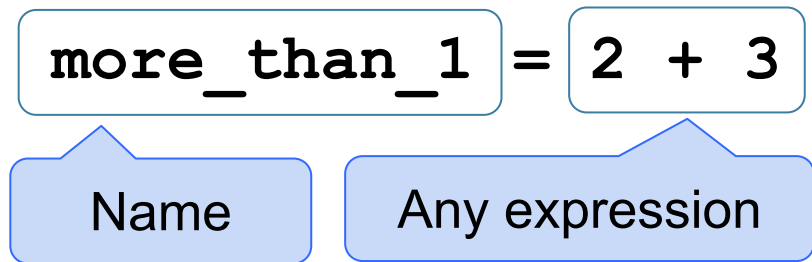


Statements



- 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 a value (not an equation).

Comparisons

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

**Arrays** - sequences of the same type that can be manipulated

- Arithmetic and comparisons are applied to each element of an array individually
  - `make_array(1,2,3) ** 2 # array([1, 4, 9])`
- Elementwise operations can be done on arrays of the same size
  - `make_array(3,2) * make_array(5,4) # array([15,8])`

Defining a Function

```
def function_name(arg1, arg2, ...):  
    # Body can contain anything inside of it  
    return # a value (the output of the function call)
```

Defining a Function with no arguments

```
def function_name():  
    # Body can contain anything inside of it  
    return # a value (the output of the function call)
```

- Functions with no arguments can be called by `function_name()`

For Statements

```
total = 0  
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, print, 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>
```

Total Variation Distance

Total variation distance is a statistic that represents the difference between two distributions

```
TVD = 0.5*(np.sum(np.abs(dist1-dist2)))
```

**Operations:** addition 2+3=5; subtraction 4-2=2; division 9/2=4.5  
multiplication 2\*3=6; division remainder 11%3=2;  
exponentiation 2\*\*3=8

**Data Types:** **string** "hello"; **boolean** True, False;  
**int** 1, -5; **float** - 2.3, -52.52, 7.9, 8.0

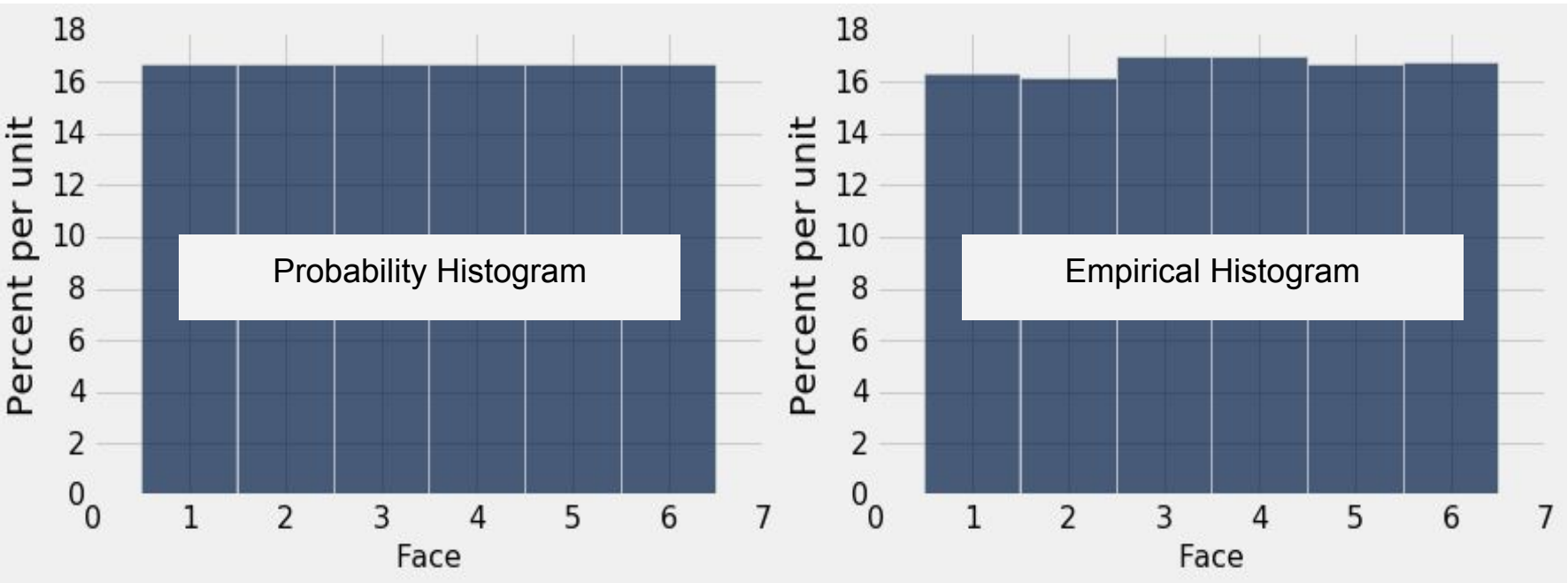
**Table.where predicates:** Any of these predicates can be negated by adding "not\_" in front of them, e.g. `are.not_equal_to(x)`

- `are.equal_to(x) # val == x`
- `are.above(x) # val > x`
- `are.above_or_equal_to(x) # val >= x`
- `are.below(x) # val < x`
- `are.between(x, y) # x <= val < y`
- `are.containing(s) # contains the string s`

A **histogram** has a few defining properties:

- The bins are continuous (though some might be empty) and are drawn to scale
- The **area** of each bar is equal to the percent of entries in the bin
- The total area is 100%

- The histogram on the left represents the theoretical probabilities in the distribution of the face that appears on one roll of a fair die
- The histogram on the right represents the observed distribution of the faces after rolling the die many times
- If we keep rolling, the right hand histogram is likely to look more like the one on the left



Calculating Probabilities

*Complement Rule:*  $P(\text{event does not happen}) = 1 - P(\text{event happens})$

*Multiplication Rule:*  $P(\text{two events both happen}) = P(\text{one happens}) * P(\text{the other happens, given that the first happened})$

*Addition Rule:* If an event can happen in ONLY one of two ways:  
 $P(\text{event happens}) = P(\text{first way it can happen}) + P(\text{second way it can happen})$

*Bayes' Rule:*  $P(\text{event A happened given event B happened}) = \frac{P(\text{both event A and event B happened})}{P(\text{event B happened})}$

For Bayes' rule, if the probabilities are displayed on a tree diagram, the denominator is the chance of the branches in which B happens, and the numerator is the chance of the branches in which both A and B happen.

Simulating a Statistic:

- Create an empty array in which to collect the simulated values
- For each repetition of the process
  - Simulate one value of the statistic
  - Append this value to the collection array
- At the end, all simulated values will be in the collection array

MATH 108 Final Reference Guide — Page 2

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.

max(array); min(array)	Maximum or minimum of an array
sum(array)	Sum of all elements in an array; The sum of an array of boolean values is the number of values that are True
len(array)	Length (num elements) in an array
round(num); np.round(array)	The nearest integer to a single number or each number in an array
abs(num); np.abs(array)	The absolute value of a single number or each number in an array
np.average(array), np.mean(array)	The average of the values in an array
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 places. Default step is 1; default start is 0.
array.item(index)	The item in the array at some index. array.item(0) is the first item of array.
np.append(array, item)	A copy of the array with item appended to the end. If item is another array, all of its elements are appended.
np.exp(array)	Calculate the exponential fo all the elements in the array.
np.random.choice(array) np.random.choice(array, n)	An item selected at random from an array. If n is specified, an array of n items selected at random with replacement is returned. Default n is 1.
np.ones(n)	An array of length n which consists of all ones.
np.diff(array)	An array of length len(array)-1 which contains the difference between adjacent elements.
np.count_nonzero(array)	An integer corresponding to the number of non-zero (or True) elements in an array.
sample_proportions(sample_size, model_proportions)	An array of proportions that add up to 1. The result of sampling sample_size elements from a distribution specified by model_proportions, and keeping track of the proportion of each element sampled.
Table()	An empty table.
Table.read_table(filename)	A table with data from a file.
tbl.num_rows	The number of rows in a table.
tbl.num_columns	The number of columns in a table.
tbl.labels	A list of the column labels of a table.
tbl.with_column(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.
tbl.column(column_name_or_index)	An array containing the values of a column
tbl.select(col1, col2, ...)	A table with only the selected columns. (Each argument is the label of a column, or a column index.)
tbl.drop(col1, col2, ...)	A table without the dropped columns. (Each argument is the label of a column, or a column index.)
tbl.relabeled(old_label, new_label)	A new table with a label changed.
tbl.take(row_index) tbl.take(row_indices)	A table with only the row(s) at the given index or multiple indices. row_indices must be an array of indices.
tbl.exclude(row_index) tbl.exclude(row_indices)	A table without the row(s) at the index or multiple indices. row_indices must be an array of indices.
tbl.sort(column_name_or_index)	A table of rows sorted according to the values in a column (specified by name/index). Default order is ascending. For descending order, use argument descending=True. For unique values, use distinct=True.
tbl.where(column, predicate)	A table of the rows for which the column satisfies some predicate. See “Table.where predicates” on Page 1.
tbl.apply(function, column_or_columns)	An array of results when a function is applied to each item in a column.
tbl.group(column_or_columns)	A table with the counts of rows grouped by unique values or combinations of values in a column or columns.
tbl.group(column_or_columns, func)	A table that groups rows by unique values or combinations of values in a column or columns. The other values are aggregated by func. All column names (except the one(s) we group by) will now be `original_name func`. If a column is named ‘price’, and we group using the min function, our new column name will be ‘price min’.
tblA.join(colA, tblB, colB) tblA.join(colA, tblB)	A table with the columns of tblA and tblB, containing rows for all values of a column that appear in both tables. Default value of colB is colA. colA is a string specifying a column name, as is colB.
tbl.pivot(col1, col2) tbl.pivot(col1, col2, vals, collect)	A pivot table where each unique value in col1 has its own column and each unique value in col2 has its own row. The cells of the grid contain row counts (two arguments) or the values from a third column, aggregated by the collect function (four arguments) .
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. If sample size n is not specified, the default is the number of rows in the original table.
tbl.scatter(x_column, y_column)	Draws a scatter plot consisting of one point for each row of the table.
tbl.barh(categories) tbl.barh(categories, values)	Displays a bar chart with bars for each category in a column, with length proportional to the corresponding frequency. If values is not specified, overlaid bar charts of all the remaining columns are drawn.
tbl.bin(column, bins)	A table of how many values in a column fall into each bin. Bins include lower bounds & exclude upper bounds.
tbl.hist(column, unit, bins, group)	Displays a histogram of the values in a column. unit and bins are optional arguments, used to label the axes and group the values into intervals (bins), respectively. Bins include lower bounds & exclude upper bounds. If group is specified, the rows are grouped by the values in the column, and histograms for all the groups are overlaid.



- **P-Value:** The chance, **under the null hypothesis**, that the test statistic comes out equal to the one in the sample, or more in the direction of the alternative:
  - If the p-value is small and the null is true, something very unlikely has happened.
  - Conclude that the data support the alternative hypothesis more than they support the null.

- Even if the null is true, your random sample might indicate the alternative, just by chance
- The **cutoff** for P is the chance that your test makes the wrong conclusion when the null hypothesis is true
- Using a small cutoff limits the probability of this kind of error

A/B test for comparing two samples

- **Example:** Among babies born at some hospital, is there an association between birth weight and whether the mother smokes?
- **Null hypothesis:** The distribution of birth weights is the same for babies with smoking mothers and non-smoking mothers.
- **Inferential Idea:** If maternal smoking and birth weight were not associated, then we could simulate new samples by replacing each baby's birth weight by a randomly picked value from among all the birth weights.
- **Simulating the test statistic under the null:**
  - Permute (shuffle) the outcome column many times. Each time:
    - Create a shuffled table that pairs each individual with a random outcome.
    - Compute a sampled test statistic that compares the two groups, such as the difference in mean birth weights.

The 80th percentile is the value in a set that is at least as large as 80% of the elements in the set

For `s = [1, 7, 3, 9, 5]`, `percentile(80, s)` is 7  
The 80th percentile is ordered element 4:  $(80/100) * 5$

For a percentile that does not exactly correspond to an element, take the next greater element instead

`percentile(10, s)` is 1      `percentile(20, s)` is 1  
`percentile(21, s)` is 3      `percentile(40, s)` is 3

- `minimize` must take in a function whose arguments are numerical, and returns an array of those numerical arguments
- If the function `rmse(a, b)` returns the root mean squared error of estimation using the line “estimate = ax + b”,
  - then `minimize(rmse)` returns array `[a0, b0]`
  - `a0` is the slope and `b0` the intercept of the line that minimizes the rmse among lines with arbitrary slope `a` and arbitrary intercept `b` (that is, among all lines)

Population (fixed) → Sample (random) → Statistic (random)  
A 95% **Confidence Interval** is an interval constructed so that it will contain the true population parameter for approximately 95% of samples  
For a particular sample, the generated interval either contains the true parameter or it doesn't; the process works 95% of the time  
**Bootstrap:** When we wish we could sample again from the population, instead sample from the *original large random sample the same number of times as there are data-points in the sample*

Using a confidence interval to test a hypothesis about a numerical parameter:

- Null hypothesis: **Population parameter = x**
- Alternative hypothesis: **Population parameter ≠ x**
- Cutoff for P-value: *p*%
- Method:
  - Construct a (100-*p*)% confidence interval for the population parameter
  - If x is not in the interval, reject the null
  - If x is in the interval, fail to reject the null

The Central Limit Theorem (CLT)

If the sample is large, and drawn at random with replacement, Then, *regardless of the distribution of the population,*

**the probability distribution of the sample average (or sample sum) is roughly bell-shaped**

- Fix a large sample size
- Draw all possible random samples of that size
- Compute the mean of each sample
- You’ll end up with a lot of means
- The distribution of those is the *probability distribution of the sample mean*
- It’s roughly normal, centered at the population mean
- The SD of this distribution is the (population SD) /  $\sqrt{\text{sample size}}$

Choosing sample size so that the 95% confidence interval is small

- CLT says the distribution of a sample proportion is roughly normal, centered at the true population proportion
- **95% confidence interval:**
  - Sample proportion ± 2 SDs of the sample proportion
- **CI Width** = 4 SDs of the sample proportion  
= 4 x (SD of 0/1 population) /  $\sqrt{\text{sample size}}$
- The SD of a 0/1 population is less than or equal to 0.5

Expression	Description
<code>percentile(n, arr)</code>	Returns the n-th percentile of array <code>arr</code>
<code>np.std(arr)</code>	Return the standard deviation of an array <code>arr</code> of numbers
<code>minimize(fn)</code>	Return an array of arguments that minimize the function <code>fn</code>
<code>tbl1.append(tbl2)</code> <code>tbl1.append(row)</code>	Append a row or all rows of <code>tbl2</code> , mutating <code>tbl1</code> . Appended object and <code>tbl1</code> must have identical columns.
<code>table.rows</code>	All rows of a table; used in <code>for row in table.rows:</code>
<code>table.row(i)</code>	Return the row of a table at index <i>i</i>
<code>row.item(j)</code>	Returns item <i>j</i> from some row



**Mean (or average):** Balance point of the histogram

Standard deviation (SD) =				
root 5	mean 4	square of 3	deviations from 2	average 1

Measures roughly how far off the values are from average

Most values are within the range “average ± z SDs”

- z measures “how many SDs above average”
- If z is negative, the value is below average
- z is a value in **standard units**
- Chebyshev: At most  $1/z^2$  are z or more SDs from the mean
- Almost all standard unit values are in the range (-5, 5)
- Convert a value to standard units: (value - average) / SD
- z \* SD + average is the original value

Percent in Range	All Distributions	Normal Distribution
average ± 1 SD	at least 0%	about 68%
average ± 2 SDs	at least 75%	about 95%
average ± 3 SDs	at least 88.888...%	about 99.73%

**Correlation Coefficient (r)** =

average of	product of	x in standard units	and	y in standard units
---------------	---------------	------------------------	-----	------------------------

Measures how clustered the scatter is around a straight line

- $-1 \leq r \leq 1$  ; r = 1 (or -1) if the scatter is a perfect straight line
- r is a pure number, with no units

Regression for y and x in standard units:  $y_{predicted,su} = r * x_{su}$

The regression line minimizes the root mean square error among all lines used to predict y from x.  
The slope and intercept found by linear regression are unique.  
**Fitted value:** height of the regression line at some x: a\*x + b  
**Residual:** difference between y and regression line height at x

$$y_{predicted} = slope * x + intercept$$

$$\text{slope of the regression line} = r \cdot \frac{\text{SD of } y}{\text{SD of } x}$$

$$\text{intercept of the regression line} = \text{mean of } y - \text{slope} * \text{mean of } x$$

**Properties of fitted values, residuals, and the correlation r:**

- mean of fitted values = mean of y
- SD of fitted values = |r| \* (SD of y)
- mean of residuals = 0
- SD of residuals =  $\sqrt{1 - r^2}$  \* (SD of y)

The following functions were defined in lecture, but will **not** be available for use during the final exam. If you would like to use one of the functions, you must define it yourself.

- standard\_units
- correlation
- slope
- intercept
- fitted\_values
- residuals
- prediction\_at

- **Regression Model:** y is a linear function of x + normal "noise"
- The errors are randomly sampled from a normal distribution that has mean 0
- Under this model, residual plot looks like a formless cloud

**Prediction Intervals (assuming the regression model)**

- Creating an interval of predictions of the true value of y based on a specified value of x
- Steps for creating an approximate 95% prediction interval:
  - Bootstrap your original sample
  - Calculate the slope and intercept of the regression line based on the new sample
  - Calculate slope \* x + intercept, for the given x
  - Repeat the above steps many times and keep track of all of your fitted values
  - Create the prediction interval by taking the middle 95% of all the fitted values

**Distance between two points**

- Two numerical attributes x and y:  $D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$ .

- Three numerical attributes x, y, and z:

$$D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$$

**k-Nearest Neighbors Classifier**

Choose k to be odd. To find the k nearest neighbors of an example:

- Find the distance between the example and each example in the training set
- Augment the training data table with a column containing all the distances
- Sort the augmented table in increasing order of the distances
- Take the top k rows of the sorted table

To classify an example into one of two classes:

- Find its k nearest neighbors
- Take a majority vote of the k nearest neighbors to see which of the two classes appears more often
- Assign the example the class that wins the majority vote

**Accuracy of a classifier:** The proportion of examples in the data set that are classified correctly