Describing Data in Code

Data descriptors

- Mean
- Mode
- Median

descriptors of "central tendency"

- Range
- Quartile
- Variance
- Standard Deviation

descriptors of "distribution"

All of these are single numbers that describe a variable (which is multiple numbers)

Code for all of these is below

Example Dataset

TABLE 2.1 Data Table Showing Five Car Records Described by Nine Variables

Name	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
Chevrolet Chevelle Malibu	18	8	307	130	3504	12	70	America
Buick Skylark 320	15	8	350	165	3693	11.5	70	America
Plymouth Satellite	18	8	318	150	3436	11	70	America
AMC Rebel SST	16	8	304	150	3433	12	70	America
Ford Torino	17	8	302	140	3449	10.5	70	America

- number of observations:
- number of variables:

Example Dataset

TABLE 2.1 Data Table Showing Five Car Records Described by Nine Variables

Name	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
Chevrolet Chevelle Malibu	18	8	307	130	3504	12	70	America
Buick Skylark 320	15	8	350	165	3693	11.5	70	America
Plymouth Satellite	18	8	318	150	3436	11	70	America
AMC Rebel SST	16	8	304	150	3433	12	70	America
Ford Torino	17	8	302	140	3449	10.5	70	America

- number of observations: 5
- number of variables: 9

Dataset in code - bad

TABLE 2.1 Data Table Showing Five Car Records Described by Nine Variables

Name	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model Year	Origin
Chevrolet Chevelle Malibu	18	8	307	130	3504	12	70	America
Buick Skylark 320	15	8	350	165	3693	11.5	70	America
Plymouth Satellite	18	8	318	150	3436	11	70	America
AMC Rebel SST	16	8	304	150	3433	12	70	America
Ford Torino	17	8	302	140	3449	10.5	70	America

```
obs0 = np.array(['chev', 18, 8, 307, 130, 3504, 12, 70, 'usa'])
```

- we want to know if there are trends in each variable or relationships between the variables
- **cannot compute the mean of this observation…!**

Dataset in code - good for small datasets

```
import numpy as np

name = np.array(['chev', 'buick', 'plymouth', 'amc', 'ford'])
mpg = np.array([18, 15, 18, 16, 17])
cylinders = np.array([8,8,8,8,8])
displacement = np.array([307, 350, 318, 304, 302])
horsepow = np.array([130, 165, 150, 150, 140])
weight = np.array([3504, 3693, 3436, 3433, 3449])
accel = np.array([12, 11.5, 11, 12, 10.5])
year = np.array([70, 70, 70, 70, 70])
origin = np.array(['usa','usa','usa','usa','usa'])
```

Mean - in code

- sum and divide by total num observations

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# sum up everything in selected variable
sum = np.sum(selected_variable)

# normalize by num elements
mean = sum / np.size(selected_variable)
```

```python
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# sum up everything in selected variable
sum = np.sum(selected_variable)

# normalize by num elements
mean = sum / np.size(selected_variable)
```

```python
# print out the results
print("Now reporting on the variable,", selected_variable_name)
print("---------------------------")
print("original data:", selected_variable)
print("sum:", sum)
print("mean:", mean)
```

```
Now reporting on the variable, mpg
---------------------------
original data: [18 15 18 16 17]
sum: 84
mean: 16.8
```

numpy functions for mean

- np.array()
- np.sum()
- np.size()

regular python

- print()

Coding concepts

- assignment: =
- division: /

Mode - in code

- most commonly observed value in a variable (categorical or numbers)
- in numpy, use the pre existing function "unique" to figure out what values are unique and what values are repeated

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# use the selected variable in the np.unique function
unique_things, counts = np.unique(selected_variable, return_counts = True)

# select the mode based on the quantity of each thing in the selected variable
mode = unique_things[np.argmax(counts)]

# sanity check in case there is no actual mode...
mode_is_good = max(counts) != min(counts)
```

```python
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# use the selected variable in the np.unique function
unique_things, counts = np.unique(selected_variable, return_counts = True)

# select the mode based on the quantity of each thing in the selected variable
mode = unique_things[np.argmax(counts)]

# sanity check in case there is no actual mode...
mode_is_good = max(counts) != min(counts)
```

```python
# print out the results
print("Now reporting on the variable,", selected_variable_name)
print("---------------------------")
print("original data:", selected_variable)
print("unique things:", unique_things)
print("counts of unique things:", counts)
print("mode:", mode)
print("mode reliable:", mode_is_good)
```

```
Now reporting on the variable, mpg
---------------------------
original data: [18 15 18 16 17]
unique things: [15 16 17 18]
counts of unique things: [1 1 1 2]
mode: 18
mode reliable: True
```

numpy functions for mode

- np.array()
- np.unique()
- np.argmax()

regular python

- max()
- min()
- print()

Coding concepts

- assignment: =
- indexing into array: [ ]
- value comparisons: !=

Median - in code

- middle, in sorted variable
- if there is no exact middle, average the two closest to the middle

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# sort the selected variable
sorted_variable = np.sort(selected_variable)

# compute which index (position) holds the middle element
midpoint = np.size(sorted_variable)//2

# read the median, but only if the midpoint is actually good
if midpoint + 1 + midpoint == np.size(sorted_variable):
  median = sorted_variable[midpoint]

# do something else only if the midpoint was not good
else:
  lower_mid = midpoint - 1
  upper_mid = midpoint
  median = sorted_variable[lower_mid]/2 + sorted_variable[upper_mid]/2
```

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# sort the selected variable
sorted_variable = np.sort(selected_variable)

# compute which index (position) holds the middle element
midpoint = np.size(sorted_variable)//2

# read the median, but only if the midpoint is actually good
if midpoint + 1 + midpoint == np.size(sorted_variable):
  median = sorted_variable[midpoint]

# do something else only if the midpoint was not good
else:
  lower_mid = midpoint - 1
  upper_mid = midpoint
  median = sorted_variable[lower_mid]/2 + sorted_variable[upper_mid]/2
```

```
# print out the results
print("Now reporting on the variable,", selected_variable_name)
print("--------------------------")
print("original data:", selected_variable)
print("sorted data:", sorted_variable)
print("midpoint:", midpoint)
print("median:", median)
```

```
Now reporting on the variable, mpg
---------------------------
original data: [18 15 18 16 17]
sorted data: [15 16 17 18 18]
midpoint: 2
median: 17
```

numpy functions for median

- np.array()
- np.sort()
- np.size()

regular python

- print()

Coding concepts

- assignment: =
- division: /
- integer division: //
- addition: +
- subtraction: -
- indexing into array: [ ]
- value comparisons: ==
- conditional: if else

Descriptors for Distributions

- Range
- Quartile
- Variance
- Standard Deviation

Range - in code

- span

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# min
lowest = min(selected_variable)

# max
highest = max(selected_variable)

# range
range = highest - lowest
```

```
# print out the results
print("Now reporting on the variable,", selected_variable_name)
print("--------------------------")
print("original data:", selected_variable)
print("lowest:", lowest)
print("highest:", highest)
print("range:", range)
```

```
Now reporting on the variable, mpg
---------------------------
original data: [18 15 18 16 17]
lowest: 15
highest: 18
range: 3
```

numpy functions for mode

- np.array()

regular python

- max()
- min()
- print()

Coding concepts

- assignment: =
- subtraction: -

Quartiles - in code

- same as median, but additional "medians" are found for upper and lower halves of the data
- if the median is between two values, they are included in their respective halves

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg

# sort the selected variable
sorted_variable = np.sort(selected_variable)

# compute which index (position) holds the middle element
midpoint = np.size(sorted_variable)//2

# read the median, but only if the midpoint is actually good
if midpoint + 1 + midpoint == np.size(sorted_variable):
  median = sorted_variable[midpoint]
  # designate new boudaries
  lower_half_endpoint = midpoint - 1
  upper_half_startpoint = midpoint + 1

# do something else only if the midpoint was not good
else:
  lower_mid = midpoint - 1
  upper_mid = midpoint
  median = sorted_variable[lower_mid]/2 + sorted_variable[upper_mid]/2
  # designate new boudaries
  lower_half_endpoint = lower_mid
  upper_half_startpoint = upper_mid
```

Quartiles - in code part 2

For Lab 6

```
# designate the lower half of the sorted data
sorted_variable_lower_half = sorted_variable[0:lower_half_endpoint + 1]

# designate the upper half of the sorted data
sorted_variable_upper_half = sorted_variable[upper_half_startpoint:]

# copy the midpoint calculation and if else statements
# adjust python variable name `median` to `lower_quartile`
# or `upper_quartile` as needed
# make sure the computation operates on the sorted upper or lower half,
# and not on the original unsorted or sorted data.

# print out the results
print("Now reporting on the variable,", selected_variable_name)
print("---------------------------")
print("original data:", selected_variable)
print("sorted data:", sorted_variable)
print("midpoint:", midpoint)
print("median:", median)
print("lower half of variable:", sorted_variable_lower_half)
print("upper half of variable:", sorted_variable_upper_half)
```

numpy functions for quartile

- np.array()
- np.sort()
- np.size()

regular python

- print()

Coding concepts

- assignment: =
- division: /
- integer division: //
- addition: +
- subtraction: -
- indexing into array: [ ]
- value comparisons: ==
- conditional: if else

Quartiles alt

Function

```
# alt: create generic instructions about how to find a median (all indented)
def find_median(sorted_variable):

  # compute which index (position) holds the middle element
  midpoint = np.size(sorted_variable)//2

  # read the median, but only if the midpoint is actually good
  if midpoint + 1 + midpoint == np.size(sorted_variable):
    median = sorted_variable[midpoint]
    # designate new boudaries
    lower_half_endpoint = midpoint - 1
    upper_half_startpoint = midpoint + 1

  # do something else only if the midpoint was not good
  else:
    lower_mid = midpoint - 1
    upper_mid = midpoint
    median = sorted_variable[lower_mid]/2 + sorted_variable[upper_mid]/2
    # designate new boudaries
    lower_half_endpoint = lower_mid
    upper_half_startpoint = upper_mid

  return median, lower_half_endpoint, upper_half_startpoint
```

Variance

- describes how much variation there is in given data around the mean
- normalized by the number of observations

Comparison to Mean

- subtracting off the mean
- squaring to amplify the "errors", discarding the sign

Normalization

- almost like finding mean
- division of sum squared errors by (num observations - 1)

Variance in code

See slides from March 1, 2024:

```python
import numpy as np

# create numpy array with original data
x = np.array([3, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 8, 9])

# find number of observations
n = np.size(x)

# mean
xbar = np.sum(x) / n

# error
error = x - xbar

# squared error
se = error**2

# sum of squared error over all observations
sse = np.sum(se)

# normalization by number observations - 1
result = sse/(n-1)

# look at the result
print(result)
```

```
2.858974358974359
```

numpy functions for variance

- np.array()
- np.sum()
- np.size()

regular python

- print()

Coding concepts

- assignment: =
- division: /
- subtraction: -
- squaring: **

Standard Deviation

- square root of variance

numpy functions for standard deviation

- np.array()
- np.sum()
- np.size()

regular python

- print()

Coding concepts

- assignment: =
- division: /
- subtraction: -
- squaring: **
- square root: **.5

Hint for Lab 6

Use this code to make your python scripts interactive and more flexible!

```
# choose a variable to examine
selected_variable_name = input("write down the name of the variable to test:")
selected_variable = eval(selected_variable_name)
```

The above code should replace the following:

```
# choose a variable to examine
selected_variable_name = "mpg"
selected_variable = mpg
```

Next time

test statistics for hypothesis testing

distributions and histograms