* Big Data Scenarios are usually classified by:
  + Volume: Terabytes of data
  + Velocity: Continuous stream of real-time data
  + Variety
  + Veracity: Unreliable, noisy, biased or incorrect data
* Data Science: Involves principles, processes and methods for identifying and understanding phenomena via automated/semi-automated analysis of data
* Data Mining: Extracting knowledge from data via algorithms which incorporate these principles
* Data Science workflow:  
  1. Data Collection: Retrieve/import Raw Data from sources  
  2. Data Preparation: Clean Manipulate and aggregate data into analyzable form  
  3. Analysis and Modelling  
  4. Report & Visualization
* Knowledge Discovery in Databases (KDD): Detailed timeline for data science projects
  + Exploration🡪Preprocessing🡪transformation🡪Data Mining🡪Interpretation and Evaluation
  + Human Interaction is important
* Analytics Base Table (ABT): Structure we use to represent a new dataset
  + Row = Case/Example which are made up of
    - Descriptive features: Characteristics and properties of cases
    - Target Feature: Feature we aim to predict/analyse
  + Features depend on domain
  + Descriptive features have types
* Important considerations to think about with features:
  + Data availability
  + Timing
  + Longevity
* Cost-sensitive features: Think about how financial cost of getting features and new data
* Privacy-sensitive features: Some might be good in an algorithm but very bad to include from legal/ethical standpoint
* When working on a new dataset, we characterize the data to understand nature of data in ways like:
  + Understand meaning of each feature in data
  + Identify type of value taken by each feature
  + Identify any data quality issues
* CSV are comma separated values
  + File format used in tabular data
  + Like plaintext but values are split by commas or a different separator
  + Has a header line with column names
  + Python has a built in module to assist with reading and writing
    - import csv  
      fin = open("countries10.csv", "r")  
      reader = csv.DictReader(fin)  
      for row in reader:  
      print(row)  
      fin.close()
    - fout = open("output.csv", "w")  
      fields = ["code", "name", "population", "life\_expect"]  
      writer = csv.DictWriter(fout, fieldnames=fields)  
      writer.writeheader()  
      for row in data:  
      writer.writerow(row)  
      fout.close()
* JSON (JavaScript Object Notation)
  + Lightweight format
  + Based on Javascript
  + Built from 2 structures:
    - Object which is a collection of name/value pairs.
      * Begins and ends with {}
      * Each name is followed by :
      * Name/Val pairs separated by commas
    - Arrays:
      * Separated by commas and like python begins and ends with []
  + Can mix both
  + Outmost value can be either
  + Whitespace isn’t needed
  + Language agnostic
  + json.dumps(): python struct🡪JSON string
  + json.loads(): JSON strong🡪Python struct
* XML
  + Markup language
  + Describes data using tags which are user define
  + Tags opened and closed using <tag> and </tag> respectively
  + Can have 0+ name/val attribute paors
  + <collection>  
    <note>  
    <to>Alice</to>  
    <from>John</from>  
    <subject>Reminder</subject>  
    <body>Remember to buy milk!</body>  
    </note>  
    <note>  
    <to>John</to>  
    <from>Alice</from>  
    <subject>Shopping</subject>  
    <body>I forgot the milk!</body>  
    </note>  
    </collection>
  + Hierarchal data format
  + Use trees with nodes to represent
    - Document = Root node
    - Lowest level has leaf node
  + Parse the xml amd finds tags interested in
    - import xml.etree.ElementTree  
      tree = xml.etree.ElementTree.parse("products.xml")

for book in tree.iterfind("book"):  
n = book.findtext("name")  
a = book.findtext("author")  
print("%s by %s" % (n,a) )

* HTML
  + Like XML but is also describes presentation
  + Usually badly-written and invalid
* Web scraping: Extracting data from website using tolls which act like a browser
  + Follows the steps identification🡪collection🡪extraction🡪cleaning
  + Check a sites terms if it allows it
  + Don’t hammer site with many automated requests
  + Re-write often due to layout changes in site
  + Collection:
    - urllib.request.urlopen(link) gets the html code from website after which we do .read().decode() on it
  + Extraction:
    - BeautifulSoup allows one to parse HTMLS even bad ones
    - Example:  
      import bs4  
      parser = bs4.BeautifulSoup(html,"html.parser")  
      for match in parser.find\_all("h3"):  
      text = match.get\_text()  
      print(text)
* API (Application programming interface)
  + Web service which contain a set of HTTP request messages with a definition of expected structure of response
  + Accept a query/call for particular data
  + Able to respond dynamically
  + 1+ endpoint which are URLs which can be used to retrieve data
* Data Serialisation
  + Flattening data structures into a format which can be stored/transferred or shared with another program
  + Use eother JSON or Pickle
  + Reverse process is deserialization
  + json.dump and json.load serialises/deserialises value into a JSON value
  + JSON can be slow and doesn’t necessarily support all Python data types but they are language agnostic and always has a text-based output
  + Pickle turns python data into bytestream representation which is easily stores and then can later reconstruct the data
  + Can serialize:
    - Python native types
    - Lists,tuples, dictionaries
    - Functions
    - Objects
  + Pickle.dump() serialises pythn object into an open file and pickle.load does the opposite
  + Note if using pickle, use a binary file
  + import pickle  
    ages = {"Steve":25,"Linda":40,"John":33}  
    fout = open("example.pkl", "wb")  
    pickle.dump(ages, fout)  
    fout.close()
  + fin = open("example.pkl", "rb")  
    x = pickle.load(fin)  
    fin.close()