contents

- ## ► Introduction to Classification

Decision Trees

- ▶ A Decision Tree Example
 - ▶ Hands-On Exercise 5.1

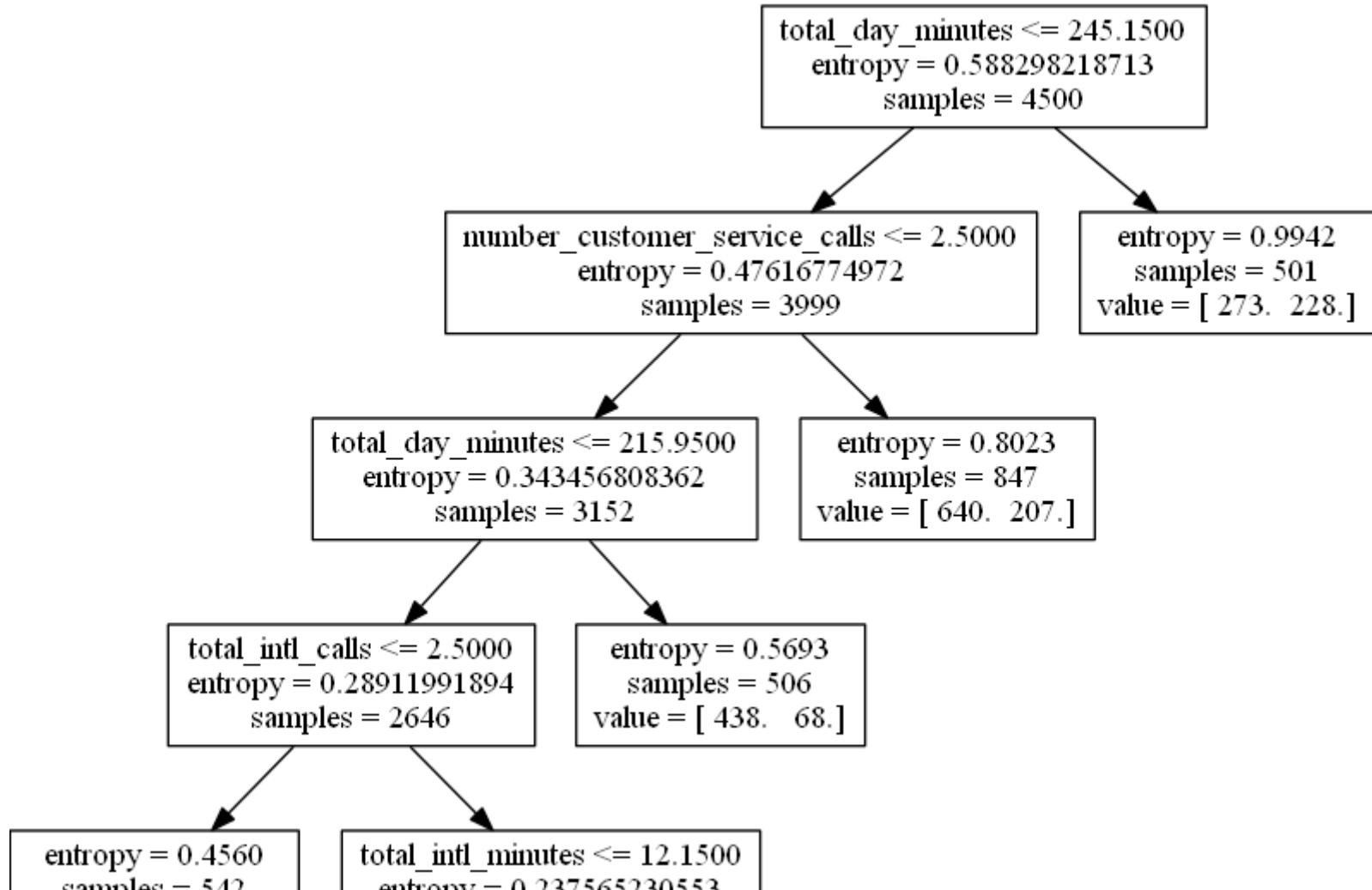


Decision Tree Classifiers

- ▶ **Decision Tree Classifiers separates data into various categories**
 - Comprises assigning a class label to a set of unclassified datasets
 - Classification is categorized as supervised learning
- ▶ **Uses recursive partitioning approach (divide and conquer)**
 - The process stops when partitioning does not improve the outcome
- ▶ **Starts with a root node and splits into multiple branches**
 - Internal nodes represent a test on a certain attribute
 - The result of the test is represented by a branch to the next level
- ▶ **Ends in leaf nodes that contain the “decisions”**

Decision Tree Classifiers

- The model can be viewed as a set of rules or a decision tree



How Are Decision Trees Used?

- ▶ **Exploring a dataset and selecting variables**
 - A decision tree selects the variables that are most useful in explaining a target variable
 - When faced with dozens of variables, decision trees can be used to direct attention to the most useful ones
 - The variables the decision tree splits on can be used in a different modeling technique
- ▶ **Classifying records**
 - Each leaf is labeled with its most probable class
- ▶ **Learning what events might lead to a resulting target value**
 - For example, what events/criteria can lead to a customer churning?

Suitability of Decision Trees

► Accuracy

- Where accuracy of the model has priority, how the predictions were made may not be as important

► Transparency

- In some situations, how the predictions were made must be transparent (e.g., a decision on a credit loan)

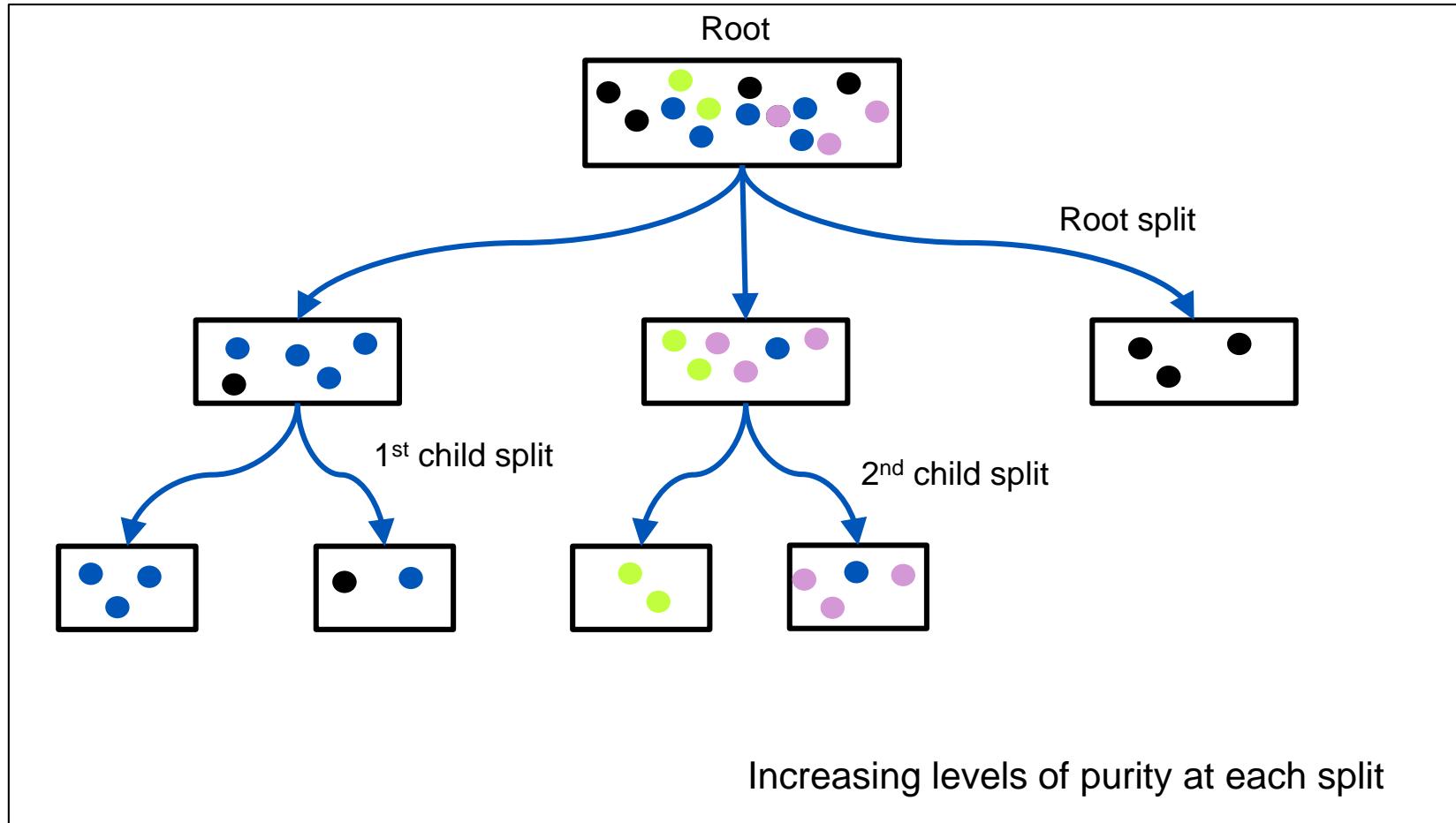


How Are Decision Trees Created?

- ▶ **Finding the initial split for recursive partitioning**
 - Starting with the known target values, a variable that is purest in its target values is chosen to split on
 - The algorithm evaluates all splits on the data
 - The best split is the one that leads to the greatest purity in the target value of the child nodes
 - A good split should produce nodes of similar size
 - This initial split produces two or more child nodes

Recursive Search

- Each child node is split in the same manner as the root mode in a recursive fashion



Finding the Initial Split

- ▶ **Splitting on numeric values**
 - Each distinct value is evaluated as a candidate for splitting on
 - The increase in purity is measured on each trial
- ▶ **Splitting on categorical values**
 - Variables are normally grouped together, such that each individually predict similar outcomes
 - A proportion of the target for each input variable is calculated, and those with similar proportions are grouped together
 - This grouping approach avoids high branching where the number of records at each node is reduced to allow further branching
- ▶ **Dealing with missing values**
 - Usually handled by using null as an allowable value
 - Alternatively, several splitting rules can be maintained for each node
 - Where the value for splitting is missing, the second splitting rule can be used

Evaluating Decision Trees

- ▶ **The quality of the tree is measured by applying it to test data and calculating how many it correctly classifies**
- ▶ **Each rule can also be evaluated individually**
 - A rule is a single path through the tree
 - Not all rules will be of equal quality
 - A node can be evaluated for correctness, as if it were a leaf node, i.e., the percentage of records that were classified correctly

Overfitting and Cross-validation

- ▶ Overfitting is a common problem in machine learning
 - Model is too closely aligned with the data it was trained on
- ▶ In k-fold cross-validation, the data-set is randomly partitioned into *k* **mutually exclusive** subsets
 - One is used for testing while the others are used for training
- ▶ This process is iterated throughout the whole *k* folds



Contents

- ▶ Introduction to Classification
 - ▶ Decision Trees

A Decision Tree Example

- ## ► Hands-On Exercise 5.1



Training Data and Test Data

- ▶ **Data needs to be split into a training dataset and a test dataset**
 - The proportions of the values in the target variable should be maintained in the training data and in the test data
- ▶ **To examine the proportion of classifications in a data attribute**

```
data = pd.read_csv('custchurn.csv')

data["churn"].value_counts()/data["churn"].count()

no      0.8586
yes     0.1414
Name: churn, dtype: float64
```

- ▶ **Use 90 percent of the data to train the model, and 10 percent to test it**
 - If data is randomly sorted, then taking the first 90 percent of the data for the training set is adequate
- ▶ **If the data is not randomly sorted, then this may lead to a poor model**
 - If the data is sorted according to churn, then the model would be created based on customers who didn't churn, and tested on customers who did

Randomizing the Dataset

- To randomize the dataset, we can use NumPy's `random.permutation()` function

```
data.head()
```

Unnamed: 0	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_
0	1	KS	128	area_code_415	no	yes	25	265.1	110
1	2	OH	107	area_code_415	no	yes	26	161.6	123
2	3	NJ	137	area_code_415	no	no	0	243.4	114
3	4	OH	84	area_code_408	yes	no	0	299.4	71
4	5	OK	75	area_code_415	yes	no	0	166.7	113

```
import numpy as np  
data_rand = data.iloc[np.random.RandomState(seed=0).permutation(len(data))]  
data_rand.head()
```

Unnamed: 0	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls
398	461	UT	90	area_code_415	no	no	0	261.8
3833	4474	MS	76	area_code_408	no	no	0	193.0
4836	3823	TX	104	area_code_415	no	no	0	304.0
4572	2077	NH	95	area_code_408	yes	no	0	228.9
636	734	NM	85	area_code_408	no	yes	37	229.6

Preprocessing the Data

- To build the classification model, you may wish to remove unwanted column attributes or convert binary columns

```
drop = ['state', 'area_code']
data_rand = data_rand.drop(drop, axis=1)
```

```
yes_no_cols = ['international_plan', 'voice_mail_plan']
data_rand[yes_no_cols] = data_rand[yes_no_cols] == 'yes'
```

Splitting the Data

- ▶ Split the dataset into a training set and a test set

```
train = data_rand[0:4500]
test = data_rand[4501:5000]
```

- ▶ Test to see that the proportion values in the target variable have remained the same

```
train["churn"].value_counts()/train["churn"].count()
```

no	0.857556
yes	0.142444
Name: churn, dtype:	float64

```
test["churn"].value_counts()/test["churn"].count()
```

no	0.869739
yes	0.130261
Name: churn, dtype:	float64

Building the Decision Model

- A number of packages need to be loaded into the session

```
from sklearn import tree
from sklearn.tree import export_graphviz
```

- Set the target variable and predictor attributes

```
Y = train.churn
X = train.drop('churn', axis=1)
```

- Build the classification model

```
clf = tree.DecisionTreeClassifier(criterion = "entropy")
clf = clf.fit(X, Y)
```

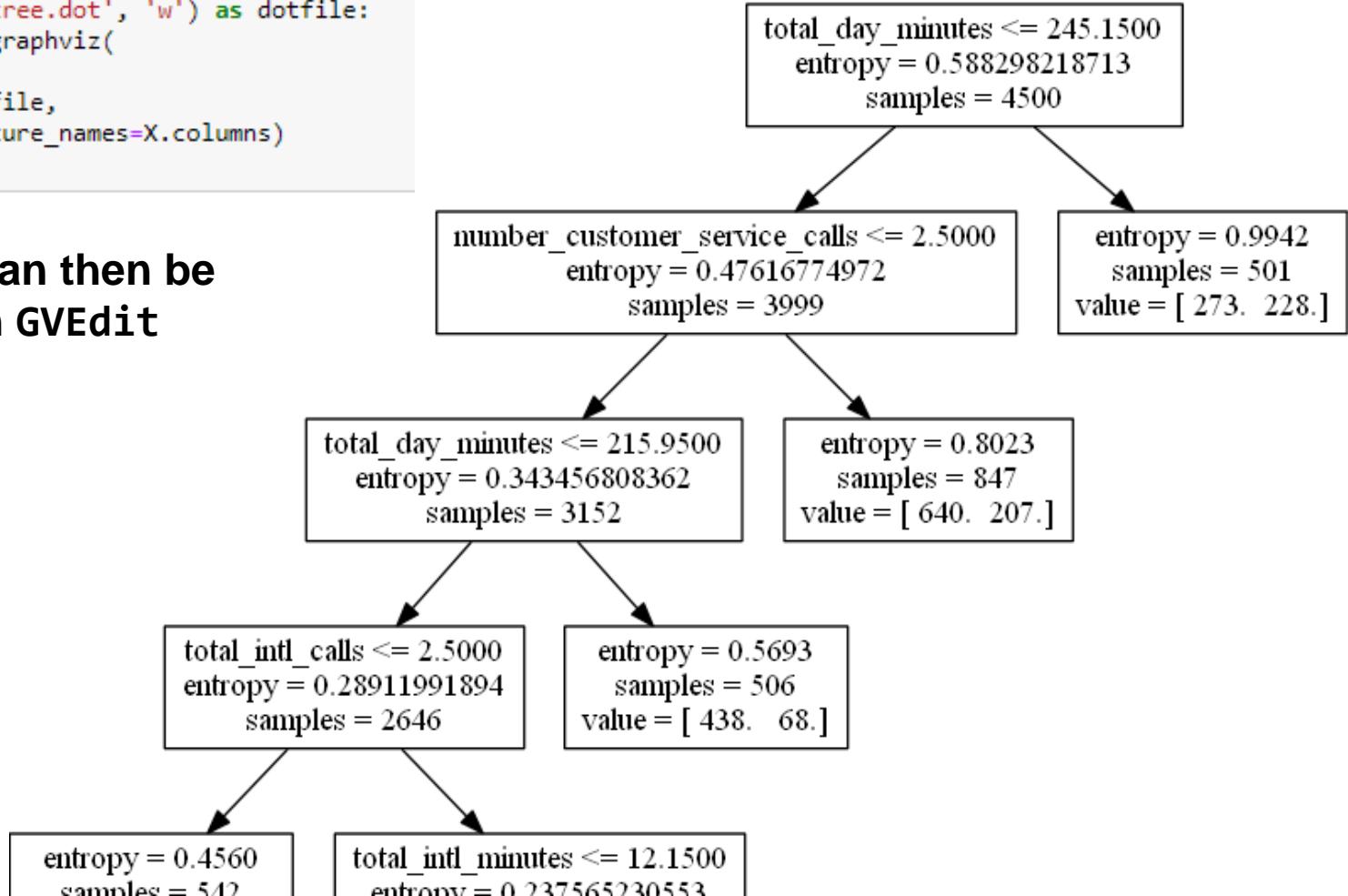
```
clf
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None,
splitter='best')
```

Reviewing the Decision Model

- Generates a GraphViz representation of the decision tree, which is then written into an output file

```
with open('tree.dot', 'w') as dotfile:  
    export_graphviz(  
        clf,  
        dotfile,  
        feature_names=X.columns)
```

- The file can then be viewed in GVEdit



Important Features

- Decision trees are a useful method for finding the most important features in a dataset

```
pd.DataFrame(clf.feature_importances_, columns = ["Imp"], index = X.columns).sort_values(['Imp'], ascending = False)
```

	Imp
total_day_minutes	0.186403
number_customer_service_calls	0.112638
international_plan	0.090864
total_intl_charge	0.087360
total_eve_minutes	0.068730
total_eve_charge	0.065168
total_intl_calls	0.057049
total_night_minutes	0.056560
Unnamed: 0	0.047556
voice_mail_plan	0.043259

Apply the Model to the Test Dataset

- To apply the model to the test dataset, use the `predict()` function

```
testY = test.churn  
  
testX = test.drop('churn', axis=1)  
  
y_pred = clf.predict(testX)
```

- To compare the predicted values with the actual values, use the `confusion_matrix()` function

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(testY, y_pred)  
  
array([[418,  14],  
       [ 13,  54]], dtype=int64)
```

Evaluating the Model

► Model evaluation

- Out of the 500 records, the model correctly predicted that 418 did not churn, and that 54 customers did churn
- This gives an accuracy of 94.4% and the error rate is 5.6%

Ensemble Methods

- ▶ Ensemble methods combine several decision trees to produce better predictive performance than a single decision tree
- ▶ A group of weak learners come together to form a strong learner
- ▶ Two techniques to create ensemble decision trees are:
 1. Bagging
 2. Boosting

Ensemble Methods: Bagging and Random Forests

- ▶ **Bagging (aka Bootstrap Aggregating) is used when the goal is to reduce the variance of a decision tree**
 - Several subsets of data are chosen randomly from the training sample
 - Each data subset is used to train decision trees
 - An average of all predictions from the different trees are used, which is more robust than a single decision tree
- ▶ **Random Forest is an extension over bagging**
 - In addition to taking the random subset of data, it also takes a random selection of features rather than using all features to grow trees
- ▶ **Advantages of using Random Forest technique:**
 - Handles higher dimensionality data very well
 - Handles missing values and maintains accuracy for missing data
- ▶ **Disadvantages of using Random Forest technique:**
 - Final prediction is based on the mean predictions from subset trees

Ensemble Methods: Boosting

- ▶ **Boosting is another ensemble technique to create a collection of predictors**
- ▶ **This technique involves sequential learning with early learners fitting simple models to the data and then analyzing the model for errors**
 - At every step, the goal is to solve for errors from the prior tree
 - When an input is misclassified, its weight is increased so that next model is more likely to classify it correctly
- ▶ **Gradient boosting is an extension over the boosting method**
 - In addition to the boosting, it also uses a gradient descent algorithm
 - The gradient descent algorithm can optimize any differentiable loss function
 - Ensemble of trees are built and are summed sequentially
 - Next tree tries to recover the loss (difference between actual and predicted values)
- ▶ **Disadvantages of using the gradient boosting technique:**
 - Prone to overfitting
 - Requires careful tuning of different hyperparameters

Contents

- ▶ Introduction to Classification
- ▶ Decision Trees
- ▶ A Decision Tree Example

Hands-On Exercise 5.1



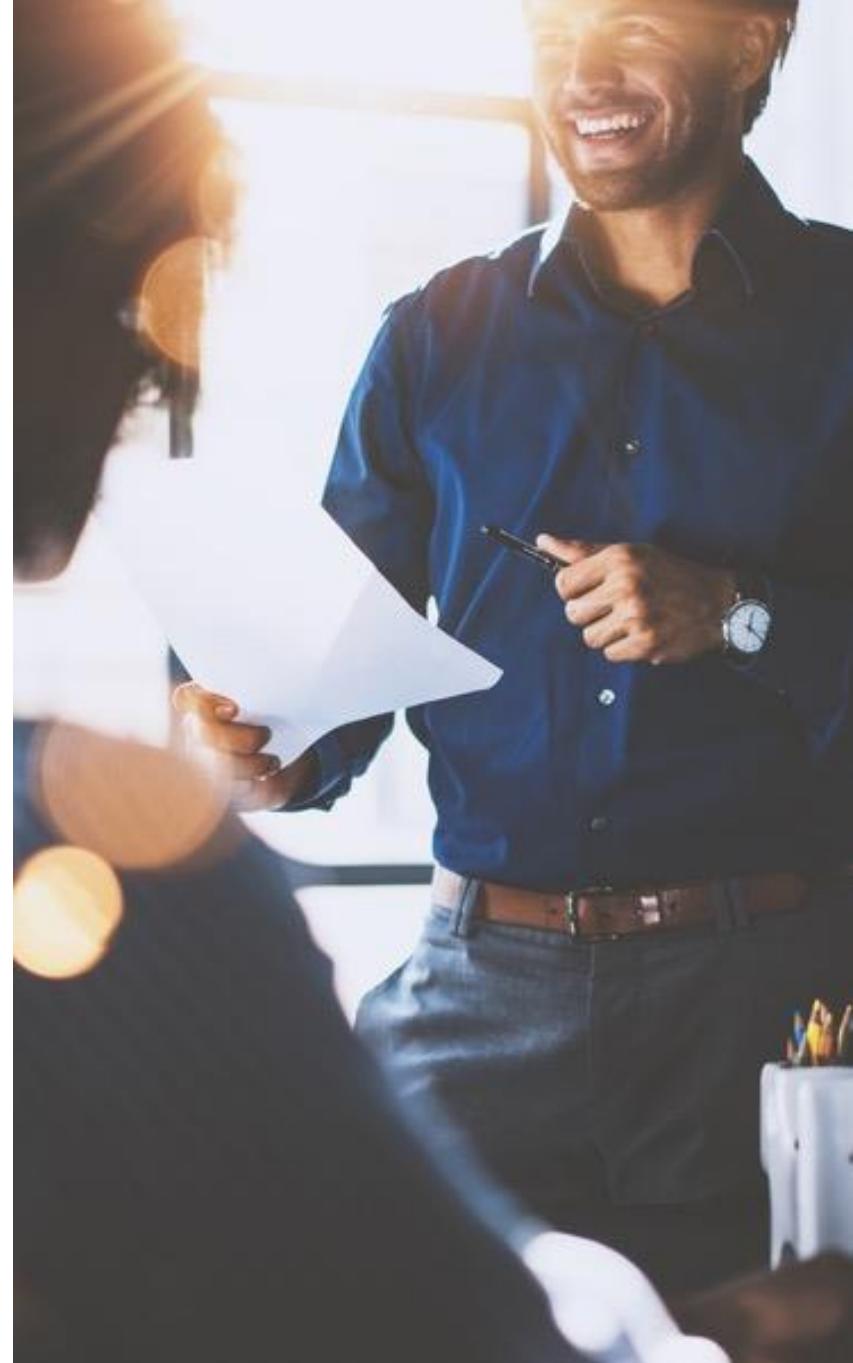


Hands-On Exercise 5.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 5.1: Working With Decision Tree Classifiers

Objectives

- ▶ Learn how classifiers are used to predict data categorization
- ▶ Explore how decision trees are built
- ▶ Build and apply a decision tree classifier

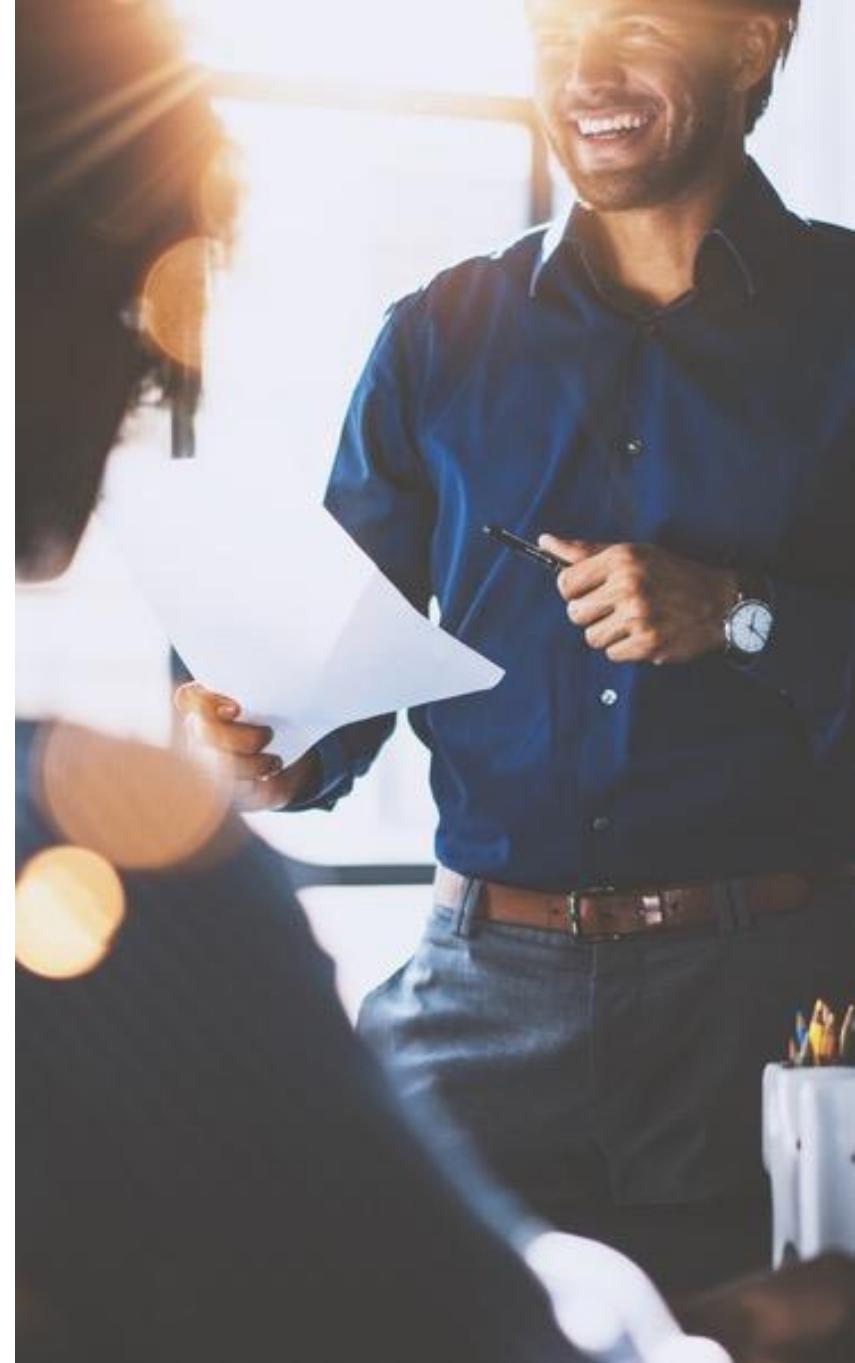


Chapter 6

Additional Classification Methods

Objectives

- ▶ Examine alternative approaches to classification
- ▶ Consider the benefits of logistic regression in binomial classification
- ▶ Investigate how neural networks can be used to make predictions
- ▶ Explore the probability foundations of Naive Bayes classifiers



Contents

Logistic Regression

- ▶ Hands-On Exercise 6.1
- ▶ Neural Networks
- ▶ Hands-On Exercise 6.2
- ▶ Naive Bayes
- ▶ Hands-On Exercise 6.3



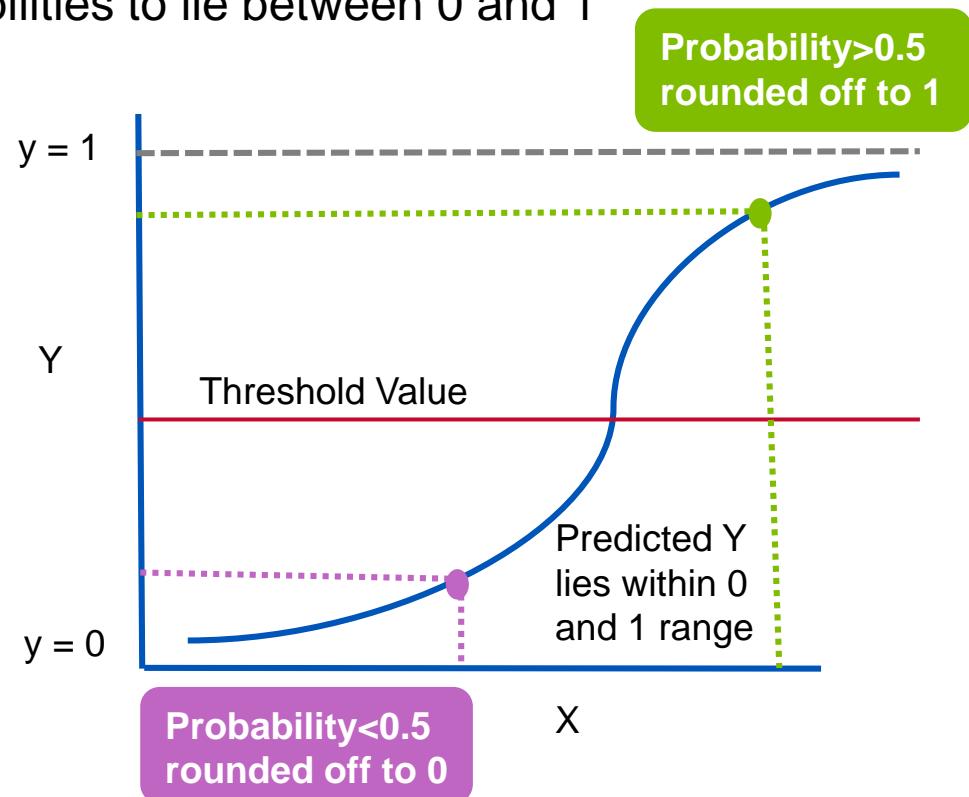
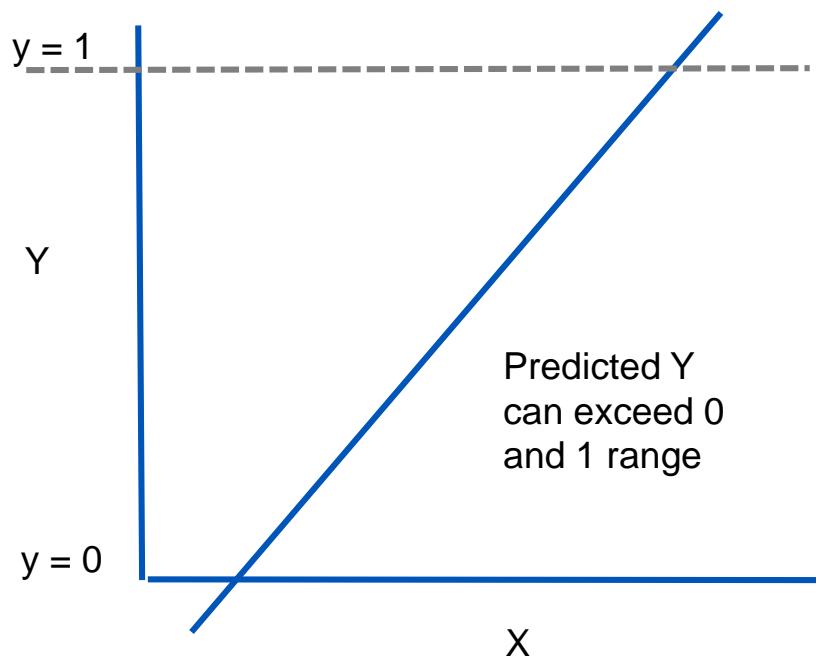
Introduction to Logistic Regression

- ▶ **Logistic Regression is typically used to predict binomial variables**
 - Deals with situations in which the outcome for a dependent variable can have two possible values (e.g., yes/no, passed/not passed)
 - Email classification might use attributes such as number of words, links and pictures to predict whether the email is spam (1) or not spam (0)
- ▶ **Used extensively in numerous disciplines, including the medical and social science fields**
 - Detecting the presence of diabetes
 - Predicting the likelihood of an online purchase
- ▶ **Can deal with both numeric and non-numeric variables**

Measuring the Relationship

- The logistic regression model is a non-linear transformation of the linear regression

- The “logistic” distribution is a sigmoid function
- When plotted, it has an S-shaped distribution
- Constrains the estimated probabilities to lie between 0 and 1



Probabilistic Model

- ▶ **Logistic regression or logit regression is a type of probabilistic classification model**
- $$\text{logit}(p) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \dots + \beta_n * x_n$$
- ▶ **The response variable in the logit model is the log of the odds in favor of the event occurring**
 - *Probability:* the number of occurrences of a certain event expressed as a proportion of *all events that could occur*
 - *Odds:* the number of occurrences of a certain event expressed as a proportion of the *number of non-occurrences* of that event
- ▶ **To convert between odds and probability:**
 - If the probability of something happening is P , then the odds of it happening is $P/(1 - P)$

$$\ln(\text{odds}) = \ln(p/(1-p)) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \dots + \beta_n * x_n$$

Logistic Regression in Python

- ▶ To perform logistic regression with Python, we will use the **Adult dataset**
- ▶ The **Adult dataset contains 32,561 census records**
 - The goal is to predict whether someone will earn more than \$50k based on demographic variables such as: race, gender, marital status, etc.
 - Uses both categorical and continuous variables

Data Preparation

- ▶ Cleaning and preprocessing the data is crucial for obtaining a good model fit and producing better predictive ability
- ▶ Load the Adult dataset into memory and view it

```
AdultData = pd.read_csv('Adult.csv')
AdultData.head()
```

	AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATIONNUM	MARITALSTATUS	OCCUPATION	RELATIONSHIP	RACE
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black

	SEX	CAPITALGAIN	CAPITALLOSS	HOURSPERWEEK	NATIVECOUNTRY	ABOVE50K
	Male	2174	0	40	United-States	0
	Male	0	0	13	United-States	0
	Male	0	0	40	United-States	0
	Male	0	0	40	United-States	0
	Female	0	0	40	Cuba	0

Data Exploration

- Once loaded, check for missing values and unique values in each variable

```
AdultData.isnull().any()
```

```
AGE           False
WORKCLASS    False
FNLWGT       False
EDUCATION    False
EDUCATIONNUM False
MARITALSTATUS False
OCCUPATION   False
RELATIONSHIP False
RACE          False
SEX           False
CAPITALGAIN  False
CAPITALLOSS  False
HOURSPERWEEK False
NATIVECOUNTRY False
ABOVE50K     False
dtype: bool
```

```
AdultData.nunique()
```

```
AGE            73
WORKCLASS      9
FNLWGT        21648
EDUCATION      16
EDUCATIONNUM  16
MARITALSTATUS  7
OCCUPATION     15
RELATIONSHIP   6
RACE           5
SEX            2
CAPITALGAIN    119
CAPITALLOSS    92
HOURSPERWEEK   94
NATIVECOUNTRY  42
ABOVE50K       2
dtype: int64
```

Data Preprocessing

► Examine the structure of the dataset

```
AdultData.shape
```

```
(32561, 15)
```

```
AdultData.columns
```

```
Index(['AGE', 'WORKCLASS', 'FNLWGT', 'EDUCATION', 'EDUCATIONNUM',
       'MARITALSTATUS', 'OCCUPATION', 'RELATIONSHIP', 'RACE', 'SEX',
       'CAPITALGAIN', 'CAPITALLOSS', 'HOURSPERWEEK', 'NATIVECOUNTRY',
       'ABOVE50K'],
      dtype='object')
```

```
AdultData.dtypes
```

```
AGE            int64
WORKCLASS      object
FNLWGT         int64
EDUCATION      object
EDUCATIONNUM   int64
MARITALSTATUS  object
OCCUPATION     object
RELATIONSHIP   object
RACE           object
SEX            object
CAPITALGAIN    int64
CAPITALLOSS    int64
HOURSPERWEEK   int64
NATIVECOUNTRY  object
ABOVE50K        int64
dtype: object
```

Treatment of Categorical Variables

- Examine the RELATIONSHIP variable and dummy code it for use in the model

```
AdultData['RELATIONSHIP'].head()
```

```
0      Not-in-family  
1          Husband  
2      Not-in-family  
3          Husband  
4          Wife  
Name: RELATIONSHIP, dtype: object
```

```
REL = pd.get_dummies(AdultData['RELATIONSHIP'], drop_first=True)
```

```
REL.head()
```

	Not-in-family	Other-relative	Own-child	Unmarried	Wife
0	1	0	0	0	0
1	0	0	0	0	0
2	1	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

- The value Husband is used as the reference

Treatment of Categorical Variables

► Dummy code the OCCUPATION variable

```
AdultData['OCCUPATION'].head()
```

```
0      Adm-clerical  
1      Exec-managerial  
2    Handlers-cleaners  
3    Handlers-cleaners  
4      Prof-specialty  
Name: OCCUPATION, dtype: object
```

```
OCC = pd.get_dummies(AdultData['OCCUPATION'], drop_first=True)
```

```
OCC.head()
```

	Adm-clerical	Armed-Forces	Craft-repair	Exec-managerial	Farming-fishing	Handlers-cleaners	Machine-op-inspct	Other-service	Priv-house-serv	Prof-specialty	Protective-serv	Sales	Tech-support	Transport-moving
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Logistic Regression With Python

► Define the predictive and target datasets

```
X = REL.join(OCC.join(AdultData[['AGE','CAPITALGAIN','EDUCATIONNUM']]))

y = AdultData['ABOVE50K']
```

► Split the data into a training and a testing set

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

► Run the logistic model

For repeatability

```
from sklearn.linear_model import LogisticRegression

logisticRegr = LogisticRegression()

logisticRegr.fit(x_train, y_train)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=None, solver='warn',
tol=0.0001, verbose=0, warm_start=False)
```

Viewing Model Results

► Examine the results from the model

- Some predictors have negative coefficients, which suggest that all other variables are equal, and that these predictors lead to a lower likelihood of earning above 50k

```
logisticRegr.coef_
```

```
array([[-2.23062995e+00, -1.61583620e+00, -3.23844750e+00,
       -2.48816186e+00,  1.35121931e-01, -1.41977118e-01,
       -6.83507396e-03, -1.06937065e-01,  8.15922861e-01,
      -7.97007644e-01, -8.56467294e-01, -5.10775889e-01,
      -1.27925776e+00, -1.66176534e-01,  4.60704823e-01,
      3.48616161e-01,  2.34974390e-01,  4.94808803e-01,
     -1.72227434e-01,  1.63352629e-02,  3.03226142e-04,
      2.55692456e-01]])
```

Predictive Ability of the Model

- Examine how well the model predicts the target on a new set of data

```
predictions = logisticRegr.predict(x_test)  
print(predictions)
```

```
[0 0 0 ... 0 1 0]
```

```
score = logisticRegr.score(x_test, y_test)  
print(score)
```

```
0.843631003562216
```

```
mse = np.mean((predictions-y_test)**2)  
print(mse)
```

```
0.15636899643778407
```

Calculates accuracy score of the predictions

Calculates the Mean Square Error (MSE) of the predictions

- The 0.84 accuracy on the test set is a reasonably good result

- The result is dependent on the manual split of the data
- For a more precise score, run a cross validation such as a k-Fold Cross-validation

ROC Curves and AUC

- ▶ We've already seen the use of Confusion Matrices in evaluating the performance of models
- ▶ Plotting a receiver operating characteristic (ROC) curve and calculating the area under the curve (AUC) are other performance measurements for a binary classifier
 - The ROC curve plots the true positive rate (TPR) on the y-axis, against the false positive rate (FPR) at various threshold settings on the x-axis
 - The TPR (also known as the *Recall* or *Sensitivity*) is calculated as:
$$TPR = TP / (TP+FN)$$
 - The FPR (also known as the *Specificity*) is calculated as:
$$FPR = FP / (FP+TN)$$
- ▶ The AUC is the area under the ROC curve
 - A model with good predictive ability should have an AUC closer to 1 rather than to 0.5 (1 is ideal)

Performance Measures

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

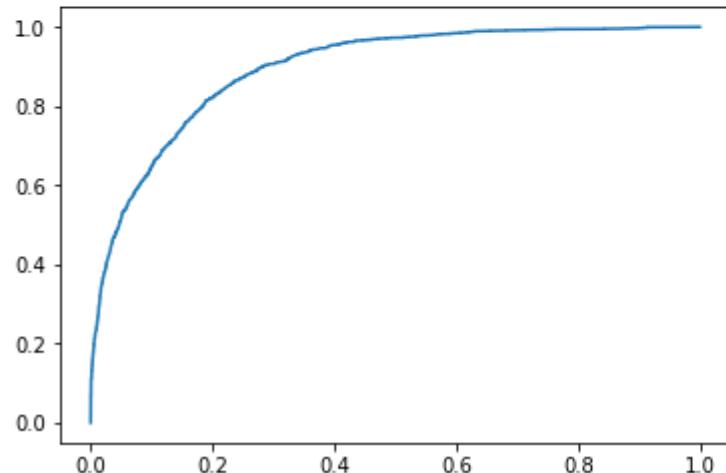
roc_auc_score(y_test, logisticRegr.predict(x_test))

0.752427759889337

import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, logisticRegr.predict_proba(x_test)[:,1])
plt.figure()
plt.plot(fpr, tpr)

[<matplotlib.lines.Line2D at 0x153020b8>]
```



Contents

- ▶ Logistic Regression

Hands-On Exercise 6.1

- ▶ Neural Networks
- ▶ Hands-On Exercise 6.2
- ▶ Naive Bayes
- ▶ Hands-On Exercise 6.3



Hands-On Exercise 6.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 6.1: Working With Logistic Regression Using Python



Contents

- ▶ Logistic Regression
- ▶ Hands-On Exercise 6.1

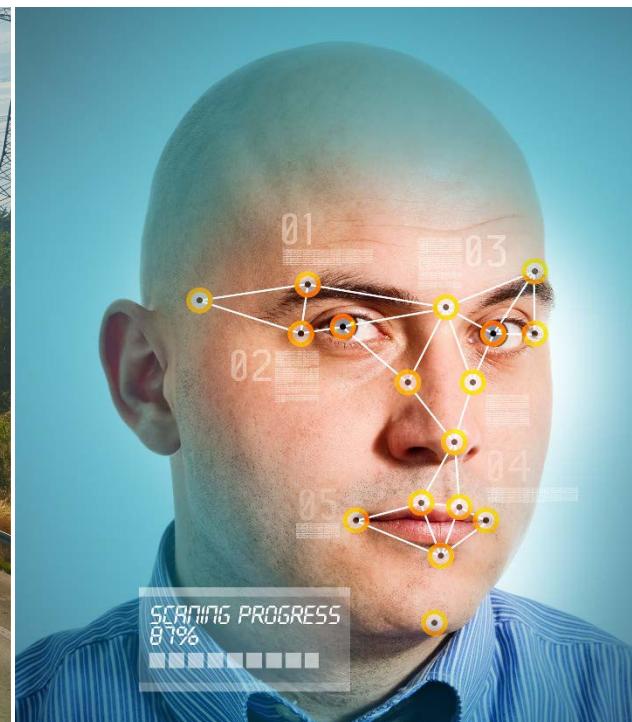
Neural Networks

- ▶ Hands-On Exercise 6.2
- ▶ Naive Bayes
- ▶ Hands-On Exercise 6.3



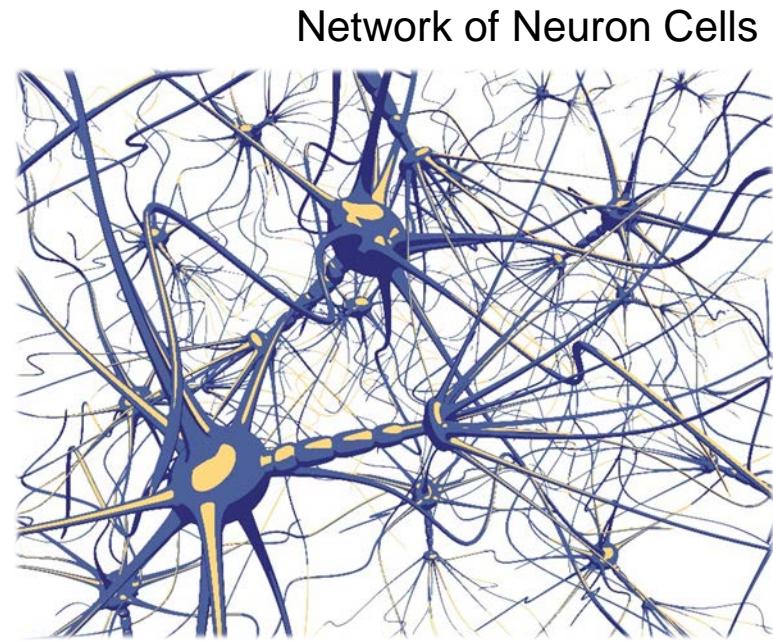
Introduction to Neural Networks

- Neural networks are algorithms modelled loosely on the human brain
 - They are designed to recognize patterns
- They interpret data through labelling or clustering raw data input
 - The patterns they recognize are numerical
 - All real-world data, be it images, sound or text must be transformed in order to input the data into a Neural Network model

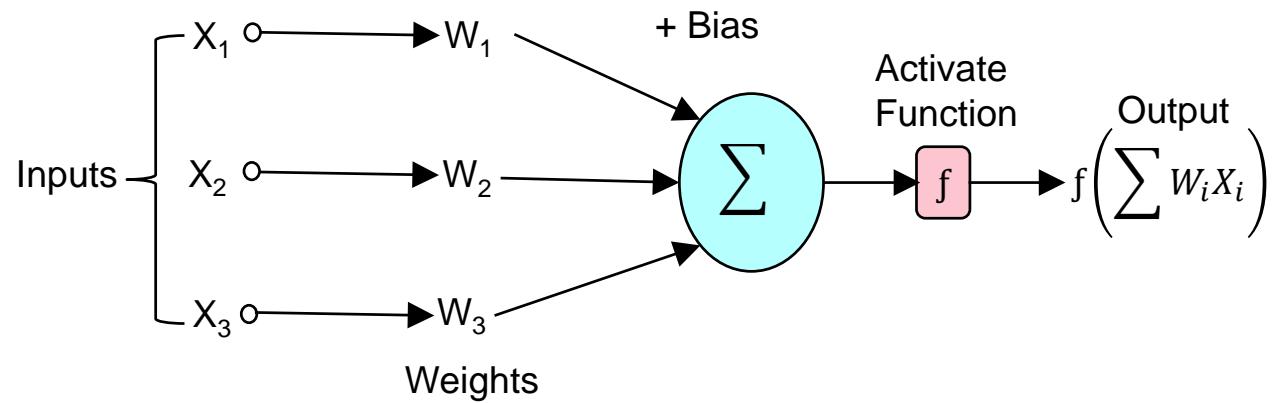


Introduction to Neural Networks

- ▶ A machine-learning technique inspired by biological neural networks
 - Attempts to mimic their learning pattern
 - Interconnected neurons produce an output signal transmitted to another neuron from inputs received
- ▶ Artificial Neural Networks (ANN) attempt to mimic this process



Structure of an Artificial Neuron

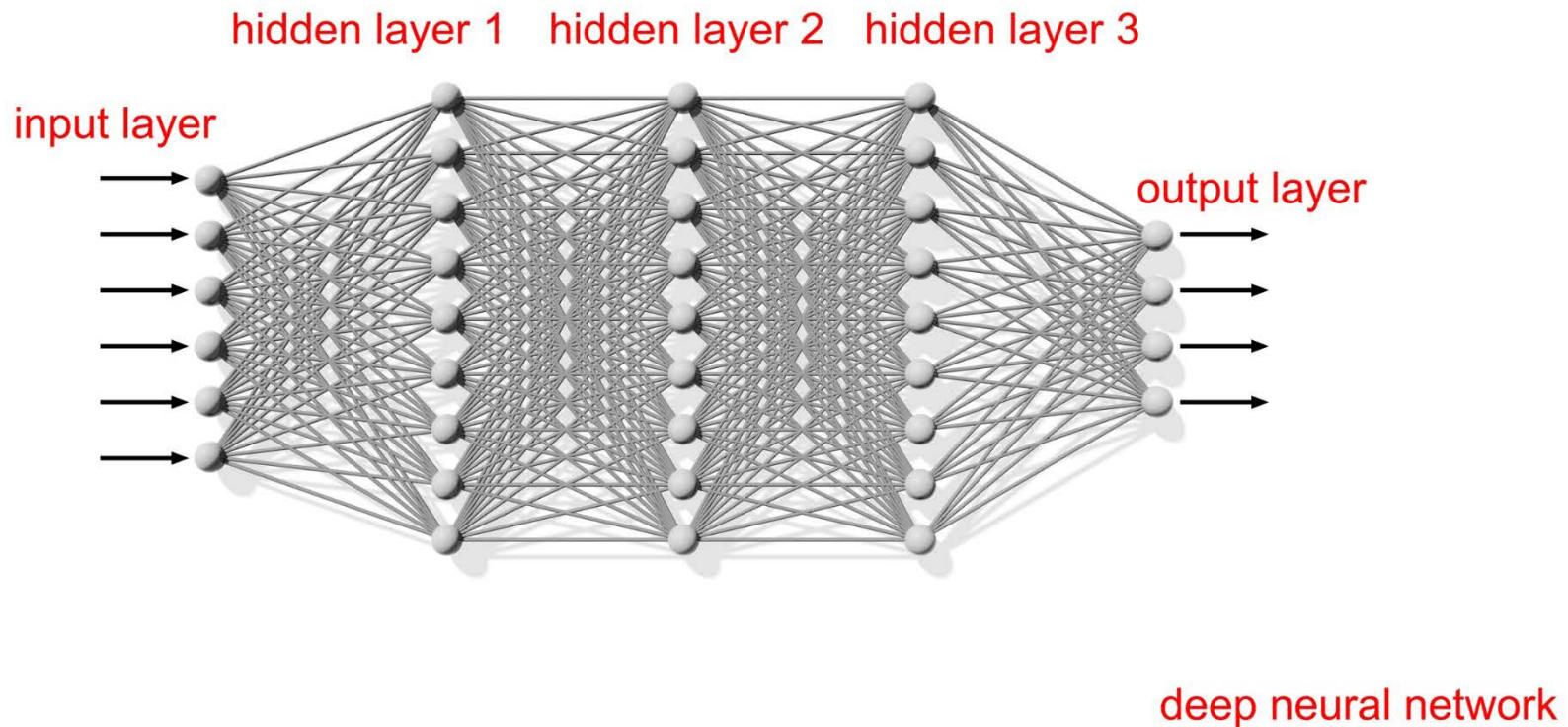


The Perceptron

- ▶ **The simplest neural network is called a *perceptron* and has**
 - One or more inputs
 - A bias
 - An activation function
 - A single output
- ▶ **The perceptron:**
 - Receives inputs
 - Multiplies them by some weight
 - Passes them into an activation function to produce an output
 - There are many possible activation functions to choose from (e.g., logistic function, a trigonometric function, a step function, etc.)
- ▶ **The bias is added to the perceptron to avoid issues where all inputs could be equal to zero (i.e., where the multiplicative weight would have no effect)**
 - Once the output is produced, it is compared to a known label, and the weights are adjusted accordingly
 - Weights usually start off with random initialization values
 - The process is repeated until a maximum number of allowed iterations or an acceptable error rate is reached

Creating a Neural Network

- ▶ To create a neural network, layers of perceptrons are added together
- ▶ The input layer takes feature inputs directly
- ▶ The output layer creates the resulting outputs
- ▶ Layers in between are known as hidden layers



Creating a Neural Network in Python

► We will use a wine dataset

- Results of a chemical analysis on three different wines
- Label is the target variable and the predictors are all continuous and represent 13 variables obtained as a result of chemical measurements

```
wines = pd.read_csv('wines.csv')
```

```
wines.head()
```

	label	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82

	Color_intensity	Hue	OD280_OD315_of_diluted_wines	Proline
	5.64	1.04	3.92	1065
	4.38	1.05	3.40	1050
	5.68	1.03	3.17	1185
	7.80	0.86	3.45	1480
	4.32	1.04	2.93	735

Data Preprocessing

- It is a good idea to normalize the data before training a neural network on it
 - Otherwise, the data may have difficulty converging before the allowed maximum number of iterations

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
winesScaled = pd.DataFrame(scaler.fit_transform(wines.iloc[:,1:14]), columns=wines.columns[1:14])
```

```
winesScaled.head()
```

	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins
0	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	0.510549	0.245283	0.274448
1	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	0.611814	0.320755	0.757098
2	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	0.664557	0.207547	0.558360
3	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	0.495781	0.490566	0.444795
4	0.834211	0.201581	0.582888	0.237113	0.456522	0.789655	0.643460	0.396226	0.492114

Encoding the Target Variable

- Neuralnet requires encoding of categorical variables
- The target variable contains three different labels: 1, 2, and 3
- These must be transformed into the following format

Label
1
1
2
2
3
3



Label1	Label2	Label3
1	0	0
1	0	0
0	1	0
0	1	0
0	0	1
0	0	1

Encoding the Target Variable

- The `get_dummies()` function can be used to generate a class indicator matrix

```
onehotTarget = pd.get_dummies(wines['label'],prefix='label')
onehotTarget.head()
```

	label_1	label_2	label_3
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

Split the Data

► Split the data into:

- A training dataset on which the neural network will be trained
- A test set to view its performance

```
from sklearn.model_selection import train_test_split  
Pred_train, Pred_test, Target_train, Target_test = train_test_split(winesScaled, onehotTarget)
```

Pred_train.shape

(132, 13)

Pred_test.shape

(45, 13)

Target_train.shape

(132, 3)

Target_test.shape

(45, 3)

Train the Neural Net

► Train the Neural Net

```
from sklearn.neural_network import MLPClassifier  
mlp = MLPClassifier(hidden_layer_sizes=(10,10,10), activation="tanh")
```

```
mlp.fit(Pred_train,Target_train)  
  
MLPClassifier(activation='tanh', alpha=0.0001, batch_size='auto', beta_1=0.9,  
    beta_2=0.999, early_stopping=False, epsilon=1e-08,  
    hidden_layer_sizes=(10, 10, 10), learning_rate='constant',  
    learning_rate_init=0.001, max_iter=200, momentum=0.9,  
    nesterovs_momentum=True, power_t=0.5, random_state=None,  
    shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,  
    verbose=False, warm_start=False)
```

Predict Results

```
predictions = mlp.predict(Pred_test)

predictions

array([[0, 1, 0],
       [0, 0, 0],
       [0, 1, 0],
       [0, 0, 0],
       [1, 0, 0],
       [0, 0, 1],
       [0, 1, 0],
       [1, 0, 0],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [1, 0, 0],
       [1, 0, 0],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [1, 0, 0],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 0, 1],
       [0, 1, 0],
       [0, 1, 0],
```

Evaluate the Results

- ▶ Create a confusion matrix to evaluate the results

```
from sklearn.metrics import classification_report,confusion_matrix  
  
print(confusion_matrix(  
    Target_test.values.argmax(axis=1),  
    predictions.argmax(axis=1)  
))  
  
[[12  0  0]  
 [ 2 14  0]  
 [ 2  0 15]]
```

Evaluate the Results

► Run a classification report to evaluate the results further

- Precision is the ratio of correctly predicted positive observations to the total predicted observations
- Recall (sensitivity) is the ratio of correctly predicted positive observations to all positive observations
- F1-score is a weighted average of Precision and Recall
- Micro average: average of total true positives, false negatives, and false positives
- Macro average: average of the result for each label
- Weighted average: uses the support of each label as a weight
- Samples average: used for multi-label (not binary) classification

```
print(classification_report(Target_test,predictions))
```

	precision	recall	f1-score	support
0	0.67	1.00	0.80	10
1	1.00	0.67	0.80	21
2	0.82	1.00	0.90	14
micro avg	0.83	0.84	0.84	45
macro avg	0.83	0.89	0.83	45
weighted avg	0.87	0.84	0.83	45
samples avg	0.79	0.84	0.81	45

Contents

- ▶ Logistic Regression
- ▶ Hands-On Exercise 6.1
- ▶ Neural Networks

Hands-On Exercise 6.2

- ▶ Naive Bayes
- ▶ Hands-On Exercise 6.3



Hands-On Exercise 6.2

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 6.2: Working With a Logistic Regression Using Python



Contents

- ▶ Logistic Regression
- ▶ Hands-On Exercise 6.1
- ▶ Neural Networks
- ▶ Hands-On Exercise 6.2

Naive Bayes

- ▶ Hands-On Exercise 6.3



Introduction to Naive Bayes Classification

- ▶ A probabilistic classifier based on applying Bayes' Rule
- ▶ Makes the assumption that there is no dependence between the attributes
 - The reason it is referred to as *Naive*
- ▶ It is one of the simplest classification algorithms, but it can also be very accurate
- ▶ Tries to classify instances based on the probabilities of previously seen attributes/instances, assuming complete attribute independence
 - Usually works out to give you a good classifier



Event Types

- ▶ **There are three types of events: independent, dependent, and mutually exclusive**
- ▶ ***Independent* events do not affect the probability of another event occurring**
 - For example, the receipt of an email offering you free entry to a technology event and a re-organization occurring within your company
- ▶ ***Dependent* events affect the probability of another event occurring, i.e., they are linked in some way**
 - For example, the probability of you getting to a conference on time can be affected by an airline staff strike with flights that may not run on time
- ▶ ***Mutually exclusive* events cannot occur simultaneously**
 - For example, the probability of rolling a 3 and a 6 on a single dice roll is zero—these two outcomes are mutually exclusive

Calculating Probability

- ▶ **Naive Bayes is based on probability fundamentals**
 - Proportions combined with addition, multiplication, and division
- ▶ **To calculate the probability of a single event occurring (*the observational probability*), take the number of times the event occurred, and divide it by the total number of processes which occurred that could have lead to that event**
 - For example, let's say a call center can receive in excess of 100 support calls in a day
 - This has occurred 50 times over the course of the last year
 - You want to know the probability of the calls being initially responded to, in under 3 minutes based on the previous times it happened
 - If the call center managed this record on 27 occasions, then the observational probability of 100 calls being answered in under 3 minutes is:
 $P(100 \text{ support calls in under 3mins}) = (27 / 50) = 0.54 \text{ (54\%)}$
- ▶ **Therefore, 100 calls can be dealt with in under 3 minutes in about half of the time, based on records of the 50 times it occurred in the past**

Multiplication Rules for AND Events

- ▶ To calculate probability of two or more events occurring simultaneously, consider whether events are independent or dependent
- ▶ If they are independent, then the *simple multiplication rule* is used

$$P(\text{outcome 1 AND outcome 2}) = P(\text{outcome 1}) * P(\text{outcome 2})$$

- ▶ To calculate the probability of the receipt of an email with a free tech event entry and a re-organization occurring in your workplace, the simple multiplication rule would be used
 - The two events are independent as the occurrence of one does not affect the other's chance of occurring
- ▶ If the tech event email receipt has a probability of 31%, and the probability of a staff re-organization is 82%, then the probability of both occurring is calculated as:

$$\begin{aligned} P(\text{email AND re-organization}) &= P(\text{email}) * P(\text{re-organization}) \\ &= (0.31) * (0.82) \\ &= 0.2542 \text{ (25%)} \end{aligned}$$

General Multiplication Rule

- ▶ If two or more events are dependent, then the *general multiplication rule* is used
- ▶ This formula is actually valid in both cases of independent and dependent events

$$P(\text{outcome 1 AND outcome 2}) = P(\text{outcome 1}) * P(\text{outcome 2} \mid \text{outcome 1})$$

- $P(\text{outcome 2} \mid \text{outcome 1})$ refers to the conditional probability of Outcome 2 occurring, given Outcome 1 has already occurred
 - The formula incorporates the dependence between the events
- If the events were independent, then the conditional probability is irrelevant as one outcome does not influence the chance of the other and $P(\text{outcome 2} \mid \text{outcome 1})$ is simply $P(\text{outcome 2})$
 - The formula just becomes the simple multiplication rule

General Multiplication Rule

► Application of the general multiplication rule to a deck of cards

- What would be the probability of drawing a King of any suit and an Ace of any suit from a 52-card deck with just two draws?
 - There are 4 Kings and 4 Aces in a standard deck
 - Drawing an Ace immediately affects the chances of drawing a King because there are fewer cards in the deck after the first draw
- The general multiplication formula can be used as follows:

$$\begin{aligned} P(\text{Ace AND King}) &= P(\text{Ace}) * P(\text{King} \mid \text{Ace}) \\ &= (4 / 52) * (4 / 51) \\ &= 0.006033183 (0.6\%) \end{aligned}$$

► If two events are mutually exclusive and cannot occur simultaneously, the multiplication rules cannot be applied

- The dice roll example describes such a scenario



Addition Rules for OR Events

- When calculating the probability of either one event or the other occurring (mutually exclusive), *the simple addition rule is used*

$$P(\text{outcome 1 OR outcome 2}) = P(\text{outcome 1}) + P(\text{outcome 2})$$

- What is the probability of rolling a 6 or a 3?
 - Both outcomes cannot occur simultaneously
 - The probability of rolling a 6 is $(1 / 6)$ and the same can be said for rolling a 3:
 $P(6 \text{ OR } 3) =$
 $(1 / 6) + (1 / 6) = 0.33 \text{ (33\%)}$

- If the events are not mutually exclusive, and can occur simultaneously, then use the following *general addition formula* which is always valid in both cases of mutual exclusiveness and of non-mutual exclusiveness

$$P(\text{outcome 1 OR outcome 2}) = P(\text{outcome 1}) + P(\text{outcome 2}) - P(\text{outcome 1 AND outcome 2})$$



General Addition Rule Example

- ▶ In the email and re-organization example, this would mean calculating the probabilities of both outcomes as well as the probability of both occurring simultaneously

$$P(\text{email}) + P(\text{re-organization}) - P(\text{email AND re-organization})$$

- ▶ The outcomes are independent
- ▶ Use the *simple multiplication rule* for two simultaneous events to calculate $P(\text{email AND re-organization})$

$$\begin{aligned}P(\text{email AND re-organization}) &= (0.31) * (0.82) \\&= 0.2542 \text{ (25\%)}\end{aligned}$$

$$\begin{aligned}P(\text{email OR re-organization}) &= (0.82) + (0.31) - (0.2542) = 0.8758 \\&\text{(88\%)}\end{aligned}$$

Summary of Rules

- The important rules to remember are the general forms of the multiplication rule and the addition rule because they are valid in all cases
- Using them in the above examples in place of the simple rules still yields the same results

$$P(\text{outcome 1 AND outcome 2}) = P(\text{outcome 1}) * P(\text{outcome 2} \mid \text{outcome 1})$$

$$\begin{aligned} P(\text{outcome 1 OR outcome 2}) &= \\ P(\text{outcome 1}) + P(\text{outcome 2}) - P(\text{outcome 1 AND outcome 2}) \end{aligned}$$

- Or how they are more generally written

$$P(A \text{ and } B) = P(A) * P(B \mid A)$$

$$P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$$

Naive Bayes in Python

- ▶ **We will use an SMS Spam data set**
 - A public set of 5,574 SMS labelled messages have been collected for mobile phone Spam research
- ▶ **The data set contains categorical data with two labels**
 - “Ham” (legitimate) or Spam
 - Each line is labelled with the correct class (“Ham” or Spam) followed by the raw message
- ▶ **There is skew in the outcome class**

```
sms['type'].value_counts()
```

```
ham    4827  
spam   747  
Name: type, dtype: int64
```

```
100 * sms['type'].value_counts() / len(sms['type'])
```

```
ham    86.598493  
spam   13.401507  
Name: type, dtype: float64
```

- This is okay for a generative Naive Bayes model as the model should depict real-world event numbers
 - Manipulating the data to achieve less skew would be dangerous

Split the Dataset

- ▶ Split the DataFrame into a DataFrame for features and another for the target

```
Pred = sms['text']
target = sms['type']
```

- ▶ Split the DataFrames into training and test datasets

```
from sklearn.model_selection import train_test_split
Pred_train, Pred_test, target_train, target_test = train_test_split(Pred, target, test_size = 0.1, random_state = 0)
```

- ▶ Create a DTM for the features

```
vectorizer = CountVectorizer()
counts = vectorizer.fit_transform(Pred_train.values)
```

Naive Bayes Classification

- Using the general multiplication formula for AND events, a general Naive Bayes formula is derived as

$$P(A \text{ and } B) = P(A) * P(B | A)$$

$$P(A | B) = P(A \text{ and } B) / P(B)$$

Gives ...

$$P(A | B) = P(A) * P(B | A) / P(B)$$

- The Naive Bayes formula in the context of the SMS data is

$$P(\text{spam} | \text{word}) = P(\text{spam}) * P(\text{word} | \text{spam}) / P(\text{word})$$

Naive Bayes Classification

- ▶ **Using this formula the Naive Bayes classifier can calculate conditional probabilities for the outcome**
 - Given the prior evidence of the words in the dataset
- ▶ **The classifier is referred to as “naive” because it is assumed there is independence between attributes**
 - There is likely some degree of dependency between the attributes
- ▶ **Calculation of the probability of the evidence is done by multiplying the individual probabilities of each piece of evidence occurring together using the simple multiplication rule for independent AND events**
- ▶ **Probabilities are not entirely correct, but are a good approximation due to “naive” assumption**
 - Adequate for the purposes of classification

Create, Train, and Test the Naive Bayes Classifier

► Create and train the classifier

```
targets = target_train.values

classifier = MultinomialNB()
classifier.fit(counts, targets)

MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

► Make a DTM for the test data features

```
test_count = vectorizer.transform(Pred_test)
```

► Make predictions for the test data

```
predictions = classifier.predict(test_count)
predictions

array(['ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'spam', 'ham', 'ham', 'spam', 'ham', 'ham', 'ham',
       'ham', 'ham', 'spam', 'ham', 'ham', 'spam', 'spam', 'ham',
       'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'spam', 'ham', 'spam', 'spam', 'spam', 'ham',
       'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham',
       'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham'],
      dtype='|S5')
```

Evaluate the Model

- ▶ Use the `classification_report()` function from `scikit-learn` to evaluate the model

```
from sklearn import metrics  
print(metrics.classification_report(target_test, predictions))
```

	precision	recall	f1-score	support
ham	0.98	0.99	0.99	469
spam	0.96	0.90	0.93	89
micro avg	0.98	0.98	0.98	558
macro avg	0.97	0.95	0.96	558
weighted avg	0.98	0.98	0.98	558

- ▶ Confusion matrix

```
from sklearn.metrics import confusion_matrix  
confusion_matrix(target_test,predictions)  
  
array([[466,    3],  
       [ 9,  80]], dtype=int64)
```

Contents

- ▶ Logistic Regression
- ▶ Hands-On Exercise 6.1
- ▶ Neural Networks
- ▶ Hands-On Exercise 6.2
- ▶ Naive Bayes

Hands-On Exercise 6.3



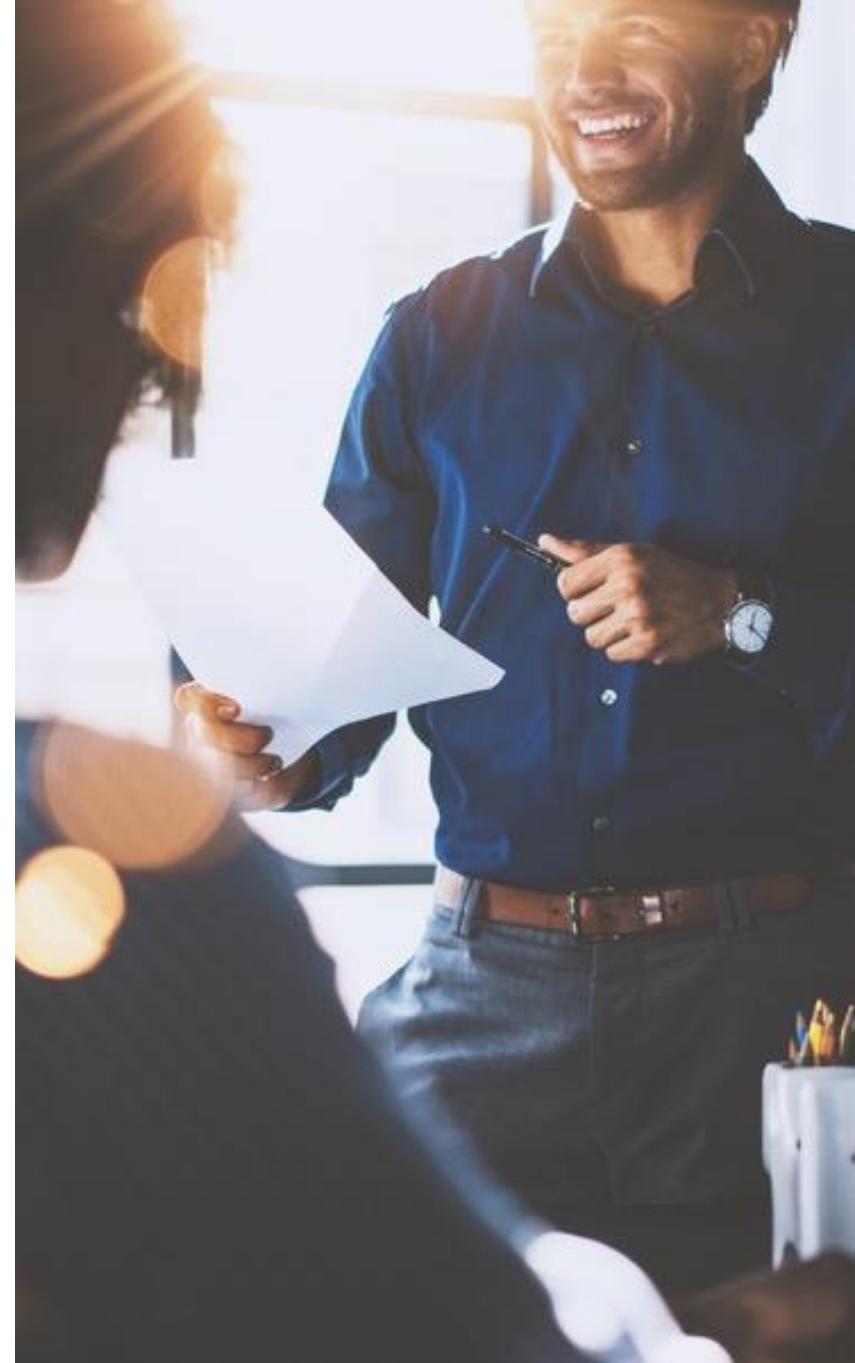


Hands-On Exercise 6.3

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 6.3: Working With Naive Bayes in Python

Objectives

- ▶ Examine alternative approaches to classification
- ▶ Consider the benefits of logistic regression in binomial classification
- ▶ Investigate how neural networks can be used to make predictions
- ▶ Explore the probability foundations of Naive Bayes classifiers



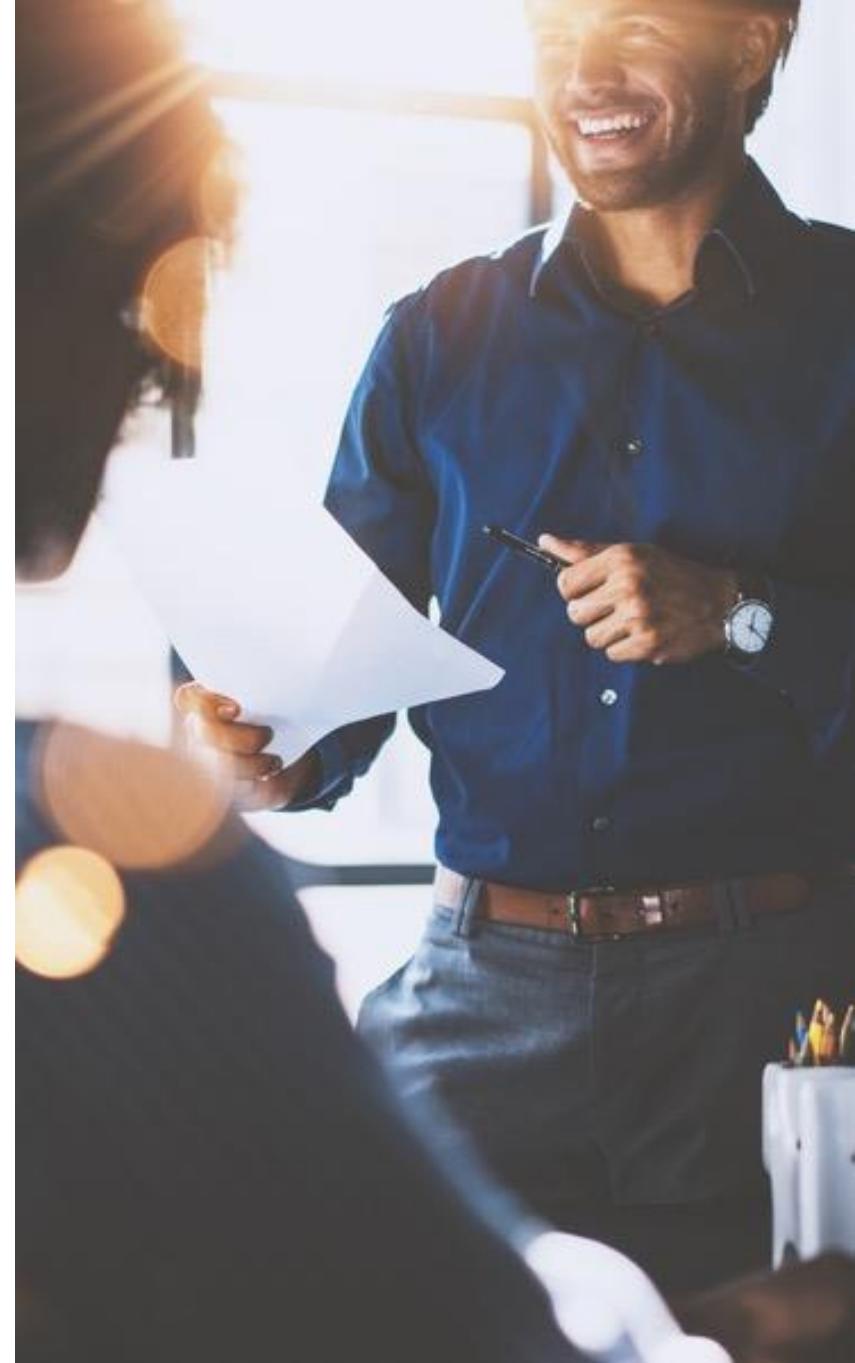
Chapter 7

Detecting Patterns in Data

With Clustering Analysis

Objectives

- ▶ Detect natural groupings in your datasets through clustering algorithms
- ▶ Explore the concept of similarity
- ▶ Use various distance measures to define similarity
- ▶ Perform top-down clustering with the K-Means algorithm
- ▶ Perform bottom-up clustering with hierarchical clustering



Contents

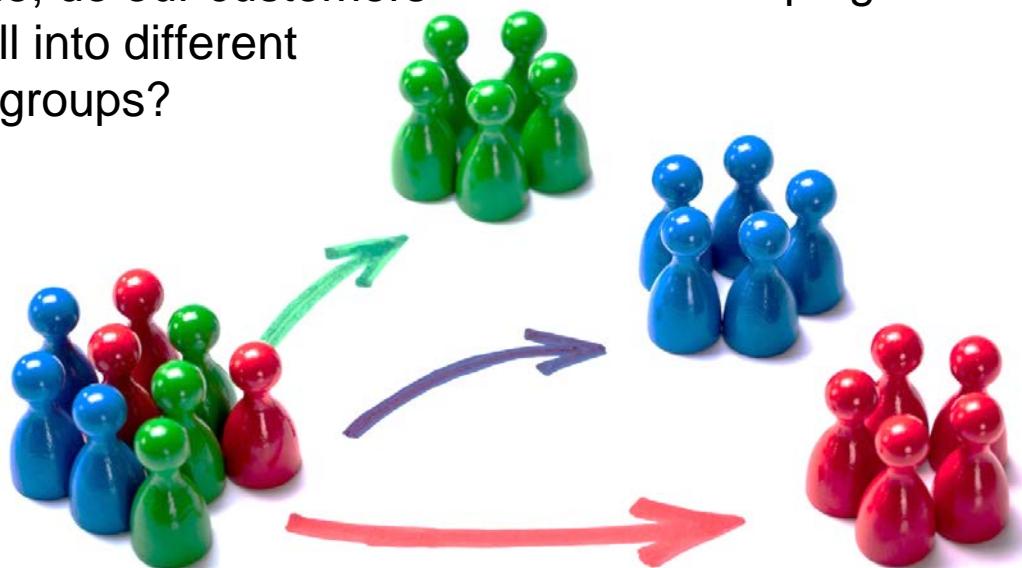
Introduction to Clustering

- ▶ **Similarity and Distance Measures**
- ▶ **K-Means Clustering**
- ▶ **Hierarchical Clustering**
- ▶ **Hands-On Exercise 7.1**



Clustering: Introduction

- ▶ **Clusters are not known apriori**
 - No prior knowledge of a target label, number or meaning of clusters
- ▶ **Clustering involves looking for natural groupings in data items that are not driven by prespecified criteria**
 - For example, do our customers naturally fall into different behavioral groups?
- ▶ **Referred to as *unsupervised learning* because there is no set target**
- ▶ **Example: Can be used to help marketers discover distinct groups from customer data**
 - Can use this knowledge to develop targeted marketing campaigns



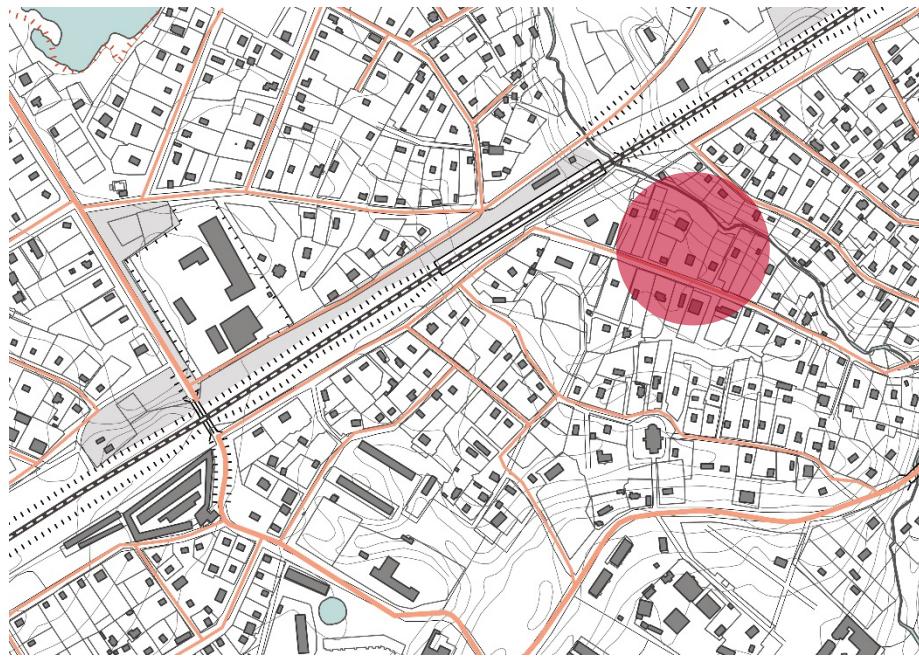
Clustering: Introduction

- ▶ Can also be used as a pre-processing step for further analytics
- ▶ Clustering may be used to define groups that are then used as target labels in a classification problem to predict new customer behavior



Business Applications for Clustering

- ▶ **Government management**
 - Crime hot spot analysis
 - Census data for economic and social analysis
- ▶ **Market research**
 - Segmentation
 - Targeted advertisements
 - Customer categorization
- ▶ **Image processing**
 - Identifying objects on an image (face detection)
- ▶ **Identify similar web usage patterns**
 - Weblog data: grouping similar access patterns



Customer Dataset

- ▶ **Customers can be clustered into natural groups according to attributes such as age, gender, spending habits, etc.**
 - The data scientist needs to determine what attribute can be fed into the clustering algorithm

	gender	age	annual_income	spending_score
CustomerID				
1	1	19	15	39
2	1	21	15	81
3	0	20	16	6
4	0	23	16	77
5	0	31	17	40

Understanding the Discovered Clusters

- ▶ On finding the clusters, we typically need to explore and understand them and finally give them more meaningful names
- ▶ For example, we might eventually end up with three good clusters with the following characteristics:

Cluster Attributes	Given Label
Females in their 30's, average income, average spending	Younger women, moderate spenders
Males and females in their 30's, high income, high spending	Wealthy, independent young adults
Males and females in their 50's, average income, average spending	Parents, moderate spenders

- ▶ These customers might then be targeted more creatively

Contents

- ▶ Introduction to Clustering

Similarity and Distance Measures

- ▶ K-Means Clustering
- ▶ Hierarchical Clustering
- ▶ Hands-On Exercise 7.1



Concept of Similarity

- ▶ Humans are quite good at recognizing similarity
- ▶ Clustering is based on the notion of similarity
 - Challenge: How do we explain why two customers, or two companies are similar to each other?
 - A data item (such as customer) has many attributes (e.g., age, marital status, home ownerships, etc.)
 - Once these attributes can be represented as numeric data, distance measures can be used as a measure of similarity
 - Requires data pre-processing in many cases (e.g., encoding categorical data)
- ▶ Observations within the group should be similar (shorter distance) to each other, and dissimilar (longer distance) from observations in other groups
 - Requires understanding the data to ensure an appropriate measure is being used



Distance Measures

- ▶ **The choice of distance measures has a strong influence on the clustering results**
 - The most common distance measure is the *Euclidean distance* (straight line)
 - Observations with high values of features will be clustered together
 - The same holds true for observations with low values of features
- ▶ **Depending on the type of the data and the question being researched, other measures might be used**
 - To identify clusters with the same overall profiles regardless of magnitude a correlation-based distance measure should be used
 - Considers objects to be similar if features are highly correlated, even though they might be far apart when using Euclidean distance
 - Distance is 0 if variables are perfectly correlated
 - Examples:
 - In gene expression data analysis, genes that are “up” and “down” together would be considered similar with a correlation-based distance measure
 - Shoppers with the same preference in purchases would be considered similar regardless of the volume of items they bought

Contents

- ▶ Introduction to Clustering
- ▶ Similarity and Distance Measures

K-Means Clustering

- ▶ Hierarchical Clustering
- ▶ Hands-On Exercise 7.1



K-Means Clustering

- ▶ **K-means is a type of unsupervised learning and one of the popular methods of clustering unlabelled data into k clusters**
- ▶ **It is a simple algorithm composed of three steps**
 1. Initialization
 - K individual data points are randomly chosen as initial centroids (cluster centers)
 - Requires that you know how many groups to look for in the dataset
 2. Cluster Assignment
 - Data points closest in distance to these centroids will create initial clusters
 3. Move the centroid
 - New centroid values are calculated as the mean of all data points in a cluster
- ▶ **Steps 2 and 3 are repeated until the centroids stop moving**

Scaling Data for Clustering

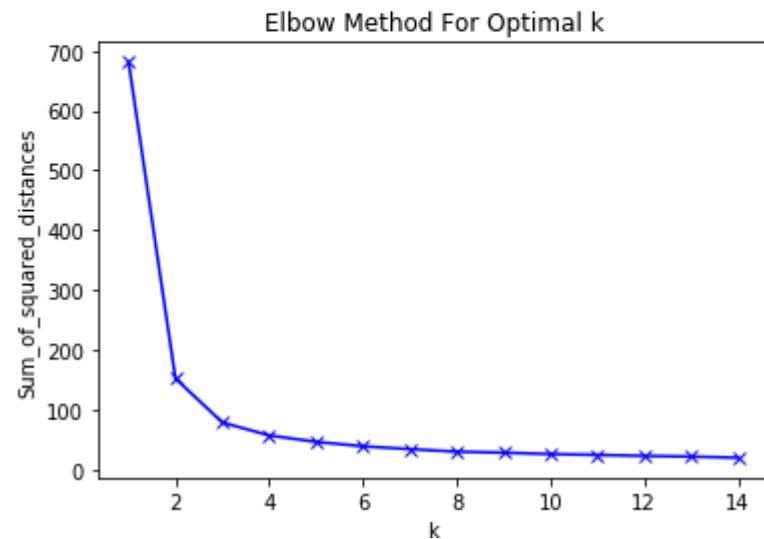
- ▶ **The value of distance measures is strongly related to the scale on which measurements are made**
 - Variables are often scaled (i.e., standardized) before measuring the inter-observation dissimilarities
- ▶ **The goal is to make the variables comparable**
 - Height and weight are measured in very different units (cm and kg)
 - It is good practice to standardize the variables for z to give each variable equal weight
- ▶ **Generally, variables are scaled to have**
 - Standard deviation one, and
 - Mean zero
- ▶ **K-Means is not very suited to binary values or to discrete and categorical attributes**
 - K-Means computes *means*
 - The mean value is not meaningful on this kind of data

Determining Optimal Number of Clusters

- One of the trickier tasks in clustering is identifying the appropriate number of clusters k
- The elbow method is one way to estimate the value k
 - For each k value, k-means is initialized and the inertia attribute is used to identify the sum of squared distances to the nearest cluster center
 - As k increases, the sum of squared distances tends to zero
 - If k were set to max value (the number of data points), each sample would form its own cluster
 - The sum of squared distances would equal zero
 - Below is a plot of sum of squared distances for k in the range 1 to 15
 - If the plot looks like an arm, then the elbow on the arm is optimal k

```
Sum_of_squared_distances = []
K = range(1,15)
for k in K:
    km = cluster.KMeans(n_clusters=k)
    km = km.fit(iris)
    Sum_of_squared_distances.append(km.inertia_)
```

```
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



Applying K-Means

► To apply K-Means using scikit-learn

```
from sklearn import cluster

iris = pd.read_csv('iris.csv')
# Drop the species column as it is non-numeric
iris = iris.drop('species', axis=1)
iris = np.array(iris)

k_means = cluster.KMeans(n_clusters=3)
k_means.fit(iris)

KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Visualizing the Clusters

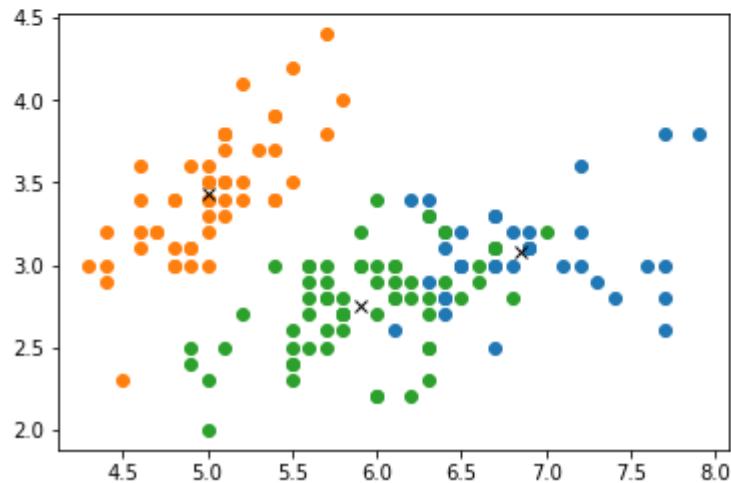
```
labels = k_means.labels_
centroids = k_means.cluster_centers_

from matplotlib import pyplot

for i in range(3):
    ds = iris[np.where(labels==i)]

    # plot the data observations
    pyplot.plot(ds[:,0],ds[:,1],'o')

    # plot the centroids
    pyplot.plot(centroids[i,0],centroids[i,1],'kx')
```



Evaluating the Clusters

- ▶ **Good clustering implies that the similarity between objects in a cluster is very high, and the similarity between the different clusters is very low**
- ▶ **The best way to evaluate clustering results is by examining the clusters visually and by making a judgment based on an understanding of**
 - The data
 - What the clusters represent
 - The purpose of the clusters
- ▶ **There are also quantitative methods of evaluating clusters that can be used as an aid to human intuition**

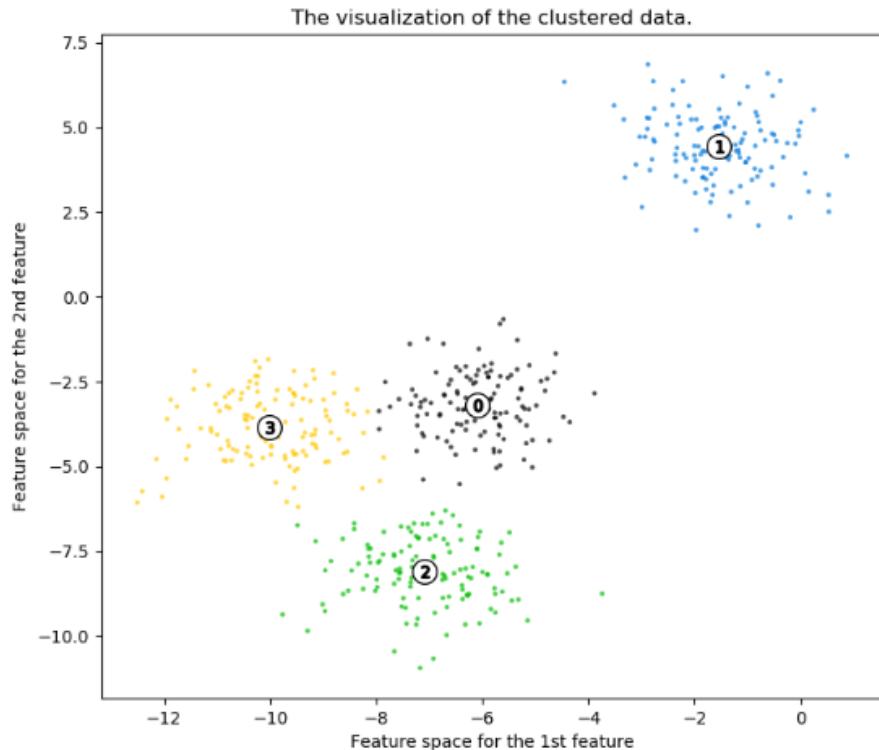
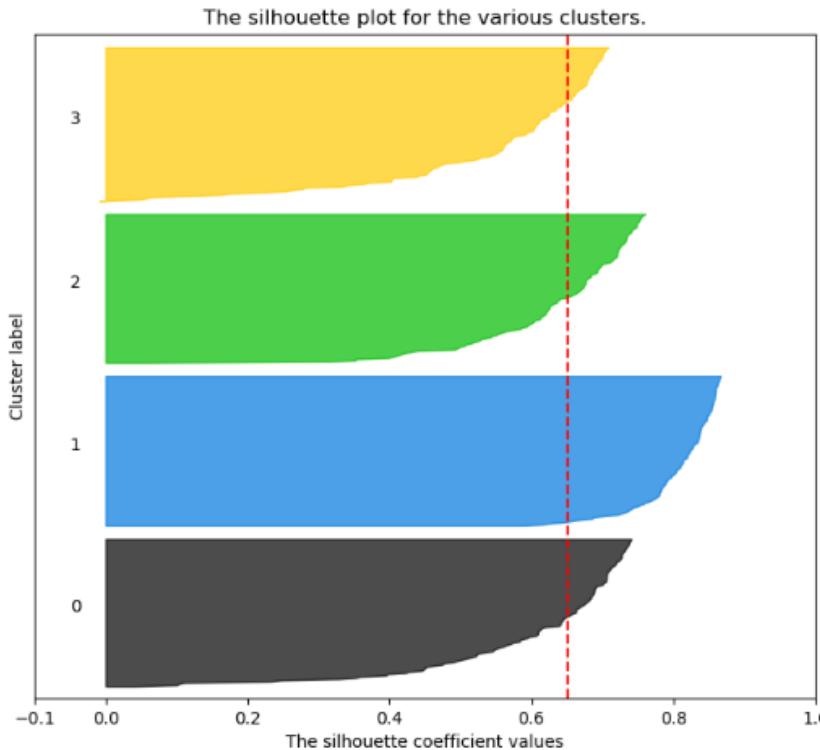
Evaluating Clusters With Silhouettes

- ▶ **Silhouettes are one such evaluation technique**
 - Compare tightness and separation in the resulting clusters
 - Show which objects lie comfortably within their cluster, and which are lying between clusters
- ▶ **For examples of this type of evaluation technique, see examples on the scikit-learn website**
http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Silhouette Plot

► The higher the silhouette value, the better

- 0.71–1.0: Good
- 0.51–0.70: Reasonable
- 0.26–0.50: Weak
- < 0.25: No cluster has been found



Limitations of K-Means Clustering

- ▶ **Applicable only when the mean is defined**
 - What about categorical data?
- ▶ **Initial number of clusters must be specified in advance**
- ▶ **Initial assignment of cluster centers is random**
 - Each time the algorithm is run, it may give different clusters
- ▶ **Each data point is assigned to only one cluster**
 - Sometimes there are points equidistant from two cluster centers
- ▶ **Not suitable for discovering clusters that have non-convex shapes**
- ▶ **Sensitive to outliers and noisy data**

Contents

- ▶ Introduction to Clustering
- ▶ Similarity and Distance Measures
- ▶ K-Means Clustering

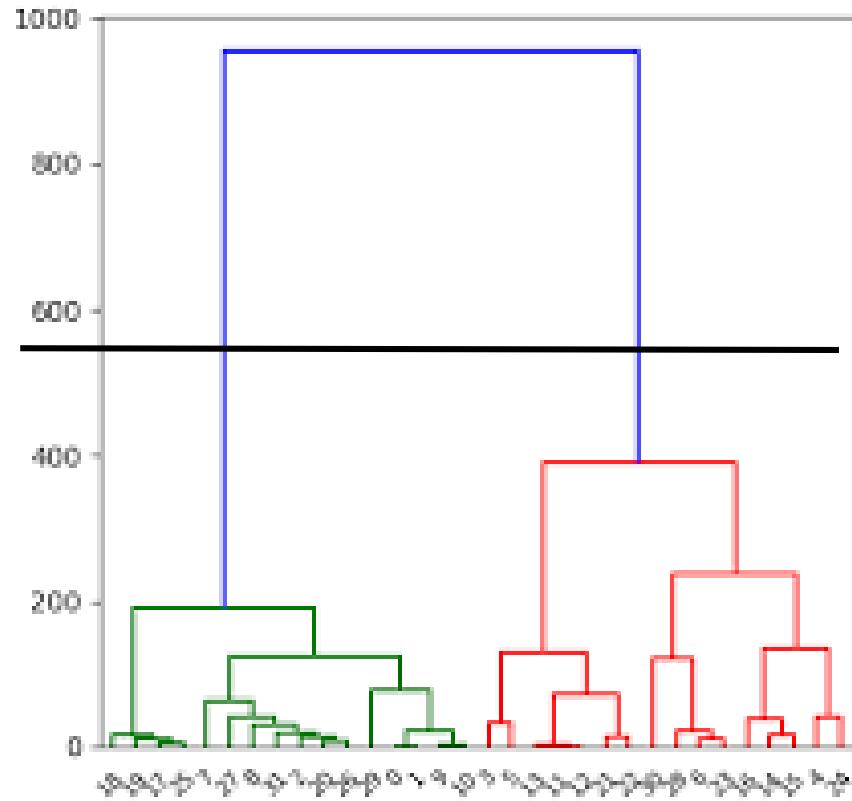
Hierarchical Clustering

- ▶ Hands-On Exercise 7.1



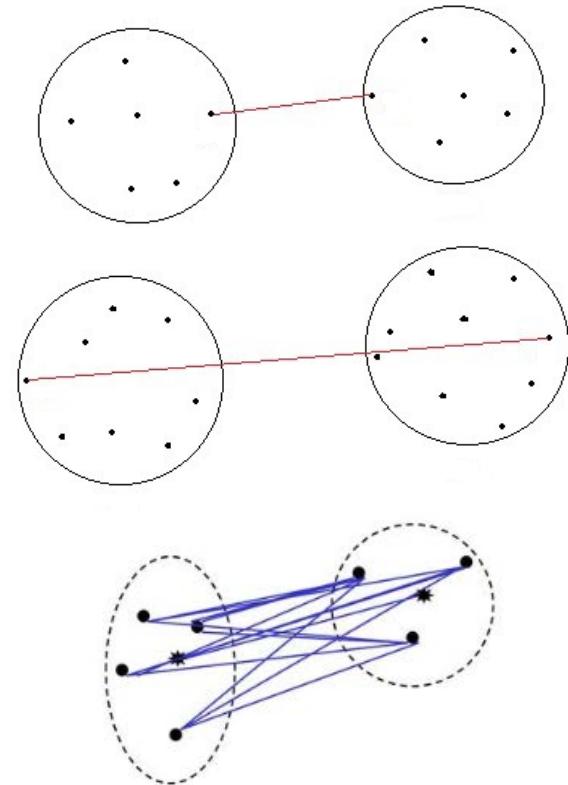
Hierarchical Clustering

- ▶ Another approach to finding natural groupings is *hierarchical clustering* (aka *bottom-up*, or *agglomerative clustering*)
- ▶ Does not require the number of clusters to be defined in advance
- ▶ Steps of hierarchical clustering
 - Start with every data point in a separate cluster
 - Iteratively merge similar data points/clusters until there is a single cluster
 - Produces a *dendrogram*
 - Height of the bars indicates how close the items are
 - Clusters are obtained by cutting the dendrogram at the appropriate level



Hierarchical Clustering Parameters

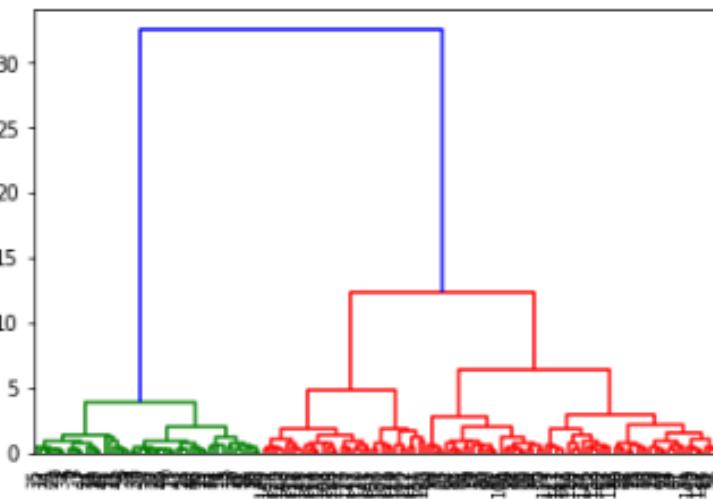
- ▶ A number of calculations need to be made
 - The distances between each data point need to first be calculated
 - The distance between each cluster then needs to be calculated
- ▶ The Agglomeration method specifies how each cluster should be merged for the next cycle of the algorithm
 - Min
 - The minimum distance between elements of each cluster
 - Max
 - The maximum distance between elements of each cluster
 - Average
 - The mean distance between elements of each cluster
 - Ward
 - Minimizes the increase in variance for the cluster being merged



Hierarchical Clustering

- To hierarchically cluster and plot the data:

```
from scipy.cluster.hierarchy import dendrogram, linkage  
  
Z = linkage(iris, 'ward')  
  
dendrogram(  
    Z,  
    leaf_rotation=90., # rotates the x axis labels  
    leaf_font_size=8., # font size for the x axis labels  
)  
plt.show()
```



Contents

- ▶ Introduction to Clustering
- ▶ Similarity and Distance Measures
- ▶ K-Means Clustering
- ▶ Hierarchical Clustering

Hands-On Exercise 7.1



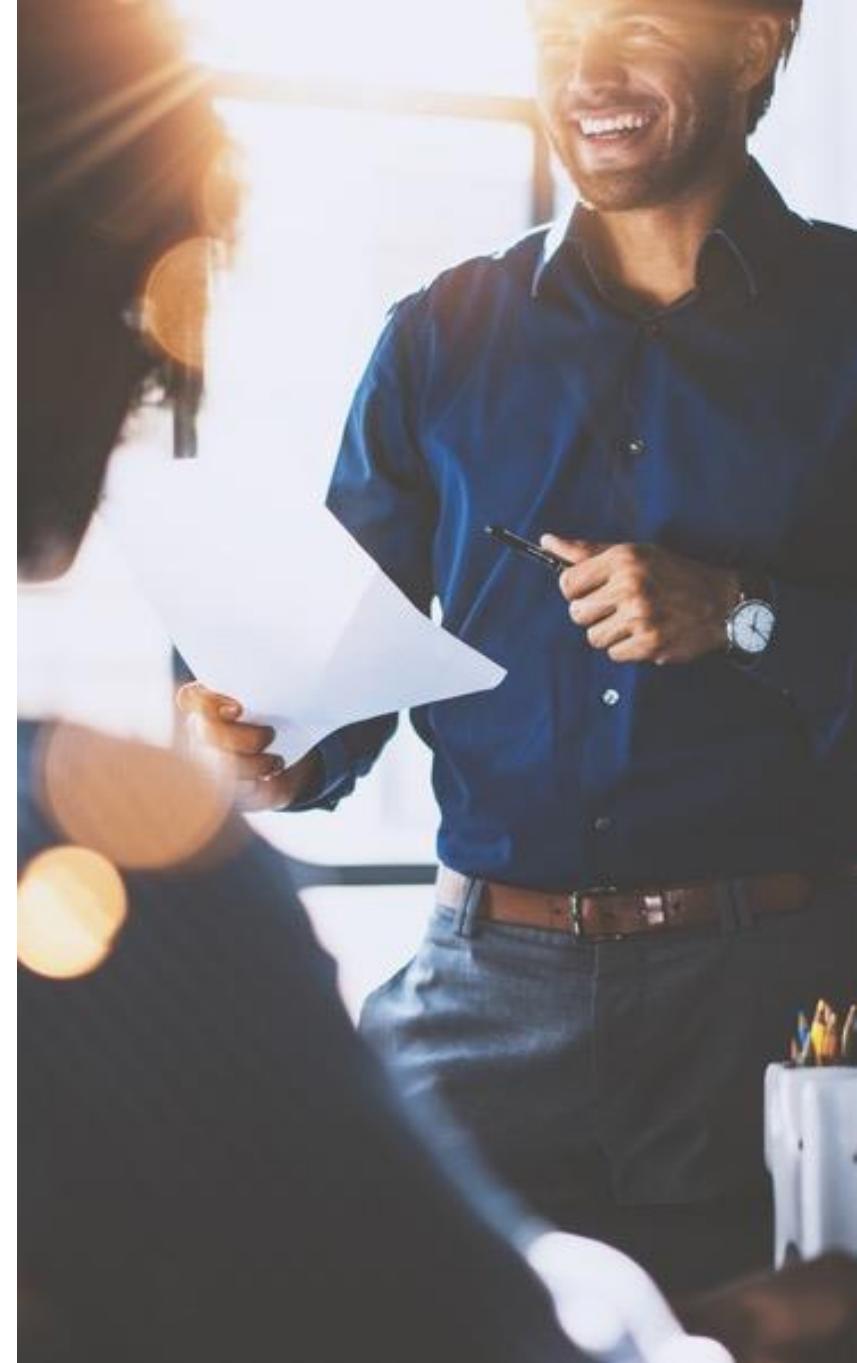


Hands-On Exercise 7.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 7.1: Applying Cluster Analysis on Structured Data With Python

Objectives

- ▶ Detect natural groupings in your datasets through clustering algorithms
- ▶ Explore the concept of similarity
- ▶ Use various distance measures to define similarity
- ▶ Perform top-down clustering with the K-Means algorithm
- ▶ Perform bottom-up clustering with hierarchical clustering



Chapter 8

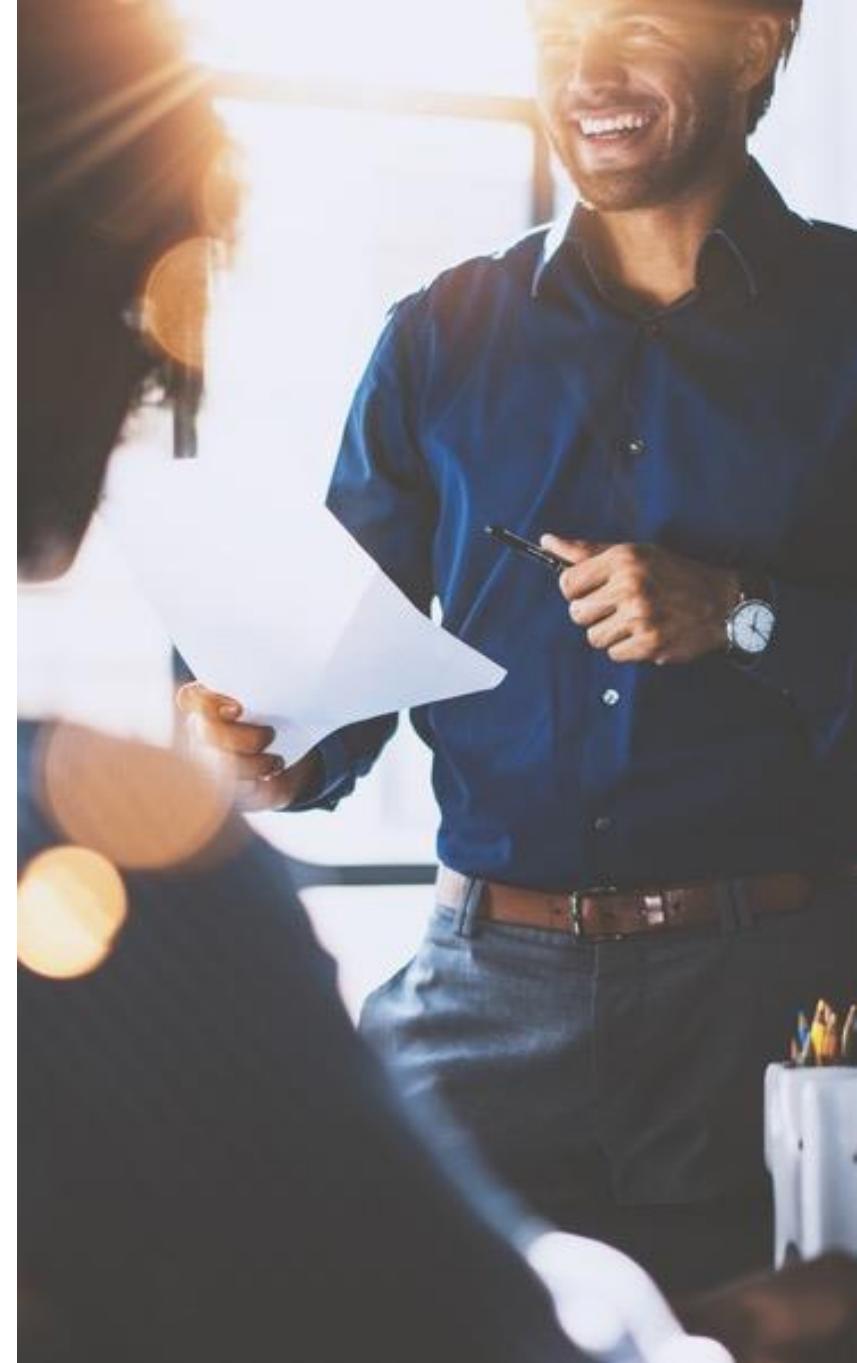
Association Rules



LEARNING TREE
INTERNATIONAL

Objectives

- ▶ **Generate association rules from transaction data**
- ▶ **Evaluate rules using measures of support, confidence, and lift**
- ▶ **Examine approaches toward improving rules**



Contents

Exploring a Transaction Dataset

- ▶ Mining Association Rules From Transaction Data
- ▶ Hands-On Exercise 8.1
- ▶ Building Recommenders From Transaction Data
- ▶ Hands-On Exercise 8.2



Machine Learning With Anonymous Transactions

- ▶ **Many transactions are anonymous**
 - The store has no knowledge about specific customers because there is no information identifying the customer in the transaction
 - The only information is the date and time, the location of the store, the cashier, the items purchased, and any coupons redeemed
- ▶ **By using market basket analysis, limited data can yield interesting and actionable results**
- ▶ **Analysts now have many more possibilities for identifying transactions and generating information about customers' behavior over time by using loyalty programs**



Loading a Transaction Dataset Into Python

- ▶ Transactional data where each row details a transactional record and each record is a comma-separated list of varying items needs to be loaded as a sparse matrix initially rather than a DataFrame

- Example csv data with records of varying length:

citrus fruit, semi-finished bread, margarine, ready soups

tropical fruit, yogurt, coffee

whole milk

pip fruit, yogurt, cream cheese, meat spreads

other vegetables, whole milk, condensed milk, long life bakery

- ▶ If the dataset were read into a DataFrame, then Python would choose the number of variables (columns) from the first row

- This would vary between transactions

Using Sparse Matrices in Python for Transaction Data

- ▶ **With a sparse matrix, every unique item in the dataset has its own column**
 - For example, if there are 2,500 unique items in the data, then there would be 2,500 columns
- ▶ **A sparse matrix is more memory efficient than a DataFrame because the entire matrix is not held in memory**
 - A DataFrame would hold the entire matrix in memory

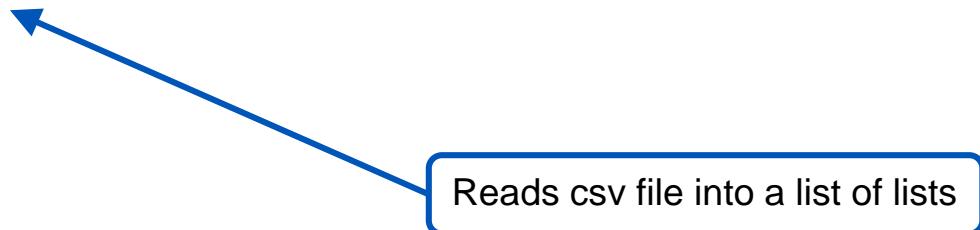
Loading the Data Into a Sparse Matrix

- ▶ Load in required libraries

```
import csv
```

- ▶ To create a sparse matrix from an external dataset, use the following code:

```
with open('Groceries.csv', 'r') as f:  
    data = list(list(rec) for rec in csv.reader(f))
```



Sparse Matrix

- We will use the `TransactionEncoder()` function here to transform the unique labels in the list into a one-hot encoded array

```
from mlxtend.preprocessing import TransactionEncoder  
te = TransactionEncoder()  
te_ary = te.fit(data).transform(data)
```

- The one-hot encoded sparse matrix can now be converted into a `DataFrame` for display

```
df = pd.DataFrame(te_ary, columns=te.columns_)
```

	Instant food products	UHT-milk	abrasive cleaner	artif. sweetener	baby cosmetics	baby food	bags	baking powder	bathroom cleaner	beef	...	turkey	vinegar	waffles	whipped/sour cream	whisk
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False

Item Set Combinations

- ▶ **Counts need to be calculated for each combination of items**
 - For example, in a set of just 100 items, there are 161,700 combinations of three items
 - A typical supermarket chain may have tens of thousands of different items in stock, leading to millions of possible combinations of two items and billions of combinations of three items
- ▶ **The number of transactions to be analyzed may be very large**
 - A supermarket chain can generate millions of transactions, each consisting of many items over the course of a year
 - Over millions of transactions, intense calculations are required to determine whether a particular combination of items is present in the transactions
 - Quickly becomes a sizable computation as the number of item combinations increases

The `apriori()` Function

- ▶ **There are a number of steps in generating association rules**
 - Counts must be generated for single-item transactions
 - A co-occurrence matrix must be generated for two items to find rules with two items (for three items to find rules with three items, etc.)
- ▶ **The `apriori` function which will be used by the `arules` unsupervised learning algorithm uses minimum support pruning to reduce the amount of calculations by requiring that a rule have the support of a minimum number of transactions**
 - For example, if a 20-million transaction dataset requires a minimum support of 1 percent, only rules supported by 200,000 transactions are of interest
 - *Minimum Support* refers to the minimum proportion of transactions that contain all the items in the set

Minimum Support Pruning in the apriori() Function

- ▶ The minimum support rule cascades to reduce the number of transactions that are of interest
- ▶ For example, if a rule with three items (apples, oranges, and pears) requires a minimum support of at least 5,000 transactions in the data, then
 - Apples must appear in at least 5,000 transactions
 - Oranges must appear in at least 5,000 transactions
 - Pears must appear in at least 5,000 transactions
- ▶ This minimum support pruning excludes items from calculations if they do not appear in the required number of transactions
 - This elimination also applies at each step of the algorithm
 - Each step eliminates combinations that do not meet the required support
 - This reduces the number of combinations that need to be considered during the next pass
 - Apples and oranges must appear together in at least 5,000 transactions
 - Apples and pears must appear together in at least 5,000 transactions, etc.

Exploring the Sparse Matrix

- To display the most frequent items/itemsets, first load in required libraries

```
from mlxtend.frequent_patterns import apriori  
from mlxtend.frequent_patterns import association_rules
```

- To generate the itemsets, use the apriori algorithm

```
apriori(df, min_support=0.1, use_colnames=True)
```

	support	itemsets
0	0.110524	[bottled water]
1	0.193493	[other vegetables]
2	0.183935	[rolls/buns]
3	0.108998	[root vegetables]
4	0.174377	[soda]
5	0.104931	[tropical fruit]
6	0.255516	[whole milk]
7	0.139502	[yogurt]

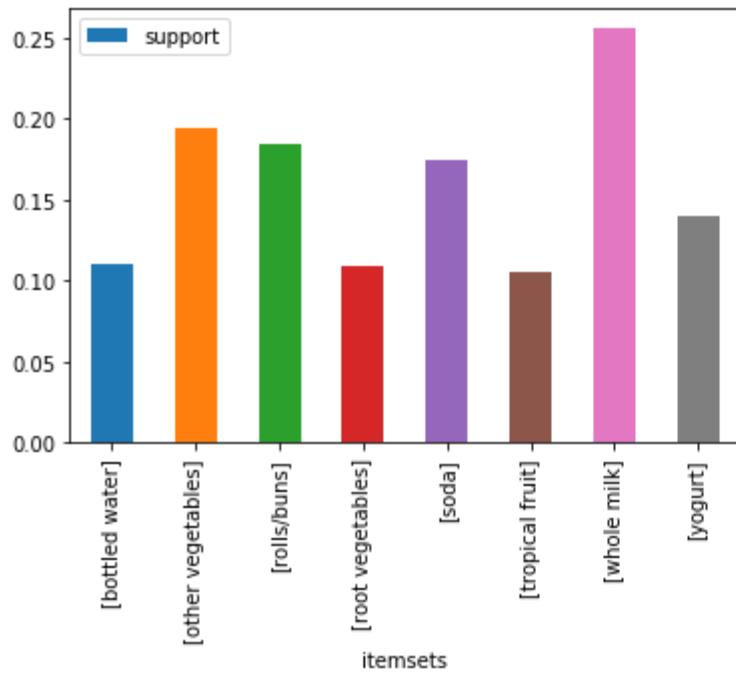
Exploring the Sparse Matrix: Plot Item Frequency

► Item frequency may also be visualized

- Produce a bar chart of the proportion of transactions containing each item
- The support parameter limits the items to those with a min level of support

```
apriori(df, min_support=0.1, use_colnames=True).plot.bar(x='itemsets', y='support')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x124b1320>
```



Exploring the Sparse Matrix: Size of Transactions

- To summarize the sizes of the transactions, the transactions are aggregated by the number of items
 - 2,159 transactions contain 1 item
 - 1 transaction contains 32 items

```
pd.DataFrame(df.sum(axis=1).value_counts()).T
```

1	2	3	4	5	6	7	8	9	10	...	21	20	23	22	29	26	28	32	24	27
0	2159	1643	1299	1005	855	645	545	438	350	246	...	11	9	6	4	3	1	1	1	1

1 rows × 29 columns

```
df.sum(axis=1).describe()
```

```
count      9835.000000
mean       4.409456
std        3.589385
min        1.000000
25%        2.000000
50%        3.000000
75%        6.000000
max       32.000000
dtype: float64
```

Summary statistics show:

- The median purchase size is three items
- Roughly half of the transactions contain more than three items



Item Frequency

- To examine the frequency of items, use the function `sum()`
 - To show the proportion of transactions that contain the item(s), divide by the number of transactions in the DataFrame

`df.sum()`

Instant food products	79
UHT-milk	329
abrasive cleaner	35
artif. sweetener	32
baby cosmetics	6
baby food	1
bags	4
baking powder	174
bathroom cleaner	27
beef	516
berries	327
beverages	256
bottled beer	792
bottled water	1087
brandy	41
brown bread	638
butter	545
butter milk	275
cake bar	130
candles	88
candy	294
canned beer	764

`df.sum()/df.shape[0]`

Instant food products	0.008033
UHT-milk	0.033452
abrasive cleaner	0.003559
artif. sweetener	0.003254
baby cosmetics	0.000610
baby food	0.000102
bags	0.000407
baking powder	0.017692
bathroom cleaner	0.002745
beef	0.052466
berries	0.033249
beverages	0.026029
bottled beer	0.080529
bottled water	0.110524
brandy	0.004169
brown bread	0.064870
butter	0.055414
butter milk	0.027961
cake bar	0.013218
candles	0.008948
candy	0.029893
canned beer	0.077682

Contents

- ▶ Exploring a Transaction Dataset

Mining Association Rules From Transaction Data

- ▶ Hands-On Exercise 8.1
- ▶ Building Recommenders From Transaction Data
- ▶ Hands-On Exercise 8.2



Association Rules

- ▶ Association Rules are used to discover relationships between seemingly unrelated items within a transactional dataset
- ▶ An unsupervised machine learning technique easily understood by business users
 - There is no specific target specified
- ▶ Based on *market basket analysis*
 - Helpful in placement of products on aisles (supermarkets, convenience stores, drug stores, and fast-food chains)
 - Reminds the customer of what relevant items they might be interested in buying, thus helping stores cross-sell in the process



Association Rules

- ▶ **Finds relationships between set of elements of every distinct transaction**
 - Helps uncover relationships between items in enormous datasets
 - Estimates how items are related to each other
 - Uniquely identified items within transactions are studied as one dataset
- ▶ **Different from collaborative filtering**
 - Collaborative filtering ties back all transactions corresponding to a user ID to identify similarity between users' preferences
 - Helpful in recommending items on e-commerce websites, recommending songs on Spotify, etc.
- ▶ **Examples of use:**
 - Amazon recommendations
 - Certain insurance claim combinations can indicate fraud
 - Medicine combinations may indicate complications or dangerous interactions
 - Movie recommendations (e.g., Netflix)

Format of a Rule

- Rules are read as “if condition, then result” or “if antecedent, then consequent”

- Antecedent => Consequent

- For example:

{Toy, Wrapping paper} =>
{Batteries}

- This is read as, “If a customer purchases a toy and wrapping paper, then he or she is also likely to buy batteries”



Generating Association Rules

- To find association rules in the dataset

```
association_rules(dataframe, metric="confidence",  
min_threshold=0.6)
```

- **dataframe:** DataFrame of frequent itemsets with columns ['support' , 'itemsets']
 - Generated via the apriori() function
- **metric:** Metric to evaluate if a rule is of interest
 - Supported metrics are 'confidence' and 'lift'
- **min_threshold:** Minimal threshold for the evaluation metric to decide whether a candidate rule is of interest

Inspecting Association Rules

► Association rules produced:

```
In [61]: M association_rules(frequent_itemsets, metric="confidence", min_threshold=0.25)
```

Out[61]:

	antecedants	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(baking powder)	(other vegetables)	0.017692	0.193493	0.007321	0.413793	2.138547	0.003898	1.375807
1	(baking powder)	(whole milk)	0.017692	0.255516	0.009253	0.522989	2.046793	0.004732	1.560725
2	(beef)	(other vegetables)	0.052466	0.193493	0.019725	0.375969	1.943066	0.009574	1.292416
3	(beef)	(rolls/buns)	0.052466	0.183935	0.013625	0.259690	1.411858	0.003975	1.102329
4	(beef)	(root vegetables)	0.052466	0.108998	0.017387	0.331395	3.040367	0.011668	1.332628
5	(beef)	(whole milk)	0.052466	0.255516	0.021251	0.405039	1.585180	0.007845	1.251315
6	(berries)	(other vegetables)	0.033249	0.193493	0.010269	0.308869	1.596280	0.003836	1.166938
7	(berries)	(whipped/sour cream)	0.033249	0.071683	0.009049	0.272171	3.796886	0.006666	1.275461
8	(berries)	(whole milk)	0.033249	0.255516	0.011795	0.354740	1.388328	0.003299	1.153774
9	(berries)	(yogurt)	0.033249	0.139502	0.010574	0.318043	2.279848	0.005936	1.261807
10	(beverages)	(whole milk)	0.026029	0.255516	0.006812	0.261719	1.024275	0.000161	1.008402

Finding Actionable Rules

- ▶ **Thousands of rules are easily generated from a transaction dataset**
- ▶ **In terms of purpose, rules can be thought of as:**
 - Trivial
 - Inexplicable
 - Actionable

Trivial Rules

- ▶ **Many rules derived are *trivial* or useless**
 - They suggest what is common knowledge about the business
 - The effort used to apply sophisticated analytical techniques is wasted
- ▶ **For example:**
 - People who buy hammers also tend to purchase nails
 - Customers who purchase wallpaper also buy wallpaper paste
 - Cereal and milk are purchased together
 - Diapers imply baby formula
 - Laptops imply warranties
- ▶ **Some rules display the results of a deliberate action**
 - People who purchase a fixed telephone line almost always buy broadband access
 - May be the result of a product bundle or marketing program
- ▶ **Where the confidence of a trivial rule is very high, data that does not comply with the trivial rule may indicate quality issues**

Inexplicable Rules

- ▶ ***Inexplicable* rules have no obvious explanation and do not suggest a course of action**
 - Do not give insight into consumer's behavior or do not suggest further action
- ▶ **Example of an inexplicable rule:**

{pickles} => {chocolate ice cream}
- ▶ **When you are applying market basket analysis, many of the results are often either trivial or inexplicable**

Actionable Rules

- ▶ **Actionable rules are easily understood by the business**
 - Lead to insight
- ▶ **Suggest possible causes (when presented to a knowledgeable audience)**
 - May suggest that a product could be more prominently placed
 - Co-occurring items may be separated, ensuring customers must walk through other aisles to retrieve the second item
 - May suggest alternative ways of advertising a product
 - One item may be advertised in a sale, while the price of the other item is raised
 - The customer will likely not notice the higher price or will deem it not worth the trouble to go elsewhere

How are Association Rules Calculated?

- ▶ **Association rules are calculated using three measures**
 - Support
 - Confidence
 - Lift

Support (Frequency) of Items

- ▶ ***Support* measures the proportion of transactions that contain all the items in the set**
- ▶ **Measured as**
 - The number of transactions containing all the items in the set, divided by the total number of transactions (in this example, 7)

t1: Beef, Chicken, Milk

t2: Beef, Cheese

t3: Cheese, Cereal

t4: Beef, Chicken, Cheese

t5: Beef, Chicken, Butter, Cheese, Milk

t6: Chicken, Butter, Milk

t7: Chicken, Milk, Butter

- Support for item set {chicken, butter, milk} = 3/7 = 0.43
 - What is the support for {milk}?
-

Confidence of Rules

- ▶ **Confidence measures how good a rule is at predicting the right-hand side (after the “then” clause) of the rule**
 - Compares how often the right-hand side appears when the condition on the left-hand side (after the “if” clause) of the rule is true
- ▶ **Calculated as the ratio of the number of transactions supporting both sides of the rule to the number of transactions supporting the left side of the rule**
 - The ratio of transactions containing all items to the number of transactions with just the “if” items
- ▶ **What is the confidence really saying?**
 - If a rule—“if beef and chicken, then apples”—has a confidence of 33 percent, it means:
 - When beef and chicken appear in a transaction, there is a 33 percent chance that apples also appear in it
 - That is, one time in three, *apples* occur with *beef* and *chicken*, and the other two times, *beef* and *chicken* appear without *apples*

Calculating Confidence Measures

► **Confidence of the following rule**

$\{\text{butter} \rightarrow \text{milk, chicken}\} = \text{support } \{\text{butter AND milk AND chicken}\} / \text{support } \{\text{butter}\} = (3/7) / (3/7) = 1$

► **What is the confidence of the following rule?**

$\{\text{butter, chicken} \rightarrow \text{milk}\} = \text{support } \{\text{butter AND chicken AND milk}\} / \text{support } \{\text{butter AND chicken}\} = \underline{\hspace{2cm}}$

Inverse Rules

- ▶ **The confidence of inverse rules are not always the same**
- ▶ **For example:**
 - Three of the five transactions that contain *chicken* also contain *beef*, implying a confidence of 60 percent in the rule, “if chicken, then beef”
 - The inverse rule, “if beef, then chicken,” has a higher confidence
 - Of the four transactions with beef, three also have chicken
 - Its confidence is 75 percent

The Lift of a Rule

- ▶ ***Lift* measures how good the rule is by comparing it to randomly guessing the right side of the rule**
 - If items on the right side of a rule are already very common, the rule is not telling us anything
- ▶ **The confidence of the rule is compared to the confidence of the null rule**
 - The null rule has the same item on the right side, but the left side is empty
- ▶ **Calculated as the ratio of the confidence of the rule to the prevalence of the right side of the rule**

Calculating Lift

- ▶ Lift for the following rule in the first dataset is calculated as

$$\begin{aligned}\{\text{chicken} \rightarrow \text{milk}\} &= \text{support } \{\text{chicken AND milk}\} / (\text{support } \{\text{chicken}\} * \text{support } \{\text{milk}\}) \\ &= (4/7) / [(5/7) * (4/7)] = 1.4\end{aligned}$$

- ▶ Lift measures how much better the rule is than guessing

- Lift > 1
 - The rule is better at predicting the result than guessing whether the right side of the rule is present based on item frequencies in the data
- Lift ≤ 1
 - The rule is doing as well as or worse than guessing

- ▶ The rule above shows an improvement over just guessing

- When chicken is purchased, milk is 40 percent more likely to be in the transaction than if chicken is not purchased
- Sometimes, the best rule contains fewer items than other rules under consideration

The Negative Rule

- ▶ When lift is less than 1, negating the result produces a better rule
 - ▶ If the rule:
 - “if apples and beef, then chicken”
 - has a confidence of 20 percent, then the rule:
 - “if apples and beef, then NOT chicken”
 - has a confidence of 80 percent
- ▶ Because chicken appears in 40 percent of the transactions, it does not appear in 60 percent of them
 - ▶ Applying the same lift measure shows that the lift of this new rule is 1.33 ($0.80/0.60$)

Trial and Error

- ▶ It will require a certain amount of experimentation to generate good rules
- ▶ Different required levels of support and confidence will generate several or no rules
- ▶ Best choice for minimum support depends on the data and the situation
 - Minimum support may be varied as the algorithm progresses
 - By *decreasing* the support level, less common combinations of common items may be found
 - By *increasing* the support level, common combinations of uncommon items may be found

Sorting Rules

- To find actionable rules, it can be useful to sort the rules by the evaluation criteria (support, confidence, lift)

```
rules.sort_values(by=['lift'], ascending=False).head

<bound method NDFrame.head of
   antecedents      consequents  antecedent support \
2  (other vegetables)  (root vegetables)  0.193493
3  (root vegetables)  (other vegetables)  0.108998
10 (whole milk)       (root vegetables)  0.255516
11 (root vegetables)  (whole milk)       0.108998
7   (yogurt)          (other vegetables)  0.139502

   consequent support  support  confidence      lift  leverage  conviction
2        0.108998  0.047382  0.244877  2.246605  0.026291  1.179941
3        0.193493  0.047382  0.434701  2.246605  0.026291  1.426693
10       0.108998  0.048907  0.191405  1.756031  0.021056  1.101913
11       0.255516  0.048907  0.448694  1.756031  0.021056  1.350401
7        0.193493  0.043416  0.311224  1.608457  0.016424  1.170929
```

Subsets of Association Rules

► Specific items involved in associations may also be queried

- Perhaps to run a particular marketing promotion
- Note that the entries on the itemset side of the filter (i.e., {chocolate} in this example) are of type frozenset, which is a built-in Python type that is similar to a Python set, but immutable (can't be modified)

```
rules[rules['antecedants'] == {'chocolate'}]
```

	antecedants	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
99	(chocolate)	(other vegetables)	0.049619	0.193493	0.012710	0.256148	1.323810	0.003109	1.084230
100	(chocolate)	(rolls/buns)	0.049619	0.183935	0.011795	0.237705	1.292332	0.002668	1.070537
102	(chocolate)	(soda)	0.049619	0.174377	0.013523	0.272541	1.562939	0.004871	1.134941
104	(chocolate)	(whole milk)	0.049619	0.255516	0.016675	0.336066	1.315243	0.003997	1.121322

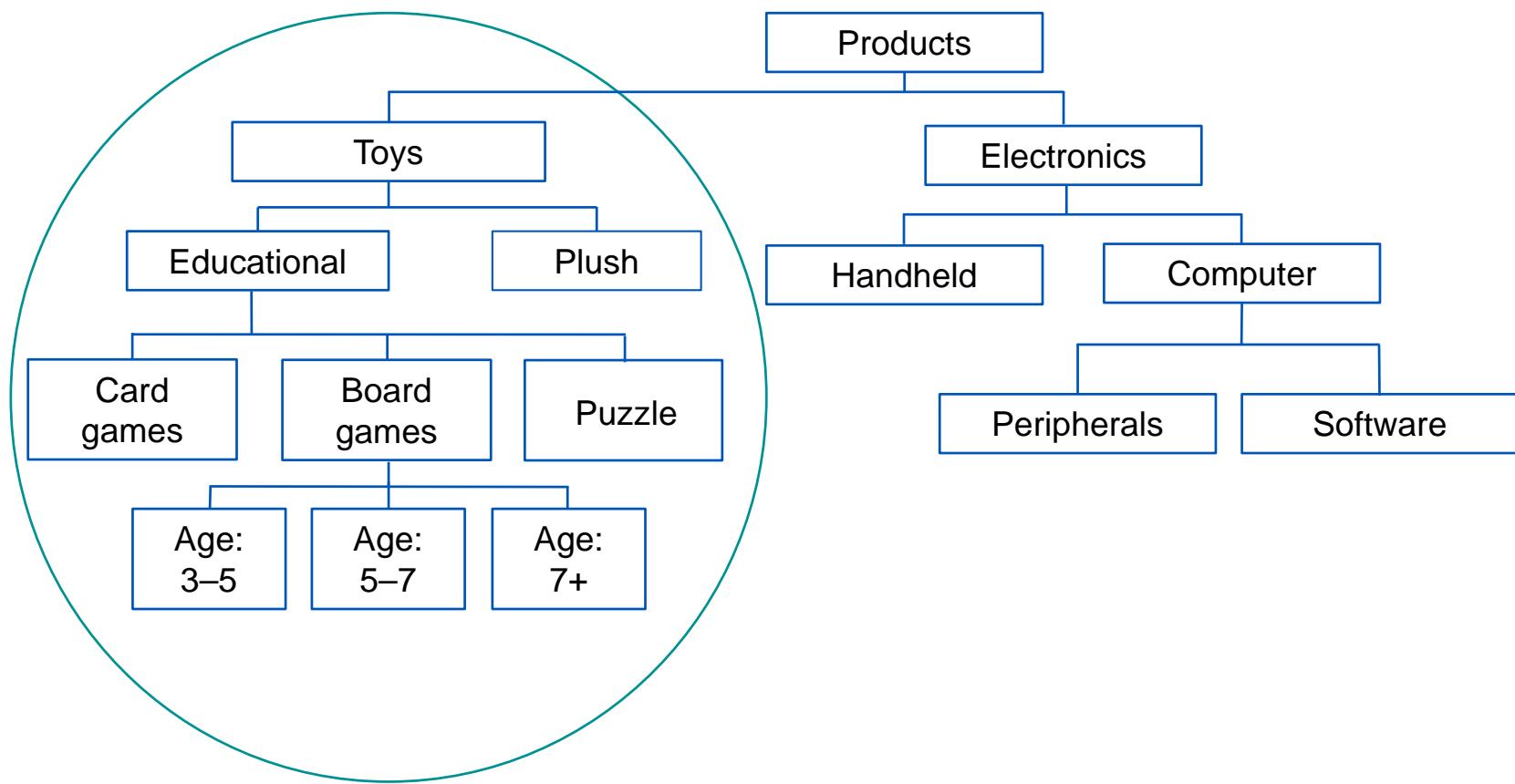
Choosing the Correct Item Set

- ▶ **Choosing the right level of detail needs to be carefully considered**
 - How useful the resulting rules are depends crucially on the items chosen for analysis
- ▶ **What constitutes a particular item depends on the business need**
- ▶ **Within a retail store, there may be tens of thousands of products for sale**
 - A laptop might be considered an item for analysis purposes
 - On the other hand, the manager may be very interested in the particular options that are chosen when the laptop is ordered
 - The laptop may be broken down into its constituent parts to use as items
 - RAM size
 - Hard drive size
 - Number of processors, etc.
 - All these items need to map up to the “laptop” item for analysis
- ▶ **The component items may vary over time**
 - Change of purchase options can pose problems when using historical data

Generalizing Items to Create Rules

- ▶ Generalizing items into product hierarchies can lead to more actionable rules
- ▶ This brings up questions such as
 - What level of the product hierarchy is the right one to use?
 - Are chunky chips and thin-cut chips the same product?
 - Will taking the size or color of clothing into account lead to useful rules?

An Example of a Product Taxonomy



Using Virtual Items to Generate Rules

- ▶ **Items for analysis may be augmented by using virtual items**
 - Brands such as Nike may appear in several departments
 - Low-fat options may come out in a grocery store
 - May include information, such as when a transaction occurred
- ▶ **Virtual items may be included if they would lead to actionable rules**
- ▶ **Demographic data may be included through virtual items to further augment rules**



Contents

- ▶ Exploring a Transaction Dataset
- ▶ Mining Association Rules From Transaction Data

Hands-On Exercise 8.1

- ▶ Building Recommenders From Transaction Data
- ▶ Hands-On Exercise 8.2





Hands-On Exercise 8.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 8.1: Generating Association Rules From Transaction Data

Contents

- ▶ Exploring a Transaction Dataset
- ▶ Mining Association Rules From Transaction Data
- ▶ Hands-On Exercise 8.1

Building Recommenders From Transaction Data

- ▶ Hands-On Exercise 8.2



Recommender Systems

- ▶ **Recommender systems are used to give personalized recommendations (to people) on music, films, books, and news**
 - Two-thirds of the movies on Netflix are recommended
 - Thirty-five percent of Amazon's sales come from recommendations
 - On Google News, recommendations generate 38 percent more click-through
 - Attempts to predict a user's preference for an item is based on past ratings on other items
- ▶ **Recommenders are typically based on either**
 - Collaborative filtering
 - Uses a “wisdom of crowds” approach
 - Finds users with tastes similar to the target user (based on past ratings), making recommendations based on the opinions of those similar users
 - User ratings can be collected either implicitly or explicitly
 - Content-based filtering
 - Measures the similarity of items to the ones that the user likes or dislikes based on item attributes (e.g., Pandora uses the Music Genome Project to compare attributes of songs played on the station)

Business Applications for Recommendation Framework

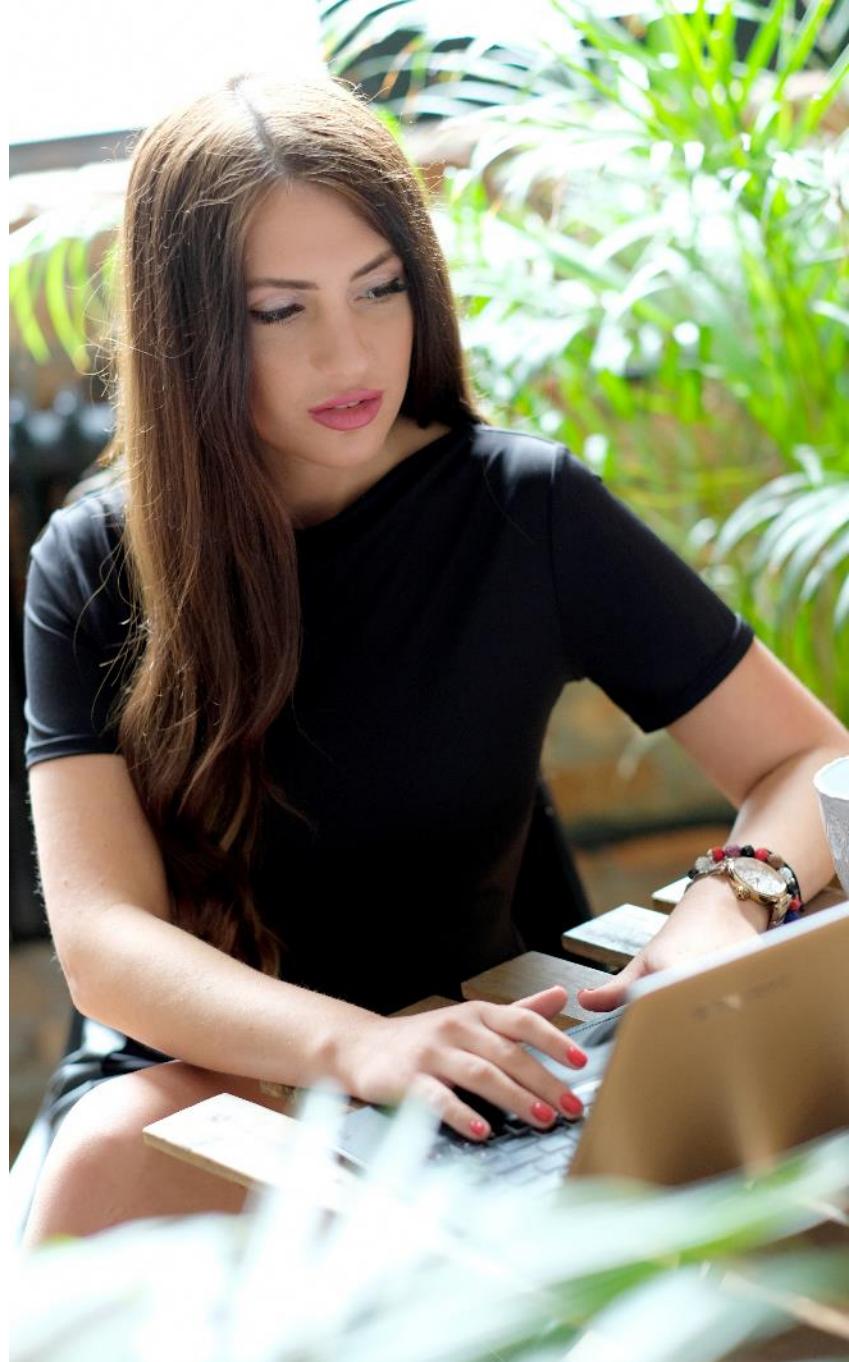
► Marketing

- For identifying users who may switch cell phone companies
- For recommending books on Amazon
- For recommending products on websites

► Education

- Universities guiding students on which courses to take
- Conference organizers assigning papers to reviewers

► What are some possible applications of recommendation frameworks in your workplace?



Similarity Measures

- ▶ A number of measures may be used to determine similarity
 - Pearson correlation
 - Euclidean distance
 - Cosine
 - Co-occurrence
- ▶ In our example, we will use a co-occurrence matrix

Co-occurrence Matrix: Explanation

- **The weight in the similarity matrix indicates the frequency of co-occurrence/match of items**
 - This matrix represents item similarity
 - Higher value represents higher similarity

	Item 1	Item 2	Item 3	Item 4	Item 5
Item 1	10	5	3	2	1
Item 2	5	10	6	5	3
Item 3	3	6	10	1	5
Item 4	2	5	1	10	3
Item 5	1	3	5	3	10

User Preference Matrix

► **User preference is a single-column matrix**

- Each entry represents a preference for an item
- The value of the first row means that the preference weight for Item 1 is 4
- The value of the second row means that there is no preference for Item 2

Item 1	4
Item 2	0
Item 3	0
Item 4	5
Item 5	0

Co-occurrence Matrix

- To make recommendations, we can multiply the matrices
 - Users are more likely to be interested in an item that co-occurs frequently with an item that they gave a high rating to

$$\text{Matrix}[S] \times \text{Matrix}[U] = \text{Matrix}[R]$$

→

10	5	3	2	1
5	10	6	5	3
3	6	10	1	5
2	5	1	10	3
2	3	5	3	10

Similarity Matrix S

↓

4
0
0
5
0

User Preference
Matrix U

$$10*4 + 5*0 + 3*0 + 2*5 + 1*0 = 50$$

50

50
45
17
58
23

Recommendation
Matrix R

Recommendation Matrix

- The recommendation is computed as $S \times U = R$
- The user already has provided preference for Item 1 and Item 4
- What other items can be recommended?
 - First recommendation is for Item 2
 - Second recommendation is for Item 5
 - Third recommendation is for Item 3

Item 1	50
Item 2	45
Item 3	17
Item 4	58
Item 5	23

Building a Recommender in Python

```
df = pd.read_csv('trans.csv')
```

Reads in the transaction's data set

```
customers = df['user'].unique()
```

Establishes the unique users

```
items = df['item'].unique()
```

Establishes the unique products

```
df['customers'] = df['user'].apply(lambda x : np.argwhere(customers == x)[0][0])
```

Establishes an index for the unique users

```
df['items'] = df['item'].apply(lambda x : np.argwhere(items == x)[0][0])
```

Establishes an index for the unique products

```
df
```

	user	item	pref	customers	items
0	1	101	5.0	0	0
1	1	102	3.0	0	1
2	1	103	2.5	0	2
3	2	101	2.0	1	0
4	2	102	2.5	1	1
5	2	103	5.0	1	2
6	2	104	2.0	1	3
7	3	101	2.0	2	0

Building a Recommender in Python

```
from scipy.sparse import csr_matrix
```

Import a compressed sparse row (csr) matrix function

```
occurrences = csr_matrix((customers.shape[0], items.shape[0]), dtype='int8')  
occurrences.toarray()
```

```
array([[0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 0, 0]])
```

Create a sparse matrix of the appropriate size

```
def set_occurrences(customer, item):  
    occurrences[customer, item] += 1
```

```
df.apply(lambda row: set_occurrences(int(row['customers']), int(row['items'])), axis=1)
```

```
cooc = occurrences.transpose().dot(occurrences)  
cooc.toarray()
```

```
array([[5, 3, 4, 4, 2, 1, 2],  
       [3, 3, 3, 2, 1, 0, 1],  
       [4, 3, 4, 3, 1, 0, 2],  
       [4, 2, 3, 4, 2, 1, 2],  
       [2, 1, 1, 2, 2, 1, 1],  
       [1, 0, 0, 1, 1, 1, 0],  
       [2, 1, 2, 2, 1, 0, 2]], dtype=int8)
```

Using a lambda function, populate the sparse matrix with occurrences

Create the co-occurrence matrix

Building a Recommender in Python

```
prefs = csr_matrix((items.shape[0], customers.shape[0]))
```

Create a sparse matrix
of the appropriate size

```
def set_prefs(item, customer, pref):  
    prefs[item, customer] = pref
```

```
df.apply(lambda row: set_prefs(int(row['items']), int(row['customers']), row['pref']), axis=1)
```

```
recommendations = cooc.dot(prefs)
```

```
rec = recommendations.toarray()  
rec
```

```
array([[44. , 45.5, 40. , 63. , 68. ],  
       [31.5, 32.5, 18.5, 37. , 42.5],  
       [39. , 41.5, 24.5, 53.5, 56.5],  
       [33.5, 36. , 38. , 55. , 59. ],  
       [15.5, 15.5, 26. , 26. , 32. ],  
       [ 5. ,  4. , 15.5,  9.5, 11.5],  
       [18. , 20.5, 16.5, 33. , 34.5]])
```

Using a lambda
function, populate the
sparse matrix

Multiply the user's preferences matrix by the
co-occurrence matrix

Building a Recommender in Python

```
for c in customers :  
    print("Customer :", c-1, "Recommendation Scores :", rec[:,c-1])
```

```
Customer : 0 Recommendation Scores : [44. 31.5 39. 33.5 15.5 5. 18. ]  
Customer : 1 Recommendation Scores : [45.5 32.5 41.5 36. 15.5 4. 20.5]  
Customer : 2 Recommendation Scores : [40. 18.5 24.5 38. 26. 15.5 16.5]  
Customer : 3 Recommendation Scores : [63. 37. 53.5 55. 26. 9.5 33. ]  
Customer : 4 Recommendation Scores : [68. 42.5 56.5 59. 32. 11.5 34.5]
```

View recommendation values for each customer



Weaknesses in Recommender Systems

- ▶ **Content-based methods require explicit item descriptions to measure similarity**
 - Might be difficult to obtain for items like ideas or opinions
- ▶ **Collaborative filtering suffers from two problems**
 - Data sparsity problem
 - Across an enormous number of items, a user will have rated only a few items, resulting in a very sparse user/item rating matrix
 - Limited number of reviews makes it difficult for recommender systems to accurately measure user similarities
 - Cold-start problem
 - When a user initially joins, the system has only a few (if any) reviews from the user, making it difficult for the system to accurately interpret the user's preference
- ▶ **Social influence can play an important role in recommenders**
 - Social relations can be used as an influencing factor on a user's preferences
 - Friends have a tendency to purchase similar items as well as give similar ratings

Contents

- ▶ Exploring a Transaction Dataset
- ▶ Mining Association Rules From Transaction Data
- ▶ Hands-On Exercise 8.1
- ▶ Building Recommenders From Transaction Data

Hands-On Exercise 8.2



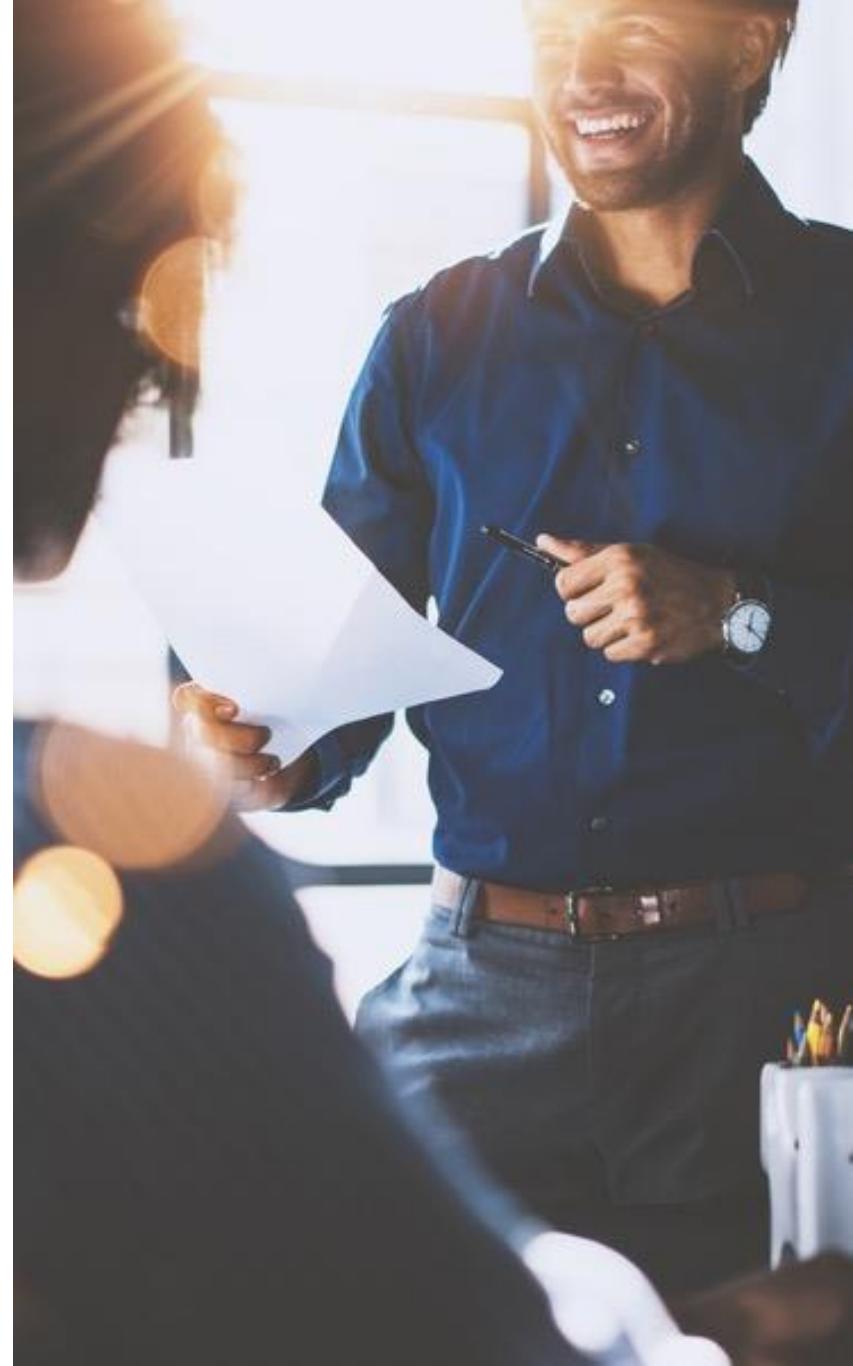


Hands-On Exercise 8.2

In your Exercise Manual, please refer to
**Hands-On Exercise 8.2: Building
Recommenders With Transaction Data**

Objectives

- ▶ **Generate association rules from transaction data**
- ▶ **Evaluate rules using measures of support, confidence, and lift**
- ▶ **Examine approaches toward improving rules**



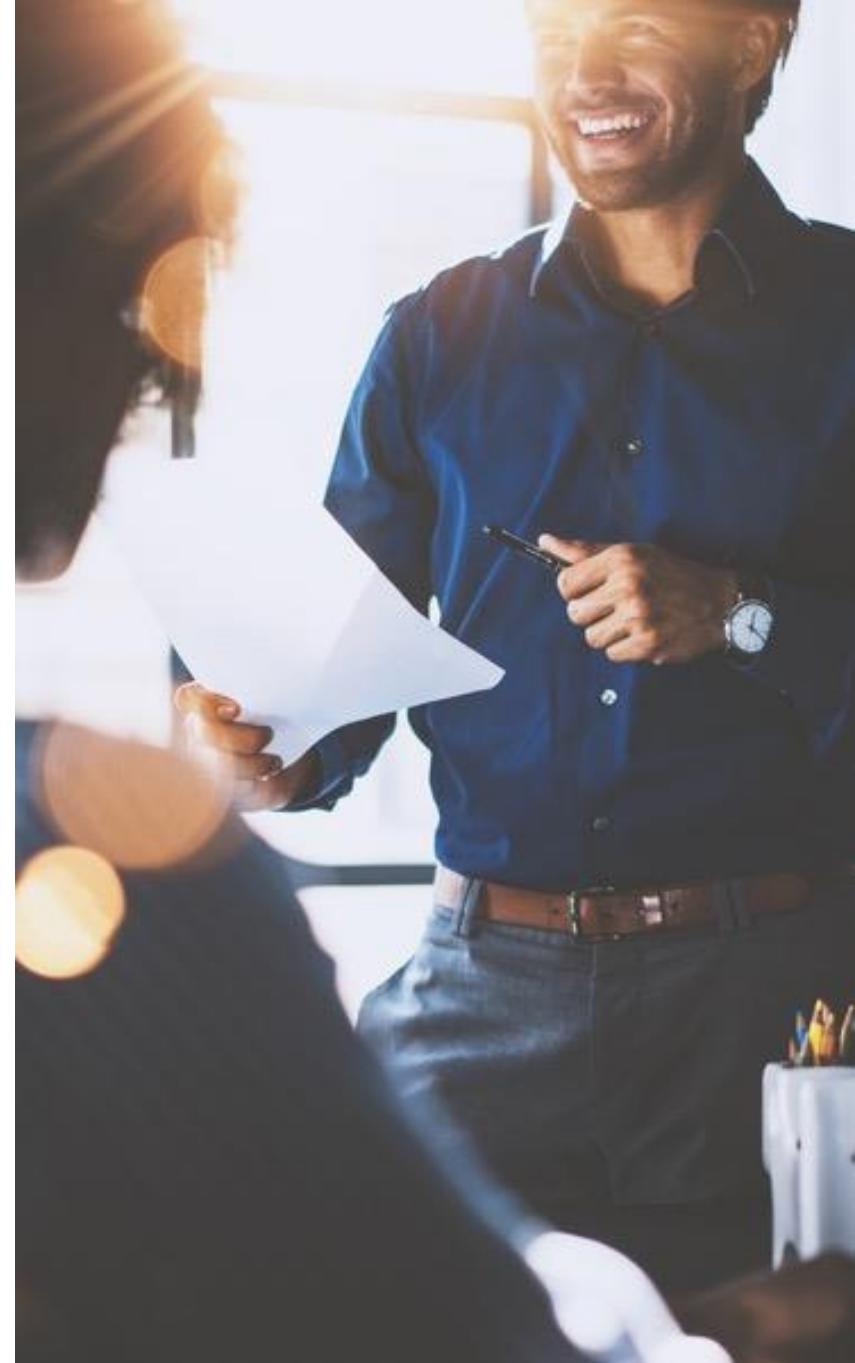
Chapter 9

Additional Techniques:

Social Network Analysis

Objectives

- ▶ **Analyze data that has a network structure**
- ▶ **Plot networks to visualize relationships**
- ▶ **Explore egocentric and sociocentric methods of analyzing networks**



Contents

Introduction to Networks

- ▶ Representing Networks as Data
- ▶ Visualizing Network Relationships
- ▶ Hands-On Exercise 9.1
- ▶ Egocentric Analysis
- ▶ Sociocentric Analysis
- ▶ Hands-On Exercise 9.2



Introduction to Network Analysis

- ▶ **Originates from the field of Graph Theory**
 - Provides the conceptual tools and principles for studying the characteristics of any type of graph (e.g., biological, technological, social)
- ▶ **A node can represent a person, an object/entity, or even a concept**
- ▶ **Helps us to uncover patterns in interactions and relationships for improving, optimizing, or negating the effects of those networks**



Introduction to Network Analysis

► Networks are everywhere

- Facebook/Twitter
- Disease outbreaks
- Page ranking on Google
- Financial networks



Applications of Network Analysis

- ▶ **Social graphs can give unique perspectives on how *relationships* can be used to explain *individual* behavior**
 - The actions of an individual are often a function of their position in a social network
 - Concentrating on identifying their position as opposed to their job function can often yield interesting and surprising results
 - A network might tell you who is influential
 - In an investment network, a first measure of influence might be to simply look at the number of connections of each investor, to tell you who is the most connected investor in the start-up realm

Applications of Network Analysis

► **Network analysis can be used to**

- Recommend people (e.g., friends of friends) you might know on social networking sites
- Improve the dissemination of information in a media campaign (e.g., political messages, health campaigns)
- Uncover potential conflicts of interest (e.g., in government and lobby groups)
- Identify criminal networks and key players in those networks
- Identify potential causes of social behavior and promote cohesion (e.g., dysfunctional communities)

Contents

- ▶ Introduction to Networks

Representing Networks as Data

- ▶ Visualizing Network Relationships
- ▶ Hands-On Exercise 9.1
- ▶ Egocentric Analysis
- ▶ Sociocentric Analysis
- ▶ Hands-On Exercise 9.2



Networks Represented as Nodes and Edges

- ▶ **Nodes**

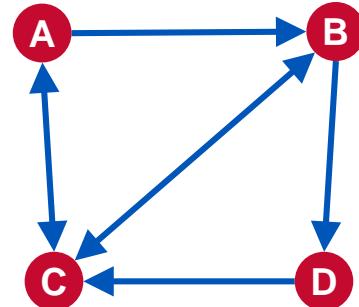
- Can represent people or items of interest in the network

- ▶ **An edge might represent**

- A business transaction
 - A friendship or working relationship
 - An exchange of information or goods

- ▶ **Edges can be described by their strength using a weight attribute**

- The strength of an edge might be related to:
 - The frequency of the interaction
 - A quantity related to the interaction
 - For example, the amount of goods being exchanged
 - An attribute of the relationship
 - For example, the length of the acquaintance



Creating Graphs Manually

► Load the networkx package

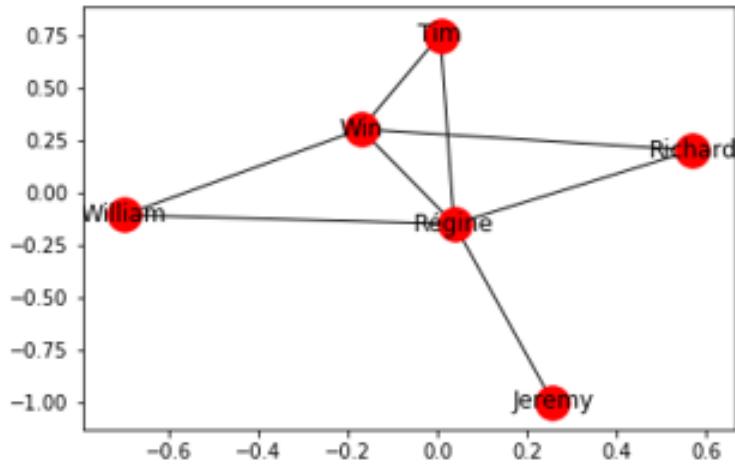
```
import networkx as nx
```

► Graphs can be manually created :

- One edge at time using the `add_edge()` method or
- From a list of pairs using the `add_edges_from()` method

► Create a graph one edge at a time

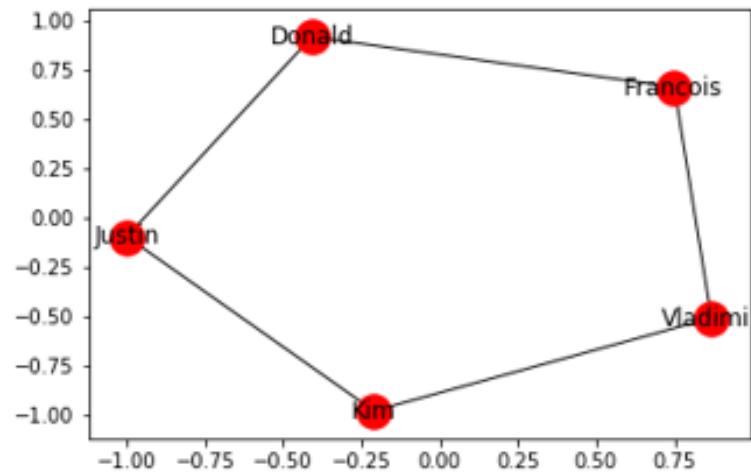
```
G_group = nx.Graph()
G_group.add_edge('Win', 'Régine')
G_group.add_edge('Win', 'William')
G_group.add_edge('Win', 'Richard')
G_group.add_edge('Win', 'Tim')
G_group.add_edge('Régine', 'William')
G_group.add_edge('Régine', 'Richard')
G_group.add_edge('Régine', 'Tim')
G_group.add_edge('Régine', 'Jeremy')
```



Creating Graphs Manually

- ▶ Create a graph from a list of pairs

```
G_names = nx.Graph()  
  
G_names.add_edges_from(  
    [("Donald", "Justin"),  
     ("Donald", "Francois"),  
     ("Vladimir", "Francois"),  
     ("Kim", "Justin"),  
     ("Vladimir", "Kim")]  
)  
  
nx.draw_networkx(G_names)
```



Creating Graphs From Data Files

- ▶ **Read data from a csv files into a graph**

```
G_media = nx.read_edgelist('SNADataset1_links.csv',
                           delimiter = ',',
                           create_using = nx.Graph(),
                           nodetype=str,
                           data=[('type',str),('weight',float)])
nx.draw_networkx(G_media)
```

- ▶ **Assign attribute data from a pandas DataFrame to the nodes**

- Attribute data is passed as a dictionary

```
import pandas as pd
nodeData = pd.read_csv('SNADataset1_nodes.csv', index_col=0)
nx.set_node_attributes(G_media, nodeData.to_dict('index'))
```

- ▶ **Existing graphs saved as .gml files can be imported as follows:**

```
mygraph=nx.read_gml("lesmis.gml")
nx.draw_networkx(mygraph)
```

.gml = graph modelling language

Viewing Nodes, Edges and Attributes of a Graph

► **View edges and nodes of a graph object**

`G.nodes()`

`G.edges()`

► **Graphs, edges, and vertices can have attributes which can be used in network analysis**

- To display an attribute of an edge

`G.edge[(1, 2)]`

- To display an attribute of a node

`G.node[(1)]`

- To view an attribute of all edges in the graph

`nx.get_edge_attributes(G_media, 'type')`

Contents

- ▶ Introduction to Networks
- ▶ Representing Networks as Data

Visualizing Network Relationships

- ▶ Hands-On Exercise 9.1
- ▶ Egocentric Analysis
- ▶ Sociocentric Analysis
- ▶ Hands-On Exercise 9.2



Visualizing Network Relationships

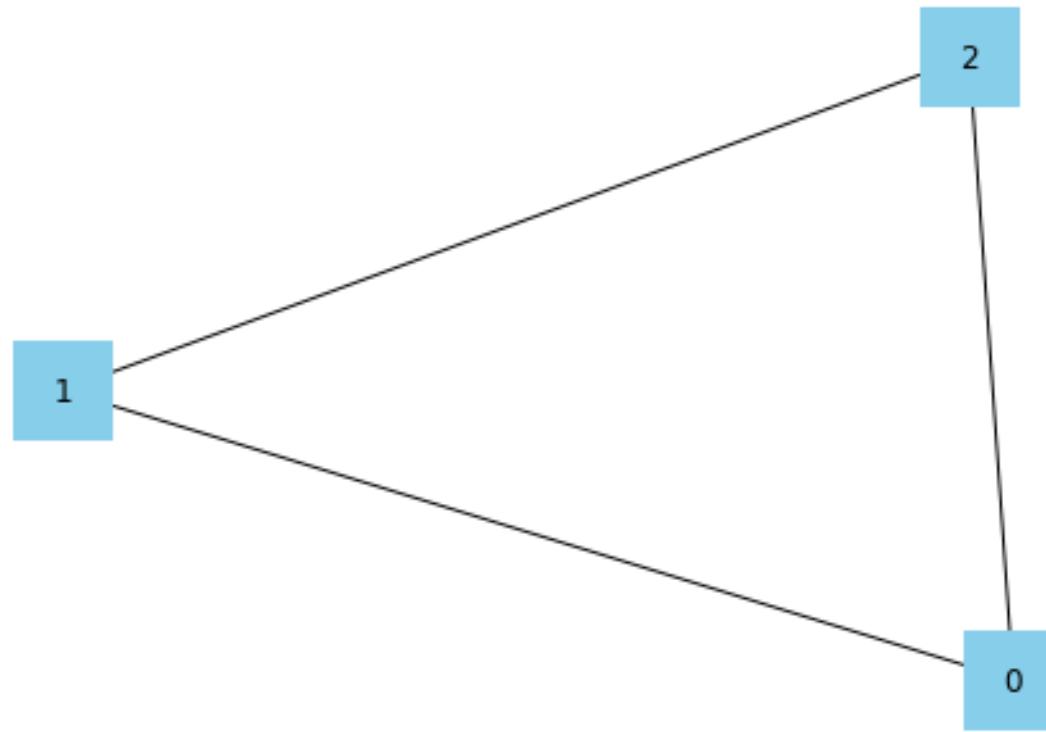
- The ability to plot different components of a graph visually according to their attributes helps in analyzing those networks



Plotting Graphs

- Nodes on a graph can be colored, shaped and sized

```
In [111]: nx.draw(G, with_labels=True, node_size=1500, node_color="skyblue", node_shape="s")
```

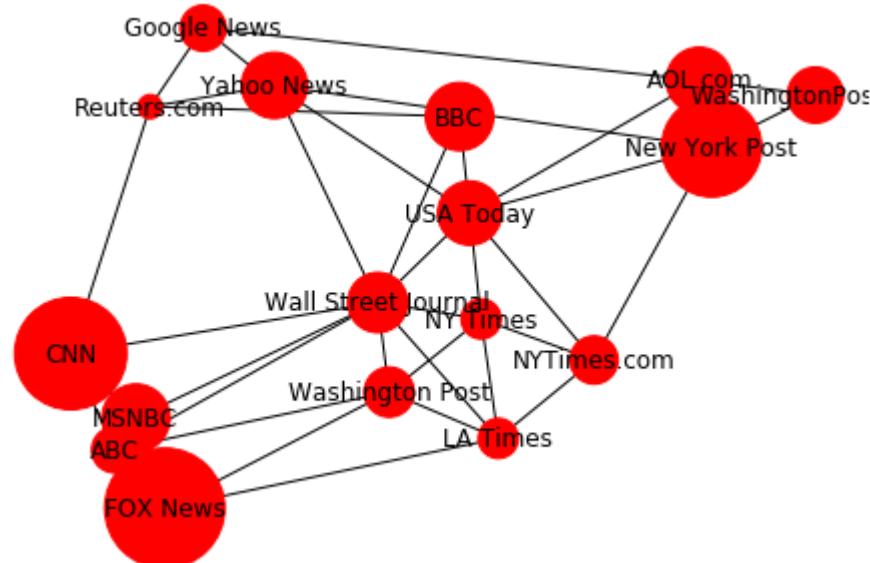


Plotting Node Sizes According to an Attribute Value

- A node's size can be based on an attribute value to draw focus
 - Size the nodes according to the size of the audience of each media outlet

```
node_size = []
for node in G_media.nodes:
    node_size.append(G_media.nodes[node]['audience.size']**2)

node_labels = nx.get_node_attributes(G_media,'media')
nx.draw(G_media,node_size = node_size, labels = node_labels)
```



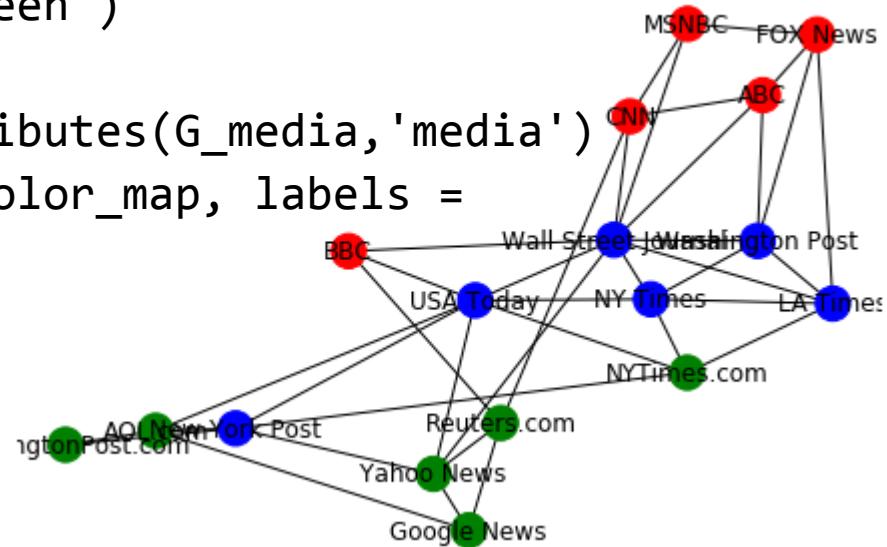
Plotting Node Colors According to an Attribute Value

► To set node color based on an attribute value:

- Color the nodes according to the type of media outlet

```
color_map = []
for node in G_media.nodes:
    if G_media.nodes[node]['media.type'] == 1:
        color_map.append('blue')
    elif G_media.nodes[node]['media.type'] == 2:
        color_map.append('red')
    else: color_map.append('green')
```

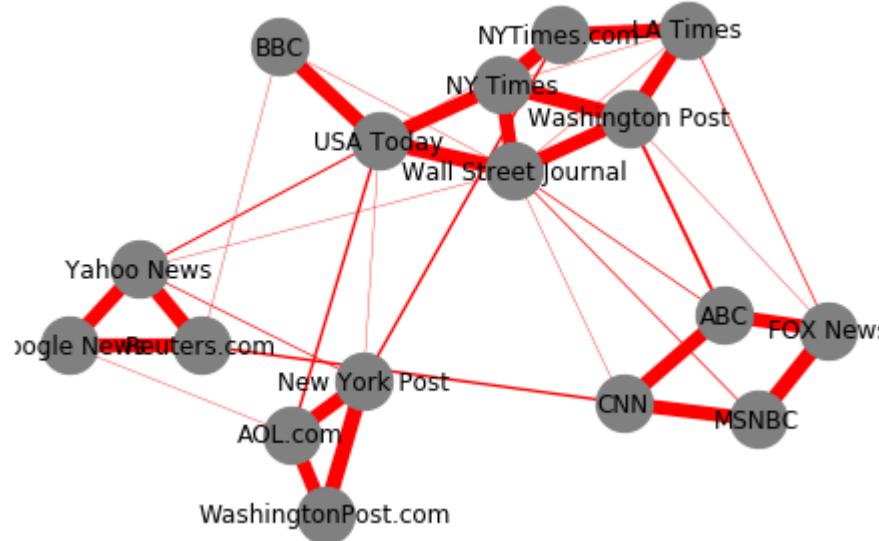
```
node_labels = nx.get_node_attributes(G_media,'media')
nx.draw(G_media,node_color = color_map, labels =
node_labels)
```



Plotting Edge Width According to an Attribute Value

- To set edge width based on an attribute value:

```
edge_weights = []
for edge in G_media.edges:
    edge_weights.append(G_media.edges[edge]['weight']/3)
edge_weights
nx.draw(G_media, node_size = 800, node_color = 'gray', labels =
node_labels, edge_color = 'red', width = edge_weights)
```



Arranging Graph Layouts

- A number of layouts are available that allow the arrangement of nodes to your liking

```
# Fruchterman Reingold
nx.draw(G_symmetric, with_labels=True, node_size=1500, node_color="skyblue",
pos=nx.fruchterman_reingold_layout(G_symmetric))

# Circular
nx.draw(G_symmetric, with_labels=True, node_size=1500, node_color="skyblue",
pos=nx.circular_layout(G_symmetric))

# Random
nx.draw(G_symmetric, with_labels=True, node_size=1500, node_color="skyblue",
pos=nx.random_layout(G_symmetric))

# Spectral
nx.draw(G_symmetric, with_labels=True, node_size=1500, node_color="skyblue",
pos=nx.spectral_layout(G_symmetric))

# Spring
nx.draw(G_symmetric, with_labels=True, node_size=1500, node_color="skyblue",
pos=nx.spring_layout(G_symmetric))
```

Contents

- ▶ Introduction to Networks
- ▶ Representing Networks as Data
- ▶ Visualizing Network Relationships

Hands-On Exercise 9.1

- ▶ Egocentric Analysis
- ▶ Sociocentric Analysis
- ▶ Hands-On Exercise 9.2





Hands-On Exercise 9.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 9.1: Creating and Visualizing Network Graphs

Contents

- ▶ Introduction to Networks
- ▶ Representing Networks as Data
- ▶ Visualizing Network Relationships
- ▶ Hands-On Exercise 9.1

Egocentric Analysis

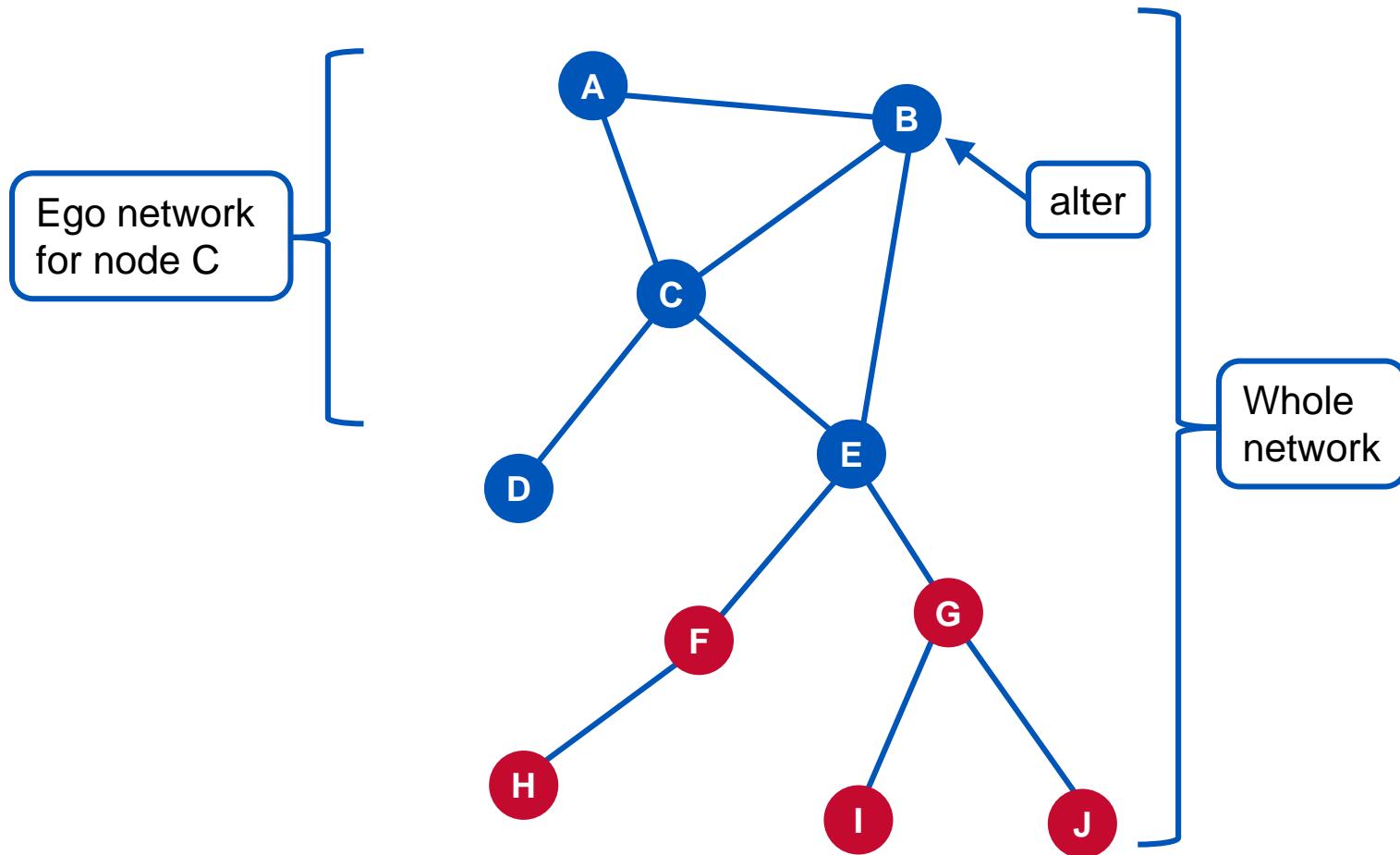
- ▶ Sociocentric Analysis
- ▶ Hands-On Exercise 9.2



Analyzing Social Networks

- ▶ **Social networks can be studied in two ways**
 - Egocentric analysis
 - Focuses on the effects a personal network has on an individual node by looking at the:
 - Number of nodes (alters) this node is connected to
 - Types of connections to those nodes
 - Can be used to predict health, levels of depression, access to economic opportunities
 - Sociocentric analysis
 - Focuses on the larger network to investigate patterns of interactions across the group as a whole
 - Investigates how network interactions can affect
 - How a disease spreads
 - Access to resources and new ideas
 - Group dynamics

Ego Networks



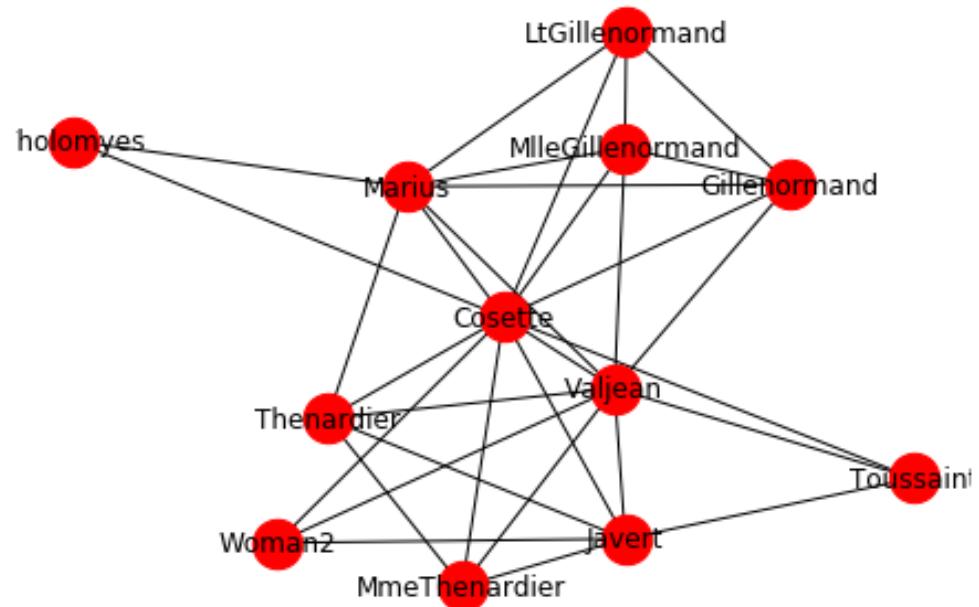
Visualization of an Ego Network

- To display the ego network of an individual node:

```
ego_g = nx.ego_graph(les_mis, 'Cosette')
```

```
pos = nx.spring_layout(ego_g)
```

```
nx.draw(ego_g, pos, node_color='red', node_size=500,  
        with_labels = True)
```

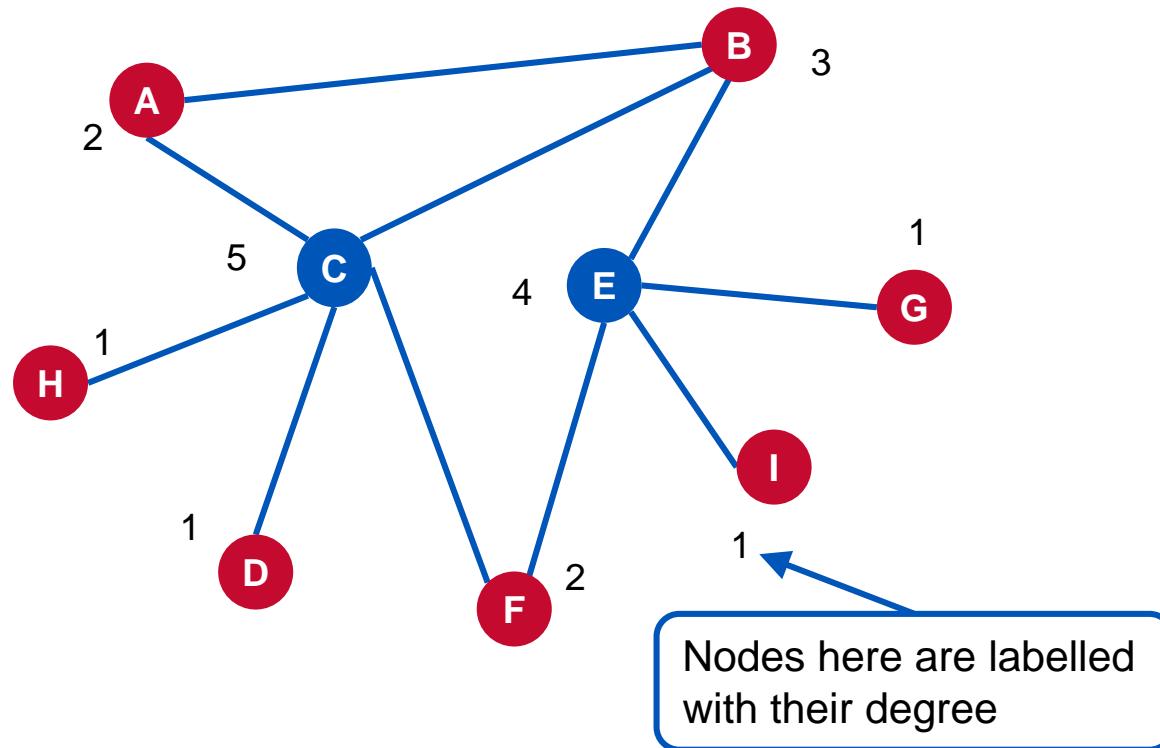


Centrality Measures

- ▶ **Centrality measures can help identify key (central) nodes of importance in the network that may be worth doing an ego analysis on**
- ▶ **There are a number of measures of centrality**
 - Degree
 - Betweenness
 - Closeness
 - Eigenvector

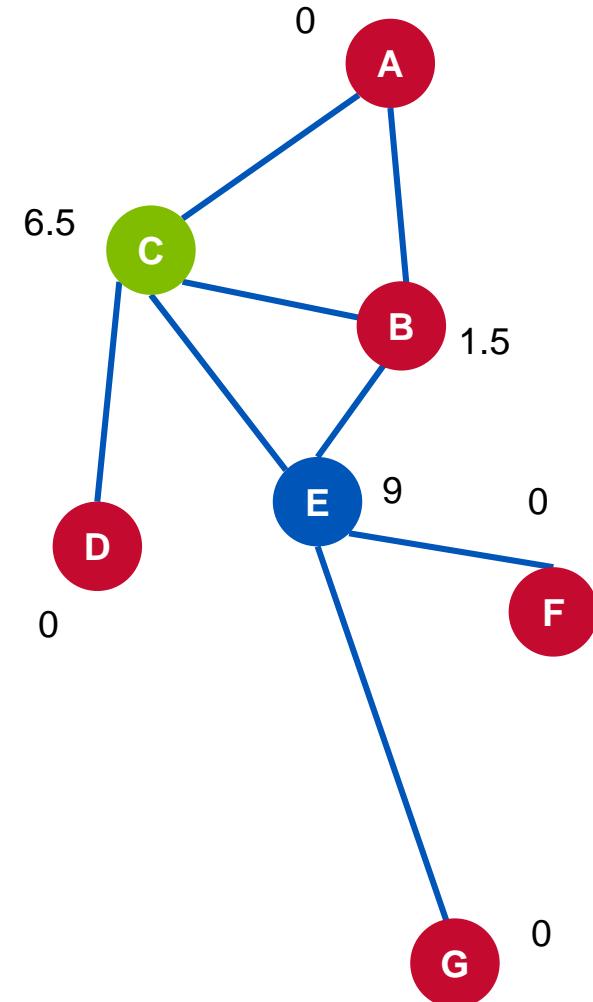
Degree

- A node's degree refers to the number of links in or links out from that node
 - Often used as a measure of popularity that an individual might have
 - Useful in planning how to spread a message quickly across a network
- Syntax:
`d = nx.degree_centrality(les_mis)`



Betweenness

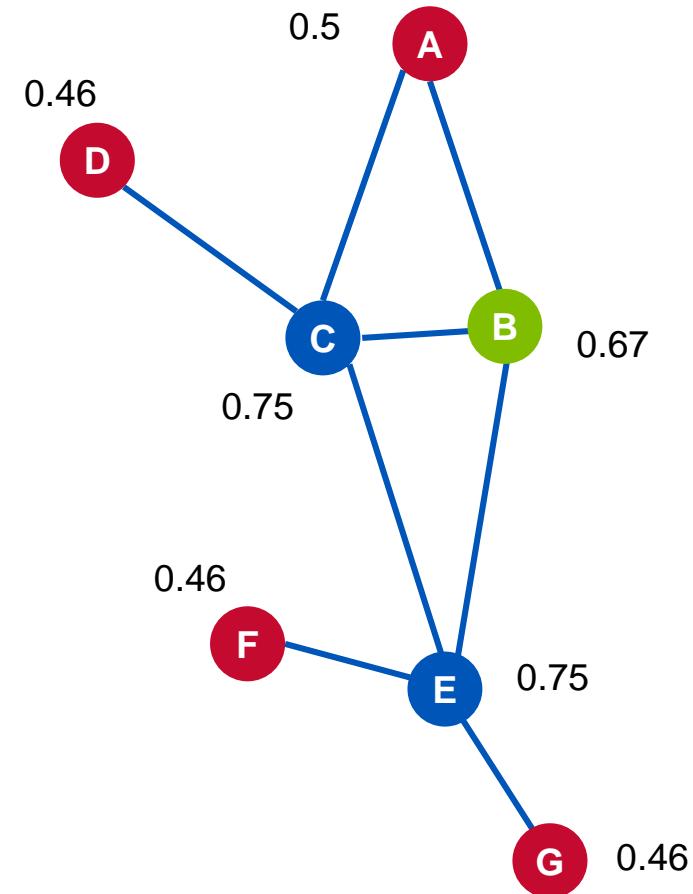
- ▶ Indicates which nodes are more likely to be in communication paths with others
 - Often used to identify how likely a person is to be the direct route between two people
- ▶ Can also be used to identify who is the person through whom most information flows, or the point where the network would fall apart, if lost
- ▶ Syntax:
`d = nx.betweenness_centrality(les_mis)`



Closeness

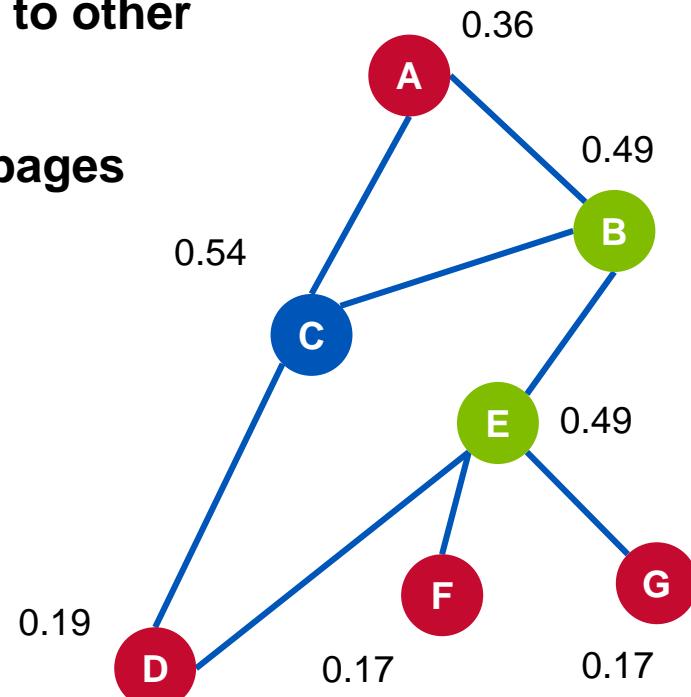
- ▶ **Closeness is a measure of a node's reach**
 - How quickly information could pass to all other nodes in the network from a given starting point
- ▶ **It can be useful in measuring how quickly a disease can spread from one infected person**
- ▶ **Syntax:**

```
d = nx.closeness_centrality(les_mis)
```



Eigenvector

- ▶ A measure of how well a node is connected to other well-connected nodes
- ▶ Google uses this approach in how it ranks pages
 - If a webpage is linked to (from a highly linked-to page), it gets a higher ranking
- ▶ Could be used in establishing experts in an academic field
 - Example: which authors are cited by other well-cited authors
- ▶ Syntax:
`d = nx.eigenvector_centrality(les_mis)`
- ▶ To plot the preceding graphs according to their centrality measures:
`nx.draw(les_mis, with_labels = True, nodelist=d.keys(),
node_size=[v * 500 for v in d.values()])`



Centrality Measures

Do Now

- **What measures would you use to get the following information about an individual within a network?**
- How quickly would this person hear “gossip” in the network (i.e., How close are they to all other members)?

 - How influential is this person in the network (i.e., How well connected are they)?

 - How quickly could this person catch a virus which is flowing through the network?

 - How good would this person be to act as a “bridge” or “gatekeeper” in the network (i.e., To what extent do they lie between other nodes in the network)?

Contents

- ▶ Introduction to Networks
- ▶ Representing Networks as Data
- ▶ Visualizing Network Relationships
- ▶ Hands-On Exercise 9.1
- ▶ Egocentric Analysis

Sociocentric Analysis

- ▶ Hands-On Exercise 9.2

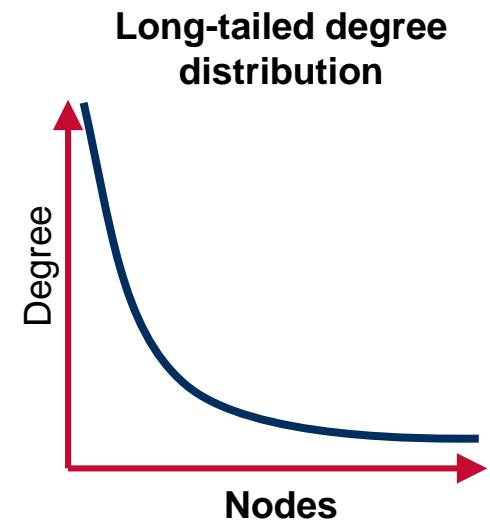


Sociocentric Analysis

- ▶ **Many types of sociocentric questions can be asked of a network:**
 - Preferential Attachment
 - Has attaching to certain nodes (people/ideas) become popular or fashionable?
 - Node combinations
 - Are there powerful combinations of nodes in the network?
 - Homophily
 - Is there a danger of group-think due to people only connecting with like-minded others?
 - Cliques
 - Have cliques formed in the network? Who are the bridges?
 - Clustering
 - Is everybody talking to everybody else or are some relationships one-way or via third parties?
 - Centralization
 - How centralized or decentralized is our network?
 - Hubs/Authorities
 - Where are the hubs or authorities of knowledge?

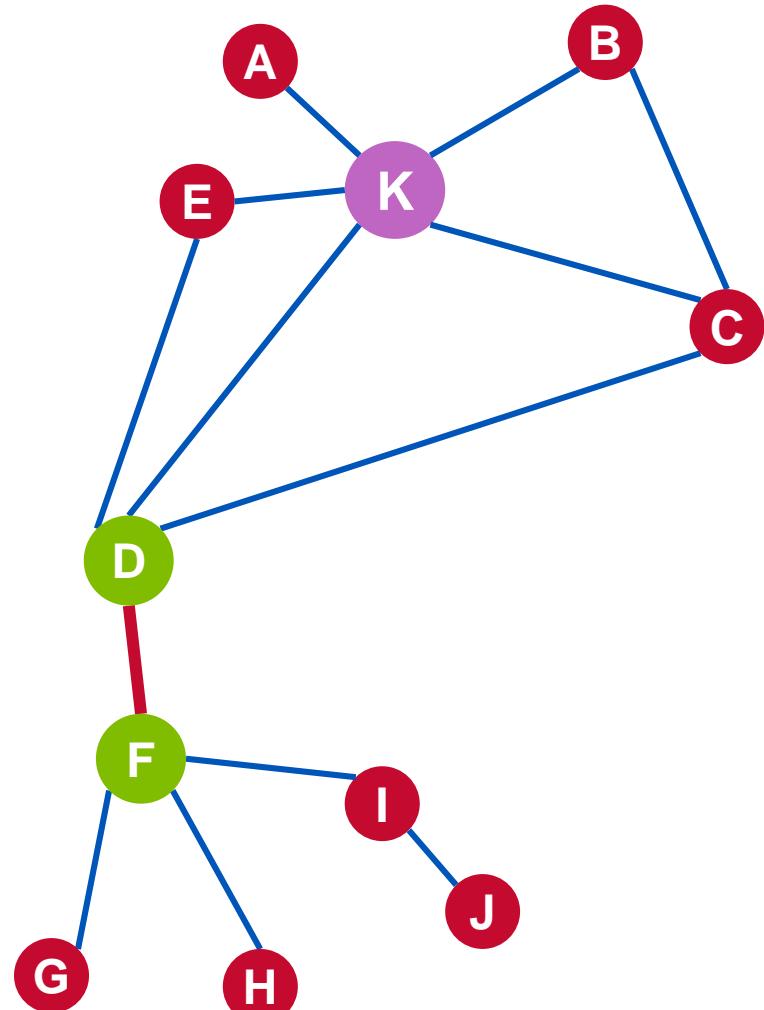
Preferential Attachment

- ▶ Some networks are typified by the majority of new nodes connecting to nodes that already have a high degree, resulting in some nodes having a disproportionately higher degree than the majority of other nodes
 - The majority of nodes have a low degree
 - A small number of nodes will have a high degree
- ▶ This may result from a tendency to associate with:
 - Popular people
 - Fashionable products or ideas
- ▶ Tends to result in further increased popularity
- ▶ May also result from the *halo effect*
 - A cognitive bias where our overall impression of a person influences our assessment of specific attributes (e.g., a person showing talent in one area, leading you to follow them in other areas)
- ▶ May also be due to purely higher quality of node



Node Combinations

- ▶ It can be useful to identify combinations of nodes in a network rather than just individual ones
- ▶ For example
 - Node showing highest individual centrality: Node K
 - Highest combined centrality: Nodes D and F combined
 - When combined these nodes will reach more nodes than Node K
 - The network would divide in two without them



Network Characteristics

- ▶ ***Homophily* or “Birds of a feather”: the tendency of nodes to connect to others who are similar on some variable**
 - Individuals tend to associate with others with whom they share similar characteristics (e.g., age, gender, class, beliefs)
 - Social interactions in such homogeneous groups are easier
 - People are more likely to be influenced by those that are similar to them
 - Can be a cause of “groupthink” and poor innovation
 - The strength of ties in such a group may be strong or weak
- ▶ **Identifying homophily:**
 - Degree assortativity of graph
`degree_assortativity_coefficient(G[, x, y, ...])`
 - Assortativity for node attributes
`attribute_assortativity_coefficient(G, attribute)`
 - Assortativity for numerical node attributes
`numeric_assortativity_coefficient(G, attribute)`

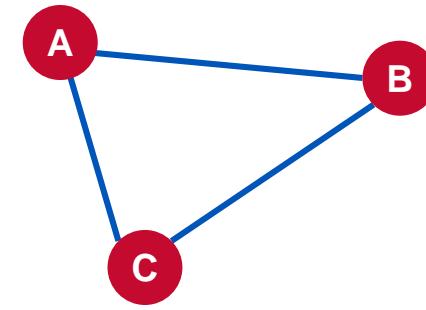
Cliques

- ▶ A *clique* is a maximally-connected subgraph where every node connects to every other node
- ▶ Identify cliques using the `find_cliques()` function

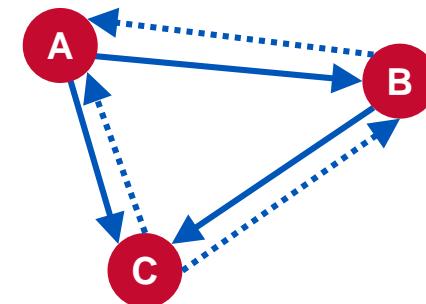
```
In [393]: list(nx.find_cliques(G_media))
Out[393]:
[['s15', 's06', 's04'],
 ['s15', 's01', 's05'],
 ['s15', 's01', 's04'],
 ['s13', 's17'],
 ['s13', 's12', 's14'],
 ['s17', 's06', 's16'],
 ['s17', 's06', 's04'],
 ['s06', 's12', 's04'],
 ['s09', 's08'],
 ['s09', 's02', 's05'],
 ['s09', 's02', 's10'],
 ['s03', 's02', 's05', 's01'],
 ['s03', 's02', 's10'],
 ['s03', 's11', 's04'],
 ['s03', 's08', 's07'],
 ['s03', 's07', 's10'],
 ['s03', 's04', 's01'],
 ['s03', 's04', 's12'],
 ['s14', 's11'],
 ['s14', 's07']]
```

Density

- ▶ “Network density” describes the portion of the *potential connections* in a network that are actual connections
 - A *potential connection* exists where a person *could* know another person, or a computer *could* connect to another
 - An *actual connection* is one that actually exists
- ▶ Network density is calculated as the ratio of the number of edges in the network compared to the number of all possible edges
- ▶ Total number of possible edges calculated as
 - Nodes * (nodes – 1) for directed graphs and
 - Nodes * (nodes – 1) / 2 for undirected graphs
- ▶ Cohesion refers to the likelihood of a return tie in a network
 - Syntax: `graph.density(g)`



density = 3/3

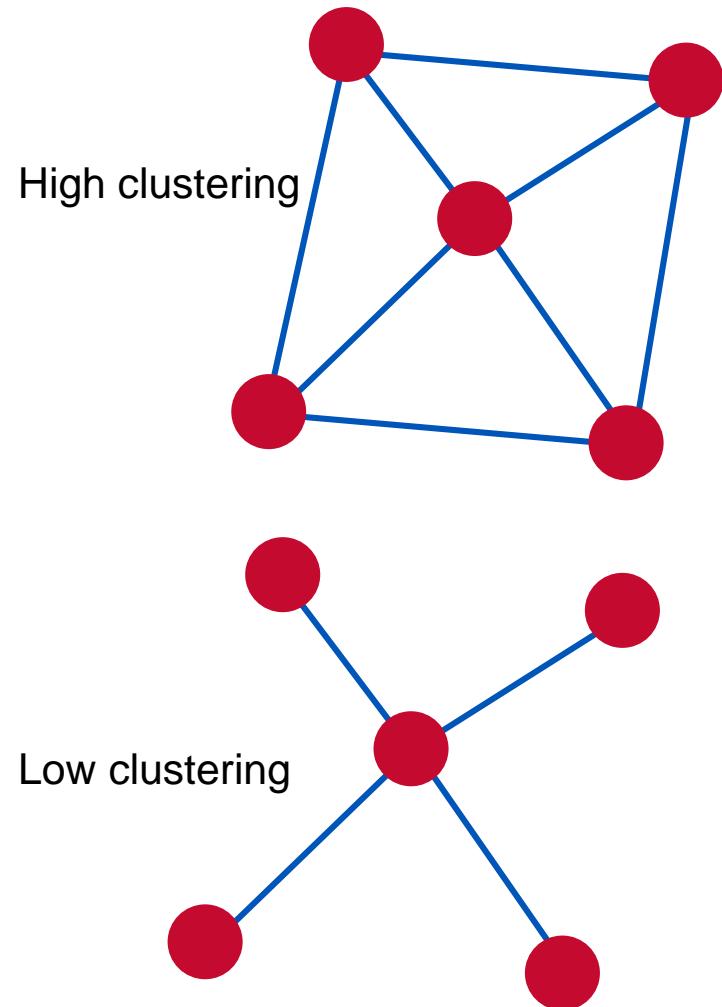


density = 3/6

— Actual connection
..... Connection not present

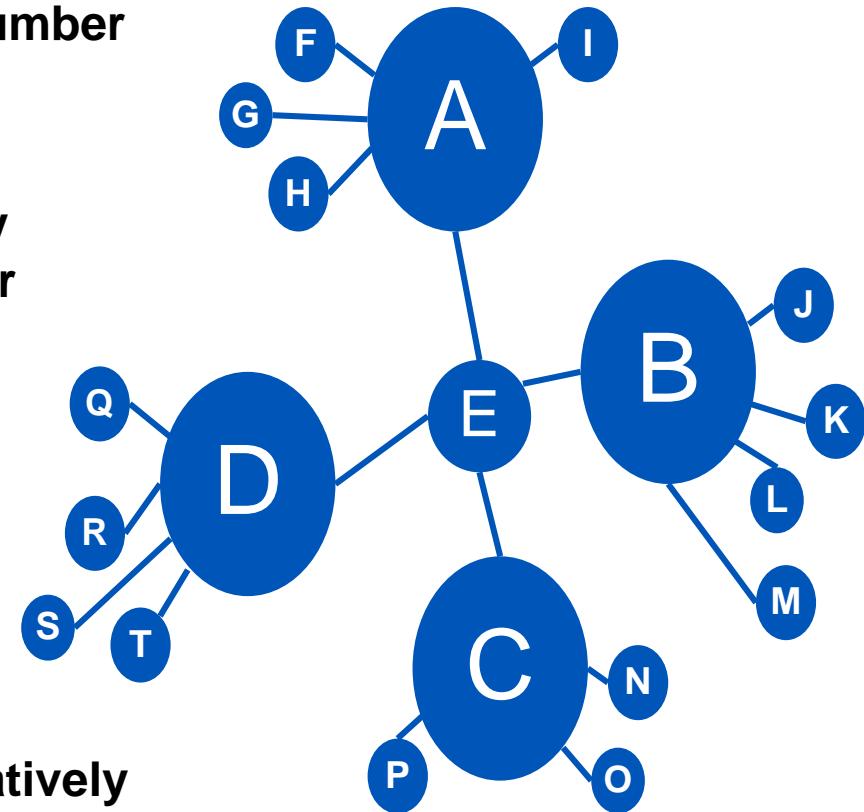
Clustering

- ▶ A node's clustering coefficient is a measure of the degree to which nodes in a network are likely to cluster together (aka *transitivity*)
- ▶ Calculated as the number of closed triplets in the node's neighborhood divided by the total number of triplets
 - The fact that *A* knows *B*, and *B* knows *C* does not guarantee that *A* knows *C* as well, but makes it more likely...the friend of my friend is not necessarily my friend, but is more likely to be my friend than random



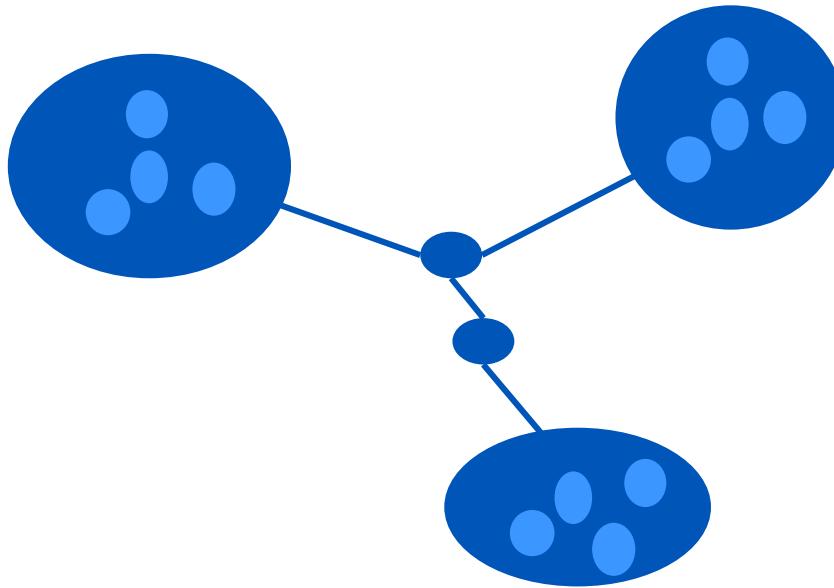
Centralized vs. Decentralized Networks

- ▶ Some networks depend on a small number of highly connected nodes (as in preferential attachment)
- ▶ They may also have a core of densely connected users that connect a larger periphery of users
 - High-degree users are connected to other high-degree users
 - To measure this, the centralization measure is calculated as the difference in degrees between nodes
- ▶ Coordination in such networks is relatively easier, but may become disconnected if key players leave the network



Small Worlds

- ▶ **Small world networks have many separate clusters with bridges between them**
 - They display high *clustering coefficients* and short *average paths*



- ▶ **Intergroup communication**
 - Nodes that connect across groups are referred to as *bridges*
 - May help to spur innovation

Hubs and Authorities

- ▶ Hubs serve as directories that point to many other nodes
- ▶ An authority represents a node that is linked by many different hubs
 - The hits() function can be used to generate node hub and authority scores

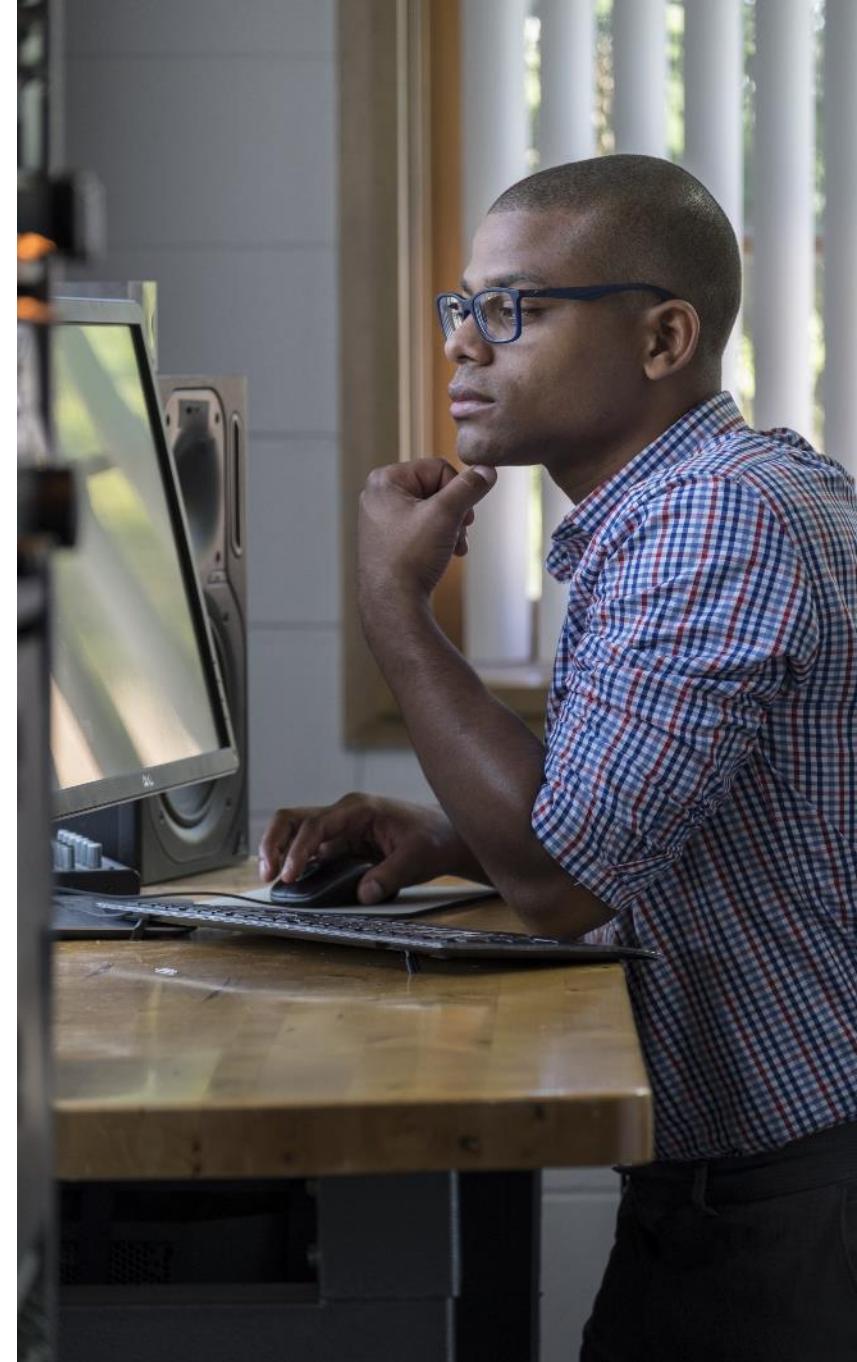
```
In [428]: h,a = nx.hits(G_media)
```

```
In [429]: h
Out[429]:
{'s01': 0.17728800304479775,
 's02': 0.1431457096038481,
 's03': 0.1558433010859018,
 's04': 0.1332740335510058,
 's05': 0.08183992968133277,
 's06': 0.01924363681142347,
 's07': 0.018200730421763525,
 's08': 0.018065907555675775,
 's09': 0.020489065545534823,
 's10': 0.028528057452318897,
 's11': 0.048031345290959225,
 's12': 0.0153545434221398,
 's13': 0.008510439857056218,
 's14': 0.0098823020707924,
 's15': 0.09038335508412862,
 's16': 0.013017410015771546,
 's17': 0.018902229505549505}
```

```
In [430]: a
Out[430]:
{'s01': 0.1772880028921963,
 's02': 0.14314570964990397,
 's03': 0.15584330100965155,
 's04': 0.13327403358466733,
 's05': 0.08183992958205069,
 's06': 0.019243636840613153,
 's07': 0.018200730455322417,
 's08': 0.018065907548802093,
 's09': 0.020489065576666764,
 's10': 0.028528057432213715,
 's11': 0.04803134524715571,
 's12': 0.01535454345259167,
 's13': 0.008510439894498168,
 's14': 0.009882302107078172,
 's15': 0.09038335513159801,
 's16': 0.013017410059311698,
 's17': 0.018902229535678883}
```

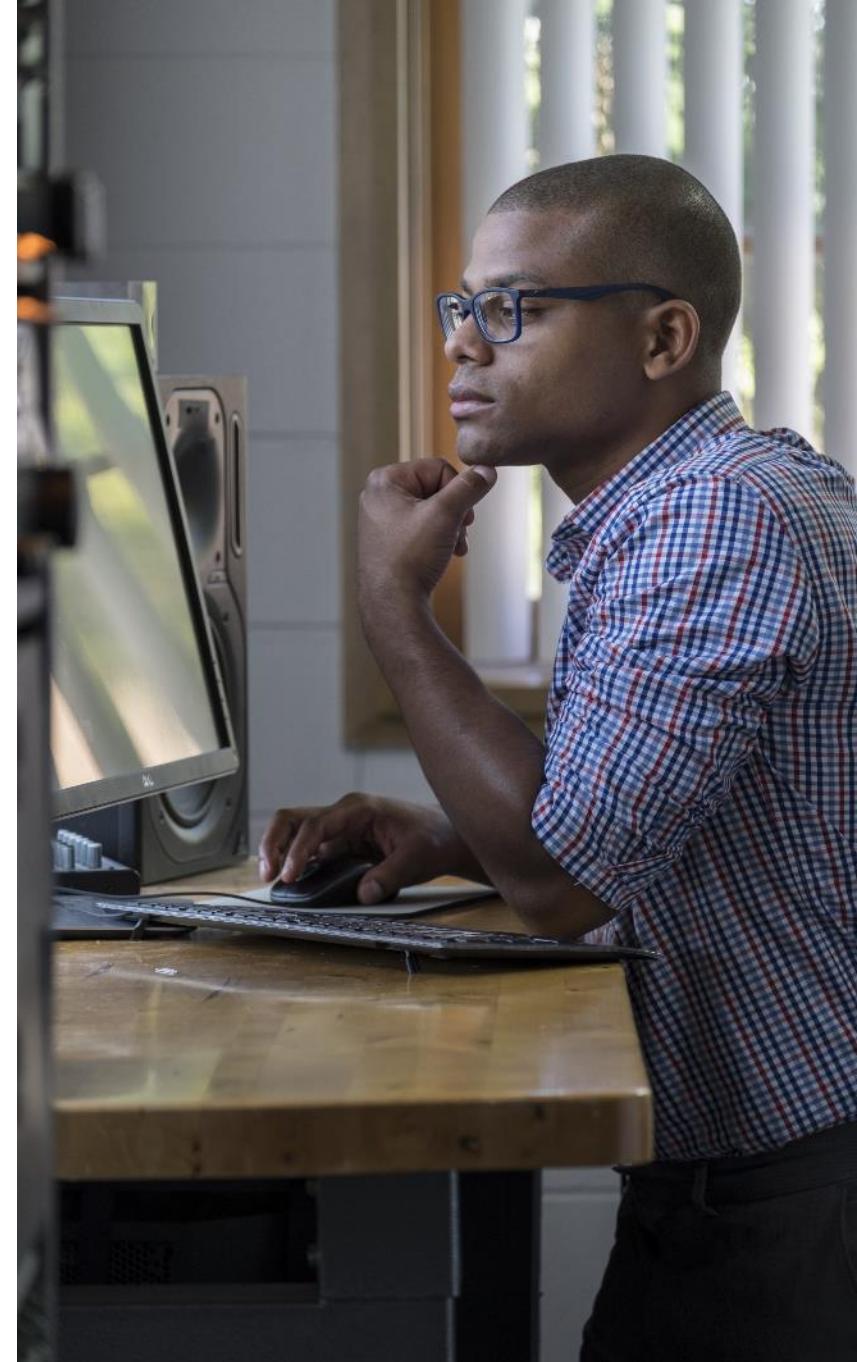
Thoughts for You as a Data Scientist

- ▶ SNA measures can be interpreted in different ways, depending on the context of the analysis
- ▶ For example, in retail
 - Highly connected people may be *negatively* influenced by purchases in their network
 - May be looking for new things to differentiate themselves from their group
 - Moderately connected people may be *positively* influenced by purchases in their network
 - Likely, these are people who are trying to “keep up with the Joneses”
 - Weakly connected people may not be influenced by purchases in their network



Thoughts for You as a Data Scientist

- ▶ Think about which network structures might most benefit your organization/work environment
- ▶ How might you advise management on the identification and use of the key actors in a network?
 - Which methods and insights from social network analysis would be most useful?
- ▶ How might you go about optimizing a network's structure?
 - Which measures would you use?
- ▶ It might be a good idea to make actors aware of their network structures so that they can work with them to their benefit, and perhaps expand it



Contents

- ▶ Introduction to Networks
- ▶ Representing Networks as Data
- ▶ Visualizing Network Relationships
- ▶ Hands-On Exercise 9.1
- ▶ Egocentric Analysis
- ▶ Sociocentric Analysis

Hands-On Exercise 9.2



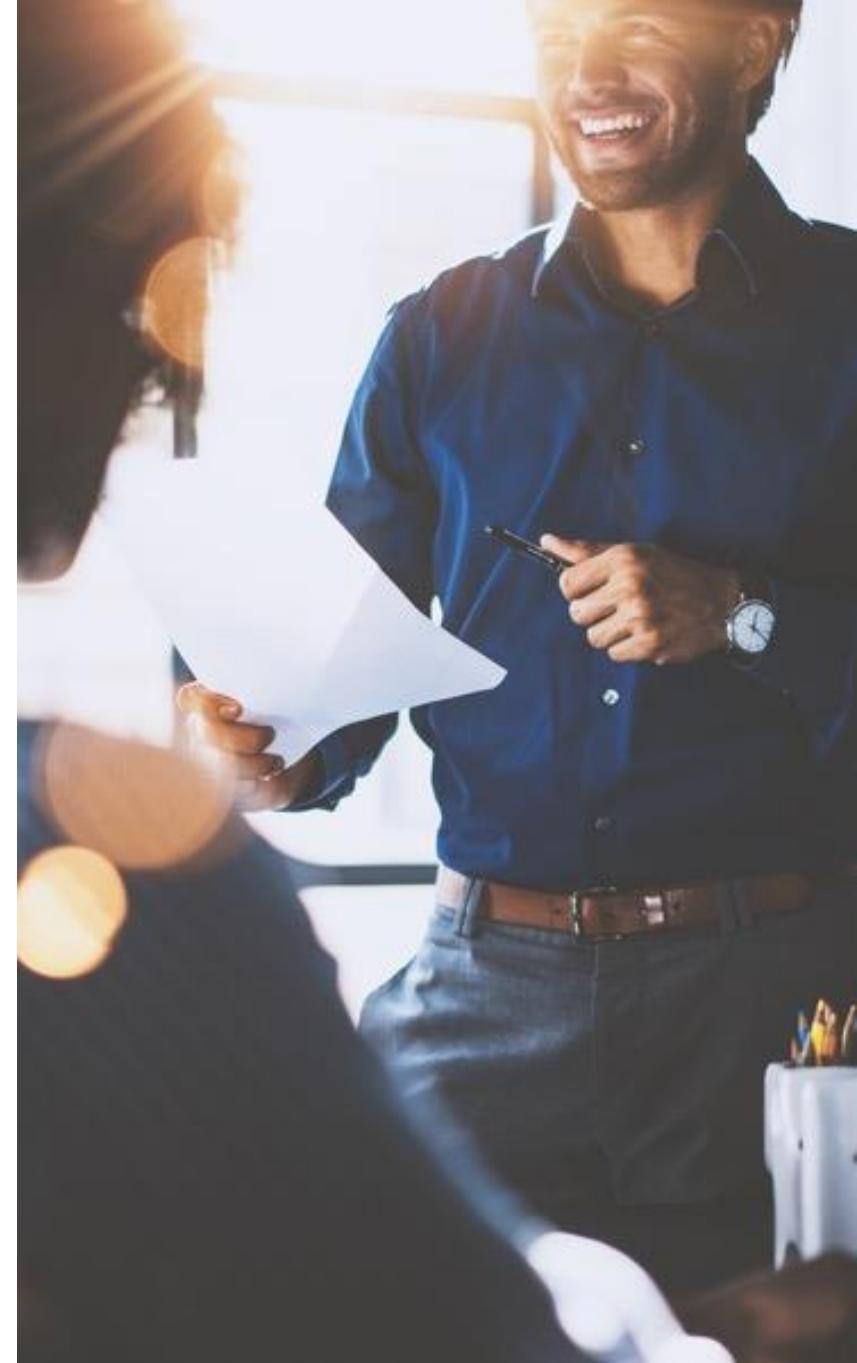


Hands-On Exercise 9.2

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 9.2: Analyzing Network Graphs

Objectives

- ▶ **Analyze data that has a network structure**
- ▶ **Plot networks to visualize relationships**
- ▶ **Explore egocentric and sociocentric methods of analyzing networks**



Chapter 10

Data Science and

Big Data Analytics:

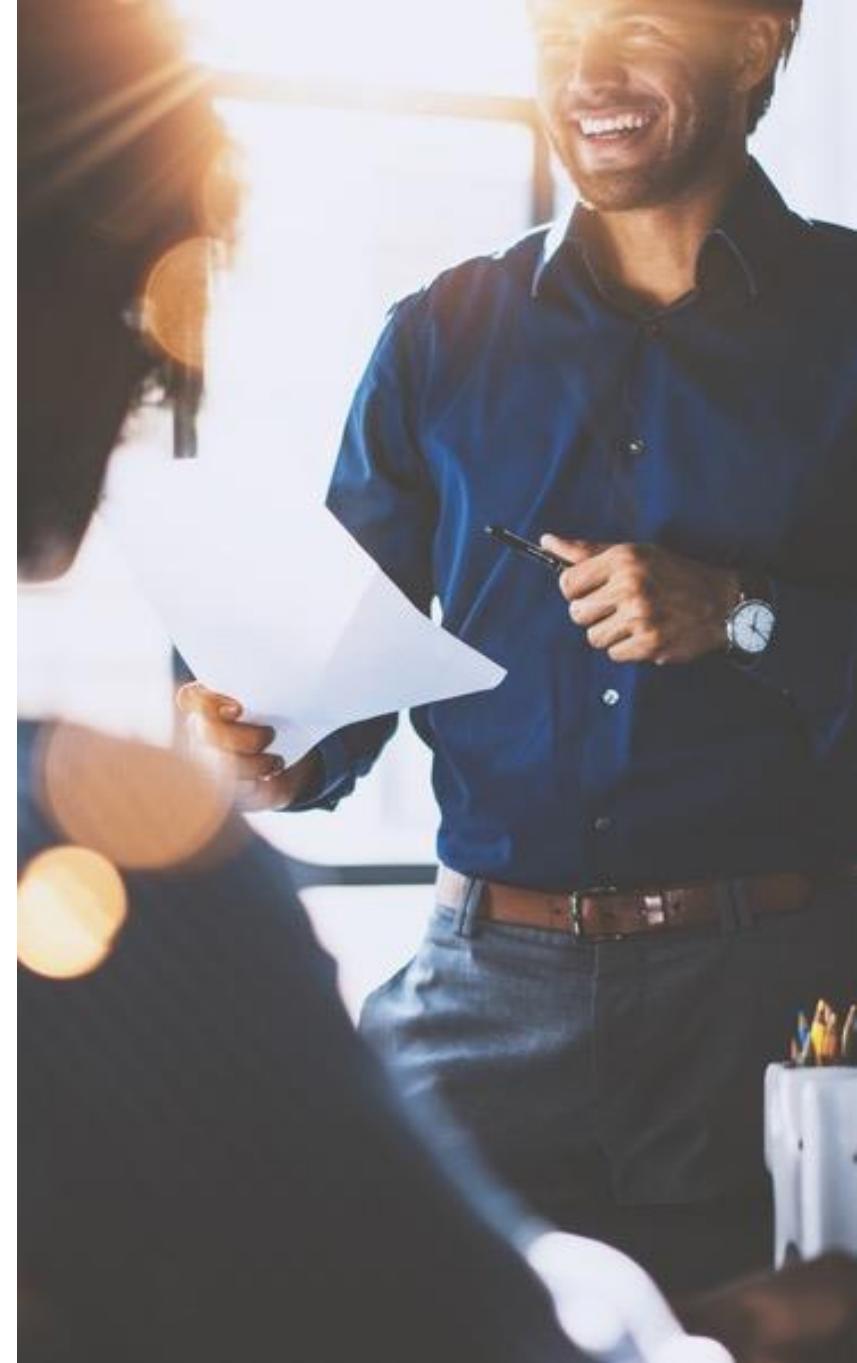
Final Thoughts



LEARNING TREE
INTERNATIONAL

Objectives

- ▶ Examine some approaches to handling Big Data analytics
- ▶ Draw the various aspects of Data Science together in the data mining life cycle
- ▶ Investigate the merging of Big Data analytics with traditional data warehouses
- ▶ Consider the ethics of data analytics
- ▶ Explore the required skillsets of a data scientist



Contents

Dealing with Big Data

- ▶ The Hadoop Environment
- ▶ ML with Spark and PySpark
- ▶ Data Science in the Cloud
- ▶ The Life Cycle: Bringing It All Together
- ▶ Hands-On Exercise 10.1
- ▶ Final Thoughts



How Big Is Big Data?

- ▶ **Big data is a name given to datasets that cannot be captured and processed by conventional hardware and software within an acceptable length of time**
 - If it cannot be easily catered for by available memory, disk space, or processing power, it can be termed “Big data”
- ▶ **The amount of data being referred to will differ over time as technological capabilities increase over time**
- ▶ **The three “Vs” are often used to define Big Data**
 - Volume: the amount of data is too large to store
 - Velocity: the speed at which the data is being produced is more than can be processed through conventional means; e.g., video streaming
 - Variety: the diversity of the data being analyzed, which may include text, video, audio, images, etc.

Drawbacks of Analyzing Data on a Single Machine

- ▶ Python can be used quite easily to analyze smaller datasets on single machines
- ▶ As the volumes of data being generated and acquired by organizations are increasing exponentially, a single machine falls short when it comes to working on very large datasets
- ▶ When analyzing Big data, loading all the data into a single machine's memory will likely fail with exceptions
- ▶ Advanced machine learning algorithms can be very effective when applied to large datasets
 - This is only possible where data can be stored and processed with distributed data storage systems

Hadoop

- ▶ **It is possible to store vast amounts of information on a low-cost platform such as Hadoop**
 - Changes the economics and the dynamics of large-scale computing
- ▶ **Hadoop is an open-source framework for processing and querying vast amounts of data on large clusters of commodity hardware**
 - Currently is the industry *de facto* framework for Big Data processing
 - Relies on an active community of contributors
- ▶ **A Hadoop cluster scales computation capacity, storage capacity, and I/O bandwidth by adding commodity servers**
 - Computations execute in parallel, close to the data
 - By analyzing the data within the server in which it resides, calculations are faster

I/O = input/output

Combining Python and Hadoop

- ▶ **Python's strength lies in enabling powerful machine learning productivity**
- ▶ **Hadoop's strength is to store and process very large amounts of data**
- ▶ **Linking Python with Hadoop for scalability is the next logical step**
 - Entire datasets can be analyzed, rather than just samples
- ▶ **While several programming frameworks for Hadoop exist, data analysts tend to be very productive in Python**
 - Leverages your existing Python skills instead of having to learn additional programming languages, such as Java

The Python and Hadoop Solution

- ▶ **With a combined Python and Hadoop system**
 - Python's packages will take care of data analysis operations with the preliminary functions such as data loading, exploration, analysis, and visualization
 - Hadoop will take care of parallel data storage as well as computation power against distributed data
- ▶ **Solutions combining Python and Hadoop can be achieved in the cloud**
 - Using platforms such as:
 - Amazon
 - Azure
 - Google Cloud

Contents

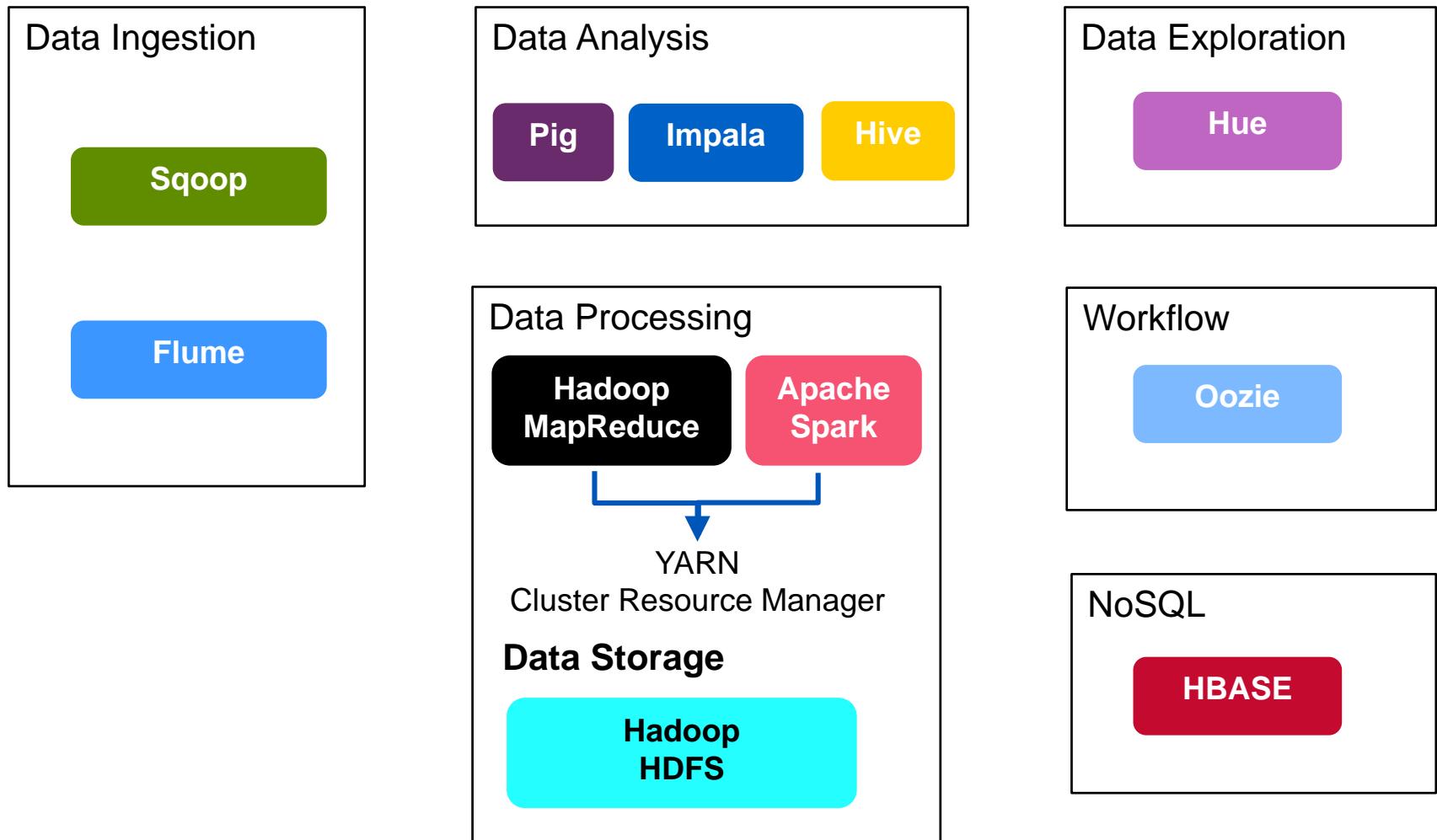
- ▶ Dealing with Big Data

The Hadoop Environment

- ▶ ML with Spark and PySpark
- ▶ Data Science in the Cloud
- ▶ The Life Cycle: Bringing It All Together
- ▶ Hands-On Exercise 10.1
- ▶ Final Thoughts



The Hadoop Ecosystem



Hadoop Ecosystem Components

► Hadoop Distributed File System

- Provides file permissions and authentication
- Streaming access to file system data
- Hadoop provides a command line interface to interact with HDFS
- Suitable for the distributed storage and processing
- A storage layer for Hadoop

► Apache HBase

- A NoSQL database or non-relational database
- The table can have thousands of columns
- Stores data in HDFS
- Mainly used when you need random, real-time, read/write access to your Big Data
- Provides support to high volume of data and high throughput

Hadoop Ecosystem Components

► Hadoop MapReduce

- An extensive and mature fault tolerance framework
- Original Hadoop processing engine
- Primarily Java based
- Based on the map and reduce programming model

► Apache Spark

- An open-source cluster computing framework
- 100 times faster performance than MapReduce
- Supports machine learning, business intelligence, streaming, and batch processing
- Incorporates Spark Core and Resilient Distributed Datasets (RDDs), Spark SQL, Spark Streaming Machine Learning Library (MLlib), GraphX

Hadoop Ecosystem Components

► Apache Pig

- An open-source dataflow system
- Converts pig script to MapReduce code
- An alternative to writing MapReduce code
- Best for ad-hoc queries such as join and filter

► Apache Impala

- High performance SQL engine which runs on Hadoop cluster
- Ideal for interactive analysis
- Supports a dialect of SQL (Impala SQL)
- Very low latency, measured in milliseconds

► Apache Hive

- Similar to Impala, executes queries using MapReduce
- Best for data processing and ETL

► Hue

- An open-source web interface for analyzing data with Hadoop
- Provides SQL editors for Hive, Impala, MySQL, Oracle, PostgreSQL, Spark SQL

Hadoop Ecosystem Components

► **Sqoop**

- It is used to import data from relational databases such as Oracle and MySQL to HDFS and export data from HDFS to relational databases

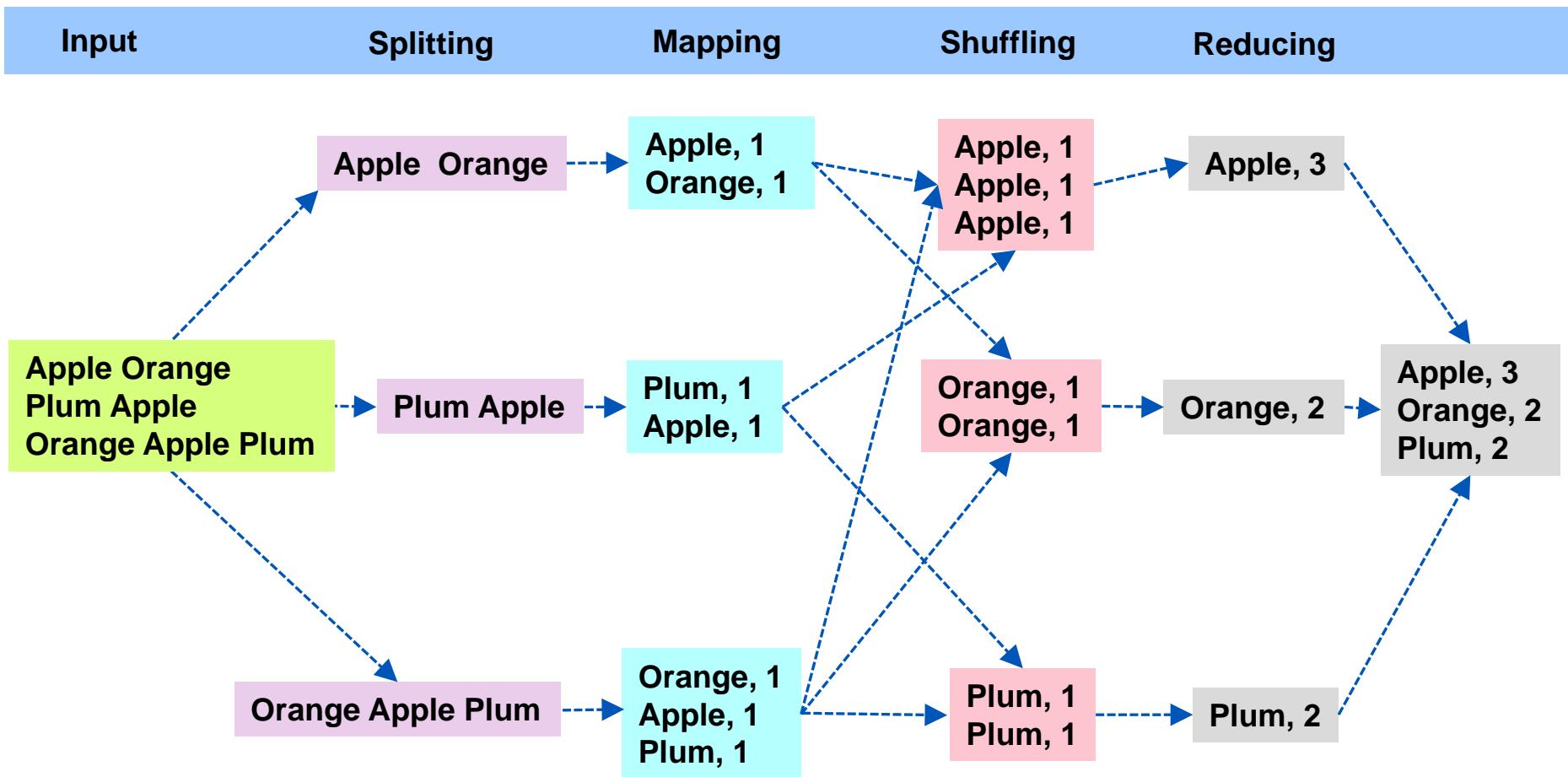
► **Apache Flume**

- A distributed service for ingesting streaming data
- Ideally suited for event data from multiple systems

► **Apache Oozie**

- A workflow or coordination system used to manage the Hadoop jobs

MapReduce Processing Big Picture



MapReduce Processing

- **The five common steps of parallel computing are as follows:**
 1. Read prepared Map input data:
 - Take a list of key-value pairs (usually requires some preprocessing of the raw data)
 - Map input: $[(k_1, v_1), (k_2, v_2) \dots]$
 2. Run the custom map function
 - Map output: $[(k_1, v_1), (k_2, v_2) \dots]$
 3. Shuffle the Map output to the Reduce processors
 - Similar keys are sorted and grouped together and input to the same reducer
 4. Run the custom reduce function:
 - Reduce input: $(k, [v_1, v_2, \dots])$
 - Reduce output: (k, v)
 5. Produce the final output:
 - Hadoop collects all reducer output and combines and writes them into a text file

Contents

- ▶ Dealing with Big Data
- ▶ The Hadoop Environment

ML with Spark and PySpark

- ▶ Data Science in the Cloud
- ▶ The Life Cycle: Bringing It All Together
- ▶ Hands-On Exercise 10.1
- ▶ Final Thoughts



Spark and PySpark

- ▶ **Apache Spark is an open source, general-purpose distributed computing engine used for processing and analyzing large amounts of data**
 - Like Hadoop MapReduce, it also distributes data across the cluster and processes the data in parallel
 - Spark can process data in-memory, while Hadoop MapReduce has to read from and write to a disk
 - As a result, Spark may be up to 100 times faster
 - Written in Scala
 - PySpark allows Spark apps to be developed in Python
- ▶ **PySpark allows exploratory data analysis and machine learning pipelines at scale**

Revoscalepy

- ▶ **Provides functions to analyze datasets using Python rather than native Java**
 - Allows Python programmers to perform computations distributed over several nodes in a cluster
 - No detailed knowledge of Hadoop MapReduce framework is required
 - Only requires basic information about connection to the Hadoop cluster
- ▶ **To perform an analysis, the following information must be provided:**
 - Where the computations should take place (the compute context)
 - The data to use (the data source)
 - What analysis to perform (the analysis function)
- ▶ **Functions within Revoscalepy may be used for small and large datasets**
 - Data analysis routines may be developed locally with smaller datasets, and the same analytics code can be later deployed to Hadoop
 - The underlying Revoscalepy code handles the distribution of the computations across nodes, so you don't have to worry about it

Revoscalepy and Spark

- ▶ When running on Spark, the Revoscalepy analysis functions process data contained in the Hadoop Distributed File System (HDFS)
 - Computations can also be run locally while accessing HDFS data
- ▶ The Revoscalepy functions go through the following steps:
 1. A master process is initiated to run the main thread of the algorithm
 2. A MapReduce job is initiated by the master process, which passes through the data
 3. The mapper produces “intermediate results objects” for each task processing a subset of data
 - These are combined using a combiner, and then a reducer
 4. For iterative algorithms, the master process initiates subsequent MapReduce jobs, if required
 5. Once complete, the final results are returned

Analysis Functions Supported in Hadoop Context

- ▶ **Some of the Revoscalepy analysis functions currently supported within the Hadoop compute context are:**
 - rx_summary: basic summary statistics of data, including computations by group
 - rx_lin_mod: fits a linear model to data
 - rx_dtree: fits a classification or regression tree to data
 - rx_predict: calculates predictions for fitted models

Example of a Linear Regression Model With Revoscalepy and Spark

- ▶ By default, the local compute context is the implicit computing environment, until you specify a remote compute context

```
rx_lin_mod("prestige ~ education + income + women", data =  
Prestige)
```

- ▶ Run the rx_lin_mod() clustering function using the Spark compute context

```
cc = rx_spark_connect()  
rx_lin_mod("prestige ~ education + income + women", data =  
Prestige)
```

Contents

- ▶ Dealing with Big Data
- ▶ The Hadoop Environment
- ▶ ML with Spark and PySpark

Data Science in the Cloud

- ▶ The Life Cycle: Bringing It All Together
- ▶ Hands-On Exercise 10.1
- ▶ Final Thoughts

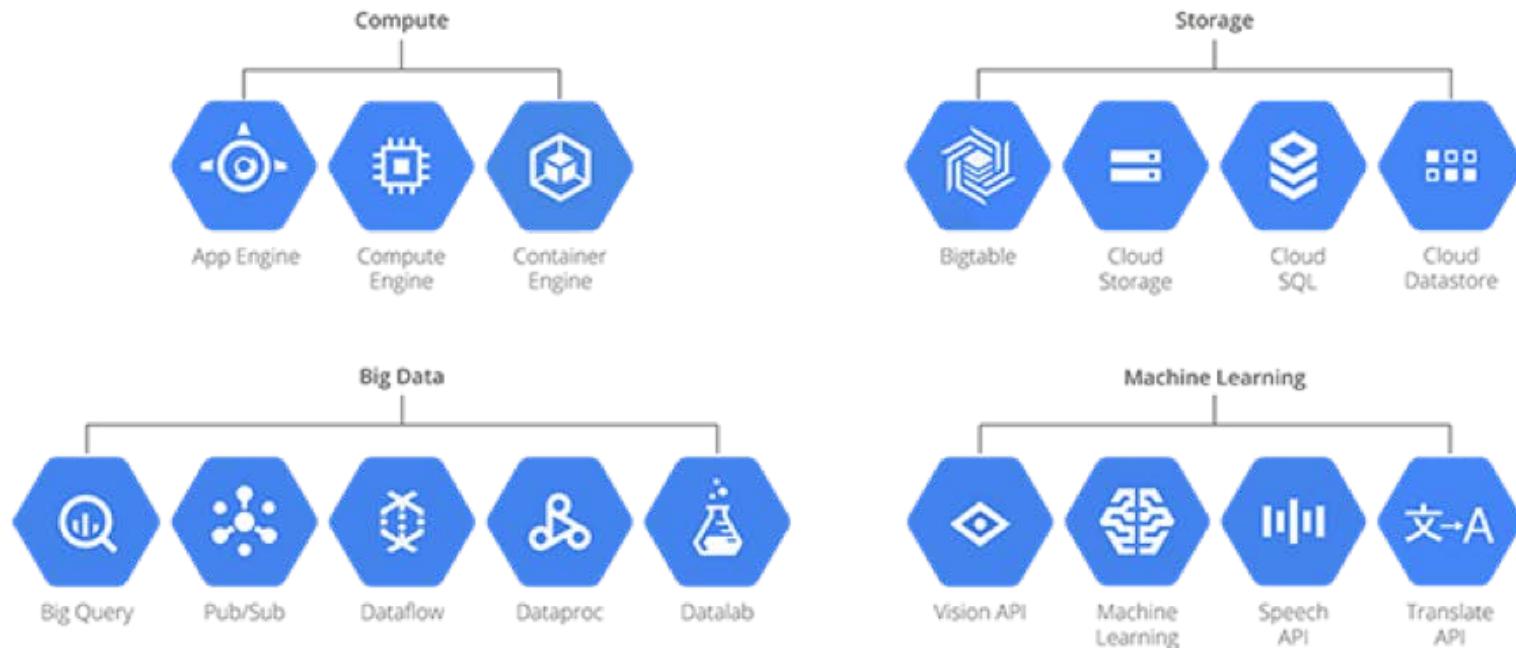


Cloud Computing Platforms

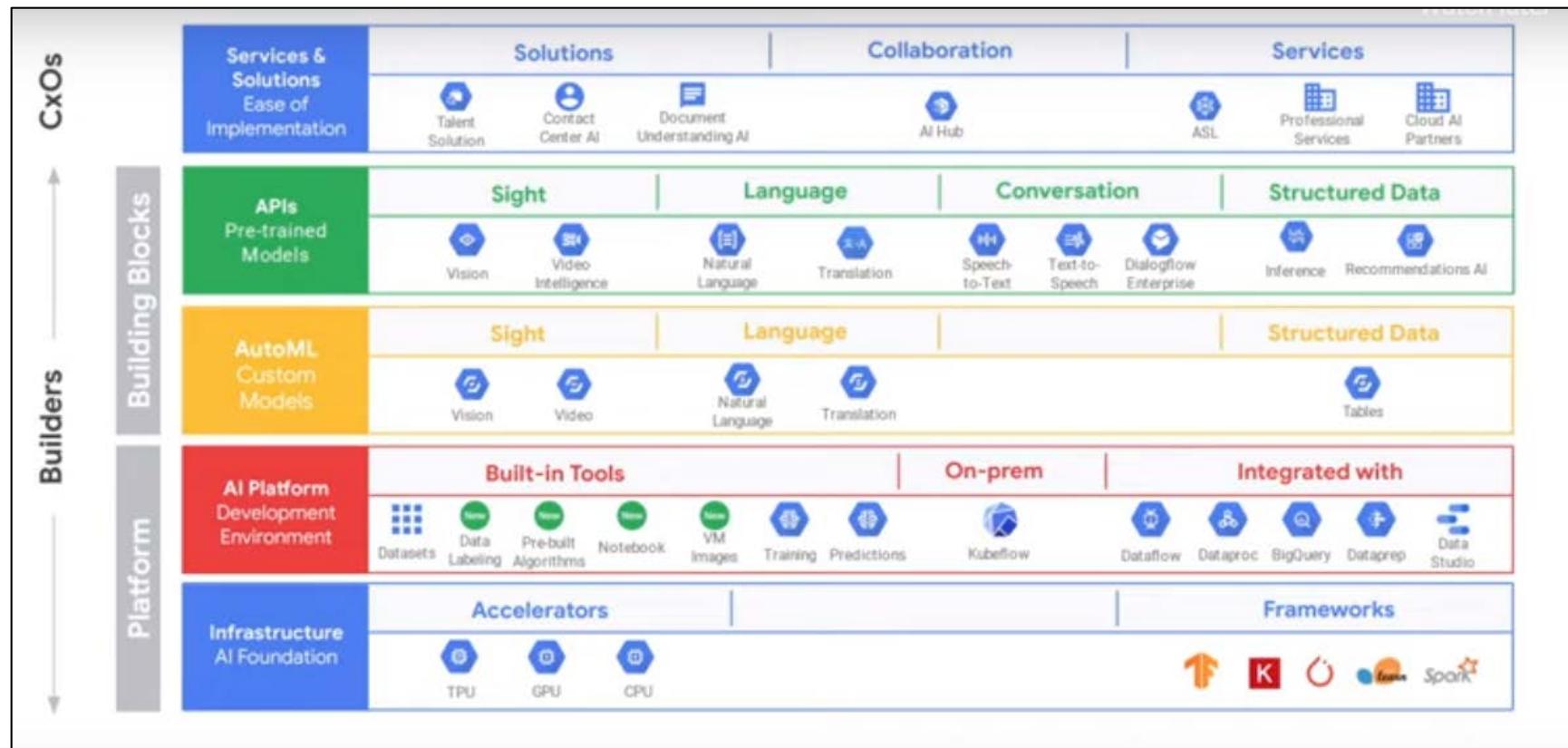
- ▶ **Allow developers to build and host services on infrastructure with on-demand computing power and storage to host, scale, and manage applications**
 - Simplifies the operation and maintenance of applications
 - Offers high availability and dynamic scaling under a pay-as-you-go pricing model
- ▶ **Give developers options that would be difficult to achieve if limited to the organization's own infrastructure**
- ▶ **Popular platforms are offered by:**
 - Microsoft
 - Google
 - Amazon

Google Cloud Platform Offering

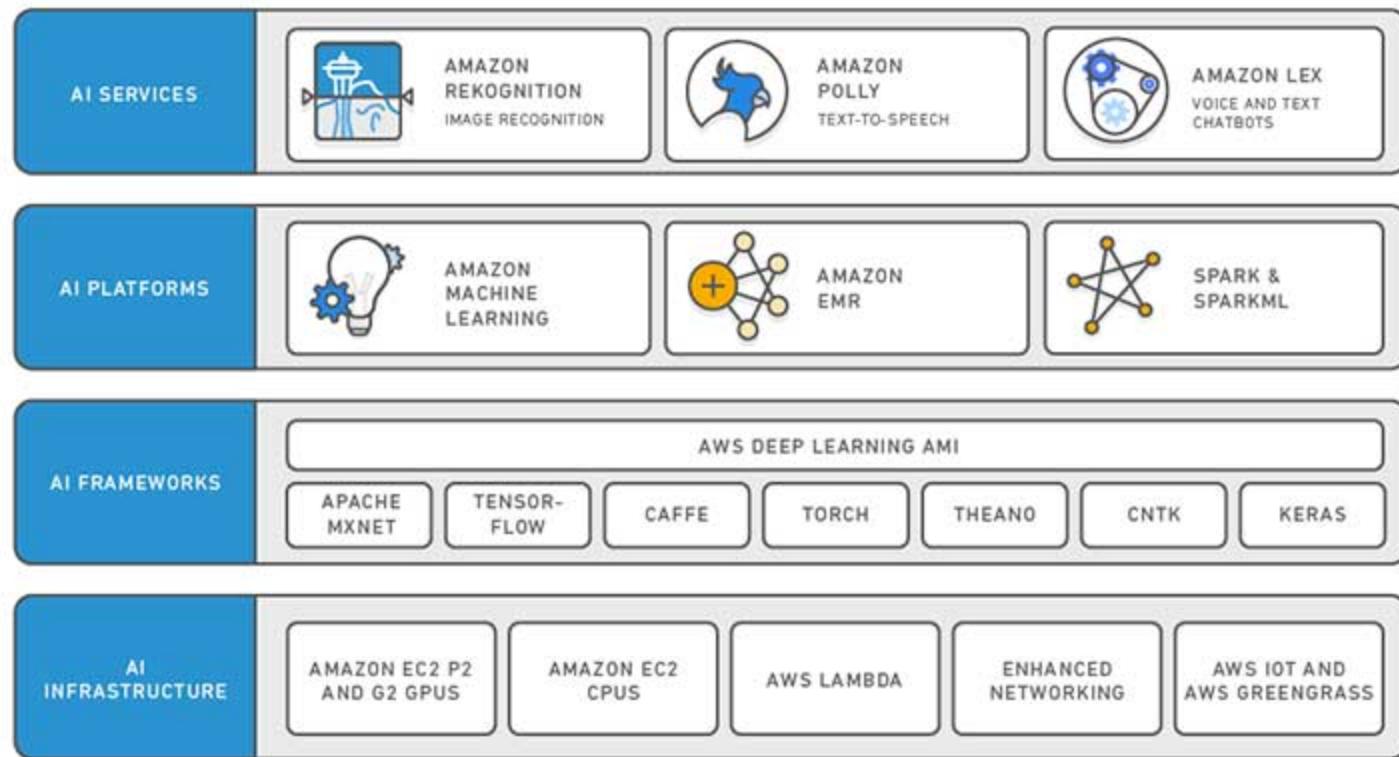
Google Cloud Platform



Google Cloud Platform Offering



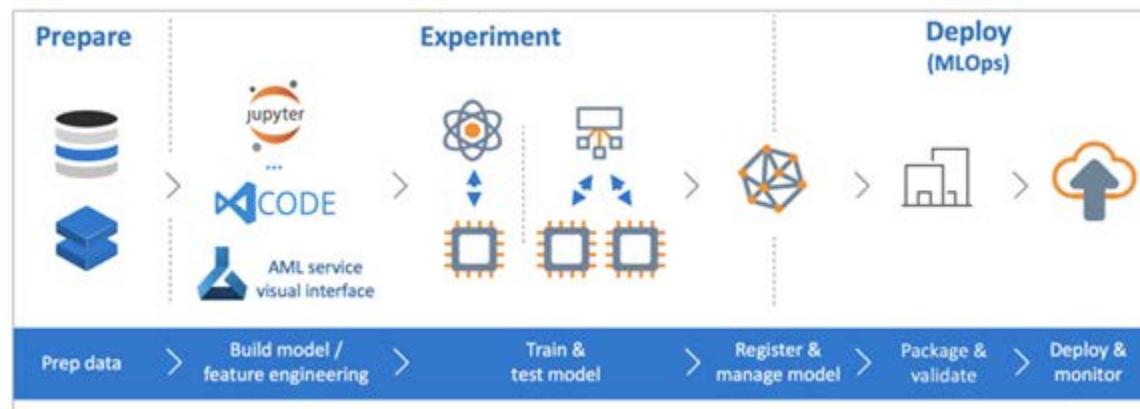
Amazon Web Services



Azure Machine Learning Service Overview

► Offers:

- Python SDK
- Cloud resources for machine learning experiments
- Ability to train models locally or with on-demand, scalable resources
- Improved pipeline productivity
- Ability to deploy as web services



Microsoft Azure Offerings

- **Some of the services offered break down into the following categories**
 - Not a comprehensive list—new services are added frequently

Virtual Machines (VMs)	From .75 GB RAM, 1 core to 448 GB RAM, and 32 cores
Application platform	Host web applications and mobile back-ends on Azure, with options to scale on demand
Data and storage	Hosted SQL database, general purpose storage, cache services, DocumentDB NoSQL database, SQL Data Warehouse
Cognitive services	Application programming interfaces (APIs) such as speech recognition, text analytics, and face analysis
Analytics	Services for storing and analyzing bulk data, including Hadoop, Spark, and Machine Learning APIs
Internet of Things (IoT)	Platform for connecting sensor-equipped devices, collecting data, and for storing or analyzing the results
Media services	Encoding and content delivery network
Developer services	Visual Studio Team Services, including build automation and load testing, application analytics, and HockeyApp for gathering and analyzing crash reports
Identity services	Azure Active Directory (also used by Office 365), multi-factor authentication

Contents

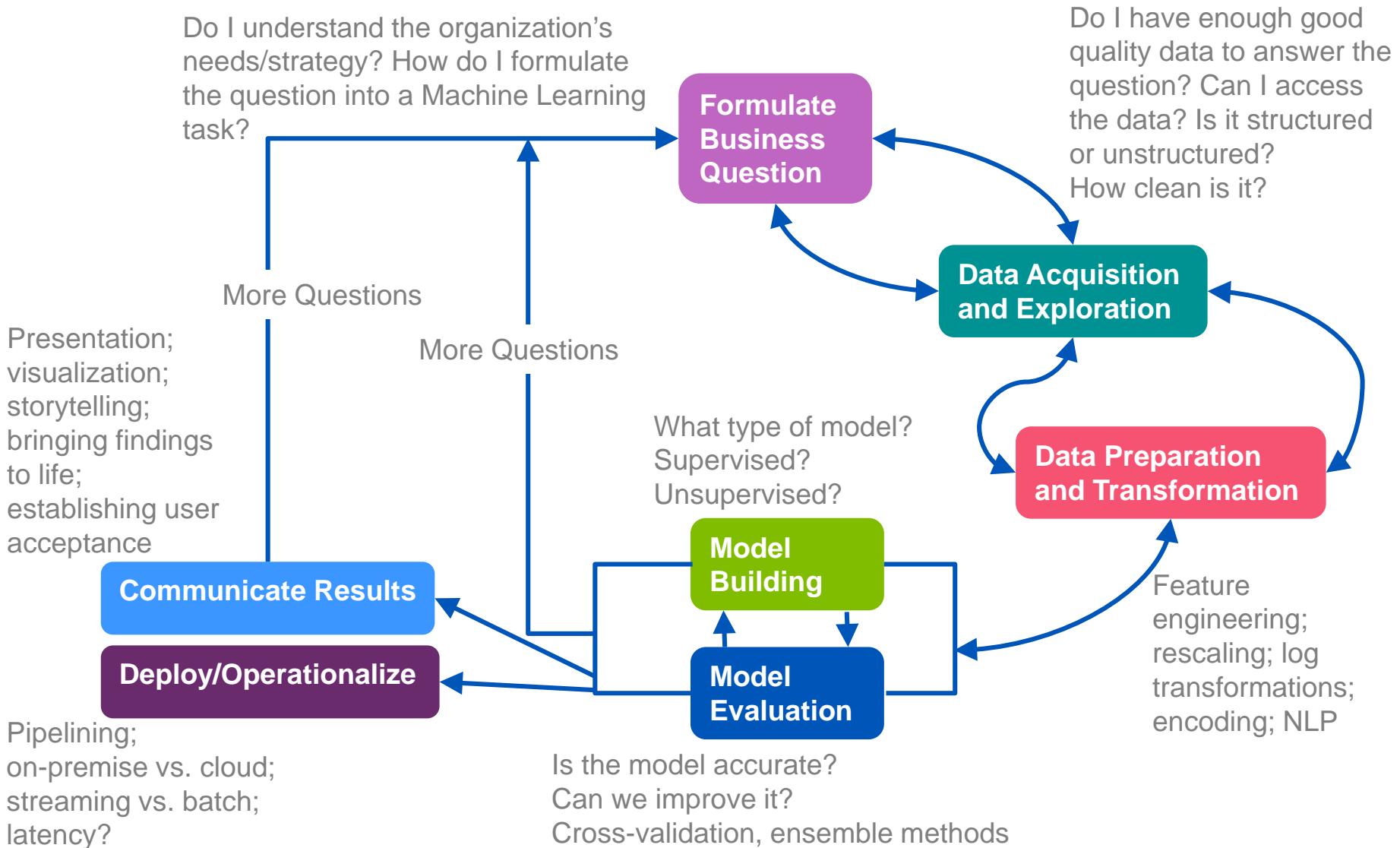
- ▶ Dealing with Big Data
- ▶ The Hadoop Environment
- ▶ ML with Spark and PySpark
- ▶ Data Science in the Cloud

The Life Cycle: Bringing It All Together

- ▶ Hands-On Exercise 10.1
- ▶ Final Thoughts



Data Science Life Cycle



Bringing It All Together: The Data Mining Life Cycle

- ▶ **Business understanding**
 - Translating the business objective into a data mining problem
 - Collecting or procuring the required datasets that can meet the data mining problem
- ▶ **Data exploration and investigation**
 - Examining the central tendency (*mean, median, mode*) and distribution (*range, variance, standard deviation*) of the data
 - Visualizing the data
- ▶ **Data preparation and cleaning**
 - Data cleaning
 - Structured
 - Substituting missing values
 - Eliminating outliers
 - Unstructured
 - Removing punctuation marks, stemming, stop-word removal

Bringing It All Together: The Data Mining Life Cycle

► Data transformation

- Structured
 - Creating aggregate values
 - Implementing range substitution for discrete values
 - Converting numerical values to categorical values
 - Scaling
 - Normalization
- Unstructured
 - Creating Term-Document Matrices

► Model design and development

- Machine learning techniques
 - Regression
 - Classification
 - Clustering
 - Association mining
 - Social Network Analysis

Bringing It All Together: The Data Mining Life Cycle

► **Model verification and testing**

- Model accuracy assessment metrics
 - Error rate
 - Confusion matrix

► **Deployment into a pipeline and/or knowledge dissemination**

- Scaling the ML product
- Dissemination of lessons learned to the wider business

Data Mining Life Cycle: Credit Card Fraud Detection

Do Now

- Apply the stages of the life cycle to describe an analytics solution you might build for fraud detection and include:
 - What data mining technique would you use?
 - What data might you use and how would you preprocess it?
 - Actions arising from the results of the model

Data Mining Life Cycle: Employee Attrition

Do Now

- Apply the stages of the life cycle to describe an analytics solution you might build to detect the likelihood that an employee will leave an organization in the first six months of hiring and include:
- What data mining technique would you use?
 - What data might you use and how would you preprocess it?
 - Actions arising from the results of the model (to benefit the organization)
-
-
-
-
-

Data Mining Life Cycle: Emergency Room Visits

Do Now

- Apply the stages of the life cycle to describe an analytics solution you might build to detect the likelihood that a patient that visits the Emergency Room (ER) will be back in the ER within three months and include:
- What data mining technique would you use?
 - What data might you use and how would you preprocess it?
 - Actions arising from the results of the model (to benefit the organization)
-
-
-
-
-

Deployment (Data Engineering)

- ▶ Putting ML models into production involves quite a lot of Data Engineering and may include some or all the following:

1. Designing, Building and Operationalizing Data Processing Systems

- Selecting the appropriate infrastructure and storage technologies, and designing data pipelines (lifecycle management of data)
- Dealing with issues such as:
 - Capacity planning, system availability and fault tolerance
 - Tradeoffs between latency and throughput
 - Event processing (at least once, in order, and exactly once, etc.)
 - Migrating from on-premise to cloud
- Estimating and Monitoring Transfer, Storage (Big Data) and Processing costs and performance
- Data acquisition and import, and integration with new data sources

Deployment (Data Engineering)

2. Operationalizing Machine Learning Models

- Measuring, monitoring, and troubleshooting machine learning models
- Dealing with the impact of dependencies of machine learning models

3. Ensuring Solution Quality

- Data security (encryption, key management)
- Legal compliance (e.g., Health Insurance Portability and Accountability Act (HIPAA), Children's Online Privacy Protection Act (COPPA), FedRAMP, General Data Protection Regulation (GDPR))

4. Ensuring scalability and efficiency

- Pipeline monitoring
- Building and running test suites
- Resizing and autoscaling resources
- Planning, executing, and stress testing data recovery (fault tolerance, rerunning failed jobs, performing retrospective re-analysis)

Communicating Results

- ▶ **The ability to communicate data findings is a critical skill for a data scientist**
- ▶ **Lessons learned must be disseminated so that the wider business understands the implications**
- ▶ **Rigorous data exploration and statistical modeling will typically come to nothing if poorly presented**
 - The outputs of advanced analytics are often complex to understand, put into action, and to monitor
- ▶ **Storytelling is essential**
 - Need to explain findings in terms that businesspeople can relate to
 - More open to any business initiatives if analysis is understood
 - Visualization is a key component of storytelling particularly for nontechnical audiences
 - Can be used to illustrate the implications of the analysis in terms of business impact

Data Visualization

► Two types of data visualization

- Exploratory
 - Facilitates understanding of the data by discovering trends or insights, or by developing a hypothesis about the data
 - Audience is typically a very small group
- Communication to a wider audience
 - Goal is to visually advocate for a hypothesis or to communicate a large volume of information very concisely
 - Requires a different skill, and is typically done using different tools

Which Chart Types?

- ▶ **Choosing the right chart type for your data will depend on the type of analysis your audience needs**
- ▶ **There are four common forms of data analysis for everyday business decisions**
 - Relationships
 - For example, what is the relationship between our click-through rate and our position on Google search results?
 - Comparisons
 - Among variables
 - For example, compare salespeople for performance
 - Over time
 - For example, how is website traffic trending over the past year?
 - Distributions
 - For example, what is the usual number of website visits?
 - Compositions
 - For example, what is the breakdown of our customer base?

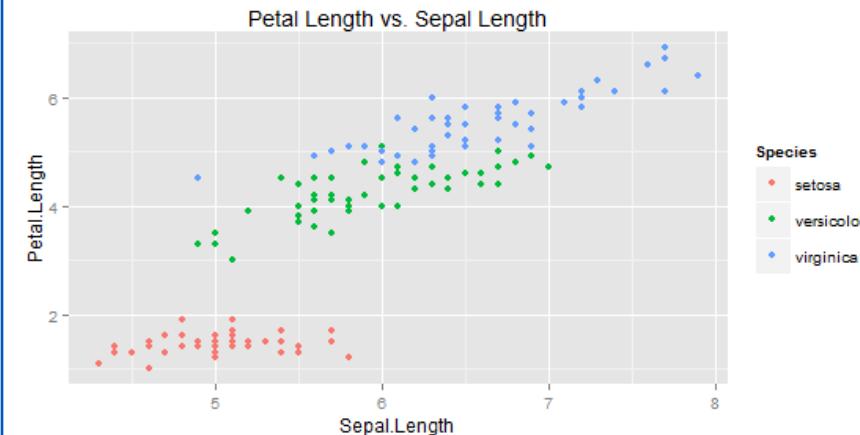
Charting Relationships and Comparisons with Matplotlib and pandas

Syntax

Relationship

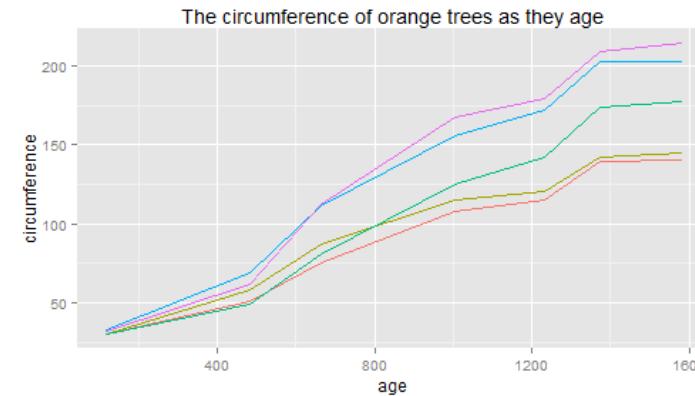
```
iris.plot.scatter(x=Sepal.Length,  
y=Petal.Length, c=Species, main =  
"Petal Length vs. Sepal Length")
```

Chart



Comparison over time

```
df.plot.line(age, circumference,  
data = Orange, c = Tree,  
title = "The circumference of orange  
trees as they age")
```



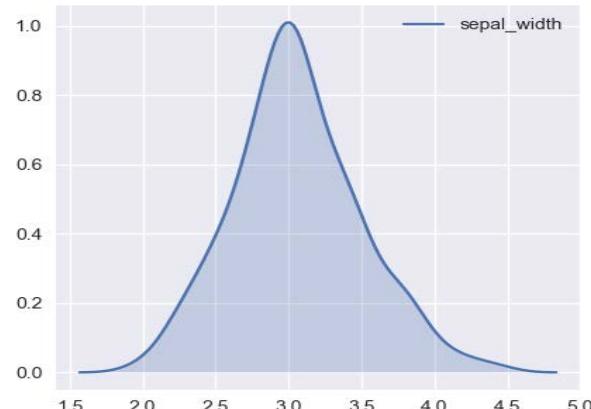
Charting Distributions and Compositions

Syntax

Distribution

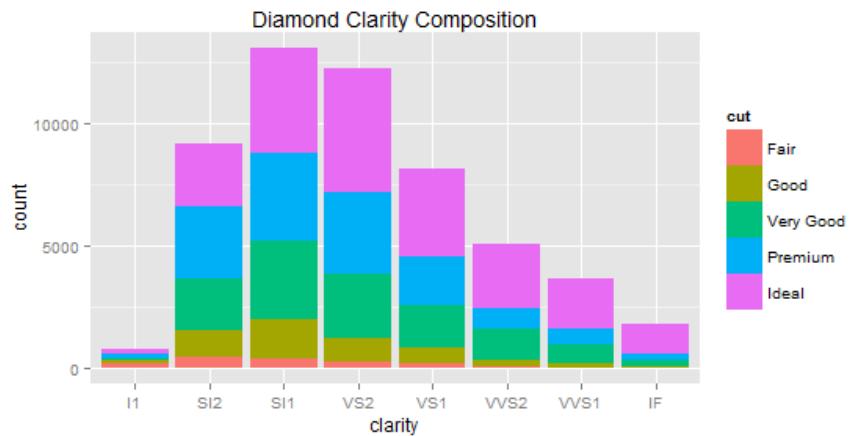
```
import seaborn as sns  
  
sns.kdeplot(iris['sepal_width'],  
            shade=True)
```

Chart



Composition

```
df.plot(kind='bar', stacked=True)
```



Contents

- ▶ Dealing with Big Data
- ▶ The Hadoop Environment
- ▶ ML with Spark and PySpark
- ▶ Data Science in the Cloud
- ▶ The Life Cycle: Bringing It All Together

Hands-On Exercise 10.1

- ▶ Final Thoughts





Hands-On Exercise 10.1

In your Jupyter Exercise Dashboard, please refer to Hands-On Exercise 10.1: Data Visualization

Contents

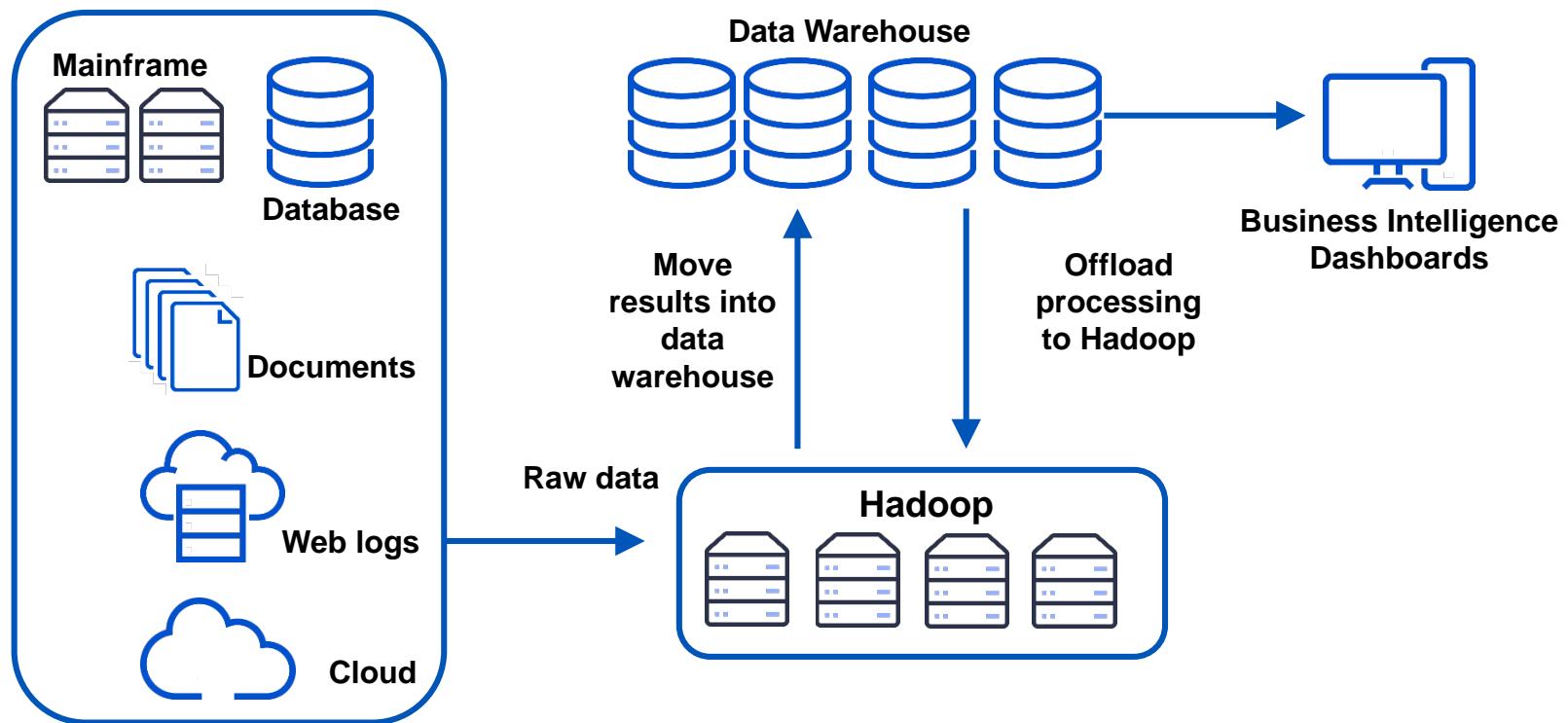
- ▶ Dealing with Big Data
- ▶ The Hadoop Environment
- ▶ ML with Spark and PySpark
- ▶ Data Science in the Cloud
- ▶ The Life Cycle: Bringing It All Together
- ▶ Hands-On Exercise 10.1

Final Thoughts



Integration With the Traditional Data Warehouse

- ▶ Hadoop and Spark can be used as a data mining engine to process raw data into analytic models
- ▶ Hadoop and Spark can act as an ETL layer by processing Big Data to then load into a traditional data warehouse for use by more traditional analytic and BI tools



BI = Business Intelligence

Ethics of Data Analytics and Machine Learning

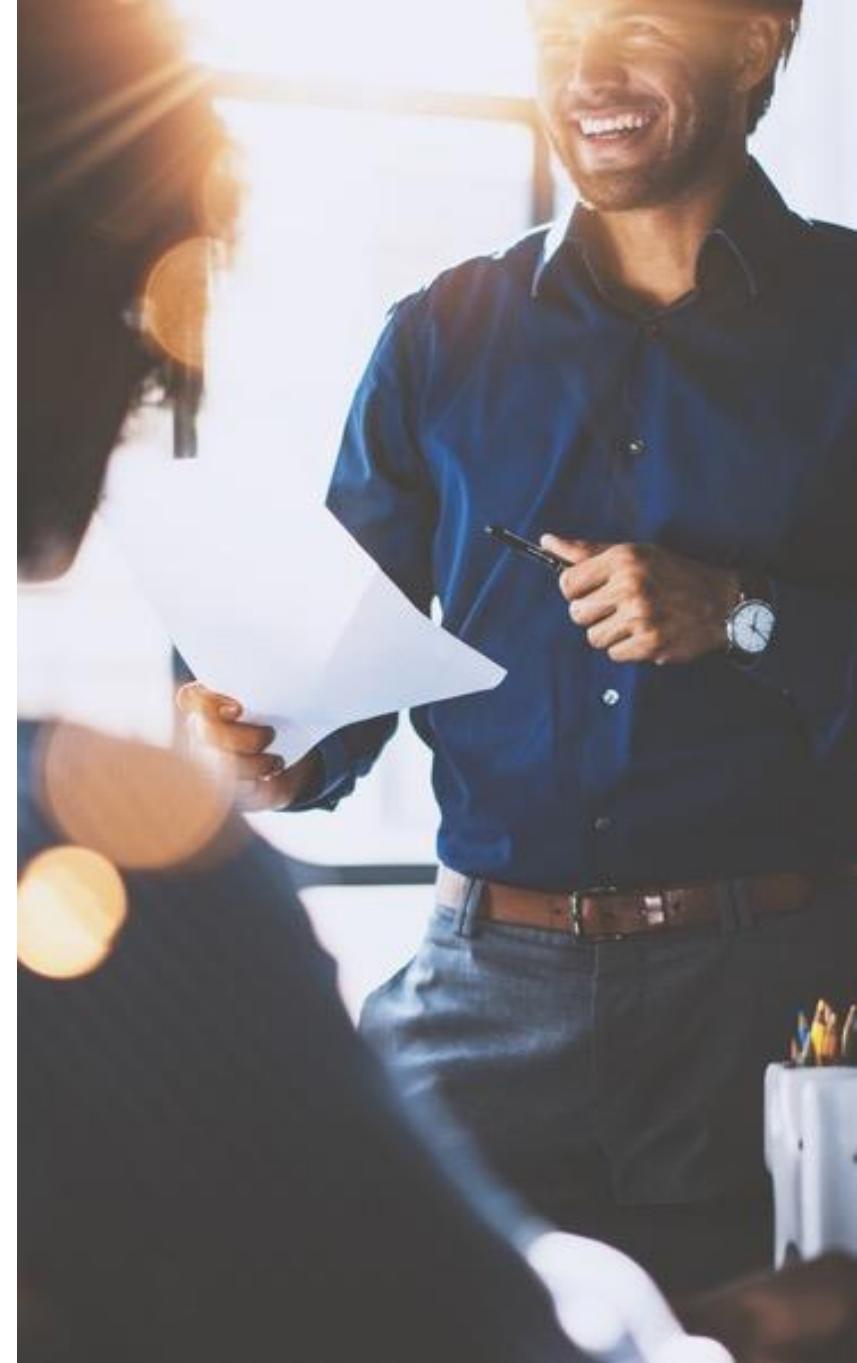
- ▶ **Personal data can help organizations customize new products and services, and thus achieve greater business efficiencies and competitive advantage**
- ▶ **However, it can easily backfire**
 - Overly zealous data mining can run the risk of destroying an organization's reputation
 - Users may stop doing business with a company due to privacy concerns
- ▶ **CIOs and data scientists need to find the right balance between deriving value and actionable insights from the private data they hold on their customers, and respecting consumers' privacy rights**
- ▶ **Transparency is the best policy**
 - Clear privacy policies
 - Incentives—consumers are willing to part with personal information if there is a good reason

Where to Go From Here?

- ▶ A data scientist is someone who is better at statistics than a typical software engineer, and better at software engineering than a typical statistician
- ▶ Improving your skills as a data scientist
 - Develop and improve your critical thinking skills
 - Be curious
 - Get good at statistics and machine learning
 - Learn to code
 - Python, R, SQL,
 - Become familiar with varied databases
 - Oracle, MySQL, MongoDB, Neo4J, etc.
 - Get comfortable “munging” (cleaning and transforming data) and visualizing data
 - Work with Big Data
 - Hadoop, MapReduce, Spark, Azure ML, etc.
 - Learn from other data scientists
 - Kaggle competitions, Kdnuggets, R-bloggers, Meetups

Objectives

- ▶ Examine some approaches to handling Big Data analytics
- ▶ Draw the various aspects of Data Science together in the data mining life cycle
- ▶ Investigate the merging of Big Data analytics with traditional data warehouses
- ▶ Consider the ethics of data analytics
- ▶ Explore the required skillsets of a data scientist



Chapter 11

Course Summary



LEARNING TREE™
INTERNATIONAL

Course Objectives

- ▶ Work with Python to analyze structured and unstructured data
- ▶ Harness data mining methods to answer crucial business questions from internal and external data sources
- ▶ Employ supervised machine learning techniques to estimate future outcomes
- ▶ Unearth patterns in data with unsupervised techniques
- ▶ Preprocess unstructured data for deeper analysis



Course Objectives

- ▶ **Apply clustering, classification, and regression to datasets**
- ▶ **Generate association rules from transaction data**
- ▶ **Mine relational patterns from network data**
- ▶ **Communicate data-driven narratives to a wider audience**

