YData: Introduction to Data Science

Class 14: seaborn continued and text manipulation

Overview

Quick review and continuation of data visualization with seaborn

Text manipulation!

If there is time

- Regular expressions

Announcement: Homework 6

Homework 6 has been posted!

It is due on Gradescope on Sunday March 5th at 11pm

- **Be sure to mark each question on Gradescope along with the page that has the answers!**

Midterm exam

Thursday March 9th in person during regular class time

- Exam is on paper

As part of homework 6, you will post a practice problem to Canvas

- I will take one of these problems and put it on the exam

A practice exam (last year's exam) has been posted

- The questions on this year's exam will be quite a bit different but it should give a general idea what the exam will be like

Midterm exam “cheat sheet”

You are allowed an exam “cheat sheet”

One page, double sided, that contains **only code**

- No code comments allowed

Cheat sheet must be on a regular 8.5 x 11 piece of paper

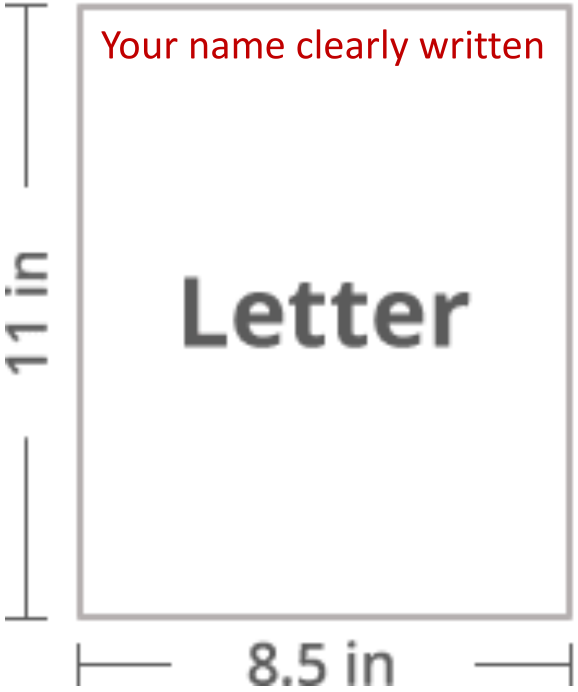
- Your name on the upper left of both sides of the paper

Strongly recommend making a typed list of all functions discussed in class and on the homework

- This will be useful beyond the exam

You must turn in your cheat sheet with the exam

- Failure to do so will result in a 20 point deduction



Review of data visualization!

Seaborn review

"Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics."

- i.e., it will create better looking plots that are easier to make

There are ways to create visualizations in seaborn:

1. **axes-level** functions that plot on a single axis
2. **figure-level** functions that plot across multiple axes

We will focus on figure level plots

To make plots better looking we can set a theme

```
import seaborn as sns

sns.set_theme()
```

Review: Seaborn figure level plots

Figure level plots are grouped based on the types of variables being plotted

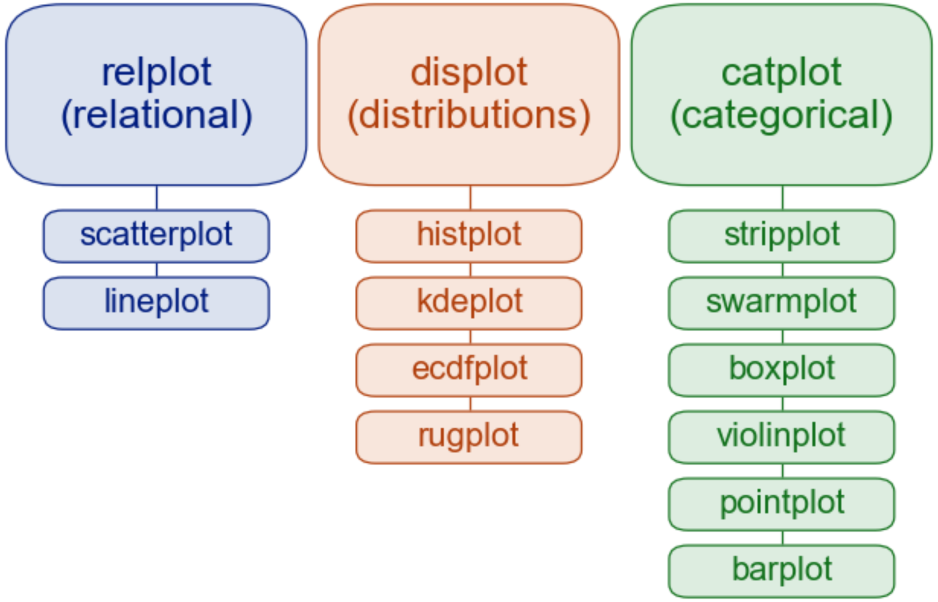
In particular, there are plots for:

1. Two quantitative variables
   - sns.relplot()

2. A single quantitative variable
   - sns.displot()

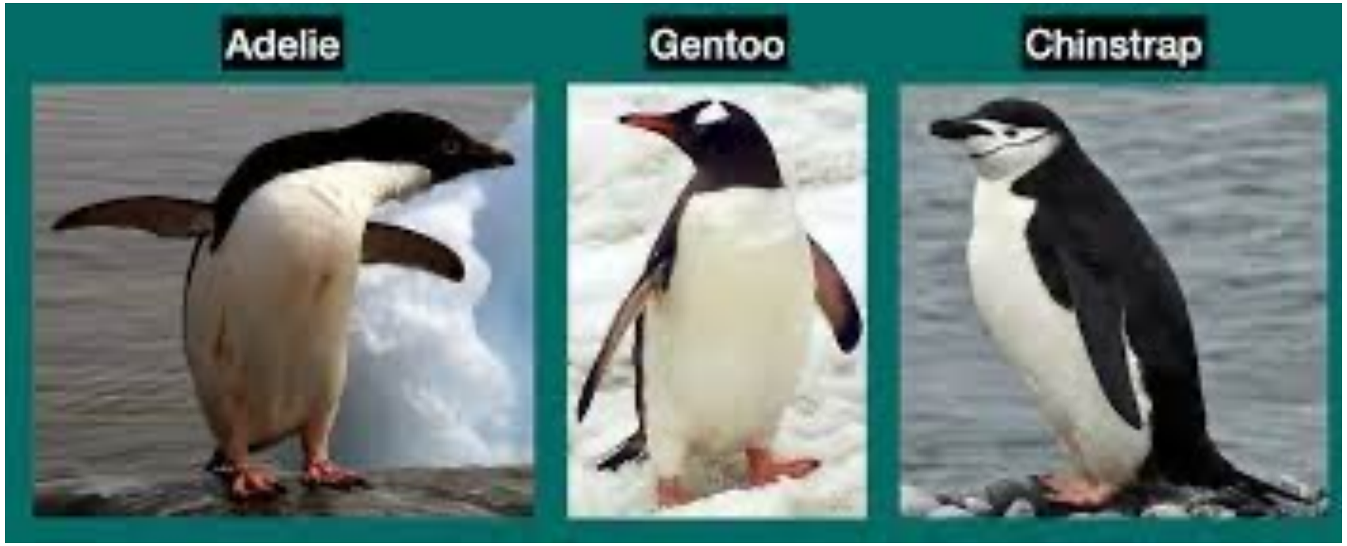
Quantitative variable compared across different categorical levels
   - sns.catplot()

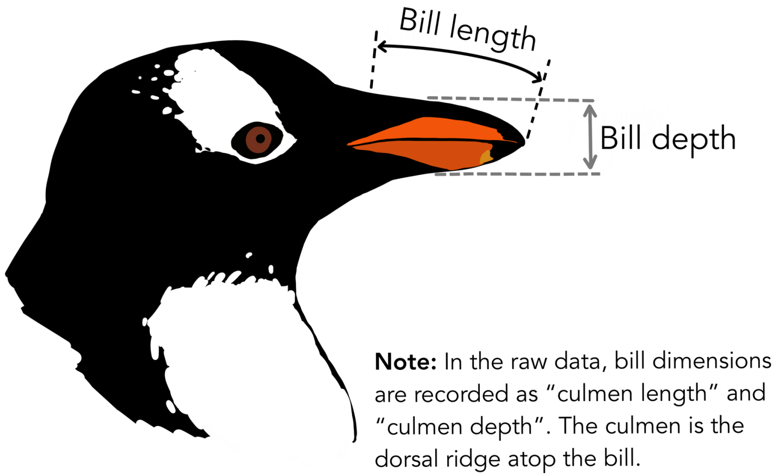
Figure level plots



Inspiration: Palmer penguins

To explore seaborn, let’s look at some data on penguins!



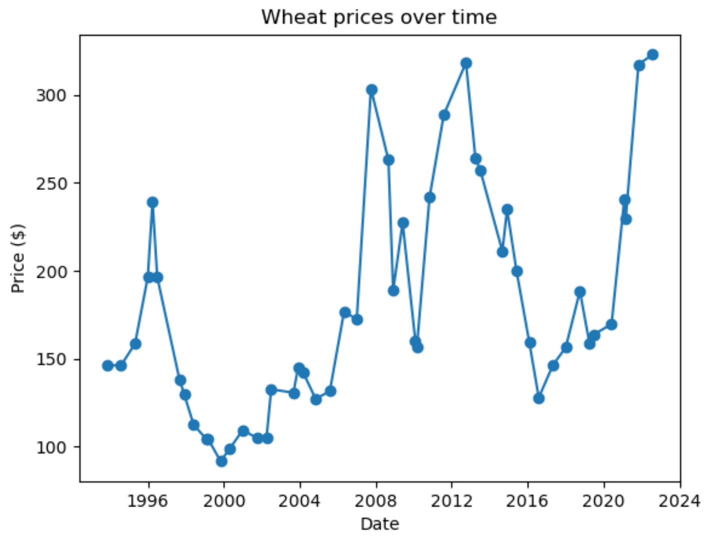


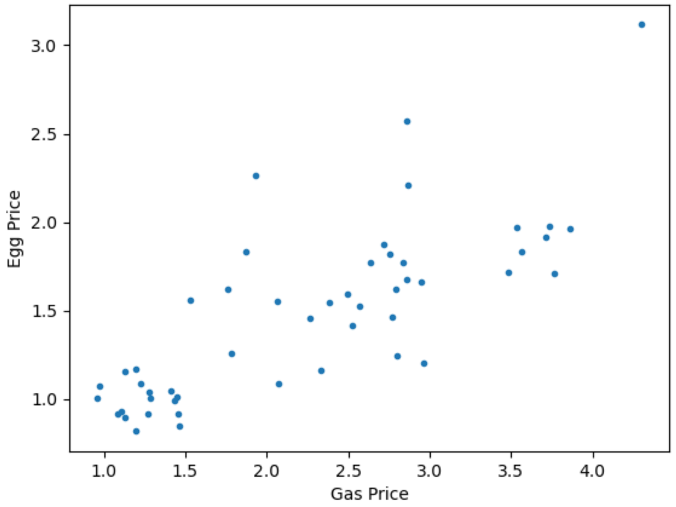
Note: In the raw data, bill dimensions are recorded as “culmen length” and “culmen depth”. The culmen is the dorsal ridge atop the bill.

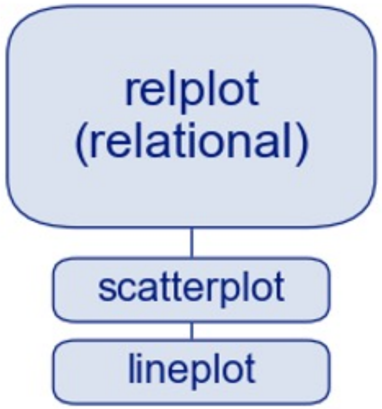
Review: Plots for two quantitative variable

What types of plots have we seen for assessing the relationships between two quantitative variable?

- Line plots and scatter plots!







relplot (relational)

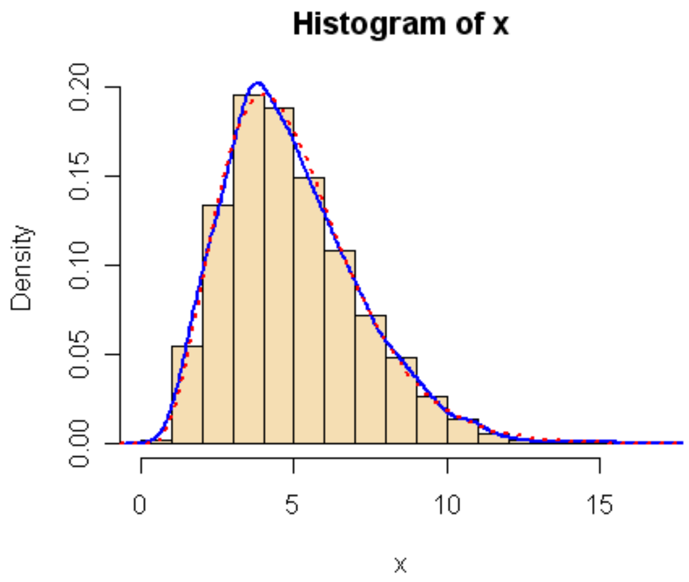
scatterplot

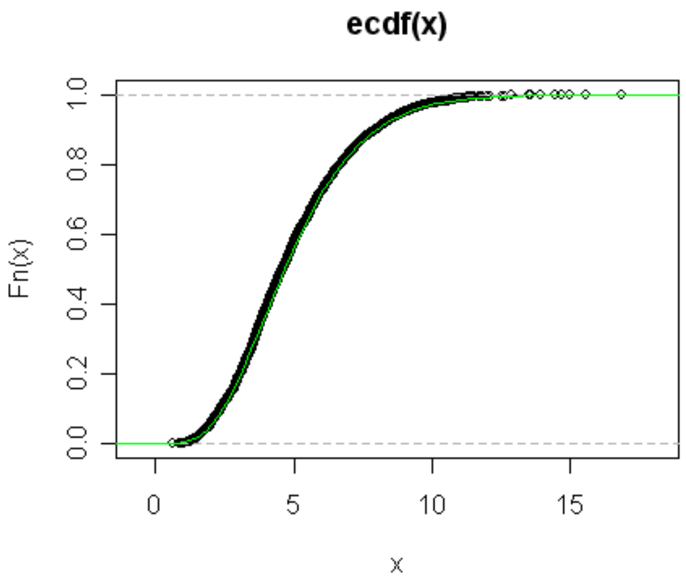
lineplot

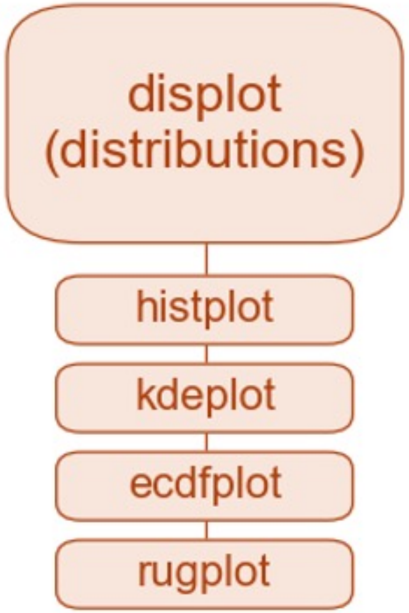
Review: Plots for a single quantitative variable

What types of plots have we seen for plotting a single quantitative variable?

- Histograms, kernel density estimates (kde), empirical distribution functions (ecdf)!







displot (distributions)

histplot

kdeplot

ecdfplot

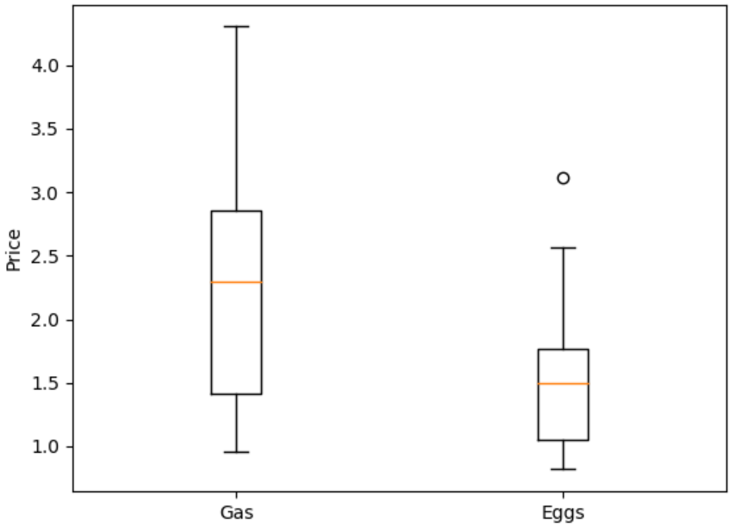
rugplot

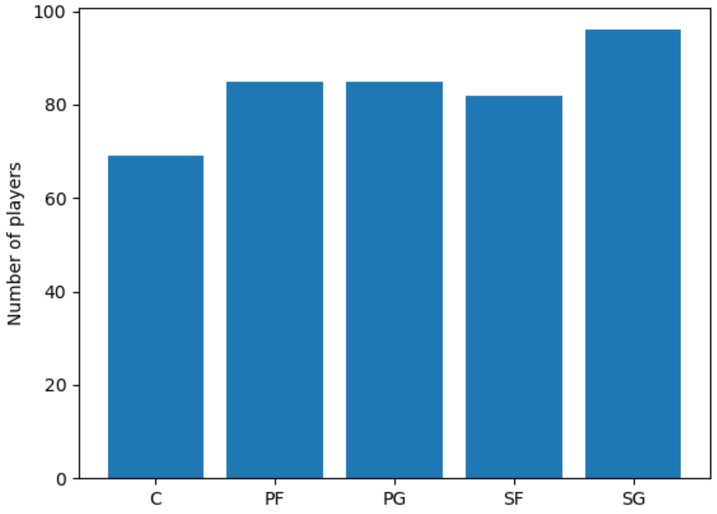
Let’s do some warm-up exercises in Jupyter!

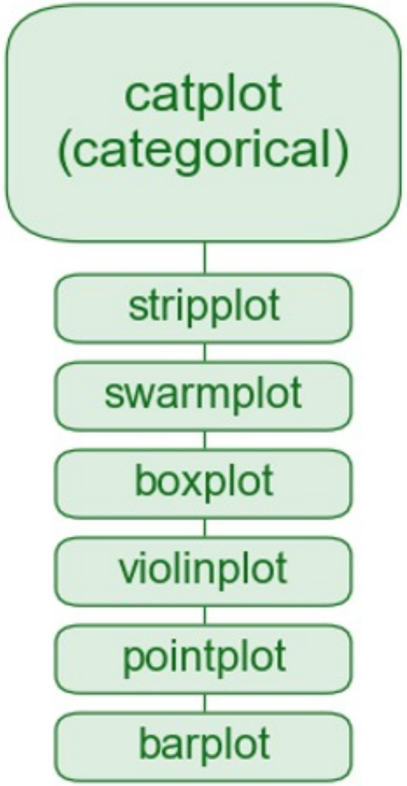
Plots for quantitative data comparing across different categorical levels

What types of plots have we seen comparing quantitative data at different levels of a categorical variable?

- Side-by-side boxplots, barplots (sort of)







catplot (categorical)

stripplot

swarmplot

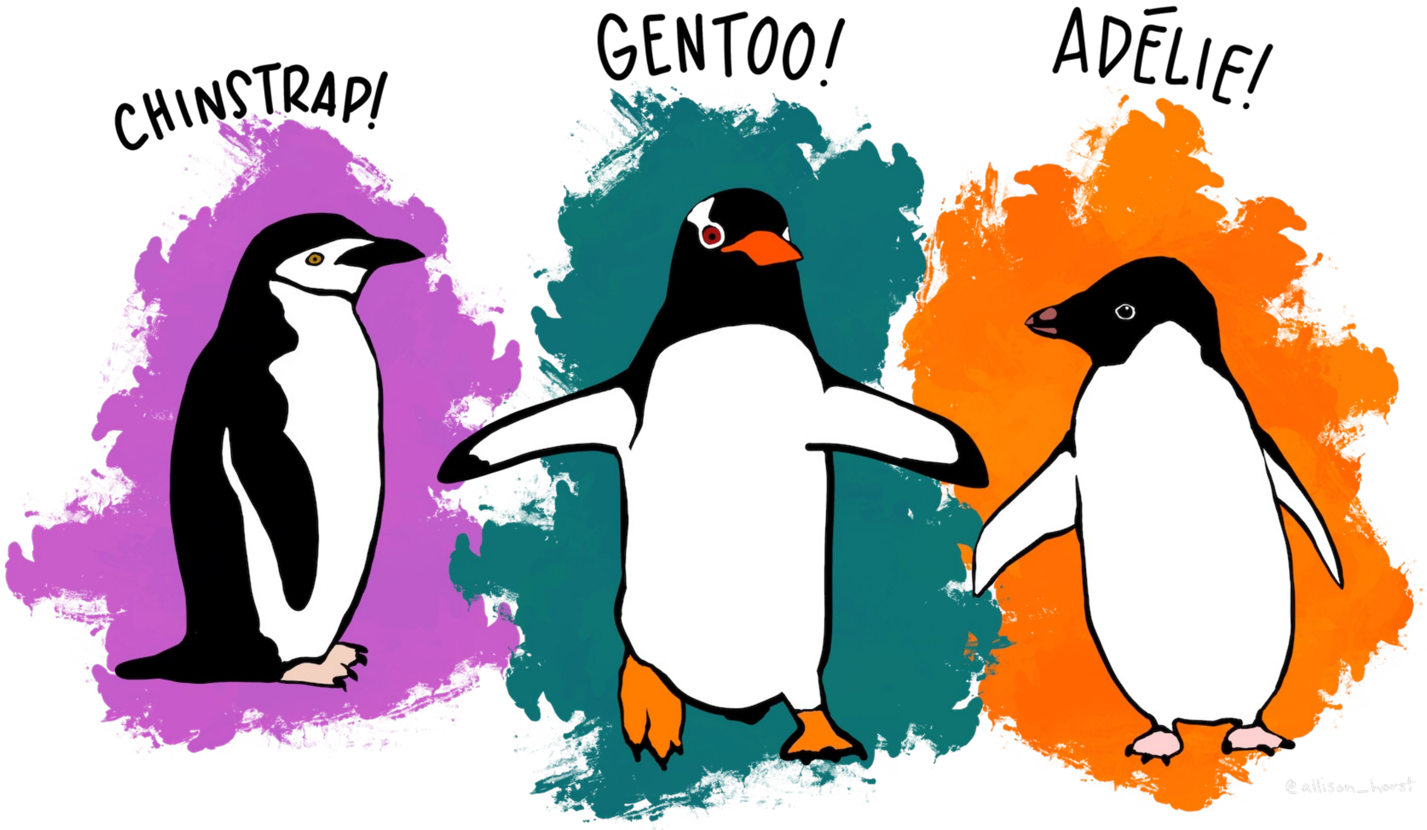
boxplot

violinplot

pointplot

barplot

Let's explore this in Jupyter!



CHINSTRAP!
GENTOO!
ADÉLIE!
@allison_horst

text
MaNiPuLaTiOn

Text manipulation

80% of a Data Scientists time is cleaning data

- Text manipulation is a big part of cleaning data

20% of a Data Scientists time is complaining about cleaning data

Python has many string methods that are useful for manipulating text and cleaning data!

Text manipulation: capitalization

Some of the simplest string methods involve changing capitalization.

Changing capitalization can be useful when joining DataFrames

- i.e., if they key values are the same, but the values have different capitalization.
  - For example, joining different countries, but in one DataFrame the country names are capitalized and in the other they are not

Text manipulation: capitalization

Python strings have a number of methods to change the capitalization of words including:

- .capitalize(): Converts the first character to upper case
- .lower(): Converts a string into lower case
- .upper(): Converts a string into upper case
- .title(): Converts the first character of each word to upper case
- .swapcase(): Swaps cases, lower case becomes upper case and vice versa

Let's explore this in Jupyter!

Text manipulation: string padding

Often we want to remove extra spaces (called "white space") from the front or end of a string.

Conversely, sometimes we want to add extra spaces to make a set of strings the same length

- This is known as "string padding"

Python strings have a number of methods that can pad/trim strings including:

- strip(): Returns a trimmed version of the string (i.e., with no leading or trailing white space).
  - Also, rstrip() and lstrip(): Returns a right/left trim version of the string
- center(num): Returns a centered string (with equal padding on both sides)
  - Also ljust(num) and rjust(num): Returns a right justified version of the string
- zfill(num): Fills the string with a specified number of 0 values at the beginning

Let's explore this in Jupyter!

Text manipulation: checking string properties

There are also many functions to check properties of strings including:

- isalnum(): Returns True if all characters in the string are alphanumeric
- isalpha(): Returns True if all characters in the string are in the alphabet
- isnumeric(): Returns True if all characters in the string are numeric

- isspace(): Returns True if all characters in the string are whitespaces

- islower(): Returns True if all characters in the string are lower case
- isupper(): Returns True if all characters in the string are upper case
- istitle()`: Returns True if the string follows the rules of a title

Let's explore this in Jupyter!

Text manipulation: splitting and joining strings

There are several methods that can help us join strings that are contained into a list into a single string, or conversely, parse a single string into a list of strings. These include:

- split(separator_string): Splits the string at the specified separator, and returns a list
- splitlines(): Splits the string at line breaks and returns a list
- join(a_list): Converts the elements of an iterable into a string

Let's explore this in Jupyter!

Text manipulation: finding and replacing substrings

Some methods for locating a substring within a larger string include:

- count(substring): Returns the number of times a specified value occurs in a string

- rfind(substring): Searches the string for a specified value and returns the last position of where it was found.

- startswith(substring): Returns true if the string starts with the specified value
- endswith(substring) : Returns true if the string ends with the specified value

- replace(original_str, replacement_str): Replace a substring with a different string.

Let's explore this in Jupyter!

Text manipulation: filling in strings with values

There are a number of ways to fill in strings parts of a string with particular values.

Perhaps the most useful is to use "f strings", which have the following syntax such as:

- value_to_fill = “my_value”
- f"my string {value_to_fill} will be filled in"

Let's explore this in Jupyter!

Regular expressions!

Regular expressions

Regular expressions are string that allow you find more complex patterns in pieces of text

- They are powerful although can be a bit hard to read

```
[^ ]*?@[^ ]*?\.[^ ]*
```

To use regular expressions in Python we can import the re module

```
import re
```

We can check if a piece of text contains a particular substring by converting the output of re.match() method into a Boolean

```
bool(re.match(“regular_expression”, “piece_of_text”))
```

Regular expressions

[ ] means match anything in the range inside the braces

- "ch[io]mp" matches ”chimp” and "chomp"

Note: if the ^ appears inside square braces it means not

- ^[^aeiou] matches words that don't start with a lower case vowel

The following are special regular expression characters that are reserved:

. * \ $ { } [ ] ^ ?

Let's explore this in Jupyter!

Regular expressions

. (period) matches any single character

- bool(re.match(”m.ss", ”mess"))

* means match 0 or more of the preceding character

- bool(re.match(”xy*z", ”xz"))

+ means match 1 or more of the preceding character

- bool(re.match(”xy+z", ”xz"))

will the following match?

- bool(re.match(”.*a.*e”, “pineapple"))

Example

```
phone_strings  =  [ "apple",
        "219  733  8965",
        "329-293-8753",
        "Work: 579-499-7527",
        "Home: 543.355.3679”]
```

The phone number can be matched with the regular expression:

```
”.*([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"
```

Let's explore this in Jupyter!

Escape sequences

In regular expressions a period (.) means any character

- So how can you detect if a period is in a string?

Escape sequences in R start with two slashes \\ and cause the next character to be treated literally rather than as a special character

- To match a period we use \\.
- To match a $ symbol we use \\$

Example

- `bool(re.match(".*\\$100", "Joanna has $100 and Chris has $0"))`

Character classes

Other special characters are also designated by using a double slash first

- \s space
- \n new line or also \r
- \t tab

Let's explore this in Jupyter!

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP