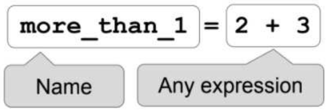
Statements



```
more_than_1 = 2 + 3
```

- Statements don't have a value; they perform an action.
- An assignment statement changes the meaning of the name to the left of the = symbol.
- The name is bound to the value of the right hand side

Comparisons

- < and > mean what you expect (less than, greater than)
- <= means ”less than or equal; likewise for >=
- == means “equal to”; != means “not equal to”
- Comparing strings compares their alphabetical order

Arrays: sequences of the same type that can be manipulated

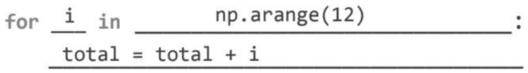
- Arithmetic and comparisons are applied to each element individually
  - `make_array(1, 2, 3) > 2 # array([False, False, True])`
- Elementwise operations can be done on arrays of the same size
  - `make_array(1, 2)* make_array(2, 3) # array([2, 6])`

Defining a Function

```
def function(arg1, arg2, ...):
    # Body can contain any code
```

Note: Many functions `return` a value

for Statements



```
for i in np.arange(12):
    total = total + i
```

- The body is executed **for** every item in a sequence
- The body of the statement can have multiple lines
- The body should do something: assign, sample, etc

Conditional Statements

```
if <if_expression>:
    <if_body>
elif <elif_expression_0>:
    <elif_body_0>
elif <elif_expression_1>:
    <elif_body_1>
...
else:
    <else_body>
```

Simulating a Statistic

- Define a function to simulate one value of the statistic
- Create an empty collection array
- For each repetition of the process:
  - Call the function to simulate one value
  - Append this value to the collection array
- At the end, all simulated values will be in the collection array

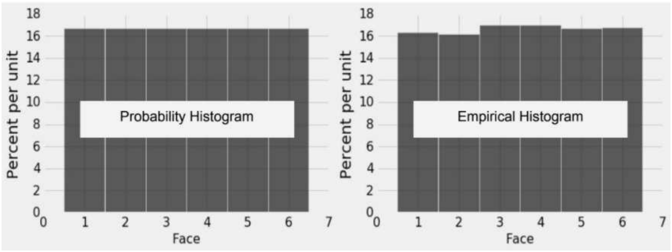
Total Variation Distance between two categorical distributions

- For each category, find the difference between the proportions in the two distributions
- Take the absolute value of the differences
- Sum all the absolute values, then divide by 2

A **histogram** has three defining properties:

- Bins are contiguous (though some may be empty) and drawn to scale.
- The area of each bar is the percent of entries in the bin.
- The total area of the histogram is 100%.

- The histogram on the left displays the theoretical probabilities of the number of spots on one roll of a fair die.
- The histogram on the right represents the empirical (or observed) distribution of the numbers of spots on many rolls of a fair die.
- Example of the *Law of Averages*: The more we roll, the more the histogram on the right is likely to resemble the one on the left.



Finding Probabilities

Complement Rule:
P(event happens) = 1 – P(event does not happen)

Multiplication Rule:
P(two events both happen) = P(first event happens) * P(second event happens given that the first event happened)

Addition Rule: If an event can happen in only one of two ways, then:
P(event happens) = P(happens in the first way) + P(happens in the second way)

p-values

- The *observed significance level* (or *p-value*) of a test is the chance, calculated under the null hypothesis, that the test statistic is equal to the one observed in the sample or more in the direction of the alternative.
- The result of a test of hypotheses is called *statistically significant* if the p-value is less than 5%; *highly statistically significant* if the p-value is less than 1%.

Table Functions and Methods

In the examples in the left column, `np` refers to the NumPy module, as usual. 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 `num` 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	Description	Input	Output
`Table()`	Create an empty table, usually to extend with data (Ch 6)	None	An empty **Table**
`Table().read_table(filename)`	Create a table from a data file (Ch 6)	**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 (Ch 6)	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) (Ch 6)	**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 (Ch 6)	None	**int**: the number of rows in the table
`tbl.num_columns`	Compute the number of columns in a table (Ch 6)	None	**int**: the number of columns in the table
`tbl.labels`	Lists the column labels in a table (Ch 6)	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. (Ch 6)	**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. (Ch 6)	**string** *or* **int**: column name(s) or index(es)	**Table** without the selected columns
`tbl.relabeled(old_label, new_label)`	Creates a new table, changing the column name specified by the old label to the new label, and leaves the original table unchanged. (Ch 6)	1. **string**: the old column name 2. **string**: the new column name	**Table**: a new Table
`tbl.show(n)`	Display `n` rows of a table. If no argument is specified, defaults to displaying the entire table. (Ch 6.1)	(Optional) **int**: number of rows you want to display	None: displays a table with `n` rows
`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. (Ch 6.1)	1. **string** *or* **int**: column index or name 2. (Optional) `descending=True`	**Table**: a copy of the original with the column sorted
`tbl.where(column, predicate)`	Create a copy of a table with only the rows that match some *predicate*. See `Table.where` predicates below. (Ch 6.2)	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)`	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. (Ch 6.2)	**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.exclude(row_indices)`	A table without 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 excluded from the Table OR **int**: the index of the row to be excluded	**Table**: a copy of the original table without 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. Include optional argument `fit_line=True` if you want to draw a line of best fit for each set of points. (Ch 7)	1. **string**: name of the column on the x-axis 2. **string**: name of the column on the y-axis 3. (Optional) `fit_line=True`	None: draws a scatter plot

`tbl.plot(x_column, y_column)`
`tbl.plot(x_column)`
	Draw a line graph consisting of one point for each row of the table. If you only specify one column, it will plot the rest of the columns on the y-axis as different colored lines. (Ch 7)	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. (Ch 7.1)	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, group)`	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. (Ch 7.2)	1. **string**: name of the column with categories 2. (Optional) **string**: units of x-axis 3. (Optional) **array** of ints/floats denoting bin boundaries 4. (Optional) **string**: name of categorical column to draw separate overlaid histograms for	None: draws a histogram

`tbl.bin(column_name_or_index)`
`tbl.bin(column_name_or_index, bins)`
	Groups values into intervals, known as bins. Results in a two-column table that contains the number of rows in each bin. The first column lists the left endpoints of the bins, except in the last row. If the `bins` argument isn't used, default is to produce 10 equally wide bins between the min and max values of the data. (Ch 7.2)	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 tables

`tbl.apply(function)`
`tbl.apply(function, col1, col2, ...)`
	Returns an array of values resulting from applying a function to each item in a column. (Ch 8.1)	1. **function**: function to apply to column
		2. (Optional) **string**: name of the column to apply function to (if you have multiple columns, the respective column's 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 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, 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`. (Ch 8.2)	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 `col2` has its own row. Count or aggregate values from a third column, collect with some function. Default `values` and `collect` return counts in cells. (Ch 8.3)	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. Default `colB` is `colA`. `colA` and `colB` must be strings specifying column names. (Ch 8.4)	1. **string**: name of a 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(n)`
`tbl.sample(n, with_replacement)`
	A new table where `n` rows are randomly sampled from the original table; by default, `n=tbl.num_rows`. Default is with replacement. For sampling without replacement, use argument `with_replacement=False`. For a non-uniform sample,	1. **int**: sample size
		2. (Optional) `with_replacement=False`
			Table: a new Table with `n` rows

	provide a third argument `weights=distribution` where `distribution` is an array or list containing the probability of each row. (Ch 10)		
`tbl.row(row_index)`	Accesses the row of a table by taking the index of the row as its argument. Note that rows are in general not arrays, as their elements can be of different types. However, you can use `.item` to access a particular element of a row using `row.item(label)`. (Ch 17.3)	**int**: row index	**Row object** with the values of the row and labels of the corresponding columns
`tbl.rows`	Can use to access all of the rows of a table.	None	**Rows object** made up of all rows as individual row objects

String Methods

Name	Description
`str.split(separator)`	Splits the string (`str`) into a list based on the `separator` that is passed in
`str.join(array)`	Combines each element of `array` into one string, with `str` being in-between each element
`str.replace(old_string, new_string)`	Replaces each occurrence of `old_string` in `str` with the value of `new_string` (Ch 4.2.1)

Array Functions and Methods

Name	Chapter	Description
`max(array)`	3.3	Returns the maximum value of an array
`min(array)`	3.3	Returns the minimum value of an array
`sum(array)`	3.3	Returns the sum of the values in an array
`abs(num)`, `np.abs(array)`	3.3	Takes the absolute value of a number or each number in an array

`round(num)`, `np.round(array)`
`round(num, decimals)`, `np.round(array, decimals)`
	3.3	Rounds a number or an array of numbers to the nearest integer. If `decimals` (integer) is included, rounds to that many digits. If `decimals` is negative, remove that many digits of precision.

`len(array)`, `len(string)`	3.3	Returns the length (number of elements) of an array. If a string (`str`) is passed in instead, returns the number of characters in the string.
`make_array(val1, val2, ...)`	5	Makes a numpy array with the values passed in

`np.average(array)`
`np.mean(array)`
	5.1	Returns the mean value of an array

`np.std(array)`	14.2	Returns the standard deviation of an array
`np.diff(array)`	5.1	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)`	5.1	Returns an array with the square root of each element

`np.arange(start, stop, step)`
`np.arange(start, stop)`
`np.arange(stop)`
	5.2	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)`	5.3	Returns the i-th item in an array (remember Python indices start at 0!)

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

`np.count_nonzero(array)`	9	Returns the number of non-zero (or `True`) elements in an array.
`np.append(array, item)`	9.2	Returns a copy of the input array with `item` appended to the end.
`percentile(percentile, array)`	13.1	Returns the corresponding percentile of an array.

Table Filtering Predicates

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

Predicate	Description
`are.equal_to(Z)`	Equal to `Z`
`are.not_equal_to(Z)`	Not equal to `Z`
`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.strictly_between(x, y)`	Greater than `x` and less than `y`
`are.contained_in(A)`	Is a substring of `A` (if `A` is a string) or an element of `A` (if `A` is a list/array)
`are.containing(S)`	Contains the string `S`

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. You can also read more about all functions in the `datascience` library on the `datascience` documentation.

Name	Description	Intput	Output
`sample_proportions(sample_size, model_proportions)`	`sample_size` should be an integer, `model_proportions` an array of probabilities that sum up to 1. The function samples `sample_size` objects from the distribution specified by `model_proportions`. It returns an array with the same size as `model_proportions`. Each item in the array corresponds to the proportion of times it was sampled out of the `sample_size` times. (Ch 11.1)	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)`	Returns an array of values such that if each value in the array was passed into `function` as arguments, it would minimize the output value of `function`. (Ch 15.4)	**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.