Sample lesson - Fitting and interpreting linear regression models in R

Set-up R environment

As usual, we need the `tidyverse` packages (which include `dplyr`, `ggplot2`, `readr`, etc). The `scales` package lets us easily format our plot axes. We will also learn about the `broom` package that allows us to extract model parameters, measures of model complexity and quality, as well as residuals and predicted values as tidy data. We load `broom` independently of the `tidyverse` because `broom` was only added to the `tidyverse` a couple months ago…

```r
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr

## Warning: package 'ggplot2' was built under R version 3.3.2

## Conflicts with tidy packages ----------------------------------------------

## filter(): dplyr, stats
## lag():    dplyr, stats
```

```r
library(scales)
```

```
## Warning: package 'scales' was built under R version 3.3.2

##
## Attaching package: 'scales'

## The following object is masked from 'package:purrr':
##
##     discard

## The following objects are masked from 'package:readr':
##
##     col_factor, col_numeric
```

```r
library(broom)
```

Explore Dataset:

We will be expoloring factors that influence median property value at the state-level in the United States of America in the year 2015. Data was downloaded from Data USA using the API in Python. We will learn how to do this in the Web and Cloud Computing course next semester. If you are interested in seeing how this was done now, see this Jupyter notebook.

The dataset has many interesting possible explanatory values that we could use to model against the response variable we are interested in, median property value. For now, we will start with a simple linear regression using only median household income as a single explanatory variable. We will build up the complexity of our model in later lectures.

Here we load the data and peak at it as a table:

```r
prop_data <- read_csv("https://raw.githubusercontent.com/ttimbers/UBC-stat-sample-lesson/master/data/ac
```

```
## Parsed with column specification:
## cols(
##   display_name = col_character(),
##   income = col_double(),
##   mean_commute_minutes = col_double(),
##   median_property_value = col_double(),
##   non_eng_speakers_pct = col_double(),
##   owner_occupied_housing_units = col_double(),
##   pop = col_integer(),
##   year = col_integer()
## )
```

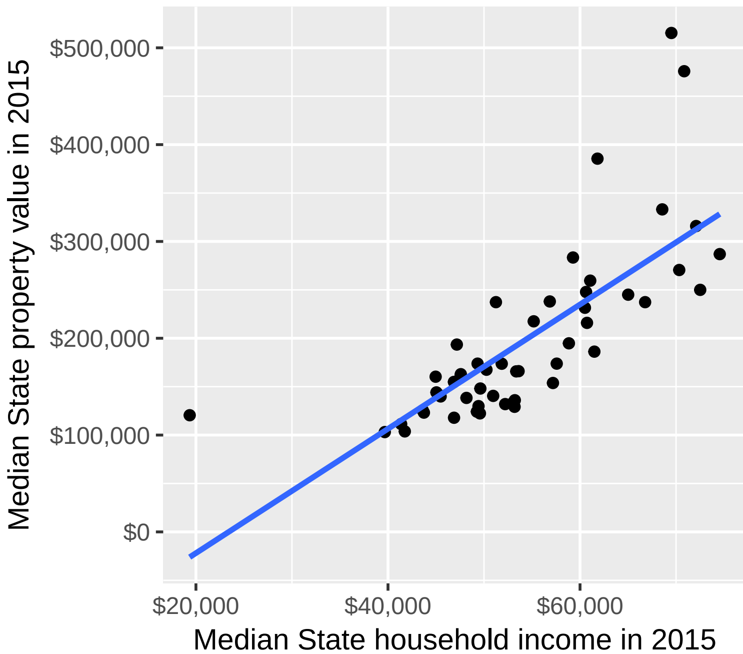
```r
# or if you are not connected to internet:
# prop_data <- read_csv("data/acs_data.csv")

head(prop_data)
```

```
## # A tibble: 6 × 8
##     display_name income mean_commute_minutes median_property_value
##            <chr>  <dbl>                <dbl>                 <dbl>
## 1          Texas  53207              24.5090                136000
## 2   Pennsylvania  53599              25.2695                166000
## 3 South Carolina  45483              23.0552                139900
## 4  New Hampshire  66779              25.2973                237300
## 5         Kansas  52205              18.2779                132000
## 6         Hawaii  69515              25.5813                515300
## # ... with 4 more variables: non_eng_speakers_pct <dbl>,
## #   owner_occupied_housing_units <dbl>, pop <int>, year <int>
```

Let's visualize the data as a scatter plot:

We are interested in whether or not median household income might influence median property value at the state-level. To get an initial idea of whether this is plausible, we will plot the data as a scatter plot. We will also use ggplot's `geom_smooth` function to overlay the oridinary least squares regression line on our plot:

```r
(prop_data_scatter <- ggplot(prop_data, aes(y = median_property_value, x = income)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  xlab("Median State household income in 2015") +
  ylab("Median State property value in 2015") +
  scale_x_continuous(labels = dollar) +
  scale_y_continuous(labels = dollar))
```



Let's fit a linear model!

As it seems that there might be some relationship, we will now perform a simple linear regression where we use `median_property_value` as our response variable ("Y"), and `income` as our explanatory variable ("X1"). We will use the base R `summary` function to initially view the output of the linear model.

note - in R, we use the same model notation with linear regression as we have previously learned with ANOVA

```
prop_model <- lm(median_property_value ~ income, data = prop_data)
summary(prop_model)
```

```
##
## Call:
## lm(formula = median_property_value ~ income, data = prop_data)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -65249 -36542  -6990   8003 219312
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.503e+05  4.393e+04  -3.422  0.00125 **
## income       6.420e+00  7.999e-01   8.026 1.52e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 58860 on 50 degrees of freedom
## Multiple R-squared:  0.563,  Adjusted R-squared:  0.5542
## F-statistic: 64.41 on 1 and 50 DF,  p-value: 1.517e-10
```

Extracting model output in R

As we learned previously, there are a number of key pieces of information that we would like to be able to look at from our model, these include:

- $\hat{\beta}_0$
- $\hat{\beta}_1$
- p-value for H_0: $\beta_0 = 0$
- p-value for H_0: $\beta_1 = 0$
- residuals
- predicted/fitted values
- $\hat{\sigma}$
- R^2

All of these pieces of information can be extracted from the the linear model object, most (except for the residuals and predicted/fitted values) can be read from the print out from `summary`. Below is code for how to extract each of these individually:

$\hat{\beta}_0$ and $\hat{\beta}_0$, respectively:

```
prop_model$coefficients
```

```
##   (Intercept)        income
## -1.503050e+05  6.420101e+00
```

p-values for H_0: $\beta_0 = 0$ and H_0: $\beta_1 = 0$, respectively:

```
summary(prop_model)$coefficients[,4]
```

```
##  (Intercept)       income
## 1.248896e-03 1.517389e-10
```

The residuals:

```
prop_model$residuals
```

```
##          1          2          3          4          5          6
## -55289.283 -27805.963  -1800.420 -41122.898 -52856.341 219311.705
##          7          8          9         10         11         12
##   7717.367  40975.289  21938.033  53192.063 -63002.765  58619.796
##         13         14         15         16         17         18
## -61935.200 -45577.895 -37134.140 -20160.379   8860.725   4305.295
##         19         20         21         22         23         24
## -41419.925  13569.538  17780.821  -4837.144 -41717.042  -7310.184
##         25         26         27         28         29         30
##   5198.744   7395.030 -58279.822 -20570.493 -56741.062  23309.448
##         31         32         33         34         35         36
##   2094.487 -22097.839 -26452.298 -32653.714 146576.088 -32762.882
##         37         38         39         40         41         42
## -65248.599  -6668.863  -8857.945  -3900.964 -45525.865 138927.225
##         43         44         45         46         47         48
## 171253.710 -30727.098   4307.740  43223.641 -13940.602 -23668.445
##         49         50         51         52
##  -4259.032 -36344.055  -1248.271   3360.684
```

The predicted/fitted values:

```
prop_model$fitted.values
```

```
##         1         2         3         4         5         6         7
## 191289.28 193805.96 141700.42 278422.90 184856.34 295988.30 155182.63
##         8         9        10        11        12        13        14
```

```
## 152524.71 138361.97 230207.94 216802.77 178680.20 191135.20 167977.89
##        15        16        17        18        19        20        21
## 167034.14 168260.38 238939.27 154694.71 328319.93 203930.46 241719.18
##        22        23        24        25        26        27        28
## 172337.14 165917.04 130510.18 138901.26 166404.97 244479.82 158970.49
##        29        30        31        32        33        34        35
## 189941.06 214690.55 140005.51 267097.84 192252.30 227453.71 -26076.09
##        36        37        38        39        40        41        42
## 150662.88 315248.60 238168.86 182557.94 115300.96 219325.87 246572.78
##        43        44        45        46        47        48        49
## 304546.29 301227.10 150592.26 289876.36 117740.60 239568.44 129759.03
##        50        51        52
## 176844.05 104348.27 312539.32
```

$\hat{\sigma}$s:

```r
summary(prop_model)$sigma
```

```
## [1] 58858.23
```

R^2:

```r
summary(prop_model)$r.squared
```

```
## [1] 0.5629882
```

Extracting model output using the `broom` package

As you can see from our methods used above to extract key pieces of information from our model in base R, the methods to do so are not consistent (sometimes you are working with the model object itself, and sometimes the summary of that object…) and the data provided to you is not in a nice, tidy data format. This is not too problematic when working with only a couple models, but if you want to combine and compare model outputs from several models, using base R requires a bit of heroic effort on the programming front. To improve the readability of our code, and our efficiency in writing it, as well as to get the model parameters back in a tidy data format we can look to the relatively recent `broom` package.

`broom` has 3 functions that we are interested in:

1. **`tidy()`** - returns output related to model parameters (e.g., coefficients and p-values)
2. **`augment()`** - returns output related to model data (e.g., residuals and predicted values)
3. **`glance()`** - returns output related to model quality, complexity and summaries (e.g., R^2 and $\hat{\sigma}$)

Thus, with 3 consistent function calls, we can easily get all the model output we are interested, and this comes to us in an tidy data format that is accessible. Let's explore the output of each of these functions when we apply it to our model object:

`tidy()`

```r
tidy(prop_model)
```

```
##          term      estimate    std.error statistic      p.value
## 1 (Intercept) -1.503050e+05 4.392739e+04 -3.421670 1.248896e-03
## 2      income  6.420101e+00 7.999334e-01  8.025795 1.517389e-10
```

`augment()`

```r
head(augment(prop_model)) #use head to preview only 6 rows of data frame
```

```
##   median_property_value income  .fitted   .se.fit    .resid       .hat
## 1                136000  53207 191289.3  8184.217 -55289.28 0.01933481
## 2                166000  53599 193806.0  8167.204 -27805.96 0.01925451
## 3                139900  45483 141700.4 10610.207  -1800.42 0.03249626
## 4                237300  66779 278422.9 13107.757 -41122.90 0.04959552
## 5                132000  52205 184856.3  8281.684 -52856.34 0.01979808
## 6                515300  69515 295988.3 14882.799 219311.70 0.06393739
##     .sigma      .cooksd  .std.resid
## 1 58918.37 0.0088702539 -0.94857880
## 2 59320.33 0.0022338366 -0.47703760
## 3 59455.21 0.0000162417 -0.03109857
## 4 59149.62 0.0134013503 -0.71667509
## 5 58964.59 0.0083088666 -0.90705194
## 6 49863.41 0.5065524273  3.85125431
```

glance()

```r
glance(prop_model)
```

```
##   r.squared adj.r.squared    sigma statistic      p.value df    logLik
## 1 0.5629882      0.554248 58858.23  64.41339 1.517389e-10  2 -643.8752
##       AIC      BIC     deviance df.residual
## 1 1293.75 1299.604 173214534971          50
```

Given that these functions output tidy data frames, and there are not bizarre symbols in any of the column names of these data frames, extracting any required piece of information just follows the regular data frame mechanics. Furthermore, in the output of `lm()` from `summary()` the p-value from the F-statistic of the model is only printed to the screen and is never stored in memory. Thus, to use it one needs to either calculate it from the F-statistic, or use the `broom` package `glance()` to get it.

Thus, I strongly recommend using `tidy()`, `augment()` and `glance()` from the `broom` package when extracting information from your linear models in R. You will also see that these `broom` functions also work with more complex models (e.g., general-linearized models) in the Regression 2 course next block. Furthermore, because these model outputs come back as nice tidy data frames, you will find them very useful when you are working with multiple models, for example, when boostrapping or performing cross validation. We will see this in Feature and Model Selection course.