In the examples in the left column, `np` refers to the NumPy module. Everything else is a function, a method, an example of an argument to a function or method, or an example of an object we might call the method on. For example, `tbl` refers to a table, `array` refers to an array, and `n` refers to a number. `array.item(0)` is an example call for the method `item`, and in that example, `array` is the name previously given to some array.

Name	Input	Output
`Table()`	None	An empty **Table**
`tbl.with_columns(name, values)` `tbl.with_columns(n1, v1, n2, v2, ...)`	1. **string:** the name of the new column 2. **array:** the values in that column	**Table:** a copy of the original table with the new columns added
`tbl.column(column_name_or_index)`	**string** *or* **int:** the column name or index	**array:** the values in that column
`tbl.num_rows`	None	**int:** the number of rows in the table
`tbl.num_columns`	None	**int:** the number of columns in the table
`tbl.labels`	None	**array:** the names of each column (as strings) in the table
`tbl.select(col1, col2, ...)`	**string** *or* **int**: column name(s) or index(es)	**Table** with the selected columns
`tbl.drop(col1, col2, ...)`	**string** *or* **int**: column name(s) or index(es)	**Table** without the selected columns
`tbl.relabeled(old_label, new_label)`	1. **string:** the old column name 2. **string:** the new column name	**Table:** a new table
`tbl.show(n)`	(Optional) **int:** number of rows you want to display	None: Dispays a table with `n` rows
`tbl.sort(column_name_or_index)`	1. **string** *or* **int**: column name or index 2. (Optional) **boolean**: `descending=True`	**Table**: a copy of the original table with the column sorted
`tbl.where(column, predicate)`	1. **string** *or* **int**: column name or index 2. `are.(...)` predicate	**Table**: a copy of the original table with only the rows that match the predicate
`tbl.take(row_indices)`	**array** of ints: the indices of the rows to be included in the table OR **int**: the index of the row to be included	**Table:** a copy of the original table with only the rows at the given indices
`tbl.scatter(x_column, y_column)`	1. **string** *or* **int**: name or index of the column on x-axis 2. **string** *or* **int**: name or index of the column on y-axis 3. (Optional) **boolean:** `fit_line=True`	None: Draws a scatter plot
`tbl.plot(x_column, y_column)` `tbl.plot(x_column)`	1. **string** *or* **int**: name or index of the column on the x-axis 2. **string** *or* **int**: name or index of the column on y-axis	None: Draws a line plot
`tbl.barh(categories)` `tbl.barh(categories, values)`	1. **string** *or* **int**: name or index of the column with categories 2. (Optional) **string** *or* **int**: name or index of the column with values for corresponding categories	None: Draws a bar chart
`tbl.hist(column, unit, bins, group)`	1. **string** *or* **int**: name or index of the column with categories 2. (Optional) **string**: units of x-axis 3. (Optional) **array**: ints/floats denoting bin boundaries 4. (Optional) **str**: name of column to group by	None: Draws a histogram
`tbl.bin(column_name_or_index)` `tbl.bin(column_name_or_index, bins)`	1. **string** *or* **int**: column name(s) or index(es) 2. (Optional) **array** of ints/floats denoting bin boundaries or an **int** of the number of bins you want	**Table**: A new table
`tbl.apply(function)` `tbl.apply(function, col1, col2, ...)`	1. **function**: function to apply to column 2. (Optional) **string** or **int**: name or index of the column to apply function to (if you have multiple columns, the respective columns' values will be passed as the corresponding argument to the function), and if there is no argument, your function will be applied to every row (Row object) in `tbl`	**array**: contains an element for each value in the original column after applying the function to it
`tbl.group(column_or_columns, collect)`	1. **string/int** *or* **array of strings/ints**: column(s) on which to group 2. (Optional) **function**: function to aggregate values in cells (defaults to count)	**Table**: A new table
`tbl.pivot(col1, col2, values, collect)` `tbl.pivot(col1, col2)`	1. **string** *or* **int**: name or index of column whose unique values will make up columns of the pivot table 2. **string** *or* **int**: name or index of column whose unique values will make up rows of the pivot table 3. (Optional) **string** or **int**: name or index of column containing the values of cell 4. (Optional) **function**: how the values are collected; e.g. `np.mean`	**Table**: A new table
`tblA.join(colA, tblB, colB)` `tblA.join(colA, tblB)`	1. **string**: name of column in `tblA` with values to join on 2. **Table**: other table 3. (Optional) **string**: if column names are different between tables, the name of the shared column in `tblB`	**Table:** A new table
`tbl.sample(k)` `tbl.sample(k, with_replacement)`	1. **int**: sample size 2. (Optional) **boolean:** `with_replacement=True`	**Table**: A new tables with `k` rows
`tbl.row(row_index)`	**int**: row index	**Row object** with the values of the row and labels of the corresponding columns
`tbl.rows`	None	**Row object** made up of all rows as individual row objects

Percent of data within k SDs of the mean

Percent in Range	Chebyshev's	Normal Distribution
mean $\pm$ 1 SD	at least 0%	about 68%
mean $\pm$ 2 SDs	at least 75%	about 95%
mean $\pm$ 3 SDs	at least 88.88...%	about 99.73%

Array Functions and Methods:

Name	Description
`max(array)`	Returns the maximum value of an array
`min(array)`	Returns the minimum value of an array
`sum(array)`	Returns the sum of the values in an array
`abs(n)`, `np.abs(array)`	Take the absolute value of a number or each number in an array
`np.round(n)`, `np.round(array)`	Round number or array of numbers to the nearest integer
`len(array)`	Returns the length (number of elements) of an array
`make_array(val1, val2, ...)`	Makes a numpy array with the values passed in
`np.mean(array)`, `np.average(array)`	Returns the mean value of an array
`np.std(array)`	Returns the standard deviation of an array
`np.diff(array)`	Returns a new array of size `len(arr) - 1` with elements equal to the difference between adjacent elements; `val_2 - val_1`, `val_3 - val_2`, etc.
`np.sqrt(n)`, `np.sqrt(array)`	Returns an array with the square root of each element

`np.arange(start, stop, step)`
`np.arange(start, stop)`
`np.arange(stop)`
	An array of numbers starting with `start`, going up in increments of `step`, and going up to but excluding `stop`. When `start` and/or `step` are left out, default values are used in their place. Default step is 1; default start is 0.

`array.item(index)`	Returns the (index + 1)-th item in an array (remember Python indices start at 0)

`np.random.choice(array, n)`
`np.random.choice(array)`
`np.random.choice(array, n, replace)`
	Picks one (by default) or some number `n` of items from an array at random. Default is with replacement. For sampling without replacement, use argument `replace=False`.

`np.count_nonzero(array)`	Returns the number of non-zero (or `True`) elements in an array
`np.append(array, item)`	Returns a copy of the input array with `item` (must be the same type as the other entries in the array) appended to the end
`np.append(array1, array2)`	Returns a copy of the input `array1` with `array2` appended to the end (must be the same type as the other entries in the other array)
`percentile(p, array)`	Returns the pth percentile of an array

`Table.where` Predicates:

Any of these predicates can be negated by adding `not_` in front of them, e.g. `are.not_equal_to(x)` or `are.not_containing(S)`.

Name	Description
`are.equal_to(x)`	Equal to `x`
`are.above(x)`	Greater than `x`
`are.above_or_equal_to(x)`	Greater than or equal to `x`
`are.below(x)`	Less than `x`
`are.below_or_equal_to(x)`	Less than or equal to `x`
`are.between(x, y)`	Greater than or equal to `x` and less than `y`
`are.between_or_equal_to(x, y)`	Greater than or equal to `x` and less than or equal to `y`
`are.contained_in(A)`	Is a substring of `A` (if `A` is a string) or an element of `A` (if `A` is an array)
`are.containing(S)`	Contains the string `S`
`are.strictly_between(x, y)`	Greater than `x` and less than `y`

Miscellaneous Functions:

These are functions in the `datascience` library that are used in the course that don't fall into any of the categories above.

Name	Input	Output
`sample_proportions(sample_size, model_proportions)`	1. **int**: sample size 2. **array**: an array of proportions that should sum to 1	**array**: each item corresponds to the proportion of times that corresponding item was sampled from `model_proportions` in `sample_size` draws. Should sum to 1.
`minimize(function)`	`function`: name of a function that will be minimized	**array**: An array in which each element corresponds to an argument that minimizes the output of the function. Values in the array are listed based on the order they are passed into the function; the first element in the array is also going to be the first value passed into the function.

Concepts Overview

This list is illustrative and **non-exhaustive**. There may be additional examples and topics in scope beyond what is listed.

Data Types

- `str`
- `int`
- `float`
- `array`
- `Table`
- `NoneType`
- `builtin_function_or_method`

Finding Probabilities

- Complement Rule
- Multiplication Rule
- Addition Rule
- Bayes' Rule

Comparison Operators

- `>`
- `<`
- `==`
- `!=`
- `>=`
- `<=`

Examples of Test Statistics

- Total variation distance
- Difference
- Absolute difference
- Sample proportion

Plot Types

- Histogram
- Line plot
- Scatter plot
- Bar chart

Correlation Coefficient, r

- The average of the product of x and y, when both are in standard units

Regression Line in Standard Units

- $y_{SU} = r * x_{SU}$