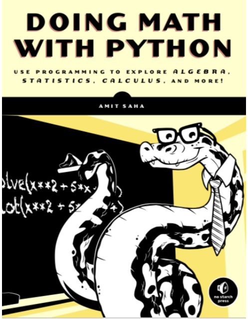
Discrete Structures: CMPSC 102

Oliver BONHAM-CARTER

Fall 2019
Week 12

Where We Are?



Saha, Chapter 3: Describing Data with Statistics

- Basic statistics: Mean, Median, Mode, Frequencies, Correlations, etc.
- Common Elements, Minimum & Maximum values, and Range

Mean

Finding the average

- The mean of the set $\{11, 12, 13\}$
  - $(11 + 12 + 13)/3 = 12$
- Could also use a `list` and the `sum()` function

Find the mean

```
num_list = [11,12,13]
sum(num_list) / len(num_list)
```

Mean

file: mean.py

Function for the mean

```
def calculate_mean(numbers_list):
  print("  Values", numbers_list)
  s_int = sum(numbers_list)
  N_int = len(numbers_list)
  # Calculate the mean
  mean_flt = s_int/N_int
  return mean_flt
#end of calculate_mean()
if __name__ == '__main__':
    donations_list = [100, 60, 70, 900, 100,
200, 500, 500, 503, 600, 1000, 1200]
    mean_flt = calculate_mean(donations_list)
    N_int = len(donations_list)
    print('  The mean of the {0} values
is {1}'.format(N_int, mean_flt))
```

Median

1, 3, 3, **6**, 7, 8, 9

Median $= \underline{\underline{6}}$

1, 2, 3, **4**, **5**, 6, 8, 9

Median $= (4 + 5) \div 2$

$= \underline{\underline{4.5}}$

- The median is the value separating the higher half from the lower half of a data sample.

Median

Median

First, arrange the observations in an ascending order.

If the number of observations (n) is odd: the median is the value at position

$$\left(\frac{n+1}{2}\right)$$

If the number of observations (n) is even:

1. Find the value at position $\left(\frac{n}{2}\right)$
2. Find the value at position $\left(\frac{n+1}{2}\right)$
3. Find the average of the two values to get the median.

Function for the Median

```
''' Calculating the median '''
def calculate_median(numbers_list):
#    print(" calculate_mean()")
    N = len(numbers_list)
    numbers_list.sort()
    # Find the median
    if N % 2 == 0:
        # if N is even
        m1 = N/2
        m2 = (N/2) + 1
        # Convert to integer, match position
        m1 = int(m1) - 1
        m2 = int(m2) - 1
        median_int = (numbers_list[m1] + numbers_list[m2])/2
    else:
        m = (N+1)/2
        # Convert to integer, match position
        m = int(m) - 1
        median_int = numbers_list[m]
    return median_int

if __name__ == '__main__':
       donations_list = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
       print("  Data:",donations_list)
       median_int = calculate_median(donations_list)
       N = len(donations_list)
       print('  Median donation over the last {0}
days is {1}'.format(len(donations_list), median_int))
```

Median

Simple Example

```
import statistics
statistics.median([1,2,3])
```

Another Quick Example with Random Data

```
import random, statistics
nums_list = []
for i in range(10):
  n = int(random.random() * 9 + 1)
  nums_list.append(n)
statistics.median(nums_list)
```

What is the Most Common Element?

What entry in the set is the most common?

```
simplelist = [4, 2, 1, 3, 4]
from collections import Counter
c = Counter(simplelist)
c.most_common() #[(4, 2), (1, 1), (2, 1), (3, 1)]
```

What entry in the set is the most common?

```
c = Counter(['a','a','a','a','a','a','a','b'])
c.most_common() #[('a', 7), ('b', 1)]
```

- Contained in the output is the number of times that an element has been found.

Mode

file: mode.py

Function for the Mode

```
'''Calculating most commonly observed value'''
from collections import Counter
def calculate_mode(numbers_list):
    print("  Values: ",numbers_list)
    c = Counter(numbers_list)
    mode_int = c.most_common(1) #print first most common
    return mode_int[0][0]
#end of calculate_mode()
if __name__=='__main__':
    scores_list = [7, 8, 9, 2, 10, 9, 9, 9, 9, 4,
5, 6, 1, 5, 6, 7, 8, 6, 1, 10]
    print("  Set: ",scores_list)
    mode_int = calculate_mode(scores_list)
    print("  Mode: ",mode_int)
```

- The most common (most frequently occurring) data point from discrete or nominal data.

Sorry about the tiny print!

Most Common Values in a List

file: colCounter.py

- Print the number of times an Integer has occurred in list

```
''' Print the number of times an Integer has occurred in list'''

from collections import Counter
scores_list = [7, 8, 9, 2, 10, 9,1,1,0]
x_colCount = Counter(scores_list)
print("type(x_colCount)",type(x_colCount)) # <class 'collections.
print("  Data: ",scores_list)
print(" + One way to collect counts:\n")
print("  Value\tCount")
for i in x_colCount:
print("  ",i,"\t",x_colCount[i])
print("\n + Another way to collect counts:\n")
for i in x_colCount.most_common():
print("  ",i)
```

Most Common Values in a List

file: colCounter_char.py

- Print the number of times a **Character** has occurred in list

```
''' Print the number of times a Character has occurred in l

from collections import Counter
scores_list = ['a','b','a','a','b','c']
print("  Data: ",scores_list)
x_colCount = Counter(scores_list)
type(x_colCount) # <class 'collections.Counter'>
print(" + One way to do it:\n")
print("  Value\tCount")
for i in x_colCount:
print("  ",i,"\t",x_colCount[i])
print("\n + Another way to do it:\n")
for i in x_colCount.most_common():
print("  ",i)
```

Dispersion

- *Dispersion*: a measurement of distance between its values and the mean of the data set.
- Three measurements of dispersion: range, variance, and standard deviation
- After finding the mean, one may want to know how *spread-out* the values are using the *Variance*.

What kind of distribution?

- The mean of 50 can come from two different distributions
  - $50 = (49 + 50 + 51)/3$
  - $50 = (82 + 23 + 45)/3$

- The **Range** is the maximum and minimum values of a data set.

Range
file: range.py

Basic Stats
Common Elements
Mode
Range
Variance
Standard Deviation
Correlation
Application

Function for the Range

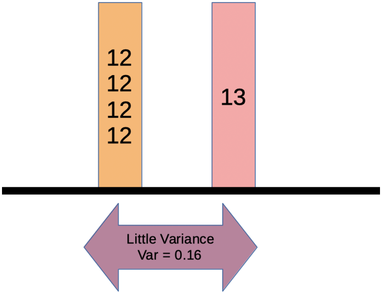
```
''' Finding the range '''
def find_range(numbers_list):
  print("  Values: ",numbers_list)
  lowest_int = min(numbers_list)
  highest_int = max(numbers_list)
  # Find the range
  r_int = highest_int - lowest_int # find distance
  return lowest_int, highest_int, r_int
#end of find_range()

if __name__ == '__main__':
  donations_list = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
  lowest, highest, r = find_range(donations_list)
print('  Lowest: {0} Highest: {1} Range: {2}'.format(lowest, highest, r))
```

- The most common (most frequently occurring) data point from discrete or nominal data.

Sorry about the tiny print!

Little Variance

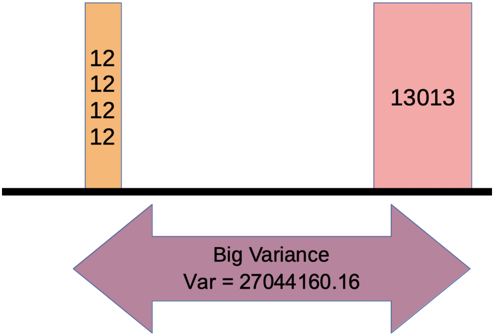
The spread of points from the mean



- The data set $\{12, 12, 12, 12, 12\}$ has a var. of zero (the numbers are identical).
- The data set $\{12, 12, 12, 12, 13\}$ has a var. of 0.16; a small change in the numbers equals a very small var

Big Variance

The spread of points from the mean



- The data set $\{12, 12, 12, 12, 13013\}$ has a var. of 27044160.16; a large distance between the values

Calculating Variance

$$\sigma^2 = \sum_{i=0}^{n} \frac{(x_i - \mu)^2}{n}$$

i	x_i	μ	$(x-\mu)$	$(x-\mu)^2$
0	17		3	9
1	15		1	1
2	23		9	81
3	7		-7	49
4	9		-5	25
5	13		-1	1
Σ	84	14		166

- $\frac{166}{6} = 27.66$ (Regular variance)
- $\frac{166}{6-1} = 33.2$ (Dividing by $n-1$, instead of n, gives you a better estimate of variance of a larger population)

Variance Code 1

See source code: `file: variance.py`


```
''' Find the variance and standard deviation of a list of numbers'''
def calculate_mean(numbers):
    s = sum(numbers)
    N = len(numbers)
    # Calculate the mean
    mean = s/N
    return mean
#end of calculate_mean()

def find_differences(numbers_list):
    # Find the mean
    mean = calculate_mean(numbers_list)
    # Find the differences from the mean
    diff_list = []
    for num in numbers_list:
       diff_list.append(num-mean)
    return diff_list
#end of find_differences()
```

Variance Code 2

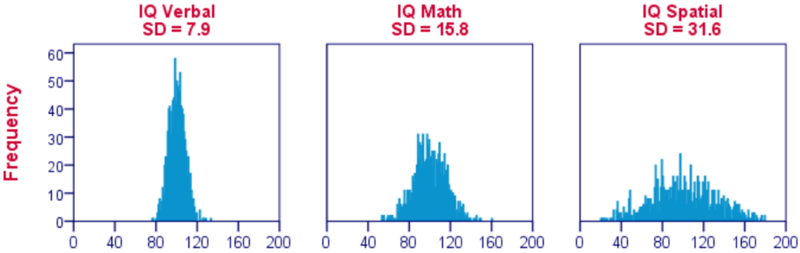
See source code: `file: variance.py`


```
def calculate_variance(numbers):
        # Find the list of differences
        diff_list = find_differences(numbers)
        # Find the squared differences
        squared_diff_list = []
        for d in diff_list:
            squared_diff_list.append(d**2)
        # Find the variance
        sum_squared_diff_list = sum(squared_diff_list)
        # better estimate for large populations
        variance = sum_squared_diff_list/(len(numbers))
        return variance
#end of calculate_variance()

if __name__ == '__main__':
        donations_list = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
        variance = calculate_variance(donations_list)
        print("  Data:",donations_list)
        print('The variance of the list of numbers is {0}'.format(variance))
        std = variance**0.5 # sqrt of variance
        print('The standard deviation of the list of numbers is {0}'.format(std))
```

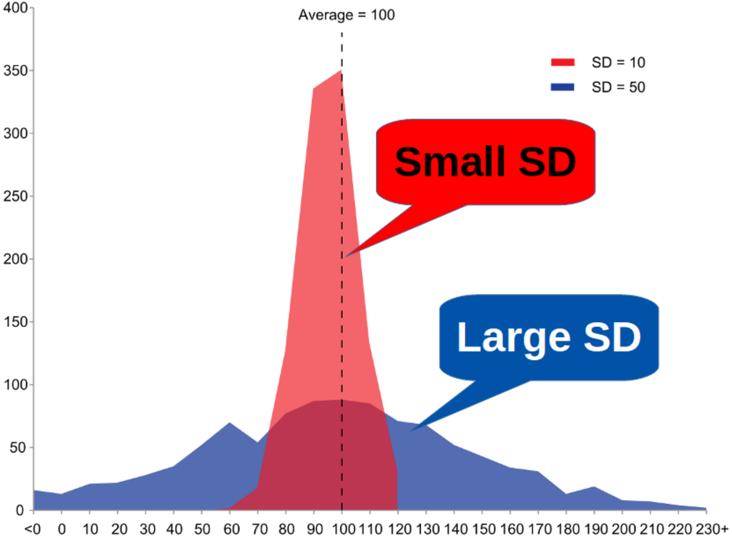
```
Data: [100, 60, 70, 900, 100,
200, 500, 500, 503, 600, 1000, 1200]
The variance of the list of numbers is 141047.35
The standard deviation of the list of numbers is 375.56
```

Standard Deviation

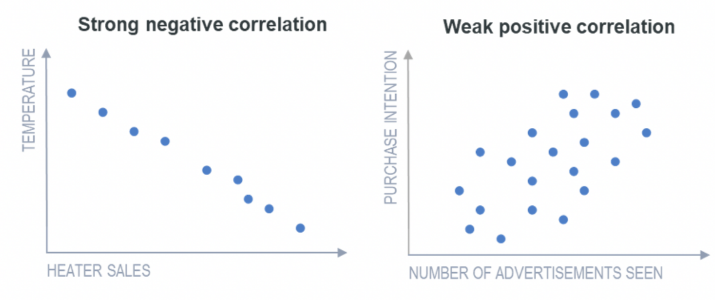
- σ (Sigma)
- The variance is the square of the St. Dev
- A measurement of variation or dispersion of a set of values
- **Low St. Dev values**: indicate values of set are situated close to the mean (the *expected value*) of a set
- **High St. Dev values**: indicate values of set spread out over a wider range.



Standard Deviation



Types of Correlation



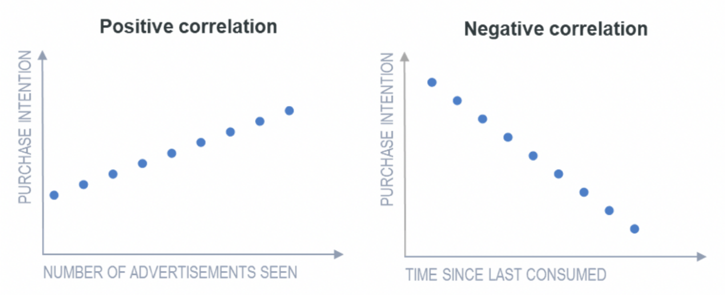
- **A strong correlation**: One variable based on the values of the other. (A scoring near 1.0 or -1.0)
- **A weak correlation**: The average of one variable are related to the other. (A score not equal to zero)
- There are many exceptions

Types of Correlation

By the numbers...

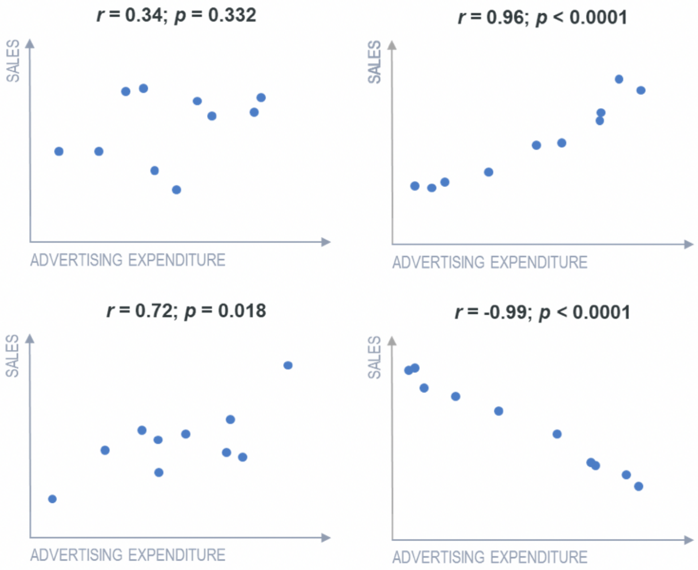
- A correlation of 1 indicates a perfect positive correlation.
- A correlation of -1 indicates a perfect negative correlation.
- A correlation of 0 indicates that there is no relationship between the different variables.
- Values between -1 and 1 denote the strength of the correlation, as shown in the example below.

Types of Correlation

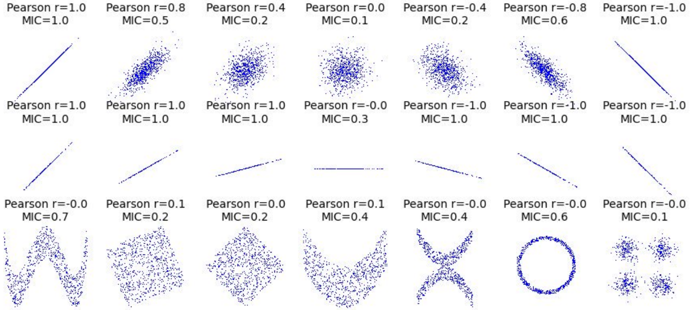


- Negative correlations describe the inverse of growth in one variable with another.

Types of Correlation



Other Types of Correlation



- A statistical measurement to describe the nature and strength of the relationship between two sets of numbers:
- Also called the Pearson correlation coefficient

Equation for Correlation

The correlation between sets, x and y is defined by the following:

$$\text{Correlation(x,y)} = \frac{n\Sigma xy - \Sigma x \Sigma y}{\sqrt{(n\Sigma x^2 - (\Sigma x)^2)(n\Sigma y^2 - (\Sigma y)^2)}}$$

$\sum xy$	Sum of the products of the individual elements of the two se of numbers, x and y
$\sum x$	Sum of the numbers in set x
$\sum y$	Sum of the numbers in set y
$(\sum x)^2$	Square of the sum of the numbers in set x
$(\sum y)^2$	Square of the sum of the numbers in set y
$\sum x^2$	Sum of the squares of the numbers in set x
$\sum y^2$	Sum of the squares of the numbers in set y

Equation for Correlation

- We will use the `zip` function in python

```
simple_list1 = [1, 2, 3]
simple_list2 = [4, 5, 6]
for x, y in zip(simple_list1, simple_list2):
  print(x, y)
# outputs:
# 1 4
# 2 5
# 3 6
```

- And now, on to the correlation code...

Correlation Code 1

See source code: `file: correlation.py`

```
def find_corr_x_y(x,y):
       n = len(x)
       # Find the sum of the products
       prod = []
       for xi,yi in zip(x,y): # the zip() function
           prod.append(xi*yi)
       sum_prod_x_y = sum(prod)
       sum_x = sum(x)
       sum_y = sum(y)
       squared_sum_x = sum_x**2
       squared_sum_y = sum_y**2
       x_square = []
       for xi in x:
           x_square.append(xi**2)
       # Find the sum
       x_square_sum = sum(x_square)
       y_square=[]
       for yi in y:
           y_square.append(yi**2)
       # Find the sum
       y_square_sum = sum(y_square)
# Use formula to calculate correlation
       numerator = n*sum_prod_x_y - sum_x*sum_y
       denominator_term1 = n*x_square_sum - squared_sum_x
       denominator_term2 = n*y_square_sum - squared_sum_y
       denominator = (denominator_term1*denominator_term2)**0.5
       correlation = numerator/denominator
       return correlation
#end of find_corr_x_y()
```

Correlation Code 2

See source code: `file: correlation.py`


```
simple_list1 = [1,2,3]
simple_list2 = [4,5,5]
result = find_corr_x_y(simple_list1,simple_list2)
print(" Set1:",simple_list1)
print(" Set2:",simple_list2)
print(" result :",result)
```

ALLEGHENY COLLEGE

Consider This Application
Data

- A fictional group of 10 students in high school
- Investigate whether there is a relationship between their *grades* in school and their performance on *college admission tests*.

```
#High_School_Grades_list
x = [90, 92, 95, 96, 87, 87, 90, 95, 98, 96]
#College_Admin_Tests_list
y = [85, 87, 86, 97, 96, 88, 89, 98, 98, 87]
```

Consider This Application

Analysis: Is there a correlation?

Produce code to ...

- For two different lists of data, find any two basic statistical measurements of each list, in addition to a correlation analysis between both.
- Produce a scatter plot and other types, as you feel necessary, of the points in each list:
- Answer the question: Is there a correlation between these two variables shown above?
- Tie all this code together to be in one program.

Basic Stats
Common Elements
Variance
Standard Deviation
Correlation
Application

statistics - Basic statistics module.

```
DESCRIPTION
    This module provides functions for calculating statistics of data, including
    averages, variance, and standard deviation.

    Calculating averages
    --------------------

    ==================  ==============================================
    Function            Description
    ==================  ==============================================
    mean                Arithmetic mean (average) of data.
    harmonic_mean       Harmonic mean of data.
    median              Median (middle value) of data.
    median_low          Low median of data.
    median_high         High median of data.
    median_grouped      Median, or 50th percentile, of grouped data.
    mode                Mode (most common value) of data.
    ==================  ==============================================
```

```
import statistics
statistics.mean([1,2,3])
statistics.pvariance([12, 12, 12, 12, 13013])
```