Final Project Information

## Overview

For the final project, you will continue to work with the commodity data. There have been several exercises you have completed to prepare you for this moment. The project involves reading data from a file, organizing it into individual data records, selecting data records of interest through user interaction, consolidating the selected data records and generating graphical representations of the data. The only new aspect is the use of **plotly** to generate grouped bar graphs. I strongly suggest that you immediately follow the instructions in the appendix to add **plotly** to your Anaconda installation and then run the "Grouped Bar Plot" example to verify its operation. Once you verify that **plotly** works in your Anaconda installation, you will be on your way to easily generating beautiful and interactive grouped bar plots.

Follow all the standard rules for writing iconic Python …

* Make your code readable and efficient.
* Do not establish variables that have no purpose.
* Do not traverse a collection using indices. If you really need an index, use **enumerate()**
* Use list comprehension and, if you find a use for them …
  + the **filter()** function
  + the **map()** function

A simple user interface is required and demonstrated below. Data validation is largely optional because I want you to focus on the data processing part. The most important things are (1) the correctness of your results and (2) the quality of your code. This is your opportunity to demonstrate everything you learned this semester. Part of this will be figuring out how to use **plotly** to create all of the required elements in a grouped bar plot. As you research the **plotly** documentation, it's important to remember that **plotly** is available for many languages … so make sure that whatever you try is from documentation that speaks to the Python version of **plotly**.

## Development Method

I continue to remind you that the way to succeed at writing code to solve this problem is to work one small step at a time, test at each step and do not move forward until you have verified that all code is correct. Putting your final solution together in a linear sequence of verified linear steps is a proven method. You should not, under any circumstances, write the entire program before testing … this is a recipe for disaster. You may be able to save some time by testing each portion separately with some hard-coded data … that's a fine way to proceed. But don't attach your successful experiments to your final product until you have completed the steps that precede it.

## My Approach (yours may be different)

Here are the steps I used to complete my version, your version may take a different approach … there are many ways to solve a complex problem.

1. Read the data table from the file and verify that all records have been accurately read into a list of lists <data>.
2. pop off the column headings and slice to obtain a list of locations
3. Traverse <data> and change all
   1. date strings to datetime objects
   2. price strings to float
4. Verify the conversions in <data>
5. Traverse <data> and convert the tabular format to individual records <records>  
   Each row in data will contain five data records, one for each city
6. Verify the data in <records>
7. Extract a list of product strings from the data records without duplicates. Sort the list and print each product and the index associated with it (this is an easy way to implement the user interface … ask the user to type indexes instead of product names and is easily done using enumerate())
8. Ask the user to specify which products are of interest by giving the index numbers. Use the index numbers provided by the user to create a list of product strings to select. Verify the interaction (print the chosen product strings).
9. Extract a list of dates from the data records without duplicates. Sort the list and print each date and the index associated with it. Determine the earliest and latest date available in the data and print them out.
10. Ask the user to specify a start and end date by giving the index numbers. Use the index numbers provided by the user to assign datetime objects to variables representing the start and end times. Verify the interaction (print the chosen start/end dates, make sure the start date precedes the end date).
11. Sort the list of location strings, print each location along with its index.
12. Ask the user to specify the locations of interest by giving index numbers. Use the index numbers to create a list of desired location strings. Verify the interaction (print the location strings for the user to see).
13. Select the data records that meet the criteria given: date range, selected products and selected cities. Check the selected data for correctness <select>.
14. Organize the selected data records in a dictionary with two keys: product and location. Each product and location will refer to a list of prices that spans the selected dates. There's no need to include the dates here. The list of prices is necessary to compute the average price. Verify each list of prices.
15. Traverse the dictionary and replace each list with the average of the prices in the list. Verify the computation.  
    Now you've compiled all the data you need to generate the grouped bar graph.
16. Create a title string for the graph.

Traverse the dictionary and create a list of **plotly** traces

Create a **plotly** layout and figure

1. update the figure layout by providing
   1. a format string for y-axis values
   2. x and y axis titles
   3. a chart title
2. plot the figure to the html file of your choice and view the result in your browser

## Debugging

Spyder can be very useful for debugging.

* The Variable Explorer tab is helpful in showing the state of all of your variables.
* Spyder will allow you to execute just one group of instructions from your source code at a time. Select the code you want to execute and then hit F9.
* Hovering over a line of code will invoke a pop-up with information about the function you are trying to use.
* There are many helpful features I don't even know about. Take a moment to explore.

# APPENDIX

## EXAMPLE INTERACTION without debugging print statements …

==========================

Analysis of Commodity Data

==========================

SELECT PRODUCTS BY NUMBER ...

< 0> Asparagus < 1> Avocados < 2> Broccoli Bunches

< 3> Broccoli Crowns < 4> Cantaloupe < 5> Carrots

< 6> Cauliflower < 7> Celery < 8> Flame Grapes

< 9> Green Leaf Lettuce <10> Honeydews <11> Iceberg Lettuce

<12> Nectarines <13> Oranges <14> Peaches

<15> Plums <16> Potatoes <17> Red Leaf Lettuce

<18> Romaine Lettuce <19> Strawberries <20> Thompson Grapes

<21> Tomatoes

Enter product numbers separated by spaces: 1 4 7 14

Selected products: Avocados Cantaloupe Celery Peaches

SELECT DATE RANGE BY NUMBER ...

< 0> 2017-02-26 < 1> 2017-03-05 < 2> 2017-03-12 < 3> 2017-03-19 < 4> 2017-03-26

< 5> 2017-04-02 < 6> 2017-04-09 < 7> 2017-04-16 < 8> 2017-04-23 < 9> 2017-04-30

<10> 2017-05-07 <11> 2017-05-14 <12> 2017-05-21 <13> 2017-05-28 <14> 2017-06-04

<15> 2017-06-11 <16> 2017-06-18 <17> 2017-06-25 <18> 2017-07-02 <19> 2017-07-09

<20> 2017-07-16 <21> 2017-07-23 <22> 2017-07-30 <23> 2017-08-06 <24> 2017-08-13

<25> 2017-08-20 <26> 2017-08-27 <27> 2017-09-03 <28> 2017-09-10 <29> 2017-09-17

<30> 2017-09-24 <31> 2017-10-01 <32> 2017-10-08 <33> 2017-10-15 <34> 2017-10-22

<35> 2017-10-29 <36> 2017-11-05 <37> 2017-11-12 <38> 2017-11-19 <39> 2017-11-26

<40> 2017-12-03 <41> 2017-12-10 <42> 2017-12-17 <43> 2017-12-24 <44> 2017-12-31

<45> 2018-01-07 <46> 2018-01-14 <47> 2018-01-21 <48> 2018-01-28 <49> 2018-02-04

<50> 2018-02-11 <51> 2018-02-18 <52> 2018-02-25

Earliest available date is: 2017-02-26

Latest available date is: 2018-02-25

Enter start/end date numbers separated by a space: 0 52

Dates from 2017-02-26 to 2018-02-25

SELECT LOCATIONS BY NUMBER ...

<0> Atlanta

<1> Chicago

<2> Farm

<3> Los Angeles

<4> New York

Enter location numbers separated by spaces: 0 1 3 4

Selected locations: Atlanta Chicago Los Angeles New York

500 records have been selected.

## THE GROUPED BAR CHART IS DISPLAYED IN A BROWSER TAB …

Machine generated alternative text:
Produce Prices from 2017-02-26 through 2018-02-25 
$3.00 
$2.50 
$2.00 
$1.50 
$1.00 
$0.50 
$0.00 
Avocados 
Atlanta 
Chicago 
Los Angeles 
New York 
Cantaloupe 
Celery 
Peaches 
Product 

## EXAMPLE INTERACTION including debugging print statements …

==========================

Analysis of Commodity Data

==========================

THE FIRST 8 ROWS FROM THE DATA FILE ...

['Commodity', 'Date', 'Farm', 'Atlanta', 'Chicago', 'Los Angeles', 'New York']

['Strawberries', '2/25/2018', '$1.50 ', '$2.13 ', '$3.03 ', '$2.75 ', '$2.74 ']

['Romaine Lettuce', '2/25/2018', '$0.78 ', '$1.79 ', '$2.09 ', '$1.47 ', '$1.97 ']

['Red Leaf Lettuce', '2/25/2018', '$0.44 ', '$1.42 ', '$1.75 ', '$1.47 ', '$1.65 ']

['Potatoes', '2/25/2018', '$1.60 ', '$3.97 ', '$3.81 ', '$3.99 ', '$5.47 ']

['Oranges', '2/25/2018', '$0.53 ', '$1.60 ', '$1.41 ', '$1.32 ', '$1.88 ']

['Iceberg Lettuce', '2/25/2018', '$0.64 ', '$0.99 ', '$1.41 ', '$1.34 ', '$1.77 ']

['Green Leaf Lettuce', '2/25/2018', '$0.56 ', '$1.42 ', '$1.71 ', '$1.47 ', '$1.57 ']

THE FIRST 8 ROWS OF CONVERTED DATA ...

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 1.5, 2.13, 3.03, 2.75, 2.74]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 0.78, 1.79, 2.09, 1.47, 1.97]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 0.44, 1.42, 1.75, 1.47, 1.65]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 1.6, 3.97, 3.81, 3.99, 5.47]

['Oranges', datetime.datetime(2018, 2, 25, 0, 0), 0.53, 1.6, 1.41, 1.32, 1.88]

['Iceberg Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 0.64, 0.99, 1.41, 1.34, 1.77]

['Green Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 0.56, 1.42, 1.71, 1.47, 1.57]

['Celery', datetime.datetime(2018, 2, 25, 0, 0), 0.32, 1.37, 1.69, 1.94, 1.62]

THE FIRST 20 DATA RECORDS ...

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 'Farm', 1.5]

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 'Atlanta', 2.13]

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 'Chicago', 3.03]

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 'Los Angeles', 2.75]

['Strawberries', datetime.datetime(2018, 2, 25, 0, 0), 'New York', 2.74]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Farm', 0.78]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Atlanta', 1.79]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Chicago', 2.09]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Los Angeles', 1.47]

['Romaine Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'New York', 1.97]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Farm', 0.44]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Atlanta', 1.42]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Chicago', 1.75]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'Los Angeles', 1.47]

['Red Leaf Lettuce', datetime.datetime(2018, 2, 25, 0, 0), 'New York', 1.65]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 'Farm', 1.6]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 'Atlanta', 3.97]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 'Chicago', 3.81]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 'Los Angeles', 3.99]

['Potatoes', datetime.datetime(2018, 2, 25, 0, 0), 'New York', 5.47]

SELECT PRODUCTS BY NUMBER ...

< 0> Asparagus < 1> Avocados < 2> Broccoli Bunches

< 3> Broccoli Crowns < 4> Cantaloupe < 5> Carrots

< 6> Cauliflower < 7> Celery < 8> Flame Grapes

< 9> Green Leaf Lettuce <10> Honeydews <11> Iceberg Lettuce

<12> Nectarines <13> Oranges <14> Peaches

<15> Plums <16> Potatoes <17> Red Leaf Lettuce

<18> Romaine Lettuce <19> Strawberries <20> Thompson Grapes

<21> Tomatoes

Enter product numbers separated by spaces: 5 7

Selected products: Carrots Celery

SELECT DATE RANGE BY NUMBER ...

< 0> 2017-02-26 < 1> 2017-03-05 < 2> 2017-03-12 < 3> 2017-03-19 < 4> 2017-03-26

< 5> 2017-04-02 < 6> 2017-04-09 < 7> 2017-04-16 < 8> 2017-04-23 < 9> 2017-04-30

<10> 2017-05-07 <11> 2017-05-14 <12> 2017-05-21 <13> 2017-05-28 <14> 2017-06-04

<15> 2017-06-11 <16> 2017-06-18 <17> 2017-06-25 <18> 2017-07-02 <19> 2017-07-09

<20> 2017-07-16 <21> 2017-07-23 <22> 2017-07-30 <23> 2017-08-06 <24> 2017-08-13

<25> 2017-08-20 <26> 2017-08-27 <27> 2017-09-03 <28> 2017-09-10 <29> 2017-09-17

<30> 2017-09-24 <31> 2017-10-01 <32> 2017-10-08 <33> 2017-10-15 <34> 2017-10-22

<35> 2017-10-29 <36> 2017-11-05 <37> 2017-11-12 <38> 2017-11-19 <39> 2017-11-26

<40> 2017-12-03 <41> 2017-12-10 <42> 2017-12-17 <43> 2017-12-24 <44> 2017-12-31

<45> 2018-01-07 <46> 2018-01-14 <47> 2018-01-21 <48> 2018-01-28 <49> 2018-02-04

<50> 2018-02-11 <51> 2018-02-18 <52> 2018-02-25

Earliest available date is: 2017-02-26

Latest available date is: 2018-02-25

Enter start/end date numbers separated by a space: 10 11

Dates from 2017-05-07 to 2017-05-14

SELECT LOCATIONS BY NUMBER ...

<0> Atlanta

<1> Chicago

<2> Farm

<3> Los Angeles

<4> New York

Enter location numbers separated by spaces: 3 4

Selected locations: Los Angeles New York

6 records have been selected.

RECORDS SELECTED ...

<0> ['Carrots', datetime.datetime(2017, 5, 14, 0, 0), 'Los Angeles', 0.99]

<1> ['Carrots', datetime.datetime(2017, 5, 14, 0, 0), 'New York', 1.07]

<2> ['Celery', datetime.datetime(2017, 5, 7, 0, 0), 'Los Angeles', 1.94]

<3> ['Celery', datetime.datetime(2017, 5, 7, 0, 0), 'New York', 2.17]

<4> ['Carrots', datetime.datetime(2017, 5, 7, 0, 0), 'Los Angeles', 0.99]

<5> ['Carrots', datetime.datetime(2017, 5, 7, 0, 0), 'New York', 2.17]

2 prices for Carrots in Los Angeles

2 prices for Carrots in New York

1 prices for Celery in Los Angeles

1 prices for Celery in New York

## THE GROUPED BAR CHART IS DISPLAYED IN A BROWSER TAB …

Machine generated alternative text:
Produce Prices from 2017-05-07 through 2017-05-14 
Los Angeles 
New York 
$2.00 
$1.50 
$1.00 
$0.50 
$0.00 
Carrots 
Product 
Celery 

## Installing plotly for your Anaconda environment

* run Anaconda navigator as Administrator
* Click on "Environments" in left pane
* Click on "Update index"
* Choose "Not installed" in the box to the left of Channels button
* Click on "Update index" (to the right of the Channels button)
* Scroll down to find plotly in the list below
* Click on the box to the left of plotly and then …
* Click on the green "Apply" button at the bottom of the page
* This should result in some packages being installed
* There may be another dialog box, click Apply or OK or whatever
* It may take a while for the installation process to complete.

## Python example of using plotly to create a grouped bar graph.

This is a very elementary example. You will need to expand on this to complete the project. For example, notice that the number of categories (number of bars plotted and how they are grouped) is static here. In your application, the number of categories will be dynamic … you will not know ahead of time what the user's choices will be. I suggest making a list of traces by traversing your dictionary (or other final data structure) and append to a list of "traces" that you will create inside the loop. Aspects of the plot that are not shown in this example are: axis titles, formatting for currency, and providing a title for the chart … those are things you will need to research.

Machine generated alternative text:
plotly grouped bar plot 
import plot Iy. offline as py 
import plot Iy. graph_objs as go 
go. Bar( 
tracel 
'giraffes', 
Y C 20, 14, 23], 
name= 'SF Zoo' 
go. Bar( 
trace2 
' giraffes' , 
Y C 12, 18, 29], 
'LA zoo' 
name= 
' orangutans ' 
' orangutans ' 
' monkeys ' 
' monkeys ' 
data 
[tracel, trace2] 
go. Layout( 
layout 
barmode= ' group ' 
fig = go. layout—layout) 
py. plot(fig, filename= ' grouped-bar. html ') 

## My gift to you … may you find a use for it …

This function will useful for printing categories with index numbers. I use it in the example interactions shown above.

Machine generated alternative text:
def : 
Function to print data across three columns 
Argument x is a list of strings 
Indices may be printed by setting enum to the number of 
characters you wish to reserve for the index. 
The wid argument sets the number of spaces for the text 
(20 by default) without printing the index. 
The total number of characters for each column is: wid+enum+2 
# start with an empty string 
for n, item in enumerate (x): 
if ten(s) < (wid+enum+2) : 
if enum: 
f' [ {n: {enum}}]' # add the index in brackets 
f' {item: <20} 
# add the item text 
print(s) 
# print three columns 
# start the next three columns 
s= 
if enum: 
f' [ {n: {enum}}]' # add the index in brackets 
f' {item: <20} 
# add the item text 
if s: 
print(s) 
# print leftovers 