

## DS Certificate Course I

Notebook: Data Science Certificate

Created: 4/15/2019 4:06 PM

Updated: 10/22/2019 11:00 AM













Author: nataliegmoore@gmail.com

URL: [https://startlearning.uw.edu/courses/course-v1:UW+DATASCI400+2019\\_Winter/courseware/8aa22af4e16a4e51b0179127c27877c0/e4eed7dd3ed44063a737dad351df1dc6/?child=first](https://startlearning.uw.edu/courses/course-v1:UW+DATASCI400+2019_Winter/courseware/8aa22af4e16a4e51b0179127c27877c0/e4eed7dd3ed44063a737dad351df1dc6/?child=first)

## Lab descriptions

SourceURL: [https://startlearning.uw.edu/courses/course-v1:UW+DATASCI400+2019\\_Winter/courseware/8aa22af4e16a4e51b0179127c27877c0/e4eed7dd3ed44063a737dad351df1dc6/?child=first](https://startlearning.uw.edu/courses/course-v1:UW+DATASCI400+2019_Winter/courseware/8aa22af4e16a4e51b0179127c27877c0/e4eed7dd3ed44063a737dad351df1dc6/?child=first)

Author: nataliegmoore@gmail.com

LESSON 1	L01-2-Hello.py	hello world tutorial	
LESSON 2	L02-1-Scalars.py	floats, integers, bools...	
	L02-2-ListDict.py	lists and dictionaries	
	L02-3-ArrayDataFrame.py	using pandas, creating dataframe, creating histogram first time finding correlation coefficient	
	L02-B-SparseMatrices.pdf	sparse matrices pdf lecture	see file
LESSON 3	L03-A-1-RemoveOutliers.py	removing outliers by redefining array x to be within limits	
	L03-A-2-ReplaceOutliers.py	replacing outliers by redefining the outliers to be the mean value of array without outliers uses the complement operator	
	L03-A-Outliers.pdf	outliers pdf lecture	see file
	L03-B-1-DataTypes.py	finds out data type of array x by using x.dtype.name	
	L03-B-2-RemoveMissing.py	removes missing values by by using the Good = [x.isdigit() for element in x] x = x[Good] uses x.astype(int) first time	
	L03-B-DataTypes.pdf	data types pdf lecture	see file
LESSON 4	L04-A-1-Dataframes.py	creates df using pandas adds rows, columns to df viewing df with df.head() uses df.dtypes first time	
	L04-A-2-DataframesCSV.py	imports url using pd.csv_read(url) uses url as df header = None writing local copy by using df.to_csv('df.csv', sep=";", index=False) finds where file is located using package os	
	L04-B-1-PandasEDA_Basic.py	separates by white spaces instead of commas (sep = '\s+') defines columns of df gets size of df using df.shape() presents values of columns of df sorts values of columns using df.loc[:, "column"].sort_values() plots columns as histograms	
	L04-B-2-PandasEDA_Datatype.py	determines locations of missing values ex: QuestionMark = Auto.loc[:, 'horsepower'] == "?"	

		<p>sums missing value amount ex: <code>sum(QuestionMark)</code></p> <p>casting type of column using <code>df.astype(float)</code> ex: <code>Auto.loc[:, 'horsepower'] = Auto.loc[:, 'horsepower'].astype(float)</code></p> <p>plots histograms</p>	
	L04-C-1-NumpyPandas.py	<p>checking distinct values using <code>df.unique()</code> ex: <code>Mamm.loc[:, "BI-RADS"].unique()</code></p> <p>determines where nans are ex: <code>HasNaN = np.isnan(Mamm.loc[:, "BI-RADS"])</code></p> <p>finds median with nan values ex: <code>np.median(Mamm.loc[~HasNaN, "BI-RADS"])</code> ex: <code>Median = np.nanmedian(Mamm.loc[:, "BI-RADS"])</code></p> <p>replacing nans with median values ex: <code>Mamm.loc[HasNaN, "BI-RADS"] = Median</code></p>	
	L04-C-2-NumpyPandasEDA.py	<p>coerce and imputing median for nans: <code>Mamm.loc[:, "BI-RADS"] = pd.to_numeric(Mamm.loc[:, "BI-RADS"], errors='coerce')</code> <code>HasNaN = np.isnan(Mamm.loc[:, "BI-RADS"])</code> <code>Mamm.loc[HasNaN, "BI-RADS"] = np.nanmedian(Mamm.loc[:, "BI-RADS"])</code></p> <p>replacing outliers: <code>TooHigh = Mamm.loc[:, "BI-RADS"] &gt; 6</code> <code>Mamm.loc[TooHigh, "BI-RADS"] = 6</code></p>	
	L04-C-3-PandasNaN.py	<p>Replace &gt;Question Marks&lt; with NaNs: <code>Mamm = Mamm.replace(to_replace="?", value=float("NaN"))</code></p> <p>Count NaNs <code>Mamm.isnull().sum()</code></p> <p>Remove rows that contain one or more NaN: <code>Mamm_FewerRows = Mamm.dropna(axis=0)</code></p> <p>Remove columns that contain one or more NaN: <code>Mamm_FewerCols = Mamm.dropna(axis=1)</code></p>	
LESSON 5	L05-A-1-Normalization.py	<p>normalizes using MinMax method <code>offset = min(x)</code> <code>spread = max(x) - min(x)</code> #spread is always the amount of range of your values <code>xNormMinMax = (x - offset)/spread</code></p> <p>normalizes using Z method <code>offset = np.mean(x)</code> <code>spread = np.std(x)</code> <code>xNormZ = (x - offset)/spread</code></p> <p>comparing methods using <code>np.hstack(np.reshape(...))</code></p> <p>median absolute deviation</p> <p>Answer for quiz L05-A</p>	
	L05-A-2-Normalization.py	comparing normalizations via histograms	
	L05-A-3-Normalization.py	comparing compounds of normalization methods via histograms	
	L05-A-4-Normalization.py	denormalizations	
	L05-A-5-Binning.py	binning manually (without histograms)	
	L05-A-6-NormBinning.py	binning manually using numpy	
	L05-B-1-CategoryDecode.py	decoding values if they're categorical	
	L05-B-2-CategoryConsolidate.py	<p>category columns are decoded and missing values are imputed</p> <p>plots using : <code>Mamm.loc[:, "Shape"].value_counts().plot(kind='bar')</code> (the order with which you consolidate and decode matters)</p>	
	L05-B-3-CategoryDummies.py	<p>creates new dummy columns from dummy variables</p> <p>drops obsolete column with which dummies were created</p>	

	L05-C-PandasPractice.py	creating dataframes and adding rows/cols to them indexing the df by an id number dictionary mapping iloc instead of loc for index location joining dfs together	
LESSON 6	L06-A-1-AccessJSONObjects.py	example of JSON object	
	L06-A-2-JSONObjectinPython.py	creates JSON object converts JSON obj to python dict using: python_obj = json.loads(json_data)	
	L06-A-3-JSONArraytoPython.py	creates JSON array converts JSON array to python dict using: data = json.loads(json_array)	
	L06-A-4-PythontoJSON.py	creates empty python dict converts python dict to JSON object (a string) using: json_obj = json.dumps(python_dict) ALSO creates python array converts python array to JSON array (a string) using: json_array = json.dumps(py_array)	
	L06-A-5-OutputJSONdata.py	converts JSON object to python dict using json.loads() pretty printing of python dict using: print(json.dumps(python_obj, sort_keys=True, indent=4)) ALSO creates python dict and appends entries prints pretty using above ex writes JSON data to file using: with open("test_file.txt", 'w') as outfile: json.dump(data, outfile)	
	L06-B-1-SimpleRequest.py	using the requests package to get a text file from the internet	
	L06-B-2-WebScrape.py	pull HTML source code from url using the requests package printing the content and headers using the BeautifulSoup package to view the content in a readable format printing the title as is and as a string using BeautifulSoup:  soup = BeautifulSoup(content, "lxml") print(soup.title.string)  finding all content with specific information (such as <a> tags (websites)) looping through attributes of the HTML code converting the data you want into a python object:  data = {} for a in all_a_https: title = a.string.strip() data[title] = a.attrs['href'] print(data)	
	L06-B-3-JSONscrape.py	using the urllib.request package pulling JSON content from a url and converting it to python obj:  data = json.load(ur.urlopen(url))	
	L06-B-4-CSVscrape.py	using pandas to pull a url from the web and putting it directly into a dataframe:  adult_df = pd.read_csv(url, header=None) print(adult_df.head())  apply column names	
	L06-C-1-Aggregating.py	reading in url and converting it directly to a pd df: pm2_file = pd.read_csv( <a href="https://library.startlearninglabs.uw.edu/DATASCI400/Datasets/BeijingPM2_IOT.csv">https://library.startlearninglabs.uw.edu/DATASCI400/Datasets/BeijingPM2_IOT.csv</a> , parse_dates=['TimeStamp'], infer_datetime_format=True)	

		creating a subset df for attributes in question: <pre>pm2_df = pm2_file[(pm2_file['Attribute'] == 'precipitation')                     (pm2_file['Attribute'] == 'TEMP')                     (pm2_file['Attribute'] == 'HUMI')].copy()</pre> using .copy function to create a copy of df instead of replacement time as an index instead of cell values: <pre>pm2_df = pm2_df.set_index(['TimeStamp'])</pre> counting number of observations by: <pre>print(pm2_df.groupby(pm2_df.index.year).count())</pre>	
	L06-C-2-Interpolate.py	dropping rows with missing values grouping by attribute upsampling/downsampling filling in NaN by interpolation methods (linear, polynomial, etc.)	
LESSON 7	L07-Clustering.py		
	L07-1-KMeans_Incomplete.py		
	L07-2-KMeansNorm_Incomplete.py		
	L07-3-KMeansNorm_Incomplete.py		

## JSON

**Author:** nataliegmoore@gmail.com

### JavaScript Object Notation (JSON)

- text-based structure for storing data
- human readable
- language independent
- widely used
- always in single quotes

When is it used?

- transmitting data between web servers and apps
- scraping info from internet via API
- easily storing nested, complex data
- EAV data structures with key/value pairs

JSON objects

- can have the data types of
  - integer (doubles, floats)
  - strings - double quotes
  - boolean
  - null (empty)
- maps to python dictionary
- typically collection of values related to ONE item
- unordered set of key/value pairs
- values are separated by commas
- begins and ends with {curly brackets}

ex JSON object:

```
{ "customer":
  [
    { "name": "Steven", "city": "Seattle" },
    { "name": "David", "city": "London" }
  ]
}
```

```
}  
    ]  
}
```

JSON arrays

- same dtypes as above
- maps to a python list
- ordered collection of values
- values separated by commas
- begins and ends with [square brackets]
- values accessible by index number

ex JSON array:

```
{  
    "name": "Steven",  
    "age": 27,  
    "siblings": ["Anna", "Peter", "Lowell"]  
}
```

---

`json.loads(json_data)`

- converts JSON obj to a python dict

`json.dumps(python_data)`

- converts python dict to json object

`json.dump(python_data)`

- converts python dict to json obj and writes to file
- ex:
  - with `open("test_file.txt", 'w')` as outfile:  
`json.dump(python_data, outfile)`

`json.load(json_data)`

- converts json obj to python dict and writes to file
  - ex:
    - with `open("test_file.txt", 'w')` as outfile:  
`json.load(json_data, outfile)`
- 

HTML documentation:

[https://www.w3schools.com/html/html\\_basic.asp](https://www.w3schools.com/html/html_basic.asp)

## EAV

**Author:** nataliegmoore@gmail.com

### Entity-Attribute-Value

- data model (table format)
- efficiently stores sparse data
- only non-empty values are stored
- one entity can have many attributes
- each attribute is stored as a key-value pair

When to use?

- for sparse, heterogeneous data
- when there are many attributes for one entity
- when attribute values are constantly changing

- when storing data in one row would leave blank attributes

>Translate the following table to EAV format:

employee_id	first_name	last_name	department
1	Ren	Stimp	IT
2	Rick	Morty	IT
3	Thelma	Louise	
4	Harold	Kumar	Marketing

entity	attribute	value
1	first_name	Ren
1	last_name	Stimp
1	department	IT
2	first_name	Rick
2	last_name	Morty
2	department	IT
3	first_name	Thelma
3	last_name	Louise
4	first_name	Harold
4	last_name	Kumar
4	department	Marketing

## Dummy Variables

**Author:** nataliegmoore@gmail.com

created so we can turn non-integer columns into integer columns

- after you decode a column from random integers (which don't mean anything and can be confusing) into categories, we still want ints. but they should mean something

created also that if you don't know the category name of the categorical data, you can "see" the category if all the dummy variables are 0 for that column,

ex:

Gender: Male, Female, Male, Female, Female, Male, Nonbinary

Male: 1, 0, 1, 0, 0, 1, 0

Female: 0, 1, 0, 1, 1, 0, 0

Nonbinary: 0, 0, 0, 0, 0, 0, 1

- we want **linear independence, not dependence**. so make sure you leave one category without a dummy variable.

**Disjunctive table:**

- table that has a dummy variable for each category in a categorical column

## SO FAR general notes

**Author:** nataliegmoore@gmail.com

- Extract your data:

- always start by acquiring the dataset
  - then use pandas to work with the variables
  - and use numpy to work with the numeric values
- Transform your data into workable sets:
  - identify outliers and remove them
  - change missing values to nans then mean values
    - or, change the missing values to 0
    - or, change the missing values to the most common value
    - or, delete the missing values
  - normalize the variables

`DATASET.dtypes`

- tells you the category of each colum, or dimension

`DATASET.head`

- gives you a preview of the dataset matrix

`DATASET.shape()`

- gives you the dimensions of the dataset

`DATASET.loc[:, "column"].unique()`

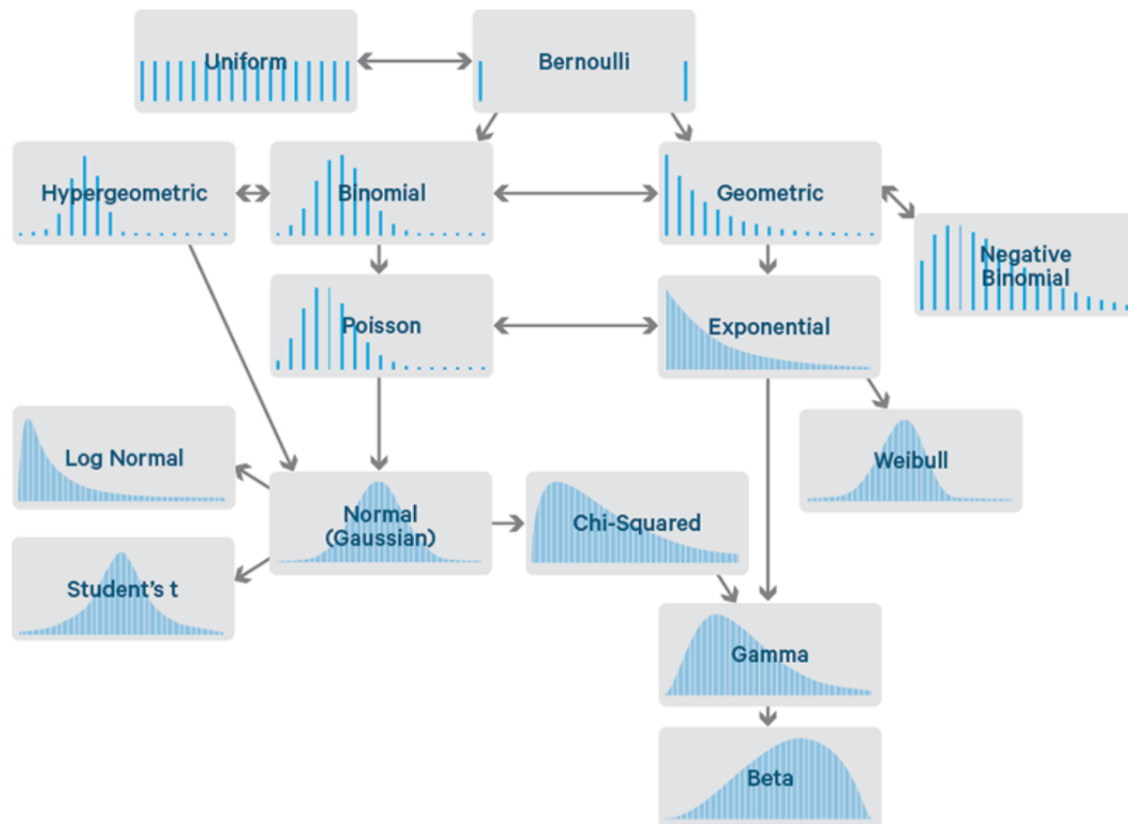
- gives you all the unique values of a column

`DATASET.loc[:, "column"].value_counts()`

- gives you the count of each unique value in a column

## Distribution Chart

Author: nataliegmoore@gmail.com



## Normalization

- "scaling" a variable
- only applies to numeric vars
- essential to data engineering

#### MIN-MAX METHOD

- finding the min and max values of the vars set and setting them to be 0 and 1
  - every other value is in between 0 and 1
- doesn't assume anything about the var distribution so any distribution works
- heavily influenced by outliers = [

#### Z-NORMALIZATION METHOD (aka STANDARDIZATION)

- mean value of var set equal to 0
    - makes the std equal to 1
  - done by subtracting the mean value from each value in var set and dividing by std of var set
    - such that mean is (mean - mean = 0) and std = 1
    - doesn't work for std = 0 obviously
  - assumes normal distribution
  - isn't influenced by outliers as much
- 

consider when normalizing:

- combining methods is unnecessary
- binary vars can be normalized, but not with min-max
- use same method for all vars
- when using a sample, use same min/max (aka mu/sigma) for normalizing all vars
- normalization can be reversed if u keep the parameters

## Binning

Author: nataliegmoore@gmail.com

- grouping values of a variable together and subbing them with a single value
  - usually an integer
- groups = bins
- loses part of the og variable
- useful for summarizing a variable into a more compact form
- bin boundaries can be predefined or selected automatically

#### STANDARD BINNING METHOD

1. Define number of bins (N)
  2. Find width of each bin:  $W = (\max(x) - \min(x)) / N$
  3. For each bin i:
    1. calculate the boundaries of each bin  $i \Rightarrow \min(i), \max(i)$
    2. find all data points in x belonging to  $[\min(i), \max(i)]$
    3. assign value i to these data points  $\Rightarrow y$
    4. repeat for each bin
  4. For elements of x equal to  $\max(N)$ , assign value N
  5. Output boundaries  $\min(x)$ ,  $\max(x)$  and bin values y
- 

consider when binning:

- number of bins matters a lot, choose wisely
- try various scenarios before committing to a single one
- binning is not reversible as a process



# Pandas

Author: nataliegmoore@gmail.com

- data frames are rectangular data set
  - equal row and column length
  - columns have headers
  - all values in a column are the same type
- look at labs for ex

## Data Types

Author: nataliegmoore@gmail.com

### primitive (scalars)

- **int** = integers
- **float** = real numbers with decimals (up to a certain precision)
- **NaN** = not a number like null or missing value
- **inf** = infinity
- **float of inf** = larger than any float
- **float of minus inf** = smaller than any float
- **bool** = Boolean (true or false)
  - 1 = True, 0 = False
- **string** = text

numbers in quotes are strings, not numbers

- you can add multiple types of scalars together:
  - ex: 7 + 3.0 + True = 11.0
    - adds together an int, float, and boolean

### containers and organizers of multiple scalars (data structures)

- **list** = ordered list of items
  - can have have different data types
  - ex: City = ['USA', 704]
  - City.append('Seattle')
    - returns City = ['USA', 704, 'Seattle']
- the **dimension** of a data structure is how many rows or lines it contains
- a **rectangular** data structure contains lists of equal length, so you can represent it as a table (called a **data frame**)
- REMEMBER THAT PYTHON INDEXING STARTS AT 0.
  - ex: Cities[2][3] returns the *third* list and *fourth* cell of that list within the data structure called Cities
- **dict** = dictionary
  - CityNamesByCountry = {'USA':['San Fransisco', 'Seattle'], 'Japan':['Kyoto', 'Tokyo'], 'Taiwan':['Taipei'], 'France':['Paris']}

### arrays and data frames

- numpy is useful for creating arrays
- pandas is useful for creating data frames (df)

---

### null or missing values

- null is a placeholder value
- null is represented by object called **none** or **nan**(float)
- need to replace "placeholder values" with some real data prior to data analysis
- missing value ex:
  - empty text: ""

- o blank text: " " (contains space character)
- BEWARE
  - o "NULL" is not NULL value
  - o "None" is not None value
  - o "nan" is not nan value
    - float("nan") = NaN which is used to represent missing values
- example problem: Student misses exam for the quarter (null grade). At the end of quarter, what to do with null grade for that exam?
  - o Remove the null and calc. average grade without it
  - o Replace null with a 0 and calc. average grade
  - o Replace null with class average for that exam

## Data Flow Diagram

**Author:** nataliegmoore@gmail.com

- required for data processing
- a defined language
- **SSADM** (structured systems analysis and design method)
- **DFD** (dataflow diagrams)
- four components:
  - o Terminator
  - o Store
  - o Process
  - o Data Flow

### DFD Symbols

Complete Rectangles = start or terminate process

—either generate or consume data.

Rectangles without sides = stores, like databases.

Ellipse = a process that transforms data.

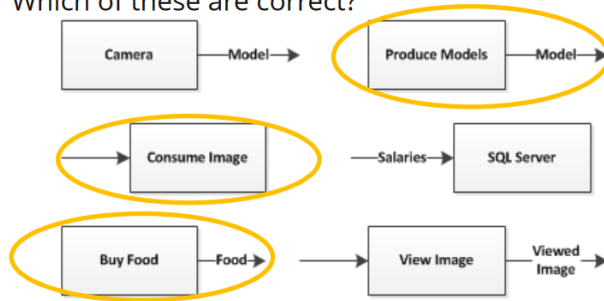
Arrow = data.



- the data arrow
  - o arrow represents data or data flow, labeled by name (noun) of data
  - o every data flow component must have *at least* one arrow
  - o ex: lunch, location, price, etc.
- DFD process ellipse
  - o ellipse represents taking data from one or more sources, transforming the data, and outputting the data
  - o a process must have at least one input *and* output data arrow
  - o labeled as a verb
  - o ex: brighten image, serve lunch, buying something, etc.
- DFD Terminator complete rectangle
  - o process that either generates or consumes data
    - verb
  - o ex: get data from Internet, view data in Monitor, Generate images, etc.

## Data Flow Practice: DFD Terminator

Which of these are correct?



- DFD Store rectangle without sides
  - a place where data is stored
    - noun
  - always has *at least* one input and output arrow
  - describes nature of DATA not DATABASE
  - ex: text file name, website, database

## my data flow diagram

goal: Sending a spacecraft from Earth to Titan. Predicting a viable landing site for a spacecraft traveling from Earth to Titan, Saturn's 6th and largest moon. A viable landing site must be reasonably flat and solid within a certain radius of the predicted site. This prediction assumes the spacecraft is already built and physically capable of the task and has the ability to transmit data to Earth for the entire journey, including post-landing.

data set(s) needed:

- launch
  - weather on/above Earth at time/place of launch
  - location of any obstacles such as
    - atmospheric crafts
    - orbital satellites including large debris
    - the moon
- journey
  - trajectory of spacecraft in relation to Saturn/Titan
  - trajectory of obstacles
    - planets and their moons
      - Mars, Jupiter
    - asteroids
      - asteroid belt between Mars and Jupiter
      - rouge asteroids
- landing
  - location of Titan relative to Saturn and the spacecraft
    - orbital path of Titan rel. to Saturn
  - location of landing site relative to spacecraft
    - rotational vel. of Titan
  - weather on/above Titan's surface
  - topography of Titan's surface
  - composition of Titan's surface
  - density of Titan's surface
  - uncertainty of landing radius
    - radius calculated

### DATA ARROWS

- orbital path of Titan rel. to Saturn
- rotational velocity of Titan
- weather (atmospheric density, wind patterns, storms current and future) patterns on Titan
- topography of Titan
- density map of Titan
- composition map of Titan

### PROCESS ELLIPSES

- gather orbital path data of Titan
- gather rotational velocity of Titan

- gather atmospheric density, wind, storm data from Titan
- gather topography from Titan's surface with spacecraft-level resolution
- gather composition/density of Titan's surface with spacecraft-level resolution

- 

#### TERMINATORS

- generate orbital/rotational velocity models of Titan as a whole
- generate atmospheric model of Titan
- generate topographic map of Titan (spacecraft-level res.)
- generate composition and density map of Titan (spacecraft-level res.)

#### STORES

- predictive model of orbital/rotational velocity of Titan
- predictive model of Titan's atmospheric conditions vs. time
- GIS map of topography
- GIS map of composition
  - GIS map of density

## Relational Algebra

SourceURL: <https://en.wikipedia.org/wiki/Tuple>

Author: nataliegmoore@gmail.com

#### TERMINOLOGY

- Table
  - part of a database
- Relation
  - A table where rows are unique
- Tuple
  - ex: single, double, triple, quadruple, etc.
  - a finite ordered list (sequence) of elements
- Arity
  - ex: unary, binary, ternary, quaternary
  - the number of arguments a function takes
- Closure
  - Operation on a type produces a value of that same type
  - ex: Natural Numbers have closure under + and \* ( $3 * 5 = 15$ ) Natural Numbers do not have closure under - or /;  $5 - 3 = -2$
  - ex: A set is **closed** under an operation if performance of that operation on members of the set always produces a member of that set.
- Procedural
  - step-by-step solution to a problem or goal
- Declarative
  - stating the end goal without describing how to achieve it
- Relational Algebra
  - algebra that describes relations as operands (arguments) and results
- Relational Calculus
  - calc that uses relations as operands (arguments) and results (SQL)
- Selection
  - $\sigma$  (sigma);  $\sigma(R)$ ;
  - SELECT \* FROM <table name> WHERE Column1 = 1
- Projection
  - $\pi$  (pi);  $\pi_{c1, c2, ..., cn}(R)$
  - SELECT Column1, Column 2 FROM <table name>
- Rename
  - P (rho)
  - as
- Union
  - u

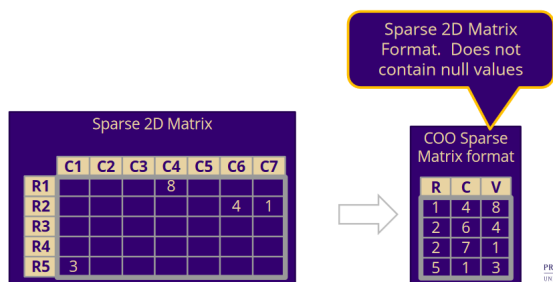
- $A \cup B$ ;  $A = \{1, 2, 3, 5\}$ ;  $B = \{0, 2\}$ ;  $\{1, 2, 3, 5\} \cup \{0, 2\} = \{0, 1, 2, 3, 5\}$
- Intersection
  - $\cap$
  - $A \cap B$ ;  $A = \{1, 2, 3, 5\}$ ;  $B = \{0, 2\}$ ;  $\{1, 2, 3, 5\} \cap \{0, 2\} = \{2\}$
- Difference
  - $\setminus$
  - $B \setminus A = B - A$ ;  $\{0, 2\} - \{1, 2, 3, 5\} = \{0\}$
- Product
  - $\times$
  - $A \times B$   $A = \{1, 2, 3, 5\}$ ;  $B = \{0, 2\}$ ;  $\{1, 2, 3, 5\} \times \{0, 2\} = \{\{1, 0\}, \{2, 0\}, \{3, 0\}, \{5, 0\}, \{1, 2\}, \{2, 2\}, \{3, 2\}, \{5, 2\}\}$
- Join2
  - $\bowtie$
  - $B \bowtie A$ ;  $\varphi: A \rightarrow B$ ;  $A = \{1, 2, 3, 5\}$ ;  $B = \{0, 2\}$ ;  $\{1, 2, 3, 5\} \bowtie \varphi \{0, 2\} = \{\{1, 0\}, \{2, 0\}, \{3, 0\}, \{3, 2\}, \{5, 0\}, \{5, 2\}\}$
- Division\
  - $\div$
  - $A \div B = C$ ; Project to show me the columns in A that are not in B; Select to show me the tuples in A that are a superset of the a tuple in B.

## Matricies

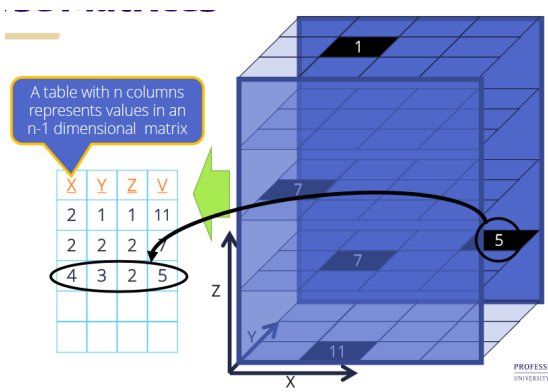
**Author:** nataliegmoore@gmail.com

Matricies and Dataframes are very similar, except:

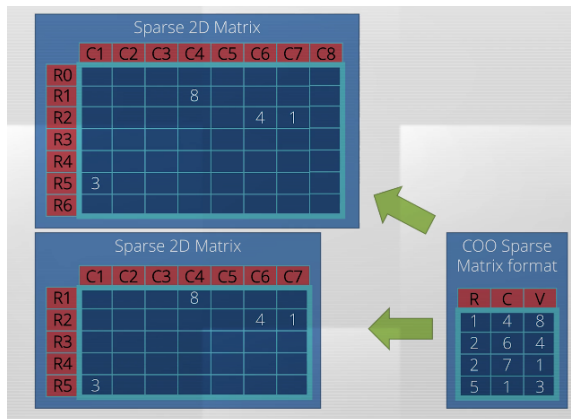
- a matrix contains only one data type
- generating a random matrix:
  - with the lowest number as 1, the highest number as 9, and a size of 5 rows and 7 columns:
  - `np.random.random_ints(low=1, high=9, size=[5,7])`
- a **sparse** matrix has almost all elements as null (missing)
  - null values could mean:
    - we could enter a value if it exists
    - a value does not exist for that element
    - a default value
  - traditional matrix would be wasteful if most elements are null
  - so we use a **Sparse 2D Matrix Format**:
    - **R** = row
    - **C** = column
    - **V** = value



- **sparse matrices can also be *n-dimensional***
  - the below ex. represents a 4 dimensional space
    - the "value" column is the 4th dimension



- if you don't know the dimension of the original matrix, use the extreme values of the rows and columns:



TAKEAWAY: most data sets should be thought of as sparse, multi-dimensional matrices

- DS work is largely to turn
  - observations into rows
  - columns into dimensions

## Intro to Python

Author: nataliegmoore@gmail.com

- **IDE** = Integrated Development Environment
- Spyder is an IDE designed for ds
- **numpy (np)** = numerical python
  - numpy is a **package** (collection of functions and other code you might use)
- variable explorer in Spyder is equivalent to the workspace in MATLAB
- `type(x)` to get the type of a variable x

to determine the **current working directory** (cwd):

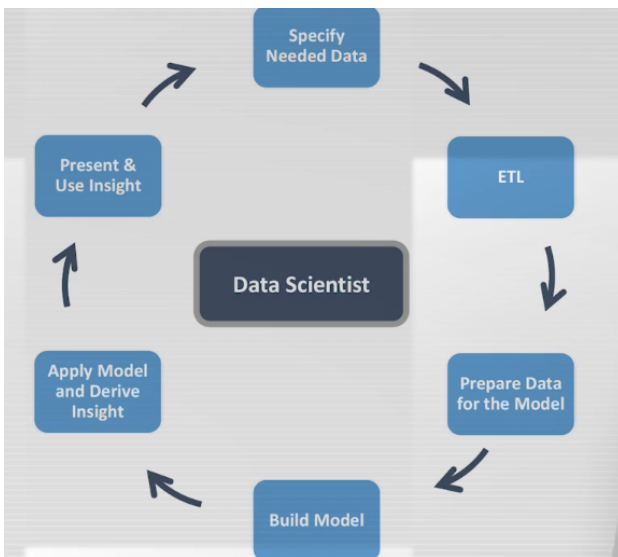
```
import os
os.getcwd() #note: the () at the end of a function tells python you want to call that function. without it, you just tell python you want to reference it.
```

to define an array named x:

```
x = np.array([0,1,2,3])
```

# Data Science Cycle

Author: nataliegmoore@gmail.com



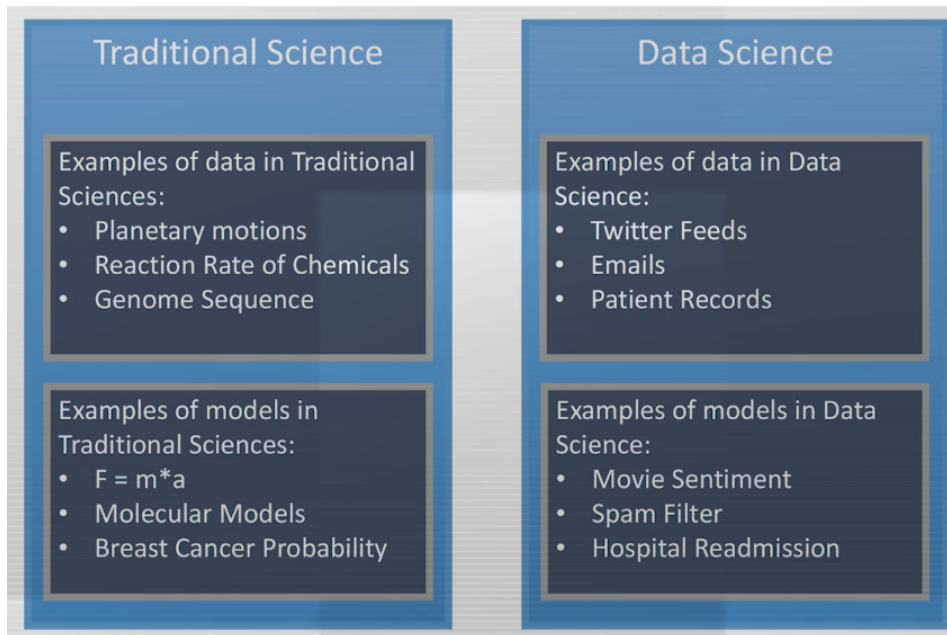
- ds cycle starts with getting the data (usually already done for you)
  - except with DOE (design of experiment)
- ETL (extract, transform, and load)
  - **extracts** data from one or more dbs
  - **transforms** the data for analytical processing
    - may complete next step in cycle: preparing the data for modeling
  - and **loads** the data into a warehouse for staging analytical data
  - ETL developers focus on transformations like changing data types, correcting time zones, joining tables, etc.
  - ETL is usually about 2/3 of ds work
- preparing the data for modeling
  - usually more involved than ETL
  - prep includes binning, decoding, category consolidation, one hot encoding, normalization, outlier removal, and data imputation
  - ETL and prepping the data is ds bulk work, while modeling is usually given to BI developers
- building the model
  - refer to the four analytics
  - "how did it happen?" is not one of the ds questions
    - the model is based on correlation and NOT on causation

# What is Data Science?

Author: nataliegmoore@gmail.com

- data science is the utilization of the scientific method outside of traditional science practices
  - there is a new abundance of data outside of traditional sciences
  - we've developed new tools to investigate all these data
- data science's general (yet poor) "definition" is that ds is the art of gaining insights from data
  - same as the def. of business intelligence (BI) so that is only part of the def. of ds
- scientific process/method
  - observe and collect data
  - a hypothesis is formed to explain observations
  - hypothesis is then tested against known results
    - called **Falsification** (trying to disprove the hypothesis)
      - if a hypothesis is not falsifiable than it is not a hypothesis
  - we can then use the hypothesis to predict new observations
  - if the hypothesis predicts new observations correctly, it becomes a theory
    - this is the continuation of falsification

- if the hypothesis does not accurately predict new observations, we need to come up with a new hypothesis
- the entire scientific method is a cycle where hypotheses continually improve
- **schema** = coordinates of the *space*
- **table, columns, attributes, dimensions, variables** = coordinates of *data points*
- **cases, rows, tuples** = data points
- **abscissa** = x-axis or input
- data model of a 2d space = an algebraic representation of the data points
  - ex:  $y = 1 + 0.5x \pm \text{uncertainty}$
  - models represent some observations better than others (hence the uncertainty)
  - data models ~ hypothesis with limitations in the form of uncertainties



- predictive hypotheses/models are often more useful than explanatory hypotheses.
  - ex: quantum mechanics. we don't know the mechanism of how it works but we can predict what will happen relatively
- data scientists are interested in the predictive value of a hypothesis rather than its explanatory value
- What is data???
  - an observation is always a datum (singular of data)
    - "I see a chair" vs. "chair"
    - "Patient has diabetes" vs. "diabetes"
  - unstructured data does not exist
- Most of data scientist's time is spent on ETL (extract, transform, load) and data prep
- There are four analytic types in ds:
  - descriptive analytics (what happened?)
  - diagnostic analytics (why did it happen?)
  - predictive analytics (what is likely to happen later?)
  - prescriptive analytics (what to do about it?)

