PSYCH 201B

Statistical Intuitions for Social Scientists

Modeling data IV

You can download these slides:
course website > Week 6 > Overview

02/12/2025

Today's Plan

1. First Half (together)
    - Review
    - Parameter inference
        - Via **analytic** uncertainty estimation
        - Via **bootstrapped** uncertainty resampling
        - Via **permuted** null resampling
2. Second Half (on your own)
    - Continue working on notebooks 1-3
    - Try notebook 4 if you're done already

Recap

Regression(s) are just "flavors" of GLM

$$y = X\beta + \epsilon$$

Simple (univariate) regression

$$Y_i = \beta_0 + \beta_1 X_{1i} + \epsilon_i$$

one predictor
one column wide

Multiple regression

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \ldots + \beta_p X_{pi} + \epsilon_i$$

many predictors
many columns wide

y

dependent variable
response variable
regressand
outcome/target
measurements
output variable

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

X

independent variables
explanatory variables
regressors
predictors/features
design matrix
input variables

$$\begin{bmatrix} \text{---} \ \vec{x}_1 \ \text{---} \\ \text{---} \ \vec{x}_2 \ \text{---} \\ \vdots \\ \text{---} \ \vec{x}_n \ \text{---} \end{bmatrix}$$

β

$$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

coefficients
betas
slopes
effects
parameter-
estimates

We estimate the *best* parameters using OLS

$$y = X\beta + \epsilon$$

Ordinary Least Squares (**OLS**):
an *analytic* (closed-form) equation for estimating betas that minimizes SSE!

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

y		X		β	
dependent variable	$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$	independent variables	$\begin{bmatrix} \text{—} \vec{x}_1 \text{—} \\ \text{—} \vec{x}_2 \text{—} \\ \vdots \\ \text{—} \vec{x}_n \text{—} \end{bmatrix}$	$\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$	coefficients
response variable		explanatory variables			betas
regressand		regressors			slopes
outcome/target		predictors/features			effects
measurements		design matrix			parameter-estimates
output variable		input variables			

What is OLS doing?

Calculating the *similarity* between each X and y after removing the covariance between Xs

*What is **unique contribution** of each x in predicting y?*

Matrix inverse

"undo" or "divide"

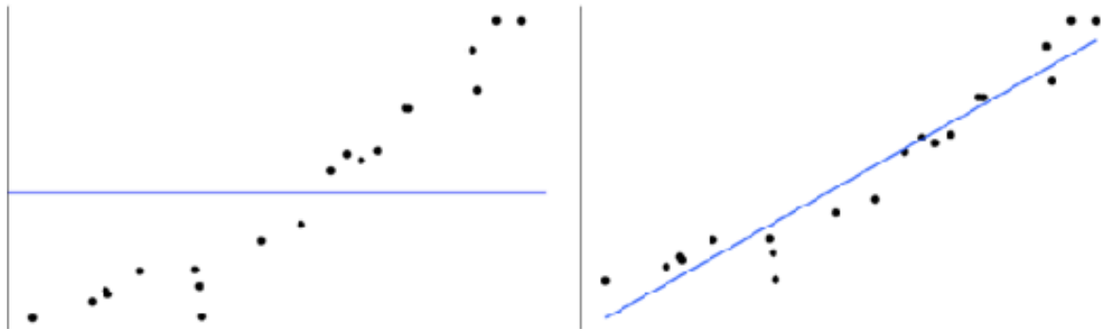
Similarity between predictors

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Similarity between predictors and outcome

We perform hypothesis tests by comparing model errors

Which model fits data best?



Compact model

model$_C$: $\hat{Balance}_i = \beta_0 + \epsilon$

Augmented model

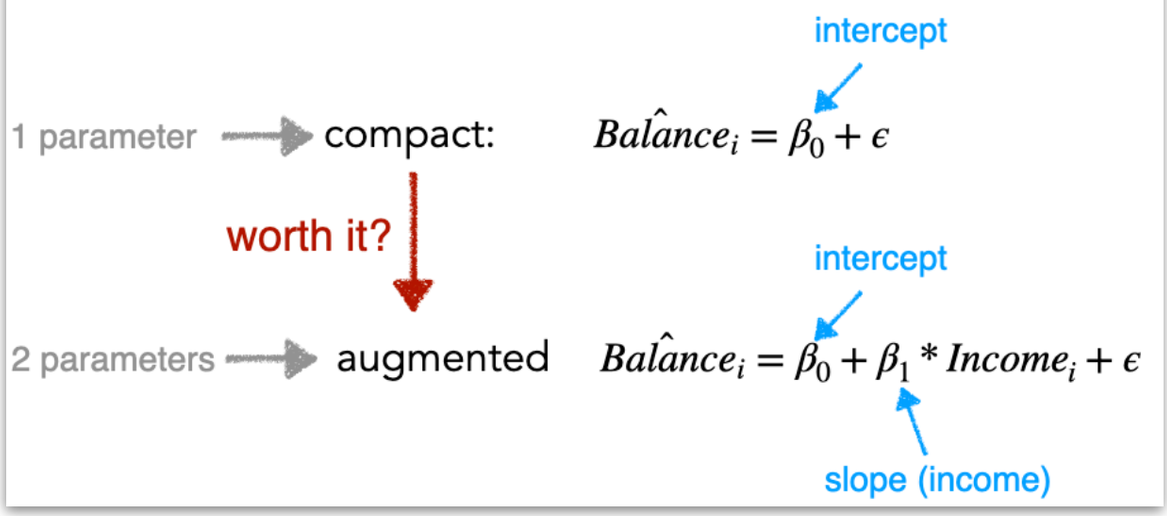
model$_A$: $\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$

$$\text{ERROR}(C) \geq \text{ERROR}(A)$$

Proportional reduction in error (PRE)

$$\text{PRE} = \frac{\text{ERROR}(C) - \text{ERROR}(A)}{\text{ERROR}(C)}$$

The **worth it?** question

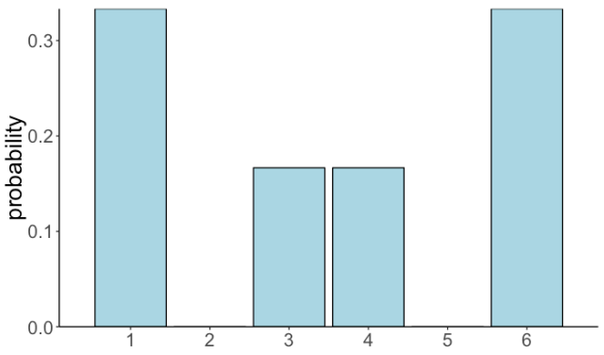


Decide whether it's worth it

- Do the additional parameter(s) reduce *enough* error?
- Have we used up *too many* degrees-of-freedom by estimating these additional parameter(s)?

How **uncertain** are we about our *estimate* of PRE?

We quantify uncertainty using sampling distributions



population

We don't know *true* PRE!

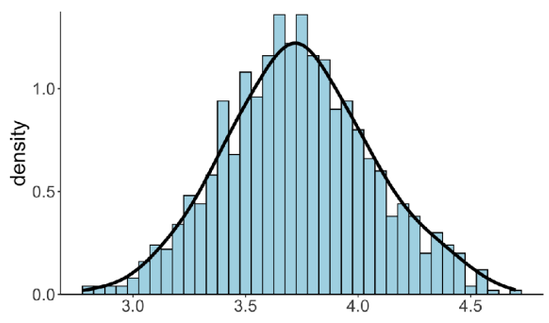
But it's what we *actually* care about

The uncertainty bridge

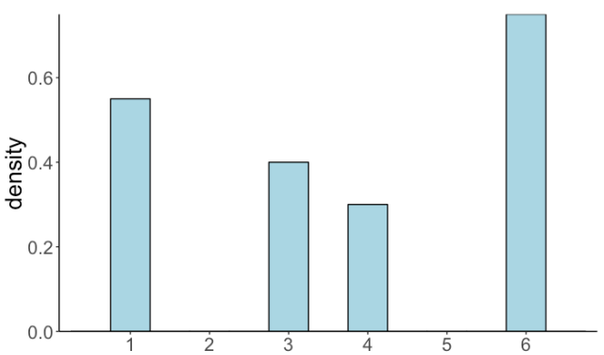
how an estimate varies between "samples"

samples = *analytic* (we look it up)

samples = *computational* (we resample)



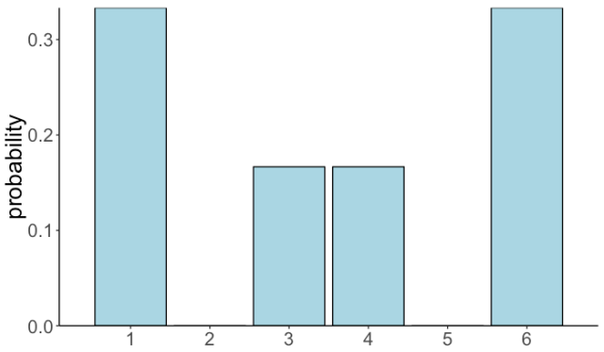
sampling distribution



sample distribution

What we used to *estimate* our PRE

We quantify uncertainty using sampling distributions



population

We don't know *true* PRE!

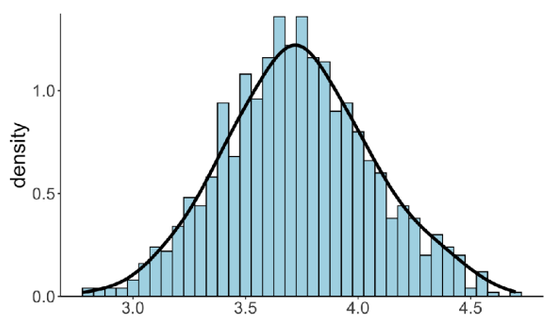
But it's what we *actually* care about

The uncertainty bridge

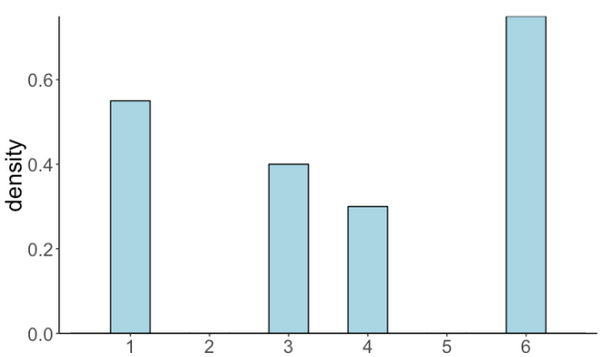
how an estimate varies between "samples"

samples = ***analytic* (we look it up)**

samples = *computational* (we resample)



sampling distribution



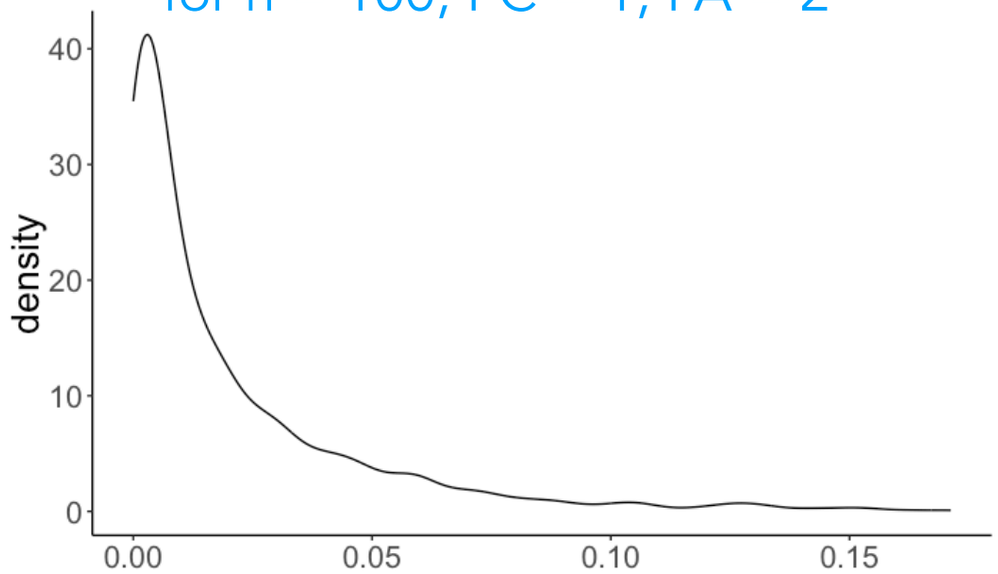
sample distribution

What we used to *estimate* our PRE

Analytic sampling distribution of PRE = F distribution

sampling distribution of PRE

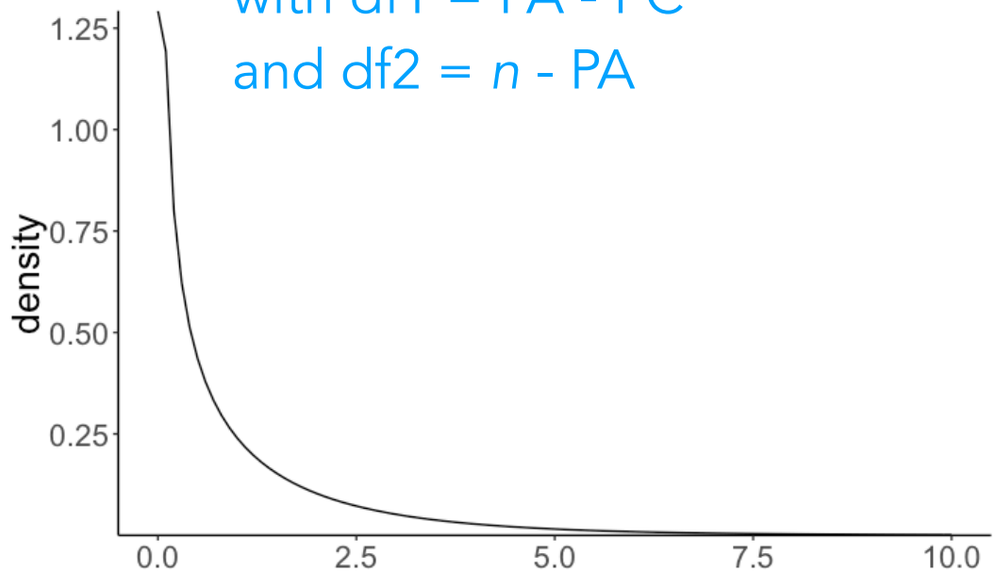
for n = 100, PC = 1, PA = 2



***F*(df1, df2) distribution**

with df1 = PA - PC

and df2 = *n* - PA




```
pa = 2 # intercept + slope
pc = 1 # intercept only
n = df.height # observations

# Difference in number of model degrees of freedom = difference in number of parameters
model_df = pa - pc

# Convert PRE -> F
F = (PRE / (pa - pc)) / ((1 - PRE) / (n - pa))

# Look up analytic p-value based on degrees of freedom (i.e. # params vs observations)
pval = 1 - f.cdf(F, model_df, error_df)
```

"look up" p-value

Analytic sampling distribution of PRE = F distribution

```python
# Or more easily
from statsmodels.stats.anova import anova_lm

anova_lm(compact_results, augmented_results)
```

✓ 0.0s

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	399.0	8.433991e+07	0.0	NaN	NaN	NaN
1	398.0	6.620874e+07	1.0	1.813117e+07	108.991715	1.030886e-22

```python
pa = 2 # intercept + slope
pc = 1 # intercept only
n = df.height # observations

# Difference in number of model degrees of freedom = difference in number of parameters
model_df = pa - pc

# Convert PRE -> F
F = (PRE / (pa - pc)) / ((1 - PRE) / (n - pa))

# Look up analytic p-value based on degrees of freedom (i.e. # params vs observations)
pval = 1 - f.cdf(F, model_df, error_df)
```

"look up" p-value

Parameter Inference

```python
# Or more easily
from statsmodels.stats.anova import anova_lm

anova_lm(compact_results, augmented_results)
```

✓ 0.0s

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	399.0	8.433991e+07	0.0	NaN	NaN	NaN
1	398.0	6.620874e+07	1.0	1.813117e+07	108.991715	1.030886e-22

$$F = t^2$$

$$t = \sqrt{F}$$

```python
print(augmented_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Balance   R-squared:                       0.215
Model:                            OLS   Adj. R-squared:                  0.213
Method:                 Least Squares   F-statistic:                     109.0
Date:                Wed, 12 Feb 2025   Prob (F-statistic):           1.03e-22
Time:                        09:16:07   Log-Likelihood:                -2970.9
No. Observations:                 400   AIC:                             5946.
Df Residuals:                     398   BIC:                             5954.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    246.5148     33.199      7.425      0.000     181.247     311.783
Income         6.0484      0.579     10.440      0.000       4.909       7.187
==============================================================================
Omnibus:                       42.505   Durbin-Watson:                   1.951
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.975
Skew:                           0.384   Prob(JB):                     2.79e-05
Kurtosis:                       2.182   Cond. No.                         93.3
==============================================================================
```

Parameter inference is proportional to nested model comparison

```python
# Or more easily
from statsmodels.stats.anova import anova_lm

anova_lm(compact_results, augmented_results)
```

✓ 0.0s

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	399.0	8.433991e+07	0.0	NaN	NaN	NaN
1	398.0	6.620874e+07	1.0	1.813117e+07	108.991715	1.030886e-22

$$F = t^2$$

$$t = \sqrt{F}$$

```python
print(augmented_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Balance   R-squared:                       0.215
Model:                            OLS   Adj. R-squared:                  0.213
Method:                 Least Squares   F-statistic:                     109.0
Date:                Wed, 12 Feb 2025   Prob (F-statistic):           1.03e-22
Time:                        09:16:07   Log-Likelihood:                -2970.9
No. Observations:                 400   AIC:                             5946.
Df Residuals:                     398   BIC:                             5954.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    246.5148     33.199      7.425      0.000     181.247     311.783
Income         6.0484      0.579     10.440      0.000       4.909       7.187
==============================================================================
Omnibus:                       42.505   Durbin-Watson:                   1.951
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               20.975
Skew:                           0.384   Prob(JB):                     2.79e-05
Kurtosis:                       2.182   Cond. No.                         93.3
==============================================================================
```

But what is standard error??

OLS Regression Results

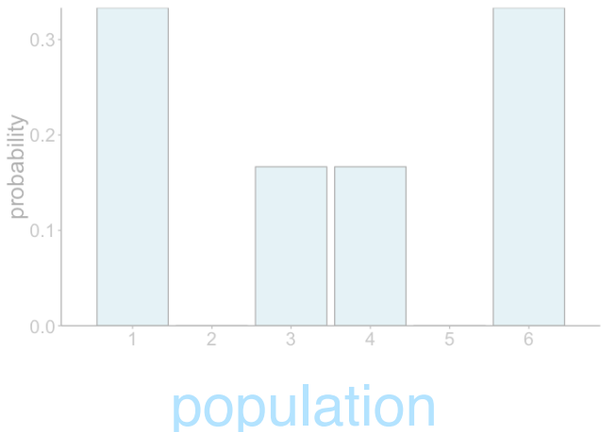
Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

But what is standard error??

How do we calculate it?



population

We don't know *true* PRE!

But it's what we *actually* care about

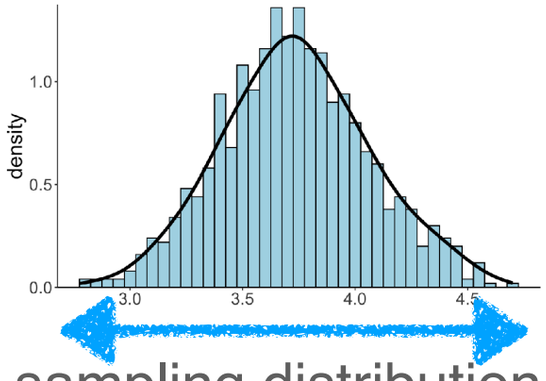
in theory….

The uncertainty bridge

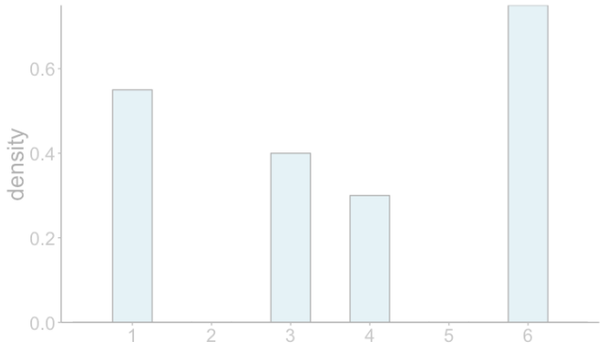
how an estimate varies between "samples"

samples = *analytic* (we look it up)

samples = *computational* (we resample)



sampling distribution



sample distribution

What we used to *estimate* our PRE

$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

Balance	Income
i64	f64
333	14.891
903	106.025
580	104.593
964	148.924
331	55.882
1151	80.18
203	20.996
872	71.408
279	15.125
1350	71.061

$$\begin{bmatrix} 333 \\ 903 \\ 580 \end{bmatrix} = \begin{bmatrix} 1 & 14.891 \\ 1 & 106.025 \\ 1 & 104.593 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

y X β

2 unknown parameters
we're *estimating*

-
-
-

$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

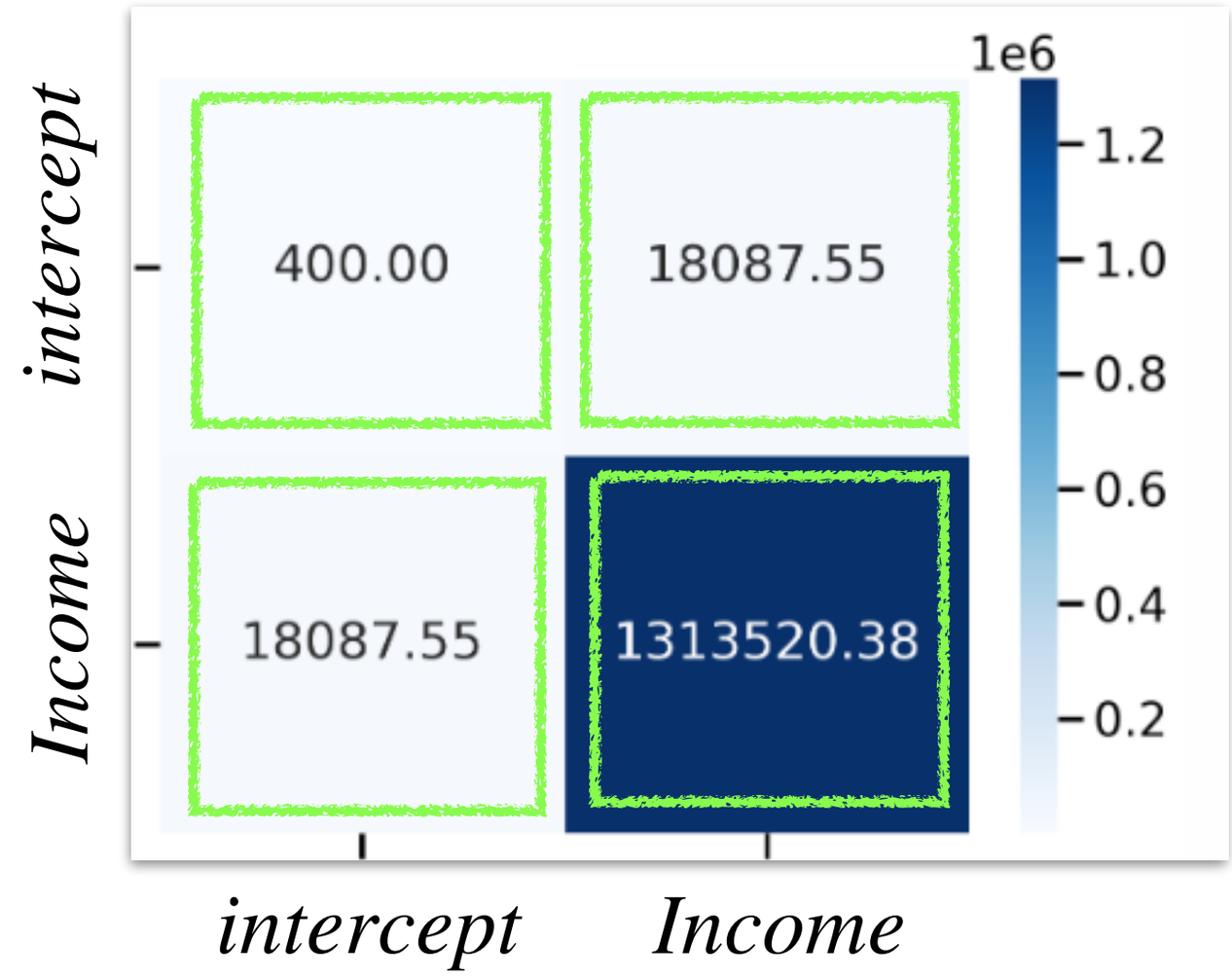
Similarity between predictors

Unscaled variance of each predictor

intercept	*Income*
1	14.891
1	106.025
1	104.593

X

Unscaled **covariance** between predictors



$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Divide by **variance** of each predictor

$$SE_{\hat{\beta}} = \sqrt{\sigma^2 (X^T X)^{-1}}$$

$$= \frac{\sum residuals^2}{df}$$

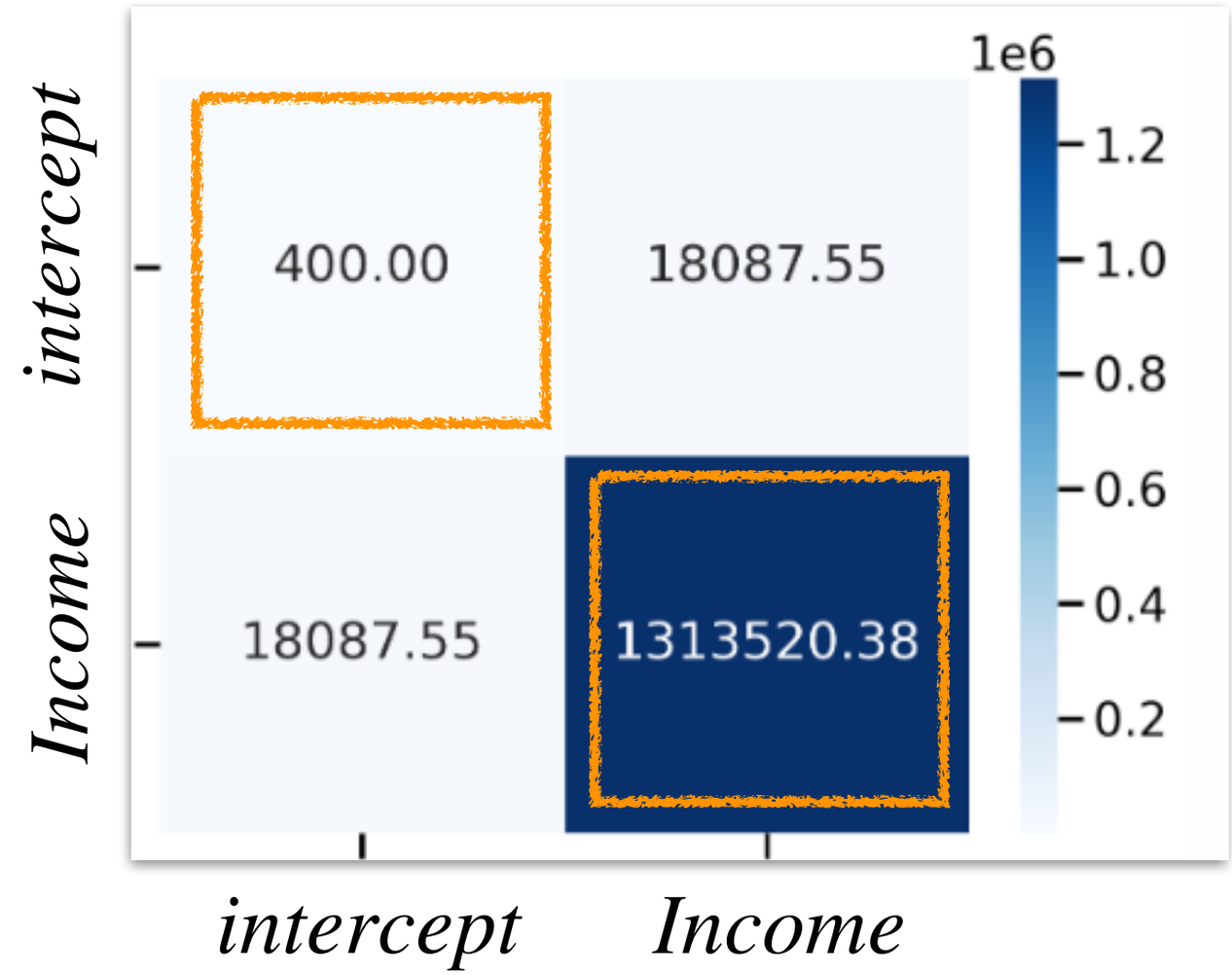
$$= \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{N - p}$$

Scale by

Variance of **residuals**

aka

Average **error** of the model



$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Divide by **variance** of each predictor

$$SE_{\hat{\beta}} = \sqrt{\sigma^2 (X^T X)^{-1}}$$

$$= \frac{\sum residuals^2}{df}$$

$$= \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{N - p}$$

Scale by
Variance of **residuals**
aka
Average **error** of the model

```
np.sqrt(
    augmented_results.mse_resid * np.diag(np.linalg.inv(X.T @ X))
    )
```

✓ 0.0s

```
array([33.19934735,  0.57935016])
```

$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

$$SE_{\hat{\beta}} = \sqrt{\sigma^2 (X^T X)^{-1}}$$

```
np.sqrt(
    augmented_results.mse_resid * np.diag(np.linalg.inv(X.T @ X))
    )
```

✓ 0.0s

```
array([33.19934735,  0.57935016])
```

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

$$SE_{\hat{\beta}} = \sqrt{\sigma^2 (X^T X)^{-1}}$$

$$t = \frac{\hat{\beta}}{SE_{\hat{\beta}}}$$

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

$$t_{intercept} = \frac{246.51}{33.19} = 7.425$$

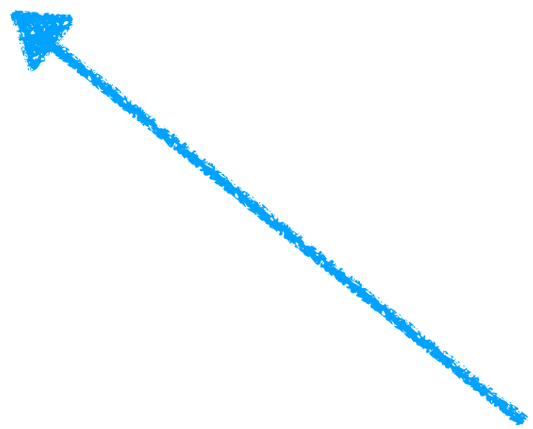
$$t_{Income} = \frac{6.04}{0.579} = 10.440$$

$$\hat{Balance}_i = \beta_0 + \beta_1 * Income_i + \epsilon$$

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

$$SE_{\hat{\beta}} = \sqrt{\sigma^2 (X^T X)^{-1}}$$

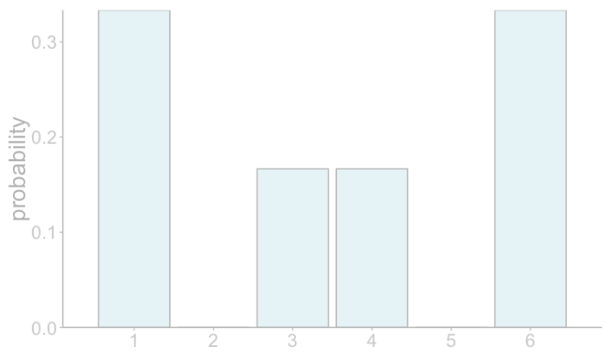
$$t = \frac{\hat{\beta}}{SE_{\hat{\beta}}}$$



Was this super confusing?!

Remember what we're trying to do….

standard error



population

We don't know *true* PRE!

But it's what we *actually* care about

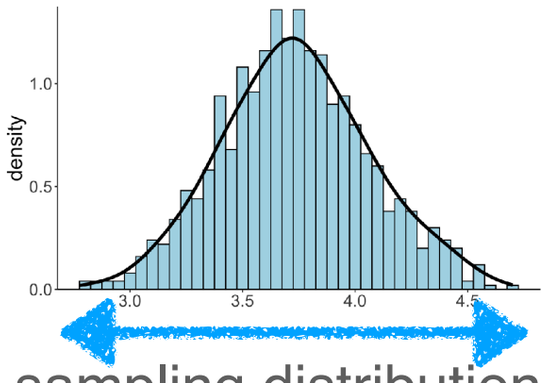
in theory….

The uncertainty bridge

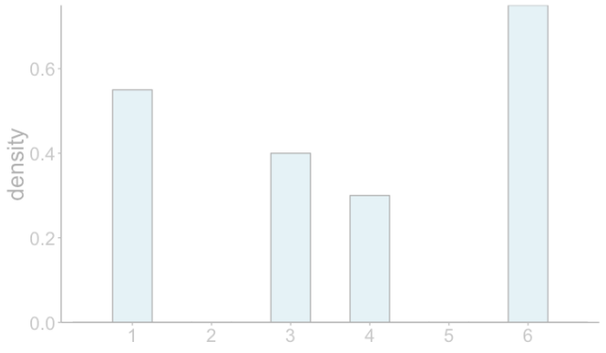
how an estimate varies between "samples"

samples = *analytic* (we look it up)

samples = *computational* (we resample)



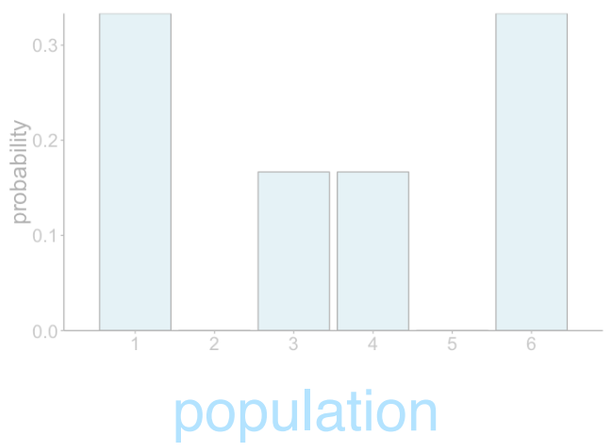
sampling distribution



sample distribution

What we used to *estimate* our PRE

What if we just build it…



population

We don't know *true* PRE!

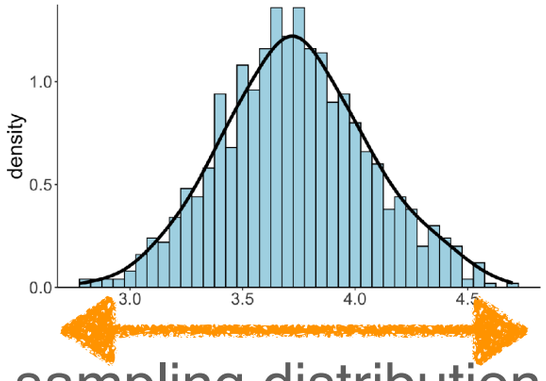
But it's what we *actually* care about

The uncertainty bridge

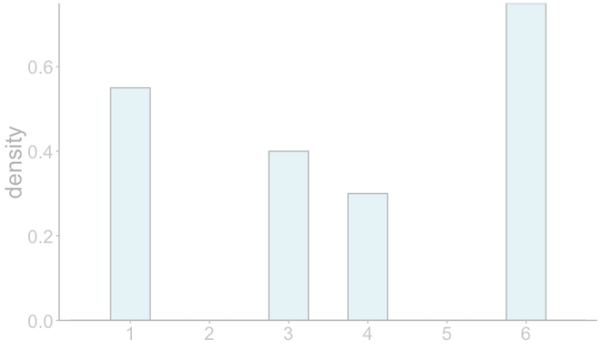
how an estimate varies between "samples"

samples = analytic (we look it up)

samples = *computational* (we resample)



sampling distribution



sample distribution

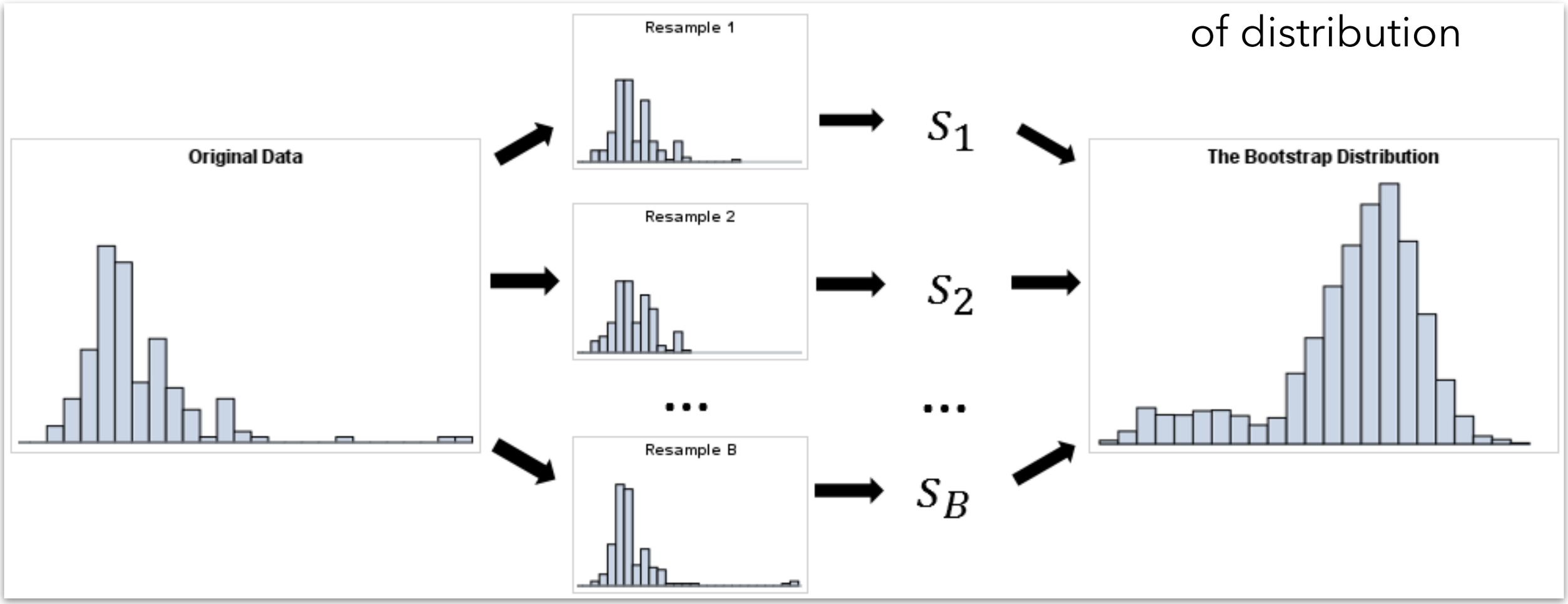
What we used to *estimate* our PRE

What if we just build it…by bootstrapping

Resample with replacement

Estimate $\hat{\beta}_{boot\_sample\_i}$

Get width (std-dev) of distribution



What if we just build it…by bootstrapping

```
nsim = 5000
params = []

for _ in range(nsim):

    # Generate new dataset with replacement
    new_data = df.sample(fraction=1., with_replacement=True)

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the parameters
    params.append(bresults.params.to_numpy())
```

Resample with replacement

Estimate $\hat{\beta}_{boot\_sample\_i}$

```
# Calculate SD, T, and CI of re-sampled distribution
params = np.array(params)
std_devs = params.std(ddof=1, axis=0)
tstats = augmented_results.params.to_numpy() / std_devs
CI_limits = np.percentile(params, [2.5, 97.5], axis=0)

# Combine into a DataFrame with original beta estimates
boot_results = pl.DataFrame(CI_limits).transpose().with_columns(
    names = np.array(['Intercept', 'Income']),
    se=std_devs,
    tstats=tstats,
    estimate=augmented_results.params.to_numpy(),
    ).select(
        col('names').alias('variable'),
        col('estimate').alias('coef'),
        col('se').alias('std err'),
        col('tstats').alias('t'),
        col('column_0').alias('[.0.025'),
        col('column_1').alias('0.975]'),
    )
```

Get width (std-dev) of distribution

What if we just build it…by bootstrapping

```python
nsim = 5000
params = []

for _ in range(nsim):

    # Generate new dataset with replacement
    new_data = df.sample(fraction=1., with_replacement=True)

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the parameters
    params.append(bresults.params.to_numpy())

# Calculate SD, T, and CI of re-sampled distribution
params = np.array(params)
std_devs = params.std(ddof=1, axis=0)
tstats = augmented_results.params.to_numpy() / std_devs
CI_limits = np.percentile(params, [2.5, 97.5], axis=0)

# Combine into a DataFrame with original beta estimates
boot_results = pl.DataFrame(CI_limits).transpose().with_columns(
    names = np.array(['Intercept', 'Income']),
    se=std_devs,
    tstats=tstats,
    estimate=augmented_results.params.to_numpy(),
    ).select(
        col('names').alias('variable'),
        col('estimate').alias('coef'),
        col('se').alias('std err'),
        col('tstats').alias('t'),
        col('column_0').alias('[0.025'),
        col('column_1').alias('0.975]'),
    )
```

variable	coef	std err	t	[0.025	0.975]
str	f64	f64	f64	f64	f64
"Intercept"	246.514751	31.862764	7.736766	186.117461	310.232929
"Income"	6.048363	0.595855	10.150734	4.877137	7.220309

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

Bootstrapping let's us simulate uncertainty

```python
nsim = 5000
params = []

for _ in range(nsim):

    # Generate new dataset with replacement
    new_data = df.sample(fraction=1., with_replacement=True)

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the parameters
    params.append(bresults.params.to_numpy())

# Calculate SD, T, and CI of re-sampled distribution
params = np.array(params)
std_devs = params.std(ddof=1, axis=0)
tstats = augmented_results.params.to_numpy() / std_devs
CI_limits = np.percentile(params, [2.5, 97.5], axis=0)

# Combine into a DataFrame with original beta estimates
boot_results = pl.DataFrame(CI_limits).transpose().with_columns(
    names = np.array(['Intercept', 'Income']),
    se=std_devs,
    tstats=tstats,
    estimate=augmented_results.params.to_numpy(),
    ).select(
        col('names').alias('variable'),
        col('estimate').alias('coef'),
        col('se').alias('std err'),
        col('tstats').alias('t'),
        col('column_0').alias('[0.025'),
        col('column_1').alias('0.975]'),
    )
```

variable	coef	std err	t	[0.025	0.975]
str	f64	f64	f64	f64	f64
"Intercept"	246.514751	31.862764	7.736766	186.117461	310.232929
"Income"	6.048363	0.595855	10.150734	4.877137	7.220309

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

What about…

```
nsim = 5000
params = []

for _ in range(nsim):

    # Generate new dataset with replacement
    new_data = df.sample(fraction=1., with_replacement=True)

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the parameters
    params.append(bresults.params.to_numpy())

# Calculate SD, T, and CI of re-sampled distribution
params = np.array(params)
std_devs = params.std(ddof=1, axis=0)
tstats = augmented_results.params.to_numpy() / std_devs
CI_limits = np.percentile(params, [2.5, 97.5], axis=0)

# Combine into a DataFrame with original beta estimates
boot_results = pl.DataFrame(CI_limits).transpose().with_columns(
    names = np.array(['Intercept', 'Income']),
    se=std_devs,
    tstats=tstats,
    estimate=augmented_results.params.to_numpy(),
    ).select(
        col('names').alias('variable'),
        col('estimate').alias('coef'),
        col('se').alias('std err'),
        col('tstats').alias('t'),
        col('column_0').alias('[0.025'),
        col('column_1').alias('0.975]'),
    )
```

variable	coef	std err	t	[0.025	0.975]
str	f64	f64	f64	f64	f64
"Intercept"	246.514751	31.862764	7.736766	186.117461	310.232929
"Income"	6.048363	0.595855	10.150734	4.877137	7.220309

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

What about…

What is a p-value?

The **p-value** is the probability of finding the observed (or more extreme) results when the null hypothesis (H_0) is true.

$$p(\text{test statistic} \geq \text{observed value} \mid H_0 = \text{true})$$

OLS Regression Results

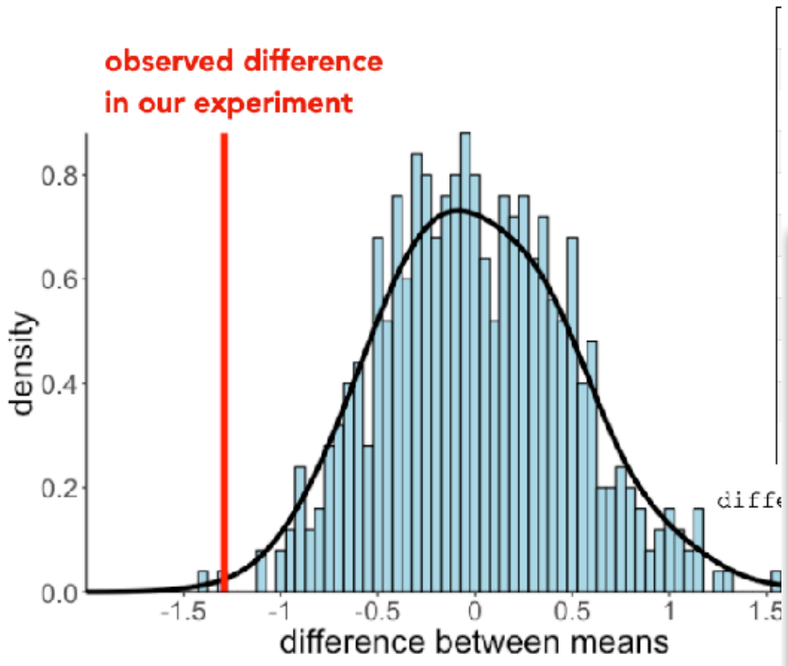
Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

We can simulate this with permutation!

Statistic after shuffling =
resampled null distribution



What is a p-value?

The **p-value** is the probability of finding the observed (or more extreme) results when the null hypothesis (H_0) is true.

$$p(\text{test statistic} \geq \text{observed value} \mid H_0 = \text{true})$$

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

Building the null…by permuting (shuffling)

Shuffle y

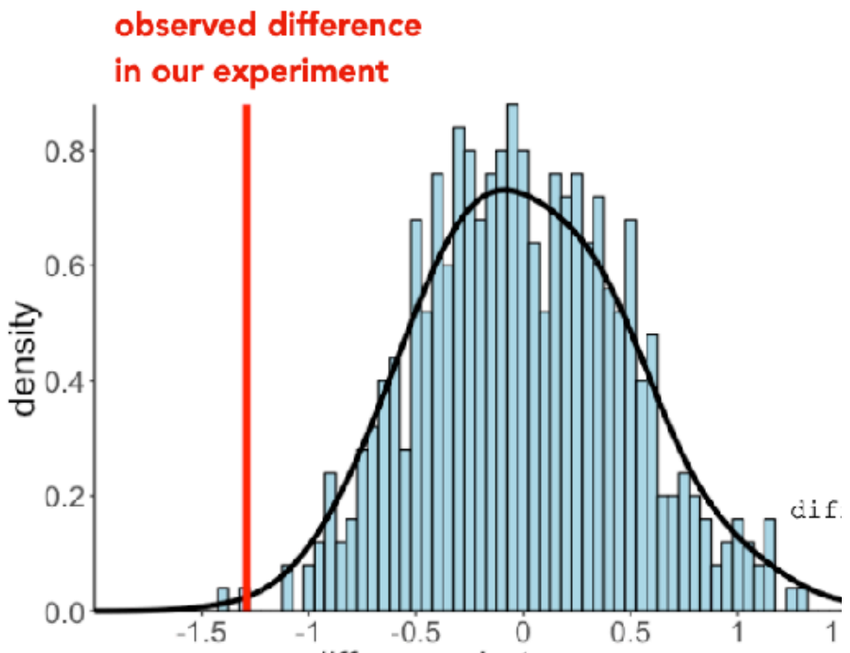
Balance	Income
i64	f64
333	14.891
903	106.025
580	104.593
964	148.924
331	55.882
1151	80.18
203	20.996
872	71.408
279	15.125
1350	71.061

Re- Estimate t_{perm_i}

$$t_{perm\_i} = \frac{\beta_{perm\_i}}{SE_{\beta_{perm\_i}}}$$

Check how often

$$t_{perm} \geq t_{observed}$$



Building the null…by permuting (shuffling)

```python
nperm = 5000
tstats = []

for _ in range(nsim):

    # Generate new dataset by shuffling rows in the DV column only
    new_data = df.with_columns(
        Balance = df['Balance'].sample(fraction=1.,
                                       with_replacement=False,
                                       shuffle=True)

        )

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the t-stats
    tstats.append(bresults.tvalues.to_numpy())

tstats = np.array(tstats)

# Get the # of permuted t-stats >= observed t-stat
proportion = np.sum(
    np.abs(tstats) >= np.abs(augmented_results.tvalues.to_numpy()),
    axis=0) + 1

# Calculate p-value
pval = proportion / (nperm + 1)
```

Shuffle y

Re- Estimate t_{perm_i}

Check how often

$t_{perm} \geq t_{observed}$

Building the null…by permuting (shuffling)

```python
nperm = 5000
tstats = []

for _ in range(nsim):

    # Generate new dataset by shuffling rows in the DV column only
    new_data = df.with_columns(
        Balance = df['Balance'].sample(fraction=1.,
                                       with_replacement=False,
                                       shuffle=True)
        )

    # Fit model to it
    bmodel = ols('Balance ~ Income', data=new_data.to_pandas())
    bresults = bmodel.fit()

    # Save the t-stats
    tstats.append(bresults.tvalues.to_numpy())

tstats = np.array(tstats)

# Get the # of permuted t-stats >= observed t-stat
proportion = np.sum(
    np.abs(tstats) >= np.abs(augmented_results.tvalues.to_numpy()),
    axis=0) + 1

# Calculate p-value
pval = proportion / (nperm + 1)
```

variable	coef	std err	t	P>|t|	[0.025	0.975]
str	f64	f64	f64	f64	f64	f64
"Intercept"	246.514751	31.862764	7.736766	NaN	186.117461	310.232929
"Income"	6.048363	0.595855	10.150734	0.0002	4.877137	7.220309

OLS Regression Results

Dep. Variable:	Balance	R-squared:	0.215
Model:	OLS	Adj. R-squared:	0.213
Method:	Least Squares	F-statistic:	109.0
Date:	Wed, 12 Feb 2025	Prob (F-statistic):	1.03e-22
Time:	09:16:07	Log-Likelihood:	-2970.9
No. Observations:	400	AIC:	5946.
Df Residuals:	398	BIC:	5954.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	246.5148	33.199	7.425	0.000	181.247	311.783
Income	6.0484	0.579	10.440	0.000	4.909	7.187

Omnibus:	42.505	Durbin-Watson:	1.951
Prob(Omnibus):	0.000	Jarque-Bera (JB):	20.975
Skew:	0.384	Prob(JB):	2.79e-05
Kurtosis:	2.182	Cond. No.	93.3

Parameter Inference Takeaways

$$t = \frac{\hat{\beta}}{SE_{\hat{\beta}}}$$

- Large t-stat or small p-value is **not** about what variables are most important for prediction!
- It's about what **signals** can be discerned from **noise**
  - *Does the addition of this parameter provide lower prediction error that what adding random noise would produce?*
- Why? Because our *uncertainty* (SE) is influence by
  - Magnitude of ($\hat{\beta}$)
  - How bad model is (MSE)
  - Effective sample size (# observations vs # params aka DF)
  - Variance of each predictor ($x1...xn$)
  - Correlation between predictors (columns of X)

Statistical significance is not practical significance

Break…then…

- Keep working on notebooks 1-3
- Make sure to do readings (or watch videos)
- HW 3 will be posted later this week