# Modu e 5: Data Manipu a tion and Ana ysis

# Functions for Data Manipulation

- filter: pick rows matching criteria
- slice: pick rows using index(es)
- select: pick columns by name
- rename: rename specific columns
- arrange: reorder rows
- mutate: add new variables
- transmute: create new data frame with variables

# More verbs

- bind_cols, bind_rows: bind multiple data frames by row and column
- group_by: create groups of rows according to a condition
- summarise: apply computations across groups of rows
- gather: make wide data longer
- spread: make long data wider
- map: apply computations across columns, related map_dbl, map_dfr
- pmap: apply computations across rows, related pmap_dbl, pmap_dfr
- *_join where * = inner, left, right, or full: join two data frames together according to common values in certain columns, and * indicates how many rows to keep

# Let's load our data!

```
library(tidyverse)
library(haven)
nhanes <- read_dta('nhanes_clean.dta')
names(nhanes)
```

```
 [1] "id"                "age"               "sex"
"race"
 [5] "height"            "weight"            "sbp"
"dbp"
 [9] "cholesterol"   "triglycerides" "hdl"
"healthstatus"
[13] "heartattack"   "diabetes"
"finalwgt"      "strata"
[17] "psu"               "today"             "dob"
"bmi"
[21] "HighBP"
```

# Filtering based on multiple conditions

- `filter(condition1, condition2)` will return rows where both conditions are met.
- `filter(condition1, !condition2)` will return all rows where condition one is true but condition 2 is not.
- `filter(condition1 | condition2)` will return rows where condition 1 and/or condition 2 is met.
- `filter(xor(condition1, condition2)` will return all rows where only one of the conditions is met, and not when both conditions are met.

# Combining multiple conditions

```
nhanes %>%
  filter(age == 50, (weight > 113 | bmi > 30))
```

# Selecting columns based pre-identified columns

```
blood_related <- c("sbp", "dbp",
                   "cholesterol", "hdl",
                   "triglycerides")

nhanes %>%
  select(!!blood_related)

nhanes %>%
  select(one_of(blood_related))
```

# Selecting columns by logical

```
nhanes %>%
  select_if(is.character) %>%
  glimpse
```

```
Rows: 1,267
Columns: 1
$ id <chr> "NC0001", "NC0002", "NC0003",
"NC0004", "NC0005", "NC0006", "NC0007…
```

# Select the negation

```
nhanes %>%
  select_if(~!is.numeric(.)) %>%
  glimpse
```

```
Rows: 1,267
Columns: 2
$ id  <chr> "NC0001", "NC0002", "NC0003",
"NC0004", "NC0005", "NC0006", "NC000…
```

```
$ dob <date> NA, 1943-08-24, 1954-05-08, 1950-
06-25, 1987-06-11, 1956-10-30, 1…
```

```
nhanes %>%
  select_if(funs(!is.numeric(.))) %>%
  glimpse
```

```
Rows: 1,267
Columns: 2
$ id  <chr> "NC0001", "NC0002", "NC0003",
"NC0004", "NC0005", "NC0006", "NC000…
$ dob <date> NA, 1943-08-24, 1954-05-08, 1950-
06-25, 1987-06-11, 1956-10-30, 1…
```

# Selecting columns by multiple logical expressions

```
nhanes %>%
    select_if(is.numeric) %>%
    select_if(~mean(., na.rm=TRUE)>4 ) %>%
    glimpse()
```

```
nhanes %>%
  select_if(~is.numeric(.) & mean(.,
```

```
na.rm=TRUE) > 4) %>%
  glimpse()
```

# Renaming columns

```
nhanes %>%
  select(systolic_blood_pressure = sbp,
         diastolic_blood_pressure = dbp) %>%
  glimpse
```

```
nhanes %>%
    rename(systolic_blood_pressure = sbp,
           diastolic_blood_pressure = dbp) %>%
  glimpse
```

# Reformatting all column names

```
nhanes %>%
  select_all(toupper) %>%
  glimpse
```

```
nhanes %>%
  select_all(tolower) %>%
  glimpse
```

# Mutating columns: the basics

- You can make new columns with the mutate() function. The options inside mutate are almost endless: pretty much anything that you can do to normal vectors, can be done inside a mutate() function.
- Anything inside mutate can either be a new column (by giving mutate a new column name), or can replace the current column (by keeping the same column name).
- One of the simplest options is a calculation based on values in other columns.

```
nhanes %>%
  mutate(weight_lbs = weight*2.2046) %>%
  glimpse
```

```
nhanes %>%
  select(id, sbp) %>%
  mutate(
    sbp_vs_AVG = sbp - round(mean(sbp), 1),
    sbp_vs_MIN = sbp - min(sbp)) %>%
  glimpse
```

# `mutate()` with `ifelse()`

```
nhanes %>%
  select(id, sbp) %>%
  mutate(hypertension = ifelse(sbp > 130,
'Yes', 'No')) %>%
  glimpse
```

# Creating new multiple levels column

- If you want more than two levels, it might be even easier to use case_when()
- The arguments are evaluated in order, so only the rows where the first statement is not true will continue to be evaluated for the next statement. For everything that is left at the end just use the TRUE ~ "newname".

```
nhanes %>%
  select(id, sbp) %>%
  mutate(status = case_when(
    sbp > 180 ~ "Hypertensive Crisis",
    sbp > 140 ~ "Hypertension Stage 2",
    sbp > 130 ~ "Hypertension Stage 1",
    sbp > 120 ~ "Elevated",
    TRUE ~ "Normal"))
```

# Note

```
nhanes <- nhanes %>%
  mutate(weight_lbs = weight*2.2046)
```

```r
nhanes2 <- nhanes %>%
  mutate(weight_lbs = weight*2.2046)
```

# Save the Cleaned Data to File System

- Once you clean up the data, you may want to save the cleaned data to your file system, so that in the future you can load the cleaned data directly.
- To save the data, you can use write.csv() function. Change the working directory
- Use write.csv() to save the data

```
write.csv(nhanes2, "nhanes_clean.csv")
```