6) Functions and Tables

Vitor Kamada

December 2019

Reference

Tables, Graphics, and Figures from

Computational and Inferential Thinking: The Foundations of Data Science

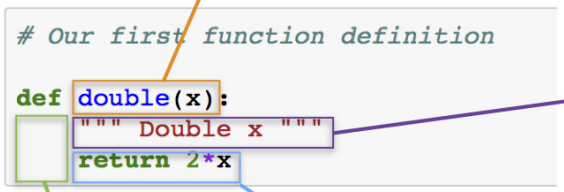
Adhikari & DeNero (2019): Ch 8 Functions and Tables

https://www.inferentialthinking.com/

Function

Signature

- Calls to the function will look like this (with the same name and number of arguments). Example: `double(3)`.
- When you call `double`, the argument can be any expression. (The name `x` doesn't affect calls.)
- In the body of the function, `x` is the name of the argument, as if the body included the code `x = <the first argument>`.



Documentation ("docstring")

- Text that describes what the function does.
- Can be any string, traditionally triple-quoted so it can span several lines.
- Traditionally, the first line describes what the function does, briefly.
- Subsequent lines can give more detail and examples.
- Running `double?` will show this text, just like `max?` will show the documentation for the built-in function `max`.

Body

- All the code in here runs each time you call the function.
- The special statement `return` tells Python what the value of each call to this function is: it's the value of the expression after `return`.
- For example, the value of `double(3)` is `6`. (Remember, when the argument is `3`, it's like the body starts with `x = 3`.)
- Often, the body will have multiple lines of code that build up to computing the `return`ed value. You can write any Python code here that you could write anywhere else.

Indentation

- Each line of code in the body is indented (that is, it's preceded by spaces).
- Traditionally, we use 2 or 4 spaces. They only need to be consistent.
- This tells Python that those lines are part of the body.
- The function's body ends at any unindented line.

Double Function

```
def double(x):
    """ Double x """
    return 2*x
```

```
double(7)
```

```
14
```

```
from datascience import *
double(make_array(3, 4, 5))
```

```
array([ 6,  8, 10])
```

```
path_data = 'https://github.com/data-8/textbook/raw/gh-pages/data/'
galton = Table.read_table(path_data + 'galton.csv')
```

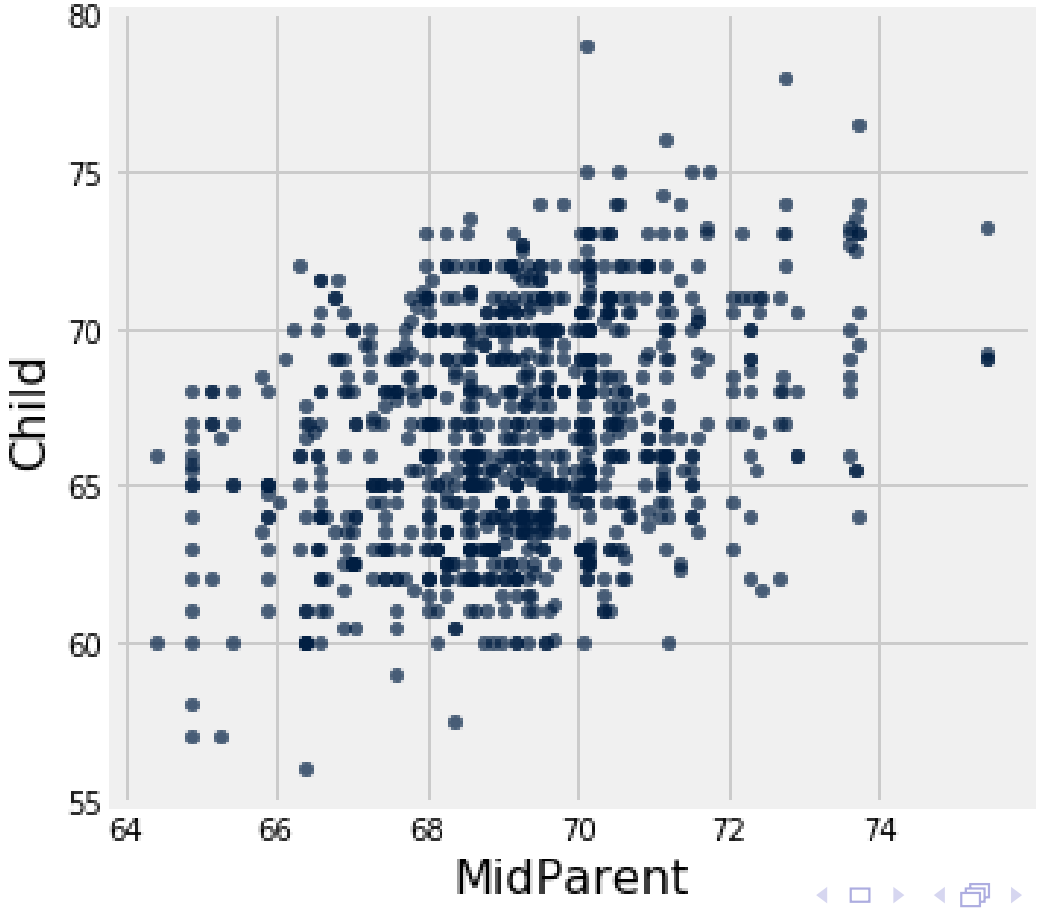
family	father	mother	midparentHeight	children	childNum	gender	childHeight
1	78.5	67	75.43	4	1	male	73.2
1	78.5	67	75.43	4	2	female	69.2

```
heights = galton.select(3, 7).relabeled(0,
        'MidParent').relabeled(1, 'Child')
```

MidParent	Child
75.43	73.2
75.43	69.2

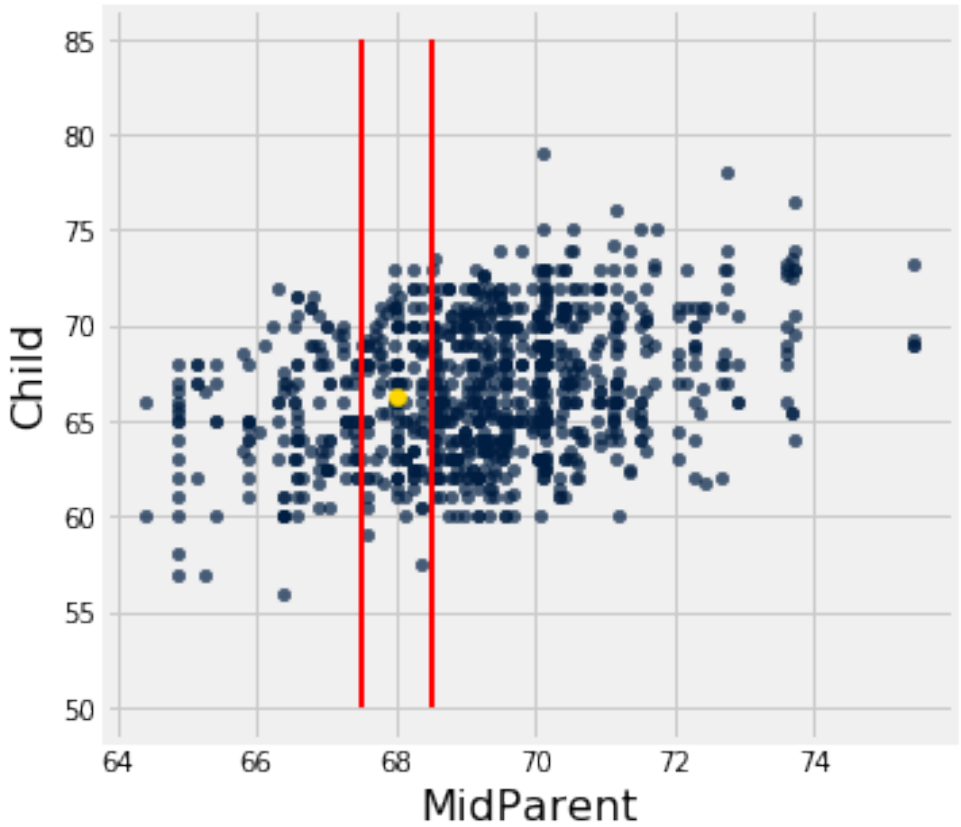
Midparent Height as Predictor Variable

```
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
%matplotlib inline
heights.scatter(0) # column for x
```



Predicting the Child's Height

```
heights.scatter('MidParent')
_ = plt.plot([67.5, 67.5], [50, 85], color='red', lw=2)
_ = plt.plot([68.5, 68.5], [50, 85], color='red', lw=2)
_ = plt.scatter(68, 66.24, color='gold', s=40)
```



```
close_to_68 = heights.where('MidParent', are.between(67.5, 68.5))
```

MidParent	Child
68.44	62
67.94	71.2
67.94	67

```
close_to_68.column('Child').mean()
```

```
66.24045801526718
```

Function predict_child

```
def predict_child(mpht):
    """Predict the height of a child whose
    parents have a midparent height of mpht.

    The prediction is the average height of
    the children whose midparent height is
    in the range mpht plus or minus 0.5.
    """
    close_points = heights.where('MidParent',
            are.between(mpht-0.5, mpht + 0.5))
    return close_points.column('Child').mean()
```

```
predict_child(68)
```

```
66.24045801526718
```

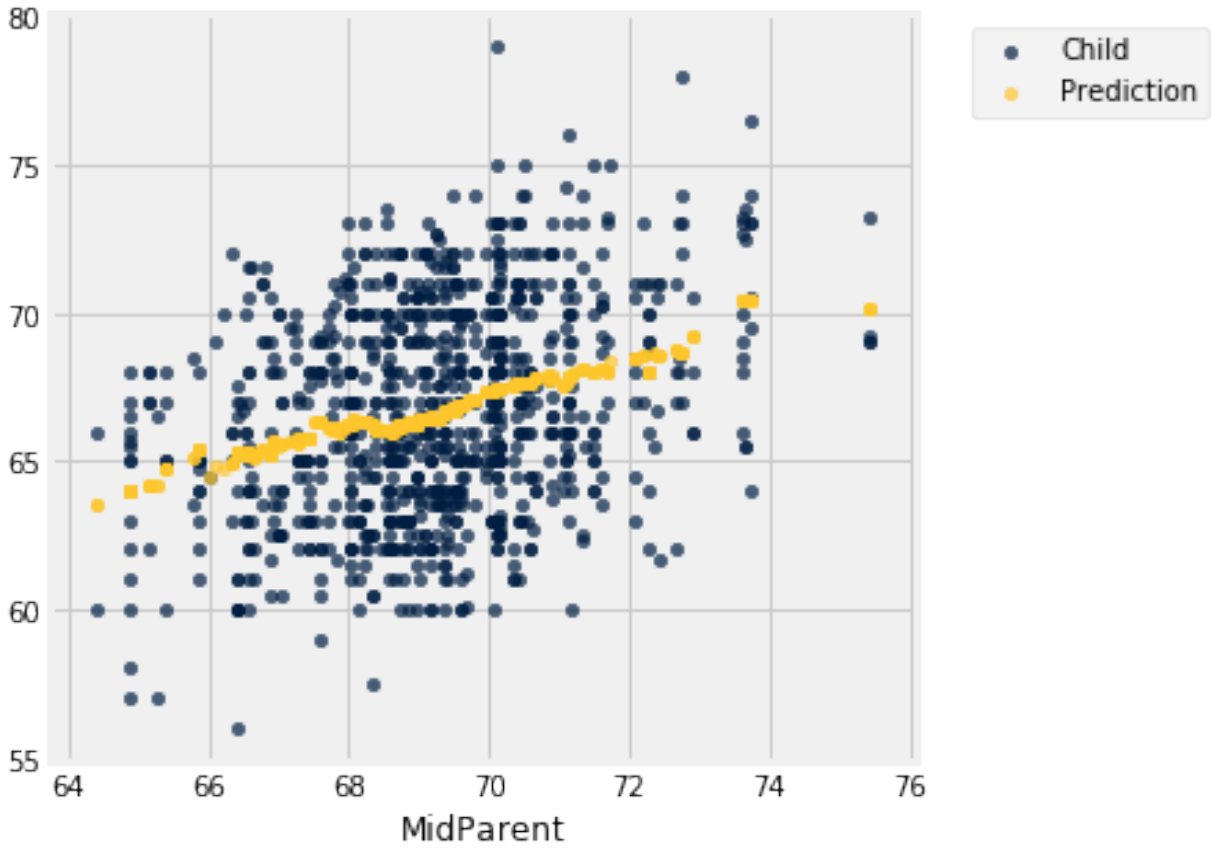
```
predict_child(74)
```

```
70.41578947368421
```

```
heights_with_predictions = heights.with_column(
    'Prediction', heights.apply(predict_child, 'MidParent')
)
```

```
heights_with_predictions.scatter('MidParent')
```

MidParent	Child	Prediction
75.43	73.2	70.1
75.43	69.2	70.1
75.43	69	70.1
75.43	69	70.1
73.66	73.5	70.4158
73.66	72.5	70.4158



US Census Current Population Survey 2008 to 2014

https://data.ca.gov/dataset/ca-educational-attainment-personal-income

```
full_table = Table.read_table(path_data + 'educ_inc.csv')
ca_2014 = full_table.where('Year',
  are.equal_to('1/1/14 0:00')).where('Age', are.not_equal_to('00 to 17'))
```

Year	Age	Gender	Educational Attainment	Personal Income	Population Count
1/1/14 0:00	18 to 64	Female	No high school diploma	H: 75,000 and over	2058
1/1/14 0:00	65 to 80+	Male	No high school diploma	H: 75,000 and over	2153
1/1/14 0:00	65 to 80+	Female	No high school diploma	G: 50,000 to 74,999	4666

Sum Up the Population Count

```
educ_inc = ca_2014.select('Educational Attainment',
                          'Personal Income', 'Population Count')
```

```
education = educ_inc.select('Educational Attainment',
                                      'Population Count')
educ_totals = education.group('Educational Attainment', sum)
```

Educational Attainment	Population Count sum
Bachelor's degree or higher	8525698
College, less than 4-yr degree	7775497
High school or equivalent	6294141
No high school diploma	4258277

```
def percents(array_x):
    return np.round( (array_x/sum(array_x))*100, 2)
```

```
import numpy as np
educ_distribution = educ_totals.with_column(
    'Population Percent', percents(educ_totals.column(1))
)
```

Educational Attainment	Population Count sum	Population Percent
Bachelor's degree or higher	8525698	31.75
College, less than 4-yr degree	7775497	28.96
High school or equivalent	6294141	23.44
No high school diploma	4258277	15.86

```
totals = educ_inc.pivot('Educational Attainment',
  'Personal Income', values='Population Count', collect=sum)
```

Personal Income	Bachelor's degree or higher	College, less than 4-yr degree
A: 0 to 4,999	575491	985011
B: 5,000 to 9,999	326020	810641
C: 10,000 to 14,999	452449	798596

```
distributions = totals.select(0).with_columns(
    "Bachelor's degree or higher", percents(totals.column(1)),
    'College, less than 4-yr degree', percents(totals.column(2)),
    'High school or equivalent', percents(totals.column(3)),
    'No high school diploma', percents(totals.column(4))
    )
```

Personal Income	Bachelor's degree or higher	College, less than 4-yr degree
A: 0 to 4,999	6.75	12.67
B: 5,000 to 9,999	3.82	10.43
C: 10,000 to 14,999	5.31	10.27

```
distributions.select(0, 1, 4).barh(0)
```