boston\_cocktails <- readr::read\_csv("<https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2020/2020-05-26/boston_cocktails.csv>")

boston\_cocktails %>%

count(ingredient, sort = TRUE)

## # A tibble: 569 x 2

## ingredient n

##

## 1 Gin 176

## 2 Fresh lemon juice 138

## 3 Simple Syrup 115

## 4 Vodka 114

## 5 Light Rum 113

## 6 Dry Vermouth 107

## 7 Fresh Lime Juice 107

## 8 Triple Sec 107

## 9 Powdered Sugar 90

## 10 Grenadine 85

## # … with 559 more rows

There’s a bit of data cleaning to do to start, both for the ingredient column and the measure column.

cocktails\_parsed <- boston\_cocktails %>%

mutate(

ingredient = str\_to\_lower(ingredient),

ingredient = str\_replace\_all(ingredient, "-", " "),

ingredient = str\_remove(ingredient, " liqueur$"),

ingredient = str\_remove(ingredient, " (if desired)$"),

ingredient = case\_when(

str\_detect(ingredient, "bitters") ~ "bitters",

str\_detect(ingredient, "lemon") ~ "lemon juice",

str\_detect(ingredient, "lime") ~ "lime juice",

str\_detect(ingredient, "grapefruit") ~ "grapefruit juice",

str\_detect(ingredient, "orange") ~ "orange juice",

TRUE ~ ingredient

),

measure = case\_when(

str\_detect(ingredient, "bitters") ~ str\_replace(measure, "oz$", "dash"),

TRUE ~ measure

),

measure = str\_replace(measure, " ?1/2", ".5"),

measure = str\_replace(measure, " ?3/4", ".75"),

measure = str\_replace(measure, " ?1/4", ".25"),

measure\_number = parse\_number(measure),

measure\_number = if\_else(str\_detect(measure, "dash$"),

measure\_number / 50,

measure\_number

)

) %>%

add\_count(ingredient) %>%

filter(n > 15) %>%

select(-n) %>%

distinct(row\_id, ingredient, .keep\_all = TRUE) %>%

na.omit()

cocktails\_parsed

## # A tibble: 2,542 x 7

## name category row\_id ingredient\_numb… ingredient measure measure\_number

##

## 1 Gauguin Cocktail … 1 1 light rum 2 oz 2

## 2 Gauguin Cocktail … 1 3 lemon jui… 1 oz 1

## 3 Gauguin Cocktail … 1 4 lime juice 1 oz 1

## 4 Fort La… Cocktail … 2 1 light rum 1.5 oz 1.5

## 5 Fort La… Cocktail … 2 2 sweet ver… .5 oz 0.5

## 6 Fort La… Cocktail … 2 3 orange ju… .25 oz 0.25

## 7 Fort La… Cocktail … 2 4 lime juice .25 oz 0.25

## 8 Cuban C… Cocktail … 4 1 lime juice .5 oz 0.5

## 9 Cuban C… Cocktail … 4 2 powdered … .5 oz 0.5

## 10 Cuban C… Cocktail … 4 3 light rum 2 oz 2

## # … with 2,532 more rows

I typically do my data cleaning with data in a tidy format, like boston\_cocktails or cocktails\_parsed. When it’s time for modeling, we usually need the data in a wider format, so let’s use pivot\_wider() to reshape our data.

cocktails\_df <- cocktails\_parsed %>%

select(-ingredient\_number, -row\_id, -measure) %>%

pivot\_wider(names\_from = ingredient, values\_from = measure\_number, values\_fill = 0) %>%

janitor::clean\_names() %>%

na.omit()

cocktails\_df

## # A tibble: 937 x 42

## name category light\_rum lemon\_juice lime\_juice sweet\_vermouth orange\_juice

##

## 1 Gaug… Cocktai… 2 1 1 0 0

## 2 Fort… Cocktai… 1.5 0 0.25 0.5 0.25

## 3 Cuba… Cocktai… 2 0 0.5 0 0

## 4 Cool… Cocktai… 0 0 0 0 1

## 5 John… Whiskies 0 1 0 0 0

## 6 Cher… Cocktai… 1.25 0 0 0 0

## 7 Casa… Cocktai… 2 0 1.5 0 0

## 8 Cari… Cocktai… 0.5 0 0 0 0

## 9 Ambe… Cordial… 0 0.25 0 0 0

## 10 The … Whiskies 0 0.5 0 0 0

## # … with 927 more rows, and 35 more variables: powdered\_sugar ,

## # dark\_rum , cranberry\_juice , pineapple\_juice ,

## # bourbon\_whiskey , simple\_syrup , cherry\_flavored\_brandy ,

## # light\_cream , triple\_sec , maraschino , amaretto ,

## # grenadine , apple\_brandy , brandy , gin ,

## # anisette , dry\_vermouth , apricot\_flavored\_brandy ,

## # bitters , straight\_rye\_whiskey , benedictine ,

## # egg\_white , half\_and\_half , vodka , grapefruit\_juice ,

## # blended\_scotch\_whiskey , port , white\_creme\_de\_cacao ,

## # citrus\_flavored\_vodka , whole\_egg , egg\_yolk ,

## # blended\_whiskey , dubonnet , blanco\_tequila ,

## # old\_mr\_boston\_dry\_gin

**Principal component analysis**

This dataset is especially delightful because we get to use  
[recipes](https://recipes.tidymodels.org/) with **recipes**. Let’s load the tidymodels metapackage and implement principal component analysis with a recipe.

library(tidymodels)

pca\_rec <- recipe(~., data = cocktails\_df) %>%

update\_role(name, category, new\_role = "id") %>%

step\_normalize(all\_predictors()) %>%

step\_pca(all\_predictors())

pca\_prep <- prep(pca\_rec)

pca\_prep

## Data Recipe

##

## Inputs:

##

## role #variables

## id 2

## predictor 40

##

## Training data contained 937 data points and no missing data.

##

## Operations:

##

## Centering and scaling for light\_rum, lemon\_juice, ... [trained]

## PCA extraction with light\_rum, lemon\_juice, ... [trained]

Let’s walk through the steps in this recipe.

* First, we must tell the recipe() what’s going on with our model (notice the formula with *no outcome*) and what data we are using.
* Next, we update the role for cocktail name and category, since these are variables we want to keep around for convenience as identifiers for rows but are not a predictor or outcome.
* We need to center and scale the numeric predictors, because we are about to implement PCA.
* Finally, we use step\_pca() for the actual principal component analysis.

Before using prep() these steps have been defined but not actually run or implemented. The prep() function is where everything gets evaluated.

Once we have that done, we can both explore the results of the PCA. Let’s start with checking out how the PCA turned out. We can tidy() any of our recipe steps, including the PCA step, which is the second step. Then let’s make a visualization to see what the components look like.

tidied\_pca <- tidy(pca\_prep, 2)

tidied\_pca %>%

filter(component %in% paste0("PC", 1:5)) %>%

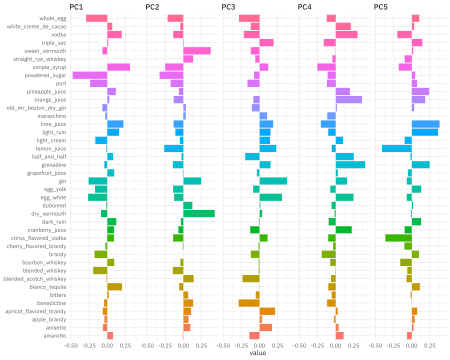
mutate(component = fct\_inorder(component)) %>%

ggplot(aes(value, terms, fill = terms)) +

geom\_col(show.legend = FALSE) +

facet\_wrap(~component, nrow = 1) +

labs(y = NULL)



The biggest difference in PC1 is powdered sugar vs. simple syrup; recipes are not likely to have both, which makes sense! Let’s zoom in on the first four components, and understand which cocktail ingredients contribute in the positive and negative directions.

library(tidytext)

tidied\_pca %>%

filter(component %in% paste0("PC", 1:4)) %>%

group\_by(component) %>%

top\_n(8, abs(value)) %>%

ungroup() %>%

mutate(terms = reorder\_within(terms, abs(value), component)) %>%

ggplot(aes(abs(value), terms, fill = value > 0)) +

geom\_col() +

facet\_wrap(~component, scales = "free\_y") +

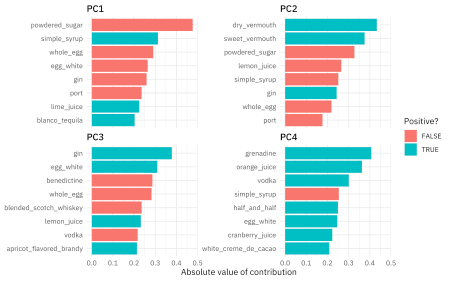
scale\_y\_reordered() +

labs(

x = "Absolute value of contribution",

y = NULL, fill = "Positive?"

)



So PC1 is about powdered sugar + egg + gin drinks vs. simple syrup + lime + tequila drinks. This is the component that explains the most variation in drinks. PC2 is mostly about vermouth, both sweet and dry.

How are the cocktails distributed in the plane of the first two components?

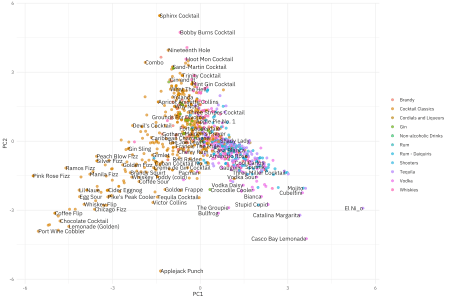
juice(pca\_prep) %>%

ggplot(aes(PC1, PC2, label = name)) +

geom\_point(aes(color = category), alpha = 0.7, size = 2) +

geom\_text(check\_overlap = TRUE, hjust = "inward", family = "IBMPlexSans") +

labs(color = NULL)



* Fizzy, egg, powdered sugar drinks are to the left.
* Simple syrup, lime, tequila drinks are to the right.
* Vermouth drinks are more to the top.

You can change out PC2 for PC4, for example, to instead see where drinks with more grenadine are.

**UMAP**

One of the benefits of the tidymodels ecosystem is the flexibility and ease of trying different approaches for the same kind of task. For example, we can switch out PCA for [UMAP](https://umap-learn.readthedocs.io/en/latest/how_umap_works.html), an entirely different algorithm for dimensionality reduction based on ideas from topological data analysis. The [embed](https://embed.tidymodels.org/) package provides recipe steps for ways to create embeddings including UMAP. Let’s switch out the PCA step for the UMAP step.

library(embed)

umap\_rec <- recipe(~., data = cocktails\_df) %>%

update\_role(name, category, new\_role = "id") %>%

step\_normalize(all\_predictors()) %>%

step\_umap(all\_predictors())

umap\_prep <- prep(umap\_rec)

umap\_prep

## Data Recipe

##

## Inputs:

##

## role #variables

## id 2

## predictor 40

##

## Training data contained 937 data points and no missing data.

##

## Operations:

##

## Centering and scaling for light\_rum, lemon\_juice, ... [trained]

## UMAP embedding for light\_rum, lemon\_juice, ... [trained]

Now we can example how the cocktails are distributed in the plane of the first two UMAP components.

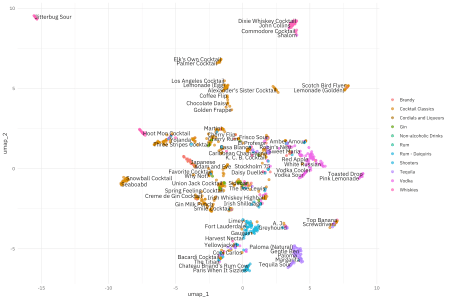
juice(umap\_prep) %>%

ggplot(aes(umap\_1, umap\_2, label = name)) +

geom\_point(aes(color = category), alpha = 0.7, size = 2) +

geom\_text(check\_overlap = TRUE, hjust = "inward", family = "IBMPlexSans") +

labs(color = NULL)



Really interesting, but also different! This is because UMAP is so different from PCA, although they are both approaching this question of how to project a set of features, like ingredients in cocktail recipes, into a smaller space.