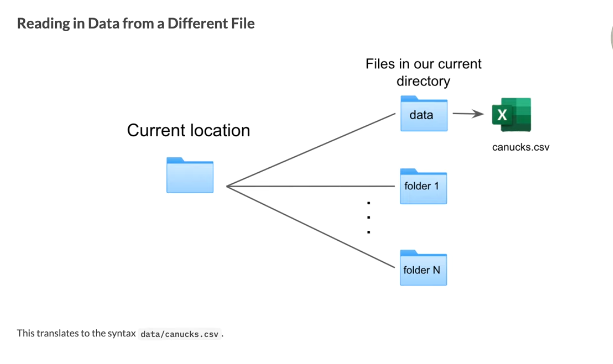
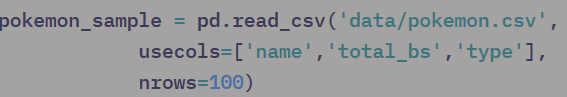
DATA 301 Python Module 2 Notes

Module 2

Reading different file types

* Pd.read\_csv(‘hyperlink’)
  + Then: Candybars.head(‘however many rows you wish to display’)
    - E.g. candybars.head(5)
* Reading a text file:
  + E.g. candybars=Pd.read\_csv(‘(file location)/.txt file’)
  + Candybars.head()
  + BETTER ALTERNATIVE: candybars=pd.read\_csv(‘.txt file’, delimiter=’\t’)
    - Defines how column values are separated
    - delimitors are also: ; , \t (spaces)
  + excel files—specify the sheet you want
    - e.g. candybars=pd.read\_excel(‘data/foods.xlsx, sheet\_name=’chocolate’)
* DATA LOCATION:
  + 

Arguments for Reading Data

* Index\_col
  + Indicates which column will be acting as index label
  + Index\_col=0
* Header
  + Specifies when dataframe begins
    - E.g. header=2
* Nrows
  + Only want to load in part of dataframe
  + E.g. nrows=7 -only loads 7 rows from df
* Usecols
  + Specifies which columns to load
  + E.g. usecols=[0,1,10]
  + E.g. 2: usecols=[‘country’, ‘city’, ‘suburb’]
* 

Column Renaming and Dropping

* (Columns={‘old column name’ : ‘new column name’})



* Column dropping (deleting)—e.g. candy.drop(columns=[’coconut’, ‘nougat’])

Column Arithmatic and Creation

* Cereal[‘fat’] \*1000 – multiplies all values in ‘fat’ column by 1000
* Cereal[‘sugars’] / Cereal[‘cups’] = divides ‘sugars’ by ‘cups’ data
* Column creation: oz\_to\_g
  + Cereal[‘weight’] \* oz\_to\_g
  + Full code: cereal= cereal.assign(weight\_g=cereal[‘weight’] \* oz\_to\_g
    - .assign—can be used to assign a new column combining the calculations (in this case, ‘weight\_g’)
    - 

Data Filtering

* Get rid of unwanted rows or analyse rows with particular column value
* E.g. cereal(cereal[‘protein’] >4)
* Cereal(cereal[‘protein’] == 4) -- == used for comparison
* Multiple condition filtering - “and”
* Cereal(cereal[‘protein’] >=4)
* For multiple conditions: cereal[(cereal[‘protein’] >=4) & (cereal[‘protein’] <=5)]
  + E.g. mighty\_pokemon = pokemon[(pokemon['attack'] >100) & (pokemon['defense'] >100)]
* Filter out 2 different columns: cereal[(cereal[‘mfr’] == ‘Q’) & (cereal[‘protein’] > 4)]
  + For “or”, use |
* Tilde – for opposites
  + E.g. cereal[~(cereal[‘protein’] > 4)]—shows all values that are not greater than 4

Conditional Value Replacement

* .loc function can filter conditions
* E.g. cereal.loc[cereal[‘mfr’] == ‘Q’]
* Can do this in 3 steps
  + Step one : use .loc to find the rows meeting certain conditions:
    - E..g cereal.loc[cereal[‘mfr’] == ‘Q’]
  + Indicate which column we wish to access:
    - Cereal.loc[cereal[‘mfr’] == ‘Q’, ‘mfr’]
  + Step 3: assign a value
    - Cereal.loc[cereal[‘mfr’] == ‘Q’, ‘mfr’] = ‘Quaker’
  + Replacing with inequalities
    - Cereal.loc[cereal[‘protein’] >=3, ‘protein\_level’] = ‘high’
    - Cereal.loc[cereal[‘protein] <3, ‘protein\_level’] = ‘low’
  + Creating new columns
    - 
    - Edits existing dataframe
* E.g. create a new column in the df called ‘Base\_score’ by assigning values >=500 from column total\_bs as ‘strong’ and values <500 as ‘weak’ pokemon
  + 

Chaining Notation

* Allows us to do multiple actions in a single line of code
  + e.g. 
    - pro-tip: use a new line for each verb in a chain to make it easier to read
      * e.g. 
  + coding preferences
    - comments are important to make sense of chaining for others

rename and create column example :

* plot\_df = pd.DataFrame(pokemon.rename(columns={'capture\_rt': 'capture\_rate'})

.assign(AD\_total=pokemon['defense'] + pokemon['attack'])

Grouping and Aggregating

* which manufacturer has the highest mean sugar content
  + cereal[cereal[‘mfr’] == ‘K’].mean()[[‘sugars’]]
  + groupby—group data based on a specific column
    - e.g. mfr\_group= cereal.groupby(by= ‘mfr’)
  + .agg – aggregates statistics
  + Cereal.agg([‘max’, ‘min’, ‘median’])
  + Aggregating groupby objects
    - Mfr\_group.agg([‘max’, ‘min’, ‘median’])
      * You get value for each group and statistic
    - Extra fance aggregation
      * Mfr\_group.agg({“calories”:[‘max’, ‘min’], “rating”: [‘sum’], “sugars: [‘mean’, ‘median’]})
        + Wrap everything in {} and use colon to separate columns form statistics

Plotting with Altair

* Altair recognises: ordinal (O), nominal (N), quantitative (Q), temporal (T)
  + Can be used to make histograms
  + Alt.bin—specify max number of bins
* Reset index—df.reset\_index

1. Groupby object and calculate mean
2. Reset index
3. Plot using Altair

* Sort argument—allows you to sort by a value – e.g. sort=”y”