## Data 8R Tables Cheat Sheet

A table can be created using...

**`Table.read_table(path)`** reads an existing table at a location on disk or the Internet. The argument **`path`** is a string.

**`Table()`** is an empty table with no columns.

A table **`t`** can be extended using...

**`t.with_columns(label1, values1)`** creates a table containing a column named **`label1`** containing **`values1`**. **`label1`** should be a string, and **`values1`** should be an array.

**`t.with_columns(label1, values1, label2, values2)`** creates a table containing a column named **`label1`** containing **`values1`**, and a column named **`label2`** containing **`values2`**.

For example:...

**`Table().with_columns("R", make_array(1, 2))`** is a table with a column named "R" containing the numbers 1 and 2.

**`t.relabeled(old_label, new_label)`** creates a copy of table **`t`** but with the specified column relabeled.

---

Table values can be accessed…

**`t.num_columns`** (no parentheses) returns the number of columns in the table, an integer.

**`t.num_rows`** (no parentheses) returns the number of columns in the table, an integer.

**`t.column(label)`** returns an array containing the values in the column in **`t`** named **`label`**. **`label`** should be a string.

---

A table with fewer columns can be produced with…

**`t.select(label1, label2, …)`** returns a copy of the table with only the selected columns.

**`t.drop(label1, label2, …)`** returns a copy of the table excluding the selected columns.

---

A table with a different number of rows can be produced with…

**`t.take(array_of_indices)`** returns a copy of the table with only the rows whose indices were in the array.

**`t.sample(sample_size, with_replacement)`** returns a new table with the specified number of rows randomly selected from the original table with or without replacement.

**`t.where(label1, condition)`** returns a copy of the table with only the rows that fulfill the condition, a "predicate".

Predicates include…

**`are.equal_to(value)`** returns a predicate that checks for being equal to **`value`**.

**`are.not_equal_to(value)`** returns a predicate that checks for not being equal to **`value`**.

**`are.above(num)`** returns a predicate that checks for being bigger than (and not equal to) **`num`**.

**`are.below(num)`** returns a predicate that checks for being less than (and not equal to) **`num`**.

**`are.containing(value)`** returns a predicate that checks for being a string that contains **`value`**, which should be a string.

**`are.contained_in(values)`** returns a predicate that checks for being an element of **`values`**, which should be an array.

---

Tables can generate visualizations with…

**`t.barh(categories, values)`** draws a horizontal bar chart using the **`categories`** column in **`t`** for bar names and the **`values`** column in **`t`** for bar lengths.

**`t.group_barh(label)`** draws a horizontal bar chart of the count distribution of the column in **`t`** named **`label`**.

**`t.hist(label)`** draws a histogram of the selected column.

**`t.plot(label_x, label_y)`** draws a line plot of the two columns selected.

**`t.scatter(label_x, label_y)`** draws a scatter plot of the columns selected.

---

Other table methods include…

**`t.sort(label)`** returns a copy of the table with the rows sorted by the selected label in either ascending (the default) or descending (with the extra argument **`descending=True`**) order.

**`t.apply(func, label)`** returns an *array* with one element per row of **`t`**. Each element is the value of calling **`func`** on a different element of the column in **`t`** named **`label`**.

**`t.group(label)`** returns a table of the count distribution of the column in **`t`** named **`label`**. One column of the returned table is named **`label`**, and the other is named "count".

**`t.group(label, func)`** returns a table with one row for each unique value in the column named **`label`**. It has a column with that name, and another column for each other column in **`t`**. Those columns are computed by grouping together the values in the original column according to the column named **`label`**, and then calling **`func`** on each such group of values. These columns have their original name, with the name of **`func`** appended.

---

Some additional useful functions in the **`np`** module are…

**`np.repeat(value, num)`** returns an array with **`num`** copies of **`value`**.

**`np.append(array1, array2)`** returns an array containing all the elements of **`array1`** followed by all the elements of **`array2`**.