1 RMBI4310/COMP4332: Introduction to Linear Regression

<table>
<thead>
<tr>
<th></th>
<th>continuous</th>
<th>categorical</th>
</tr>
</thead>
<tbody>
<tr>
<td>supervised regression</td>
<td></td>
<td>classification</td>
</tr>
<tr>
<td>unsupervised</td>
<td>dimension reduction</td>
<td>clustering</td>
</tr>
</tbody>
</table>

1.1 Motivation

Why are we learning linear regression?
- widely used
- runs fast
- easy to use (not a lot of tuning required)
- highly interpretable
- basis for many other methods

1.2 Libraries

Will be using Statsmodels for teaching purposes since it has some nice characteristics for linear modeling. However, we recommend that you spend most of your energy on scikit-learn since it provides significantly more useful functionality for machine learning in general.

In [94]: # imports
   :   import pandas as pd
   :   import matplotlib.pyplot as plt

   :   # this allows plots to appear directly in the notebook
   :   %matplotlib inline

1.3 Example: Advertising Data

Let’s take a look at some data, ask some questions about that data, and then use linear regression to answer those questions!

In [95]: # read data into a DataFrame
   :   data = pd.read_csv('http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv', index_col=0)
   :   data.head()

```
Out[95]:
          TV    Radio  Newspaper  Sales
                  230.1   37.8     69.2   22.1
                  44.5   39.3     45.1   10.4
                  17.2   45.9     69.3    9.3
                151.5   41.3     58.5   18.5
                180.8   10.8     58.4   12.9
```
What are the features? - TV: advertising dollars spent on TV for a single product in a given market (in thousands of dollars) - Radio: advertising dollars spent on Radio - Newspaper: advertising dollars spent on Newspaper

What is the response? - Sales: sales of a single product in a given market (in thousands of widgets)

In [96]: # print the shape of the DataFrame
data.shape

Out[96]: (200, 4)

There are 200 observations, and thus 200 markets in the dataset.

In [97]: # visualize the relationship between the features and the response using scatterplots
fig, axs = plt.subplots(1, 3, sharey=True)
data.plot(kind='scatter', x='TV', y='Sales', ax=axs[0], figsize=(16, 8))
data.plot(kind='scatter', x='Radio', y='Sales', ax=axs[1])
data.plot(kind='scatter', x='Newspaper', y='Sales', ax=axs[2])

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x7f96d9fc5940>

1.4 Questions About the Advertising Data

Let’s pretend you work for the company that manufactures and markets this widget. The company might ask you the following: On the basis of this data, how should we spend our advertising money in the future? This general question might lead you to more specific questions: 1. Is there a relationship between ads and sales? 2. How strong is that relationship? 3. Which ad types contribute to sales? 4. What is the effect of each ad type of sales? 5. Given ad spending in a particular market, can sales be predicted?

We will explore these questions below!

1.5 Simple Linear Regression

Simple linear regression is an approach for predicting a quantitative response using a single feature (or “predictor” or “input variable”). It takes the following form:
\[ y = \beta_0 + \beta_1 x \]

What does each term represent? - \( y \) is the response - \( x \) is the feature - \( \beta_0 \) is the intercept - \( \beta_1 \) is the coefficient for \( x \)

Together, \( \beta_0 \) and \( \beta_1 \) are called the **model coefficients**. To create your model, you must “learn” the values of these coefficients. And once we’ve learned these coefficients, we can use the model to predict Sales!

### 1.6 Estimating (“Learning”) Model Coefficients

Generally speaking, coefficients are estimated using the **least squares criterion**, which means we are find the line (mathematically) which minimizes the **sum of squared residuals** (or “sum of squared errors”):

What elements are present in the diagram? - The black dots are the **observed values** of \( x \) and \( y \). - The blue line is our **least squares line**. - The red lines are the **residuals**, which are the distances between the observed values and the least squares line.

How do the model coefficients relate to the least squares line? - \( \beta_0 \) is the **intercept** (the value of \( y \) when \( x=0 \)) - \( \beta_1 \) is the **slope** (the change in \( y \) divided by change in \( x \))

Here is a graphical depiction of those calculations:

Let’s use **Statsmodels** to estimate the model coefficients for the advertising data:

```python
In [98]: # this is the standard import if you're using "formula notation" (similar to R)
import statsmodels.formula.api as smf

# create a fitted model in one line
lm = smf.ols(formula='Sales ~ TV', data=data).fit()

# print the coefficients
lm.params
```

```
Out[98]: Intercept   7.032594
        TV          0.047537
dtype: float64
```

### 1.7 Interpreting Model Coefficients

How do we interpret the TV coefficient (\( \beta_1 \))? - A “unit” increase in TV ad spending is **associated with** a 0.047537 “unit” increase in Sales. - Or more clearly: An additional $1,000 spent on TV ads is **associated with** an increase in sales of 47.537 widgets.

Note that if an increase in TV ad spending was associated with a **decrease** in sales, \( \beta_1 \) would be **negative**.

### 1.8 Using the Model for Prediction

Let’s say that there was a new market where the TV advertising spend was **$50,000**. What would we predict for the Sales in that market?

\[ y = \beta_0 + \beta_1 x \]

\[ y = 7.032594 + 0.047537 \times 50 \]

```python
In [99]: # manually calculate the prediction
    7.032594 + 0.047537*50
Out[99]: 9.409444
```

Thus, we would predict Sales of **9,409 widgets** in that market. Of course, we can also use Statsmodels to make the prediction:
In [100]: # you have to create a DataFrame since the Statsmodels formula interface expects it
   :   X_new = pd.DataFrame({'TV': [50]})
   :   X_new.head()
Out[100]:
   :    TV
   : 0   50

In [101]: # use the model to make predictions on a new value
   :   lm.predict(X_new)
Out[101]: array([ 9.40942557])

1.9 Plotting the Least Squares Line

Let’s make predictions for the smallest and largest observed values of x, and then use the predicted values to plot the least squares line:

In [102]: # create a DataFrame with the minimum and maximum values of TV
   :   X_new = pd.DataFrame({'TV': [data.TV.min(), data.TV.max()]})
   :   X_new.head()
Out[102]:
   :    TV
   : 0   0.7
   : 1  296.4

In [103]: # make predictions for those x values and store them
   :   preds = lm.predict(X_new)
   :   preds
Out[103]: array([ 7.0658692 , 21.12245377])

In [104]: # first, plot the observed data
   :   data.plot(kind='scatter', x='TV', y='Sales')
   : # then, plot the least squares line
   :   plt.plot(X_new, preds, c='red', linewidth=2)
Out[104]: [<matplotlib.lines.Line2D at 0x7f96d9f43f98>]


1.10 Confidence in our Model

**Question:** Is linear regression a high bias/low variance model, or a low bias/high variance model?

**Answer:** High bias/low variance. Under repeated sampling, the line will stay roughly in the same place (low variance), but the average of those models won’t do a great job capturing the true relationship (high bias). Note that low variance is a useful characteristic when you don’t have a lot of training data!

A closely related concept is **confidence intervals**. Statsmodels calculates 95% confidence intervals for our model coefficients, which are interpreted as follows: If the population from which this sample was drawn was **sampled 100 times**, approximately **95 of those confidence intervals** would contain the “true” coefficient.

```
In [105]: # print the confidence intervals for the model coefficients
   : lm.conf_int()
```

```
Out[105]:

        0      1
    Intercept  6.129719  7.935468
        TV   0.042231  0.052843
```

Keep in mind that we only have a **single sample of data**, and not the **entire population of data**. The “true” coefficient is either within this interval or it isn’t, but there’s no way to actually know. We estimate the coefficient with the data we do have, and we show uncertainty about that estimate by giving a range that the coefficient is **probably** within.

Note that using 95% confidence intervals is just a convention. You can create 90% confidence intervals (which will be more narrow), 99% confidence intervals (which will be wider), or whatever intervals you like.

1.11 Hypothesis Testing and p-values

Closely related to confidence intervals is **hypothesis testing**. Generally speaking, you start with a **null hypothesis** and an **alternative hypothesis** (that is opposite the null). Then, you check whether the data supports rejecting the null hypothesis or failing to reject the null hypothesis.
(Note that “failing to reject” the null is not the same as “accepting” the null hypothesis. The alternative hypothesis may indeed be true, except that you just don’t have enough data to show that.)

As it relates to model coefficients, here is the conventional hypothesis test: - null hypothesis: There is no relationship between TV ads and Sales (and thus $\beta_1$ equals zero) - alternative hypothesis: There is a relationship between TV ads and Sales (and thus $\beta_1$ is not equal to zero)

How do we test this hypothesis? Intuitively, we reject the null (and thus believe the alternative) if the 95% confidence interval does not include zero. Conversely, the p-value represents the probability that the coefficient is actually zero:

In [106]: # print the p-values for the model coefficients
   lm.pvalues

Out[106]: Intercept 1.406300e-35
TV 1.467390e-42

dtype: float64

If the 95% confidence interval includes zero, the p-value for that coefficient will be greater than 0.05. If the 95% confidence interval does not include zero, the p-value will be less than 0.05. Thus, a p-value less than 0.05 is one way to decide whether there is likely a relationship between the feature and the response. (Again, using 0.05 as the cutoff is just a convention.)

In this case, the p-value for TV is far less than 0.05, and so we believe that there is a relationship between TV ads and Sales.

Note that we generally ignore the p-value for the intercept.

1.12 How Well Does the Model Fit the data?

The most common way to evaluate the overall fit of a linear model is by the R-squared value. R-squared is the proportion of variance explained, meaning the proportion of variance in the observed data that is explained by the model, or the reduction in error over the null model. (The null model just predicts the mean of the observed response, and thus it has an intercept and no slope.)

R-squared is between 0 and 1, and higher is better because it means that more variance is explained by the model. Here’s an example of what R-squared “looks like”:

You can see that the blue line explains some of the variance in the data (R-squared=0.54), the green line explains more of the variance (R-squared=0.64), and the red line fits the training data even further (R-squared=0.66). (Does the red line look like it’s overfitting?)

Let’s calculate the R-squared value for our simple linear model:

In [107]: # print the R-squared value for the model
   lm.rsquared

Out[107]: 0.61187505085007099

Is that a “good” R-squared value? It’s hard to say. The threshold for a good R-squared value depends widely on the domain. Therefore, it’s most useful as a tool for comparing different models.

1.13 Multiple Linear Regression

Simple linear regression can easily be extended to include multiple features. This is called multiple linear regression:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

Each $x$ represents a different feature, and each feature has its own coefficient. In this case:

$$y = \beta_0 + \beta_1 \times TV + \beta_2 \times Radio + \beta_3 \times Newspaper$$

Let’s use Statsmodels to estimate these coefficients:
In [108]: # create a fitted model with all three features
lm = smf.ols(formula='Sales ~ TV + Radio + Newspaper', data=data).fit()

# print the coefficients
lm.params

Out[108]: Intercept 2.938889
TV 0.045765
Radio 0.188530
Newspaper -0.001037
dtype: float64

How do we interpret these coefficients? For a given amount of Radio and Newspaper ad spending, an increase of $1000 in TV ad spending is associated with an increase in Sales of 45.765 widgets.

A lot of the information we have been reviewing piece-by-piece is available in the model summary output:

In [109]: # print a summary of the fitted model
lm.summary()

Out[109]: <class 'statsmodels.iolib.summary.Summary'>

OLS Regression Results
==============================================================================
Dep. Variable:    Sales      R-squared:     0.897
Model:              OLS       Adj. R-squared: 0.896
Method: Least Squares      F-statistic:    570.3
Date: Fri, 19 Feb 2016     Prob (F-statistic): 1.58e-96
Time: 16:43:14             Log-Likelihood: -386.18
No. Observations: 200      AIC:             780.4
Df Residuals: 196           BIC:             793.6
Df Model: 3
Covariance Type: nonrobust
==============================================================================
                           coef    std err      t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept                 2.9389    0.3120     9.422     0.000         2.3240    3.5539
TV                        0.0458    0.0010    32.809     0.000         0.0430    0.0480
Radio                     0.1885    0.0091    21.893     0.000         0.1721    0.2050
Newspaper                -0.0010    0.0061   -0.1774     0.860        -0.0128    0.0107
==============================================================================
Omnibus: 60.4140, Durbin-Watson: 2.0840,
Prob(Omnibus): 0.0000, Jarque-Bera (JB): 151.2410,
Skew: -1.3270, Prob(JB): 1.44e-33,

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

What are a few key things we learn from this output?

• TV and Radio have significant p-values, whereas Newspaper does not. Thus we reject the null hypothesis for TV and Radio (that there is no association between those features and Sales), and fail to reject the null hypothesis for Newspaper.
- TV and Radio ad spending are both positively associated with Sales, whereas Newspaper ad spending is slightly negatively associated with Sales. (However, this is irrelevant since we have failed to reject the null hypothesis for Newspaper.)
- This model has a higher R-squared (0.897) than the previous model, which means that this model provides a better fit to the data than a model that only includes TV.

### 1.14 Feature Selection

How do I decide which features to include in a linear model? Here’s one idea: - Try different models, and only keep predictors in the model if they have small p-values. - Check whether the R-squared value goes up when you add new predictors.

What are the drawbacks to this approach? - Linear models rely upon a lot of assumptions (such as the features being independent), and if those assumptions are violated (which they usually are), R-squared and p-values are less reliable. - Using a p-value cutoff of 0.05 means that if you add 100 predictors to a model that are pure noise, 5 of them (on average) will still be counted as significant. - R-squared is susceptible to overfitting, and thus there is no guarantee that a model with a high R-squared value will generalize. Below is an example:

```
In [110]: # only include TV and Radio in the model
   : lm = smf.ols(formula='Sales ~ TV + Radio', data=data).fit()
   : lm.rsquared
Out[110]: 0.89719426108289557

In [111]: # add Newspaper to the model (which we believe has no association with Sales)
   : lm = smf.ols(formula='Sales ~ TV + Radio + Newspaper', data=data).fit()
   : lm.rsquared
Out[111]: 0.89721063817895219
```

R-squared will always increase as you add more features to the model, even if they are unrelated to the response. Thus, selecting the model with the highest R-squared is not a reliable approach for choosing the best linear model.

There is alternative to R-squared called adjusted R-squared that penalizes model complexity (to control for overfitting), but it generally under-penalizes complexity.

So is there a better approach to feature selection? Cross-validation. It provides a more reliable estimate of out-of-sample error, and thus is a better way to choose which of your models will best generalize to out-of-sample data. There is extensive functionality for cross-validation in scikit-learn, including automated methods for searching different sets of parameters and different models. Importantly, cross-validation can be applied to any model, whereas the methods described above only apply to linear models.

### 1.15 Linear Regression in scikit-learn

Let’s redo some of the Statsmodels code above in scikit-learn:

```
In [112]: # create X and y
   : feature_cols = ['TV', 'Radio', 'Newspaper']
   : X = data[feature_cols]
   : y = data.Sales

   : # follow the usual sklearn pattern: import, instantiate, fit
   : from sklearn.linear_model import LinearRegression
   : lm = LinearRegression()
   : lm.fit(X, y)
```
```python
# print intercept and coefficients
print(lm.intercept_)
print(lm.coef_)

2.9388936946
[ 0.04576465 0.18853002 -0.00103749]

In [113]: # pair the feature names with the coefficients
   a = zip(feature_cols, lm.coef_)
   print(list(a))

[(\'TV\', 0.045764645455397587), (\'Radio\', 0.18853001691820465), (\'Newspaper\', -0.0010374930424762452)]

In [114]: # predict for a new observation
   lm.predict([100, 25, 25])

/home/ubuntu/anaconda3/lib/python3.5/site-packages/sklearn/utils/validation.py:386: DeprecationWarning: Passing 1d arrays ... data either using X.reshape(-1, 1) if your data has a single feature or X.reshape(1, -1) if it contains a single sample.
DeprecationWarning)

Out[114]: array([ 12.20266701])

In [115]: # calculate the R-squared
   lm.score(X, y)

Out[115]: 0.89721063817895208

Note that p-values and confidence intervals are not (easily) accessible through scikit-learn.

1.16 Handling Categorical Predictors with Two Categories

Up to now, all of our predictors have been numeric. What if one of our predictors was categorical? Let’s create a new feature called Size, and randomly assign observations to be small or large:

In [116]: import numpy as np
   # set a seed for reproducibility
   np.random.seed(12345)

   # create a Series of booleans in which roughly half are True
   nums = np.random.rand(len(data))
   mask_large = nums > 0.5

   # initially set Size to small, then change roughly half to be large
   data[\'Size\'] = 'small'
   data.loc[mask_large, \'Size\'] = 'large'

   data.head()

Out[116]:
            TV   Radio Newspaper   Sales  Size
       0   230.1   37.8    69.2   22.1  large
       1    44.5   39.3    45.1   10.4  small
       2   17.2   45.9    69.3    9.3  small
       3  151.5   41.3    58.5   18.5  small
       4  180.8   10.8    58.4  12.9  large

For scikit-learn, we need to represent all data numerically. If the feature only has two categories, we can simply create a dummy variable that represents the categories as a binary value:
In [117]: # create a new Series called IsLarge
data['IsLarge'] = data.Size.map({'small':0, 'large':1})data.head()

Out[117]:

<table>
<thead>
<tr>
<th></th>
<th>TV</th>
<th>Radio</th>
<th>Newspaper</th>
<th>Sales</th>
<th>Size</th>
<th>IsLarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>230.1</td>
<td>37.8</td>
<td>69.2</td>
<td>22.1</td>
<td>large</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>44.5</td>
<td>39.3</td>
<td>45.1</td>
<td>10.4</td>
<td>small</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>17.2</td>
<td>45.9</td>
<td>69.3</td>
<td>9.3</td>
<td>small</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>151.5</td>
<td>41.3</td>
<td>58.5</td>
<td>18.5</td>
<td>small</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>180.8</td>
<td>10.8</td>
<td>58.4</td>
<td>12.9</td>
<td>large</td>
<td>1</td>
</tr>
</tbody>
</table>

Let’s redo the multiple linear regression and include the IsLarge predictor:

In [118]: # create X and y

feature_cols = ['TV', 'Radio', 'Newspaper', 'IsLarge']
X = data[feature_cols]
y = data.Sales

# instantiate, fit
lm = LinearRegression()
lm.fit(X, y)

# print coefficients
print(list(zip(feature_cols, lm.coef_)))

[('TV', 0.045719820924362747), ('Radio', 0.18872814313427866), ('Newspaper', -0.0010976794483516239), ('IsLarge', 0.057423850854827561)]

How do we interpret the IsLarge coefficient? For a given amount of TV/Radio/Newspaper ad spending, being a large market is associated with an average increase in Sales of 57.42 widgets (as compared to a Small market, which is called the baseline level).

What if we had reversed the 0/1 coding and created the feature ‘IsSmall’ instead? The coefficient would be the same, except it would be negative instead of positive. As such, your choice of category for the baseline does not matter, all that changes is your interpretation of the coefficient.

1.17 Handling Categorical Predictors with More than Two Categories

Let’s create a new feature called Area, and randomly assign observations to be rural, suburban, or urban:

In [119]: # set a seed for reproducibility
np.random.seed(123456)

# assign roughly one third of observations to each group
nums = np.random.rand(len(data))
mask_suburban = (nums > 0.33) & (nums < 0.66)
mask_urban = nums > 0.66
data['Area'] = 'rural'
data.loc[mask_suburban, 'Area'] = 'suburban'
data.loc[mask_urban, 'Area'] = 'urban'
data.head()

Out[119]:

<table>
<thead>
<tr>
<th></th>
<th>TV</th>
<th>Radio</th>
<th>Newspaper</th>
<th>Sales</th>
<th>Size</th>
<th>IsLarge</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>230.1</td>
<td>37.8</td>
<td>69.2</td>
<td>22.1</td>
<td>large</td>
<td>1</td>
<td>rural</td>
</tr>
<tr>
<td>2</td>
<td>44.5</td>
<td>39.3</td>
<td>45.1</td>
<td>10.4</td>
<td>small</td>
<td>0</td>
<td>urban</td>
</tr>
<tr>
<td>3</td>
<td>17.2</td>
<td>45.9</td>
<td>69.3</td>
<td>9.3</td>
<td>small</td>
<td>0</td>
<td>rural</td>
</tr>
<tr>
<td>4</td>
<td>151.5</td>
<td>41.3</td>
<td>58.5</td>
<td>18.5</td>
<td>small</td>
<td>0</td>
<td>urban</td>
</tr>
<tr>
<td>5</td>
<td>180.8</td>
<td>10.8</td>
<td>58.4</td>
<td>12.9</td>
<td>large</td>
<td>1</td>
<td>suburban</td>
</tr>
</tbody>
</table>
We have to represent Area numerically, but we can’t simply code it as 0=rural, 1=suburban, 2=urban because that would imply an ordered relationship between suburban and urban (and thus urban is somehow “twice” the suburban category).

Instead, we create another dummy variable:

In [120]: # create three dummy variables using get_dummies, then exclude the first dummy column
   area_dummies = pd.get_dummies(data.Area, prefix='Area').iloc[:, 1:]

   # concatenate the dummy variable columns onto the original DataFrame (axis=0 means rows, axis=1 means columns)
   data = pd.concat([data, area_dummies], axis=1)

   data.head()

Out[120]:
       TV  Radio  Newspaper  Sales  Size  IsLarge  Area  Area_suburban
0    230.1  37.8     69.2  22.1  large    1      rural          0
1    44.5  39.3     45.1  10.4   small    0      urban          0
2    17.2  45.9     69.3   9.3   small    0      rural          0
3  151.5  41.3     58.5  18.5   small    0      urban          0
4  180.8  10.8     58.4  12.9  large    1   suburban          1

Here is how we interpret the coding: - rural is coded as Area_suburban=0 and Area_urban=0 - suburban is coded as Area_suburban=1 and Area_urban=0 - urban is coded as Area_suburban=0 and Area_urban=1

Why do we only need two dummy variables, not three? Because two dummies captures all of the information about the Area feature, and implicitly defines rural as the baseline level. (In general, if you have a categorical feature with k levels, you create k-1 dummy variables.)

If this is confusing, think about why we only needed one dummy variable for Size (IsLarge), not two dummy variables (IsSmall and IsLarge).

Let’s include the two new dummy variables in the model:

In [121]: # create X and y
   feature_cols = ['TV', 'Radio', 'Newspaper', 'IsLarge', 'Area_suburban', 'Area_urban']
   X = data[feature_cols]
   y = data.Sales

   # instantiate, fit
   lm = LinearRegression()
   lm.fit(X, y)

   # print coefficients
   print(list(zip(feature_cols, lm.coef_)))

[(‘TV’, 0.045744010363313721), (‘Radio’, 0.18786669552525828), (‘Newspaper’, -0.0010876977267108567), (‘IsLarge’, 0.077396607497478995), (‘Area_suburban’, -0.1065629901595863), (‘Area_urban’, 0.26813802165220091)]

How do we interpret the coefficients? - Holding all other variables fixed, being a suburban area is associated with an average decrease in Sales of 106.56 widgets (as compared to the baseline level, which is rural). - Being an urban area is associated with an average increase in Sales of 268.13 widgets (as compared to rural).

A final note about dummy encoding: If you have categories that can be ranked (i.e., strongly disagree, disagree, neutral, agree, strongly agree), you can potentially use a single dummy variable and represent the categories numerically (such as 1, 2, 3, 4, 5).
1.18 What Didn’t We Cover?

- Detecting collinearity
- Diagnosing model fit
- Transforming predictors to fit non-linear relationships
- Interaction terms
- Assumptions of linear regression
- And so much more!

You could certainly go very deep into linear regression, and learn how to apply it really, really well. It’s an excellent way to start your modeling process when working a regression problem. However, it is limited by the fact that it can only make good predictions if there is a linear relationship between the features and the response, which is why more complex methods (with higher variance and lower bias) will often outperform linear regression.

Therefore, we want you to understand linear regression conceptually, understand its strengths and weaknesses, be familiar with the terminology, and know how to apply it. However, we also want to spend time on many other machine learning models, which is why we aren’t going deeper here.

1.19 Resources

- To go much more in-depth on linear regression, read Chapter 3 of An Introduction to Statistical Learning, from which this lesson was adapted. Alternatively, watch the related videos or read my quick reference guide to the key points in that chapter.
- To learn more about Statsmodels and how to interpret the output, DataRobot has some decent posts on simple linear regression and multiple linear regression.
- This introduction to linear regression is much more detailed and mathematically thorough, and includes lots of good advice.
- This is a relatively quick post on the assumptions of linear regression.