* From UC Berkeley's documentation on the datascience package

Name	Description	Input	Output
Table()	Create an empty table, usually to extend with data	None	An empty **Table**
Table().read_table(filename)	Create a table from a data file	**string**: the name of the file	**Table** with the contents of the data file

tbl.with_columns(name, values)
tbl.with_columns(n1, v1, n2, v2,...)
	A table with an additional or replaced column or columns. name is a string for the name of a column, values is an array	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)	The values of a column (an array)	**string** *or* **int**: the column name or index	**array**: the values in that column
tbl.num_rows	Compute the number of rows in a table	None	**int**: the number of rows in the table
tbl.num_columns	Compute the number of columns in a table	None	**int**: the number of

			columns in the table
tbl.labels	Lists the column labels in a table	None	**array**: the names of each column (as strings) in the table
tbl.select(col1, col2, ...)	Create a copy of a table with only some of the columns. Each column is the column name or index.	**string** *or* **int**: column name(s) or index(es)	**Table** with the selected columns
tbl.drop(col1, col2, ...)	Create a copy of a table without some of the columns. Each column is the column name or index.	**string** *or* **int**: column name(s) or index(es)	**Table** without the selected columns
tbl.relabel(old_label, new_label)	Modifies the existing table *in place*, changing the column heading in the first argument to the second	1. **string**: the old column name 2. **string**: the new column name	**Table**: a copy of the original with the changed label
tbl.sort(column_name_or_index)	Create a copy of a table sorted by the values in a column. Defaults to ascending order unless descending = True is included.	1. **string** *or* **int**: column index or name 2. (Optional) descending = True	
tbl.where(column, predicate)	Create a copy of a table with only the rows that match some *predicate* See Table.where predicates below.	1. **string** *or* **int**: column name or index 2. are.(...) predicate	

tbl.take(row_indices)	A table with only the rows at the given indices. row_indices is either an array of indices or an integer corresponding to one index.	**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 with only the rows at the given indices
tbl.scatter(x_column, y_column)	Draws a scatter plot consisting of one point for each row of the table. Note that x_column and y_column must be strings specifying column names.	1. **string**: name of the column on the x-axis 2. **string**: name of the column on the y-axis	None: draws a scatter plot
tbl.plot(x_column, y_column)	Draw a line graph consisting of one point for each row of the table.	1. **string**: name of the column on the x-axis 2. **string**: name of the column on the y-axis	None: draws a line graph

tbl.barh(categories)
tbl.barh(categories, values)
	Displays a bar chart with bars for each category in a column, with height proportional to the corresponding frequency. values argument unnecessary if table has only a column of categories and a column of values.	1. **string**: name of the column with categories
		2. (Optional) **string**: the name of the column with values for corresponding categories
			None: draws a bar chart

tbl.hist(column, unit, bins)	Generates a histogram of the numerical values in a column. unit and bins are optional arguments, used to label the axes and group the values into intervals (bins), respectively. Bins have the form [a, b), where a is included in the bin and b is not.	1. **string**: name of the column with categories 2. (Optional) **string**: units of x-axis 3. (Optional) **array** of ints/floats denoting bin boundaries	None: draws a histogram
tbl.apply(function, column)	Returns an array of values resulting from applying a function to each item in a column.	1. **function**: function to apply to column	**array**: contains an

		2. **string**: name of the column to apply function to	element for each value in the original column after applying the function to it
tbl.group(column_or_columns, func)	Group rows by unique values or combinations of values in a column(s). Multiple columns must be entered in array or list form. Other values aggregated by count (default) or optional argument func.	1. **string** or **array of strings**: 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)	A pivot table where each unique value in col1 has its own column and each unique value in ccol2 has its own row. Count or aggregate values from a third column, collect with some function. Default values and collect return counts in cells.	1. **string**: name of column whose unique values will make up columns of pivot table 2. **string**: name of column whose unique values will make up rows of pivot table 3. (Optional) **string**: name of column that describes the values of cell 4. (Optional) **function**: how the values are collected, e.g. sum or np.mean	**Table**: a new Table

tblA.join(colA, tblB, colB)
tblA.join(colA, tblB)
	Generate a table with the columns of tblA and tblB, containing rows for all values of a column that appear in both tables.	1. **string**: name of column in tblA with values to join on
		2. **Table**: other Table
			Table: a new Table

	Default colB is colA. colA and colBmust be strings specifying column names.	3. (Optional) **string**: if column names are different between Tables, the name of the shared column in tblB	
tbl.sample(n) tbl.sample(n, with_replacement)	A new table where n rows are randomly sampled from the original table. Default is with replacement. For sampling without replacement, use argument with_replacement=False. For a non-uniform sample, provide a third argument weights=distribution where distribution is an array or list containing the probability of each row.	1. **int**: sample size 2. (Optional) with_replacement=True	**Table**: a new Table with nrows

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(num), np.abs(array)	Take the absolute value of number or each number in an array.
round(num), 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.average(array)
np.mean(array)
	Returns the mean value 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(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 i-th item in an array (remember Python indices start at 0!)

np.random.choice(array, n)
np.random.choice(array)
	Picks one (by default) or some number 'n' of items from an array at random. By default, with replacement.

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.

percentile(percentile, array)	Returns the corresponding percentile of an array.