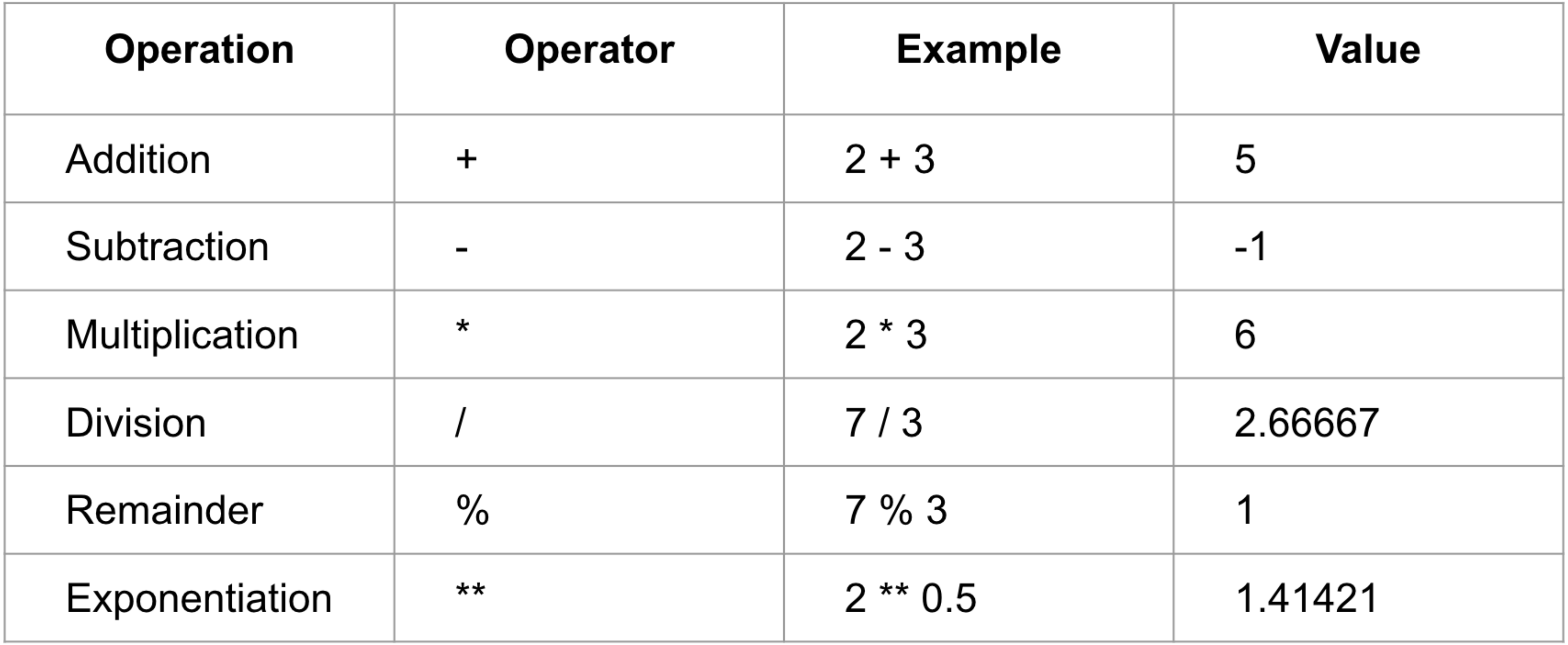
Midterm Review

Logistics

- Need to end class @ 12 today
- Last 20 minutes of class (start @ 11:40): quiz on Tidy Data
- Midterm in class next Thursday
- Midterm cheat sheet: everyone gets one 8 ½ x 11” sheet – front and back, fit what you want on it
- Will cover material up through Lecture 10 - Intro to Prediction
  - Materials covered in class, demos, homeworks are free game

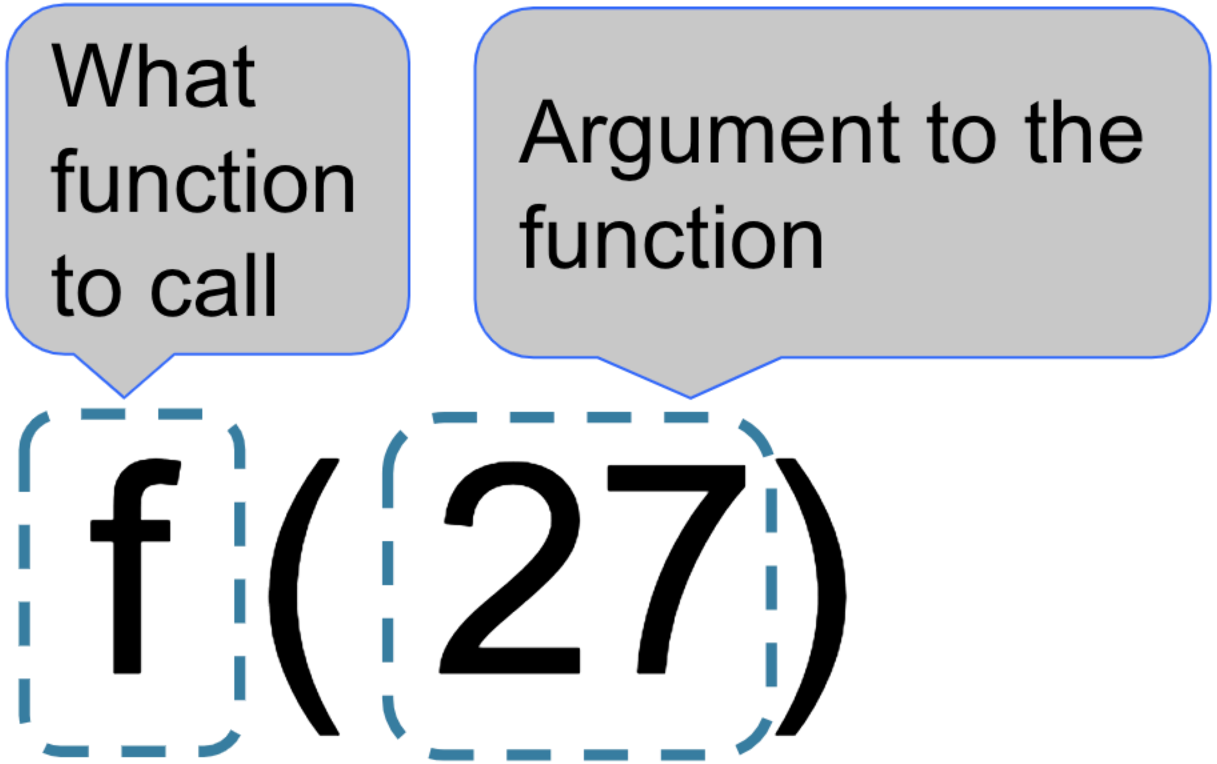
Monty Hall Problem w/ 100 Doors

Expressions



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

Functions



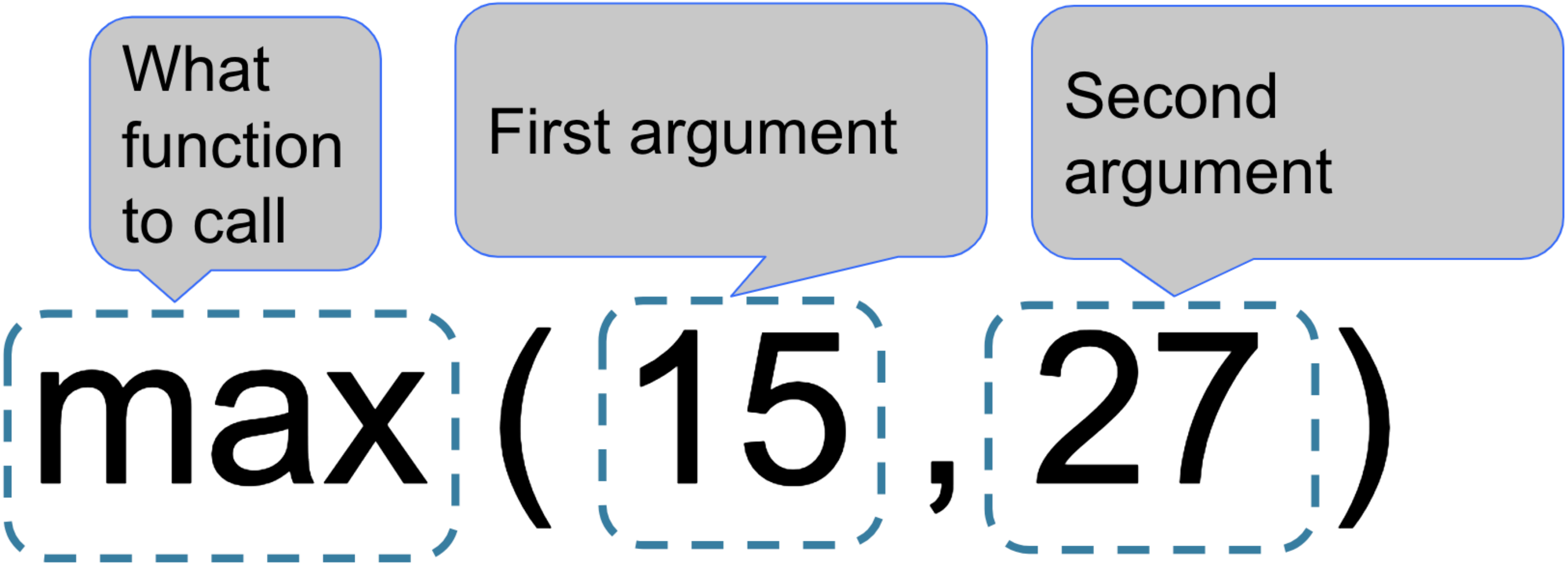
What function to call

Argument to the function

f(27)

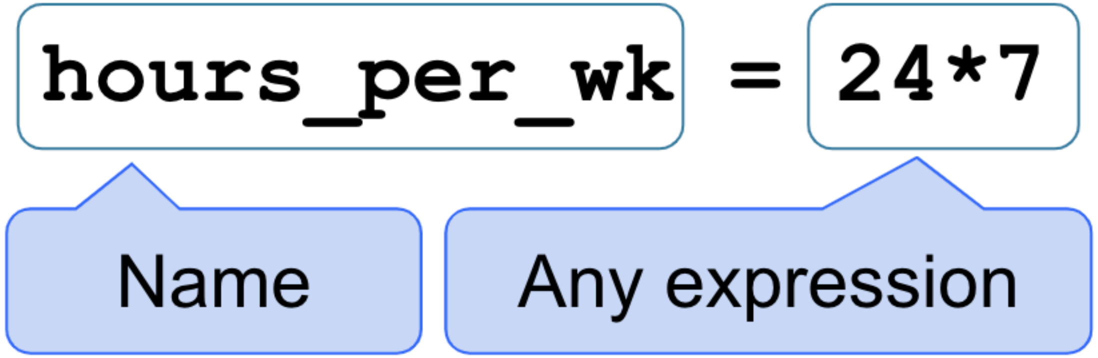
“Call f on 27”

Functions



```
max(15,27)
```

Assignment Statements



```
hours_per_wk = 24*7
```

Name

Any expression

- 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)

Data Types

We’ve seen 5 types so far:

- `int: 2`
- `float: 2.2`
- `bool: True`
- `str: 'Red fish, blue fish’`
- `Builtin_function_or_method: abs`

The `type` function can tell you the type of a value

- `type(2)`
- `type(2.2)`
- `type(True)`
- `type(’Red fish, blue fish’)`
- `type(abs)`

Conversions

- Strings that contain numbers can be converted to numbers
    - `int(‘12’)`
    - `float(‘1.2’)`

- Any value can be converted to a string
    - `str(5)`
    - `str(True)`
    - `str(abs)` ← anyone know what this would return?

- Numbers can be converted to other numeric types
    - `float(1)`
    - `int(1.2)`
    - `round(1.2)`

Lists

- Container that holds a number of objects in an order

```
L = [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’]
```

- Accessing / Indexing

```
L[0]            ’yellow’
L[1:4]          [‘red’, blue’, ‘green’]
L[3:]           [‘green’, ‘black’]
L[-1]           [‘black’]
```

- Length

```
len(L)          5
```

Lists

- Built-in methods for adding objects

```
L.append(‘pink’)
print(L)
```

```
    [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’]
```

```
L.insert(0,’white’)
print(L)
```

```
    [‘white’, ‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’]
```

```
L2 = [‘orange’, ‘cyan’, ‘magenta’]
L.extend(L2)
print(L)
```

```
    [‘white’, ‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’, ‘orange’, ‘cyan’, ‘magenta’]
```

Lists

```
L = [‘white’, ‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’,
          ‘pink’, ‘orange’, ‘cyan’, ‘magenta’]
```

- Built-in methods for removing objects

```
L.remove(‘white’)
print(L)
```

```
[‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’, ‘orange’, ‘cyan’, ‘magenta’]
```

```
del L[0]
print(L)
```

```
[‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’, ‘orange’, ‘cyan’, ‘magenta’]
```

```
L.pop()
```

```
‘magenta’
```

```
print(L)
```

```
[‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’, ‘orange’, ‘cyan’]
```

Lists

```
L = [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’,
                    ‘orange’, ‘cyan’]
```

- Other built in methods

```
L.sort()
print(L)
```

```
['black', 'blue', 'cyan', 'green', 'orange', 'pink', 'red', 'yellow’]
```

```
L.count(‘red’)
```

```
1
```

```
L.reverse()
```

```
['yellow', 'red', 'pink', 'orange', 'green', 'cyan', 'blue', 'black']
```

Control Structures

- Control structure: direct the order of execution of statements in a program

- `if / else`
  - "If the weather is nice, I will mow the lawn, otherwise I'll watch tv"

```
nice_weather = True
if nice_weather:
    mow_lawn()
else:
    watch_tv()
```

Control Structures

- `if / elif / else`
  - “If the temperature is above 80, I will swim; if it’s between 60 and 80, I will hike; otherwise, I will watch tv.”

```
if temperature > 80:
    swim()
elif temperature >= 60:
    hike()
else:
    watch_tv()
```

Control Structures

- `for` loops

```
L = [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’,
‘orange’, ‘cyan’]


for color in L:
    print color
```

```
yellow
red
pink
orange
green
cyan
blue
black
```

Control Structures

```
L = [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’,
‘orange’, ‘cyan’]

for color in L:
    if ‘e’ in color:
        print(color)
```

```
yellow
red
orange
green
blue
```

Control Structures

- while loops

```
L = [‘yellow’, ‘red’, ‘blue’, ‘green’, ‘black’, ‘pink’,
‘orange’, ‘cyan’]


idx = 0
while idx < 3:
    print(L[idx])
    idx += 1
```

```
yellow
red
pink
```

List Comprehension

- A concise way to create lists

```
list = [ expression for item in list if conditional ]
```

```
squares = [x**2 for x in range(10)]
```

```
even_squares = [x**2 for x in range(10) if x**2 % 2 == 0]
```

```
odd_squares = [x**2 for x in range(10) if x**2 % 2 != 0]
```

or:

```
odd_squares = [x for x in squares if x not in even_squares]
```

Control Structures

- Loop control statements: change execution from its normal sequence
- `break`: exit out of a loop when a condition is triggered

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
for number in range(10):
    if number == 5:
        break
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Out of loop
```

Control Structures

- Loop control statements: change execution from its normal sequence

- `continue`: skip over the part of the loop where a condition is triggered, but complete the rest of the loop

```
for number in range(10):
    if number == 5:
        continue
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Number is 6
Number is 7
Number is 8
Number is 9
Out of loop
```

Control Structures

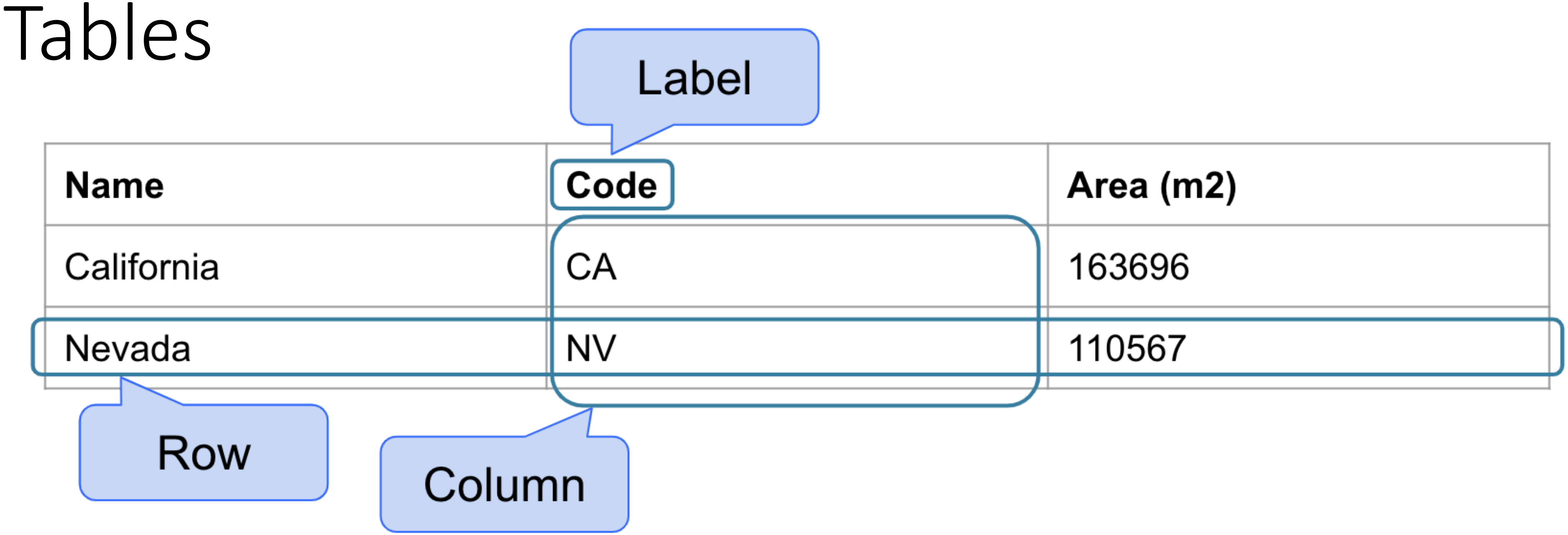
- Loop control statements: change execution from its normal sequence

- `pass`: handle the condition without the loop being impacted at all

```
for number in range(10):
    if number == 5:
        pass
    print('Number is ' + str(number))
print('Out of loop')
```

```
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9
Out of loop
```

Tables



Name	Code	Area (m2)
California	CA	163696
Nevada	NV	110567

Label
Row
Column

- A Table is a sequence of labeled columns
- Each row represents one individual
- Data within a column represents one attribute of the individuals

Table Operations

- **`Table.read_table(filename)`** – reads a table from a spreadsheet
- **`Table()`** – creates an empty table

```
t = Table.read_table(‘table_file.csv’)
```

Table Operations

- **`t.select(label)`** - constructs a new table with just the specified columns
- **`t.drop(label)`** - constructs a new table in which the specified columns are omitted
- **`t.sort(label)`** - constructs a new table with rows sorted by the specified column
- **`t.where(label, condition)`** - constructs a new table with just the rows that match the condition

Table Methods

- Creating and extending tables:
  - `Table().with-column` and `Table.read_table(csv_file)`
- Finding the size:
  - `num_rows` and `num_columns`
- Referring to columns: labels, relabeling, and indices
  - `labels` and `relabeled`; column indices start at 0
- Accessing data in a column
  - `column` takes a label or index and returns an array
- Using array methods to work with data in columns
  - `item`, `sum`, `min`, `max`, etc.
- Creating new tables containing some of the original columns:
  - `select`, `drop`

Manipulating Rows

- `t.sort(column)` sorts the rows in increasing order
- `t.take(row_numbers)` keeps the numbered rows
  - Each row has an index, starting at 0
- `t.where(`*`column`*`, are.`*`condition`*`)` keeps all rows for which a column's value satisfies a condition
  - `are.equal_to(5), are.above(20), are.below(10), are.between(30, 38),` etc
  - http://data8.org/datascience/predicates.html
- `t.where(`*`column, value`*`)` keeps all rows for which a column's value equals some particular value
- `t.with_row(`*`list`*`)` makes a new table that has another row with values in *`list`*

Types of Data

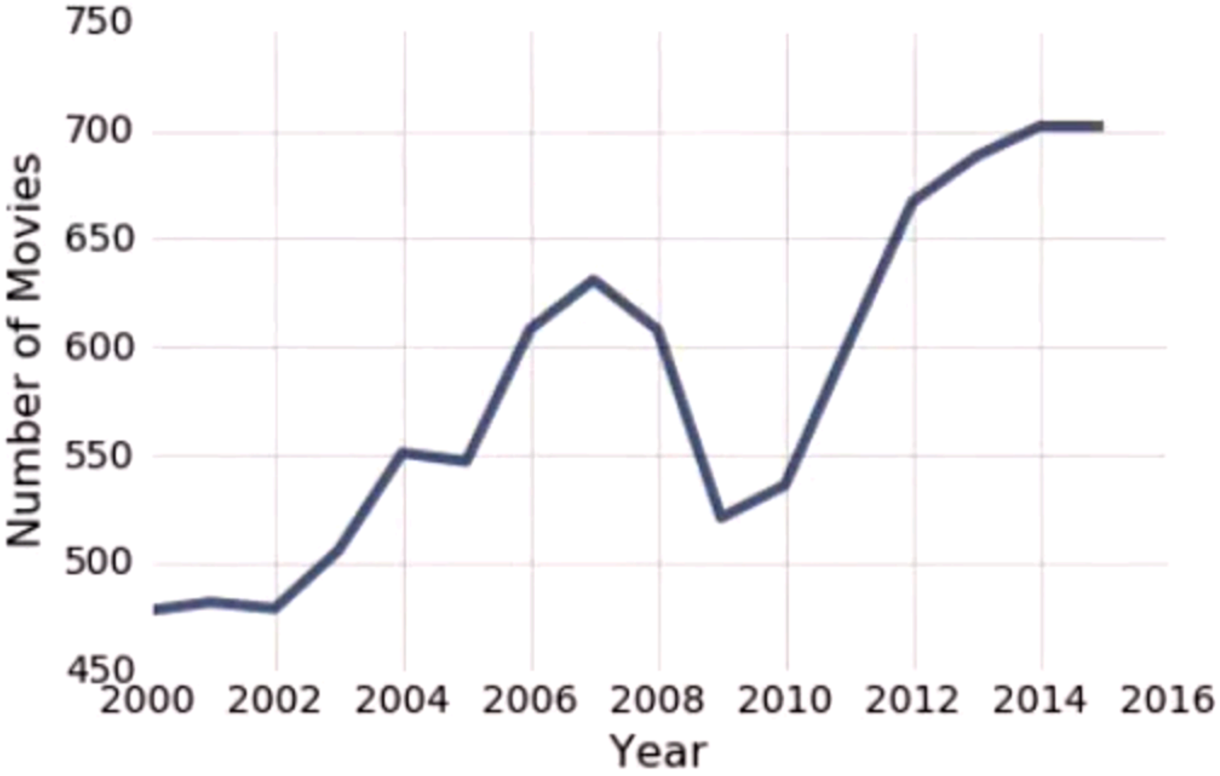
- All values in a column should be both the same type ***and*** comparable to each other in some way
  - **Numerical –** each value is from a numerical scale
    - Numerical measurements are ordered
    - Differences are meaningful
  - **Categorical –** each value is from a fixed inventory
    - May or may not have an ordering
    - Categories are the same or different

“Numerical” Data

- Just because the values are numbers doesn’t mean the variable is numerical
  - Census example had numerical SEX code (0, 1 and 2)
  - It doesn’t make sense to perform arithmetic on these “numbers”, e.g., 1 - 0 or (0+1+2)/3 are meaningless
- The variable SEX is still categorical even though they’re numbers

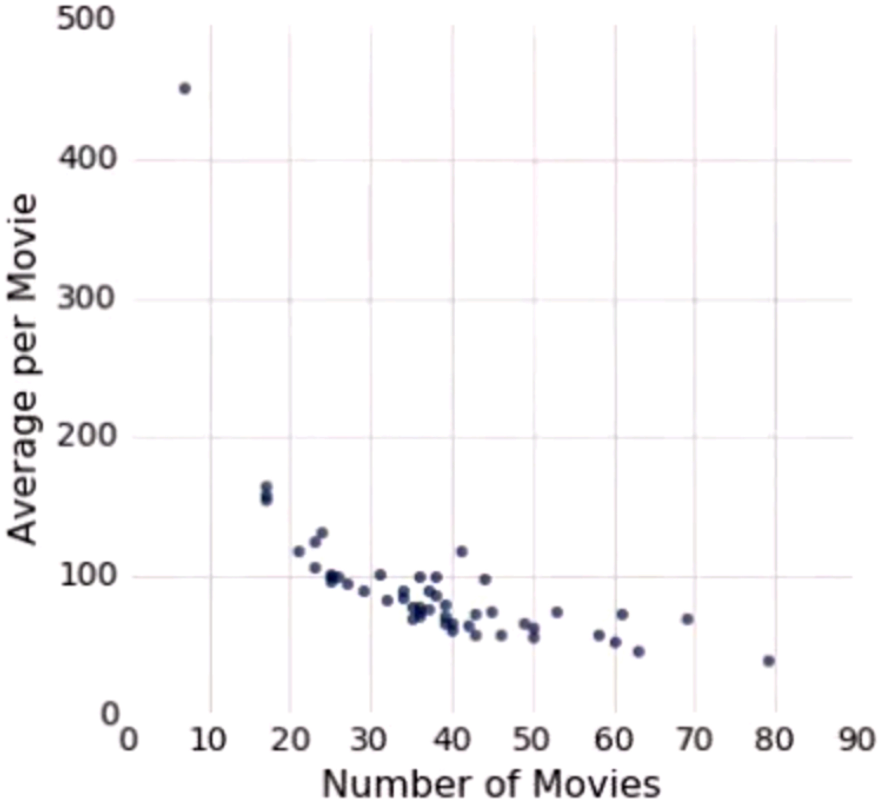
Plotting Two Numerical Variables

Line graph: `plot`



How something changes as the X-axis changes (often chronologically)

Scatter plot: `scatter`



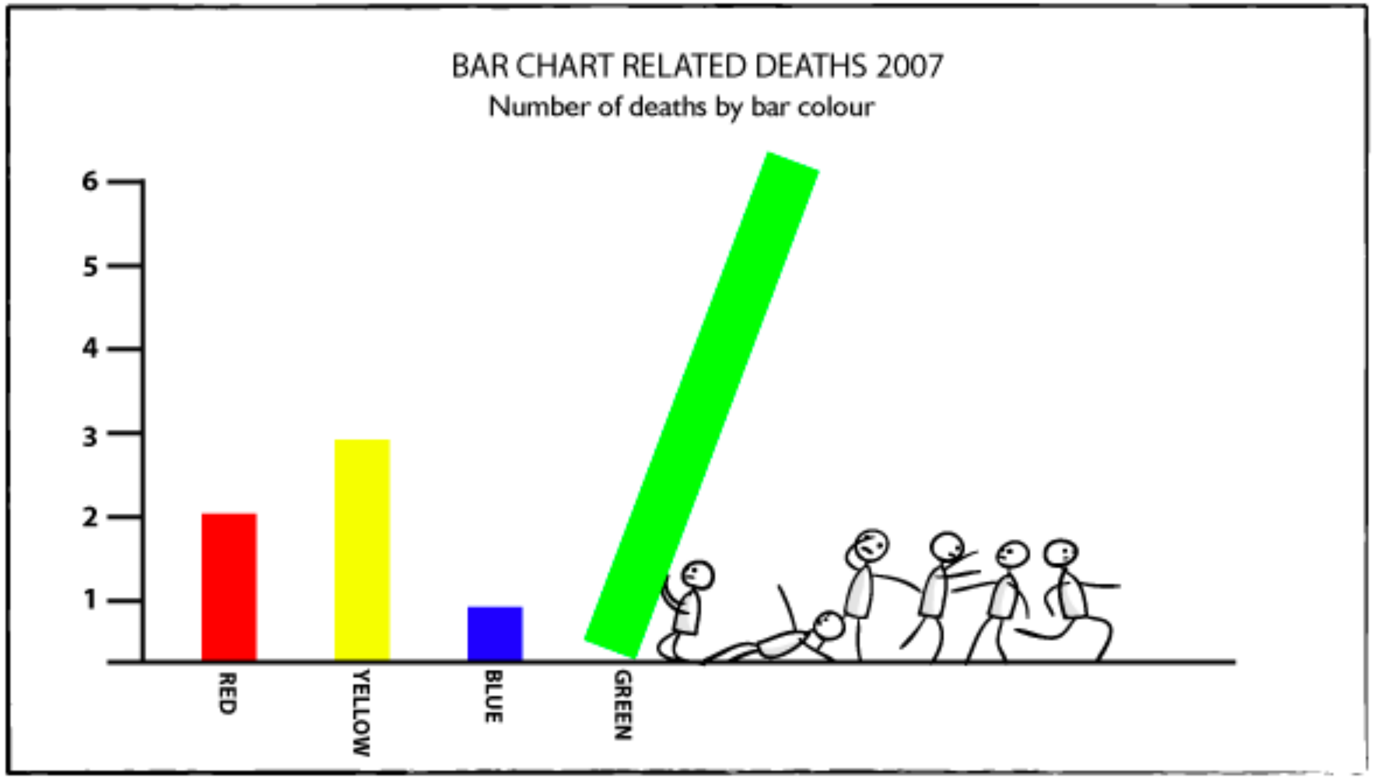
Comparing two numerical variables

Terminology

- **Individuals**: those whose features are recorded
- **Variable**: a feature or attribute
  - Variables have different values
  - Values can be categorical or numerical (and many sub-types within these)
- Each individual has ***one*** value of the variable
- **Distribution**: for each different value of the variable, what is the frequency of individuals that have that value

Categorical Visualization

- Bar charts!
  - One axis is categorical, one is numerical



Numerical Visualization

- For categorical data, visualization of distribution is easy → plot # of individuals in a category

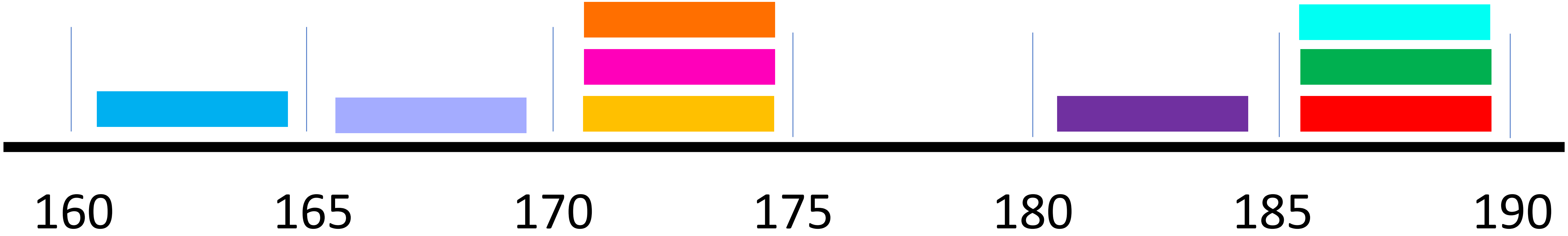
- What about for numerical data?
  - E.g., height (person A is 68.3” tall, person B is 68.4” tall, person C is 61” tall, person D is 61.5” tall, etc.)

Binning Numerical Values

- Count the number of numerical values that lie within a range or bin
  - Typical convention: Bins are defined by their lower bounds (inclusive)
  - The upper bound is the lower bound of the next bin

Binning Numerical Values

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

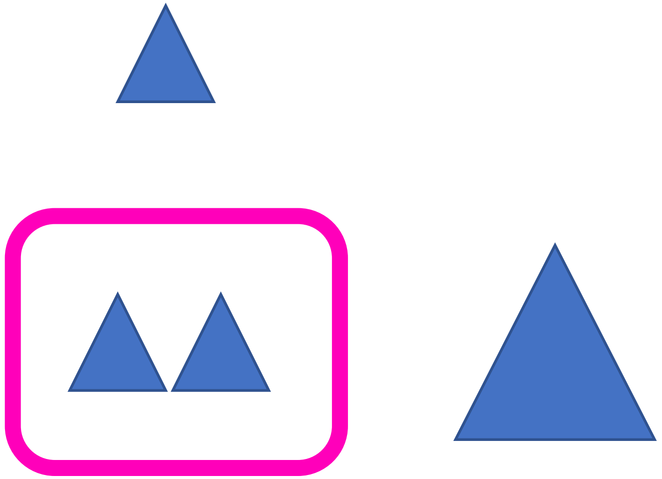


Area Principle

- ***Areas*** should be proportional to the values they represent (not length and width)

20% of the population

Which of these can be 40%?



Histograms

- Chart that displays the distribution of a numerical variable
- Uses bins → one bar corresponding to each bin
- The *area* of each par is the percent of individuals in the corresponding bin

Histograms

- Chart that displays the distribution of a numerical variable
- Uses bins → one bar corresponding to each bin
- The *area* of each par is the percent of individuals in the corresponding bin

Histogram Axes

- By default, hist uses a scale (normed=True) that ensures the area of the chart sums to 100%
- The area of each bar is a percentage of the whole
- The horizontal axis is a number line (e.g., years0, and the bin sizes don’t have to be equal to each other
- Vertical axis is numerical

Height Measures Density

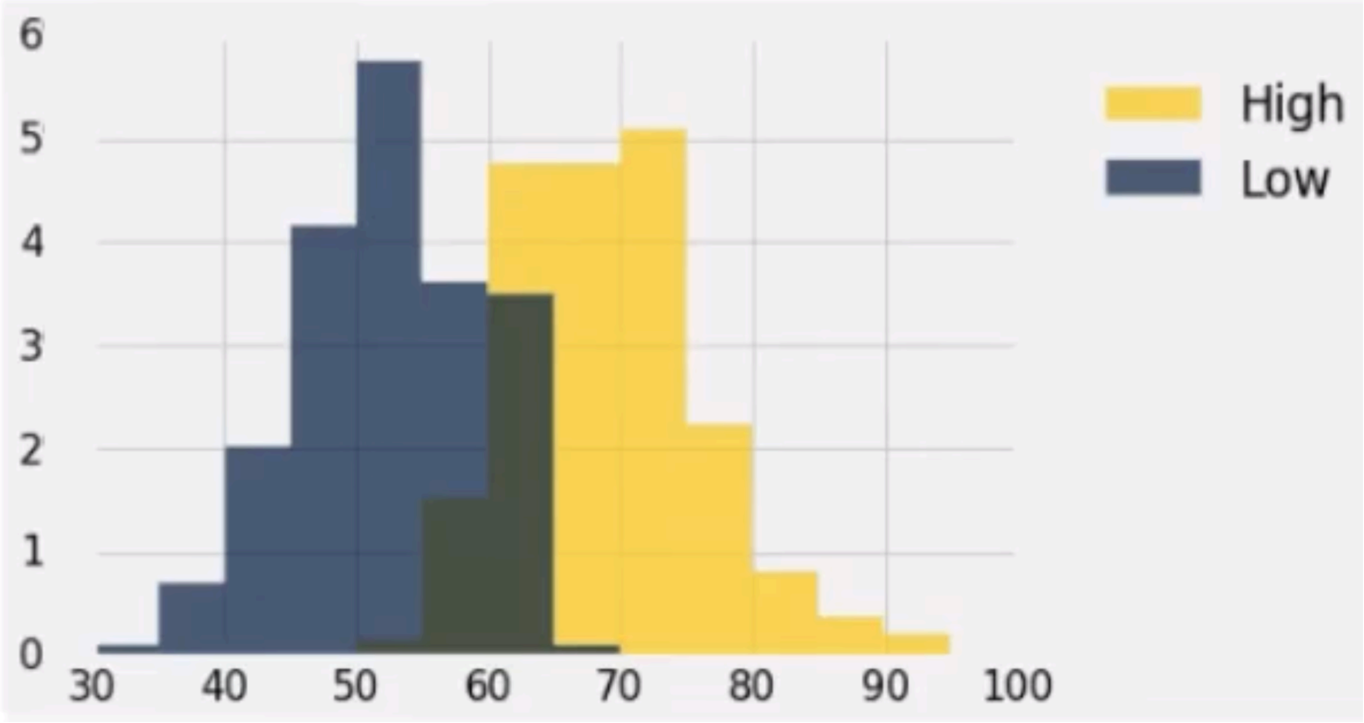
- Height $= \dfrac{\text{\% in bin}}{\text{width of bin}}$

- Height measures the percent of data in the bin ***relative to the amount of space in that bin.***
- Height measures crowdedness, or **density**
- Units: percent per unit on the horizontal axis

Discussion Questions

This histogram describes a year of daily temps

1) What proportion of days had a high temp in the range 60-69?
2) What proportion had a low of 45 or more?
3) How many days had a difference of more than 20 degrees between their high and low temperatures?



Bar Chart vs. Histogram?

Bar Chart

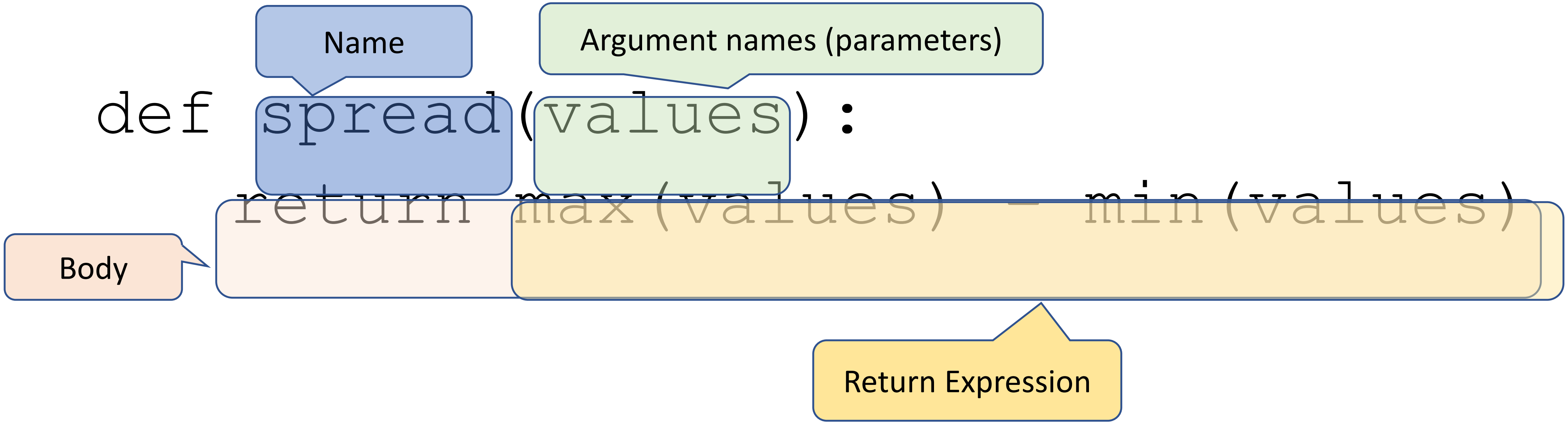
- Distribution of categorical variable
- Bars have arbitrary (but equal) widths and spacings
- Height (or length) of bars proportional to the percent of individuals

Histogram

- Distribution of numerical variable
- Horizontal axis is numerical: to scale, n gaps, bins can be unequal
- Area of bars is proportional to % of individuals; height measure density

def Statements

User-defined functions give names to blocks of code



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

Name

Argument names (parameters)

Body

Return Expression

Apply w/ Multiple Arguments

- The `apply` method creates an array by calling a function on every element in input column(s)

```
table_name.apply(no_arg_function)
table_name.apply(one_arg_function, ‘column_label’)
table_name.apply(two_arg_function,
                  ‘column_label_for_first_arg’,
                  ’column_label_for_second_arg’)
```

Group

- 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: (Optional) how to combine values
    - len – number of grouped values (default)
    - sum - total of all grouped values
    - list – list of all grouped values

```
table_name.group(‘column_label’,group_by_what)
```

Pivot

- Cross-classifies according to two categorical variables
- Produces a grid of counts or aggregated values
- Two required arguments:
  - First: variable that forms column labels of grid
  - Second: variable that forms row labels of grid
- Two optional arguments (include both or neither):
  - `values` = ‘column_label_to_aggregate’
  - `collect` = function_with_which_to_aggregate

`group` & `pivot` for cross-classification

- Classification: assign individuals to different groups based on shared properties

- When individuals have multiple features, there are many different ways to classify them.
  - e.g., we have a population of college w/ a major and the number of years in college
  - students could be classified by major, or by year, or by a combination of major and year

- `group` and `pivot` are table operations that allow us to classify individuals according to multiple variable (or to ‘cross-classify’ them)

Joining Tables by Columns

- When you have related data in multiple tables, you can ‘join’ by shared column

```
Table_1.join(‘Table_1_column_label’, Table_2,
‘Table_2_column_label')
```

“Data Mining” and Prediction

- “Data Mining” attempts to extract patterns from data
  - Associative patterns

    What data attributes occur together ?
  - Classification

    What indicates a given category ?
  - Sequential/temporal patterns

    What sequences of events occur frequently ?

Example Patterns

- Associative pattern

  When Bob is in the living room he likes to watch TV and eat popcorn with the light turned off.

- Classification

  Action movie fans like to watch Terminator, drink beer, and have pizza.

- Sequential patterns

  After coming out of the bedroom in the morning, Bob turns off the bedroom lights, then goes to the kitchen where he makes coffee, and then leaves the house.

Data Mining and Prediction

- Prediction attempts to form patterns that help to predict the next event(s) given the available input data.
  - Deterministic predictions

    If Bob leaves the bedroom before 7:00 am on a workday, then he will make coffee in the kitchen.
  - Probabilistic sequence models

    If Bob turns on the TV in the evening then he will 80% of the time go to the kitchen to make popcorn.

What to Predict

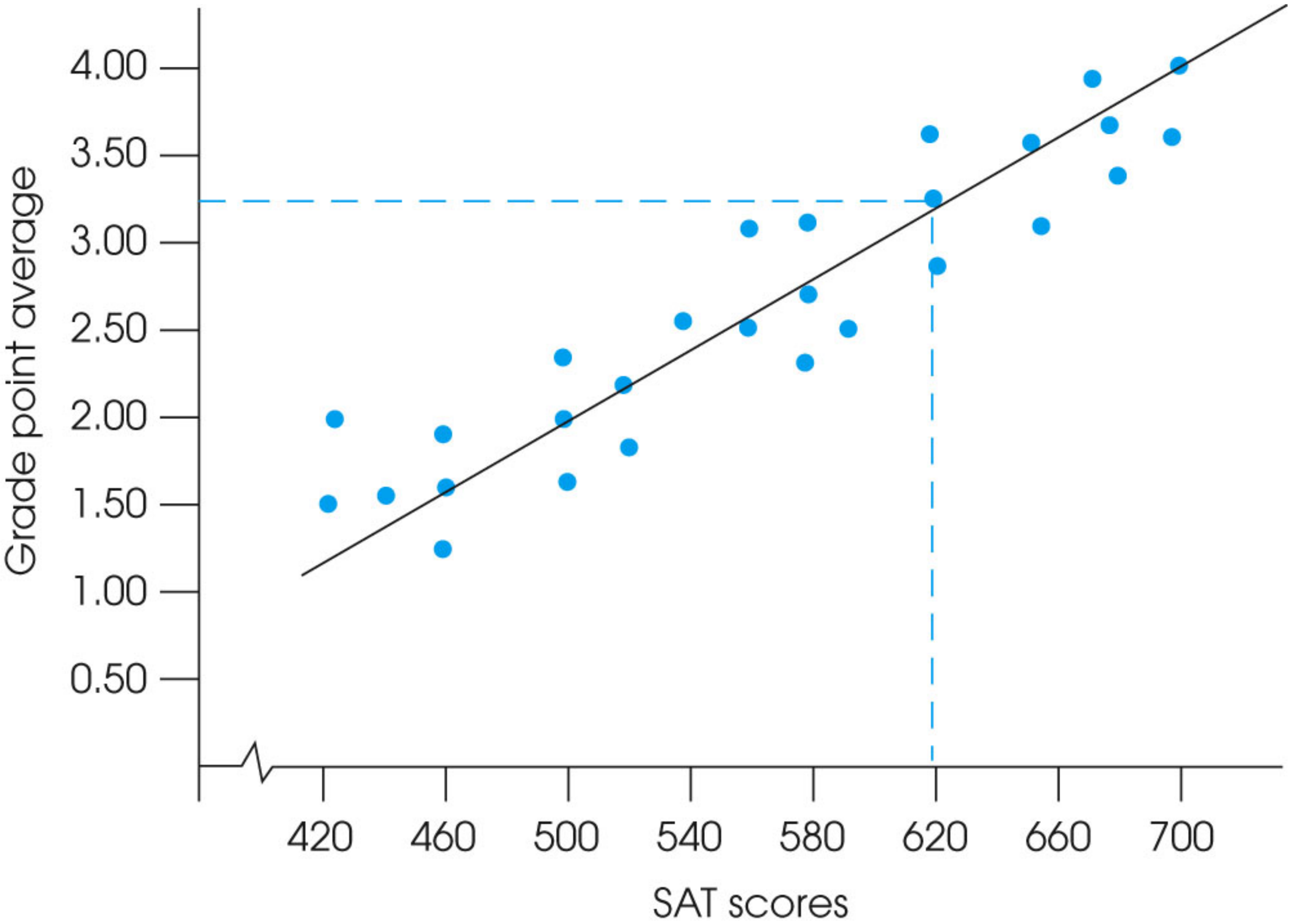
- Behavior of Individuals
    - Location
    - Tasks / goals
    - Actions
- Behavior of the Environment
    - Device behavior (e.g. heating, AC)
    - Interactions

Introduction to Linear Regression

- Any straight line can be represented by an equation of the form y = mx + b, where m and a are constants.

- m: slope constant, determines the direction and degree to which the line is tilted (= Δy – Δx)

- b: the Y-intercept, determines the point where the line crosses the Y-axis.

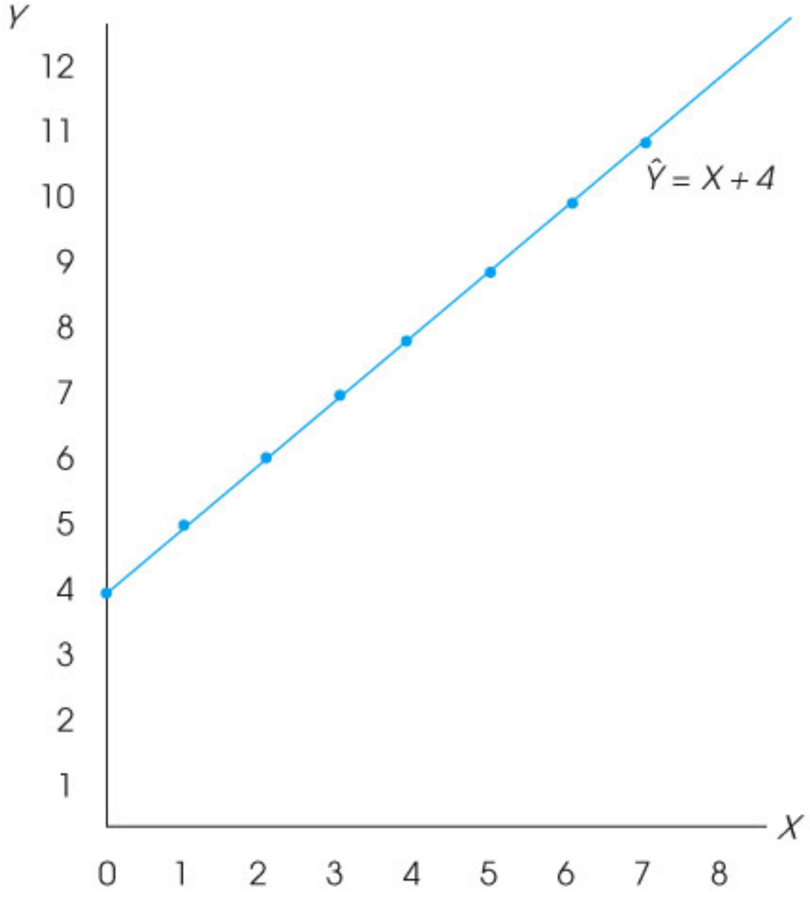

Grade point average
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
420
460
500
540
580
620
660
700
SAT scores

Introduction to Linear Regression

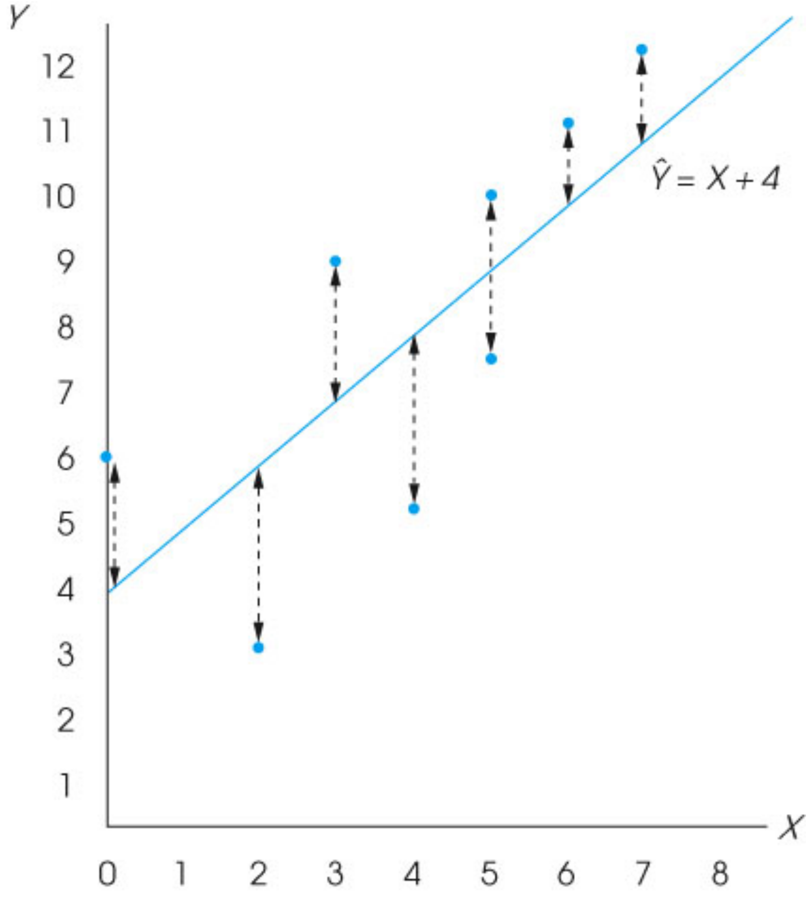
- How well a set of data points fits a straight line can be measured by calculating the distance between the data points and the line.

- The total error between the data points and the line is obtained by squaring each distance and then summing the squared values.

- Simple linear regression: find the equation of the straight line that produces the minimum sum of squared errors.

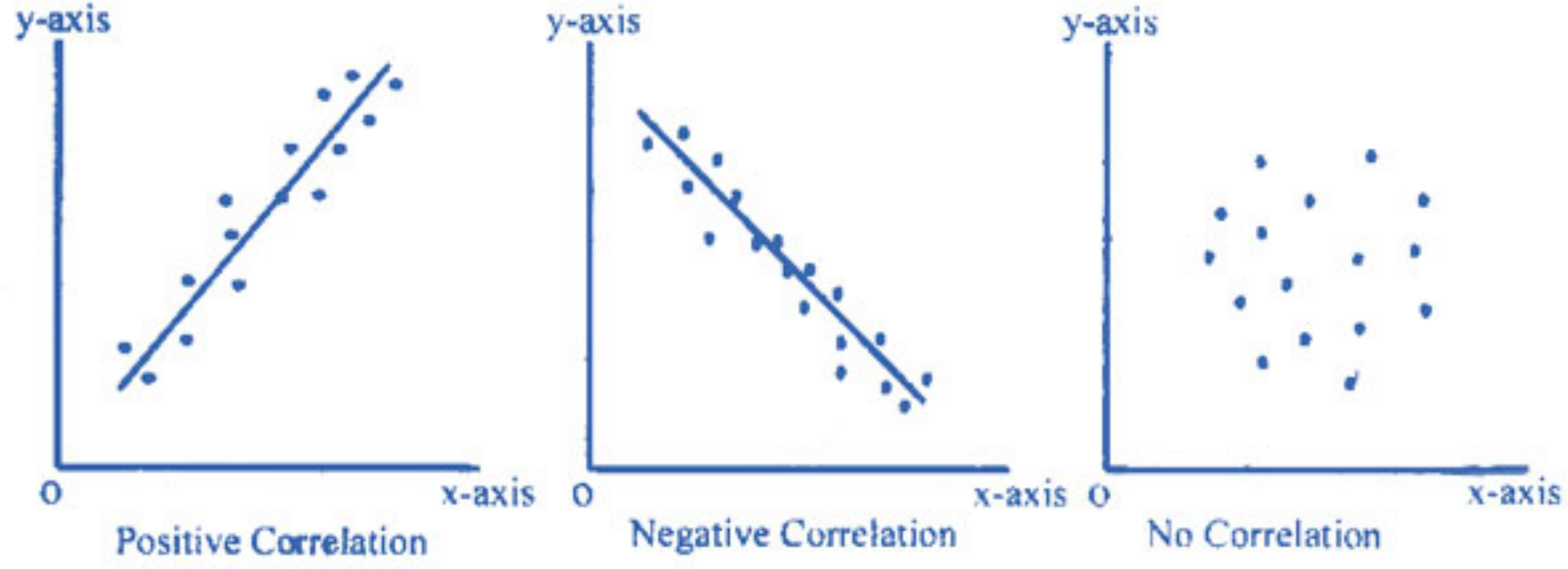


No error – exactly correlated



Some error, but still positive correlation

Introduction to Linear Regression



- Can use this line to understand how correlated data is
- For correlated data, can use this 'line' to make predictions about the values of unseen or new data