Day4: Regression Analysis and Reporting Results

Qingyin Cai

Department of Applied Economics
University of Minnesota

WebR Status

🟢 Ready!

Recap

So far, we have learned…

- The basic types of data structures in R, and how to create and manipulate them.
- Data wrangling with `data.table` package.
- Data visualization with `ggplot2` package.

With the tools we learned so far, you can do a lot of tasks for descriptive data analysis!

Once you have a good understanding of the data, you can move on to the next step: econometric analysis!

Learning Objectives

Today’s goal is to:

- create a descriptive summary table for the data.
- use `lm` function to estimate a regression model and report the results with publication-ready summary tables.
- understand how to create a report document (html and PDF) with Quarto.

Reference

- `modelsummary` package [See here for the package documentation, important!]

Notes

- Today's lecture is an introduction to basic regression analysis with R.
- The more advanced R functions such as `feols()` function from `fixest` package for fixed effects models and `glm()` function for generalized linear models will be covered in the Econometric class (APEC 8211-8214).
    - But the basic syntax are the same. So, you can easily apply the knowledge you learn today to the more advanced functions.

Today's outline:

1. Introduction to Regression analysis with R
2. Create a summary table
   - Introduction to `modelsummary` package
   - `modelsummary()` function to report regression results
   - `modelsummary()` function: Customization
   - `datasummary()` function to report descriptive statistics

Introduction to Regression analysis with R

Before We Start

We will use the CPS1988 dataset from the AER package. It's a cross-section dataset originating from the March 1988 Current Population Survey by the US Census Bureau. For further information, see ?CPS1988 after loading the package.

Run the following code:

Run Code

```
library(AER)
data(CPS1988)

# I prefer to convert the data to data.table.
setDT(CPS1988)

# For practice, I converted some factor variables into character variables.
CPS1988[,`:=`(
  ethnicity = as.character(ethnicity),
  region = as.character(region),
  parttime = as.character(parttime)
)]
```

Introduction to Regression Analysis with R

Basics of lm()

The most basic function to estimate a linear regression model in R is the `lm` function from `stats` package, which is a built-in R package.

Suppose we have a regression model of the form:

With the `lm` function, we can estimate the above model as follows:

```r
# Example R-code
lm(formula = Y ~ X1 + X2, data = dataset)
```

- In the first argument of the `lm` function, you specify the formula of the regression model.
- The intercept is included by default. So, you don't need to include it in the formula.
- `~` splits the left side and right side in a formula.

Introduction to Regression Analysis with R

Example

Let's estimate the following model with the `CPS1988` data:

$$wage = \beta_0 + \beta_1 education + \beta_2 experience + e$$

Run Code

```
# Your turn. What is the code? What does the output look like? Can you find any other information other than the estimated coefficients?

reg <- # write your code here
```

Introduction to Regression Analysis with R

Summary Results

To see the summary of the regression results, use the `summary` function.

Run Code

```
reg <- lm(formula= wage ~ education + experience, data = CPS1988)
reg_summary <- summary(reg)
```

Introduction to Regression Analysis with R

Extracting Information1

The results from `lm()` and `summary()` contain a lot of information (In your Rstudio, you can check them on the Environment pane).

Run Code

```
# See the objects stored in the results of lm() function.
ls(reg)
# See the objects stored inside the result of summary() function
ls(reg_summary)
```

You can access any information stored in object via the `$` operator.

Example 1: Extract the fitted values | Example 2: The coefficient estimates | Example 3: The coefficient estimates with standard errors and t-statistics.

Run Code

```
fitted_values <- reg$fitted.values
```

Introduction to Regression Analysis with R

Extracting Information2

- Contents of `lm()` vs `summary(lm())` objects.

Category	In `lm()` object (`reg`)	In `summary(lm())` object (`reg_summary`)
Coefficients	`coefficients`: estimated β-hats	`coefficients`: table of β-hat, Std. Error, t, p
Residuals	`residuals`: raw residuals	`residuals`: residuals (trimmed for summary)
Fitted Values	`fitted.values`: predicted ŷ	–
Model Info	`call`, `terms`, `model`, `assign`, `xlevels`	`call`, `terms`, plus degrees of freedom info
Diagnostics	–	`r.squared`, `adj.r.squared`, `sigma`, `fstatistic`
Variance–Covariance	`qr`, `effects`, `rank`, `df.residual`	`cov.unscaled`, `aliased`
Degrees of Freedom	`df.residual`	`df`: regression, residual, total

Introduction to Regression Analysis with R

In-class Exercise

Questions

Let’s get the value of the standard error of the coefficient estimate of `education`.

Run Code

```
# You can write your code here.
```

Introduction to Regression Analysis with R

In-class Exercise

Answers

Let’s get the value of the standard error of the coefficient estimate of `education`.

Run Code

```
reg_summary$coefficients["education", "Std. Error"]

# or reg_summary$coefficients[2, 2]
```

Regression with Various Functional Forms

Basics

- To include interaction terms in the formula in `lm()` function:
  - `*` = main effects + interactions.
  - `:` = interaction only.
- To include arithmetic terms in the formula in `lm()` function, use the `I()` function.
  - `I()` = arithmetic (square, product, etc).
- For log transformation, use the `log()` function in the formula.
  - Or you define a new variable with the transformed variable and include it in the formula.

Example:

To estimate:

$$log(wage) = \beta_0 + \beta_1 education + \beta_2 experience + \beta 3 experience^2 +$$

Run Code

```
summary(
  lm(log(wage) ~ education + experience + I(experience^2), data = CPS1988)
)

#lm(log(wage) ~ education + experience + I(experience^2), data = CPS1988) %>%
#  summary()
```

Categorical Variables

Basics

What if we want to include a categorical variable (e.g., `region`, `parttime`, `ethnicity`) in the regression model?

`lm()` function is smart enough to convert the categorical variable into dummy variables without any additional coding.

- Even the variables you want to use as dummy variables are character type, `lm()` function automatically coerced it into a factor variable.

Categorical Variables

Examples

Two categories

What if we want to include a dummy variable that takes 1 if `parttime` is yes, otherwise 0?

The model is as follows:

$$log(wage) = \beta_0 + \beta_1 education + \beta_2 experience + \beta_3 experience^2 +$$

▶ Run Code

```
summary(
  lm(log(wage) ~ education + experience + I(experience^2) + parttime , data = CPS1988)
)

#lm(log(wage) ~ education + experience + I(experience^2) + parttime, data = CPS1988) %>%
#  summary()
```

Categorical Variables

Examples

More than Two Categories

What if we want to include dummy variables for each `region`?

Run Code

```
CPS1988[, region := factor(region)]
levels(CPS1988$region)

# set the base level
CPS1988[, region := relevel(region, ref = "south")]

# run the model
cps_region <- lm(log(wage) ~ ethnicity + education + experience + I(experience^2) + region,
                 data = CPS1988)
summary(cps_region)
```

Categorical Variables

Set the Base Group

By default, R picks the first group in the alphabetical order for the base group.

You can use `relevel()` function (a built-in R function) to set the base group of the categorical variable.

Syntax:

```
relevel(factor_variable, ref = "base_group")
```

Example:

Let's compare the two regression results:

- use `parttime==no` as the base group
- use `parttime==yes` as the base group

Run Code

```
# 1. Use the group with parttime==no as the base group (default)
CPS1988[, parttime := relevel(as.factor(parttime), ref = "no")]
# check
unique(CPS1988$parttime)

summary(
  lm(log(wage) ~ ethnicity + education + experience + I(experience^2) + parttime,
     data = CPS1988)
)$coefficients
```

```
# 2. Use the group with parttime==Yes as the base group
CPS1988[, parttime := relevel(as.factor(parttime), ref = "yes")]

summary(
  lm(log(wage) ~ ethnicity + education + experience + I(experience^2) + parttime,
     data = CPS1988)
)$coefficients
```

Prediction

To do prediction with the estimated model on a new dataset, you can use the `predict` function (built-in R function).

Syntax

```
predict(lm_object, newdata = new_dataset)
```

Example

▶ Run Code

```
reg <- lm(log(wage) ~ experience + I(experience^2), data = CPS1988)

new_data <-
  data.table(
    experience = seq(from = 10, to = max(CPS1988$experience), by = 0.5)
  )

new_data[, predicted_wage := predict(reg, newdata = new_data)]

# visualize the predicted values
ggplot(new_data) +
  geom_point(aes(x = experience, y = predicted_wage), color = "blue") +
  theme_bw()
```

Key Points

You should at least know these key points:

- the basic usage of `lm()` and `summary()` function.
- how to retrieve the information stored in the outputs of `lm()` and `summary()` functions.
- how to include log-transformed variable, interaction terms and quadratic terms in the formula of `lm()` function.
- how to include categorical variables in the formula of `lm()` function, and how to set the base group.
- how to do prediction with the estimated model on a new dataset.

That's it!

Create Publication-Ready Summary Tables

Introduction to `modelsummary` package

Intro

`modelsummary` package lets you create a nice summary table to report the descriptive statistics of the data and the regression results.

We focus on two functions in the `modelsummary` package:

- `datasummary()`: to create a summary table for the descriptive statistics of the data.
- `modelsummary()`: to create a summary table for the regression results.

Check the documentation for more details.

Introduction to `modelsummary` package

Example

Descriptive Statistics

Table 1: Example of Summary Statistics

	Mean	SD	Min	Max
Wage	603.73	453.55	50.05	18777.20
Education	13.07	2.90	0.00	18.00
Experience	18.20	13.08	-4.00	63.00

Regression Summary Table

Table 2: Example regression results

	OLS 1	OLS 2	OLS 3
Education	0.076***	0.087***	0.086***
	(0.001)	(0.001)	(0.001)
Experience		0.078***	0.077***
		(0.001)	(0.001)
Experience squared		-0.001***	-0.001***
		(0.000)	(0.000)
White			-0.243***
			(0.013)
Num.Obs.	28155	28155	28155
R2	0.095	0.326	0.335
R2 Adj.	0.095	0.326	0.335

* p < 0.05, ** p < 0.01, *** p < 0.001

Std. Errors in parentheses

modelsummary() function to report regression results

Basics

The very basic argument of the `modelsummary()` function is the `models` argument, which takes **a list of regression models** you want to report in the table.

```
# --- 1. Estimate regression models --- #
lm1 <- lm(y ~ x1, data = dataset)
lm2 <- lm(y ~ x1 + x2, data = dataset)
lm3 <- lm(y ~ x1 + x2 + x3, data = dataset)

# --- 2. Then, provide those a list of lm objects in the "models" argument  --- #
modelsummary(models=list(lm1, lm2, lm3))
```

modelsummary() function to report regression results

Default Appearance

Example

```r
reg1 <- lm(log(wage) ~ education, data = CPS1988)
reg2 <- lm(log(wage) ~ education + experience + I(experience^2), data = CPS1988)

modelsummary(models=list(reg1, reg2))
```

	(1)	(2)
(Intercept)	5.178	4.278
	(0.019)	(0.019)
education	0.076	0.087
	(0.001)	(0.001)
experience		0.078
		(0.001)
I(experience^2)		-0.001
		(0.000)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326
AIC	405753.0	397432.7
BIC	405777.7	397473.9
Log.Lik.	-29139.853	-24977.715
F	2941.787	4545.929
RMSE	0.68	0.59

modelsummary() function: Customization

List of Options

The default table is okay. But you can customize the appearance of the table. Here, I listed the bare minimum of options you might want to know (There are lots of other options!).

- `models`: you can change the name of the models
- `coef_map`: to reorder coefficient rows and change their labels
- `stars`: to change the significance stars
- `vcov`: to replace the standard errors with the robust ones (we will see this later)
- `gof_map`: to define which model statistics to display
- `gof_omit`: to define which model statistics to omit from the default selection of model statistics
- `notes`: to add notes at the bottom of the table
- `fmt`: change the format of numbers

Note

+ See `?modelsummary` for more details or see this. + Also check the vignette of the function from here.

modelsummary() function: Customization

models

By naming the models when you make a list of regression models, you can change the name of the models in the table.

Example

```
reg1 <- lm(log(wage) ~ education, data = CPS1988)
reg2 <- lm(log(wage) ~ education + experience + I(experience^2), data = CPS1988)

ls_regs <- list("OLS 1" = reg1, "OLS 2" = reg2)

modelsummary(models = ls_regs)
```

	OLS 1	OLS 2
(Intercept)	5.178	4.278
	(0.019)	(0.019)
education	0.076	0.087
	(0.001)	(0.001)
experience		0.078
		(0.001)
I(experience^2)		-0.001
		(0.000)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326
AIC	405753.0	397432.7
BIC	405777.7	397473.9
Log.Lik.	-29139.853	-24977.715
F	2941.787	4545.929
RMSE	0.68	0.59

modelsummary() function: Customization

coef_map

- `coef_map` argument helps you clean up your regression table.
  - You can choose which coefficients to show (subset).
  - You can change their order (reorder).
  - You can give them nicer labels (rename).
  - If you give a named vector, the names you supply will replace the default variable names in the table.

Example

In this example, I renamed the variables and moved the `intercept` row to the bottom row.

```r
modelsummary(
  models =  list("OLS 1" = reg1, "OLS 2" = reg2),
  coef_map = c(
    "education" = "Education",
    "experience" = "Experience",
    "I(experience^2)" = "Experience squared",
    "(Intercept)" = "Intercept"
    )
  )
```

	OLS 1	OLS 2
Education	0.076	0.087
	(0.001)	(0.001)
Experience		0.078
		(0.001)
Experience squared		-0.001
		(0.000)
Intercept	5.178	4.278
	(0.019)	(0.019)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326
AIC	405753.0	397432.7
BIC	405777.7	397473.9
Log.Lik.	-29139.853	-24977.715
F	2941.787	4545.929
RMSE	0.68	0.59

modelsummary() function: Customization

stars

`stars = TRUE` shows the significance stars in the table (Try it!).

If you don't like it, you can modify significance levels and markers by specifying a named numeric vector (e.g., `stars = c("*" = .05, "**" = .01, "***" = .001)`).

Example

```r
modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  coef_map = c(
    "education" = "Education",
    "experience" = "Experience",
    "I(experience^2)" = "Experience squared",
    "(Intercept)" = "Intercept"
    ),
  stars  =  c("*" = .05, "**" = .01, "***" = .001),
  )
```

	OLS 1	OLS 2
Education	0.076***	0.087***
	(0.001)	(0.001)
Experience		0.078***
		(0.001)
Experience squared		-0.001***
		(0.000)
Intercept	5.178***	4.278***
	(0.019)	(0.019)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326
AIC	405753.0	397432.7
BIC	405777.7	397473.9
Log.Lik.	-29139.853	-24977.715
F	2941.787	4545.929
RMSE	0.68	0.59

* p < 0.05, ** p < 0.01, *** p < 0.001

modelsummary() function: Customization

vcov

Basics | Preparation | Report robust-standard errors

`vcov` argument let you replace the non-robust standard errors (default) with the robust one. Here are some options to use the `vcov` argument (see this more options).

Option 1: Supply a list of named variance-covariance matrices:

```
vcov_reg1 <- vcovHC(reg1, type = "HC1")
vcov_reg2 <- vcovHC(reg2, type = "HC1")

modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  vcov = list(vcov_reg1, vcov_reg2)
  )
```

Option 2: Supply a name or a list of names of variance-covariance estimators (e.g, “HC0”, “HC1”, “HC2”, “HC3”, “HAC”).

```
modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  vcov = "HC1"
  )
```

In this case, HC1 estimator is used for all the models.

Note

By default, `modelsummary()` calculates the robust variance-covariance matrix using the `sandwich` package (`sandwich::vcovHC`, `sandwich::vcovCL`).

modelsummary() function: Customization

coef_omit

`coef_omit` lets you omit coefficient rows from the default selections. In the argument, you specify a vector of names or row number of variables you want to omit from the table.

- e.g., `coef_omit = c(2,3,5)` omits the second, third, and fifth coefficients.

Example

Let's remove the intercept from the table.

```
modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  coef_map = c(
    "education" = "Education",
    "experience" = "Experience",
    "I(experience^2)" = "Experience squared",
    "(Intercept)" = "Intercept"
    ),
  stars  =  c("*" = .05, "**" = .01, "***" = .001),
  coef_omit = 1
  )
```

modelsummary() function: Customization

gof_map and gof_omit

By default, the `modelsummary()` function reports lots of model statistics (e.g., R^2, AIC, BIC). You can select or omit the model statistics by specifying the `gof_map` and `gof_omit` arguments.

- You can see the list of model statistics in `modelsummary()` by running `modelsummary::gof_map`

Example

For example, let's select only the number of observations, R^2, and adjusted R^2 using the `gof_map` argument.

```r
modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  coef_map = c(
    "education" = "Education",
    "experience" = "Experience",
    "I(experience^2)" = "Experience squared"
    ),
  stars  =  c("*" = .05, "**" = .01, "***" = .001),
  gof_map = c("nobs", "r.squared",  "adj.r.squared", "logLik")
  )
```

	OLS 1	OLS 2
Education	0.076***	0.087***
	(0.001)	(0.001)
Experience		0.078***
		(0.001)
Experience squared		-0.001***
		(0.000)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326
Log.Lik.	-29139.853	-24977.715

* p < 0.05, ** p < 0.01, *** p < 0.001

modelsummary() function: Customization

others

- `notes` lets you add notes at the bottom of the table.
- `fmt` lets you change the format of numbers in the table.

Example

For example, let's select only the number of observations, R^2, and adjusted R^2 using the `gof_map` argument.

```r
modelsummary(
  models = list("OLS 1" = reg1, "OLS 2" = reg2),
  coef_map = c(
    "education" = "Education",
    "experience" = "Experience",
    "I(experience^2)" = "Experience squared"
    ),
  stars  =  c("*" = .05, "**" = .01, "***" = .001),
  gof_map = c("nobs", "r.squared",  "adj.r.squared"),
  notes = list("Std. Errors in parentheses"),
  fmt = 2 #report the numbers with 2 decimal points
  )
```

	OLS 1	OLS 2
Education	0.08***	0.09***
	(0.00)	(0.00)
Experience		0.08***
		(0.00)
Experience squared		0.00***
		(0.00)
Num.Obs.	28155	28155
R2	0.095	0.326
R2 Adj.	0.095	0.326

* p < 0.05, ** p < 0.01, *** p < 0.001
Std. Errors in parentheses

datasummary() function to report descriptive statistics

`modelsummary` package has another function called `datasummary()` that can create a summary table for the descriptive statistics of the data.

Example

```r
datasummary(
  formula =
    (`Metropolitan area` = smsa) * (
      wage + education + experience
    ) ~
    ethnicity * (Mean + SD),
  data = CPS1988
  )
```

Metropolitan area		afam		cauc	
		Mean	SD	Mean	SD
no	wage	337.42	214.57	527.31	419.65
	education	11.47	2.78	12.71	2.69
	experience	19.69	13.87	19.05	13.13
yes	wage	470.38	324.97	649.39	471.05
	education	12.51	2.73	13.28	2.96
	experience	18.54	13.43	17.83	12.99

datasummary() function: Introduction

Basics

`datasummary()` function creates a summary table for the descriptive statistics of the data.

Syntax

```
datasummary(
  formula = rows ~ columns,
  data = dataset
  )
```

Note

- Just like `lm`, `formula` takes a two-side formula devided by `~`.
- The left-hand (right-hand) side of the formula describes the rows (columns).

Let's see how it works with an example.

datasummary() function: Introduction

Example

```
datasummary(
  formula = wage + education + experience ~ Mean + SD,
  data = CPS1988
  )
```

	Mean	SD
wage	603.73	453.55
education	13.07	2.90
experience	18.20	13.08

Note

- Use `+` to include more rows and columns.
- The `modelsummary` package offers multiple summary functions of its own:
  - `Mean`, `SD`, `Median`, `Min`, `Max`, `P0`, `P25`, `P50`, `P75`, `P100`, `Histogram`
- `NA` values are automatically stripped before the computation proceeds. So you don't need to worry about it.

datasummary() function: Introduction

All()

In the `formula` argument, you can use `All()` function to create a summary table for all the numeric variables in the dataset.

```
datasummary(
  formula = All(CPS1988)~ Mean + SD,
  data = CPS1988
  )
```

	Mean	SD
wage	603.73	453.55
education	13.07	2.90
experience	18.20	13.08

datasummary() function: Introduction

In-class Exercise

```r
datasummary(
  formula = wage + education + experience ~ mean + SD,
  data = CPS1988
  )
```

	mean	SD
wage	603.73	453.55
education	13.07	2.90
experience	18.20	13.08

Play with the datasummary() function:

- Exchange the rows and columns in the formula and see how the table looks.
- Add other statistics or variables to the formula and see how the table looks.

datasummary() Function: Further Tuning

Nesting with ∗ operator

`datasummary` can nest variables and statistics inside categorical variables using the ∗ symbol. For example, you can display separate means and SD's for each value of `ethnicity`.

Example 1: Nested rows

```r
datasummary(
  formula =  ethnicity * (wage + education + experience) ~ mean + SD,
  data = CPS1988
  )
```

ethnicity		mean	SD
afam	wage	446.85	312.44
	education	12.33	2.77
	experience	18.74	13.51
cauc	wage	617.23	461.21
	education	13.13	2.90
	experience	18.15	13.04

Example 2: Nested columns

```r
datasummary(
  formula = wage + education + experience ~ ethnicity * (mean + SD),
  data = CPS1988
  )
```

	afam		cauc	
	mean	SD	mean	SD
wage	446.85	312.44	617.23	461.21
education	12.33	2.77	13.13	2.90
experience	18.74	13.51	18.15	13.04

datasummary() Function: Further Tuning

Multiple Nests

- You can nest variables and statistics inside multiple categorical variables using the $*$ symbol.
- The order in which terms enter the formula determines the order in which labels are displayed.

Example

```
datasummary(
  formula = wage + education + experience ~ region * ethnicity * (mean + SD),
  data = CPS1988
  )
```

	midwest				northeast				south				west			
	afam		cauc		afam		cauc		afam		cauc		afam		cauc	
	mean	SD	mean	SD	mean	SD	mean	SD	mean	SD	mean	SD	mean	SD	mean	SD
wage	458.32	312.58	613.19	450.94	505.58	342.00	663.04	438.30	416.01	293.52	582.93	487.13	518.20	346.94	617.96	457.77
education	12.53	2.62	13.29	2.52	12.28	2.75	13.32	2.77	12.15	2.84	12.91	3.08	13.22	2.38	13.05	3.16
experience	18.53	13.92	17.78	12.90	20.73	14.32	18.69	13.49	18.52	13.22	18.41	13.20	16.89	12.72	17.70	12.48

datasummary() Function: Further Tuning

Renaming the Variables with =

By default, variable and statistics names are used as the labels in the table. You can rename the default labels with the following syntax: `(label = variable/statistic)`.

Example

Before renaming:

```
datasummary(
  formula = wage + education ~ ethnicity * (mean + SD),
  data = CPS1988
)
```

	afam		cauc	
	mean	SD	mean	SD
wage	446.85	312.44	617.23	461.21
education	12.33	2.77	13.13	2.90

After renaming:

```
datasummary(
  formula = (`Wage (in dollars per week)` = wage) + (`Years of Education` = education) ~ ethnicity * (mean + (`Std.Dev` = SD)),
  data = CPS1988
)
```

	afam		cauc	
	mean	Std.Dev	mean	Std.Dev
Wage (in dollars per week)	446.85	312.44	617.23	461.21

	afam		cauc	
	mean	Std.Dev	mean	Std.Dev
Years of Education	12.33	2.77	13.13	2.90

Caution

- In R, `` ` ` `` is used to define a variable name with spaces or special characters such as the parentheses symbol `()`.

datasummary() Function: Further Tuning

In-class Exercise

Data Problem Answers

For this exercise problem, we use `CPSSW3` dataset from the `AER` package. The `CPSSW3` dataset provides trends (from 1992 to 2004) in hourly earnings in the US of working college graduates aged 25–34 (in 2004 USD).

```r
library(AER)
data("CPSSW9204")
head(CPSSW9204)
```

```
  year  earnings     degree gender age
1 1992 11.188810   bachelor   male  29
2 1992 10.000000   bachelor   male  33
3 1992  5.769231 highschool   male  30
4 1992  1.562500 highschool   male  32
5 1992 14.957260   bachelor   male  31
6 1992  8.660096   bachelor female  26
```

Appendix: Other Functions of the modelsummary Package

datasummary_skim

Basics

`datasummary_skim()` with the `type = categorical` option might be helpful to quickly generate a summary table for categorical variables:

Syntax

```
datasummary_skim(data = dataset, type = "categorical")
```

datasummary_skim

Example

```r
datasummary_skim(data = CPS1988[,.(ethnicity, smsa, region, parttime)], type = "categorical")
```

		N	%
ethnicity	afam	2232	7.9
	cauc	25923	92.1
smsa	no	7223	25.7
	yes	20932	74.3
region	midwest	6863	24.4
	northeast	6441	22.9
	south	8760	31.1
	west	6091	21.6
parttime	no	25631	91.0
	yes	2524	9.0

datasummary_balance

Basics

`datasummary_balance()` function creates balance tables with summary statistics for different subsets of the data (e.g., control and treatment groups).

Syntax

```
datasummary_balance(
  formula = variables to summarize ~ group_variable,
  data = dataset
  )
```

datasummary_balance

Example

```r
datasummary_balance(
  formula = wage + education + experience ~ ethnicity,
  data = CPS1988,
  dinm_statistic = "std.error" # or "p.value"
)
```

	afam (N=2232)		cauc (N=25923)			
	Mean	Std. Dev.	Mean	Std. Dev.	Diff. in Means	Std. Error
wage	446.9	312.4	617.2	461.2	170.4	7.2
education	12.3	2.8	13.1	2.9	0.8	0.1
experience	18.7	13.5	18.2	13.0	-0.6	0.3

datasummary_correlation

Basics

`datasummary_correlation()` function creates a correlation table. It automatically identifies all the numeric variables, and calculates the correlation between each of those variables (You don't need to select the numeric variables manually!).

Syntax

```
datasummary_correlation(data = dataset)
```

datasummary_correlation

Example

```
datasummary_correlation(data = CPS1988)
```

	wage	education	experience
wage	1	.	.
education	.30	1	.
experience	.19	-.29	1

How to create present results in Quarto?

- After running some regression models, ultimately you want to report the results in a neat table.
- Usually, we report the regression results in a formatted document like the Rmarkdown or Quarto document (html or PDF).
- So, let's practice how to create a summary table for your analysis results in the Quarto document!
- From my GuitHub, "materials" under "Data/Materials for Class Use" download and open the document file "practice_modelsummary_html.qmd".
- See details here - Documents

Exercise Problems

You can find today's after-class exercise problems under "Data/Materials for Class Use" on my GuitHub.