Lecture 2

For this week we'll be analzying a subset (had to get rid of the embarassing stuff) of my Amazon orer data. The goal of this lecture is to get everyone acclimated to the workflow of R - importing data, piping your way to results, and then asking questions about the data. I'll be utilizing two main packages today: `tidyverse` (which is a collection of many packages) and `lubridate` which is a package dedicated to making working with dates easier.

On the packages note, recall that a package is just a collection of functions. In particular, the functions are solutions to problems that people commonly have. The `tidyverse` on the other hand, is a collection of packages (e.g. a collection of functions) that follow similar syntax. For instance, all tidyverse function use snake_case as their names, and I believe most of the functions are backed with C++ for speed. I recommend you all keep a tidyverse-approach for the course - it was created to make data wrangling and analysis MUCH easier and standardized. Now let's load the packages that we want:

```
library(tidyverse)
library(lubridate)
```

Note that we should always have our packages loaded at the top of every script! DO NOT load packages within the middle of end of a script! This is bad coding practice as people want to know what packages you are using when reading a script. Remember, you need to a load a package every time before using it.

Now let's import the data using the `readr::read_csv` function. Note that there is a difference between `read.csv` and `read_csv`.

```
amazon <- read_csv("lectures/lecture_2/amazon.csv")
```

Cleaning Data

The first thing to do when cleaning the data is to clean the column names of the data. We're going to do this using the `janitor::clean_names` function. Instead of loading in the library though, I want to show you how you can call a function from a certain package without loading it in. You can do this with the `::` syntax. For instance, `library(janitor)` followed by `clean_names` is the same as `janitor::clean_names`. The double colon tells us we want the function from a specific package.

The `janitor::clean_names` function simply changes all column names to snake_case. Let's open the documentation. We can see that there are other arguments that can be passed into it. Let's try one:

```
janitor::clean_names(amazon, case = "sentence") ## try a couple of the cases out
```

We're going to use the default "snake_case" as I mentioned multiple times. However, let's not talk about the pipe operator a little more in-depth.

The Pipe

The pipe operator is meant to signify "and-then" in your code. It is a nice way to chain multiple functions together. What the pipe does is take your data frame and automatically input it in for the data-frame argument of a function. Note that the following are equivalent statements:

```
## equivalent statements
janitor::clean_names(amazon)

amazon %>%
  janitor::clean_names()
```

While this seems silly in this example to prefer one over the other, I will soon show another example in which this becomes more relevant. But first, we need to save the changes we made to the amazon data frame. Remember that simply performing a function will not save the results - we need to actually update the data frame!

```
## saving the changes
amazon <- amazon %>%
  janitor::clean_names()
```

Counting, Filtering, Mutating, and Relocating

We're first going to go through the `dplyr::count` and the `base::head` functions and I'll demonstrate why the pipe is so nice.

```
## equivalent statements
head(count(amazon,condition, sort = T), 10)

amazon %>%
  count(condition, sort = T) %>%
  head(10)
```

See how much easier this is to read. Note my syntax as well: one function per line separated by a pipe.

IN CLASS QUESTION: How can I rename the `n` column using the count function arguments?

Now let's use talk about the `dplyr::filter` function. This simply filters things based on criteria: if this criteria is true, it will return. We'll pull up the documentation too.

```
amazon %>%
  count(category, sort = T) %>%
  filter(n > 5)
```

Next function we're going to look at is the `dplyr::mutate` function. This creates a new column OR it can also "mutate" an existing column. Observe:

```
## Equivalent statements
amazon$unspsc_code * 100

amazon %>%
  mutate(unspsc_code = unspsc_code * 100)
```

Let's now take a look at the `item_subtotal` column.What is it? It's a character with weird strings. This is probably a typical problem is there a solution to use it? use google! make to sure to type in tidyverse solution. We can use the `dplyr::mutate` function to change columns on the fly without actually having to create a new column. Change to just an integer using the round function and look up the documentation. Look at the arguments for mutate and what is benefit of having `.before = 1`?

```
amazon <- amazon %>%
  mutate(item_subtotal = parse_number(item_subtotal)) %>% ## comes from readr package!- part of the tid
  mutate(item_subtotal_integer = round(item_subtotal, 0), .before = 1) %>%
  relocate(item_subtotal)
```

Lubridate and Dplyr::Select

Now let's talk about how to get dates in correctly.

```
amazon %>%
  mutate(order_date = mdy(order_date))

## what other columns have to do with dates? we can actually use the selct() function
## let's talk about tidy selectors
amazon %>%
  select(starts_with("order"))
amazon %>%
  select(ends_with("date"))
```

STOP: IN CLASS: Create a new column that shows the difference in shipment date and the order date. Call this `ship_lag`.

```
## change the release_date column to a date not a date-time. google this to find an answer
amazon %>%
  mutate(order_date = mdy(order_date),
         shipment_date = mdy(shipment_date),
         release_date = as_date(release_date)) %>%
  select(ends_with("date"))
```

```
## create a new column that shows the difference in shipment date and the order date
amazon %>%
  mutate(order_date = mdy(order_date),
         shipment_date = mdy(shipment_date),
         release_date = as_date(release_date)) %>%
  select(ends_with("date"))  %>%
  mutate(ship_lag = shipment_date - order_date)  %>%
  group_by(category) %>%
  summarize(avg_ship_lag = mean(ship_lag))
```

```
## can we get the average shipment delay by category?
amazon %>%
  mutate(order_date = mdy(order_date),
         shipment_date = mdy(shipment_date),
         release_date = as_date(release_date)) %>%
  mutate(ship_lag = shipment_date - order_date)  %>%
  group_by(category) %>%
```

```
  summarize(avg_ship_lag = mean(ship_lag)) %>%
  arrange(desc(avg_ship_lag))
```

In Class Assignment:

1. How many different people use my account?
2. How many different places have I lived?
3. What year did I spend the most money?
4. Can you find what years I lived in San Diego and which dates I lived in Santa Barbara?