1 Selecting Rows from Tables

We have the dataset `trips`, which contains data on trips taken as part of a Bay Area bikesharing program. The first few rows of the table are shown below:

Trip ID	Duration	Start Date	Start Station	Start Terminal	End Date	End Station	End Terminal	Bike #	Subscriber Type	Zip Code
913460	765	8/31/2015 23:26	Harry Bridges Plaza (Ferry Building)	50	8/31/2015 23:39	San Francisco Caltrain (Townsend at 4th)	70	288	Subscriber	2139
913459	1036	8/31/2015 23:11	San Antonio Shopping Center	31	8/31/2015 23:28	Mountain View City Hall	27	35	Subscriber	95032
913455	307	8/31/2015 23:13	Post at Kearny	47	8/31/2015 23:18	2nd at South Park	64	468	Subscriber	94107
913454	409	8/31/2015 23:10	San Jose City Hall	10	8/31/2015 23:17	San Salvador at 1st	8	68	Subscriber	95113

We want to know how many trips were long trips, for various values of length. Write a function `num_long_trips` that, given a particular duration, finds the number of trips above that duration.

```python
def num_long_trips(cutoff):
    return len(trips.where("Duration", are.above(cutoff)))
```

Now write a function, `percent_long_trips`, that, given a particular duration, finds the percentage of trips above that duration.

```python
def percent_long_trips(cutoff):
    num_trips = len(trips.column("Duration")
    return 100 * num_long_trips(cutoff)/num_trips
```

We find that most trips have smaller length, but a few are very long. We want to see what the distribution of commute lengths looks like, and reason that commuters will tend of have trips of smaller length. We also figure that commuters will be subscribers to the program, not one-time users.

Write a function, `commuter_distribution`, that, given a particular duration, creates a histogram of trip lengths for trips below that duration, where each trip was taken by someone with a `Subscriber Type` of `Subscriber`. Have the function return the average trip length for trips in the histogram.

```python
def percent_long_trips(cutoff):
    commute_trips = trips.where("Subscriber Type" are.equal_to("Subscriber"))
    short_commute = commute_trips.where("Duration", are.below(cutoff))
    short_commute.hist("Duration")
    return np.mean(short_commute.column("Duration")
```

Now let's consider the locations of the trip. Create a new table `station_data`, with two columns: `station` and `number_of_trips`. Which station had the most departures? Save the name of this station as `busiest_station`.

```python
station_data = trips.group('Start Station').sort('count', descending=True)
station_data = station_data.select('Start Station', 'count')
station_data = station_data.relabeled('count', 'number_of_trips').relabeled('Start Station', 'station')
busiest_station = station_data.column("station").item(np.argmax(station_data.column("number_of_trips")))
```

Now, write a function that calculates the average trip duration for trips leaving from a given station. Name it `avg_trip_length`.

```python
def avg_trip_length(station):
    return np.mean(trips.where('Start Station', are.equal_to(station)).column('Duration'))
```

Add a new column, `trip_length` to the `station_data` table, consisting of the average trip length for the station in question.

```python
station_data = station_data.with_column(
    "trip_length", station_data.apply(avg_trip_length, "station")
)
```

Now add a fourth column, `total_trip_time` to the `station_data` table, consisting of the total duration of all trips that started at that station.

```python
station_data = station_data.with_column(
    "total_trip_time", station_data.column(number_of_trips)*station_data.column(avg_trip_length)
)
```

Finally, let's consider the ridership of each station. First, write a function that takes in an array of strings, where each string is either `"Subscriber"` or `"Customer"`, and returns the percentage of values that are the string `"Subscriber"`.

```python
def ridership_func(rider_array):
    rider_table = Table().with_columns('riders', rider_array).where('riders', are.equal_to("Subscriber"))
    return len(rider_table.column("riders") / len(rider_array)
```

Now, using that function, find the percentage of riders that are subscribers, for each station. Name the station that has the highest percentage of subscribers `high_commute_station`. Consider how you could do this with either `group` or `apply`. What extra step would be needed to use `apply`?

```python
stations_by_ridership = station_data.group("station", ridership_func)
high_commute_station = stations_by_ridership.sort("Subscriber Type ridership_func", descending=True).column("
    Subscriber Type ridership_func").item(0)
```

2 Defining Functions

Write a function that takes an array and returns its range - the difference between the highest value in the array and the lowest.

```python
def range(array):
```

```python
def range(array):
    """Given an array, return its range."""
    return max(array) - min(array)
```

Write a function that takes two strings and prints a third string, which consists of the first two strings with a comma (,) in between them.

```python
def combine_strings(str1, str2):
```

```python
def combine_strings(str1, str2):
    """Given two strings, join them with a comma"""
    return str1 + "," + str2
```

Write a function that takes an array and returns an array of proportions, where each value represents its proprotion of the original array

```python
def proportions(array):
```

```python
def proportions(array):
    """Given an array, convert it to proportions."""
    return array/sum(array)
```

Write a function that takes three positive numbers and returns the sum of the squares of the two largest numbers.

```python
def two_of_three(a, b, c):
```

```python
def two_of_three(a, b, c):
    """Return x*x + y*y, where x and y are the two largest members of the
    positive numbers a, b, and c."""
    return a**2 + b**2 + c**2 - min(a, b, c)**2
```

Write a function that takes a string, and prints it out with any appearance of the string "trees" replaced by the new string, "STANFORD SUCKS". Then, it returns another string, with any appearance of the string "Berkeley" replaced by the new string, "GO BEARS".

```python
def cal_string_func(string):
```

```python
def cal_string_func(string):
    """Given a string, make some modifications."""
    print(string.replace("trees", "STANFORD SUCKS"))
    return string.replace("Berkeley, GO BEARS")
```