Homework 2 Econ 235

Sayer Dobyns

Introduction

For this assignment, we'll using the following packages.

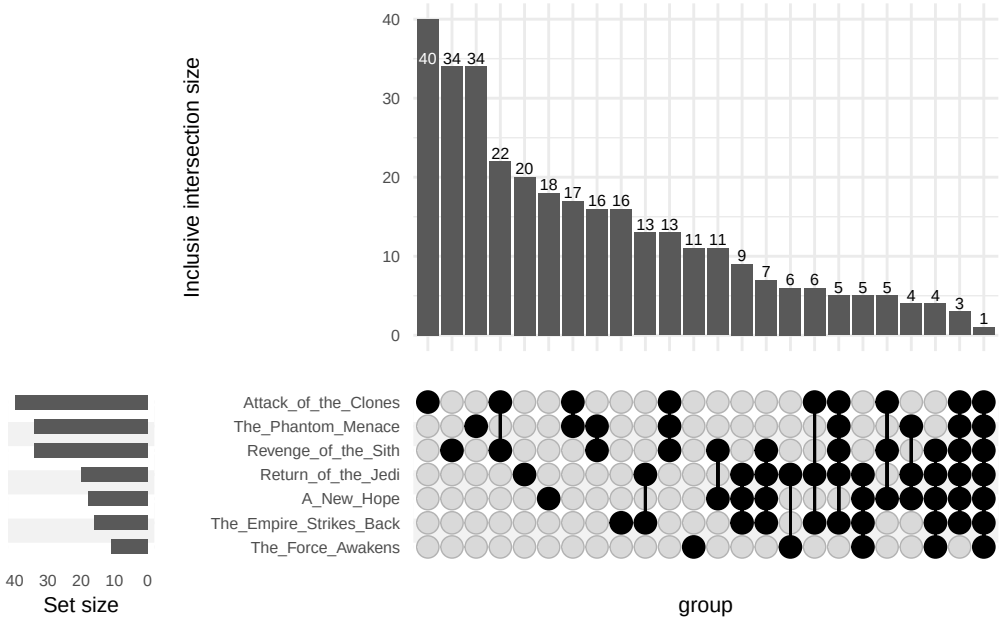
```
library(dplyr)
library(tibble)
library(ggplot2)
library(stringr)
library(tidyr)

`%nin%` <- Negate(`%in%`)

# This code installs then loads the ComplexUpset pkg if not already installed,
# and just loades the pkg if it's already been installed
if ("ComplexUpset" %nin% rownames(installed.packages())) {
  install.packages("ComplexUpset")
  library(ComplexUpset)
} else {
  library(ComplexUpset)
}
```

Exercise 1 - Set Theory

In class we looked at simple set relationships between two sets at a time. These are fundamental to understanding more complicated set relationships that might be more likely in other settings. In this exercise we'll deal with a setting a long time ago in a galaxy far, far away. The `starwars` data set that is included in the `dplyr` package has information on 87 named characters that appear and speak in the first seven films of the Star Wars franchise.

For our purposes, we'll consider each film to be set and the elements of the set will characters from the data set. This means we're now considering 7 different sets. It is very difficult to

visualize set relationships using any type of Venn diagram with that many sets. One solution to that issue is to generate an "UpSet" plot. An UpSet plot includes two bar charts and a dot matrix. The plot on the left is bar chart of the size (cardinality) of each original set – e.g. the numbers of characters in each movie. The plot on the top shows the size (cardinality) of the derived from intersecting different sets. Which sets are part of the intersection is represented by the dots in column of the dot matrix below each bar. So, for example, there are 8 characters in the `starwars` data set that appear/speak in the The Phantom Menace, Attack of the Clones, and Revenge of the Sith (see the 13th column of the dot matrix).



Below I have written code to create the sets of characters from each film. That way we can use the set functions from `dplyr` including `union`, `intersect`, and `setdiff` to answer some questions about how characters overlap or not across movies. For some of these questions, you can use the UpSet plot to check that your answers have the right number of characters.

```
starwars_chars_films_long <- starwars |>
  select(name,films) |>
  unnest_longer(films)

# Create the character set for: The Phantom Menace
TPM <- starwars_chars_films_long |>
  filter(films=="The Phantom Menace") |> pull(name)
# Create the character set for: Attack of the Clones
AotC <- starwars_chars_films_long |>
  filter(films=="Attack of the Clones") |> pull(name)
```

```
# Create the character set for: Revenge of the Sith
RotS <- starwars_chars_films_long |>
  filter(films=="Revenge of the Sith") |> pull(name)
# Create the character set for: A New Hope
ANH <- starwars_chars_films_long |>
  filter(films=="A New Hope") |> pull(name)
# Create the character set for: the Empire Strikes Back
ESB <- starwars_chars_films_long |>
  filter(films=="The Empire Strikes Back") |> pull(name)
# Create the character set for: Return of the Jedi
RotJ <- starwars_chars_films_long |>
  filter(films=="Return of the Jedi") |> pull(name)
# Create the character set for: The Force Awakens
TFA <- starwars_chars_films_long |>
  filter(films=="The Force Awakens") |> pull(name)
```

In the code chunk below use the `union`, `intersect`, and `setdiff` functions from the `dplyr` package to answer the question posed in each comment

```
# Characters in *all three* of the prequel trilogy films
prequel_characters <- Reduce(intersect, list(TPM, AotC, RotS))

# Characters in both Return of the Jedi and The Force Awakens
return_and_force <- intersect(RotJ, TFA)

# Characters in The Empire Strikes Back but *NOT* A New Hope
empire_not_newhope <- setdiff(ESB, ANH)

# Extra Credit: Character in *all* the films
all_films_character <- Reduce(intersect, list(TPM, AotC, RotS, ANH, ESB, RotJ, TFA))

# Print the results
print("Characters in all three prequel trilogy films:")
```

```
[1] "Characters in all three prequel trilogy films:"
```

```
print(prequel_characters)
```

```
 [1] "C-3PO"            "R2-D2"            "Obi-Wan Kenobi"   "Anakin Skywalker"
 [5] "Yoda"             "Palpatine"        "Nute Gunray"      "Padmé Amidala"
 [9] "Ayla Secura"      "Mace Windu"       "Ki-Adi-Mundi"     "Kit Fisto"
[13] "Plo Koon"
```

```
print("Characters in both Return of the Jedi and The Force Awakens:")
```

```
[1] "Characters in both Return of the Jedi and The Force Awakens:"
```

```
print(return_and_force)
```

```
[1] "Luke Skywalker" "R2-D2"          "Leia Organa"    "Chewbacca"
[5] "Han Solo"       "Ackbar"
```

```
print("Characters in The Empire Strikes Back but NOT A New Hope:")
```

```
[1] "Characters in The Empire Strikes Back but NOT A New Hope:"
```

```
print(empire_not_newhope)
```

```
[1] "Yoda"             "Palpatine"        "Boba Fett"        "IG-88"
[5] "Bossk"            "Lando Calrissian" "Lobot"
```

```
print("Character in all films:")
```

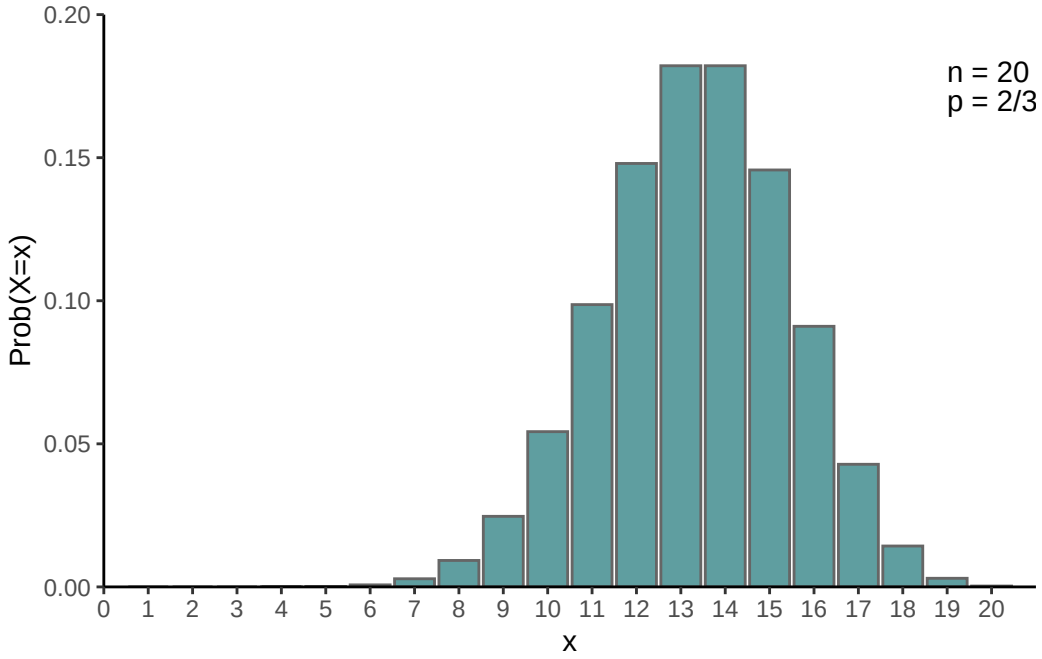
```
[1] "Character in all films:"
```

```
print(all_films_character)
```

```
[1] "R2-D2"
```

Exercise 2 - Expectation & Variance

Below is a plot of the probability mass function for a variable X which is distributed according to the binomial distribution with 10 Bernoulli trials each with probability of success of 2/3. Or, in short, $X \sim B(n = 20, p = 2/3)$.



The code chunk below creates a tibble with each of the values of X and their associated probabilities. Use the `mutate` and `summarise` functions from the `dplyr` package to calculate the expected value, $E[X]$, and variance, $Var(X)$ of the random variable X.

```
# This code creates the data set with outcomes & probabilities
binom_dist_n10_p67 <- tibble(x=0:20,x_prob=dbinom(x,size=20,prob=2/3))

# Put your code to calculate expected value below this comment
expected_value <- 20 * (2/3)
print(paste("Expected Value:", expected_value))
```

```
[1] "Expected Value: 13.3333333333333"
```

```
# Put your code to calculate variance below this comment
variance <- 20 * (2/3) * (1 - (2/3))
print(paste("Variance:", variance))
```

```
[1] "Variance: 4.44444444444444"
```

Tip: Expected Value & Variance of Discrete Variables

Remember, expected value of a discretely distributed random variable is calculated as:

$$E[X] = \sum_{i \in \Omega} x_i \cdot Pr(X = x_i)$$

And likewise, variance of a discretely distributed random variable is calculated as:

$$\begin{aligned} Var(X) &= E[X^2] - E[X]^2 \\ &= \sum_{i \in \Omega} x_i^2 \cdot Pr(X = x_i) - \left[\sum_{i \in \Omega} x_i \cdot Pr(X = x_i) \right]^2 \end{aligned}$$

Exercise 3 - Conditional Expectations & the LLN

Calculating conditional expectations can sometimes be trickier with continuously distributed random variables, especially if you haven't taken calculus (and even when you have). It turns out though, that with a computer we can actually approximate quite well some numbers that might otherwise require higher level math to determine.

In this question, we're going to use `rnorm`, `filter`, and `summarise` to approximate an answer to a question that would otherwise require calculus.

What we want to approximate is the answer to the question, what is the expected value of X given that $X < 0$ and is otherwise distributed like *standard normal* random variable? In other words, what is $E[X \mid X < 0]$ when $X \sim \mathcal{N}(\mu = 0, \sigma^2 = 1)$? This is difficult to calculate even with calculus skills because X's unconditional expected value is calculated as:

$$E[X] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} x \cdot e^{-\frac{x^2}{2}} \, dx$$

which doesn't have a simple integration strategy to solve. In any case, it turns out that when X is standard normally distributed, then $E[X \mid X < 0] = -2/\sqrt{2\pi} \approx -0.7979$. In the code chunk below, use the law of large numbers (LLN) to approximate this value.

```
# Setting the psuedo-RNG seed will lead to consistent results.
set.seed(1234)

# Start by creating a tibble with a variable called x, and use
# rnorm to generate a lot (maybe a million) random draws of x.
data <- tibble(x = rnorm(1e6))
```

```r
# filter this tibble to keep only negative numbers
negative_data <- data %>%
  filter(x < 0)

# calculate the sample mean (i.e. expected value)
negative_mean <- mean(negative_data$x)
```

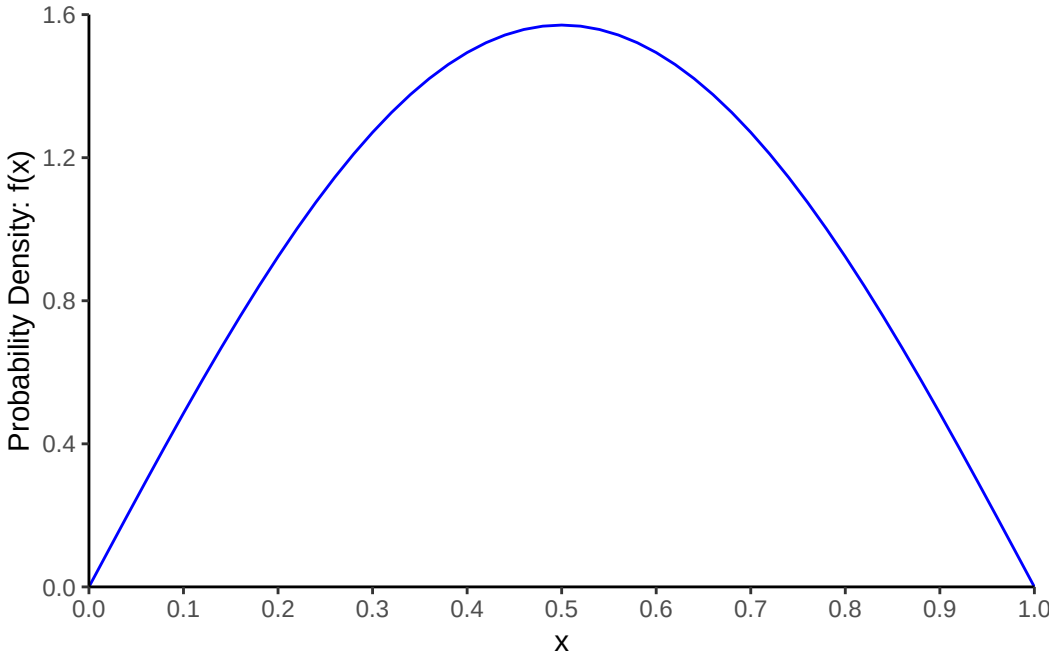
Exercise 4 - Central Limit Theorem (CLT)

As I mentioned in class, the CLT is an incredibly powerful result related to sampling means. It holds in case where the uncertainty modeled by the distribution of X has the following properties:

- Every X in the sample is drawn independently and from the same distribution, so that the expected value, μ, and variance, σ^2, are always the same – i.e. $X \sim iid(\mu, \sigma^2)$.
- The distribution of X has an expected value and finite variance – only a small class of distributions (e.g. Cauchy) lack these properties.

This means the CLT can apply in a vast array of different situations involving uncertainty. It could even apply to made-up distribution as long as that distribution and the draws from it obey the two points above.

Suppose that we say that X is distributed according to the "sine distribution", whose PDF is $f(x) = (\pi/2)\sin(\pi x)$ with support over the interval from 0 to 1. A plot of the PDF of our sine distribution is shown below.



If we integrate our PDF over the support of the distribution, it does actually turn out to be 1.

$$\int_{x\in\Omega} f(x)\ dx = \int_0^1 \frac{\pi}{2}\sin(\pi x)\ dx = 1$$

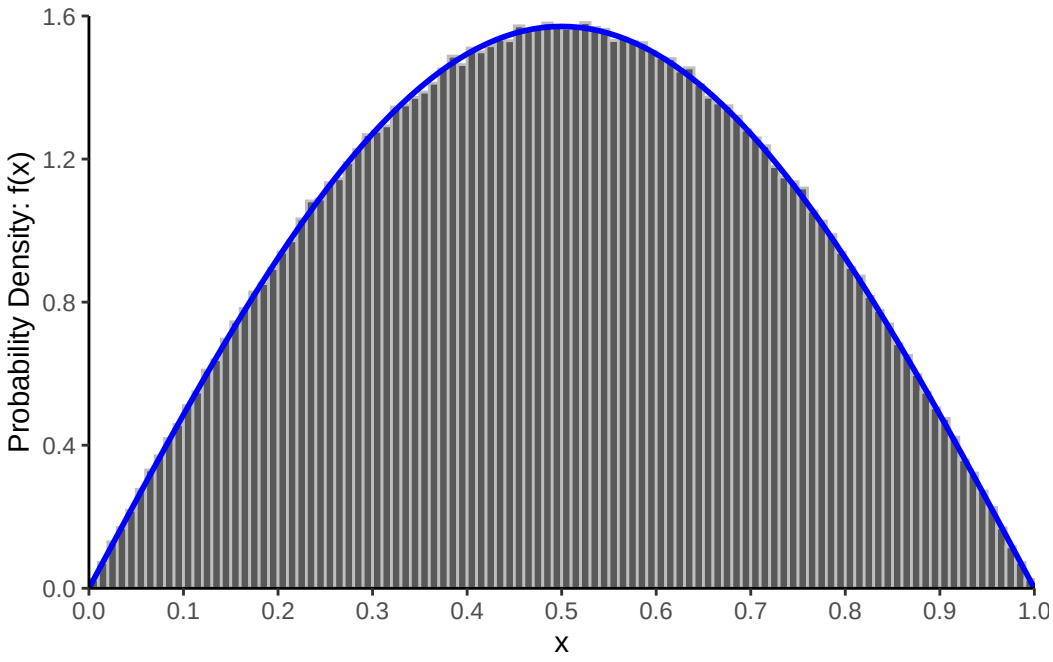
This is one of our requirements of a function in order for it to be a probability function. The other two requirements can be shown with more math, but that isn't the point of this assignment so we'll leave it be for now.

Now we will apply the CLT to our made up distribution[1], starting by making a bunch of *iid* random draws. The code below generates a bunch of random draws from our sine distribution, puts them in a table, and visualizes them to demonstrate that they are actually distributed according to the distribution we expect.

```
# Use u-substitution to generate sine distributed RVs
rv_sine <- acos(1-2*runif(n = 1e6,min = 0,max = 1))/pi

# Create a table w/RVs and assign them to 10,000 groups of 100
rv_sine_tbl <- tibble(x=rv_sine,grp=rep(1:1e4,each=100))
```

[1]The expected value and variance of our sine distribution exist and are finite. $E[X] = 0.5$ and $Var(X) = 0.25 - 2/\pi^2$.

```r
# Visualize our random draws to ensure they are properly distributed
rv_sine_tbl |>
  ggplot() +
  geom_histogram(
    aes(x=x,y=after_stat(density)),
    binwidth = 0.01,boundary=0,color="grey") +
  stat_function(
    fun=~(pi/2)*sin(pi*.),
    xlim=c(0,1),color="blue",linewidth=1) +
  scale_x_continuous(limits=c(0,1),expand = expansion(),breaks=seq(0,1,0.1)) +
  scale_y_continuous(limits=c(0,1.6),expand=expansion(mult = 0)) +
  labs(y="Probability Density: f(x)") +
  theme_classic()
```



Our random sample clearly follows our made up sine distribution. Now, we can work towards checking that the CLT applies in this instance, as it does in so many others. In order for the normal distribution to emerge, we first need to calculate sample means. In the code chunk below, use `group_by`, `summarise`, and `mean` to calculate the sample means.

```r
# Calculate the sample means and assign the results to a tibble called
# rv_means. In dplyr::summarise, call your summary variable "sample_mean"
```

```
# rv_means <- rv_sine_tbl |>
rv_means <- rv_sine_tbl %>%
  summarise(sample_mean = mean(x))
```

The next step involves centering and rescaling the sample means that we calculated above. We do this according to the formula below:

$$\bar{X}^* = \sqrt{n}\left(\frac{\bar{X} - \mu}{\sigma}\right)$$

where $\bar{X}$ is our sample mean, μ is the expected value of our original distribution, σ is the standard deviation of our original distribution, and n is the size of the sample for which we're computing the sample mean. In the code chunk below, use `mutate` to calculate the centered and rescaled sample means.

```
# Calculate the centered and rescaled sample means. Call this new
# variable "normed_xbar".
# rv_means <- rv_means |>
rv_means <- rv_means %>%
  mutate(
    normed_xbar = (sample_mean - mean(sample_mean)) / sd(sample_mean)
  )
```

Now that you've "normalized" the sample means, they will be distributed according to the standard normal distribution instead of our original and made up sine distribution. To get the code chunk to run change `#| eval:` argument from `false` to `true`.

```
rv_means |>
  ggplot() +
  geom_histogram(
    aes(x = normed_xbar, y = after_stat(density)),
    binwidth = 0.1, color = "grey"
  ) +
  stat_function(
    fun = dnorm, xlim = c(-4, 4),
    color = "blue", linewidth = 1
  ) +
  annotate(
    geom = "text", x = 1.5, y = 0.3,
    label = "Standard Normal\nPDF", color = "blue"
  ) +
```

```
  labs(
    x = expression(Normalized~bar(X)),
    y = "Density"
  ) +
  theme_classic()
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_bin()`).
```