

Data 8 Midterm Study Guide — Page 1

- Group by some *treatment* and measure some *outcome*
- Simplest setting: a *treatment group* and a *control group*
- If the *outcome* differs between these two groups, that's evidence of an *association* (or *relation*)
 - E.g., the top-tier chocolate eaters died of heart disease at a lower rate (12%) than chocolate abstainers (17%)
- If the two groups are similar in all ways but the *treatment* itself, a difference in the *outcome* is evidence of *causality*
- When a group is divided *randomly*, it's unlikely that there are systematic differences between sub-groups

Operation	Operator	Example	Value
Addition	+	2 + 3	5
Subtraction	-	2 - 3	-1
Multiplication	*	2 * 3	6
Division	/	7 / 3	2.66667
Remainder	%	7 % 3	1
Exponentiation	**	2 ** 0.5	1.41421

`more_than_1 = 2 + 3`

Name

Any expression

- 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)

Growth rate: The rate of increase per unit time

- After one time unit, a quantity *x* growing at rate *g* will be
$$x * (1 + g)$$
- After *t* time units, a quantity *x* growing at rate *g* will be
$$x * (1 + g) ** t$$
- If *after* and *before* are measurements of the same quantity taken *t* time units apart, then the growth rate is
$$(after/before) ** (1/t) - 1$$

A **range** is an array of consecutive numbers

- `np.arange(end)`:
An array of increasing integers from 0 up to *end*
- `np.arange(start, end)`:
An array of increasing integers from *start* up to *end*
- `np.arange(start, end, step)`:
A range with *step* between consecutive values

The range always includes *start* but excludes *end*

The **group** method aggregates all rows with the same value for a column into a single row in the result

- First argument: Which column to group by
- Second argument: How to combine values
 - `len` — number of grouped values (default)
 - `sum` — total of all grouped values
 - `list` — list of all grouped values

The **pivot** method aggregates by a combination of columns and organizes the result into a grid

- First argument: Which column gives the pivot columns
- Second argument: Which column gives the pivot rows
- Third argument: (Optional) Column of pivot values
- Fourth argument: (Optional) How to combine values

<code>max, min</code>	Maximum or minimum of a sequence
<code>str, float, int</code>	Convert a value to a string, number, or integer
<code>np.prod</code>	Multiply all elements together
<code>sum, np.sum</code>	Add all elements together
<code>all, np.all</code>	Test whether all elements are true values (non-zero numbers are true)
<code>any, np.any</code>	Test whether any elements are true values (non-zero numbers are true)
<code>np.count_nonzero</code>	Count the number of non-zero elements
<code>np.diff</code>	Difference between adjacent elements
<code>round, np.round</code>	Round each number to the nearest integer (whole number)
<code>np.cumprod</code>	A cumulative product: for each element, multiply all elements so far
<code>np.cumsum</code>	A cumulative sum: for each element, add all elements so far
<code>np.exp</code>	Exponentiate each element
<code>np.log</code>	Take the natural logarithm of each element
<code>np.sqrt</code>	Take the square root of each element
<code>np.sort</code>	Sort the elements

- < and > mean what you expect (less than, greater than)
- <= means "less than or equal"; likewise for >=
- == means "equal"; != means "not equal"
- Both numbers and strings can be compared, but a string cannot be compared to a number
- Comparing strings compares their alphabetical order

A table can be created using...

- `Table.read_table(url)` reads an existing table
- `Table(labels)` is a table with empty columns
- `Table()` is an empty table with no columns

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

- `t.with_column(label, values)`
- `t.with_columns(labels_and_values)`
- `t.with_row(row)` and `t.with_rows(rows)`

- `t.select(columns)` keeps only certain columns
- `t.drop(columns)` keeps all but certain columns
- `t.relabeled(a, b)` substitutes label *a* with *b*
- Each **column** has a label and an index (starting at 0)
 - The resulting array has one **item** per row

- `t.where(column, value)` keeps all rows containing a certain value in a column
- `t.where(condition)` keeps all rows for which the condition is true, where *condition* is a boolean array
- `t.take(row_numbers)` keeps the numbered rows
- `t.sort(column)` sorts the rows in increasing order
- Each **row** has an index (starting at 0)
 - The resulting row value has one **item** per column
 - Row items can be selected by column index or label

Tables can generate charts using these methods:

- `barh` draw a horizontal bar chart
- `hist` draw a histogram (later)
- `scatter` draw a scatter diagram (later)

Data 8 Midterm Study Guide — Page 2

Types of values: All values in a column should have the same type **and** be comparable to each other in some way

- **Categorical** — Each value is from a fixed inventory
 - May or may not have an ordering
 - Categories are the same or different
 - Allows grouping by value (**group**, **pivot**, **join**)
- **Numerical** — Each value is from a fixed scale
 - Numerical measurements are ordered
 - Allows binning by value (**bin**, **hist**)

Binning is counting the number of numerical values that lie within ranges, called bins.

- Bins are defined by their lower bounds (inclusive)
- The upper bound is the lower bound of the next bin

188, 170, 189, 163, 183, 171, 185, 168, 173, ...



A **histogram** has two defining properties:

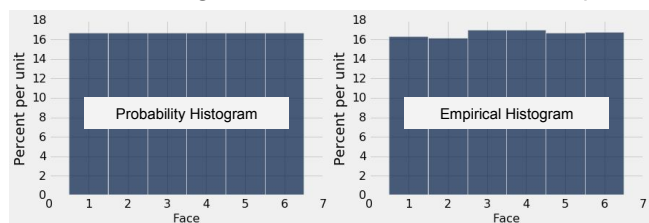
- The bins are contiguous (though some might be empty) and are drawn to scale
- The **area** of each bar is proportional to the number of entries in the bin

A histogram drawn *on the density scale* has total area 1

Vertical axis units: Proportion / Unit on the horizontal axis

Height of a bar: (proportion of entries in bin) / (width of bin)

- A histogram of proportions of all possible outcomes of a *known* random process is called a *probability histogram*
- A histogram is a summary visualization of a *distribution*
- A histogram of proportions of actual outcomes generated by sampling is called an *empirical histogram*
- When a histogram is created from data, it is empirical



Sampling from a population means selecting a subset

Probability sample: Use a random procedure for which you *know in advance* the chance that each subset of the population will enter the sample (called the *distribution*)

Uniform random sample example: choose each movie at random with equal chance

Each item has equal chance of being drawn

- **With replacement:** Items can be drawn more than once, and no draw affects the chance of any other draw
- **Without replacement:** Once an item is selected, it cannot be selected again

```
def spread(values):
    return max(values) - min(values)

for i in np.arange(12):
    print(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: print, assign, hist, etc.

Law of averages: If a chance experiment is **repeated independently** under identical conditions, in the long run:

The proportion of times that an event occurs gets closer and closer to the theoretical probability of the event

Biased estimate: On average across all possible samples, the estimate is either too high or too low.

- Bias creates a systematic error in one direction.
- Good estimates typically have low bias.
- The value of an estimate **varies** across samples.
- High variability makes it hard to estimate accurately.
- Good estimates typically have low variability.

Mean (or average): Balance point of the histogram

- **Not** the “half-way point” of the data; the mean is not the median unless the histogram is symmetric
- If the histogram is skewed, then the mean is pulled away from the median in the direction of the tail

Standard deviation (SD) =

root	mean	square of	deviations from	average
5	4	3	2	1

Measures roughly how far off the values are from average

“average \pm z SDs”

- z measures “how many SDs above average”
- If z is negative, the value is below average
- z is called **standard units**
- Almost all standard units are in the range (-5, 5)
- Convert a value to standard units: $z = (\text{value} - \text{average}) / \text{SD}$

Percent in Range	All Distributions	Normal Distribution
average \pm 1 SD	at least 0%	about 68%
average \pm 2 SDs	at least 75%	about 95%
average \pm 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

- $-1 \leq r \leq 1$
- $r = \pm 1$ if the scatter is a perfect straight line
- r is a pure number, with no units
- r is not affected by changing units of measurement
- r is not affected by switching the horizontal and vertical axes

Regression to the mean: a statement about x and y pairs

- Measured in *standard units*
- Describing the deviation of x from 0 (the average of x's)
- And the deviation of y from 0 (the average of y's)

On average, y deviates from 0 less than x deviates from 0

$$\text{Regression Line } y(\text{su}) = r \times x(\text{su})$$

Correlation

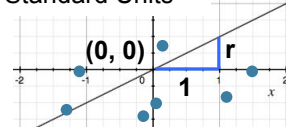
In original units, the regression line has this equation:

$$\frac{\text{estimate of } y - \text{average of } y}{\text{SD of } y} = r \times \frac{\text{the given } x - \text{average of } x}{\text{SD of } x}$$

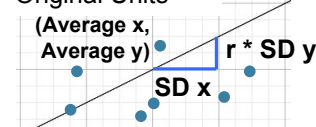
y in standard units

x in standard units

Standard Units



Original Units



Data 8 Midterm Study Guide — Page 3

```
census = Table.read_table(census_url)
census.labels
census.num_columns
census.num_rows
census.row(5)
census.column(2)
census.row(0).item(2) == census.column(2).item(0)

census.where('AGE', 40)
census.where(census.column(2) < 40)
census.where(np.logical_or(census.column('AGE') == 18,
                           census.column('AGE') == 19))
census.where(np.logical_and(census.column('AGE') >= 18,
                           census.column('AGE') <= 19))

def age_group(age):
    if age < 2:
        return 'Baby'
    elif age < 13:
        return 'Child'
    elif age < 20:
        return 'Teen'
    else:
        return 'Adult'

population = census.with_columns([
    'Age Group', census.apply(age_group, 'AGE'),
    'Male/Female', census.column('SEX'),
    'Population', census.column('2014')])

population.group('Age Group')
# Creates a table with Labels: Age Group | count

sums = population.select([0, 2]).group('Age Group', sum)
# Creates a table with Labels: Age Group | Population sum

population.pivot('Age Group', 'Male/Female')
# Counts Labeled: Male/Female | Adult | Baby | Child | Teen

population.join('Age Group', sums)
# Adds a column for the Population sum of each age group

top = Table.read_table('top_movies.csv')
top.take([3, 18, 100])
top.take(np.arange(0, top.num_rows, 40))
np.random.randint(3, 8) # select randomly from 3, 4, 5, 6, 7
start = np.random.randint(0, 10)
top.take(np.arange(start, top.num_rows, 10))

die = Table().with_column('Face', [1, 2, 3, 4, 5, 6])
dice_bins = np.arange(0.5, 7, 1)
die.hist(bins=dice_bins)
die.sample(10, with_replacement=True)

for i in np.arange(5):
    print(i)

# Generating a table of dice rolls
def roll_once(n):
    return np.random.randint(1, n+1)
rolls = Table(['First roll', 'Second roll'])
for i in np.arange(1000):
    row = [roll_once(6), roll_once(6)]
    rolls.append(row)
rolls.hist(bins=dice_bins)
```

<code>np.average</code> , <code>np.mean</code>	Average of values
<code>np.std</code>	Standard deviation of values
<code>np.var</code>	Variance of values (standard deviation squared)
<code>Table.append</code>	Append a row or all rows of a table.
<code>Table.append_column</code>	Appends a column to the table.
<code>Table.apply</code>	Returns an array where a function is applied to each item in a column.
<code>Table.bin</code>	Group values by bin and compute counts per bin by column.
<code>Table.column</code>	Return the values of a column as an array.
<code>Table.drop</code>	Return a table with only columns other than selected label or labels.
<code>Table.group</code>	Group rows by unique values in a column; count or aggregate other values.
<code>Table.join</code>	Generate a table with the columns of self and other, containing rows for all values of a column that appear in both tables.
<code>Table.pivot</code>	Group rows by unique values in two columns; count or aggregate values from a third column.
<code>Table.relabel</code>	Change the label of a column
<code>Table.relabeled</code>	Return a new table with a label changed.
<code>Table.sample</code>	Returns a new table where k rows are randomly sampled from the original table.
<code>Table.select</code>	Return a table with only the selected columns.
<code>Table.sort</code>	Return a table of rows sorted according to the values in a column.
<code>Table.take</code>	Return a table of the rows taken by index.
<code>Table.where</code>	Return a table of the rows for which the column matches a value or is a non-zero value.
<code>Table.with_column</code> <code>Table.with_columns</code>	Return a table with an additional or replaced column or columns.
<code>Table.with_row</code> <code>Table.with_rows</code>	Return a table with an additional row or rows.

```
annual_growth_rate = 0.035
```

```
ten_year_growth_rate = (1 + annual_growth_rate) ** 10 - 1
```

```
ten_year_growth_rate # evaluates to: 0.410598760621121
```

```
baseline_high = 14.48
```

```
highs = np.array([baseline_high - 0.880, baseline_high - 0.093,
                  baseline_high + 0.105, baseline_high + 0.684])
```

```
highs # evaluates to: array([ 13.6 , 14.387, 14.585, 15.164])
(9/5) * highs + 32 # array([ 56.48 , 57.89, 58.253 , 59.295])
```

```
def percent(x):
```

```
    if x < 1e-8:
```

```
        return 0.0
```

```
    elif x < 0.00005:
```

```
        return 100 * x
```

```
    else:
```

```
        return round(100 * x, 2)
```

```
percent(1/6) # returns 16.67
```

```
percent(1/6000) # returns 0.02
```

```
percent(1/60000) # returns 0.0016666666666666668
```

```
percent(1/60000000000) # returns 0.0
```

```
Table(['Odd', 'Even']).with_row([3, 4])
```

```
Table(['Odd', 'Even']).with_rows([3, 4], [5, 6], [7, 8])
```

```
Table().with_column('Odd', [3, 5, 7])
```

```
Table().with_columns(['Odd', [3, 5, 7], 'Even', [4, 6, 8]])
```

```

def standard_units(any_numbers):
    return (any_numbers - np.mean(any_numbers))/np.std(any_numbers)

def correlation(t, x, y):
    return np.mean(standard_units(t.column(x))*standard_units(t.column(y)))

def slope(table, x, y):
    r = correlation(table, x, y)
    return r * np.std(table.column(y))/np.std(table.column(x))

def intercept(table, x, y):
    a = slope(table, x, y)
    return np.mean(table.column(y)) - a * np.mean(table.column(x))

def fit(table, x, y):
    a = slope(table, x, y)
    b = intercept(table, x, y)
    return a * table.column(x) + b

# Regression effect
heights = Table.read_table('heights.csv')
six_foot_fathers = heights.where(np.round(heights.column('father')) == 72)
six_foot_fathers.hist('son', bins=np.arange(55.5, 80, 1))
np.count_nonzero(six_foot_fathers.column('son') < 72) / six_foot_fathers.num_rows
np.mean(six_foot_fathers.column('son'))

# Graph of averages
fathers = heights.column('father')
sons = heights.column('son')
heights_su = Table().with_columns([
    'father (su)', standard_units(fathers),
    'son (su)', standard_units(sons)])
rounded = heights_su.with_column('father (su)', np.round(father_su))
rounded.join('father (su)', rounded.group(0, np.average)).scatter(0, s=80)

# Regression model
def signal_and_noise(x, r):
    return r * x + np.random.normal() * (1-r**2)**0.5

def regression_model(r, sample_size):
    pairs = Table(['x', 'y'])
    for i in np.arange(sample_size):
        x = np.random.normal()
        y = signal_and_noise(x, r)
        pairs.append([x, y])
    return pairs

regression_model(1/2, 1000).scatter('x', 'y')

def compare(true_r, sample_size):
    pairs = regression_model(true_r, sample_size)
    estimated_r = correlation(pairs, 'x', 'y')
    pairs.scatter('x', 'y', fit_line=True, s=10)
    plt.plot([-3, 3], [-3 * true_r, 3 * true_r], color='g', lw=4)
    print("The true r is ", true_r, " and the estimated r is ", estimated_r)

compare(0.5, 1000) # Compare true line to regression line

```