````
---
title: "nbs03a-03c_teacher-notes"
author: "Jeremy Mikecz"
format: html
editor: visual
---

# \[PRES\]

# 1. Explore dataset

```{r}
library(tidyverse)
```

```{r}
#data()
help(msleep)
```

```{r}
msleep
```

```{r}
class(msleep)
```

```{r}
dim(msleep)
colnames(msleep)
head(msleep)
summary(msleep)
str(msleep)

```

# \[PRES - exercise\]

### 1.3 Exploring Categorical Data

We can use the **distinct()** function to identify a list of unique values in the "genus" column. To apply it to one column of the dataset, let's practice using the piping syntax of tidyverse:

```
dataset_name |>   function(column)
```

```{r}
msleep |>
  distinct(genus)
```

To generate a frequency count of the values found in a column we can use the function **count()**. To sort the results, we can also add the parameter `sort=TRUE`.

```{r}
msleep |>
    count(genus, sort=TRUE)
```

Let's explore a different column: "conservation" (for conservation status of the mammals):
````

```{r}
msleep |>
    count(conservation, sort=TRUE)
```

The dataset did not come with a dictionary defining these abbreviations. However, after a few minutes browsing online I found this:

\[PRES - conservation\]

2. Histograms

Histograms are useful for visualizing the distribution of numerical data for a particular variable.

For this simple plot type, we can use the built-in `hist` function in base R:

```{r}
hist(msleep$sleep_total)
```

Of course, ggplot allows us to create the same thing.

The conventional syntax to plot with ggplot is:

```
ggplot(data=dataset_name, aes(columns assigned to particular visual variables such as x coord, y coord, color fill, color outline, etc.)) +
    geom_name(any_additional_arguments)
```

Thus, to create a histogram, we will want to use **geom_histogram** as our geom_name and set x as the column we want to inspect:

```{r}
ggplot(data=msleep, aes(x = sleep_total)) +
    geom_histogram()   #binwidth=4
```

However, the gglot version doesn't look as nice. Since ggplot allows a lot more customization, we will need to be more specific in our instructions, adding in parameters for the color fill (`fill`), color outline (`color`), and `binwidth`.

```{r}
ggplot(data=msleep, aes(x = sleep_total)) +
    geom_histogram(binwidth=2, color="black", fill="gray")
```

::: callout-tip
Exercise 2.2
:::

3. Bar plots

A histogram is a type of bar plot designed to show the distribution of numerical data by placing this data into "bins" covering a particular range of numerical values.

More conventional bar plots visualize the number of items found in different categories. Since we are interested in counts of categorical data (i.e. the number of carnivores vs. herbivores in the dataset), we will need to pass the argument `stat='count'` into the geom_bar.

```{r}
ggplot(data=msleep, aes(x=vore)) +
    geom_bar(stat="count")
```

```

We can re-order (or re-\*arrange\* using tidyverse terminology) the columns using the **fct_infreq()** function which is imported with tidyverse as part of the **forcats** package. *Note: it still places the NA's on the furthest right, regardless of their frequency.*

Syntax:

```
...aes(x=fct_infreq(column_name)...)
```

```{r}
ggplot(msleep, aes(x=fct_infreq(vore), fill=vore)) +
    geom_bar(stat="count")
```

We can even divide each bar into stacks by assigning the categories in another column to the **fill** parameter. Let's place the `order` column on the x axis and pass in the `conservation` column into the fill parameter.

```{r}
ggplot(msleep, aes(x=order, fill=conservation)) +
    geom_bar(stat="count")
```

\[PRES: bar plots\]

3.2. Relationships - Scatter Plots

This dataset seems to be especially suited to examining relationships in the animal world. More specifically: what types of animals (by genus, order, size, diet) sleep the most / least? What is the relationship between length of sleep cycles and Rem sleep to total daily sleep?

Let's first examine the relationship between size and total sleep. We can add in diet and other variables later.

For a scatter plot we will want to use `geom_point()`.

```{r}
ggplot(msleep, aes(x=sleep_total, y=bodywt)) +
    geom_point()
```

You may notice two extremely heavy animals in this scatter plot. Can you guess what they are?

We can discover what these two animals are by using tidyverse's **filter()** function to filter out all animals less than a given weight in kilograms:

```{r}
msleep |>
  filter(bodywt>1000)

```

Given the way a few heavy animals skews the distribution of animal weights in this dataset, perhaps we should try distributing weights using a logarithmic scale. We can do that by adding:

```{r}
ggplot(msleep, aes(x=sleep_total, y=bodywt)) +
  geom_point() +
```
```

```
  scale_y_continuous(trans="log10")
```

We can also adjust the `size` and transparency (`alpha`) values for our points:

```{r}
ggplot(msleep, aes(x=sleep_total, y=bodywt)) +
  geom_point(size=2, alpha=0.5, color="red") +
  scale_y_continuous(trans="log10")
```

... and add a smoothed trend line using `geom_smooth()`:

```{r}
ggplot(msleep, aes(x=sleep_total, y=bodywt)) +
  geom_point(size=2, alpha=0.5, color="red") +
  scale_y_continuous(trans="log10") +
  geom_smooth()
```

We can remove the confidence interval bar with: `geom_smooth(se=FALSE)`.

```{r}
ggplot(msleep, aes(x=sleep_total, y=bodywt, color=vore)) +
  geom_point(size=2, alpha=0.5) +
  scale_y_continuous(trans="log10") +
  geom_smooth(se=FALSE)
```

\[PRES - exercise\]

4. More Complex Plots for Exploratory Data Visualization

4.1. [Scatterplot Matrix](https://r-graph-gallery.com/199-correlation-matrix-with-ggally.html)

```{r}
#install.packages("GGally")
```

```{r}
library(GGally)
ggpairs(msleep, columns=6:11, ggplot2::aes(color=vore))
```

Notebook 3b - Titanic

Start with hypothesis and summary data like 03a

```
dataset_name |>
    function(column)
```

```{r}
titanic = Titanic |>
    as_tibble()
```

```{r}
titanic |>
    distinct(Class)
```

```{r}
titanic |>
    distinct(Class, Sex)
```

To generate a frequency count of the values found in a column we can use the function **count()**. To sort the results, we can also add the parameter `sort=TRUE`.

```{r}
titanic |>
    count(Class, sort=TRUE)
```

::: callout-tip
Exercise 2.2.

Try applying these same summary functions to a different preloaded dataset to better understand the data it contains.
:::

```{r}
ggplot(data=titanic, aes(x=Sex, y=n)) +
    geom_bar(stat="identity")
```

```{r}
ggplot(data=titanic, aes(x=Age, y=n)) +
    geom_bar(stat="identity")
```

Bar Plots 2: Compare Survival Rates

```{r}
ggplot(titanic, aes(x = Class, y = n, fill = Survived)) +
    geom_bar(stat = "identity", position = "fill") +
    facet_grid(Sex ~ Age)
```

Calculate Survival Rate

At the moment, the dataset is grouped by sex, age (child or adult), passenger class, and survival. To help us identify the passengers most likely to survive, we first need to know how many people were in each demographic group (i.e. sex, age, class with survival excluded) and then calculate the percent of each group that survived.

```{r}
titanic_summary <- titanic |>
    group_by(Sex, Class, Age) |>
    summarize(total = sum(n))
```

```{r}
survivors <- titanic |>
    filter(Survived == "Yes") |>
    group_by(Sex, Age, Class) |>
    summarize(survived = sum(n))
```

```{r}
#titanic_summary <- left_join(titanic_summary, survivors, by = c("Sex", "Age", "Class"))

titanic_summary <- titanic_summary |>
    left_join(survivors, by = c("Sex", "Age", "Class"))
```

```{r}
titanic_summary <- titanic_summary |>
  mutate(survival_rate = survived / total)
```

```{r}
titanic_summary <- titanic_summary |>
    arrange(survival_rate)
```

Mosaic Plots

```{r}
# Calculate survival rates by Class and Sex

survival_rates <- titanic_summary |>
    group_by(Class, Sex) |>
    summarize(survival_rate = mean(survival_rate, na.rm=TRUE))

# Heatmap of survival rates
ggplot(survival_rates, aes(x = Class, y = Sex, fill = survival_rate)) +
    geom_tile()
```

Heatmap of survival rates

ggplot(survival_rates, aes(x = Class, y = Sex, fill = survival_rate)) + geom_tile()

Notebook 3c: Weather

1.2 import data

```{r}
year_avgs <- read.csv("../data/Hanover_temp-yearaverages.csv", row.names=1)

tail(year_avgs)
```

2. Graph Yearly Temperature Data

```{r}
ggplot() +
    geom_line(data=year_avgs |> filter (type=="maxtemp" & Year < 2024), aes(x=Year,
y=T_yravg), color="red") +
    geom_line(data=year_avgs |> filter (type=="mintemp" & Year < 2024), aes(x=Year,
y=T_yravg))
```

2.1 Adjust temperature data (long to wide), add color, and a rolling average

```{r}
year_avgs_wide <- year_avgs |>
    pivot_wider(names_from = type, values_from = T_yravg)
```

```{r}
library(zoo)

ggplot(year_avgs_wide |> filter(Year<2025), aes(x=Year)) +
    geom_line(aes(y=maxtemp), color="red", size=1, alpha=0.5) +
    geom_line(aes(y=rollmean(maxtemp, 10, na.pad=TRUE)), linetype="dashed") +
    geom_line(aes(y=mintemp), color="blue", size=1, alpha=0.5)+
    geom_line(aes(y=rollmean(mintemp, 10, na.pad=TRUE)), linetype="dashed")
```

3. Daily Weather Data

3.1 Import data

```{r}
temps <- read.csv("../data/Hanover-temps21c_w-daynums.csv")
```

3.2 Examine daily temperature data

```{r}
head(temps)
```

```{r}
colnames(temps)
```

3.3 Graph one year of daily temp data

```{r}
library(viridis)
ggplot(temps |> filter(Year == 2022), aes(x=daynum, y=maxdiff20c, fill=maxdiff20c)) +
    geom_bar(stat="identity") +
    scale_fill_viridis(option = "plasma")

#scale_fill_gradient(low = "blue", high = "red")
```

3.4 Graph multiple years of daily temp data

```{r fig.width=14, fig.height = 60}

#options(repr.plot.width=30, repr.plot.height=50)
ggplot(temps |> filter(Year>2015), aes(x=daynum, y=maxdiff20c, fill=maxdiff20c)) +
    geom_bar(stat="identity") +
    facet_wrap(~Year, ncol=1) +
    scale_fill_viridis(option = "plasma")
```