Lab 03: Data Types, Extending Tables

Data 8 Discussion Worksheet

In lecture, you have been introduced to various *data types* in Python such as integers, strings, and arrays. These data types are particularly important for manipulating and extracting useful information out of data, an important skill for data science. In this section, we'll be analyzing some of the behavior that Python displays when dealing with particular data types.

1. Suppose we have executed the following lines of code. Answer each question with the appropriate output associated with each line of code.

```
odd_array = make_array(1, 3, 5, 7)
even_array = np.arange(2, 10, 2)
```

a.
```
odd_array + even_array
```
```
array([3, 7, 11, 15])
```

b.
```
odd_array.item(2)
```
```
5
```

c.
```
even_array.item(3) * odd_array.item(1)
```
```
24
```

d.
```
odd_array*3
```
```
array([3, 9, 15, 21])
```

In this section, we will practice working with tables. In particular, we'll be focusing on table methods and what data types they return. This will help in understanding how to effectively manipulate tables.

2. Your friend Raymond is curious to see whether or not it's cheaper to buy his favorite items on eBay rather than through some other platform! Raymond stumbles upon some auction data from eBay, and decides to use his newly developed Table skills to do some data-crunching. However, Raymond is making a few mistakes and needs your help. For the following questions, identify why the code won't work as is and fix it.

The table below is called `ebay` and contains more than just the 3 rows displayed.

Auction_ID	Item	Opening_Bid	Closing_Price
1	Jacket	50	75
2	Smartwatch	100	150
3	Tablet	350	600

a. `ebay.where('Opening_Bid', are.above(60))/ ebay.num_rows`
Note: The line should evaluate to the proportion of items that have an opening bid above $60.
This won't work because the numerator (`ebay.where('Opening_Bid', are.above(60)`) is not a number, but it is actually a table. Therefore, we are taking a table and trying to divide it by a number, which means Python will throw an error.

Correct code:
```
ebay.where('Opening_Bid', are.above(60)).num_rows /
ebay.num_rows
```

b. `ebay.column('Closing_Price') - ebay.select("Opening_Bid")`
Note: The output should be an array of differences between an item's opening bid and closing price.
`ebay.column('Closing_Price')` is an array in Python, while `ebay.select('Opening_Bid')` is another table. This line of code is attempting to add an array to a table, which will throw an error.

Correct code:
```
ebay.column('Closing_Price') - ebay.column('Opening_Bid')
```
or
```
ebay.column('Closing_Price') -
ebay.select('Opening_Bid').column ('Opening Bid')
```
Why does the second one work?

c. Raymond really wants a new jacket, but his budget is only $150. To see whether eBay has had good deals on jackets historically, Raymond tries to filter the auction data such that it only contains jackets that were sold for a closing price less than

$150. He writes the following code to do so, but hasn't realized the mistake he's making. Help Raymond fix it so that your friend can hopefully get the jacket he deserves! You are allowed to add/delete variables.

```
only_jackets = ebay.where('Item', are.equal_to('Jacket'))
jackets_under_price = ebay.where('Closing_Price',
are.below(150))
```

While they assigned `only_jackets` to a table that only contains Jackets, they are still using the original `ebay` table in assigning `jackets_under_price`. Therefore, `jackets_under_price` will have all items with a closing price underneath 150 dollars, instead of only jackets.

Correct code:

```
only_jackets = ebay.where('Item', are.equal_to('Jacket'))
jackets_under_price = only_jackets.where('Closing_Price',
are.below (150))
```

3. You are working on analyzing the podcast "Dan Carlin's Hardcore History", and have collected the following information in the table podcasts. Some rows are shown below:

Release Date	Title	Description	Era	Series	Downloads
8/7/2016	Supernova in the East IV	Coral Sea, Midwat and Guadlcanal are three of the most...	Modern	Yes	8900000
8/7/2016	King of Kings III	...Xerxes, Spartans, Immortals, Alexandar the Great...	Ancient	Yes	5200000
4/22/2013	Prophets of Doom	Millenial preachers take over the German city of Munster...	Early Modern	No	3180000
4/1/2011	Death Throes of the Republic V	The last great generation of the Roman...	Ancient	Yes	2500000
12/14/2007	Judgement at Nineveh	...empire in history, the biblical era Assyrians...	Ancient	No	149000

a. Before you start your exploration, you want to understand your data better. For each of the columns **Era**, **Series**, and **Downloads** identify if the data contained in that column is numerical or categorical.

Era- Categorical
Series- Categorical
Downloads- Numerical

b. Suppose you were only interested in podcasts that were downloaded between 1,000,000 and 5,000,000 times (excluding 5,000,000). Assign `range_pods` to a

table that only contains rows from the podcasts table that correspond to `podcasts` that were downloaded that many times.

```
range_pods = _____________________________________________________
```

```
range_pods = podcasts.where("Downloads", are.between(1000000,
5000000)
```

Or

```
range_pods = podcasts.where("Downloads",
are.above_or_equal_to(1000000)).where("Downloads",
are.below(5000000))
```

c. You're curious about finding out when the most downloaded ancient era podcast was released. Assign `ancient_date` to a string that represents the date of the podcast on an Ancient era topic that received the most downloads. For this question, assume that the maximum number of downloads is unique. It is okay for your code to wrap along the next line for `ancient_pods_sorted`. You may not assume that the table is already sorted.

```
ancient_pods_sorted = ____________________________________________
ancient_date = ___________________________________________________
```

```
ancient_pods_sorted = podcasts.where("Era",
are.equal_to("Ancient")).sort("Downloads", descending=True)

ancient_date = ancient_pods_sorted.column("Release
Date").item(0)
```