

# 431 Class 07

Thomas E. Love, Ph.D.

2022-09-20

# Today's Agenda

- Working with `dm1000`
- Some More on Identifying Missing Data and working around it
- Summarizing Categorical Data

# Today's Package Setup

```
1 library(Epi) ## for twoby2() function
2 library(gt) ## making tables
3 library(gtExtras) ## fancier tables
4 library(janitor)
5 library(kableExtra) ## for kbl() function
6 library(naniar)
7 library(patchwork)
8 library(tidyverse)
9
10 theme_set(theme_bw())
```

# Loading Some New Data

```
1 dm1000 <- read_csv("c07/data/dm_1000.csv", show_col_types = FALSE) |>
2   clean_names() |>
3   mutate(across(where(is.character), as_factor)) |>
4   mutate(subject = as.character(subject))
```

- 1000 (simulated) patients with diabetes between the ages of 31 and 75 who live in Cuyahoga County and are in one of four race-ethnicity categories, as well as one of four insurance categories.
- Same variables we saw in `dm431` last week, but 1000 new subjects, and one new variable (`residence`)

# Listing of **dm1000** tibble

```
1 dm1000
```

```
# A tibble: 1,000 × 17
```

	subject	age	insurance	n_income	ht	wt	sbp	dbp	a1c	ldl	tobacco
	<chr>	<dbl>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
	1 M-0001	55	Medicaid	29853	1.63	103.	145	70	6.4	221	
Current											
	2 M-0002	52	Commercial	31248	1.75	112.	151	77	8.5	116	Never
	3 M-0003	69	Medicare	23362	1.65	74.9	127	73	8.9	52	
Former											
	4 M-0004	57	Medicaid	26033	1.63	81.4	125	74	6.8	122	Never
	5 M-0005	68	Medicare	85374	1.69	92.6	120	73	10.3	94	Never
	6 M-0006	56	Medicaid	31273	1.71	54.6	127	75	12.3	NA	
Current											
	7 M-0007	54	Commercial	25445	1.68	81.6	114	81	6.5	100	
Current											
	8 M-0008	45	Medicaid	67500	1.60	80.0	100	110	5.4	113	Never

# dm1000 Code Book (1 of 3)

Variable	Description
<code>subject</code>	subject code (M-0001 through M-1000)
<code>age</code>	subject's age, in years
<code>insurance</code>	primary insurance, 4 levels
<code>n_income</code>	neighborhood median income, in \$
<code>ht</code>	height, in meters (2 decimal places)
<code>wt</code>	weight, in kilograms (2 decimal places)

# dm1000 Code Book (2 of 3)

Variable	Description
sbp	most recent systolic blood pressure (mm Hg)
dbp	most recent diastolic blood pressure (mm Hg)
a1c	most recent Hemoglobin A1c (%)
ldl	most recent LDL cholesterol level (mg/dl)
tobacco	most recent tobacco status, 3 levels
statin	1 = prescribed a statin in past 12m, 0 = not

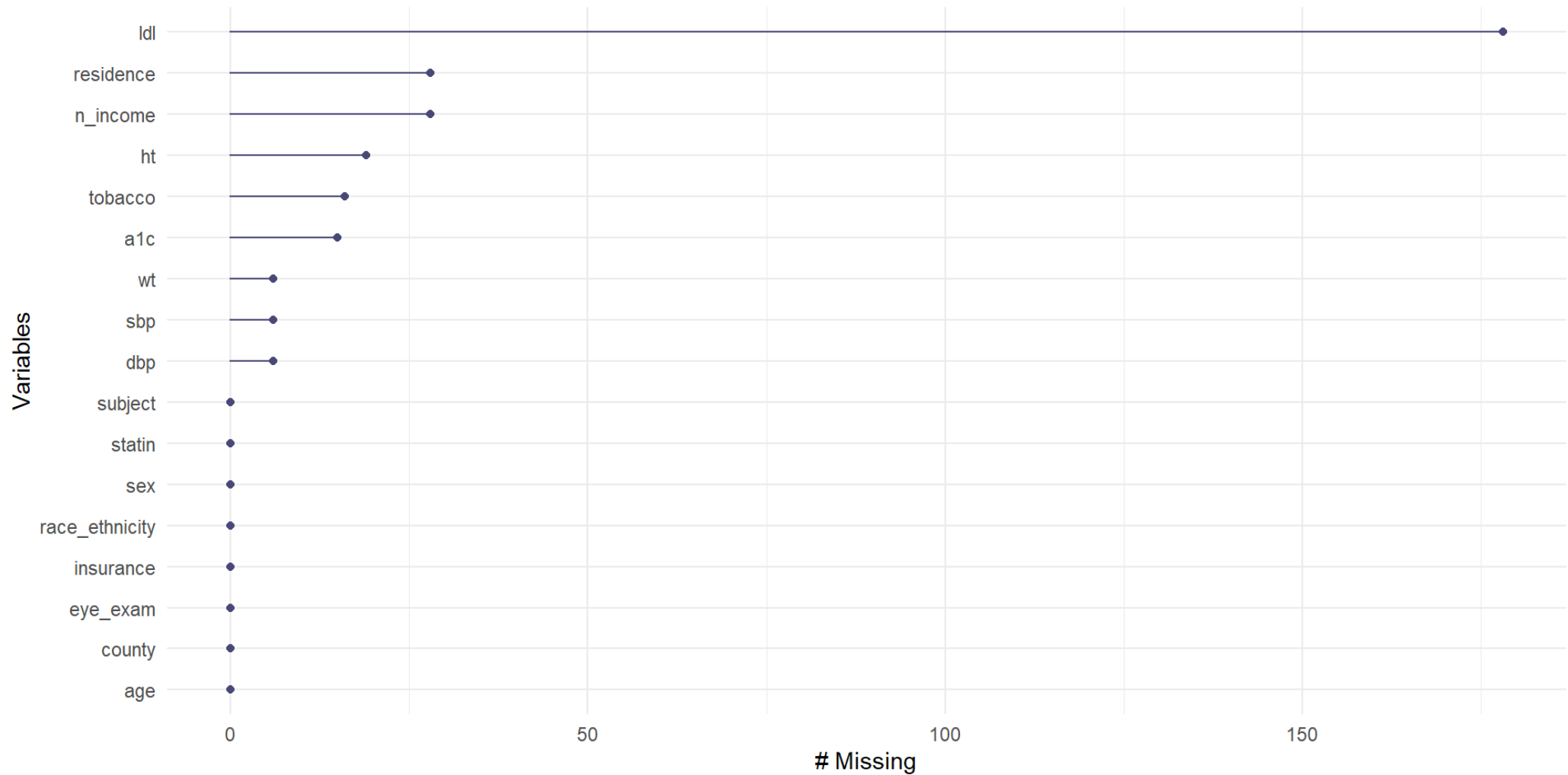
# Remainder of **dm1000** codebook

Variable	Description
<b>eye_exam</b>	1 = diabetic eye exam in past 12m, 0 = no record of exam in past 12m
<b>race_ethnicity</b>	race/ethnicity category, 3 levels
<b>sex</b>	Female or Male
<b>county</b>	all subjects live in Cuyahoga County
<b>residence</b>	Cleveland or Suburbs



# Any Missing Data?

```
1 gg_miss_var(dm1000)
```



# Counts of missingness, by variable

```
1 miss_var_summary(dm1000)
```

```
# A tibble: 17 × 3
```

	variable <chr>	n_miss <int>	pct_miss <dbl>
1	ldl	178	17.8
2	n_income	28	2.8
3	residence	28	2.8
4	ht	19	1.9
5	tobacco	16	1.6
6	alc	15	1.5
7	wt	6	0.6
8	sbp	6	0.6
9	dbp	6	0.6
10	subject	0	0
11	age	0	0
12	insurance	0	0
13	...	...	...

# What does `miss_var_table()` do?

```
1 miss_var_table(dm1000)
```

```
# A tibble: 7 × 3
```

	n_miss_in_var <int>	n_vars <int>	pct_vars <dbl>
1	0	8	47.1
2	6	3	17.6
3	15	1	5.88
4	16	1	5.88
5	19	1	5.88
6	28	2	11.8
7	178	1	5.88

# What does `miss_case_table()` do?

```
1 miss_case_table(dm1000)
```

```
# A tibble: 7 × 3
```

	n_miss_in_case <int>	n_cases <int>	pct_cases <dbl>
1	0	772	77.2
2	1	181	18.1
3	2	32	3.2
4	3	7	0.7
5	4	5	0.5
6	5	2	0.2
7	6	1	0.1

# miss\_case\_summary()?

```
1 miss_case_summary(dm1000)
```

```
# A tibble: 1,000 × 3
  case n_miss pct_miss
  <int> <int>    <dbl>
1    230      6    35.3
2    440      5    29.4
3    970      5    29.4
4    107      4    23.5
5    284      4    23.5
6    288      4    23.5
7    385      4    23.5
8    891      4    23.5
9     91      3    17.6
10   241      3    17.6
# ... with 990 more rows
```

# How should we summarize data with missing values?

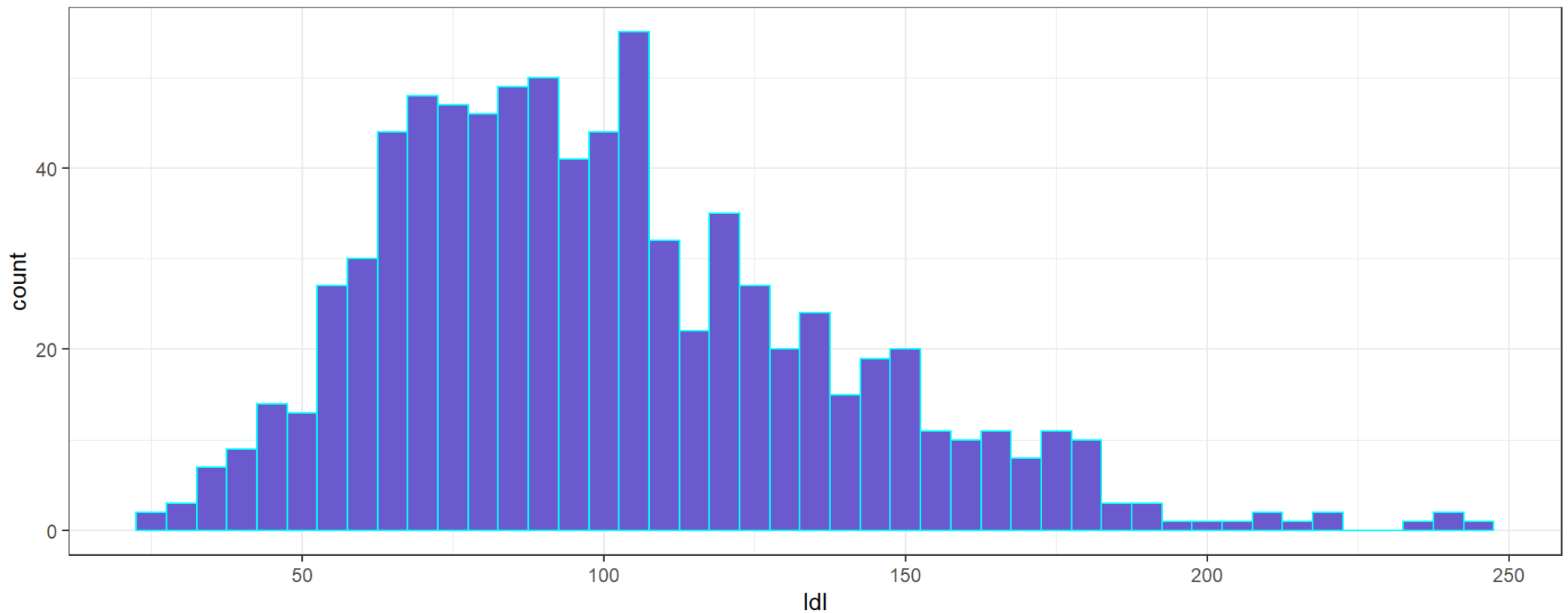
It depends on what you'll do with the data.

- If you are providing a data summary, then you should summarize the complete cases, and specify the number of missing values.
- If you are intending to use the sample you've collected to make an inference about a process or population or to build a model, then you may want to consider whether or not a complete-case analysis will introduce bias.

# What do graphs do with missing data?

```
1 ggplot(data = dm1000, aes(x = ldl)) +  
2   geom_histogram(binwidth = 5, fill = "slateblue", col = "cyan")
```

Warning: Removed 178 rows containing non-finite values (stat\_bin).



# Exploring **ldl** in **dm1000**

```

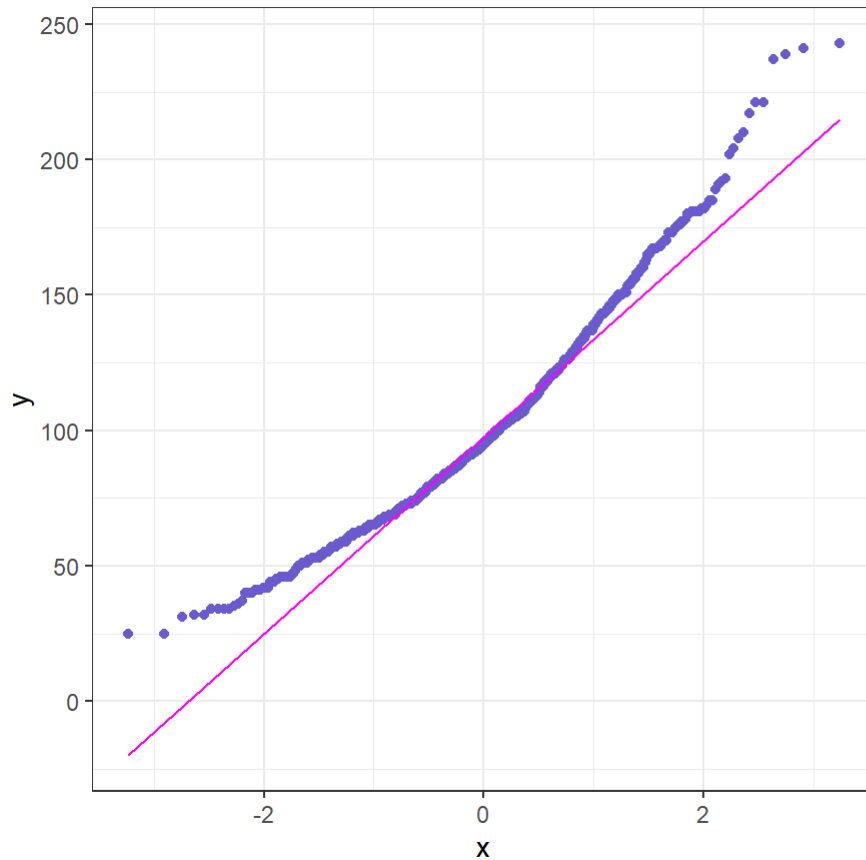
1 p1 <- ggplot(dm1000, aes(sample = ldl)) +
2   geom_qq(col = "slateblue") +
3   geom_qq_line(col = "magenta") +
4   theme(aspect.ratio = 1) +
5   labs(title = "Normal Q-Q plot: dm1000 LDL")
6
7 p2 <- ggplot(dm1000, aes(x = ldl)) +
8   geom_histogram(aes(y = stat(density)),
9                 bins = 20, fill = "slateblue", col = "cyan") +
10  stat_function(fun = dnorm,
11              args = list(mean = mean(dm1000$ldl, na.rm = TRUE),
12                            sd = sd(dm1000$ldl, na.rm = TRUE)),
13              col = "magenta", lwd = 1.5) +
14  labs(title = "Density Function: dm1000 LDL")
15
16 p3 <- ggplot(dm1000, aes(x = ldl, y = "")) +
17   geom_boxplot(fill = "slateblue", outlier.color = "slateblue") +
18   labs(title = "Boxplot: dm1000 LDL", y = "")
19

```

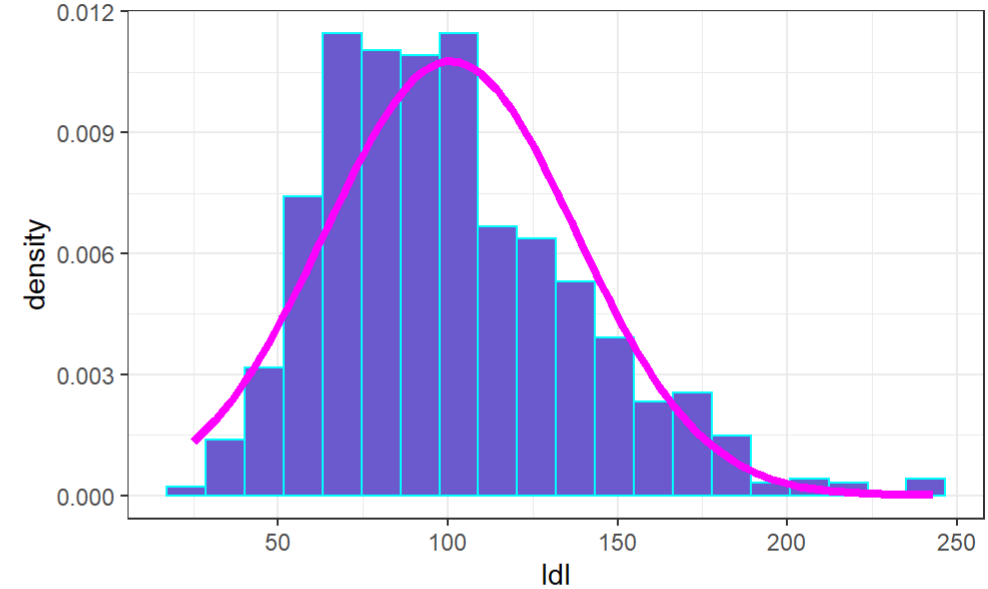


# Exploring **ldl** in **dm1000**

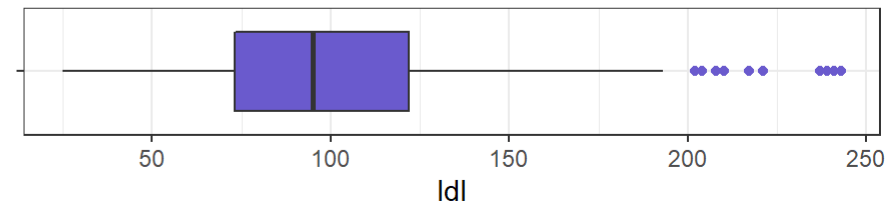
Normal Q-Q plot: dm1000 LDL



Density Function: dm1000 LDL



Boxplot: dm1000 LDL



# Silenced Warnings for previous plot

```
Warning: Removed 178 rows containing non-finite values (stat_qq).  
Warning: Removed 178 rows containing non-finite values (stat_qq_line).  
Warning: Removed 178 rows containing non-finite values (stat_bin).  
Warning: Removed 178 rows containing non-finite values (stat_boxplot).
```

# Numerical Summaries and Missing Data

```
1 summary(dm1000$ldl)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	NA's
25.0	73.0	95.0	100.7	122.0	243.0	178

```
1 mosaic::favstats(~ ldl, data = dm1000)
```

Registered S3 method overwritten by 'mosaic':

method	from
fortify.SpatialPolygonsDataFrame	ggplot2

min	Q1	median	Q3	max	mean	sd	n	missing
25	73	95	122	243	100.7202	37.05353	822	178

- There, I could/should have silenced the message with `{r, message = FALSE}` in the code chunk header.

# Subgroups of Interest

```
1 dm1000 |> tabyl(residence, insurance)
```

residence	Medicaid	Commercial	Medicare	Uninsured
Suburbs	114	75	163	19
Cleveland	201	119	259	22
<NA>	15	2	10	1

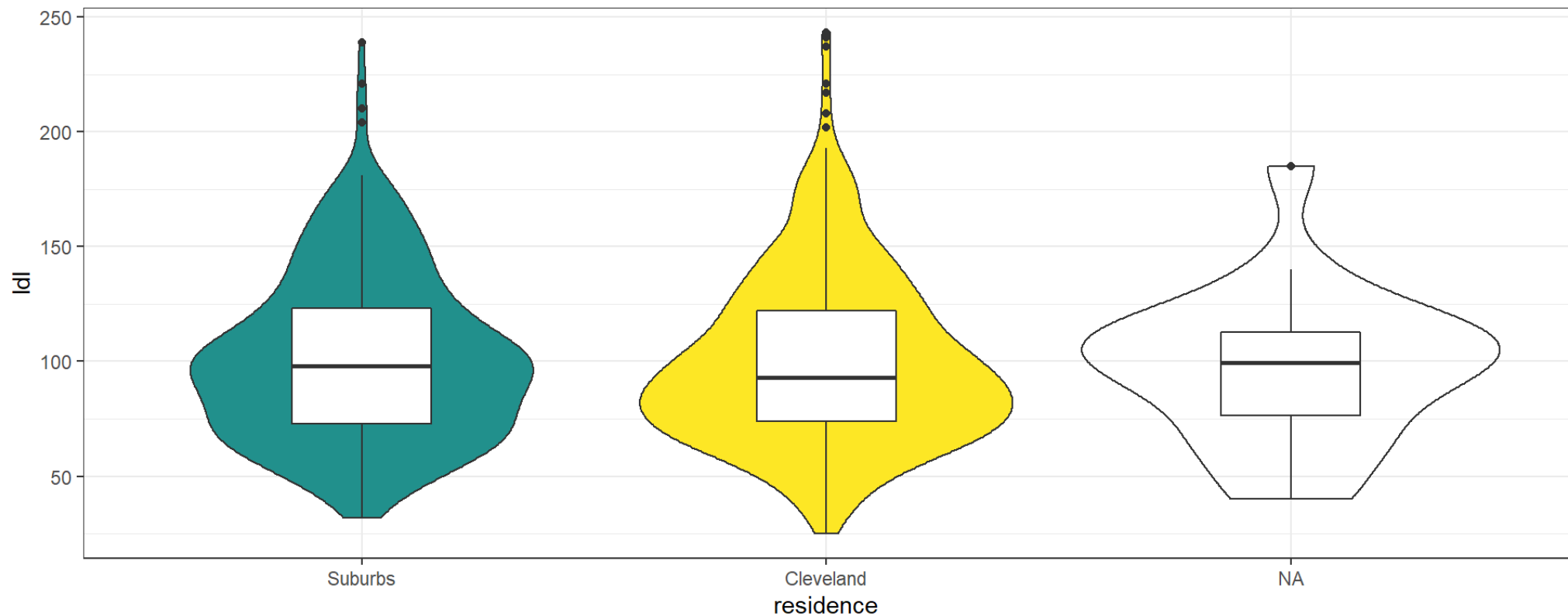
I don't like that **insurance** ordering.

```
1 dm1000 <- dm1000 |>
2   mutate(insurance = fct_relevel(insurance,
3                                   "Medicare", "Commercial", "Medicaid"))
4
5 dm1000 |> tabyl(residence, insurance)
```

residence	Medicare	Commercial	Medicaid	Uninsured
Suburbs	163	75	114	19
Cleveland	259	119	201	22
<NA>	10	2	15	1

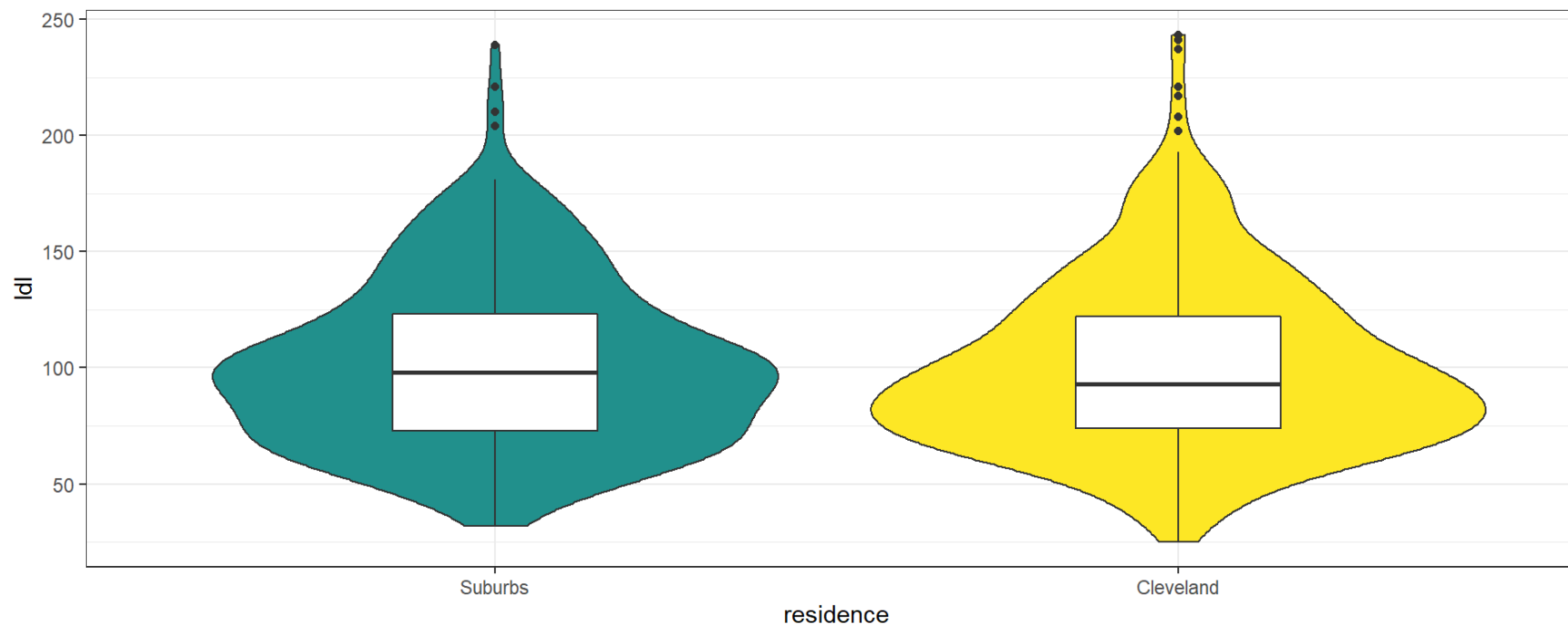
# LDL by Residence

```
1 ggplot(data = dm1000, aes(x = residence, y = ldl)) +  
2   geom_violin(aes(fill = residence)) +  
3   geom_boxplot(width = 0.3) +  
4   scale_fill_viridis_d(begin = 0.5, option = "D") +  
5   guides(fill = "none")
```



# LDL by Residence, again

```
1 tempdat <- dm1000 |> filter(complete.cases(residence))  
2  
3 ggplot(data = tempdat, aes(x = residence, y = ldl)) +  
4   geom_violin(aes(fill = residence)) +  
5   geom_boxplot(width = 0.3) +  
6   scale_fill_viridis_d(begin = 0.5, option = "D") +  
7   guides(fill = "none")
```



# Grouped Numerical Summaries

## LDL by Residence

```
1 mosaic::favstats(ldl ~ residence, data = dm1000)
```

	residence	min	Q1	median	Q3	max	mean	sd	n	missing
1	Suburbs	32	73	98	123	239	101.3814	36.51435	312	59
2	Cleveland	25	74	93	122	243	100.4347	37.57225	490	111

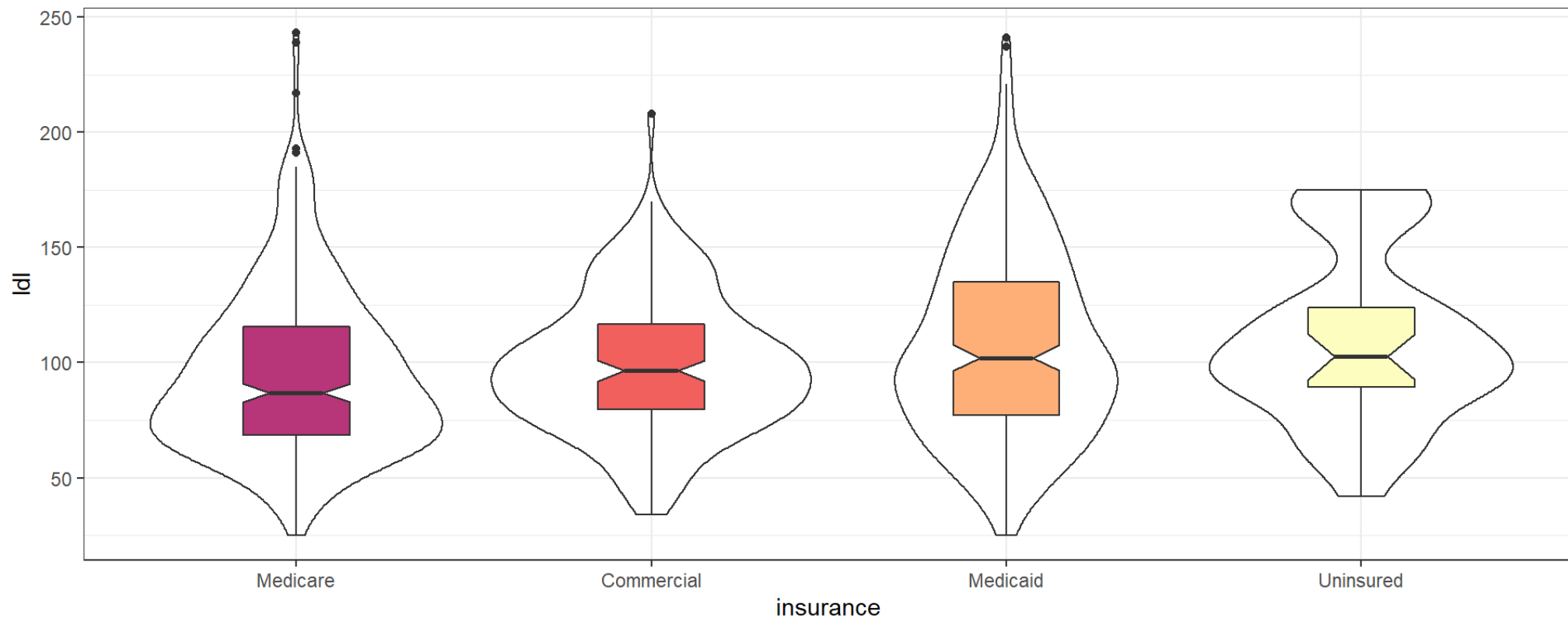
## LDL by Insurance

```
1 mosaic::favstats(ldl ~ insurance, data = dm1000)
```

	insurance	min	Q1	median	Q3	max	mean	sd	n	missing
1	Medicare	25	68.50	87.0	115.5	243	95.32958	35.91871	355	77
2	Commercial	34	79.75	96.5	116.5	208	98.94643	29.67796	168	28
3	Medicaid	25	77.00	102.0	135.0	241	108.00372	41.31084	269	61
4	Uninsured	42	89.50	102.5	124.0	175	109.13333	36.57560	30	12

# LDL by Insurance

```
1 ggplot(data = dm1000, aes(x = insurance, y = ldl)) +  
2   geom_violin() +  
3   geom_boxplot(aes(fill = insurance), width = 0.3, notch = TRUE) +  
4   scale_fill_viridis_d(begin = 0.5, option = "A") +  
5   guides(fill = "none")
```





# Visualizing Categorical Data in **dm1000**

# Categorical Variables from **dm1000**

```
1 dm_cat <- dm1000 |>
2   select(subject, sex, residence, insurance,
3           tobacco, race_ethnicity, statin, eye_exam)
4
5 dm_cat
```

# A tibble: 1,000 × 8

	subject	sex	residence	insurance	tobacco	race_ethnicity	statin
eye_e... <sup>1</sup>	<chr>	<fct>	<fct>	<fct>	<fct>	<fct>	<dbl>
<dbl>							

1	M-0001	Female	Suburbs	Medicaid	Current	Non-Hispanic Black	1
1							
2	M-0002	Male	Cleveland	Commercial	Never	Non-Hispanic Black	0
1							
3	M-0003	Female	Cleveland	Medicare	Former	Hispanic or Latinx	1
0							
4	M-0004	Female	Cleveland	Medicaid	Never	Hispanic or Latinx	1
1							
5	M-0005	Female	Suburbs	Medicare	Never	Non-Hispanic White	1
0							
6	M-0006	Male	Cleveland	Medicaid	Current	Hispanic or Latinx	1

# Codebook for `dm_cat`

- `sex` = Female or Male (no missing data)
- `insurance` = Medicare, Commercial, Medicaid, Uninsured
- `eye_exam` = 1 for eye examination in past year, else 0
- `statin` = 1 statin prescription in past year, else 0
- `race_ethnicity` = 4 levels (Hispanic or Latinx, Non-Hispanic White, Non-Hispanic Black, Non-Hispanic Asian)
- `residence` = 2 levels (Suburbs, Cleveland), some NA
- `tobacco` = 3 levels (Current, Former, Never), some NA

# summary() check of levels

```
1 dm_cat |> select(-subject) |> summary()
```

sex	residence	insurance	tobacco
Female:550	Suburbs :371	Medicare :432	Current:274
Male :450	Cleveland:601	Commercial:196	Never :343
	NA's : 28	Medicaid :330	Former :367
		Uninsured : 42	NA's : 16

race_ethnicity	statin	eye_exam
Non-Hispanic Black:533	Min. :0.000	Min. :0.000
Hispanic or Latinx: 91	1st Qu.:1.000	1st Qu.:0.000
Non-Hispanic White:356	Median :1.000	Median :1.000
Non-Hispanic Asian: 20	Mean :0.758	Mean :0.562
	3rd Qu.:1.000	3rd Qu.:1.000
	Max. :1.000	Max. :1.000

- Do we need to treat `statin` and `eye_exam` differently?

# Using **count** to create a tibble of counts

```
1 dm_cat |> count(tobacco)
```

```
# A tibble: 4 × 2
```

	tobacco	n
	<fct>	<int>
1	Current	274
2	Never	343
3	Former	367
4	<NA>	16

# Tabulating a categorical variable

```
1 dm_cat |> tabyl(tobacco) |>
2   adorn_pct_formatting() |>
3   adorn_totals()
```

tobacco	n	percent	valid_percent
Current	274	27.4%	27.8%
Never	343	34.3%	34.9%
Former	367	36.7%	37.3%
<NA>	16	1.6%	-
Total	1000	-	-

- Does this order make sense?

# Changing Order of **tobacco** levels

```
1 dm_cat <- dm_cat |>
2   mutate(tobacco = fct_relevel(tobacco, "Current", "Former"))
3
4 dm_cat |> tabyl(tobacco)
```

tobacco	n	percent	valid_percent
Current	274	0.274	0.2784553
Former	367	0.367	0.3729675
Never	343	0.343	0.3485772
<NA>	16	0.016	NA

- Does this order make more sense?
- **fct\_relevel()** is one of many useful tools in **forcats**.

# Using **gt** to make a table prettier

```
1 dm_cat |>
2   tabyl(tobacco) |>
3   adorn_pct_formatting() |>
4   adorn_totals() |>
5   gt() |>
6   tab_header(title = "Tobacco Status from dm1000")
```

Tobacco Status from dm1000			
tobacco	n	percent	valid_percent
Current	274	27.4%	27.8%
Former	367	36.7%	37.3%
Never	343	34.3%	34.9%
NA	16	1.6%	-
Total	1000	-	-

- <https://gt.rstudio.com/> provides an amazing array of options.



# gtExtras lets us build 538-style tables

```

1 dm_cat |>
2   tabyl(tobacco) |>
3   adorn_pct_formatting() |>
4   adorn_totals() |>
5   gt() |>
6   gt_theme_538() |>
7   tab_header(title = "Table styled like 538")

```

Table styled like 538

TOBACCO	N	PERCENT	VALID_PERCENT
Current	274	27.4%	27.8%
Former	367	36.7%	37.3%
Never	343	34.3%	34.9%
NA	16	1.6%	-
Total	1000	-	-

# gtExtras lets us build NYT-style tables

```
1 dm_cat |> tabyl(tobacco) |> adorn_pct_formatting() |> adorn_totals() |>
2   gt() |>
3   gt_theme_nytimes() |>
4   tab_header(title = "Table styled like the New York Times")
```

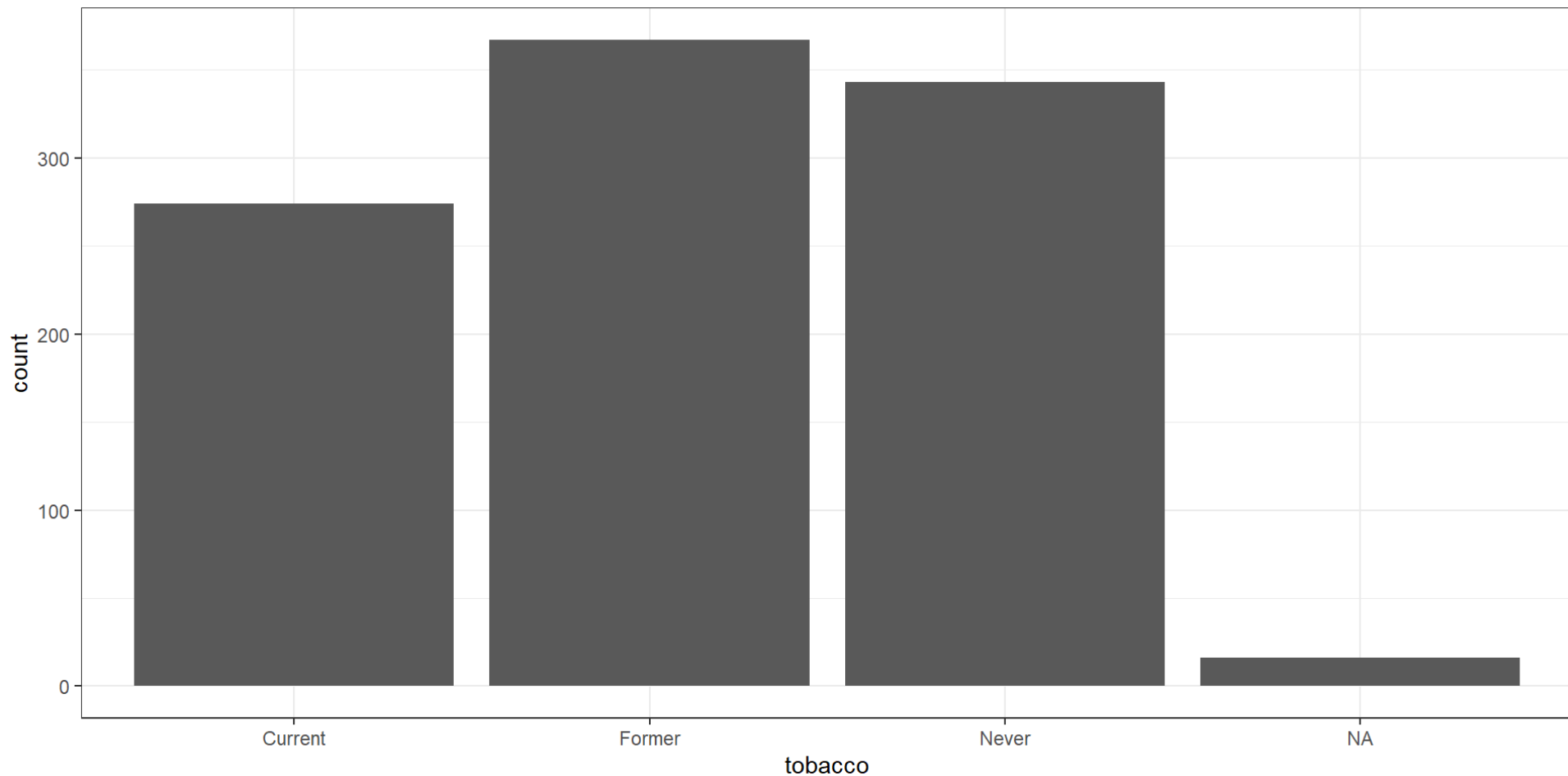
**Table styled like the New York Times**

TOBACCO	N	PERCENT	VALID_PERCENT
Current	274	27.4%	27.8%
Former	367	36.7%	37.3%
Never	343	34.3%	34.9%
NA	16	1.6%	-
Total	1000	-	-

- There's also a `gt_theme_espn()` and several others.

# Using `geom_bar` to show a distribution

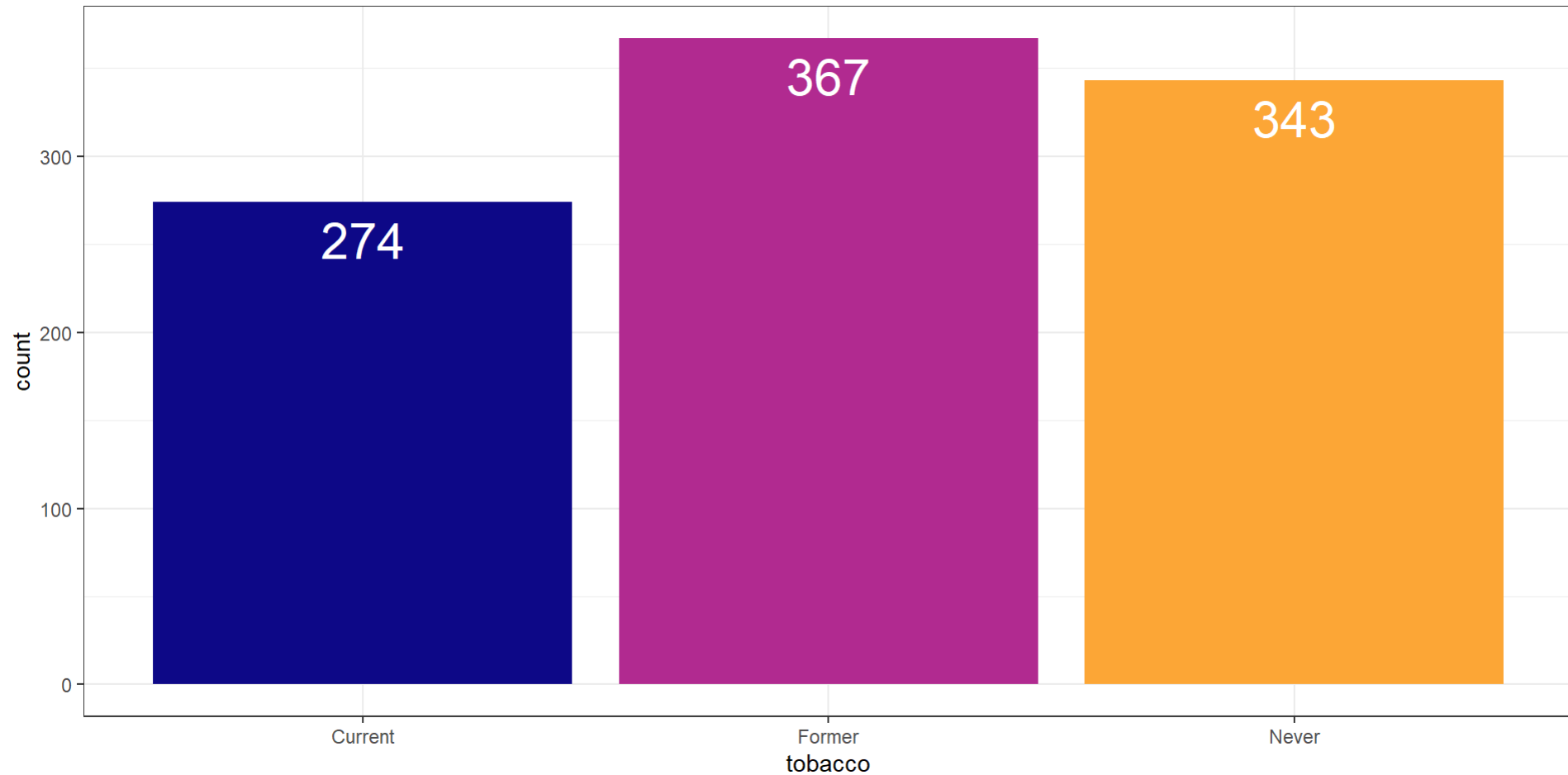
```
1 ggplot(dm_cat, aes(x = tobacco)) +  
2   geom_bar()
```



# Augmenting the `geom_bar` result

```
1 tempdat <- dm_cat |> filter(complete.cases(tobacco))
2
3 ggplot(data = tempdat, aes(x = tobacco, fill = tobacco)) +
4   geom_bar() +
5   geom_text(aes(label = ..count..), stat = "count",
6             vjust = 1.5, col = "white", size = 8) +
7   scale_fill_viridis_d(option = "C", end = 0.8) +
8   guides(fill = "none")
```

# Augmenting the `geom_bar` result



# Using **count** to create a tibble of counts

```
1 dm_cat |>
2   count(statin, tobacco)
```

```
# A tibble: 8 × 3
  statin tobacco      n
  <dbl> <fct>    <int>
1      0 Current     67
2      0 Former     80
3      0 Never      92
4      0 <NA>        3
5      1 Current    207
6      1 Former    287
7      1 Never    251
8      1 <NA>     13
```

```
1 dm_cat |>
2   count(insurance, residence)
```

```
# A tibble: 12 × 3
  insurance residence      n
  <fct>      <fct>    <int>
1 Medicare  Suburbs     163
2 Medicare  Cleveland    259
3 Medicare  <NA>         10
4 Commercial Suburbs      75
5 Commercial Cleveland   119
6 Commercial <NA>          2
7 Medicaid  Suburbs     114
8 Medicaid  Cleveland   201
9 Medicaid  <NA>        15
10 Uninsured Suburbs      19
11 Uninsured Cleveland    22
12 Uninsured <NA>          1
```

# Cross-Tabulations

```
1 dm_cat |> tabyl(insurance, residence)
```

```
insurance Suburbs Cleveland NA_
Medicare   163          259  10
Commercial  75          119   2
Medicaid  114          201  15
Uninsured   19           22   1
```

```
1 dm_cat |>
2   filter(complete.cases(insurance, residence)) |>
3   tabyl(insurance, residence) |>
4   adorn_totals(where = c("row", "col")) |>
5   gt()
```

insurance	Suburbs	Cleveland	Total
Medicare	163	259	422
Commercial	75	119	194
Medicaid	114	201	315
Uninsured	19	22	41
Total	371	601	972

# Were suburban residents more likely to have a statin prescription?

```
1 dm_cat |>
2   filter(complete.cases(statin, residence)) |>
3   tabyl(residence, statin)
```

residence	0	1
Suburbs	77	294
Cleveland	157	444



# Revise statin order, add percentages

```
1 dm_cat |> filter(complete.cases(statin, residence)) |>
2   mutate(statin = fct_relevel(factor(statin), "1", "0")) |>
3   tabyl(residence, statin)
```

residence	1	0
Suburbs	294	77
Cleveland	444	157

```
1 dm_cat |> filter(complete.cases(statin, residence)) |>
2   mutate(statin = fct_relevel(factor(statin), "1", "0")) |>
3   tabyl(residence, statin) |>
4   adorn_percentages(denom = "row") |>
5   adorn_pct_formatting()
```

residence	1	0
Suburbs	79.2%	20.8%
Cleveland	73.9%	26.1%

# Create using **table** instead

```
1 tempdat1 <- dm_cat |>
2   filter(complete.cases(statin, residence)) |>
3   mutate(statin = fct_relevel(factor(statin), "1", "0"))
4
5 tab1 <- table(tempdat1$residence, tempdat1$statin)
6
7 tab1
```

	1	0
Suburbs	294	77
Cleveland	444	157

# Assess 2x2 table

```
1 twoby2(tab1) # twoby2() is part of the Epi package
```

2 by 2 table analysis:

-----  
 Outcome : 1  
 Comparing : Suburbs vs. Cleveland

	1	0	P(1)	95% conf. interval
Suburbs	294	77	0.7925	0.7482 0.8307
Cleveland	444	157	0.7388	0.7022 0.7723

	95% conf. interval
Relative Risk: 1.0727	0.9996 1.1510
Sample Odds Ratio: 1.3501	0.9903 1.8407
Conditional MLE Odds Ratio: 1.3497	0.9805 1.8679
Probability difference: 0.0537	-0.0018 0.1065

Forest Plot: Relative Risk = 1.0628

# A three-by-four two-way table

```
1 dm_cat |> filter(complete.cases(tobacco, insurance)) |>
2   tabyl(tobacco, insurance) |>
3   adorn_totals(where = c("row", "col"))
```

tobacco	Medicare	Commercial	Medicaid	Uninsured	Total
Current	99	44	118	13	274
Former	183	70	103	11	367
Never	140	80	105	18	343
Total	422	194	326	42	984

- 3 rows, 4 columns: hence, this is a 3 x 4 table
- It's a two-way table, because we are studying the association of two variables (**tobacco** and **insurance**)
- Compare insurance percentages by tobacco group?

# Insurance rates by tobacco group

```
1 dm_cat |> filter(complete.cases(tobacco, insurance)) |>
2   tabyl(tobacco, insurance) |>
3   adorn_percentages(denominator = "row") |>
4   adorn_totals(where = "col") |> kbl(digits = 3)
```

tobacco	Medicare	Commercial	Medicaid	Uninsured	Total
Current	0.361	0.161	0.431	0.047	1
Former	0.499	0.191	0.281	0.030	1
Never	0.408	0.233	0.306	0.052	1

- These are **proportions** and not percentages.
- Proportions fall between 0 and 1: multiply by 100 for percentages.

# Insurance rates by tobacco group?

```
1 tempdat2 <- dm_cat |>
2   filter(complete.cases(tobacco, insurance))
3
4 tab2 <- table(tempdat2$tobacco, tempdat2$insurance)
5
6 tab2
```

	Medicare	Commercial	Medicaid	Uninsured
Current	99	44	118	13
Former	183	70	103	11
Never	140	80	105	18

```
1 chisq.test(tab2)
```

Pearson's Chi-squared test

data: tab2

X-squared = 25.592, df = 6, p-value = 0.0002651

# Using **count** for three variables

```
1 dm_cat |> count(sex, statin, residence)
```

```
# A tibble: 12 × 4
```

	sex <fct>	statin <dbl>	residence <fct>	n <int>
1	Female	0	Suburbs	42
2	Female	0	Cleveland	87
3	Female	0	<NA>	4
4	Female	1	Suburbs	160
5	Female	1	Cleveland	245
6	Female	1	<NA>	12
7	Male	0	Suburbs	35
8	Male	0	Cleveland	70
9	Male	0	<NA>	4
10	Male	1	Suburbs	134
11	Male	1	Cleveland	199
12	Male	1	<NA>	8

# A three-way table via `tabyl`

```
1 dm_cat |>
2   filter(complete.cases(statin, residence, sex)) |>
3   tabyl(statin, residence, sex) |>
4   adorn_totals(where = c("row", "col")) |>
5   adorn_title()
```

\$Female

	residence		
statin	Suburbs	Cleveland	Total
0	42	87	129
1	160	245	405
Total	202	332	534

\$Male

	residence		
statin	Suburbs	Cleveland	Total
0	35	70	105
1	134	199	333
Total	169	269	438



# Flattening a three-way table

```
1 ftable(dm_cat$sex, dm_cat$residence, dm_cat$statin)
```

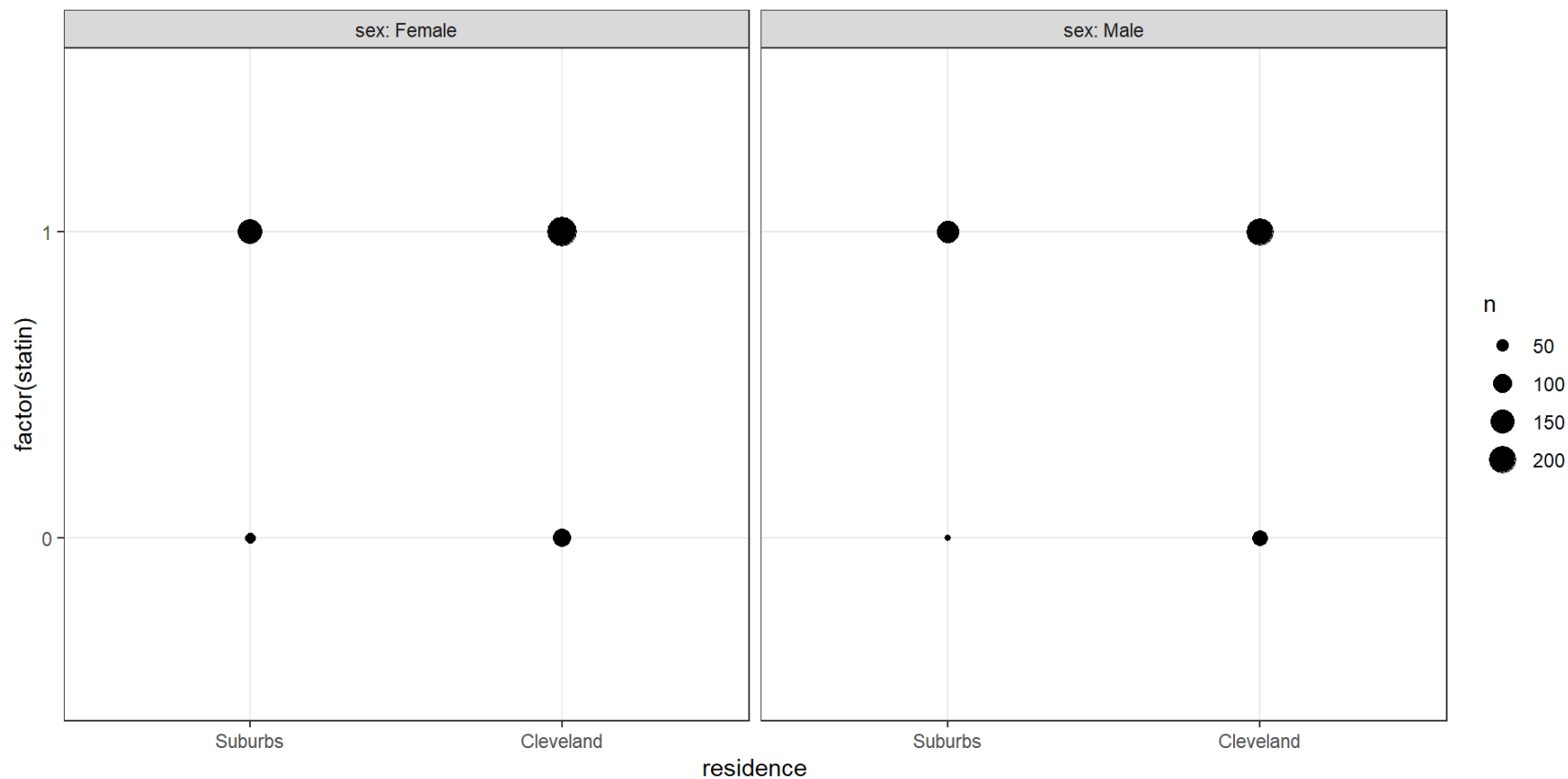
0 1

Female	Suburbs	42	160
	Cleveland	87	245
Male	Suburbs	35	134
	Cleveland	70	199

- Note that `ftable()` excludes the missing `residence` values by default.

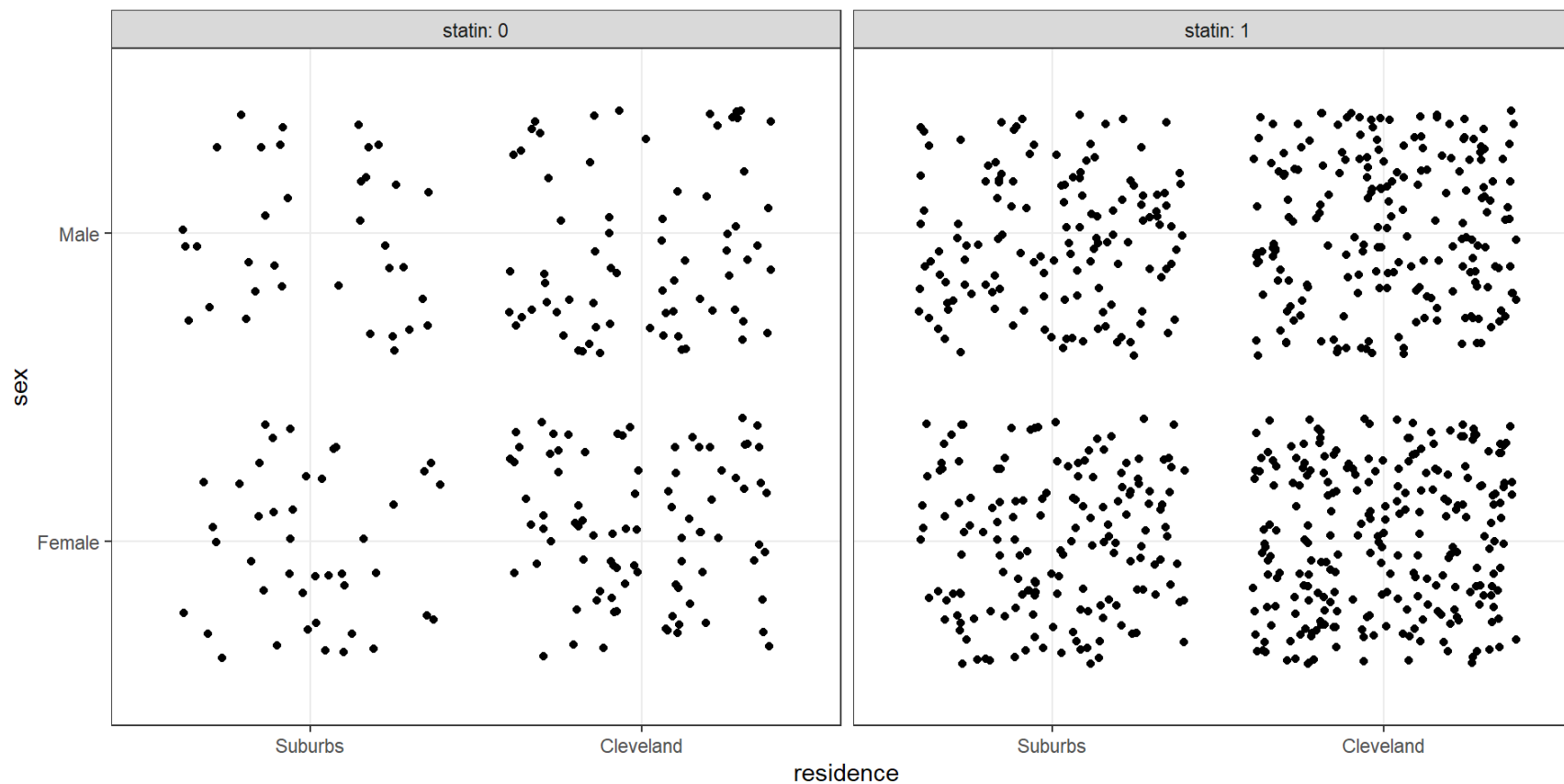
# Plotting a 3-Way Table (Counts)

```
1 ggplot(data = filter(dm_cat, complete.cases(residence)),
2       aes(x = residence, y = factor(statin))) +
3   geom_count() +
4   facet_wrap(~ sex, labeller = "label_both")
```



# Plotting a 3-Way Table (Jitter)

```
1 ggplot(data = filter(dm_cat, complete.cases(residence)),
2       aes(x = residence, y = sex)) +
3   geom_jitter() +
4   facet_wrap(~ statin, labeller = "label_both")
```



# Multi-categorical 3-Way Table

```
1 dm_cat |>
2   filter(complete.cases(insurance, race_ethnicity, tobacco)) |>
3   tabyl(race_ethnicity, insurance, tobacco) |>
4   adorn_totals(where = c("row", "col")) |>
5   adorn_title()
```

\$Current

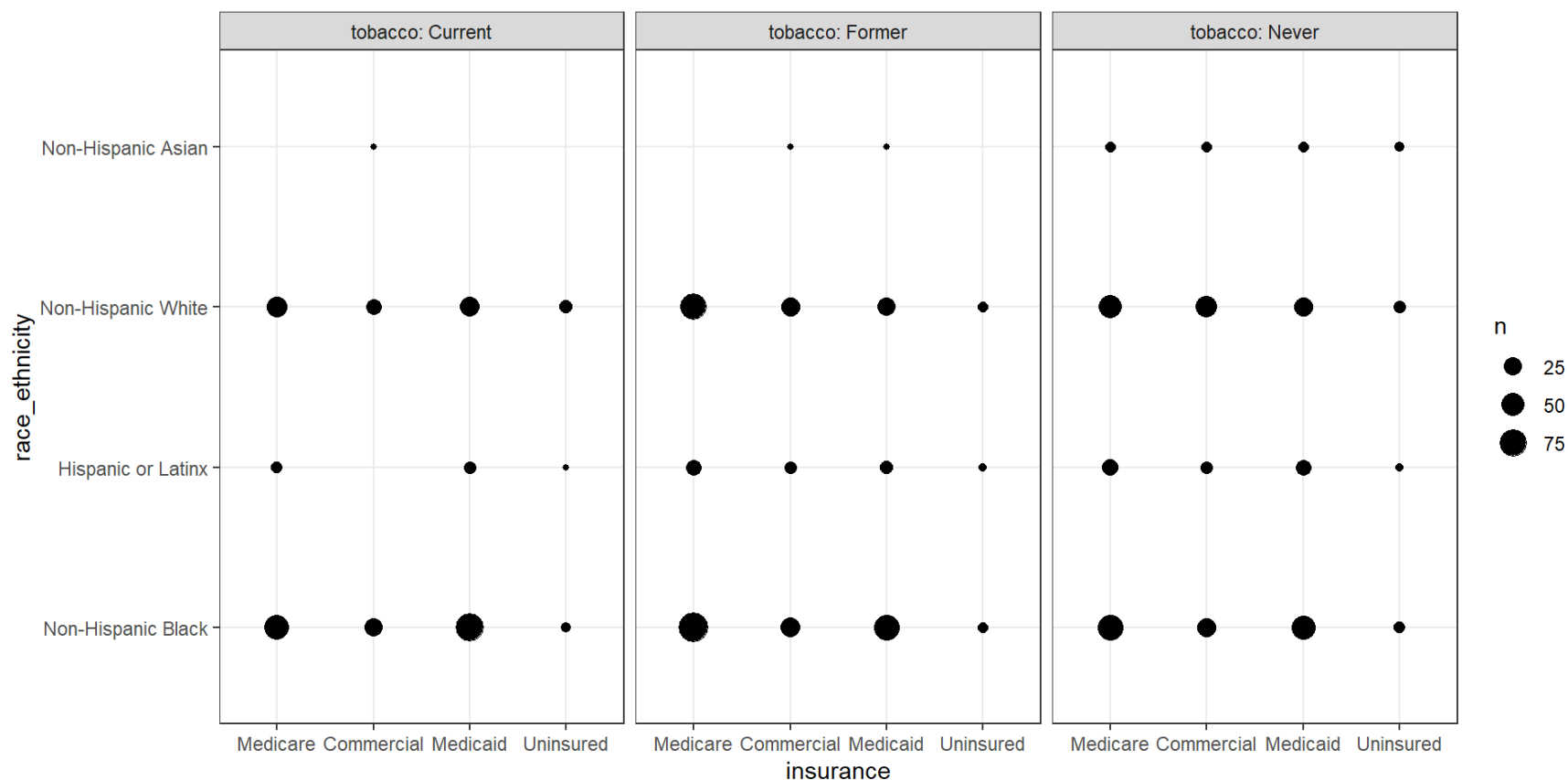
	insurance				
race_ethnicity	Medicare	Commercial	Medicaid	Uninsured	Total
Non-Hispanic Black	58	26	80	3	167
Hispanic or Latinx	6	0	7	1	14
Non-Hispanic White	35	17	31	9	92
Non-Hispanic Asian	0	1	0	0	1
Total	99	44	118	13	274

\$Former

	insurance				
race_ethnicity	Medicare	Commercial	Medicaid	Uninsured	Total
Non-Hispanic Black	96	34	66	5	201
Hispanic or Latinx	15	7	9	2	33
Non-Hispanic White	72	28	27	4	131
Non-Hispanic Asian	0	1	1	0	2

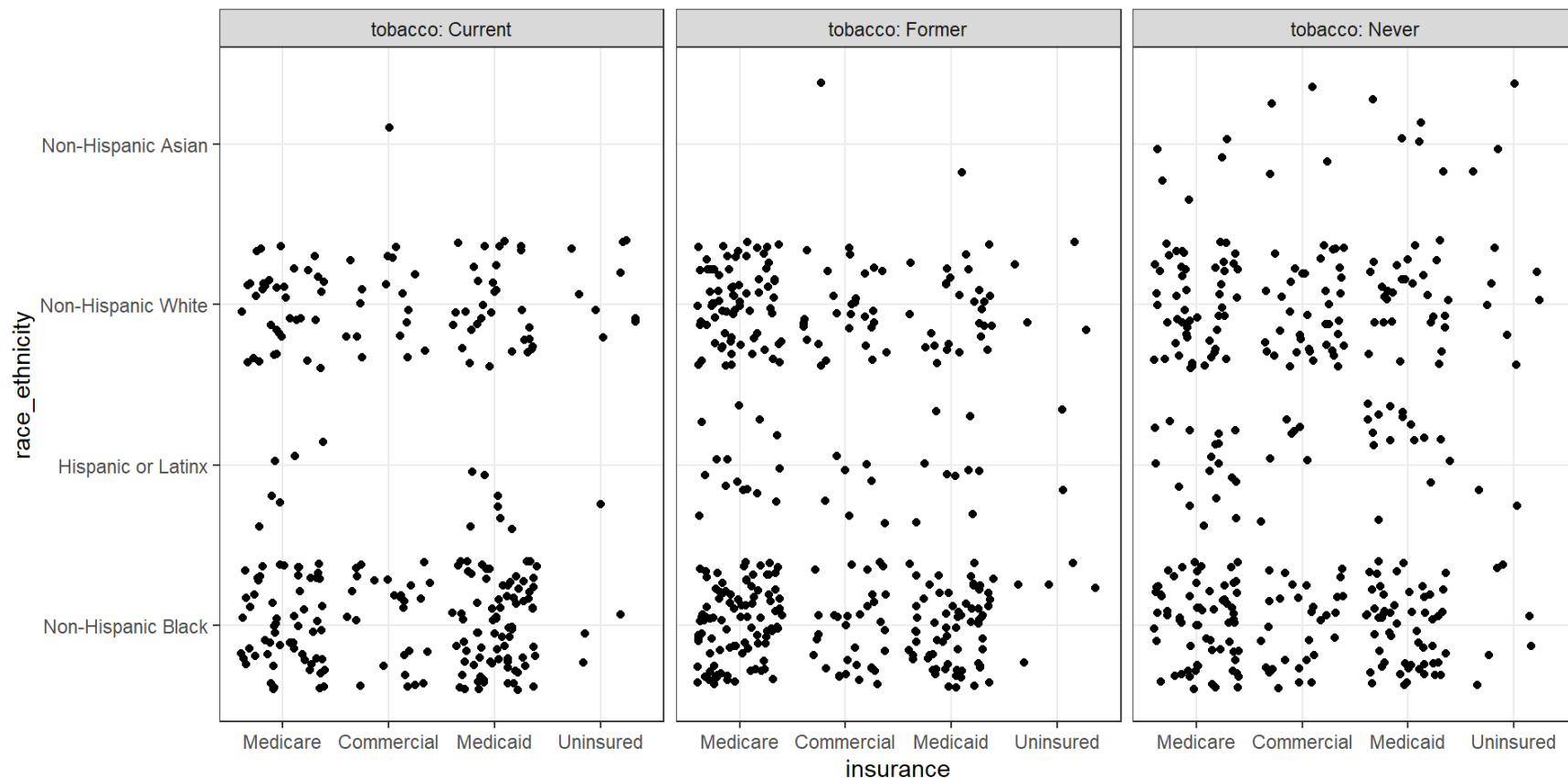
# Multi-categorical 3-Way Counts

```
1 ggplot(data = filter(dm_cat, complete.cases(tobacco)),
2       aes(x = insurance, y = race_ethnicity)) +
3   geom_count() +
4   facet_wrap(~ tobacco, labeller = "label_both")
```



# Multi-categorical 3-Way Jitter Plot

```
1 ggplot(data = filter(dm_cat, complete.cases(tobacco)),
2       aes(x = insurance, y = race_ethnicity)) +
3   geom_jitter() +
4   facet_wrap(~ tobacco, labeller = "label_both")
```



# RStudio Cheat Sheets

<https://www.rstudio.com/resources/cheatsheets/>

- Data visualization with ggplot2 Cheatsheet shown on next two slides...

Other cheatsheets I use a lot include:

- Data transformation with dplyr
- Data import with readr, readxl, and googlesheets4
- Factors with forcats
- Dynamic documents with rmarkdown

# Data visualization with ggplot2 : : CHEAT SHEET



## Basics

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same components: a data set, a coordinate system, and geoms—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (aesthetics) like size, color, and x and y locations.



Complete the template below to build a graph.

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
  stat = <STAT>, position = <POSITION>) +
  <COORDINATE_FUNCTION> +
  <FACET_FUNCTION> +
  <SCALE_FUNCTION> +
  <THEME_FUNCTION>
```

required

Not required, sensible defaults supplied

`ggplot(data = mpg, aes(x = cty, y = hwy))` Begins a plot that you finish by adding layers to. Add one geom function per layer.

`last_plot()` Returns the last plot.

`ggsave("plot.png", width = 5, height = 5)` Saves last plot as 5" x 5" file named "plot.png" in working directory. Matches file type to file extension.

## Aes

Common aesthetic values.

color and fill - string ("red", "#RRGGBB")  
linetype - integer or string (0 = "blank", 1 = "solid", 2 = "dashed", 3 = "dotted", 4 = "dotted", 5 = "longdash", 6 = "twodash")

lineend - string ("round", "butt", or "square")

linejoin - string ("round", "mitre", or "bevel")

size - integer (line width in mm)

shape - integer/shape name or a single character ("a")



## Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

### GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

**a + geom\_blank()** and **a + expand\_limits()**  
Ensure limits include values across all plots.

**b + geom\_curve()** `aes(yend = lat + 1, xend = long + 1, curvature = 1)` - x, y, xend, yend, alpha, angle, color, curvature, linetype, size

**a + geom\_path()** `lineend = "butt", linejoin = "round", linemitre = 1` - x, y, alpha, color, group, linetype, size

**a + geom\_polygon()** `aes(alpha = 50)` - x, y, alpha, color, fill, group, subgroup, linetype, size

**b + geom\_rect()** `aes(xmin = long, ymin = lat, xmax = long + 1, ymax = lat + 1)` - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

**a + geom\_ribbon()** `aes(ymin = unemploy - 900, ymax = unemploy + 900)` - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

### LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

**b + geom\_abline()** `aes(intercept = 0, slope = 1)`  
**b + geom\_hline()** `aes(yintercept = lat)`  
**b + geom\_vline()** `aes(xintercept = long)`

**b + geom\_segment()** `aes(yend = lat + 1, xend = long + 1)`  
**b + geom\_spoke()** `aes(angle = 1:1155, radius = 1)`

### ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

**c + geom\_area()** `stat = "bin"` - x, y, alpha, color, fill, linetype, size

**c + geom\_density()** `kernel = "gaussian"` - x, y, alpha, color, fill, group, linetype, size, weight

**c + geom\_dotplot()** - x, y, alpha, color, fill

**c + geom\_freqpoly()** - x, y, alpha, color, group, linetype, size

**c + geom\_histogram()** `binwidth = 5` - x, y, alpha, color, fill, linetype, size, weight

**c2 + geom\_qq()** `aes(sample = hwy)` - x, y, alpha, color, fill, linetype, size, weight

### discrete

```
d <- ggplot(mpg, aes(f))
```

**d + geom\_bar()** - x, alpha, color, fill, linetype, size, weight

### TWO VARIABLES

both continuous

```
e <- ggplot(mpg, aes(cty, hwy))
```

**e + geom\_label()** `aes(label = cty), nudge_x = 1, nudge_y = 1` - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

**e + geom\_point()** - x, y, alpha, color, fill, shape, size, stroke

**e + geom\_quantile()** - x, y, alpha, color, group, linetype, size, weight

**e + geom\_rug()** `sides = "bl"` - x, y, alpha, color, linetype, size

**e + geom\_smooth()** `method = lm` - x, y, alpha, color, fill, group, linetype, size, weight

**e + geom\_text()** `aes(label = cty), nudge_x = 1, nudge_y = 1` - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

### one discrete, one continuous

```
f <- ggplot(mpg, aes(class, hwy))
```

**f + geom\_col()** - x, y, alpha, color, fill, group, linetype, size

**f + geom\_boxplot()** - x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

**f + geom\_dotplot()** `binaxis = "y", stackdir = "center"` - x, y, alpha, color, fill, group

**f + geom\_violin()** `scale = "area"` - x, y, alpha, color, fill, group, linetype, size, weight

### both discrete

```
g <- ggplot(diamonds, aes(cut, color))
```

**g + geom\_count()** - x, y, alpha, color, fill, shape, size, stroke

**e + geom\_jitter()** `height = 2, width = 2` - x, y, alpha, color, fill, shape, size

### THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

**l + geom\_contour()** `aes(z = z)` - x, y, z, alpha, color, group, linetype, size, weight

**l + geom\_contour\_filled()** `aes(fill = z)` - x, y, alpha, color, fill, group, linetype, size, subgroup

### continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

**h + geom\_bin2d()** `binwidth = c(0.25, 500)` - x, y, alpha, color, fill, linetype, size, weight

**h + geom\_density\_2d()** - x, y, alpha, color, group, linetype, size

**h + geom\_hex()** - x, y, alpha, color, fill, size

### continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

**i + geom\_area()** - x, y, alpha, color, fill, linetype, size

**i + geom\_line()** - x, y, alpha, color, group, linetype, size

**i + geom\_step()** `direction = "hv"` - x, y, alpha, color, group, linetype, size

### visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4.5, se = 1.2)
j <- ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))
```

**j + geom\_crossbar()** `fatten = 2` - x, y, ymax, ymin, alpha, color, fill, group, linetype, size

**j + geom\_errorbar()** - x, ymax, ymin, alpha, color, group, linetype, size, width  
Also `geom_errorbarh()`.

**j + geom\_linerange()** - x, ymin, ymax, alpha, color, group, linetype, size

**j + geom\_pointrange()** - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

### maps

```
data <- data.frame(murder = USArrests$Murder, state = tolower(rownames(USArrests)))
```

```
map <- map_data("state")
```

```
k <- ggplot(data, aes(fill = murder))
```

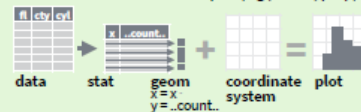
**k + geom\_map()** `aes(map_id = state), map = map` + `expand_limits(x = map$long, y = map$lat)`  
map\_id, alpha, color, fill, linetype, size



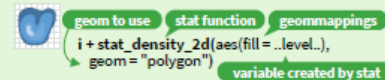
## Stats

An alternative way to build a layer.

A stat builds new variables to plot (e.g., count, prop).



Visualize a stat by changing the default stat of a geom function, `geom_bar(stat="count")` or by using a stat function, `stat_count(geom="bar")`, which calls a default geom to make a layer (equivalent to a geom function). Use `..name..` syntax to map stat variables to aesthetics.



```
c + stat_bin(binwidth = 1, boundary = 10)
x, y | ..count.., ..density..
c + stat_count(width = 1) x, y | ..count.., ..prop..
c + stat_density(adjust = 1, kernel = "gaussian")
x, y | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..
e + stat_bin_hex(bins = 30) x, y, fill | ..count.., ..density..
e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..
e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) x, y, z, order | ..level..
l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..
l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5)
x, y | ..lower.., ..middle.., ..upper.., ..width.., ..ymin.., ..ymax..
f + stat_ydensity(kernel = "gaussian", scale = "area") x, y
| ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

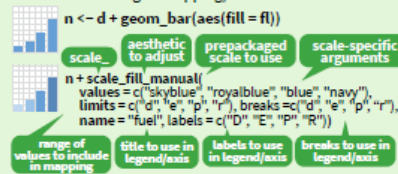
e + stat_ecdf(n = 40) x, y | ..x.., ..y..
e + stat_quantile(quantiles = c(0.1, 0.9),
formula = y ~ log(x), method = "rq") x, y | ..quantile..
e + stat_smooth(method = "lm", formula = y ~ x, se = T,
level = 0.95) x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + xlim(-5, 5) + stat_function(fun = dnorm,
n = 20, geom = "point") x | ..x.., ..y..
ggplot() + stat_qq(aes(sample = 1:100))
x, y, sample | ..sample.., ..theoretical..
e + stat_sum() x, y, size | ..n.., ..prop..
e + stat_summary(fun.data = "mean_cl_boot")
h + stat_summary_bin(fun = "mean", geom = "bar")
e + stat_identity()
e + stat_unique()
```

## Scales

Override defaults with scales package.

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.



### GENERAL PURPOSE SCALES

Use with most aesthetics

```
scale_*_continuous() - Map cont' values to visual ones.
scale_*_discrete() - Map discrete values to visual ones.
scale_*_binned() - Map continuous values to discrete bins.
scale_*_identity() - Use data values as visual ones.
scale_*_manual(values = c()) - Map discrete values to manually chosen visual ones.
scale_*_date(date_labels = "%m/%d") - Treat data values as dates.
date_breaks = "2 weeks"
scale_*_datetime() - Treat data values as date times.
Same as scale_*_date(). See ?strptime for label formats.
```

### X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

```
scale_x_log10() - Plot x on log10 scale.
scale_x_reverse() - Reverse the direction of the x axis.
scale_x_sqrt() - Plot x on square root scale.
```

### COLOR AND FILL SCALES (DISCRETE)

```
n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()
n + scale_fill_grey(start = 0.2,
end = 0.8, na.value = "red")
```

### COLOR AND FILL SCALES (CONTINUOUS)

```
o <- c + geom_dotplot(aes(fill = ..x..))
o + scale_fill_distiller(palette = "Blues")
o + scale_fill_gradient(low = "red", high = "yellow")
o + scale_fill_gradient2(low = "red", high = "blue",
mid = "white", midpoint = 25)
o + scale_fill_gradientn(colors = topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(),
cm.colors(), RColorBrewer::brewer.pal()
```

### SHAPE AND SIZE SCALES

```
p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))
p + scale_size_manual(values = c(1:6))
p + scale_size_area(max_size = 6)
```

## Coordinate Systems

```
r <- d + geom_bar()
r + coord_cartesian(xlim = c(0, 5)) - xlim, ylim
The default cartesian coordinate system.
```

```
r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim - Cartesian coordinates with
fixed aspect ratio between x and y units.
```

```
ggplot(mpg, aes(y = fl)) + geom_bar()
Flip cartesian coordinates by switching
x and y aesthetic mappings.
```

```
r + coord_polar(theta = "x", direction = 1)
theta, start, direction - Polar coordinates.
```

```
r + coord_trans(y = "sqrt") - x, y, xlim, ylim
Transformed cartesian coordinates. Set xtrans
and ytrans to the name of a window function.
```

```
π + coord_quickmap()
π + coord_map(projection = "ortho", orientation
= c(41, -74, 0)) - projection, xlim, ylim
Map projections from the mapproj package
(mercator (default), azequalarea, lagrange, etc.).
```

## Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

```
s <- ggplot(mpg, aes(fl, fill = drv))
```

```
s + geom_bar(position = "dodge")
Arrange elements side by side.
```

```
s + geom_bar(position = "fill")
Stack elements on top of one
another, normalize height.
```

```
e + geom_point(position = "jitter")
Add random noise to X and Y position of
each element to avoid overplotting.
```

```
e + geom_label(position = "nudge")
Nudge labels away from points.
```

```
s + geom_bar(position = "stack")
Stack elements on top of one another.
```

Each position adjustment can be recast as a function with manual width and height arguments:

```
s + geom_bar(position = position_dodge(width = 1))
```

## Themes

```
r + theme_bw()
White background
with grid lines.
```

```
r + theme_gray()
Grey background
(default theme).
```

```
r + theme_dark()
Dark for contrast.
```

```
r + theme_classic()
r + theme_light()
r + theme_linedraw()
r + theme_minimal()
r + theme_void()
```

```
r + theme() Customize aspects of the theme such
as axis, legend, panel, and facet properties.
r + ggtitle("Title") + theme(plot.title.position = "plot")
r + theme(panel.background = element_rect(fill = "blue"))
```

## Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

```
t <- ggplot(mpg, aes(cty, hwy)) + geom_point()
```

```
t + facet_grid(cols = vars(fl))
Facet into columns based on fl.
```

```
t + facet_grid(rows = vars(year))
Facet into rows based on year.
```

```
t + facet_grid(rows = vars(year), cols = vars(fl))
Facet into both rows and columns.
```

```
t + facet_wrap(vars(fl))
Wrap facets into a rectangular layout.
```

Set scales to let axis limits vary across facets.

```
t + facet_grid(rows = vars(drv), cols = vars(fl),
scales = "free")
x and y axis limits adjust to individual facets:
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust
```

Set labeller to adjust facet label:

```
t + facet_grid(cols = vars(fl), labeller = label_both)
```

```
fl: c fl: d fl: e fl: p fl: r
```

```
t + facet_grid(rows = vars(fl),
labeller = label_bquote(alpha ^ .(fl)))
```

```
αc αd αe αp αr
```

## Labels and Legends

Use `labs()` to label the elements of your plot.

```
t + labs(x = "New x axis label", y = "New y axis label",
title = "Add a title above the plot",
subtitle = "Add a subtitle below title",
caption = "Add a caption below plot",
alt = "Add alt text to the plot",
aes = "New <AES> legend title")
```

```
t + annotate(geom = "text", x = 8, y = 9, label = "A")
Places a geom with manually selected aesthetics.
```

```
p + guides(x = guide_axis(n.dodge = 2)) Avoid crowded or
overlapping labels with guide_axis(n.dodge or angle).
```

```
n + guides(fill = "none") Set legend type for each
aesthetic: colorbar, legend, or none (no legend).
```

```
n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right".
```

```
n + scale_fill_discrete(name = "Title",
labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.
```

## Zooming

Without clipping (preferred):

```
t + coord_cartesian(xlim = c(0, 100), ylim = c(10, 20))
```

With clipping (removes unseen data points):

```
t + xlim(0, 100) + ylim(10, 20)
```

```
t + scale_x_continuous(limits = c(0, 100)) +
scale_y_continuous(limits = c(0, 100))
```



RStudio® is a trademark of RStudio, PBC • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at [ggplot2.tidyverse.org](https://ggplot2.tidyverse.org) • ggplot2 3.3.5 • Updated: 2021-08

# Session Information

```
1 sessionInfo()
```

```
R version 4.2.1 (2022-06-23 ucrt)  
Platform: x86_64-w64-mingw32/x64 (64-bit)  
Running under: Windows 10 x64 (build 22000)
```

```
Matrix products: default
```

```
locale:
```

```
[1] LC_COLLATE=English_United States.utf8  
[2] LC_CTYPE=English_United States.utf8  
[3] LC_MONETARY=English_United States.utf8  
[4] LC_NUMERIC=C  
[5] LC_TIME=English_United States.utf8
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices  utils      datasets  methods   base
```