Exploring Data

CENG 499

Introduction to Data Science

Erdoğan Doğdu

Content

- Ch.10 Working with Data

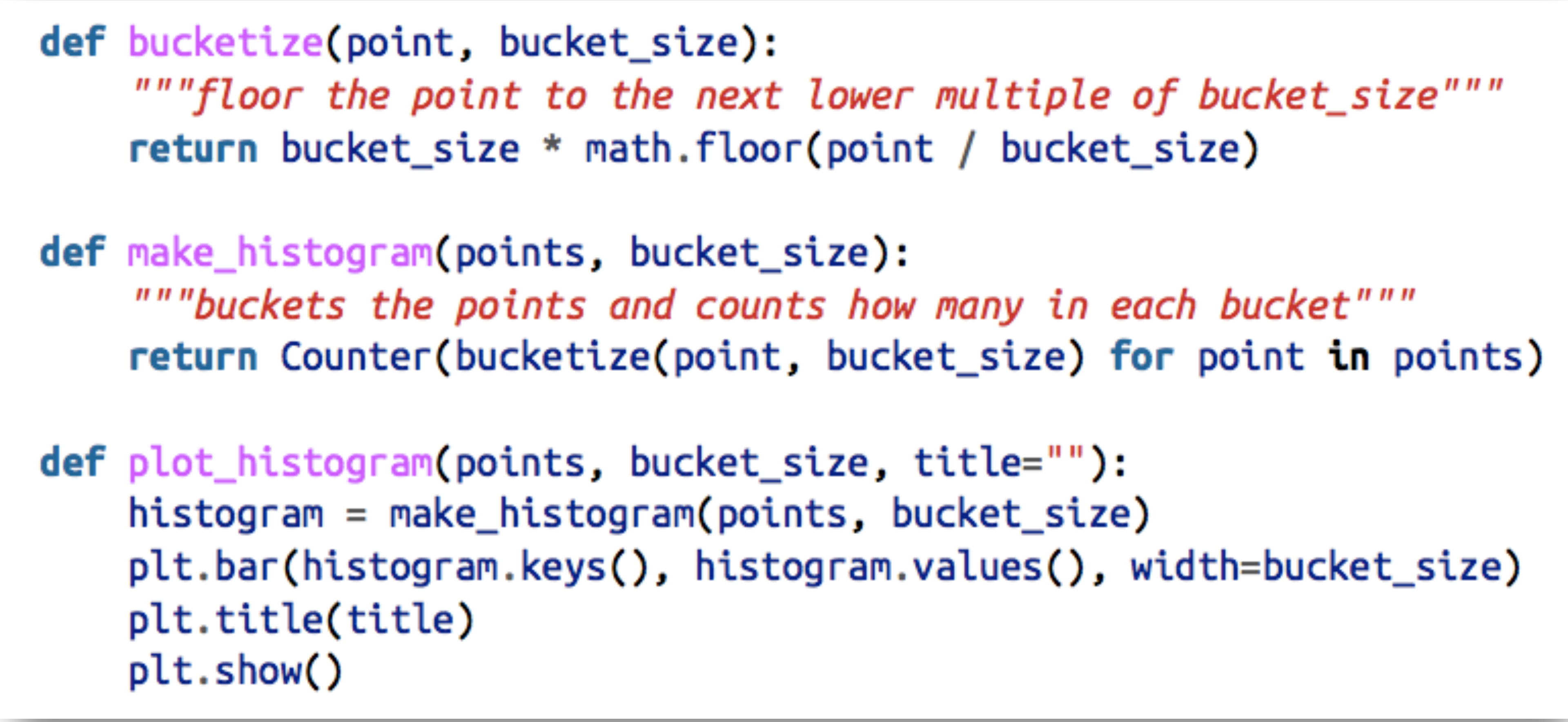
Exploring Data

- Before you start building models and predicting, know your data
  - Explore your data first

One-dimensional data

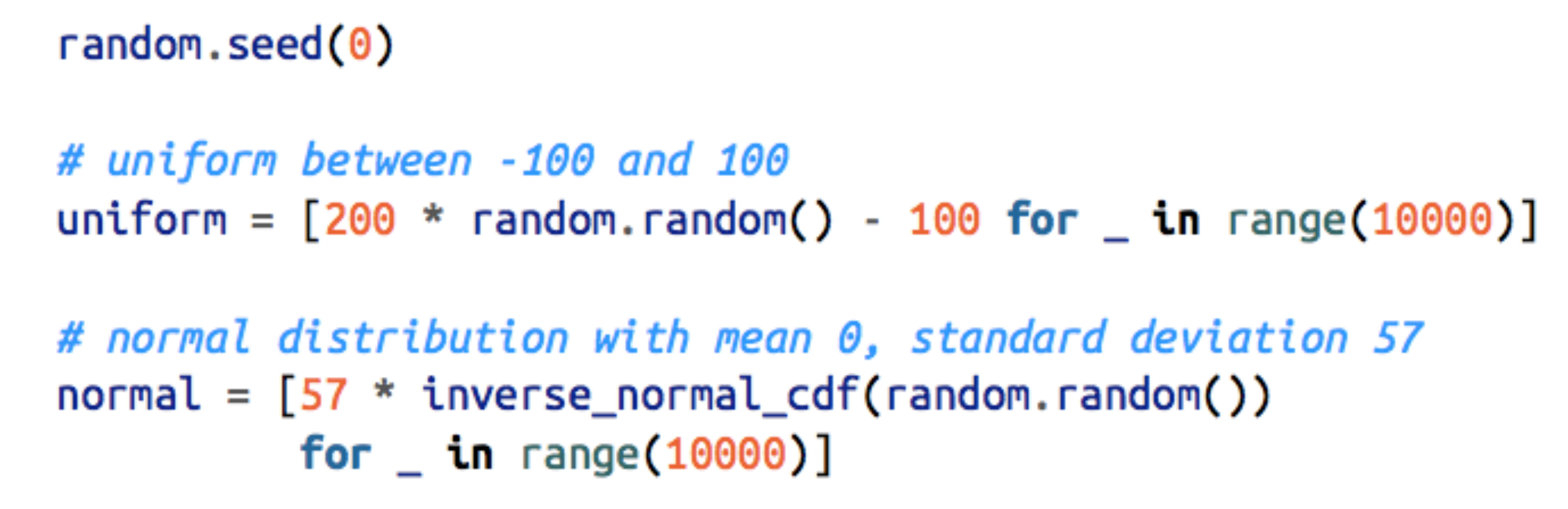
- Example. A collection of numbers
  - The number of minutes each user spend on your web site
- How to explore?
  - Summary statistics
    - # of items, the smallest, the largest, the mean, std.dev
  - Histograms
    - Group data into *buckets*

Histograms



```
def bucketize(point, bucket_size):
    """floor the point to the next lower multiple of bucket_size"""
    return bucket_size * math.floor(point / bucket_size)

def make_histogram(points, bucket_size):
    """buckets the points and counts how many in each bucket"""
    return Counter(bucketize(point, bucket_size) for point in points)

def plot_histogram(points, bucket_size, title=""):
    histogram = make_histogram(points, bucket_size)
    plt.bar(histogram.keys(), histogram.values(), width=bucket_size)
    plt.title(title)
    plt.show()
```

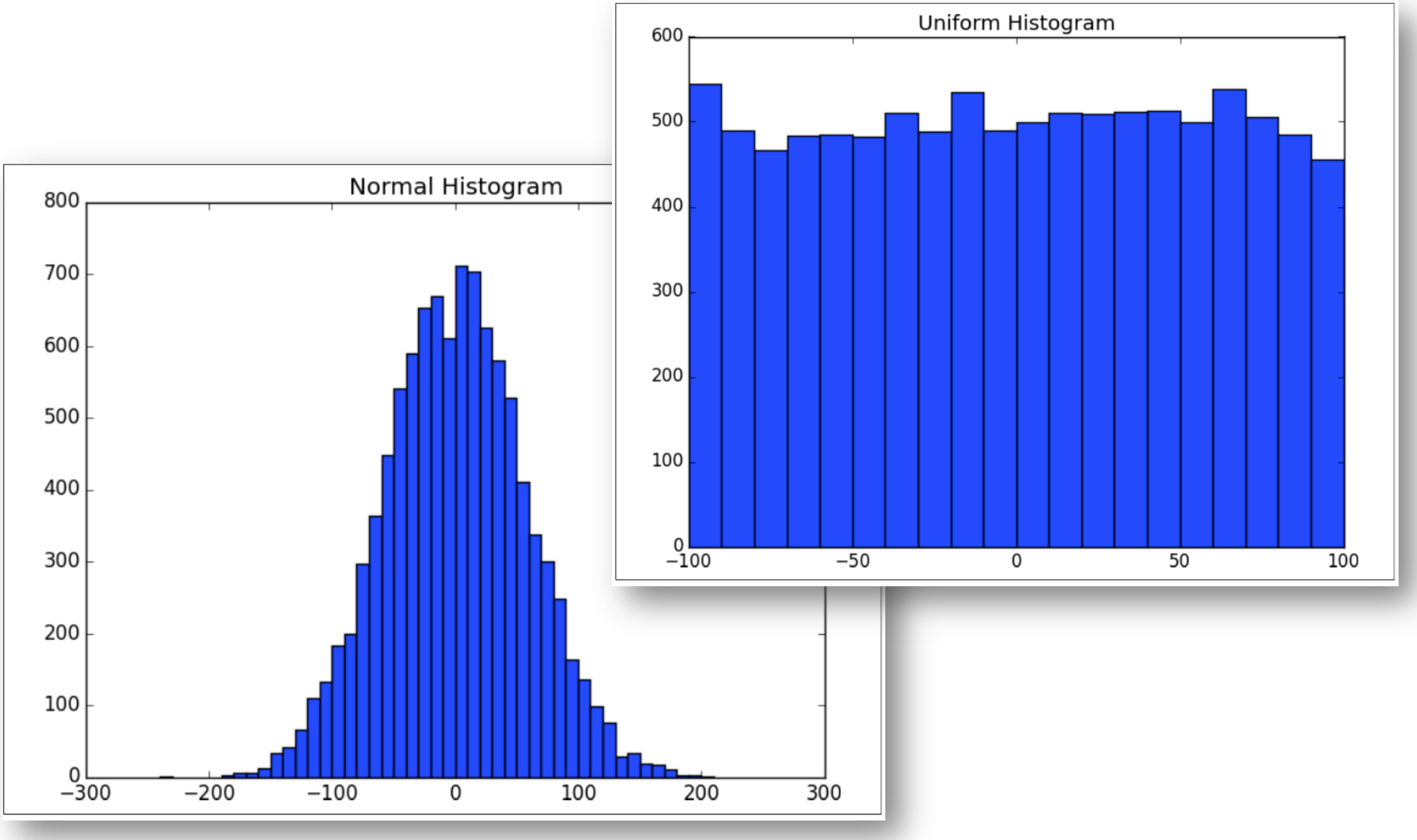
Histograms



```
random.seed(0)

# uniform between -100 and 100
uniform = [200 * random.random() - 100 for _ in range(10000)]

# normal distribution with mean 0, standard deviation 57
normal = [57 * inverse_normal_cdf(random.random())
          for _ in range(10000)]
```

- Mean: 0, Std.dev = 58 for both distributions
- Distribution?
  - – plot_histogram(uniform, 10, "Uniform Histogram")
  - – plot_histogram(normal, 10, "Normal Histogram”)

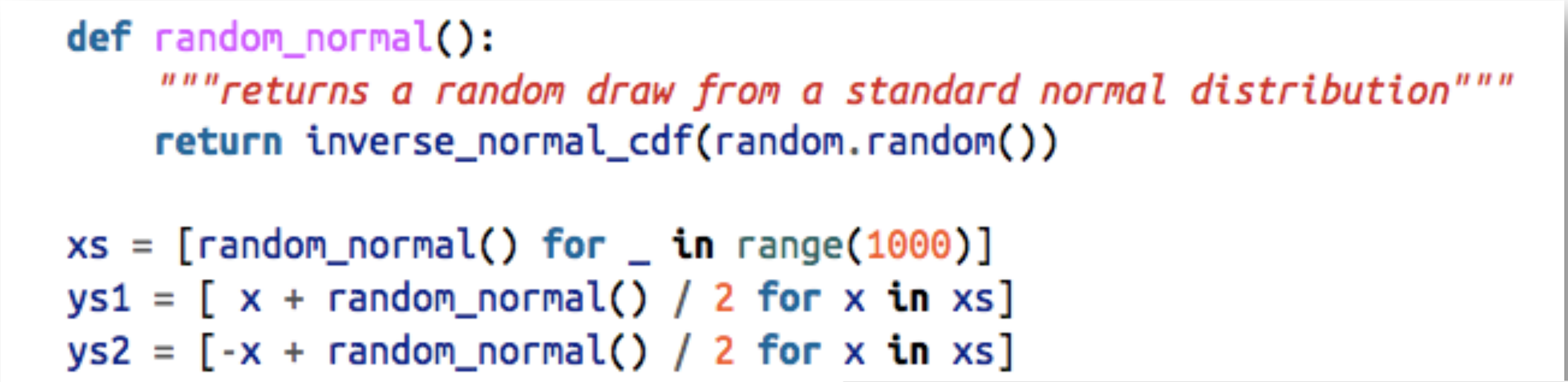
Histograms


Normal Histogram
800
700
600
500
400
300
200
100
0
−300
−200
−100
0
100
200
300
Uniform Histogram
600
500
400
300
200
100
0
−100
−50
0
50
100

Two dimensions

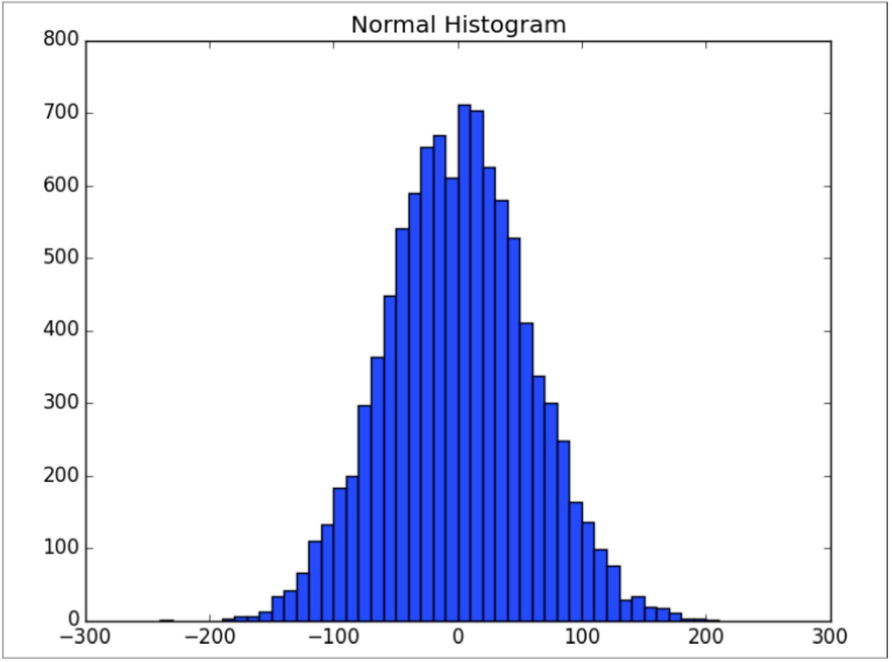
- Example:
  - Users’ daily minutes in the web site (dim1)
  - Users’ experience in years in data science (dim2)
  - How do they vary together?

Two dimensions



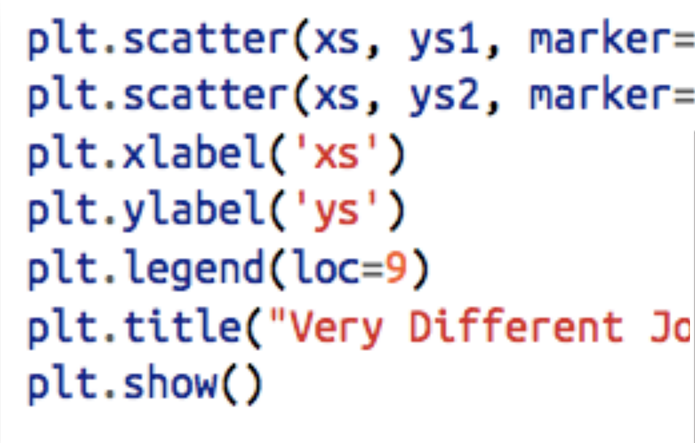
```
def random_normal():
    """returns a random draw from a standard normal distribution"""
    return inverse_normal_cdf(random.random())

xs = [random_normal() for _ in range(1000)]
ys1 = [ x + random_normal() / 2 for x in xs]
ys2 = [-x + random_normal() / 2 for x in xs]
```

- plot_histogram(ys1, 10, ”ys1")
- plot_histogram(ys2, 10, ”ys2")
- Same mean, std.dev, normally distributed



Two dimensions

- But each has a very different joint distribution with xs



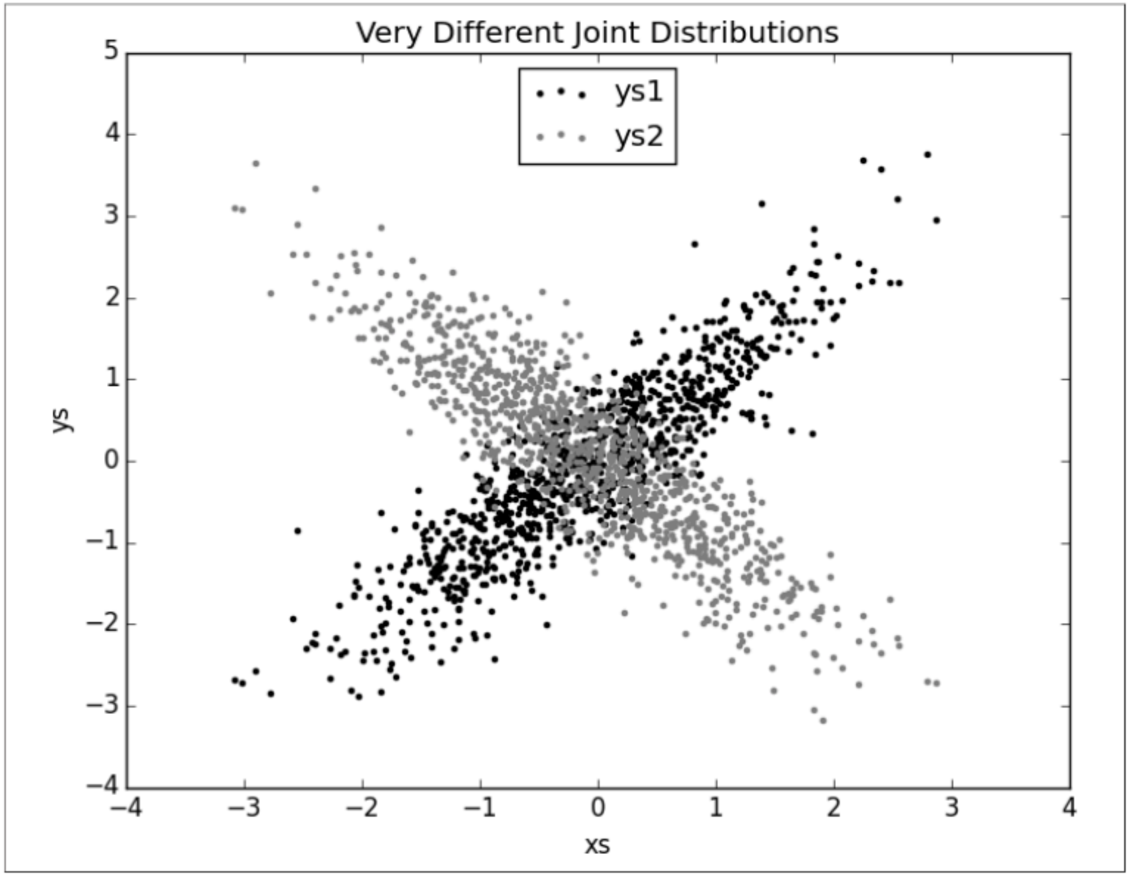
```
plt.scatter(xs, ys1, marker='.', color='black', label='ys1')
plt.scatter(xs, ys2, marker='.', color='gray',  label='ys2')
plt.xlabel('xs')
plt.ylabel('ys')
plt.legend(loc=9)
plt.title("Very Different Jo
plt.show()
```

print correlation(xs, ys1)

# 0.9

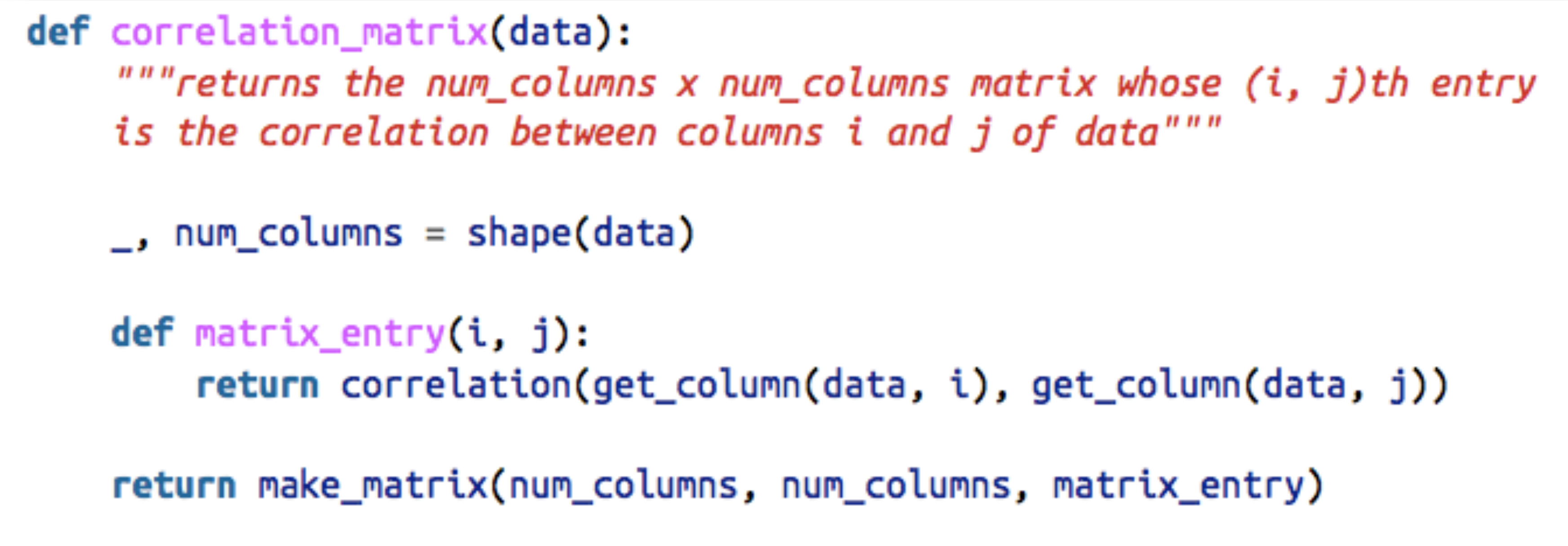
print correlation(xs, ys2)

# -0.9



Many Dimensions

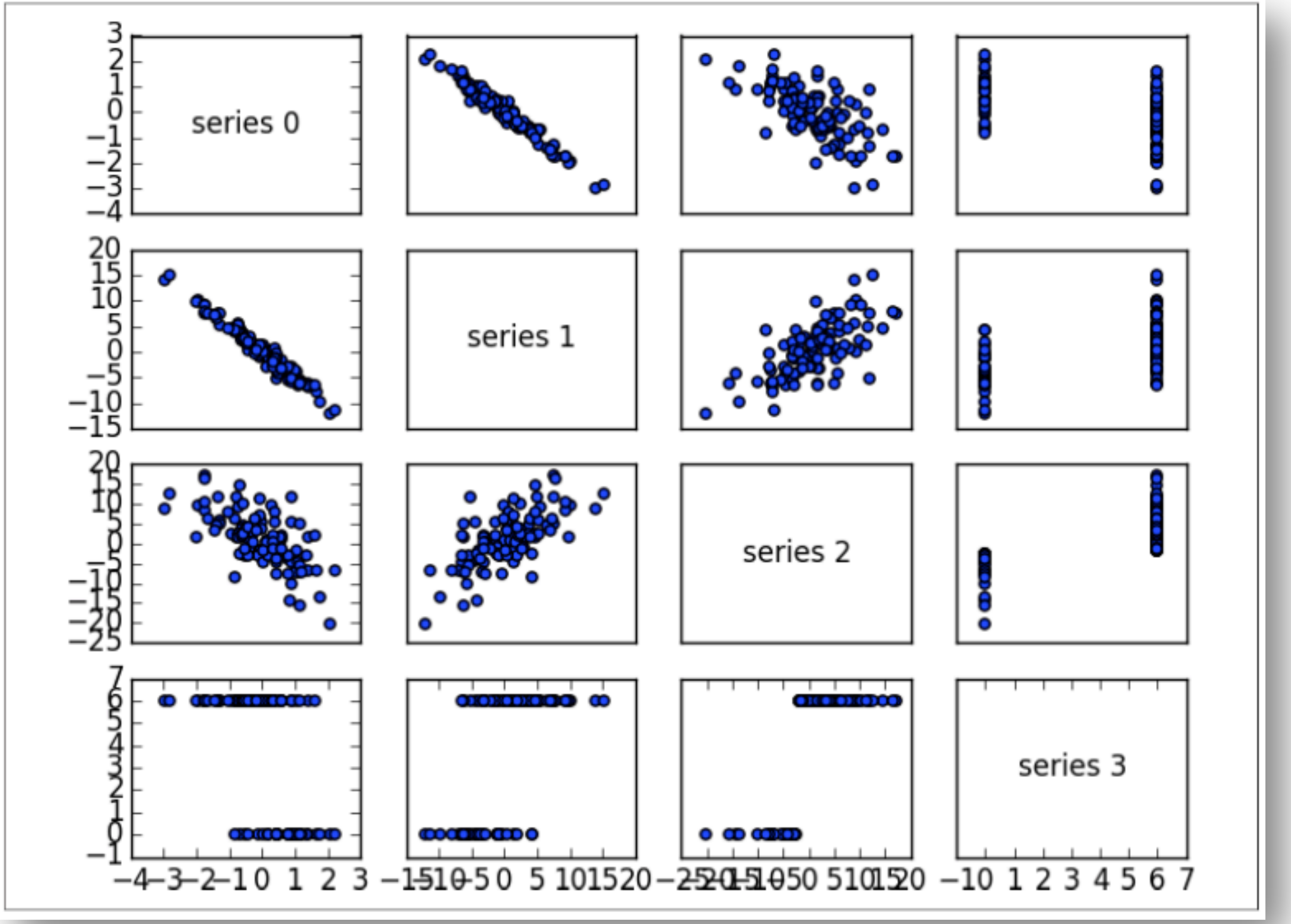
- How do all the dimensions relate to one another?
- ***Correlation matrix***
  - Row *i*, Col *j*: Correlation of dim *i* and dim *j*



```
def correlation_matrix(data):
    """returns the num_columns x num_columns matrix whose (i, j)th entry
    is the correlation between columns i and j of data"""

    _, num_columns = shape(data)

    def matrix_entry(i, j):
        return correlation(get_column(data, i), get_column(data, j))

    return make_matrix(num_columns, num_columns, matrix_entry)
```

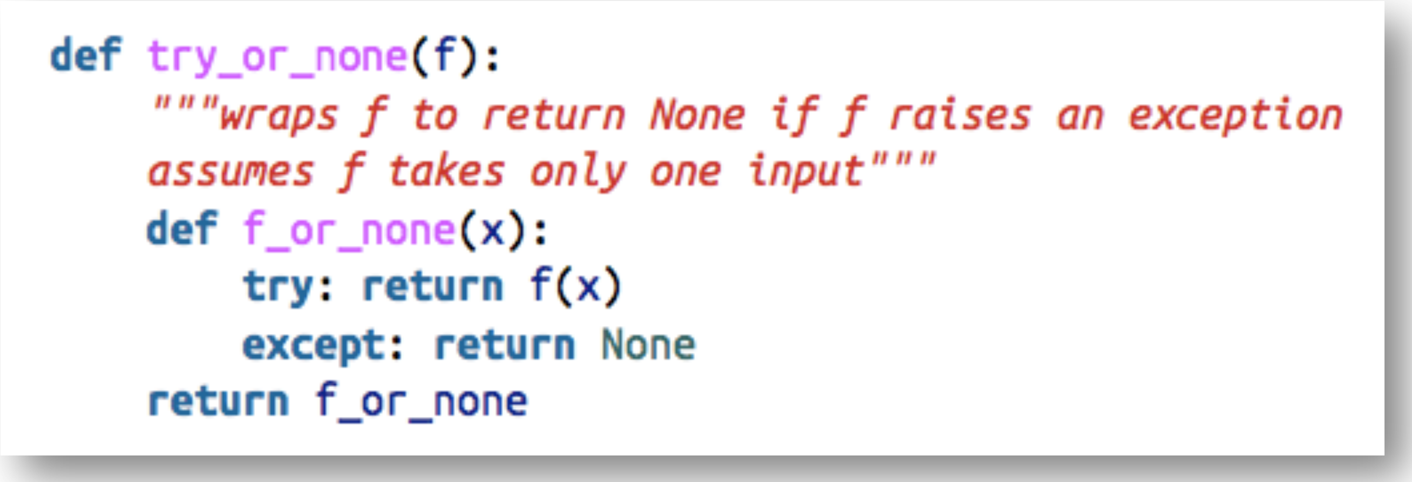
Many Dimensions

- Scatter plot matrix **plt.subplots()**



Cleaning and Munging

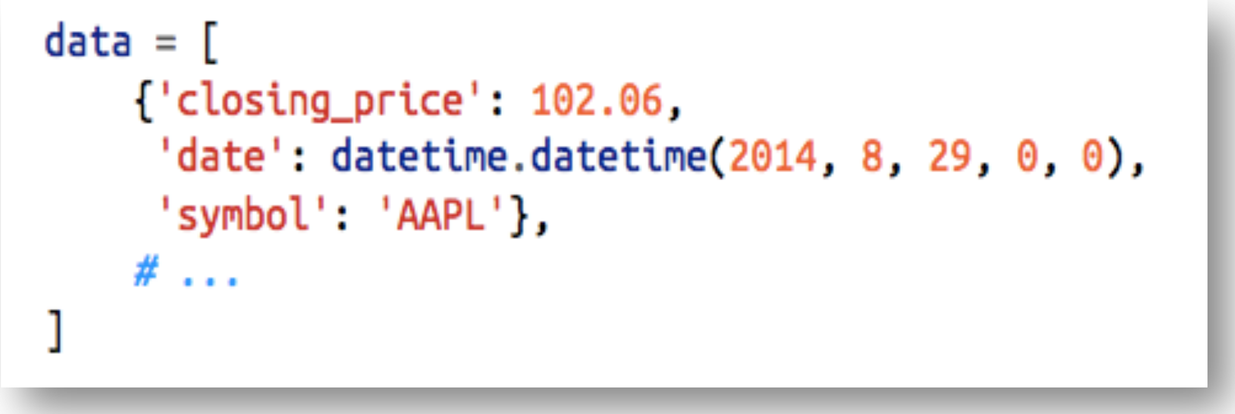
- Real world data is dirty
- Convert string to numbers (ex. float[str])
- If cannot convert?



```
def try_or_none(f):
    """wraps f to return None if f raises an exception
    assumes f takes only one input"""
    def f_or_none(x):
        try: return f(x)
        except: return None
    return f_or_none
```

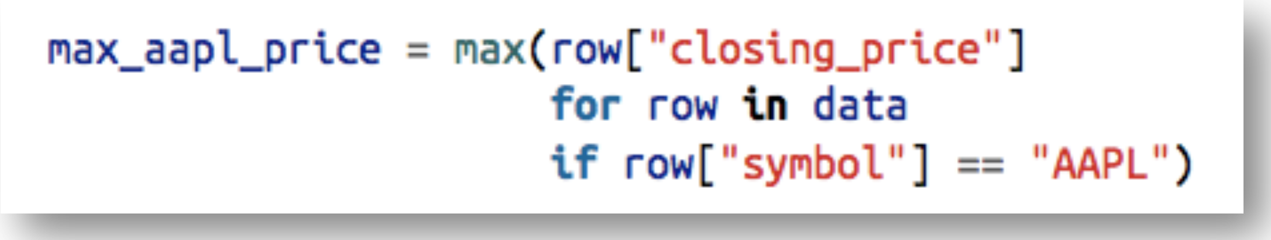
Manipulating Data

- Stock prices data



```
data = [
    {'closing_price': 102.06,
     'date': datetime.datetime(2014, 8, 29, 0, 0),
     'symbol': 'AAPL'},
    # ...
]
```

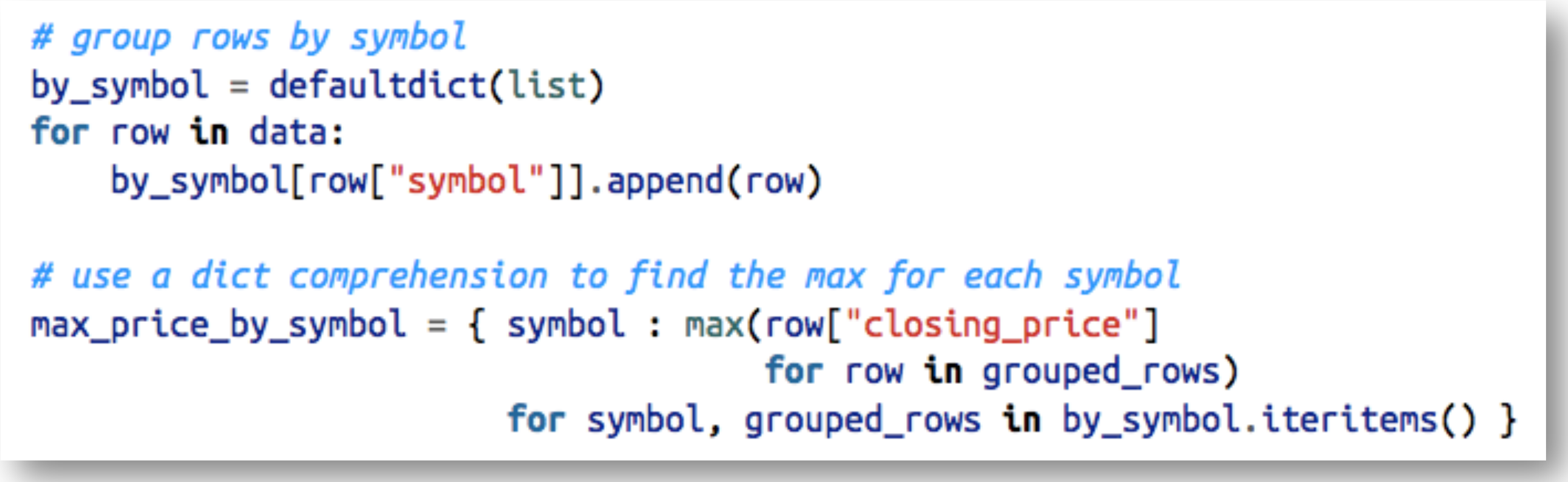
- The highest-ever closing price for **AAPL**?
  - Restrict ourselves to AAPL rows.
  - Grab the closing_price from each row.
  - Take the max of those prices.



```
max_aapl_price = max(row["closing_price"]
                     for row in data
                     if row["symbol"] == "AAPL")
```

Manipulating Data

- The highest-ever closing price **for each stock** in our data set?



```
# group rows by symbol
by_symbol = defaultdict(list)
for row in data:
    by_symbol[row["symbol"]].append(row)

# use a dict comprehension to find the max for each symbol
max_price_by_symbol = { symbol : max(row["closing_price"]
                                     for row in grouped_rows)
                        for symbol, grouped_rows in by_symbol.iteritems() }
```

Rescaling

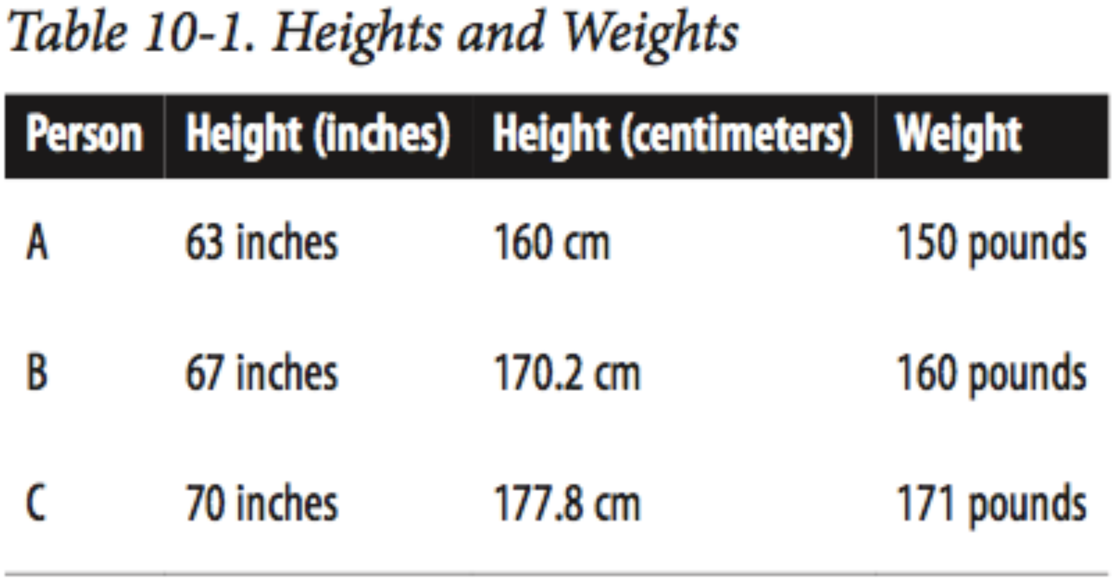
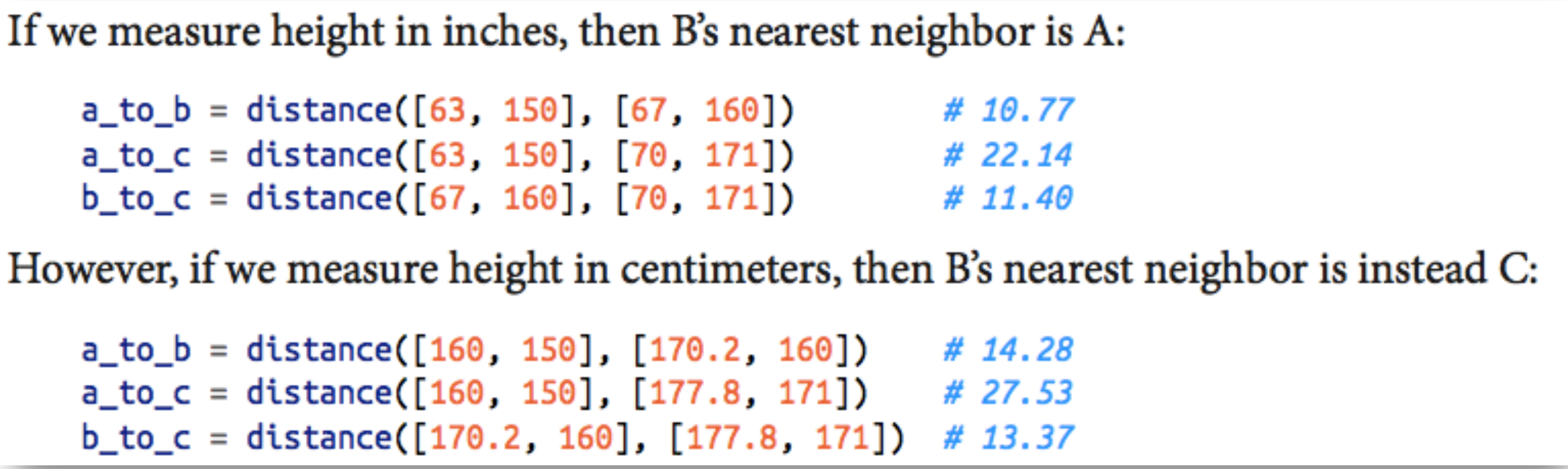


Table 10-1. Heights and Weights

Person	Height (inches)	Height (centimeters)	Weight
A	63 inches	160 cm	150 pounds
B	67 inches	170.2 cm	160 pounds
C	70 inches	177.8 cm	171 pounds

- Cluster body sizes?
  - – Euclidian distance between (height,weight) pairs

Rescaling



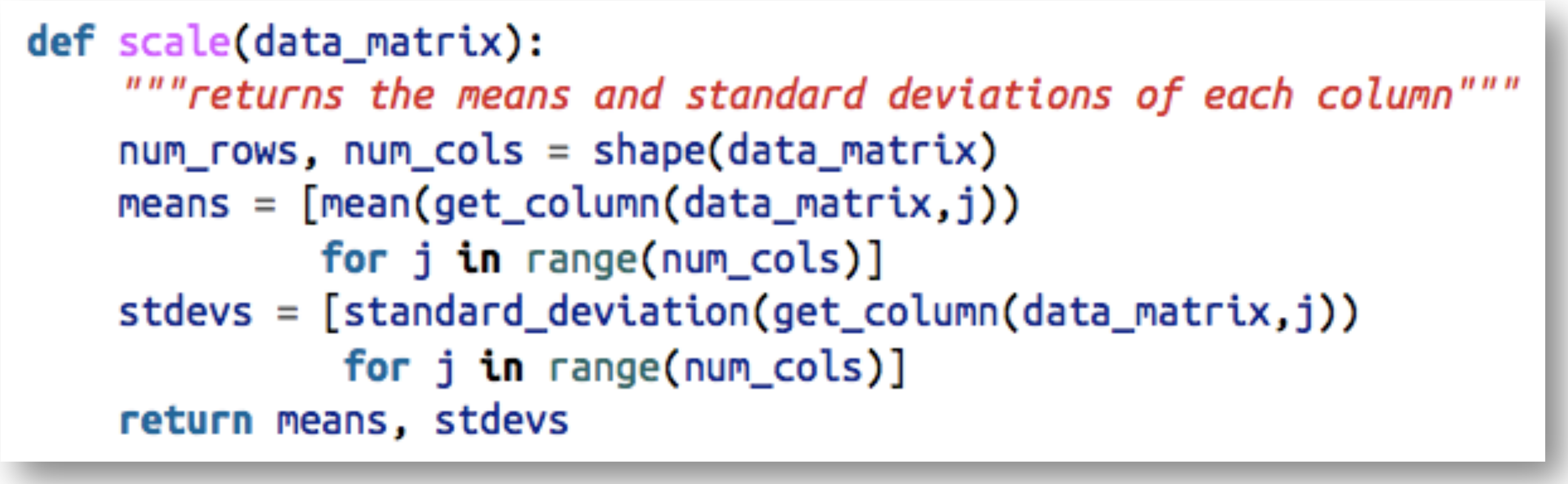
If we measure height in inches, then B's nearest neighbor is A:

```
a_to_b = distance([63, 150], [67, 160])        # 10.77
a_to_c = distance([63, 150], [70, 171])        # 22.14
b_to_c = distance([67, 160], [70, 171])        # 11.40
```

However, if we measure height in centimeters, then B's nearest neighbor is instead C:

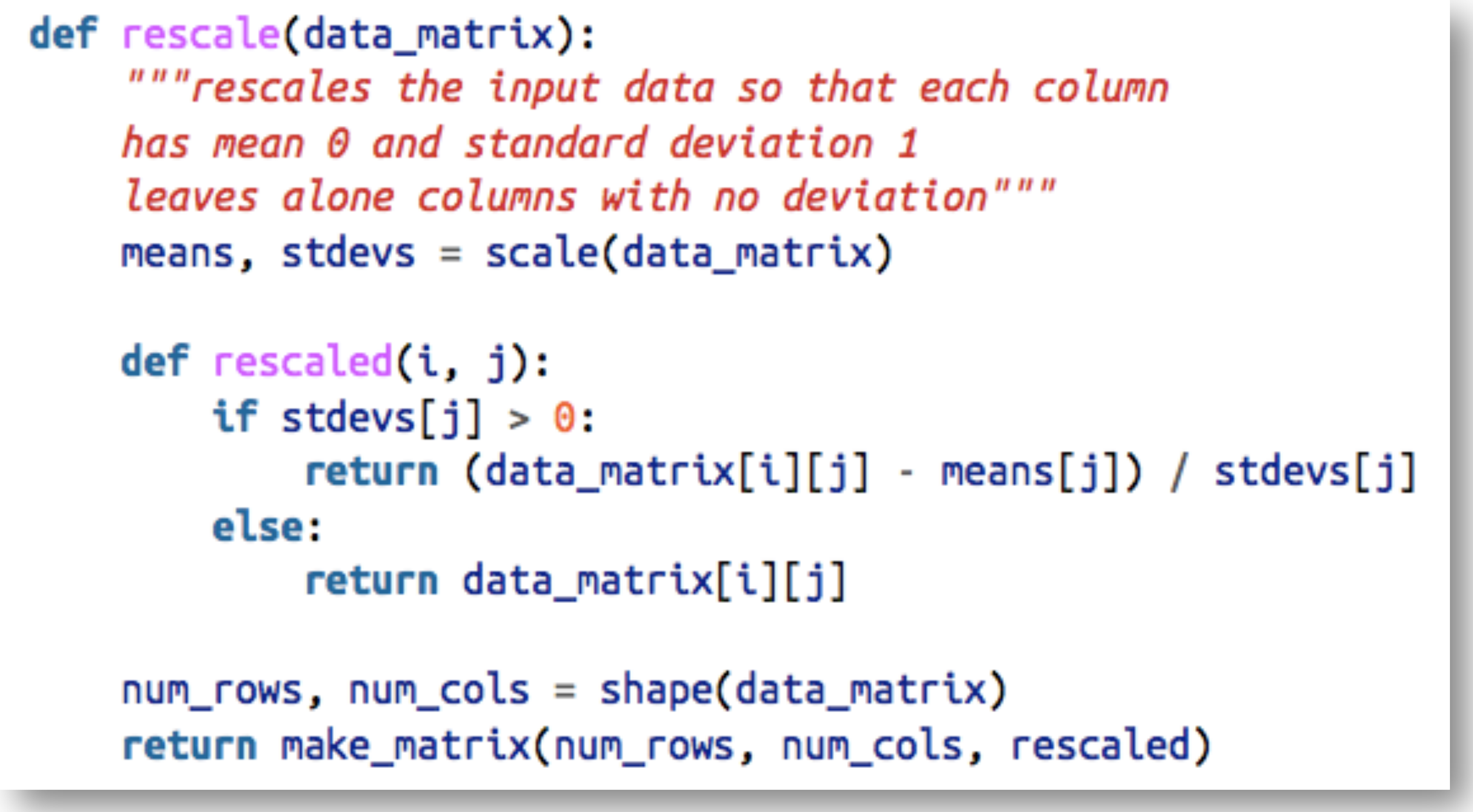
```
a_to_b = distance([160, 150], [170.2, 160])    # 14.28
a_to_c = distance([160, 150], [177.8, 171])    # 27.53
b_to_c = distance([170.2, 160], [177.8, 171])  # 13.37
```

Rescaling



```
def scale(data_matrix):
    """returns the means and standard deviations of each column"""
    num_rows, num_cols = shape(data_matrix)
    means = [mean(get_column(data_matrix,j))
             for j in range(num_cols)]
    stdevs = [standard_deviation(get_column(data_matrix,j))
              for j in range(num_cols)]
    return means, stdevs
```

Rescaling



```
def rescale(data_matrix):
    """rescales the input data so that each column
    has mean 0 and standard deviation 1
    leaves alone columns with no deviation"""
    means, stdevs = scale(data_matrix)

    def rescaled(i, j):
        if stdevs[j] > 0:
            return (data_matrix[i][j] - means[j]) / stdevs[j]
        else:
            return data_matrix[i][j]

    num_rows, num_cols = shape(data_matrix)
    return make_matrix(num_rows, num_cols, rescaled)
```