**Python Notes**

Data visualisation next steps

* Plotting tools:
  + [Creating 3D plots using Plotly](https://plot.ly/python/3d-scatter-plots/)
  + [Creating interactive visualizations using bokeh](http://bokeh.pydata.org/en/latest/)
  + [Creating interactive map visualizations using folium](https://folium.readthedocs.io/en/latest/)
* The art and science of data visualization:
  + [Visual Display of Quantitative Information](https://www.amazon.com/Visual-Display-Quantitative-Information/dp/0961392142)
  + [Visual Explanations: Images and Quantities, Evidence and Narrative](https://www.amazon.com/Visual-Explanations-Quantities-Evidence-Narrative/dp/0961392126)

**DATA SOURCES**:

you can start your search at these sites:

* [Data.gov](https://www.data.gov/) - A directory of government data downloads
* [/r/datasets](https://reddit.com/r/datasets) - A subreddit that has hundreds of interesting data sets
* [Awesome datasets](https://github.com/caesar0301/awesome-public-datasets) - A list of data sets hosted on GitHub
* [rs.io](http://rs.io/100-interesting-data-sets-for-statistics/) - A great blog post with hundreds of interesting data sets

Data Frames

1. Multiple datatypes
2. dypes - gives column types
3. Extracting row - using loc[] on data frame - includes both indices
4. Extracting column - using dataframe[“name”] - gives a series object - can use arithmetic operations on cols - Can find stats - using min(), max(), mean() -
5. Add a new column - dataframs[“new column name”]
6. dataframe.shape - tuple of size
7. dataframe.columns - gives index object of colnames - can covert to list using tolist() method
8. Can combine multiple columns
9. datatypes
   1. object == string
   2. int
   3. float
   4. datetime
10. dataframe.sort\_values(“col name”, inplace = bool, ascending = True)
11. check null val — pandas.isnull(<SeriesName>) - series of True/False
12. ~ = toggle a series or list
13. series.mean() - mean
14. len(series) - size
15. sum(series) - total
16. pivot\_table — dataframe(index = <>, values = <>, aggfunc = np.mean) — index = column we want to grip values according to. values = where we are performing calculation, aggfunc == function on values column
17. dataframe.dropna(axis, subset, inplace …) — axis:0 - drop rows, axis:1, drop columns, subset - if drop rows - then mention cols, if drop along cols - then mention rows (eg. drop rows specific to the columns)
18. dataframe.iloc[<>] - get rows with index locations (integer locations - indexing works the same way as for list and arrays). .iloc[0:10] - is 0 to 9, iloc[4] - is 5th integer location - BUT - .loc[0:10] is labeles from 0 to 10 both included
19. Two arguments in loc and iloc can be used in same way - but will return series or df. (slicing for iloc in same way as for bumpy array, and for loc in same way as labeles )
20. apply a function - dataframe.apply(<function>, axis) - AXIS = 1 — apply on rows, axis = 0 (default) - applies on cols
21. Series.Unique() - unique elements in the series
22. Python’s usual arithmetic operations can be used on the whole series. This is data alignment of pandas. In a DATAFRAME - a values is attaches to index and column. In SERIES - a values is attached to index
23. Groupby and agg function to apply function on filtered data frame in directly

————

Series

1. default series is with integer index
2. series can be with string index as well -
3. Creating a Series - pandas.Series(data = <>, index = <>, …) — data can be any pandas data type and index can be string or any integer. If index is not provided, default 0, 1, 2, .. is assumed
4. The integer index is preserved - elements can be accessed with both integer index (like lists) and with string index (acts as keys in dictionary) — integer index acts like list - [a:b] — index b is not included
5. Series.reindex(index = index, …)
6. Series.short\_index() — shorts the series by indices
7. Series.short\_values() - shorts the series by values
8. In both the above methods data alignment is taken care of
9. Series is derived using a numpy ndarray - can use vectorised operations - add, sub, mean - on the entire series
10. Comparing and filtering can be done in same way as in numpy ndarray

——

DataFrame

1. dataframe[0:5] - from row 0 to 5 (5 not included) and returns a df
2. dataframe[5:] - from row 5 to end, returns a df
3. dataframe.loc[5] - row at index 5
4. dataframe.loc[[4,6]] - row at index 4 and 5
5. When a row is retrieved - a Series is returned, for more than one rows a df is returned
6. .loc is necessary when only one row is retieved - .loc can take —
   1. integer
   2. list of ints
   3. string or list of strings
   4. boolean
7. CANNOT USE DIRECT SLICING WHEN NON-CONTINUOS ROW IS SELECTED
8. Can set custom index, same way as for Series - using set\_index() method
9. lambda function: dataframe.apply(lambda x: np.std(x), …)

———————————

* Figure out the most common dessert people eat.
* Figure out the most common complete meal people eat.
* Identify how many people work on Thanksgiving.
* Find regional patterns in the dinner menus.
* Find age, gender, and income based patterns in dinner menus.

————————————

* Use a hexagonal bin plot to visualize the columns that had dense scatter plots from earlier in the project.

———————————

* Determing wheter there's a correlation between class size and SAT scores
* Figuring out which neighborhoods have the best schools
  + If we combine this information with a dataset containing property values, we could find the least expensive neighborhoods that have good schools.
* Investigating the differences between parent, teacher, and student responses to surveys.
* Assigning scores to schools based on sat\_score and other attributes.

———————————

STAR WARS:

Here are some potential next steps:

* Try to segment the data based on columns like Education, Location (Census Region), and Which character shot first?, which aren't binary. Are they any interesting patterns?

———————————

types

type function

type conversion

PEMDAS

console - REPL

lists - append, accessing, slicing

file:

open

read

split a string

list of lists

in statement

Variable scope

1. local
2. global - can access cannot change
3. built in function
4. with global keyword - can create and assign values in separate lines and can change the value

—————————————

numpy

1. all the elements in an ndarray should be of same type
2. genfromtxt — read csv- also mention delimiter, dtype, skip\_header, .. — returns numpy.ndarray
3. numpy.array - takes a list or list of lists as an argument — returns numpy.ndarray
4. numpy.shape - size of the numpy array as a tuple
5. numpy.array.dtype - returns type of data in the bumpy array
6. indexing - a[1,2] — gives row 1, column 2 element — indices start from 0
7. comparison - (==), (&), (|)
8. ndarry.astype(<type>) - converts all the values to type
9. math operations - ndarray.sum(), ndarray.mean(), … | for matrix specify axis argument
   1. ndarray.sum(axis=1) - sum of all rows
   2. ndarray.sum(axis=0) - sum of all columns