
# Week 5 – **Review, Project 1**

Slides by Suraj Rampure

Fall 2017

# Histogram Review

- The area of each bar in the histogram represents the proportion of the total population which is found in the specific range
- Area of a rectangle = **Width** $*$ **Height** in a histogram (and for all rectangles)
- Area is in proportion, width is in a certain specified unit (e.g square inches), therefore, height (area ÷ width) has units proportion per whatever the unit is on the x-axis (e.g proportion per square inch)
- The sum of the areas should add up to 1, since they are all proportions
- Sometimes, people will use percentages instead of proportions for areas/height. In this case, every- thing on the y-axis would be multiplied by 100 and the areas should add up to 100 (instead of 1)

**Taken from http://fhdkmrn.github.io/resources/cs8/Histogram_Extra_Practice.pdf**

# Histogram Practice – Find the Missing Height

Consider the following table of average prices paid for textbook per student:

| $0-$50 | $50-$75 | $75-$150 | $150-$300 |
|---|---|---|---|
| .0075 | .002 | x | y |

Given that:

- There are 20,000 students
- There are 8,000 students that spend $150-$300
- The y-axis of this histogram is “proportion of students per dollar”

Find the missing heights **x** and **y**.

Given that:

- There are 20,000 students
- There are 8,000 students that spend $150-$300
- The y-axis of this histogram is "proportion of students per dollar"

| $0-$50 | $50-$75 | $75-$150 | $150-$300 |
|---|---|---|---|
| .0075 | .002 | x | y |

y-rectangle

Area = Proportion $\longrightarrow$ (width)(height) $= \frac{8000}{2000} = \frac{2}{5}$

$$y = \text{height} = \frac{2}{5} \cdot \left(\frac{1}{\text{width}}\right) = \frac{2}{5 \cdot 150} \approx 0.0027$$

x-rectangle

Total Area = 1

$$(50)(0.0075) + (25)(0.002) + (75)(x) + (150)(0.0027) = 1$$

$$x = \frac{1 - 0.375 - 0.05 - 0.4}{75} = \frac{0.175}{75} = 0.0023$$

# Joins

The **join** function combines two tables into one. To use it, you specify two columns (one from each table) to be used as "keys", which are the values that it will check for matches in.

`names: Table`

| name | email |
|---|---|
| john | john@berkeley.edu |
| jack | jackisawesome@gmail.com |
| jim | j.stanfordsucks@stanford.edu |
| jeffery | jeffbezos@amazon.com |
| james | lebron@cavs.com |
| jay | contactjay@hotmail.com |

`SIDs: Table`

| first_name | SID |
|---|---|
| jay | 30313414 |
| john | 87634123 |
| jack | 88954446 |
| jack | 24659076 |

To join the above two tables, we'd probably want to use the column **`name`** from the first table and **`first_name`** from the second table as our keys. Upon calling **join**, the function will look for matches between these two columns.

# Joins

`names: Table`

| name | email |
|---|---|
| john | john@berkeley.edu |
| jack | jackisawesome@gmail.com |
| jim | j.stanfordsucks@stanford.edu |
| jeffery | jeffbezos@amazon.com |
| james | lebron@cavs.com |
| jay | contactjay@hotmail.com |

`SIDs: Table`

| first_name | SID |
|---|---|
| jay | 30313414 |
| john | 87634123 |
| jack | 88954446 |
| jack | 24659076 |

```
names.join("name", SIDs, "first_name")
```

The general syntax for using joins is **`<table1>.join(<column1>, <table2>, <column2>)`**. As per usual, column names are strings and table names are variables. Notice that there are two ways to join any two tables, since we could've done this the other way around. The difference is that **`joins`** adds the values from the second table to the end of the first table – you should experiment with this!

# Joins

**names: Table**

| name | email |
|---|---|
| john | john@berkeley.edu |
| jack | jackisawesome@gmail.com |
| jim | j.stanfordsucks@stanford.edu |
| jeffery | jeffbezos@amazon.com |
| james | lebron@cavs.com |
| jay | contactjay@hotmail.com |

**SIDs: Table**

| first_name | SID |
|---|---|
| jay | 30313414 |
| john | 87634123 |
| jack | 88954446 |
| jack | 24659076 |

**join**

| name | email | SID |
|---|---|---|
| jack | jackisawesome@gmail.com | 88954446 |
| jack | jackisawesome@gmail.com | 24659076 |
| jay | contactjay@hotmail.com | 30313414 |
| john | john@berkeley.edu | 87634123 |

Joins looks for **all** matches between the two tables, and outputs all possible combinations. Since there are two "jack" entries in the **"first_name"** column of **SIDs**, there are two lines for **"jack"** in the final table.

**Question:** If there were 3 **"jack"** rows in the **names** table, and 4 **"jack"** rows in the **SIDs** table, how many **"jack"** rows would be in the joined table, if we used the same columns as keys?

# Joins

**names: Table**

| name | email |
|---|---|
| john | john@berkeley.edu |
| jack | jackisawesome@gmail.com |
| jim | j.stanfordsucks@stanford.edu |
| jeffery | jeffbezos@amazon.com |
| james | lebron@cavs.com |
| jay | contactjay@hotmail.com |

**SIDs: Table**

| first_name | SID |
|---|---|
| jay | 30313414 |
| john | 87634123 |
| jack | 88954446 |
| jack | 24659076 |

**join**

| name | email | SID |
|---|---|---|
| jack | jackisawesome@gmail.com | 88954446 |
| jack | jackisawesome@gmail.com | 24659076 |
| jay | contactjay@hotmail.com | 30313414 |
| john | john@berkeley.edu | 87634123 |

Joins looks for **all** matches between the two tables, and outputs all possible combinations. Since there are two "jack" entries in the **"first_name"** column of **SIDs**, there are two lines for **"jack"** in the final table.

**Question:** If there were 3 **"jack"** rows in the **names** table, and 4 **"jack"** rows in the **SIDs** table, how many **"jack"** rows would be in the joined table, if we used the same columns as keys?

There would be **12** – one for every possible combination of rows from the first table and second table (3 x 4).