Lab 3: Project Jumpstart (Part 1/2)

Learning Objectives:

- Describe the big picture of how to apply prescriptive analytics in a complex scenario (like the final project). (Model)
- Describe possible interventions in the final project. (Model)
- Applying Python commands for manipulating lists and strings.

Big picture of prescriptive analytics

1. Measure what is not good.

- Define computable, actionable, simple and enlightening metrics.
- Computing the metrics for the given data, using simulation and reasonable assumptions to fill in missing data.
- Descriptive analytics (not covered in this course or assessed in this project. See DSO 545.)
- Statistical modeling (not covered in this course or assessed in this project. See DSO 530.)

2. Think about ways to improve.
3. Quantify the possible improvements.

Step 1 is essentially deliverable 1 of the project. Step 3 is deliverable 2. See the project document posted on blackboard for due dates of each and more detailed description. After deliverable 1, professor will share results with the whole class and we will brainstorm step 2 together, so you can use other people's findings in step 1 to do your deliverable 2.

Possible interventions and scope of class project

Possible interventions in this project includes (and are not limited to):

- Tool to help Shannon and Hal to come up with better slots to allocate to departments.
- Tool to help department coordinators optimize the schedule within the given slots, taking into account preferences of faculty (which are currently not collected but can be collected in the future).
- Tool for centralized optimization taking into account inputs from all departments simultaneously and outputing a tentative master schedule.

You do not have to create a tool that works on the real data for this project, but simply a proof of concept, perhaps on a smaller data set. For data that is not available, you can use simulation to make it up for now, and do sensitivity analysis to try a few different inputs. Even if your tool cannot be used right now because of lack of data, it can be useful in the future as one can plug in the new data into your framework once it is collected.

Lab 3: Manipulating raw data to compute metrics

Example metrics we will work toward producing together are:

- Percentage of "prime time" that is utilized by room and day of week.
- Percentage of faculty who teach in the evenings by department.
- List of "undesirable conflicts" between courses and number of such conflicts in current schedule.

We will begin with some of the basic building blocks, which are lists and strings. Below is the code used in class.

Indexing and slicing lists

Indexing a list is the act of accessing one of the elements. Slicing is to obtain a contiguous fragment of the original list, which will also be a list. Strings can be viewed as lists of characters and can be indexed and sliced in a similar way. You can check whether something is part of the list or string using the command `in`. See code example below. In the comments are simple exercises to help you practice. (See the discussion on learning from code example by disecting, explaining and modifying in the notes to the 2/8 session.)

```
[5]: # Example of indexing and slicing
     l=[1,2,3,6,9,1]
     l[3]
     l[0:3]
     l[:3]
     # Exercise 1 (E1): obtain the 5th element
     # Solution:
     print('E1:',l[4])

     # E2: slice to obtain [3,6,9]
     # Solution:
     print('E2:',l[2:5])

     s='hello world'
     s[4]
     s[:5]
     # E3: try to get "world"
     # Solution:
     print('E3:',s[6:])

     # Example of negative indexing (from the back)
     l[-1]
     s[-5:]
     # E4: try to get "wor" by backward indexing
     # Solution
     print('E4:',s[-5:-2])

     # Example of complex lists
     l=[5,[2,'g'],'good',[3,4,7]]
     l[1][1] # obtaining letter 'g'

     # E5: try to get the [4,7]
     # Solution
     print('E5:',l[-1][-2:])

     # E6: get 'oo' by list indexing.
     # Solution
     print('E6:', l[2][1:3])


     # Example of list membership
     5 in l
     6 in l
```

```
[2,'g'] in l
'oo' in 'good'

day='M'
days1='MW'
days2='TH'
# E7: test if days1 includes day, days2 includes day
# Solution
print('E7 a) {0} b) {1}'.format(day in days1,day in days2))
```

```
E1: 9
E2: [3, 6, 9]
E3: world
E4: wor
E5: [4, 7]
E6: oo
E7 a) True b) False
```

Manipulating Strings by Splitting, Joining, and Stripping.

Splitting a string by a given substring breaks the original string into chunks, wherever the substring occurs. For example, splitting `Hello World` by the letter `o` will break it into `['Hell','W', 'rld']`. Splitting it by the string `or` will break it into `['Hello W','ld']`. Joining a list of strings by a substring performs the reverse operation. Stripping a string removes any leading whitespace or trailing whitespace. A whites space chracter includes `(space),(newline)` and (tab). This only removes whitespace in the beginning and end of the given string, but not any in the middle.

[13]:
```
# Example of splitting and joining
class1='HOH 303'
class1.split(' ')
'\n'.join(['HOH','303'])

name='John Doe'
name2='Shi, Peng'
#E8: split John Doe into first name and last.
#Solution:
print('E8:',name.split(' '))

#E9: join name and name2 by tab '\t'
#Solution:
print('E9:','\t'.join([name,name2]))

#Example of stripping. The '\n' denotes a new line, which also counts as whitespace.
'   hello \n'.strip()

#E10: split name2 and get first="Peng" last="Shi
# Solution to E10
last,first=name2.split(',')
last=last.strip()
first=first.strip()
print('E10: first={0}, last={1}'.format(first,last))
```

```
E8: ['John', 'Doe']
E9: John Doe        Shi, Peng
E10: first=Peng, last=Shi
```

You can also capitalize the first letter, change everything to upper case, or change all to lower case using the commands below. Casting a string into an integer by the command `int` will make it an integer. For decimal numbers, use `float` instead. To change a number into a string format, you can use `str`. Finally, given a list `l=['a','b']`, you can assign individual members directly to variables using multiple assignment, with the format 'a,b=l

```
[14]: # Examples to illustrate

      'shi'.capitalize()
      'shi'.upper()
      'SHi'.lower()
      str(3.3)
      int('5')
      float('3.3')
      a,b=[3,4]

      room='HOH 303'
      # E11: given text as above, obtain the room number
      pre,suf=room.split(' ')
      print('E11:',int(suf))
```

```
E11: 303
```

List Comprehension

List comprehension is a way of creating a list directly in a `for` loop. For example, the following code can be shortened using list comprehension below.

```
l=[]
for ind in range(5):
    l.append(ind*ind)
```

See additional examples of list comprehension in the course notes for the 2/8 session (explaining Lab 2 solutions).

```
[18]: # List comprehension
      [ind*ind for ind in range(5)]

      # E12: Create 2 * ind when ind is ranging across the list [3,5,6]
      # Solution
      print('E12:',[2*ind for ind in [3,5,6]])

      # Multiple dimensional list comprehension
      [(i,j,'{0}+{1}'.format(i,j)) for i in range(5) for j in range(3)]

      # E13: Create a list where i ranges from [3,5,7] and j from [2,4,6], and the list shoul
```

```
#      [(3,2,'3*2=6'),(3,4,'3*4=7')...]
# Solution
print('E13',[(i,j,'{0}*{1}={2}'.format(i,j,i*j)) for i in range(3,8,2) for j in range(2
```

```
E12: [6, 10, 12]
E13 [(3, 2, '3*2=6'), (3, 4, '3*4=12'), (3, 6, '3*6=18'), (5, 2, '5*2=10'), (5, 4, '5*4=20'),
```