Reference Guide for R

Links to jump to…

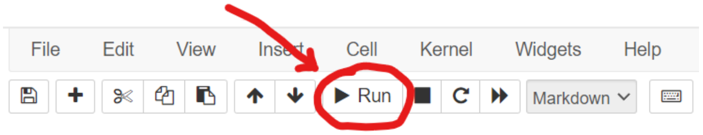
- Useful R commands from Notebook 1 (Basic R & Data Exploration)
- Useful R commands from Notebook 2 (Simple Linear Regression)
- Useful R commands from Notebook 3 (Multiple Regression)
- Useful R commands from Notebook 4 (Machine Learning)

Helpful tips for using notebooks

- Avoid opening multiple notebooks in separate tabs, as this can lead to memory overload, forcing you to restart your session.

How to run a code cell

Click into the code cell and then click the “Run” button in the top toolbar:



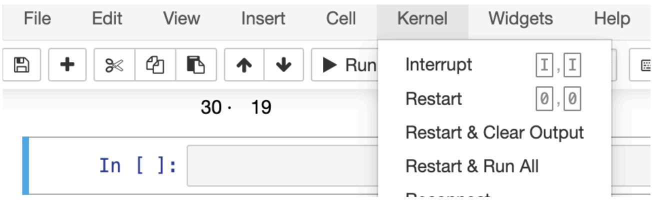
Or use keyboard shortcuts:

- For PCs and Chromebooks: Hold Ctrl and press Enter (on PCs)
- For Macs: Hold Command (⌘) and press Enter

Running all code cells from top down

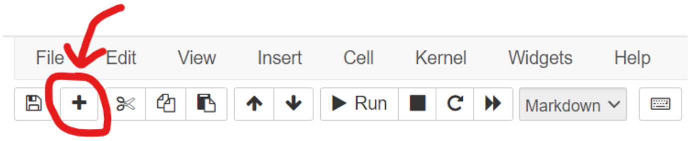
Although clicking the “save” button will save any code or text you’ve written, it won’t save your data in its memory. So, whenever you return to a notebook, make sure to re-run all code cells from the top down.

Shortcut:
To re-run all code cells (top down) in notebook, go to “Restart & Run All” under “Kernel”

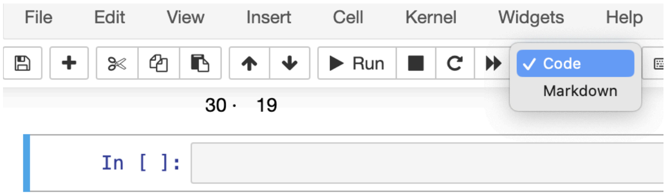


Adding new code or text cells

To add code or markdown cells, use the “+” in the top toolbar:



Then change type as needed (choose “code” for code and “markdown” for text).



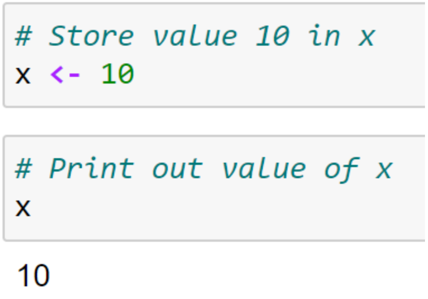
Useful R Commands

Notebook 1: Basic R & Data Exploration

`<-` - store values

Example: `x <- 10`

- stores the value `10` in the object `x`



```
# Store value 10 in x
x <- 10
```

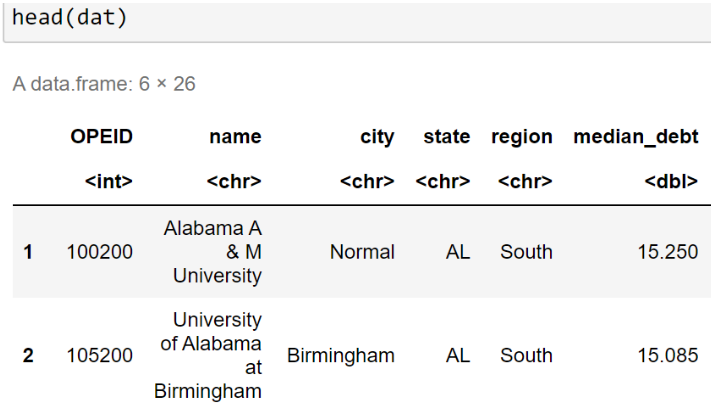
```
# Print out value of x
x
```

```
10
```

`head()` - display head of dataset

Example: `head(dat)`

- displays the head of a dataset named `dat`



```
head(dat)
```

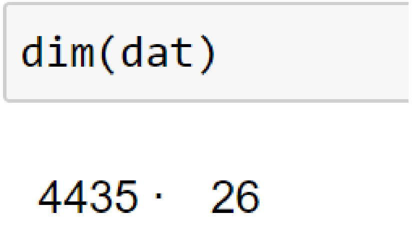
A data.frame: 6 × 26

	OPEID <int>	name <chr>	city <chr>	state <chr>	region <chr>	median_debt <dbl>
1	100200	Alabama A & M University	Normal	AL	South	15.250
2	105200	University of Alabama at Birmingham	Birmingham	AL	South	15.085

`dim()` - display dimensions of dataset

Example: `dim(dat)`

- displays dimensions of a dataset named `dat`
- first number is the number of rows (horizontal), next is the number of columns (vertical)



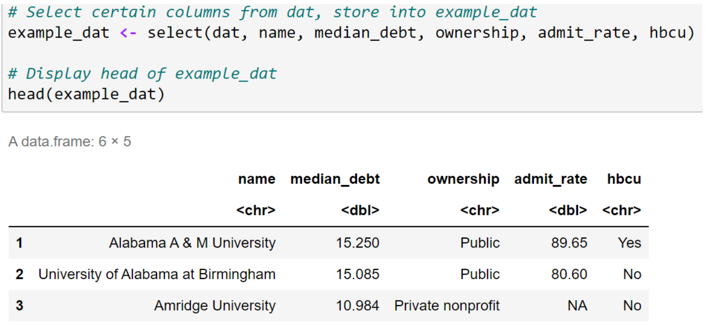
```
dim(dat)
```

```
4435 ·   26
```

`select()` - select only certain variables (columns) from dataset

Example: `example_dat <- select(dat, name, median_debt, ownership, admit_rate, hbcu)`

- selects the columns `name`, `median_debt`, `ownership`, `admit_rate`, and `hbcu` from a dataset named `dat` and stores the results in a new dataset named `example_dat`



```
# Select certain columns from dat, store into example_dat
example_dat <- select(dat, name, median_debt, ownership, admit_rate, hbcu)

# Display head of example_dat
head(example_dat)
```

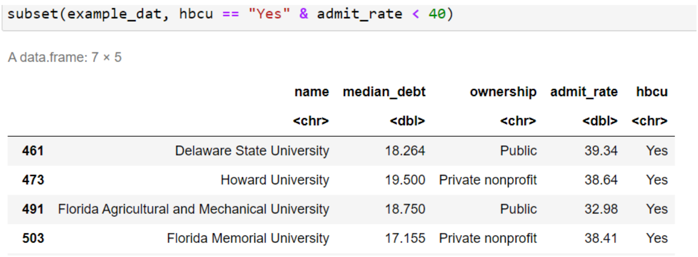
A data.frame: 6 × 5

	name <chr>	median_debt <dbl>	ownership <chr>	admit_rate <dbl>	hbcu <chr>
1	Alabama A & M University	15.250	Public	89.65	Yes
2	University of Alabama at Birmingham	15.085	Public	80.60	No
3	Amridge University	10.984	Private nonprofit	NA	No

`subset()` - filter dataset to obtain certain observations (rows), based on conditions

Example: `subset(example_dat, hbcu == "Yes" & admit_rate < 40)`

- Filters dataset `example_dat` to only include observations (rows) that are HBCUs and that have an admit rate lower than 40%



```
subset(example_dat, hbcu == "Yes" & admit_rate < 40)
```

A data.frame: 7 × 5

	name	median_debt	ownership	admit_rate	hbcu
	<chr>	<dbl>	<chr>	<dbl>	<chr>
461	Delaware State University	18.264	Public	39.34	Yes
473	Howard University	19.500	Private nonprofit	38.64	Yes
491	Florida Agricultural and Mechanical University	18.750	Public	32.98	Yes
503	Florida Memorial University	17.155	Private nonprofit	38.41	Yes

Conditions

- `==` means `equals exactly`
- `!=` means `does not equal`
- `<` means `less than`
- `>` means `greater than`
- `<=` means `less than or equal to`
- `>=` means `greater than or equal to`
- `|` means `or`
- `&` means `and`

`arrange()` - order data based on values

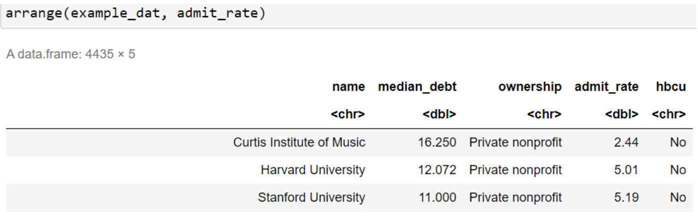
`desc()` - modifies `arrange()` to put data in descending order

Example: `arrange(example_dat, admit_rate)`

- Orders the rows in dataset `example_dat` based on admission rates, with lowest admission rates first.

Example: `arrange(example_dat, desc(admit_rate))`

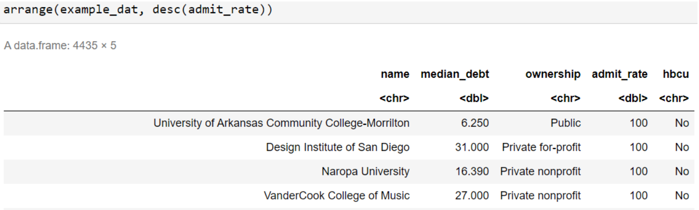
- Orders the rows in dataset `example_dat` based on admission rates, with highest admission rates first.



```
arrange(example_dat, admit_rate)
```

A data.frame: 4435 × 5

name	median_debt	ownership	admit_rate	hbcu
<chr>	<dbl>	<chr>	<dbl>	<chr>
Curtis Institute of Music	16.250	Private nonprofit	2.44	No
Harvard University	12.072	Private nonprofit	5.01	No
Stanford University	11.000	Private nonprofit	5.19	No



```
arrange(example_dat, desc(admit_rate))
```

A data.frame: 4435 × 5

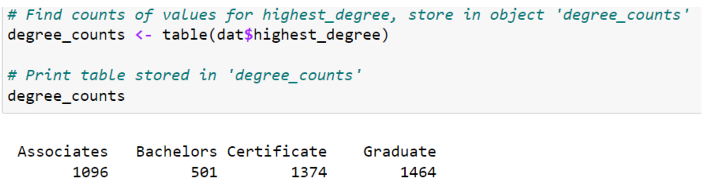
name	median_debt	ownership	admit_rate	hbcu
<chr>	<dbl>	<chr>	<dbl>	<chr>
University of Arkansas Community College-Morrilton	6.250	Public	100	No
Design Institute of San Diego	31.000	Private for-profit	100	No
Naropa University	16.390	Private nonprofit	100	No
VanderCook College of Music	27.000	Private nonprofit	100	No

`$` - selects a single variable from a dataset

`table()` - find counts of a categorical variable

Example: `table(dat$highest_degree)`

- `dat$highest_degree` selects the `highest_degree` column from `dat`.
- The `table()` command then displays the number of colleges in `dat` that have different types of highest degrees



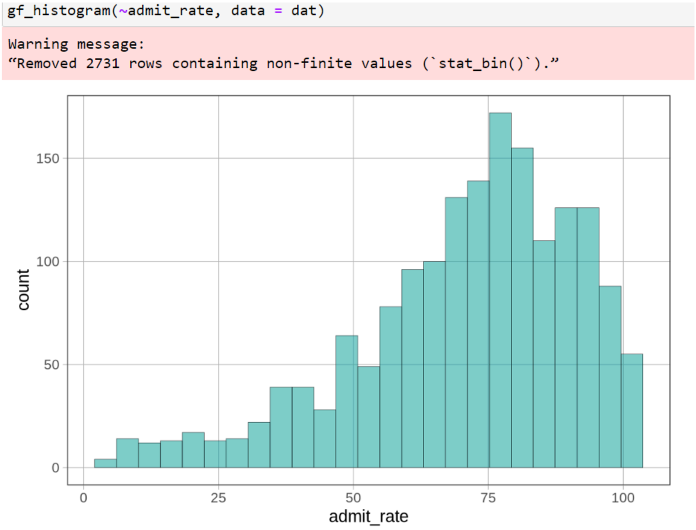
```
# Find counts of values for highest_degree, store in object 'degree_counts'
degree_counts <- table(dat$highest_degree)

# Print table stored in 'degree_counts'
degree_counts
```

```
 Associates   Bachelors Certificate    Graduate 
       1096         501        1374        1464 
```

`gf_histogram()` - creates histogram of a quantitative variable

Example: `gf_histogram(~admit_rate, data = dat)`

- Creates histogram of the `admit_rate` variable from the dataset `dat`



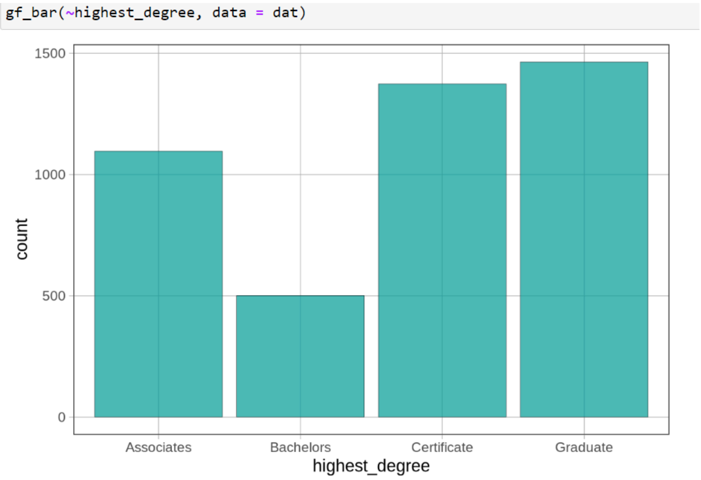
```
gf_histogram(~admit_rate, data = dat)
```

```
Warning message:
"Removed 2731 rows containing non-finite values (`stat_bin()`)."
```

`gf_bar()` - creates barplot of a categorical variable

Example: `gf_bar(~highest_degree, data = dat)`

- Creates barplot of the `highest_degree` variable from the dataset `dat`



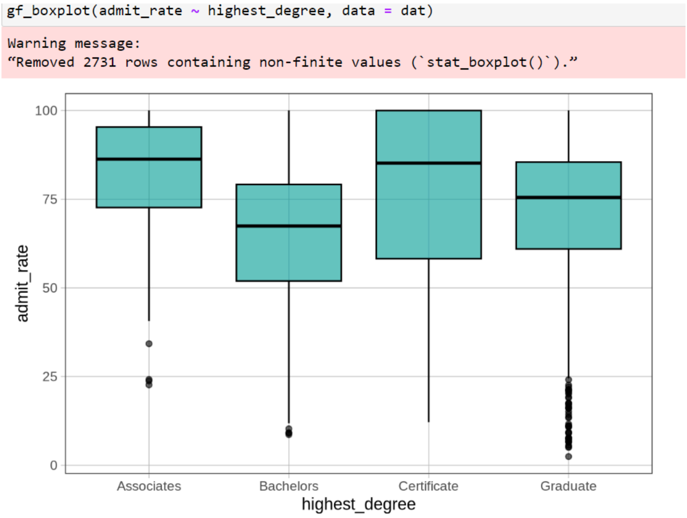
```
gf_bar(~highest_degree, data = dat)
```

`~` - often used to delineate the outcome variable (y) and the predictor variable (x) in graphs and models, like this: `outcome ~ predictor`

`gf_boxplot()` - creates boxplots

Example: `gf_boxplot(admit_rate ~ highest_degree, data = dat)`

- `admit_rate ~ highest_degree` signifies that admit rate is the outcome (y) variable and highest degree is the predictor (x) variable
- The `gf_boxplot()` command then generates the full boxplot, plotting admit rate on the y-axis and highest degree on the x-axis



```
gf_boxplot(admit_rate ~ highest_degree, data = dat)
```

```
Warning message:
"Removed 2731 rows containing non-finite values (`stat_boxplot()`)."
```

Useful R Commands

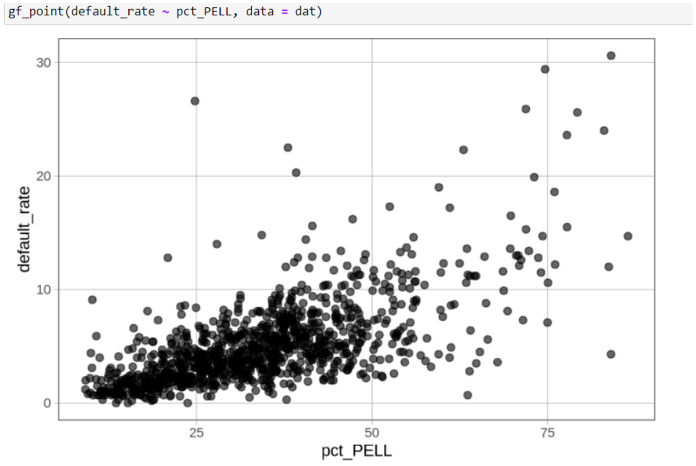
Notebook 2: Simple Linear Regression

`~` - often used to delineate the outcome variable (y) and the predictor variable (x) in graphs and models, like this: `outcome ~ predictor`

`gf_point()` - creates scatterplots

Example: `gf_point(default_rate ~ pct_PELL, data = dat)`

- `default_rate ~ pct_PELL` signifies that default rate is the outcome (y) variable and percent PELL is the predictor (x) variable
- The `gf_point()` command then generates the full scatterplot, plotting default rate on the y-axis and percent PELL on the x-axis
- `data = dat` tells the command that the data is come from the dataset named `dat`



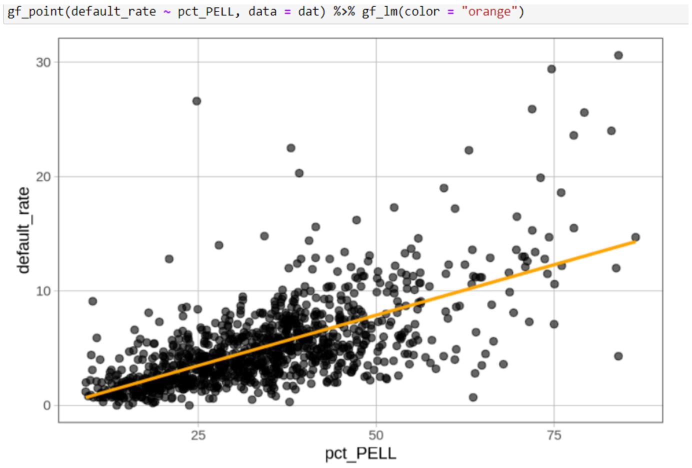
```
gf_point(default_rate ~ pct_PELL, data = dat)
```

`%>%` - used to pipe (overlay) models on top of graphs

`gf_lm()` - plots linear model

Example: `gf_point(default_rate ~ pct_PELL, data = dat) %>% gf_lm(color = "orange")`

- The `gf_point(default_rate ~ pct_PELL, data = dat)` part creates a scatterplot with default rate as the outcome (y) variable and percent PELL as the predictor (x) variable
- The `%>% gf_lm(color = "orange")` part graphs the linear model on top of the scatterplot. command then generates the full scatterplot, plotting default rate on the y-axis and percent PELL on the x-axis
- `color = "orange"` sets the color of the linear model to orange

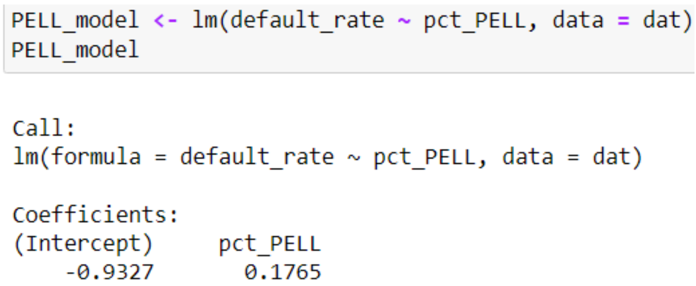


```
gf_point(default_rate ~ pct_PELL, data = dat) %>% gf_lm(color = "orange")
```

`lm()` - fits a linear regression model

Example: `PELL_model <- lm(default_rate ~ pct_PELL, data = dat)`

- The `lm(default_rate ~ pct_PELL, data = dat)` part fits a linear model with default rate as the outcome (y) variable and percent PELL as the predictor (x) variable, using the dataset `dat`
- The `PELL_model <-` part stores the linear model in an object called `PELL_model`
- If we create a new line of code and run simply `PELL_model` (the name of our stored model), it will print the coefficients of the model



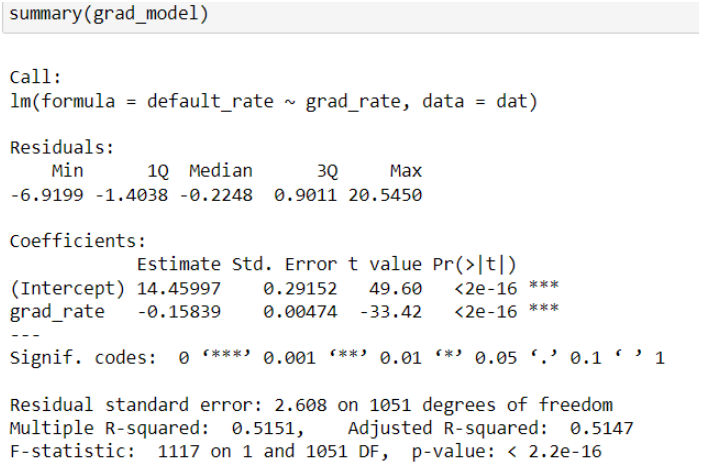
```
PELL_model <- lm(default_rate ~ pct_PELL, data = dat)
PELL_model
```

```

Call:
lm(formula = default_rate ~ pct_PELL, data = dat)

Coefficients:
(Intercept)     pct_PELL
    -0.9327       0.1765
```

`summary()` - displays detailed information about a regression model

Example: `summary(grad_model)`

- Prints out detailed information (including R^2) about a previously fit linear model named `grad_model`
- Look for `Multiple R-squared` in the output to find the R^2 value



```
summary(grad_model)
```

```

Call:
lm(formula = default_rate ~ grad_rate, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max 
-6.9199 -1.4038 -0.2248  0.9011 20.5450 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 14.45997    0.29152   49.60   <2e-16 ***
grad_rate   -0.15839    0.00474  -33.42   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.608 on 1051 degrees of freedom
Multiple R-squared:  0.5151,	Adjusted R-squared:  0.5147 
F-statistic:  1117 on 1 and 1051 DF,  p-value: < 2.2e-16
```

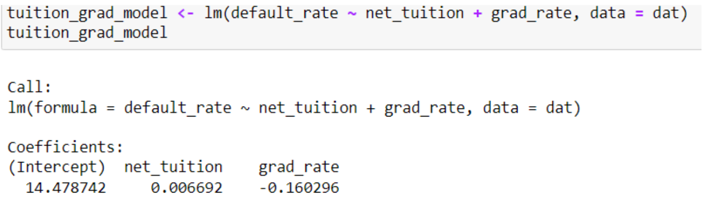
Useful R Commands

Notebook 3: Multiple Regression

`lm(y ~ x1 + x2 + x3 + ..., data = dat)`
- syntax for a multiple regression model

Example: `tuition_grad_model <- lm(default_rate ~ net_tuition + grad_rate, data = dat)`

- The `lm(default_rate ~ net_tuition + grad_rate, data = dat)` part fits a regression model with default rate as the outcome (y) variable and two predictors: net tuition (x_1) and grad rate (x_2)
- The `tuition_grad_model <-` part stores the regression model in an object called `tuition_grad_model`
- If we create a new line of code and run simply `tuition_grad_model` (the name of our stored model), it will print the coefficients of the model

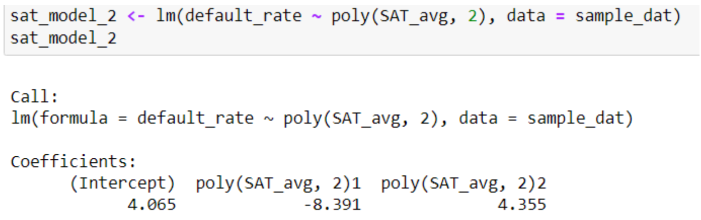


```
tuition_grad_model <- lm(default_rate ~ net_tuition + grad_rate, data = dat)
tuition_grad_model
```

```

Call:
lm(formula = default_rate ~ net_tuition + grad_rate, data = dat)

Coefficients:
(Intercept)  net_tuition    grad_rate  
  14.478742     0.006692    -0.160296  
```

Useful R Commands

Notebook 4: Machine Learning

`poly(variable, degree)` - adds polynomial terms to a model
 	
Example: `lm(default_rate ~ poly(SAT_avg, 2), data = sample_dat)`
● Fits a regression model with default rate as the outcome (y) variable and SAT average as the predictor (x). The SAT average will have two polynomial terms: x and x^2
	
	`sat_model_2 <- lm(default_rate ~ poly(SAT_avg, 2), data = sample_dat)`
	`sat_model_2`
	
	`Call:`
	`lm(formula = default_rate ~ poly(SAT_avg, 2), data = sample_dat)`
	
	`Coefficients:`
	`        (Intercept)  poly(SAT_avg, 2)1  poly(SAT_avg, 2)2`
	`              4.065             -8.391              4.355`

`predict(model,newdata = dat)` - gives predictions from model on new data Example: `predict(my_model, newdata = test)` ● Will use a previously fit model named `my_model` to make predictions on a dataset named `test`	