Data Visualization

1.1 Data Types

- **Categorical data** are variables that are defined by a small number of groups.
    - Ordinal categorical data have an inherent order to the categories (mild/medium/hot, for example).
    - Non-ordinal categorical data have no order to the categories.
- **Numerical data** take a variety of numeric values.
    - Continuous variables can take any value.
    - Discrete variables are limited to sets of specific values.

1.2 Distribution

- A distribution is a function or description that shows the possible values of a variable and how often those values occur.
- For categorical variables, the distribution describes the proportions of each category.
- A *frequency table* is the simplest way to show a categorical distribution.
  Use `prop.table` to convert a table of counts to a frequency table. *Barplots* display the distribution of categorical variables and are a way to visualize the information in frequency tables.
- For continuous numerical data, reporting the frequency of each unique entry is not an effective summary as many or most values are unique. Instead, a distribution function is required.
- The *cumulative distribution function (CDF)* is a function that reports the proportion of data below a value a for all values of a: $F(a)=Pr(x \leq a)$.
- The proportion of observations between any two values a and b can be computed from the CDF as $F(b)-F(a)$.
- A *histogram* divides data into non-overlapping bins of the same size and plots the counts of number of values that fall in that interval.

1.3 Smooth Density Plots

- *Smooth density plots* can be thought of as histograms where the bin width is extremely or infinitely small. The smoothing function makes estimates of the true continuous trend of the data given the available sample of data points.
- The degree of smoothness can be controlled by an argument in the plotting function.
- While the histogram is an assumption-free summary, the smooth density plot is shaped by assumptions and choices you make as a data analyst.
- The y-axis is scaled so that the area under the density curve sums to 1. This means that interpreting values on the y-axis is not straightforward. To determine the proportion of data in between two values, compute the area under the smooth density curve in the region between those values.
- An advantage of smooth densities over histograms is that densities are easier to compare visually.

  A further note on histograms: note that the choice of binwidth has a determinative effect on shape. There is no "true" choice for binwidth, and you can sometimes gain insights into the data by experimenting with binwidths.

1.4 Normal Distribution

- The normal distribution:
- Is centered around one value, the *mean*
- Is symmetric around the mean
- Is defined completely by its mean (μ) and standard deviation (σ)
- Always has the same proportion of observations within a given distance of the mean (for example, 95% within 2 σ)
- The standard deviation is the average distance between a value and the mean value.
- Calculate the mean using the `mean` function.
- Calculate the standard deviation using the `sd` function or manually.
- Standard units describe how many standard deviations a value is away from the mean. The z-score, or number of standard deviations an observation x is away from the mean μ:

$$Z = \frac{x-\mu}{\sigma}$$

- Compute standard units with the `scale` function.
- **Important**: to calculate the proportion of values that meet a certain condition, use the `mean` function on a logical vector. Because `TRUE` is converted to 1 and `FALSE` is converted to 0, taking the mean of this vector yields the proportion of `TRUE`.

Equation for the normal distribution

The normal distribution is mathematically defined by the following formula for any mean μ and standard deviation σ:

$$\Pr\left(a < x < b\right) = \int_a^b \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}\,dx$$

Code

```
# define x as vector of male heights
library(tidyverse)
library(dslabs)
data(heights)
index <- heights$sex=="Male"
x <- heights$height[index]

# calculate the mean and standard deviation manually
average <- sum(x)/length(x)
SD <- sqrt(sum(x - average)^2))/length(x)

# built-in mean and sd functions - note that the audio and
printed values disagree
average <- mean(x)
SD <- sd(x)
c(average = average, SD = SD)

# calculate standard units
z <- scale(x)

# calculate proportion of values within 2 SD of mean
mean(abs(z) < 2)
```

1.5 Quantile-Quantile Plots

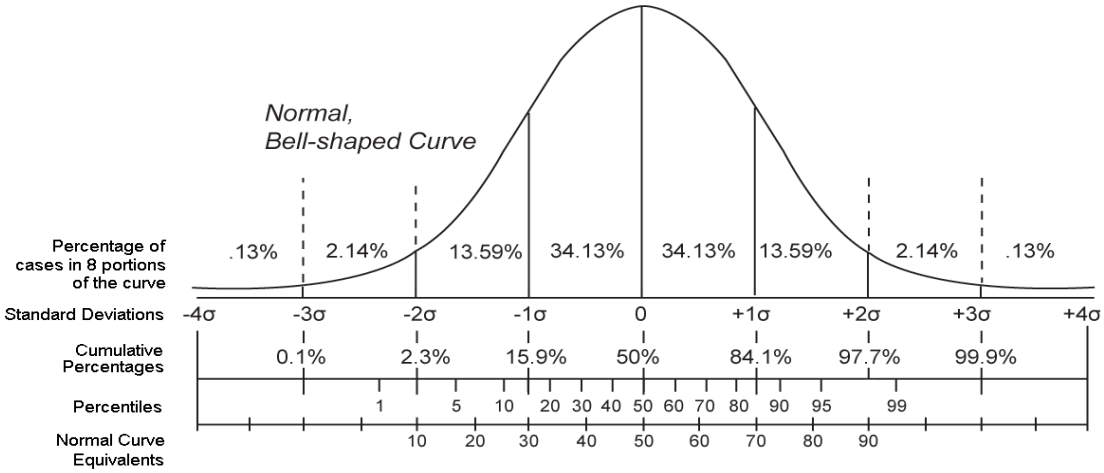
- Quantile-quantile plots, or QQ-plots, are used to check whether distributions are well-approximated by a normal distribution.
- Given a proportion p, the quantile q is the value such that the proportion of values in the data below q is p.
- In a QQ-plot, the sample quantiles in the observed data are compared to the theoretical quantiles expected from the normal distribution. If the data are well-approximated by the normal distribution, then the points on the QQ-plot will fall near the identity line (sample = theoretical).
- Calculate sample quantiles (observed quantiles) using the `quantile` function.
- Calculate theoretical quantiles with the `qnorm` function. `qnorm` will calculate quantiles for the standard normal distribution ($\mu=0,\sigma=1$) by default, but it can calculate quantiles for any normal distribution given `mean` and `sd` arguments. We will learn more about `qnorm` in the probability course.
- Note that we will learn alternate ways to make QQ-plots with less code later in the series.

Code

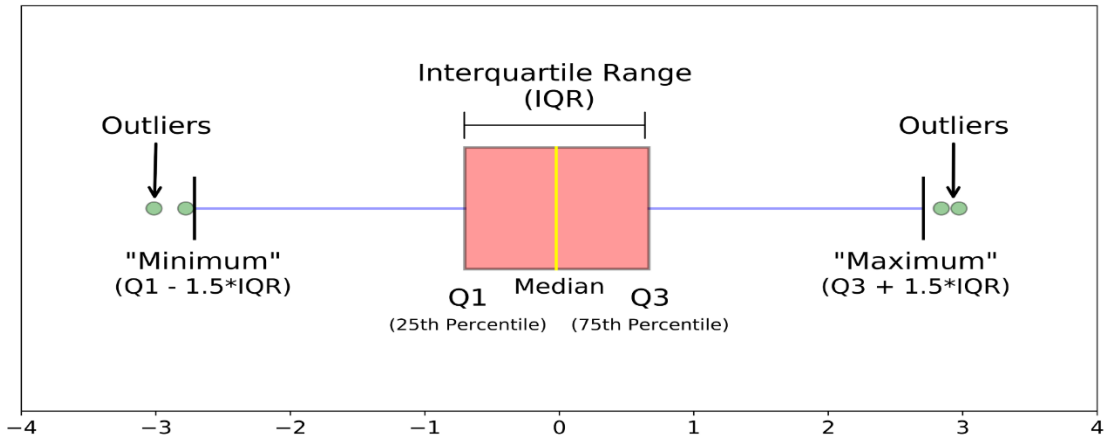
```
# define x and z
library(tidyverse)
library(dslabs)
data(heights)
index <- heights$sex=="Male"
x <- heights$height[index]
z <- scale(x)

# proportion of data below 69.5
mean(x <= 69.5)

# calculate observed and theoretical quantiles
p <- seq(0.05, 0.95, 0.05)
observed_quantiles <- quantile(x, p)
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))

# make QQ-plot
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)

# make QQ-plot with scaled values
observed_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
plot(theoretical_quantiles, observed_quantiles)
abline(0,1)
```

1.6 Percentiles and Boxplots

- *Percentiles* are the quantiles obtained when defining p as $0.01, 0.02, \ldots, 0.99$. They summarize the values at which a certain percent of the observations are equal to or less than that value.
- The 50th percentile is also known as the *median*.
- The *quartiles* are the 25th, 50th and 75th percentiles.



- When data do not follow a normal distribution and cannot be succinctly summarized by only the mean and standard deviation, an alternative is to report a five-number summary: range (ignoring outliers) and the quartiles (25th, 50th, 75th percentile).
- In a *boxplot*, the box is defined by the 25th and 75th percentiles and the median is a horizontal line through the box. The whiskers show the range excluding outliers, and outliers are plotted separately as individual points.
- The *interquartile* range is the distance between the 25th and 75th percentiles.
- Boxplots are particularly useful when comparing multiple distributions.



2. ggplot

2.1 ggplot2

- ggplot2 is part of the tidyverse, which you can load with `library(tidyverse)`.
- Note that you can also load ggplot2 alone using the command `library(ggplot2)`, instead of loading the entire tidyverse.
- ggplot2 uses a *grammar of graphics* to break plots into building blocks that have intuitive syntax, making it easy to create relatively complex and aesthetically pleasing plots with relatively simple and readable code.
- ggplot2 is designed to work exclusively with tidy data (rows are observations and columns are variables).

2.2 Graph Components

- Plots in ggplot2 consist of 3 main components:
    - Data: The dataset being summarized
    - Geometry: The type of plot (scatterplot, boxplot, barplot, histogram, qqplot, smooth density, etc.)
    - Aesthetic mapping: Variables mapped to visual cues, such as x-axis and y-axis values and color
- There are additional components:
    - Scale
    - Labels, Title, Legend
    - Theme/Style

2.3 Creating a New Plot

- You can associate a dataset `x` with a ggplot object with any of the 3 commands:
  - `ggplot(data = x)`
  - `ggplot(x)`
  - `x %>% ggplot()`
- You can assign a ggplot object to a variable. If the object is not assigned to a variable, it will automatically be displayed.
- You can display a ggplot object assigned to a variable by printing that variable.

Code

```
library(tidyverse)
library(dslabs)
data(murders)

ggplot(data = murders)

murders %>% ggplot

p <- ggplot(data = murders)
class(p)
print(p)    # this is equivalent to simply typing p
```

2.4 Layers

- In ggplot2, graphs are created by adding *layers* to the `ggplot` object:
  DATA %>% ggplot() + LAYER_1 + LAYER_2 + ... + LAYER_N
- The *geometry layer* defines the plot type and takes the format `geom_X` where `X` is the plot type.
- *Aesthetic mappings* describe how properties of the data connect with features of the graph (axis position, color, size, etc.) Define aesthetic mappings with the `aes` function.
- `aes` uses variable names from the object component (for example, `total` rather than `murders$total`).
- `geom_point` creates a scatterplot and requires `x` and `y` aesthetic mappings.

- `geom_text` and `geom_label` add text to a scatterplot and require `x`, `y`, and `label` aesthetic mappings.
- To determine which aesthetic mappings are required for a geometry, read the help file for that geometry.
- You can add layers with different aesthetic mappings to the same graph.

Code: Adding layers to a plot

```
library(tidyverse)
library(dslabs)
data(murders)

murders %>% ggplot() +
    geom_point(aes(x = population/10^6, y = total))

# add points layer to predefined ggplot object
p <- ggplot(data = murders)
p + geom_point(aes(population/10^6, total))

# add text layer to scatterplot
p + geom_point(aes(population/10^6, total)) +
    geom_text(aes(population/10^6, total, label = abb))
```

Code: Example of `aes` behavior

```
# no error from this call
p_test <- p + geom_text(aes(population/10^6, total, label =
abb))

# error - "abb" is not a globally defined variable and cannot
be found outside of aes
p_test <- p + geom_text(aes(population/10^6, total), label =
abb)
```

2.5 Tinkering

- You can modify arguments to geometry functions other than `aes` and the data. Additional arguments can be found in the documentation for each geometry.
- These arguments are not aesthetic mappings: they affect all data points the same way.
- *Global aesthetic mappings* apply to all geometries and can be defined when you initially call `ggplot`. All the geometries added as layers will default to this mapping. Local aesthetic mappings add additional information or override the default mappings.

Code

```
# change the size of the points
p + geom_point(aes(population/10^6, total), size = 3) +
    geom_text(aes(population/10^6, total, label = abb))

# move text labels slightly to the right
p + geom_point(aes(population/10^6, total), size = 3) +
    geom_text(aes(population/10^6, total, label = abb),
nudge_x = 1)

# simplify code by adding global aesthetic
p <- murders %>% ggplot(aes(population/10^6, total, label =
abb))
p + geom_point(size = 3) +
    geom_text(nudge_x = 1.5)

# local aesthetics override global aesthetics
p + geom_point(size = 3) +
    geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```

2.6 Scales, Labels, and Colors

- Convert the x-axis to log scale with `scale_x_continuous(trans = "log10")` or `scale_x_log10`. Similar functions exist for the y-axis.
- Add axis titles with `xlab` and `ylab` functions. Add a plot title with the `ggtitle` function.
- Add a color mapping that colors points by a variable by defining the `col` argument within `aes`. To color all points the same way, define `col`outside of `aes`.

- Add a line with the `geom_abline` geometry. `geom_abline` takes arguments `slope` (default = 1) and `intercept` (default = 0). Change the color with `col` or `color` and line type with `lty`.
- Placing the line layer after the point layer will overlay the line on top of the points. To overlay points on the line, place the line layer before the point layer.
- There are many additional ways to tweak your graph that can be found in the ggplot2 documentation, cheat sheet, or on the internet. For example, you can change the legend title with `scale_color_discrete`.

Code: Log-scale the x- and y-axis

```
# define p
library(tidyverse)
library(dslabs)
data(murders)
p <- murders %>% ggplot(aes(population/10^6, total, label =
abb))

# log base 10 scale the x-axis and y-axis
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.05) +
    scale_x_continuous(trans = "log10") +
    scale_y_continuous(trans = "log10")

# efficient log scaling of the axes
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.075) +
    scale_x_log10() +
    scale_y_log10()
```

Code: Add labels and title

```
p + geom_point(size = 3) +
    geom_text(nudge_x = 0.075) +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Population in millions (log scale)") +
    ylab("Total number of murders (log scale)") +
    ggtitle("US Gun Murders in 2010")
```

Code: Change color of the points

```
# redefine p to be everything except the points layer
p <- murders %>%
```

```
    ggplot(aes(population/10^6, total, label = abb)) +
    geom_text(nudge_x = 0.075) +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Population in millions (log scale)") +
    ylab("Total number of murders (log scale)") +
    ggtitle("US Gun Murders in 2010")

# make all points blue
p + geom_point(size = 3, color = "blue")

# color points by region
p + geom_point(aes(col = region), size = 3)
```

Code: Add a line with average murder rate

```
# define average murder rate
r <- murders %>%
    summarize(rate = sum(total) / sum(population) * 10^6) %>%
    pull(rate)

# basic line with average murder rate for the country
p + geom_point(aes(col = region), size = 3) +
    geom_abline(intercept = log10(r))    # slope is default of
1

# change line to dashed and dark grey, line under points
p +
    geom_abline(intercept = log10(r), lty = 2, color =
"darkgrey") +
    geom_point(aes(col = region), size = 3)
```

Code: Change legend title

```
p <- p + scale_color_discrete(name = "Region")    # capitalize
legend title
```

2.7 Add-On Packages

- The style of a ggplot graph can be changed using the `theme` function.
- The **ggthemes** package adds additional themes.

- The **ggrepel** package includes a geometry that repels text labels, ensuring they do not overlap with each other: `geom_text_repel`.

  Code: Adding themes

```
# theme used for graphs in the textbook and course
library(dslabs)
ds_theme_set()

# themes from ggthemes
library(ggthemes)
p + theme_economist()    # style of the Economist magazine
p + theme_fivethirtyeight()    # style of the FiveThirtyEight
website
```

  Code: Putting it all together to assemble the plot

```
# load libraries
library(tidyverse)
library(ggrepel)
library(ggthemes)
library(dslabs)
data(murders)

# define the intercept
r <- murders %>%
    summarize(rate = sum(total) / sum(population) * 10^6) %>%
    .$rate

# make the plot, combining all elements
murders %>%
    ggplot(aes(population/10^6, total, label = abb)) +
    geom_abline(intercept = log10(r), lty = 2, color =
"darkgrey") +
    geom_point(aes(col = region), size = 3) +
    geom_text_repel() +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Population in millions (log scale)") +
    ylab("Total number of murders (log scale)") +
    ggtitle("US Gun Murders in 2010") +
    scale_color_discrete(name = "Region") +
    theme_economist()
```

2.8 Other Examples

- `geom_histogram` creates a histogram. Use the `binwidth` argument to change the width of bins, the `fill` argument to change the bar fill color, and the `col` argument to change bar outline color.
- `geom_density` creates smooth density plots. Change the fill color of the plot with the `fill` argument.
- `geom_qq` creates a quantile-quantile plot. This geometry requires the `sample` argument. By default, the data are compared to a standard normal distribution with a mean of 0 and standard deviation of 1. This can be changed with the `dparams` argument, or the sample data can be scaled.
- Plots can be arranged adjacent to each other using the grid.arrange function from the gridExtra package. First, create the plots and save them to objects (p1, p2, ...). Then pass the plot objects to grid.arrange.

Code: Histograms in ggplot2

```
# load heights data
library(tidyverse)
library(dslabs)
data(heights)

# define p
p <- heights %>%
    filter(sex == "Male") %>%
    ggplot(aes(x = height))

# basic histograms
p + geom_histogram()
p + geom_histogram(binwidth = 1)

# histogram with blue fill, black outline, labels and title
p + geom_histogram(binwidth = 1, fill = "blue", col = "black")
+
    xlab("Male heights in inches") +
    ggtitle("Histogram")
```

Code: Smooth density plots in ggplot2

```
p + geom_density()
p + geom_density(fill = "blue")
```

Code: Quantile-quantile plots in ggplot2

```
# basic QQ-plot
p <- heights %>% filter(sex == "Male") %>%
    ggplot(aes(sample = height))
p + geom_qq()

# QQ-plot against a normal distribution with same mean/sd as
data
params <- heights %>%
    filter(sex == "Male") %>%
    summarize(mean = mean(height), sd = sd(height))
p + geom_qq(dparams = params) +
    geom_abline()

# QQ-plot of scaled data against the standard normal
distribution
heights %>%
    ggplot(aes(sample = scale(height)) +
    geom_qq() +
    geom_abline()
```

Code: Grids of plots with the grid.extra package

```
# define plots p1, p2, p3
p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x =
height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue", col =
"black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue", col =
"black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue", col =
"black")

# arrange plots next to each other in 1 row, 3 columns
library(gridExtra)
grid.arrange(p1, p2, p3, ncol = 3)
```

3.0 dplyr

3.1 Intro to dplr

- `summarize` from the dplyr/tidyverse package computes summary statistics from the data frame. It returns a data frame whose column names are defined within the function call.

- `summarize` can compute any summary function that operates on vectors and returns a single value, but it cannot operate on functions that return multiple values.
- Like most dplyr functions, `summarize` is aware of variable names within data frames and can use them directly.

Code

```
library(tidyverse)
library(dslabs)
data(heights)

# compute average and standard deviation for males
s <- heights %>%
    filter(sex == "Male") %>%
    summarize(average = mean(height), standard_deviation =
sd(height))

# access average and standard deviation from summary table
s$average
s$standard_deviation

# compute median, min and max
heights %>%
    filter(sex == "Male") %>%
    summarize(median = median(height),
                        minimum = min(height),
                        maximum = max(height))

# alternative way to get min, median, max in base R
quantile(heights$height, c(0, 0.5, 1))

# generates an error: summarize can only take functions that
return a single value
heights %>%
    filter(sex == "Male") %>%
    summarize(range = quantile(height, c(0, 0.5, 1)))
```

3.2 The Dot Placeholder

- The dot operator allows you to access values stored in data that is being piped in using the `%>%` character. The dot is a placeholder for the data being passed in through the pipe.
- The dot operator allows dplyr functions to return single vectors or numbers instead of only data frames.
- `us_murder_rate %>% .$rate` is equivalent to `us_murder_rate$rate`.
- Note that an equivalent way to extract a single column using the pipe is `us_murder_rate %>% pull(rate).` The `pull` function will be used in later course material.

Code

```
library(tidyverse)
library(dslabs)
data(murders)

murders <- murders %>% mutate(murder_rate =
total/population*100000)
summarize(murders, mean(murder_rate))

# calculate US murder rate, generating a data frame
us_murder_rate <- murders %>%
    summarize(rate = sum(total) / sum(population) * 100000)
us_murder_rate

# extract the numeric US murder rate with the dot operator
us_murder_rate %>% .$rate

# calculate and extract the murder rate with one pipe
us_murder_rate <- murders %>%
    summarize(rate = sum(total) / sum(population * 100000) %>%
    .$rate
```

3.3 Group By

- he `group_by` function from **dplyr** converts a data frame to a grouped data frame, creating groups using one or more variables.
- `summarize` and some other **dplyr** functions will behave differently on grouped data frames.

- Using `summarize` on a grouped data frame computes the summary statistics for each of the separate groups.

  Code

```
# libraries and data
library(tidyverse)
library(dslabs)
data(heights)
data(murders)

# compute separate average and standard deviation for
male/female heights
heights %>%
    group_by(sex) %>%
    summarize(average = mean(height), standard_deviation =
sd(height))

# compute median murder rate in 4 regions of country
murders <- murders %>%
    mutate(murder_rate = total/population * 100000)
murders %>%
    group_by(region) %>%
    summarize(median_rate = median(murder_rate))
```

3.4 Sorting Data Tables

- The `arrange` function from **dplyr** sorts a data frame by a given column.
- By default, `arrange` sorts in ascending order (lowest to highest). To instead sort in descending order, use the function `desc` inside of `arrange`.
- You can `arrange` by multiple levels: within equivalent values of the first level, observations are sorted by the second level, and so on.
- The `top_n` function shows the top results ranked by a given variable, but the results are not ordered. You can combine `top_n` with `arrange` to return the top results in order.

  Code

```
# libraries and data
library(tidyverse)
library(dslabs)
data(murders)
```

```
# set up murders object
murders <- murders %>%
    mutate(murder_rate <- total/population * 100000)

# arrange by population column, smallest to largest
murders %>% arrange(population) %>% head()

# arrange by murder rate, smallest to largest
murders %>% arrange(murder_rate) %>% head()

# arrange by murder rate in descending order
murders %>% arrange(desc(murder_rate)) %>% head()

# arrange by region alphabetically, then by murder rate within
each region
murders %>% arrange(region, murder_rate) %>% head()

# show the top 10 states with highest murder rate, not ordered
by rate
murders %>% top_n(10, murder_rate)

# show the top 10 states with highest murder rate, ordered by
rate
murders %>% arrange(desc(murder_rate)) %>% top_n(10)
```

3.5 Faceting (Side-by-side Plots)

- Faceting makes multiple side-by-side plots stratified by some variable. This is a way to ease comparisons.
- The `facet_grid` function allows faceting by up to two variables, with rows faceted by one variable and columns faceted by the other variable. To facet by only one variable, use the dot operator as the other variable.
- The `facet_wrap` function facets by one variable and automatically wraps the series of plots so they have readable dimensions.
- Faceting keeps the axes fixed across all plots, easing comparisons between plots.
- The data suggest that the developing versus Western world view no longer makes sense in 2012.

Code

```
# facet by continent and year
filter(gapminder, year %in% c(1962, 2012)) %>%
    ggplot(aes(fertility, life_expectancy, col = continent)) +
    geom_point() +
    facet_grid(continent ~ year)

# facet by year only
filter(gapminder, year %in% c(1962, 2012)) %>%
    ggplot(aes(fertility, life_expectancy, col = continent)) +
    geom_point() +
    facet_grid(. ~ year)

# facet by year, plots wrapped onto multiple rows
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
    filter(year %in% years & continent %in% continents) %>%
    ggplot(aes(fertility, life_expectancy, col = continent)) +
    geom_point() +
    facet_wrap(~year)
```

3.6 Time Series Plots

- Time series plots have time on the x-axis and a variable of interest on the y-axis.
- The `geom_line` geometry connects adjacent data points to form a continuous line. A line plot is appropriate when points are regularly spaced, densely packed and from a single data series.
- You can plot multiple lines on the same graph. Remember to group or color by a variable so that the lines are plotted independently.
- Labeling is usually preferred over legends. However, legends are easier to make and appear by default. Add a label with geom_text, specifying the coordinates where the label should appear on the graph.

Code: Single time series

```
# scatterplot of US fertility by year
gapminder %>%
    filter(country == "United States") %>%
```

```
    ggplot(aes(year, fertility)) +
    geom_point()

# line plot of US fertility by year
gapminder %>%
    filter(country == "United States") %>%
    ggplot(aes(year, fertility)) +
    geom_line()
```

Code: Multiple time series

```
# line plot fertility time series for two countries- only one
line (incorrect)
countries <- c("South Korea", "Germany")
gapminder %>% filter(country %in% countries) %>%
    ggplot(aes(year, fertility)) +
    geom_line()

# line plot fertility time series for two countries - one line
per country
gapminder %>% filter(country %in% countries) %>%
    ggplot(aes(year, fertility, group = country)) +
    geom_line()

# fertility time series for two countries - lines colored by
country
gapminder %>% filter(country %in% countries) %>%
    ggplot(aes(year, fertility, col = country)) +
    geom_line()
```

Code: Adding text labels to a plot

```
# life expectancy time series - lines colored by country and
labeled, no legend
labels <- data.frame(country = countries, x = c(1975, 1965), y
= c(60, 72))
gapminder %>% filter(country %in% countries) %>%
    ggplot(aes(year, life_expectancy, col = country)) +
    geom_line() +
    geom_text(data = labels, aes(x, y, label = country), size
= 5) +
    theme(legend.position = "none")
```

3.7 Transformations

- We use GDP data to compute income in US dollars per day, adjusted for inflation.
- Log transformations convert multiplicative changes into additive changes.
- Common transformations are the log base 2 transformation and the log base 10 transformation. The choice of base depends on the range of the data. The natural log is not recommended for visualization because it is difficult to interpret.
- The mode of a distribution is the value with the highest frequency. The mode of a normal distribution is the average. A distribution can have multiple local modes.
- There are two ways to use log transformations in plots: transform the data before plotting or transform the axes of the plot. Log scales have the advantage of showing the original values as axis labels, while log transformed values ease interpretation of intermediate values between labels.
- Scale the x-axis using `scale_x_continuous` or `scale_x_log10` layers in ggplot2. Similar functions exist for the y-axis.
- In 1970, income distribution is bimodal, consistent with the dichotomous Western versus developing worldview.

Code

```
# add dollars per day variable
gapminder <- gapminder %>%
    mutate(dollars_per_day = gdp/population/365)

# histogram of dollars per day
past_year <- 1970
gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    ggplot(aes(dollars_per_day)) +
    geom_histogram(binwidth = 1, color = "black"))

# repeat histogram with log2 scaled data
gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    ggplot(aes(log2(dollars_per_day))) +
    geom_histogram(binwidth = 1, color = "black"))

# repeat histogram with log2 scaled x-axis
gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    ggplot(aes(dollars_per_day)) +
```

```
    geom_histogram(binwidth = 1, color = "black")) +
    scale_x_continuous(trans = "log2")
```

3.8 Stratify and Boxplot

- Make boxplots stratified by a categorical variable using the `geom_boxplot` geometry.
- Rotate axis labels by changing the theme through `element_text`. You can change the angle and justification of the text labels.
- Consider ordering your factors by a meaningful value with the `reorder` function, which changes the order of factor levels based on a related numeric vector. This is a way to ease comparisons.
- Show the data by adding data points to the boxplot with a `geom_point` layer. This adds information beyond the five-number summary to your plot, but too many data points it can obfuscate your message.

Code: Boxplot of GDP by region

```
# add dollars per day variable
gapminder <- gapminder %>%
    mutate(dollars_per_day = gdp/population/365)

# number of regions
length(levels(gapminder$region))

# boxplot of GDP by region in 1970
past_year <- 1970
p <- gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    ggplot(aes(region, dollars_per_day))
p + geom_boxplot()

# rotate names on x-axis
p + geom_boxplot() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

Code: The reorder function

```
# by default, factor order is alphabetical
fac <- factor(c("Asia", "Asia", "West", "West", "West"))
levels(fac)
```

```
# reorder factor by the category means
value <- c(10, 11, 12, 6, 4)
fac <- reorder(fac, value, FUN = mean)
levels(fac)
```

Code: Enhanced boxplot ordered by median income, scaled, and showing data

```
# reorder by median income and color by continent
p <- gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    mutate(region = reorder(region, dollars_per_day, FUN =
median)) %>%    # reorder
    ggplot(aes(region, dollars_per_day, fill = continent))
+    # color by continent
    geom_boxplot() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    xlab("")
p

# log2 scale y-axis
p + scale_y_continuous(trans = "log2")

# add data points
p + scale_y_continuous(trans = "log2") +
geom_point(show.legend = FALSE)
```

3.9 Comparing Distributions

- Use `intersect` to find the overlap between two vectors.
- To make boxplots where grouped variables are adjacaent, color the boxplot by a factor instead of faceting by that factor. This is a way to ease comparisons.
- The data suggest that the income gap between rich and poor countries has narrowed, not expanded.

Code: Histogram of income in West versus developing world, 1970 and 2010

```
# add dollars per day variable and define past year
gapminder <- gapminder %>%
    mutate(dollars_per_day = gdp/population/365)
past_year <- 1970
```

```
# define Western countries
west <- c("Western Europe", "Northern Europe", "Southern
Europe", "Northern America", "Australia and New Zealand")

# facet by West vs devloping
gapminder %>%
    filter(year == past_year & !is.na(gdp)) %>%
    mutate(group = ifelse(region %in% west, "West",
"Developing")) %>%
    ggplot(aes(dollars_per_day)) +
    geom_histogram(binwidth = 1, color = "black") +
    scale_x_continuous(trans = "log2") +
    facet_grid(. ~ group)

# facet by West/developing and year
present_year <- 2010
gapminder %>%
    filter(year %in% c(past_year, present_year) & !is.na(gdp))
%>%
    mutate(group = ifelse(region %in% west, "West",
"Developing")) %>%
    ggplot(aes(dollars_per_day)) +
    geom_histogram(binwidth = 1, color = "black") +
    scale_x_continuous(trans = "log2") +
    facet_grid(year ~ group)
```

Code: Income distribution of West versus developing world, only countries with data

```
# define countries that have data available in both years
country_list_1 <- gapminder %>%
    filter(year == past_year & !is.na(dollars_per_day)) %>%
.$country
country_list_2 <- gapminder %>%
    filter(year == present_year & !is.na(dollars_per_day)) %>%
.$country
country_list <- intersect(country_list_1, country_list_2)

# make histogram including only countries with data available
in both years
gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in%
country_list) %>%    # keep only selected countries
    mutate(group = ifelse(region %in% west, "West",
```

```
"Developing")) %>%
    ggplot(aes(dollars_per_day)) +
    geom_histogram(binwidth = 1, color = "black") +
    scale_x_continuous(trans = "log2") +
    facet_grid(year ~ group)
```

Code: Boxplots of income in West versus developing world, 1970 and 2010

```
p <- gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in%
country_list) %>%
    mutate(region = reorder(region, dollars_per_day, FUN =
median)) %>%
    ggplot() +
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    xlab("") + scale_y_continuous(trans = "log2")
p + geom_boxplot(aes(region, dollars_per_day, fill =
continent)) +
    facet_grid(year ~ .)

# arrange matching boxplots next to each other, colored by
year
p + geom_boxplot(aes(region, dollars_per_day, fill =
factor(year)))
```

3.10 Density Plots

- Change the y-axis of density plots to variable counts using `..count..` as the y argument.
- The `case_when` function defines a factor whose levels are defined by a variety of logical operations to group data.
- Plot stacked density plots using `position="stack"`.
- Define a weight aesthetic mapping to change the relative weights of density plots - for example, this allows weighting of plots by population rather than number of countries.

Code: Faceted smooth density plots

```
# see the code below the previous video for variable
definitions

# smooth density plots - area under each curve adds to 1
```

```
gapminder %>%
    filter(year == past_year & country %in% country_list) %>%
    mutate(group = ifelse(region %in% west, "West",
"Developing")) %>% group_by(group) %>%
    summarize(n = n()) %>% knitr::kable()

# smooth density plots - variable counts on y-axis
p <- gapminder %>%
    filter(year == past_year & country %in% country_list) %>%
    mutate(group = ifelse(region %in% west, "West",
"Developing")) %>%
    ggplot(aes(dollars_per_day, y = ..count.., fill = group))
+
    scale_x_continuous(trans = "log2")
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~
.)
```

Code: Add new region groups with case_when

```
# add group as a factor, grouping regions
gapminder <- gapminder %>%
    mutate(group = case_when(
        .$region %in% west ~ "West",
        .$region %in% c("Eastern Asia", "South-Eastern Asia")
~ "East Asia",
        .$region %in% c("Caribbean", "Central America", "South
America") ~ "Latin America",
        .$continent == "Africa" & .$region != "Northern
Africa" ~ "Sub-Saharan Africa",
        TRUE ~ "Others"))

# reorder factor levels
gapminder <- gapminder %>%
    mutate(group = factor(group, levels = c("Others", "Latin
America", "East Asia", "Sub-Saharan Africa", "West")))
```

Code: Stacked density plot

```
# note you must redefine p with the new gapminder object first
p <- gapminder %>%
  filter(year %in% c(past_year, present_year) & country %in%
country_list) %>%
    ggplot(aes(dollars_per_day, fill = group)) +
    scale_x_continuous(trans = "log2")
```

```
# stacked density plot
p + geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
    facet_grid(year ~ .)
```

Code: Weighted stacked density plot

```
# weighted stacked density plot
gapminder %>%
    filter(year %in% c(past_year, present_year) & country %in%
country_list) %>%
    group_by(year) %>%
    mutate(weight = population/sum(population*2)) %>%
    ungroup() %>%
    ggplot(aes(dollars_per_day, fill = group, weight =
weight)) +
    scale_x_continuous(trans = "log2") +
    geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
facet_grid(year ~ .)
```

3.11 Ecological Fallacy

- The `breaks` argument allows us to set the location of the axis labels and tick marks.
- The *logistic* or *logit transformation* is defined as $f(p)=\log(p/1-p)$, or the log of odds. This scale is useful for highlighting differences near 0 or near 1 and converts fold changes into constant increases.
- The *ecological fallacy* is assuming that conclusions made from the average of a group apply to all members of that group.

Code

```
# define gapminder
library(tidyverse)
library(dslabs)
data(gapminder)

# add additional cases
gapminder <- gapminder %>%
    mutate(group = case_when(
        .$region %in% west ~ "The West",
        .$region %in% "Northern Africa" ~ "Northern Africa",
        .$region %in% c("Eastern Asia", "South-Eastern Asia")
```

```
~ "East Asia",
        .$region == "Southern Asia" ~ "Southern Asia",
        .$region %in% c("Central America", "South America",
"Caribbean") ~ "Latin America",
        .$continent == "Africa" & .$region != "Northern
Africa" ~ "Sub-Saharan Africa",
        .$region %in% c("Melanesia", "Micronesia",
"Polynesia") ~ "Pacific Islands"))

# define a data frame with group average income and average
infant survival rate
surv_income <- gapminder %>%
    filter(year %in% present_year & !is.na(gdp) &
!is.na(infant_mortality) & !is.na(group)) %>%
    group_by(group) %>%
    summarize(income = sum(gdp)/sum(population)/365,
                         infant_survival_rate = 1 -
sum(infant_mortality/1000*population)/sum(population))
surv_income %>% arrange(income)

# plot infant survival versus income, with transformed axes
surv_income %>% ggplot(aes(income, infant_survival_rate, label
= group, color = group)) +
    scale_x_continuous(trans = "log2", limit = c(0.25, 150)) +
    scale_y_continuous(trans = "logit", limit = c(0.875,
.9981),
                                        breaks = c(.85, .90,
.95, .99, .995, .998)) +
    geom_label(size = 3, show.legend = FALSE)
```

4.0 Data Visualization Principles

4.1 Data Visualization Principles, Part 1

Encoding Data Using Visual Cues

- Visual cues for encoding data include position, length, angle, area, brightness and color hue.
- Position and length are the preferred way to display quantities, followed by angles, which are preferred over area. Brightness and color are even harder to quantify but can sometimes be useful.
- Pie charts represent visual cues as both angles and area, while donut charts use only area. Humans are not good at visually quantifying angles and are even worse at quantifying area. Therefore, pie and donut charts should be avoided - use a bar plot instead. If you must make a pie chart, include percentages as labels.
- Bar plots represent visual cues as position and length. Humans are good at visually quantifying linear measures, making bar plots a strong alternative to pie or donut charts.

Know When to Include Zero

- When using bar plots, always start at 0. It is deceptive not to start at 0 because bar plots imply length is proportional to the quantity displayed. Cutting off the y-axis can make differences look bigger than they actually are.
- When using position rather than length, it is not necessary to include 0 (scatterplot, dot plot, boxplot).

Do not Distort Quantities

- Make sure your visualizations encode the correct quantities.
- For example, if you are using a plot that relies on circle area, make sure the area (rather than the radius) is proportional to the quantity.

Order by Meaningful Values

- It is easiest to visually extract information from a plot when categories are ordered by a meaningful value. The exact value on which to order will depend on your data and the message you wish to convey with your plot.
- The default ordering for categories is alphabetical if the categories are strings or by factor level if factors. However, we rarely want alphabetical order.

4.2 Data Visualization Principles, Part 2

Show Data

- A dynamite plot - a bar graph of group averages with error bars denoting standard errors - provides almost no information about a distribution.
- By showing the data, you provide viewers extra information about distributions.
- Jitter is adding a small random shift to each point in order to minimize the number of overlapping points. To add jitter, use the `geom_jitter`geometry instead of `geom_point`. (See example below.)
- Alpha blending is making points somewhat transparent, helping visualize the density of overlapping points. Add an `alpha` argument to the geometry.

Code

```
# dot plot showing the data
heights %>% ggplot(aes(sex, height)) + geom_point()

# jittered, alpha blended point plot
heights %>% ggplot(aes(sex, height)) + geom_jitter(width =
0.1, alpha = 0.2)
```

Ease Comparisons: Use Common Axes

- Ease comparisons by keeping axes the same when comparing data across multiple plots.
- Align plots vertically to see horizontal changes. Align plots horizontally to see vertical changes.
- Bar plots are useful for showing one number but not useful for showing distributions.

Consider Transformations

- Use transformations when warranted to ease visual interpretation.
- The log transformation is useful for data with multiplicative changes. The logistic transformation is useful for fold changes in odds. The square root transformation is useful for count data.
- We learned how to apply transformations earlier in the course.

Ease Comparisons: Compared Visual Cues Should Be Adjacent

- When two groups are to be compared, it is optimal to place them adjacent in the plot.
- Use color to encode groups to be compared.
- Consider using a color blind friendly palette like the one in this video.

Code

```
color_blind_friendly_cols <- c("#999999", "#E69F00",
"#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")

p1 <- data.frame(x = 1:8, y = 1:8, col = as.character(1:8))
%>%
    ggplot(aes(x, y, color = col)) +
    geom_point(size = 5)
p1 + scale_color_manual(values = color_blind_friendly_cols)
```