Lab 2 Guide

Shark Tank

Question 0 deals with **reading** in and **cleaning** data. We can use the read_csv() and head() functions we learned from Lab 1 to read in data from our csv file. Next, we want to deal with **missing data** (NaNs). To do this, we can use the fillna() function. For example, if we wanted to replace all NaNs with zeros in a column labeled Name, we could use the following code:

```
df.loc[ : , 'Name' ] = df.loc[ : , 'Name' ].fillna(0)
```

Finally, we need to **clean** the Amount and Equity columns by turning them into **floats**. To do this, we can **remove** nonnumeric characters with the str.replace() functions. For example, if we wanted to remove commas from column labeled Price, we could use the following code:

```
df [ 'Price' ] = df [ 'Price' ].str.replace( ',' , '' )
```

The replace() function can take a wide range of parameters, but we are simply interested in **character replacement** (specifically we are replacing commas/dollar signs with the empty string). Again, we will need to **replace NaNs** with 0 using fillna(0) and **cast** the column values as the float data type using the astype() function we learned in Lab 1:

```
df [ 'Price' ] = df [ 'Price' ].fillna(0).astype(float)
```

Question 1 asks for the companies with the highest **valuation** and highest **amount**. There is no explicit Valuation column, but it can easily be calculated by dividing **the amount invested** by the **equity percentage**. However, before we calculate valuation, let's clean our data further by **removing** the companies that didn't receive any investments (e.g. companies with Equity = 0 or NaN) to avoid **divide-by-0 errors**. To keep dataframe rows based on a certain condition (e.g. df[Equity] > 0), we can use the following code:

```
df = df [ df [ 'Equity' ] > 0 ]
```

Finally, let's calculate the valuation of each company and save it inside a new column named Valuation:

```
df [ 'Valuation' ] = df [ 'Amount' ] / df [ 'Equity' ] * 100
```

Finally, to find the company with the highest valuation, we can use the idxmax() function, which returns the **index** of a column's **highest value**. For idxmax() to work properly, we need to **reset** our **indices**, since we removed a bunch of rows earlier:

```
df = df.reset_index()
```

Now, we can use idxmax() to find the row index with the highest valuation and use iloc() to view the entire row data:

```
df.iloc[ df [ 'Valuation' ].idxmax() ]
```

Question 2 requires us to calculate the amount each shark invested in **total**. To do this, we can:

1. **Calculate** the percentage of Amount each shark invested into every company
2. **Multiply** these percents by Amount to find shark investment amounts to each company
3. **Sum** these amounts together to find the amount each shark invested in total

We need to perform step 1 because sharks sometimes **split** the investment amount equally. Thus, we need to calculate the **correct** amount to use before summing them together. First, let's calculate the **number of sharks** each company had using the sum() function:

```
num_sharks = df.loc[ : , 'Corcoran' : 'Guest' ].sum(axis=1)
```

The sum() function sums entire **columns** by default (axis=0). Since we are adding the values across each **row**, we need to include the axis=1 parameter. Now we can calculate the Amount percentages. It may make more sense to calculate these percentages one shark at a time:

```
df [ 'Corcoran' ] = df [ 'Corcoran' ] / num_sharks
df [ 'Cuban' ] = df [ 'Cuban' ] / num_sharks
...
```

Or we can use the divide() function to perform all of these calculations **in one go**:

```
df.loc[ : , 'Corcoran':'Guest' ] = df.loc[ : , 'Corcoran':'Guest' ].divide( num_sharks , axis=0)
```

Now we have our percentages, we can do something similar for **step 2**: this time using * or multiply()

```
df [ 'Corcoran' ] = df [ 'Corcoran' ] * df [ 'Amount' ]
df [ 'Cuban' ] = df [ 'Cuban' ] * df [ 'Amount' ]
...
```

or

```
df.loc[ : , 'Corcoran':'Guest' ] = df.loc[ : , 'Corcoran':'Guest' ].multiply( df [ 'Amount' ], axis=0)
```

Finally, we can use sum() to calculate the amount totals for all the sharks:

```
df.loc[ : , 'Corcoran':'Guest' ].sum()
```

Question 3 wants us to **tabulate** the number of **funded companies** (e.g. Equity > 0) based on Industry. To do this, we can **group** the companies by industry using the groupby() function and then **tallying** the companies using the count() function:

```
df.groupby( 'Industry' )[ 'Equity' ].count()
```

The groupby() function works by **splitting** a dataframe by some criteria (in our case, 'Industry'), **applying** a function (e.g. sum, mean, count) and then returning the **aggregated** data. Don't forget your visualization as well.

Evidence of Discrimination

This section introduces the pivot table, a powerful tool used for **summarizing** and **organizing** data. Pandas pivot tables take 3 main parameters:

- values - the dependent variable(s) of interest
- index - the independent variable(s) of interest
- aggfunc - how you want to summarize your data (e.g. sum, mean, count, min/max)

Question 1 wants us to visualize **average** expenditures by Ethnicity. This can be quickly tabulated using the pivot_table() function using the **aggregation function** np.mean:

```
table = pd.pivot_table(data = df , values = 'Expenditures' , index = 'Ethnicity' , aggfunc = 'mean' )
```

This will return the average expenditures for **all** ethnicities. Since we specifically want to visualize **White** vs **Hispanic** groups, we can index those with loc:

```
table.loc [ [ 'Hispanic' , 'White not Hispanic' ] ].plot( kind = 'bar' )
```

Question 2 now asks to summarize by Ethnicity and Age Cohort. We can summarize multiple columns by simply including both as the index parameter:

```
index = [ 'Ethnicity' , 'Age Cohort' ]
```

Question 3 wants us to figure out why the results in Question 1 and Question 2 may seem contradictory. To answer this, let's try to:

1. **Visualize** the distribution of Expenditures across each Age Cohort (e.g. Expenditures vs Age Cohort)
2. **Compare** the age distributions of Whites and Hispanics (e.g. Age vs Ethnicity)

For **step 1**, we can follow the same syntax we used in Questions 1 and 2. For **step 2**, we might want to use aggfunc = 'count' instead of 'mean' to tabulate the age **counts**. Let's also be mindful when viewing/interpreting the **x-axes** of our visualizations as they may appear in **alphabetical** order (rather than numeric order).