Introduction to Regression in R
by Richie Cotton and datacamp

Descriptiive statistics using R
#access the dplyr package
library(dplyr)
swedish_motor_insurance %>%
    summarize_all(mean)

```
# A tibble: 1 x 2
  n_claims total_payment_sek
     <dbl>             <dbl>
1     22.9              98.2
```

For correlation
swedish_motor_insurance %>%
    summarize(
        correlation = cor(n_claims, total_payment_sek))

```
# A tibble: 1 x 1
  correlation
        <dbl>
1       0.881
```

*results in a positive correlation meaning that as claims increase, payments increase

Regression
a class of statistical modesl that let you explore the relationship between a response variable and some explanatory variables
given some explanatory variables, you can make predictions about the value of the response variable

*Jargon
-response variable is also called the dependent variable

this is the variable that you want to predict (y)
-explanatory variable is also called the independent variable
variables that explain how the response variable will change (x)

More important jargon
*linear regression > the response variable is numeric
*logisitc regression > the response variable is logical

Logical variables or boolean variables represent one of two values (true or false)
logical variables are used to store binary information or to express the truth or falsity of a condition or statement
true is denoted as 1
false is denoted as 0

A little on logistic regression:
a statistical method used for modeling the relationship between a binary dependent variable and one or more independent variables. It is a fundamental technique in statistics and machine learning, particularly for classification problems where the goal is to predict a categorical outcome.

Key characteristics of logistic regression include:

1. **Binary Outcome:** Logistic regression is typically used when the dependent variable (response or outcome) is binary, meaning it has only two possible values, often represented as 0 and 1. For example, it is commonly used for problems such as yes/no, true/false, pass/fail, or spam/ham classification.

2. **Log-Odds Transformation:** Logistic regression models the relationship between the independent variables and the probability of the binary outcome being 1. To do this, it uses a log-odds transformation, which converts the probability into the log-odds or logit scale. The logistic function (sigmoid curve) is used to map the log-odds back to a probability.

3. **Independent Variables:** Logistic regression can involve one or more independent variables (also called features or predictors). The relationship between the independent variables and the log-odds of the binary outcome is modeled using a linear equation.

4. **Model Interpretation:** The coefficients (parameters) of the model represent the change in the log-odds of the outcome for a one-unit change in the corresponding independent variable, while holding other variables constant. These coefficients can be used to interpret the impact of each independent variable on the probability of the binary outcome.

5. **Maximum Likelihood Estimation:** The logistic regression model is estimated using maximum likelihood estimation (MLE). This method finds the set of coefficients that maximizes the likelihood of the observed data under the model.

6. **Assumptions:** Logistic regression assumes that the relationship between the independent variables and the log-odds of the outcome is linear. It also assumes that there is no multicollinearity among the independent variables and that the residuals follow a logistic distribution.

7. **Regularization:** Regularized forms of logistic regression, such as L1 (Lasso) and L2 (Ridge) regularization, can be used to prevent overfitting and select important features when dealing with high-dimensional data.

Logistic regression is widely used in fields like epidemiology, medicine, social sciences, and machine learning for tasks such as spam classification, credit scoring, churn prediction, and medical diagnosis. It is a valuable tool for modeling and understanding the relationships between variables in binary classification problems.

simple linear or simple logistic regression means that there is only one explanatory variable

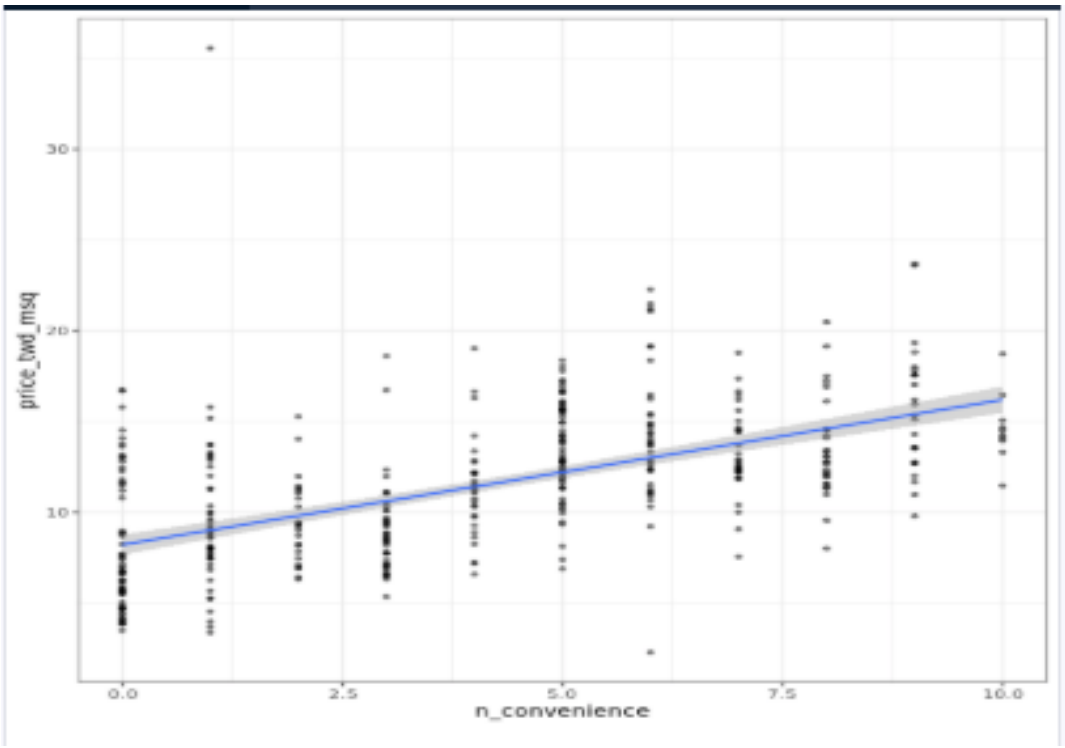
Visualizing pairs of variables
#pull ggplot package
library(ggplot2)
#first argument is the dataframe, second uses the aes method to align independent and dependent variables
#geom_point > builds a scatter plot
#geom_smooth > bulids a trend line, 'lm' sets the method argument to linear model
#'se' > standard error, to FALSE leaves it out
```
ggplot(swedish_motor_insurance, aes(n_claims, total_payment_sek)) +
    geom_point() +
    geom_smooth(
        method = 'lm',
        se = FALSE)
```

'View' function to view the dataset

Example
Add a linear trend line without a confidence ribbon
```
ggplot(taiwan_real_estate, aes(n_convenience, price_twd_msq)) +
```

```
  geom_point(alpha = 0.5) +
  geom_smooth(method='lm')
```

output>



Fitting a linear regression
straight lines are a defining feature of a linear regression
straight lines are completely defined by two properties:

- intercept is the y value when x is zero
- slope is the steepness of the line > equal to the amount y increases if you increase x by 1

*equation for a straight line is > y = intercept + slope * x

Estimating the slope
find the trend line and determine the y-intercept by where it intersects the x-axis
*to estimate the slope we need two points (make it easy and pick points close to gridlines)
then calculate the change in y values between the points
then calculate the change in x values between the points
then divide the change in y by the change in x
this will result in our estimate for the slope

Running a model with R
lm(total_payment_sek ~ n_claims, data = swedish_motor_insurance)

#the first argument is a formula with the response variable to the left and the explanatory variable to the right
#data argument takes in the dataframe
ouput>

```
Coefficients:
(Intercept)     n_claims
     19.994        3.414
```

this result gives us two coefficients (intercept and slope (here slope is denoted under the explanatory variable title)
not written above, but on visualization and estimation this lines up with our guess of 20 and 3.5
y = intercept + m * x
our y is the total payment in Swedish krona
plug and play
total_payment_sek = 19.994 + 3.414 * n_claims
**what this all means? > we expect with every additional claim for the total payment to increase by 3.4

Categorical explanatory variables
*scatter plots are not ideal for categorical data
a histogram is a better option here
example using the fish dataset

```
ggplot(fish, aes(mass_g)) +
    geom_histomgram(bins=9) +
    facet_wrap(vars(species))
```

#facet_wrap gives a panel for each species
#it takes the name of the variable to split on, wrapped in the vars() function

Summary statistics using R, example 'mean mass' by species

```
fish %>%
    group_by(species) %>%
    summarize(mean_mass_g = mean(mass_g))
```

Linear model example

```
lm(mass_g ~ species, data = fish)
```

```
Coefficients:
 (Intercept)  speciesPerch   speciesPike  speciesRoach
       617.8        -235.6         100.9        -465.8
```

*important to break this down because it shows how confusing labeling can become
here the results give us four coefficients
as you can see there are only three of the four fish listed
the reason is that here the intercept is the mean mass of the bream
*it gets its own column because it is used as the intercept and in turn the other three species mass are calculated relative to the bream
*this is why you get negative mass as a result
*this approach can be useful for models with multiple explanatory variables but just plain confusing for simple linear regression

How do we fix this?
lm(mass_g ~ species + 0, data = fish)
*this specifies that all the coefficients should be given relative to 0
*this also means that we are fitting a linear regression without an intercept term
output>

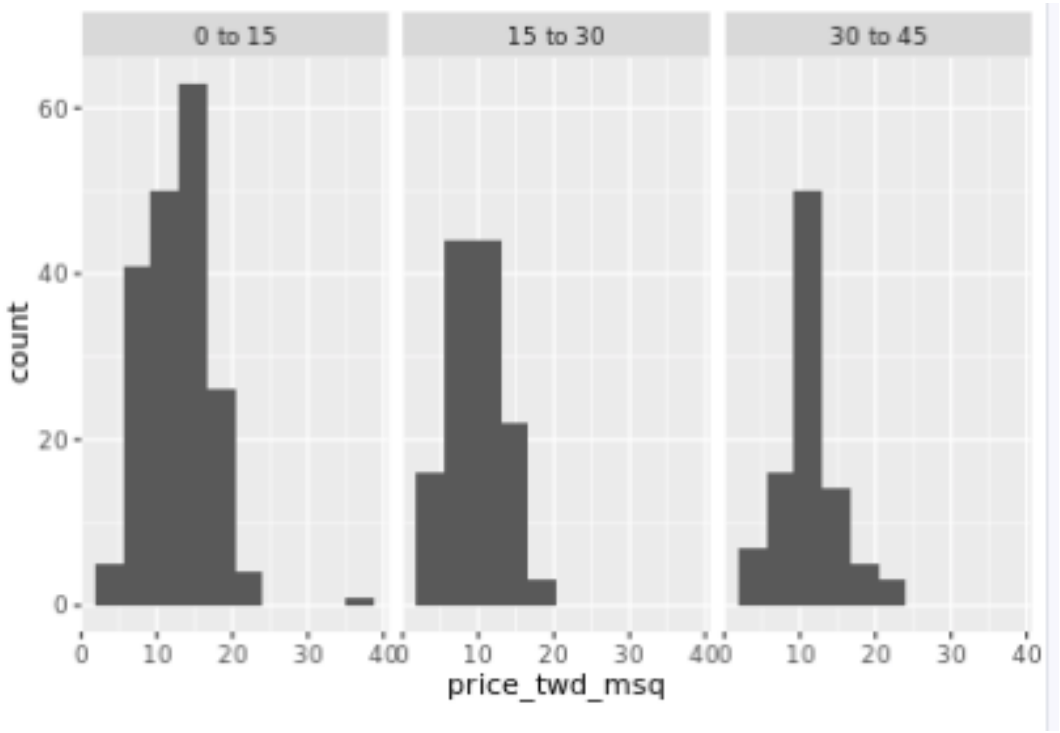
```
Coefficients:
speciesBream  speciesPerch   speciesPike  speciesRoach
       617.8         382.2         718.7         152.0
```

**key note > when you have a single categorical explanatory variable, the linear regression coefficients are the means of each category

Example

```
# Using taiwan_real_estate, plot price_twd_msq
ggplot(taiwan_real_estate, aes(price_twd_msq)) +
  # Make it a histogram with 10 bins
  geom_histogram(bins = 10) +
  # Facet the plot so each house age group gets its own panel
  facet_wrap(vars(house_age_years))
```

ouput>



Example
summary_stats <- taiwan_real_estate %>%
  # Group by house age
  group_by(house_age_years) %>%
  # Summarize to calculate the mean house price/area
  summarize(mean_by_group = mean(price_twd_msq))

See the result
summary_stats

Making predictions
the principle behind predicting is to ask questions of the form > If I set the explanatory variables to these values, what value would the response variable have?
example using our fish dataset
using explanatory variable ‘length of fish’
we ask the question what is the mass of a fish at parameters of our choosing (we choose between 20 and 40cm)
we use ‘tibble’ which is a data frame variant that is easier to work with
R:
explanatory_data <- tibble(length_cm = 20:40)
#now call ‘predict’
predict(mdl_mass_vs_length, explanatory_data)
#predict returns a vector of predictions, one for each row of the explanatory data

output>

```
       1          2          3          4          5          6
 55.65205  110.20203  164.75202  219.30200  273.85198  328.40196
       7          8          9         10         11         12
382.95194  437.50192  492.05190  546.60188  601.15186  655.70184
      13         14         15         16         17         18
710.25182  764.80181  819.35179  873.90177  928.45175  983.00173
      19         20         21
1037.55171 1092.10169 1146.65167
```

this vector data is hard to work with
it is best to place it into the data frame alongside the explanatory variables
like this:

```
prediction_data <- explanatory_data %>%
    mutate(mass_g = predict(mdl_mass_vs_length, explanatory_data))
```

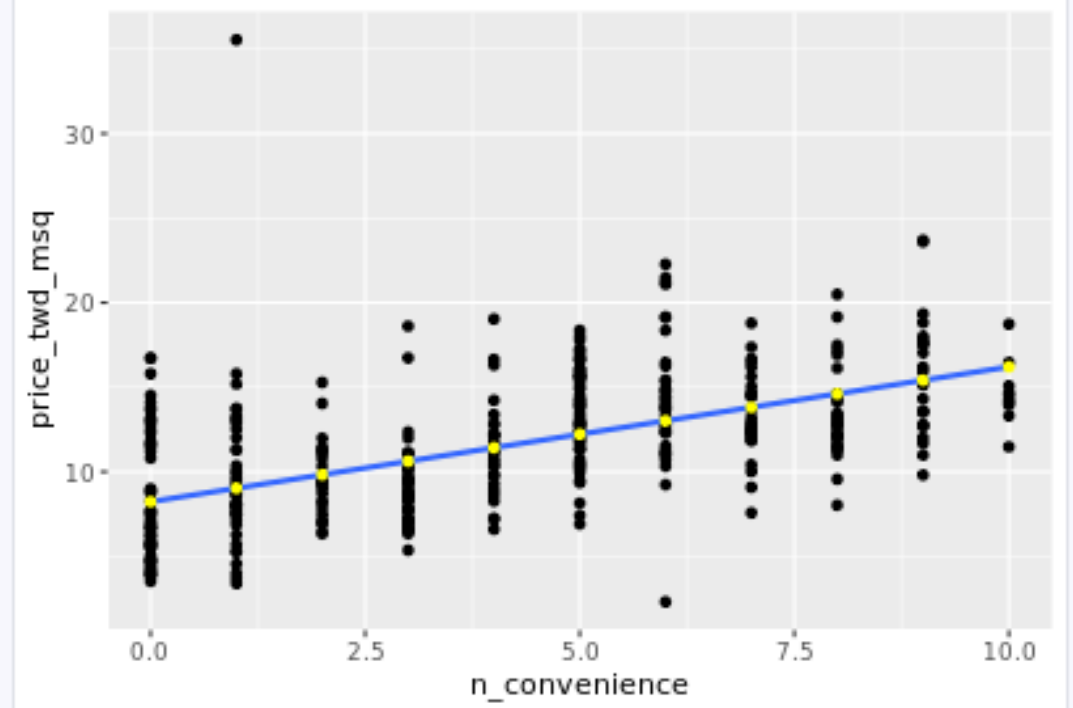
ouput>

```
# A tibble: 21 x 2
   length_cm mass_g
       <int>  <dbl>
 1        20   55.7
 2        21  110.
 3        22  165.
 4        23  219.
 5        24  274.
 6        25  328.
 7        26  383.
 8        27  438.
 9        28  492.
10        29  547.
# ... with 11 more rows
```

showing these predictions

```
ggplot(bream, aes(length_cm, mass_g)) +
    geom_point() +
```

```
        geom_smooth(method = ‘lm’, se = FALSE) +
        geom_point(data = prediction_data, color = ‘blue’)
# we’ ve created the original scatter plot then overlayed our prediction data as
blue dots
*notice that the predictions follow the trend line exactly
this allows us to make predictions outside the observed data range (this is called
extrapolating)
*remember that extrapolating can lead to some ridiculous results > you always
need to know the context of the data that you are working in

Example
# Create a tibble with n_convenience column from zero to ten
explanatory_data <- tibble(
  n_convenience = 0:10
)

# Edit this, so predictions are stored in prediction_data
predict(mdl_price_vs_conv, explanatory_data)

# See the result
prediction_data <- explanatory_data %>%
  mutate(
    price_twd_msq = predict(mdl_price_vs_conv, explanatory_data)
  )

# Add to the plot
ggplot(taiwan_real_estate, aes(n_convenience, price_twd_msq)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  # Add a point layer of prediction data, colored yellow
  geom_point(data = prediction_data, color = 'yellow')

output>
```



Working with model objects
example
mdl_mass_vs_length <- lm(mass_g ~ length_cm, data = bream)
#can then pull the coefficients with the coefficients function
coefficients(mdl_mass_vs_length)

'fitted values' is jargon for predictions on the original dataset used to create the model
fitted(mdl_mass_vs_length
output>

1	2	3	4	5
230.2120	273.8520	268.3970	399.3169	410.2269
6	7	8	9	10
426.5919	426.5919	470.2319	470.2319	519.3269
11	12	13	14	15
513.8719	530.2369	552.0569	573.8769	568.4219
16	17	18	19	20
568.4219	622.9719	622.9719	650.2468	655.7018
21	22	23	24	25
672.0668	677.5218	682.9768	699.3418	704.7968
26	27	28	29	30
699.3418	710.2518	748.4368	753.8918	792.0768
31	32	33	34	35
873.9018	873.9018	939.3617	1004.8217	1037.5517

*shortcut for taking the explanatory variable columns from the dataset, then feeding them to the predict function

‘residuals’ are a measure of inaccuracy in the model fit
residuals(mdl_mass_vs_length)
*each residual is the actual response value minus the predicted response value
output>

1	2	3	4	5
11.788	16.148	71.603	-36.317	19.773
6	7	8	9	10
23.408	73.408	-80.232	-20.232	-19.327
11	12	13	14	15
-38.872	-30.237	-52.057	-233.877	31.578
16	17	18	19	20
31.578	77.028	77.028	-40.247	-5.702
21	22	23	24	25
-97.067	7.478	-62.977	-19.342	-4.797
26	27	28	29	30
25.658	9.748	-34.437	96.108	207.923
31	32	33	34	35

equivalent to >
bream$mass_g - fitted(mdl_mass_vs_length)

summary function
shows an extended printout of the details of the model
*key points from the printout >
residuals > if the model is a good fit, the residuals should follow a normal distribution
*median should be close to 0 and 1Q and 3Q should have similar absolute values
summary() is meant to be read not to manipulate

In R to manipulate functions should return vectors or dataframes
we can do this with >
library(broom)
#broom package provides functions that return data frames
tidy(mdl_mass_vs_length)
#tidy() returns the coefficient details in a data frame

augement(mdl_mass_vs_length)
#one row for each row of the data frame used to create the model
output>

```
# A tibble: 35 × 8
   mass_g length_cm .fitted .resid   .hat .sigma .cooksd .std.resid
    <dbl>     <dbl>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>      <dbl>
 1    242      23.2    230.   11.8 0.144    75.3 0.00247      0.172
 2    290      24      274.   16.1 0.119    75.2 0.00364      0.232
 3    340      23.9    268.   71.6 0.122    74.1 0.0738       1.03
 4    363      26.3    399.  -36.3 0.0651   75.0 0.00894     -0.507
 5    430      26.5    410.   19.8 0.0616   75.2 0.00248      0.275
 6    450      26.8    427.   23.4 0.0566   75.2 0.00317      0.325
 7    500      26.8    427.   73.4 0.0566   74.1 0.0311       1.02
 8    390      27.6    470.  -80.2 0.0452   73.9 0.0291      -1.11
 9    450      27.6    470.  -20.2 0.0452   75.2 0.00185     -0.279
10    500      28.5    519.  -19.3 0.0360   75.2 0.00132     -0.265
# ... with 25 more rows
```

glance(mdl_mass_vs_length)
#returns model-level results
output>

```
# A tibble: 1 × 12
  r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
      <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
1     0.878         0.874  74.2      238. 1.22e-16     1  -199.  405.  409.
# ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Example
# Get the coefficients of mdl_price_vs_conv
coeffs <- coefficients(mdl_price_vs_conv)

# Get the intercept
intercept <- coeffs[1]

# Get the slope
slope <- coeffs[2]

explanatory_data %>%
  mutate(
    # Manually calculate the predictions
    price_twd_msq = intercept + (slope * n_convenience)

)

# Compare to the results from predict()
predict(mdl_price_vs_conv, explanatory_data)

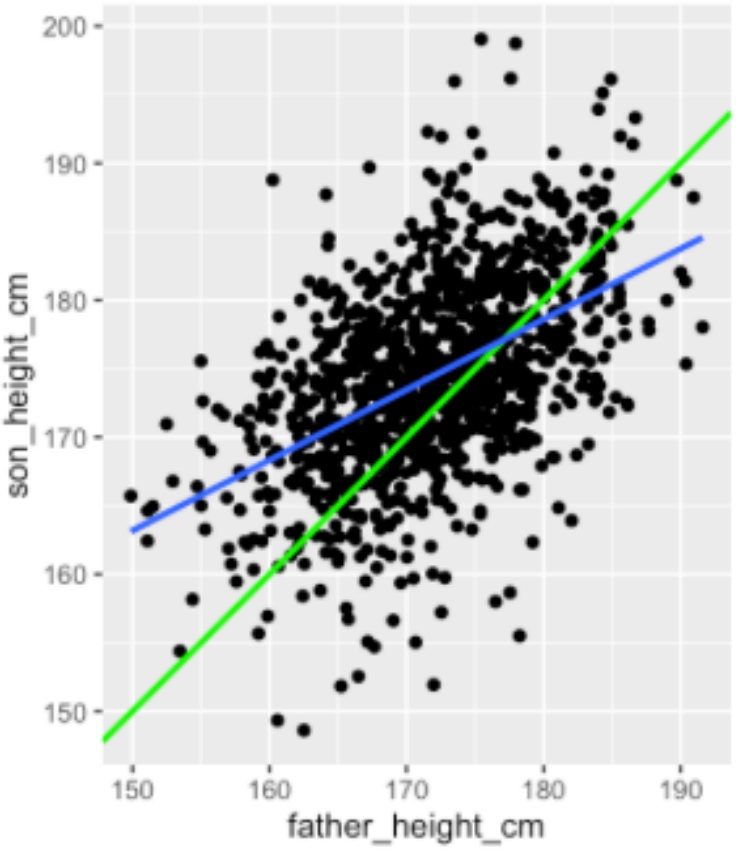
Regression to the mean
is a property of the data, not a type of model
linear regression can be used to quantify its effect
the concept > response value = fitted value + residual
again fitted value is the prediction by the model and residual is how much the model missed by
another way of looking at it > the stuff you explained + the stuff you couldn't explain

Why do residuals exist?
usually two reasons > problems with the model or fundamental randomness
a degree of randomness is our friend because it appropriately mimics the real world
outliers are often due to randomness
*extremes don't persist over time > the idea being that eventually their luck runs out
*this is the core of what we mean by the regression to the mean

A famous way of explaining this concept is using heritable height
we can use the Pearson father/son height pair dataset (reason for the infamy - Pearson of Pearson coeff fame)
with this data set we can ask do short fathers have short sons or do tall fathers have tall sons?
example

```
plt_son_vs_father <- ggplot(father_son, aes(father_height_cm, son_height_cm)) +
      geom_point() +
      geom_abline(color = 'green', size=1) +
      coord_fixed()
#abline represents a line where sons and fathers heights are equal
#coord_fixed ensures that one centimeter on the x-axis appears the same as one centimeter on the y-axis
#going to add a regression line
plt_son_vs_father +
      geom_smooth(method = 'lm', se = FALSE)
```

output>

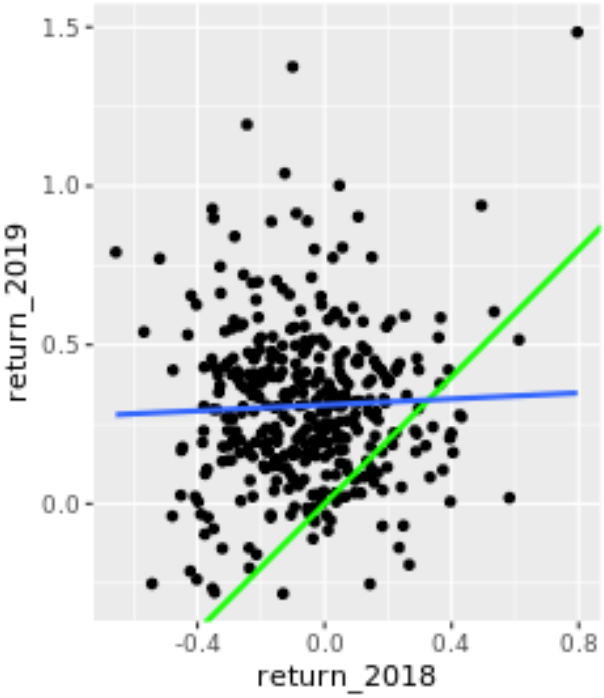


what does this tell us?
the blue regression line tells us that short fathers tend to have taller sons and the tall fathers tend to have shorter sons
*there is this invisible pull back to the center, ie the regression to the mean

Example

```
# Using sp500_yearly_returns, plot return_2019 vs. return_2018
ggplot(sp500_yearly_returns, aes(return_2018, return_2019)) +
  # Make it a scatter plot
  geom_point() +
  # Add a line at y = x, colored green, size 1
  geom_abline(color='green', size=1) +
  # Add a linear regression trend line, no std. error ribbon
  geom_smooth(method='lm', se = FALSE) +
  # Fix the coordinate ratio
  coord_fixed()
```

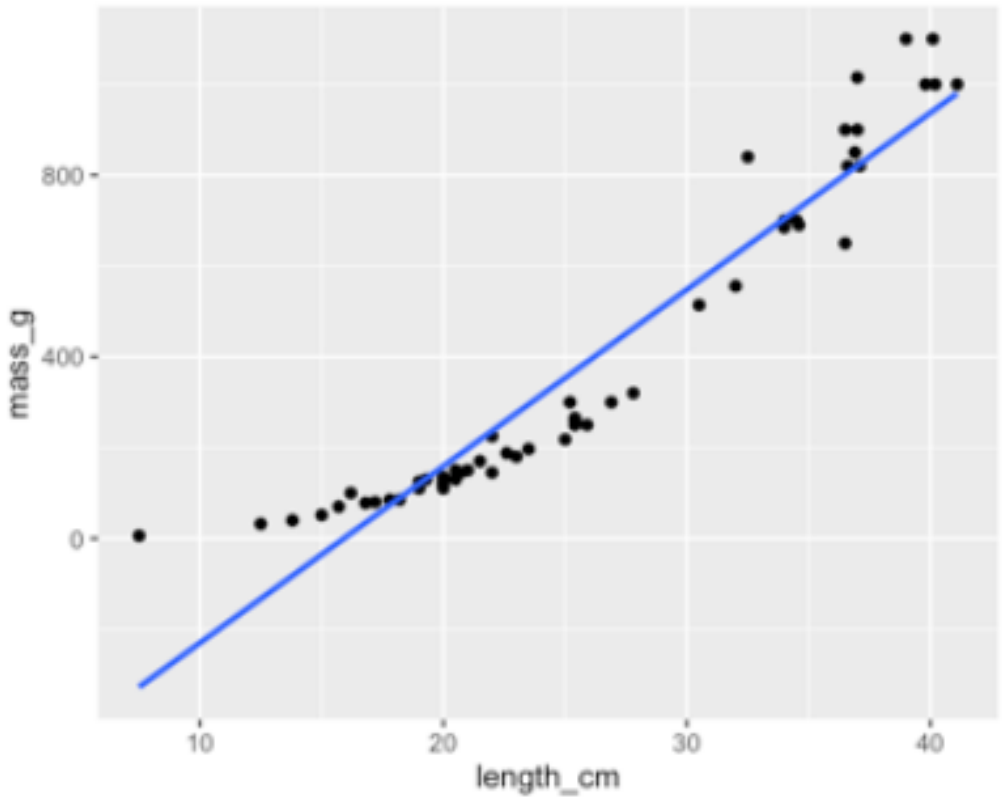
output>



*clear signs that past performance is no guarantee of future results

```
# Run a linear regression on return_2019 vs. return_2018 using
sp500_yearly_returns
mdl_returns <- lm(
  return_2019 ~ return_2018,
  data = sp500_yearly_returns
)

# Create a data frame with return_2018 at -1, 0, and 1
explanatory_data <- tibble(return_2018 = -1:1)

# Use mdl_returns to predict with explanatory_data
predict(mdl_returns, explanatory_data)
```

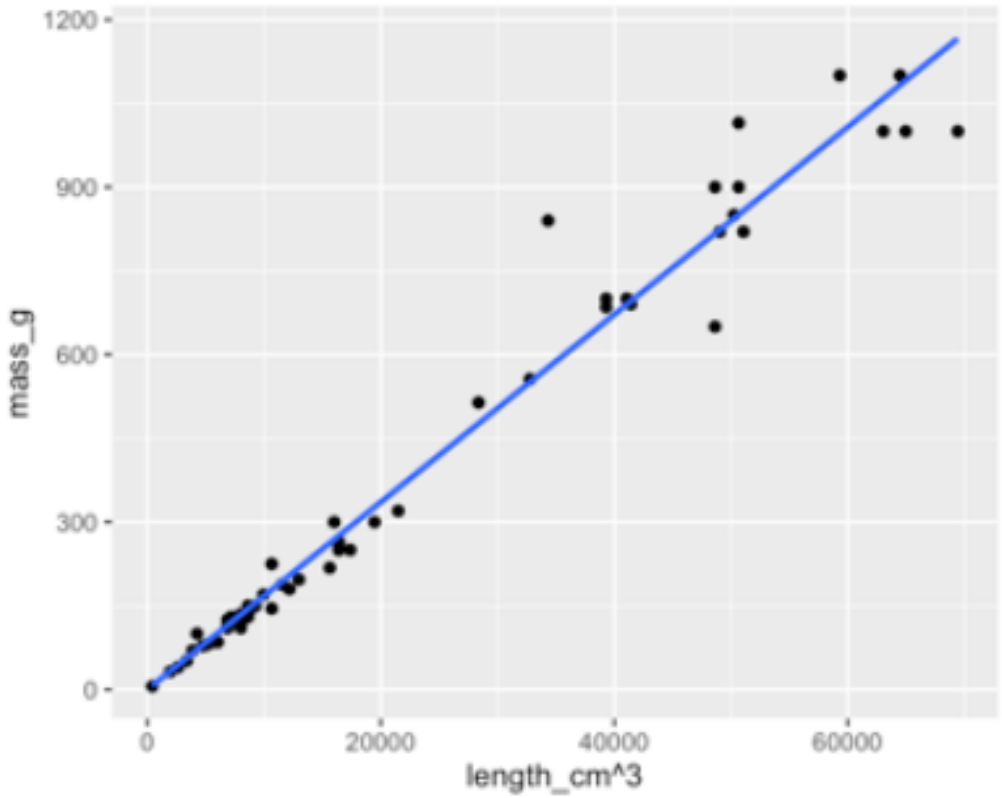
Transforming variables
what happens when the relationship is not linear?
in these instances you may need to transform the explanatory variable or the response variable or both of them
example comparing mass vs length in the perch fish



why does it curve?
possibly mass on the perch is being affected not just by length but also by width and height
mass is being affected by 3 dimensions instead of just 1
how can we update this to change the relationship?
we can cube length to represent that the perch is growing in 3 directions

```
ggplot(perch, aes(length_cm ^ 3, mass_g)) +
    geom_point() +
    geom_smooth(method = 'lm', se = FALSE)
```

output>



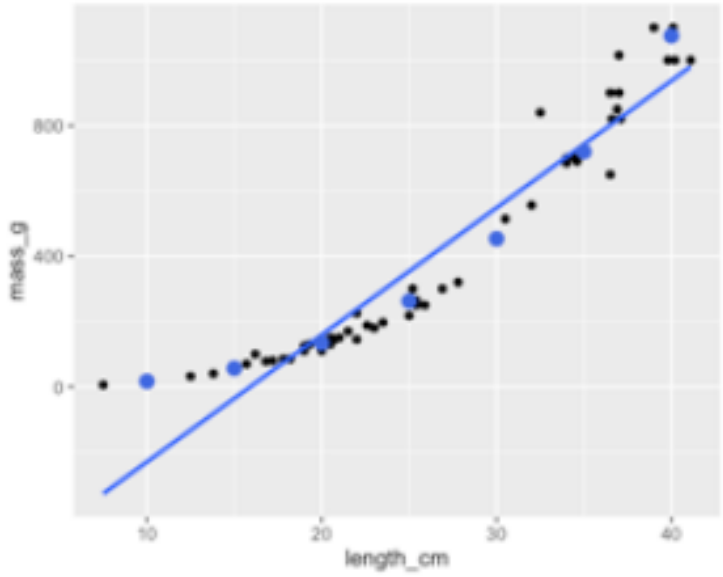
we now have our linear relationship

*to model a variable that is to an exponent with R you need to change specific syntax
mdl_perch <- lm(mass_g ~ I(length_cm ^ 3), data = perch)
#'I' is the 'I' function, it represents 'as is'

Predicting mass vs length
*in R need to specify just the lengths, not the lengths cubed
R will take care of the transformation automatically
explanatory_data <- tibble(length_cm = seq(10, 40, 5))
predictions code is the same
prediction_data <- explanatory_data %>%
    mutate(
        mass_g = predict(mdl_perch, explanatory_data))

*Now using above prediction data on your original non-transformed linear model has non-linear predictions
ggplot(perch, aes(length_cm, mass_g)) +
    geom_point() +
    geom_smooth(method = 'lm', se = FALSE) +
    geom_point(data = prediction_data, color = 'blue')

output>

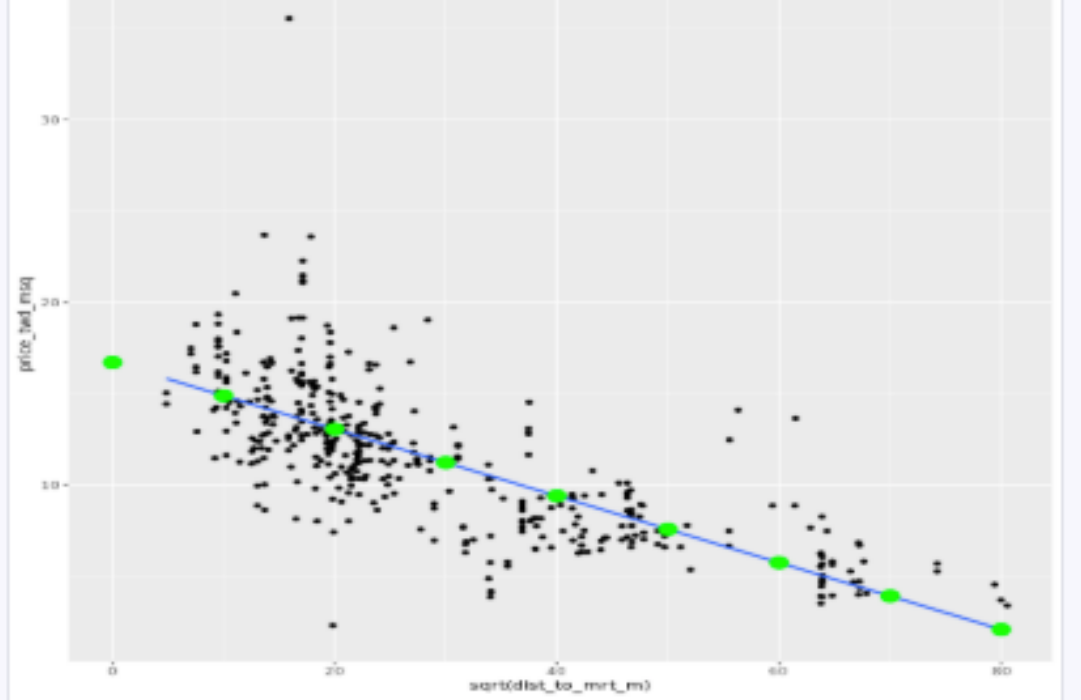


*tip for transforming right skewed distribution data > use sqrt

Example

```
# From previous steps
mdl_price_vs_dist <- lm(
  price_twd_msq ~ sqrt(dist_to_mrt_m),
  data = taiwan_real_estate
)
explanatory_data <- tibble(
  dist_to_mrt_m = seq(0, 80, 10) ^ 2
)
prediction_data <- explanatory_data %>%
  mutate(
    price_twd_msq = predict(mdl_price_vs_dist, explanatory_data)
  )

ggplot(taiwan_real_estate, aes(sqrt(dist_to_mrt_m), price_twd_msq)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  # Add points from prediction_data, colored green, size 5
  geom_point(data = prediction_data, color = 'green', size = 5)
```

output>



Example

```
# From previous steps
mdl_click_vs_impression <- lm(
  I(n_clicks ^ 0.25) ~ I(n_impressions ^ 0.25),
  data = ad_conversion
)
explanatory_data <- tibble(
  n_impressions = seq(0, 3e6, 5e5)
)
prediction_data <- explanatory_data %>%
  mutate(
    n_clicks_025 = predict(mdl_click_vs_impression, explanatory_data),
    n_clicks = n_clicks_025 ^ 4
  )

ggplot(ad_conversion, aes(n_impressions ^ 0.25, n_clicks ^ 0.25)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  # Add points from prediction_data, colored green
  geom_point(data=prediction_data, color='green')
```

Quantifying model fit

how strong is the linear relationship

first metric >

coefficient of determination or r-squared or R-squared
just for history sake > little r for simple linear regression and R for multiple explanatory variables
r-squared > the proportion of the variance in the response variable that is predictable from the explanatory variable
how it is read >
- 1 means a perfect fit
- 0 means the worst possible fit (no better than randomness)

as always the context of the data will determine how good a score is
0.5 for a human subjective analysis may be considered a good score
where 0.9 for a less complex process may be considered a bad score

Coefficient of determination is referred to as “Multiple R-squared’ in the summary() print out
we can pull it using the glance function

```
library(broom)
library(dplyr)
mdl_bream %>%
    glance() %>%
    pull(r.squared)
```

*For simple linear regression, the interpretation of the coefficient of determination is simply the correlation between the explanatory and response variables, squared

```
bream %>%
    summarize(
        coeff_determination = cor(length_cm, mass_g) ^ 2)
```

Residual standard error (RSE) is the second metric
a typical difference between a prediction and an observed response
ie how much the predictions are typically wrong by
*has the same unit as the response variable
how to pull >

```
mdl_bream %>%
    glance() %>%
    pull(sigma)
```

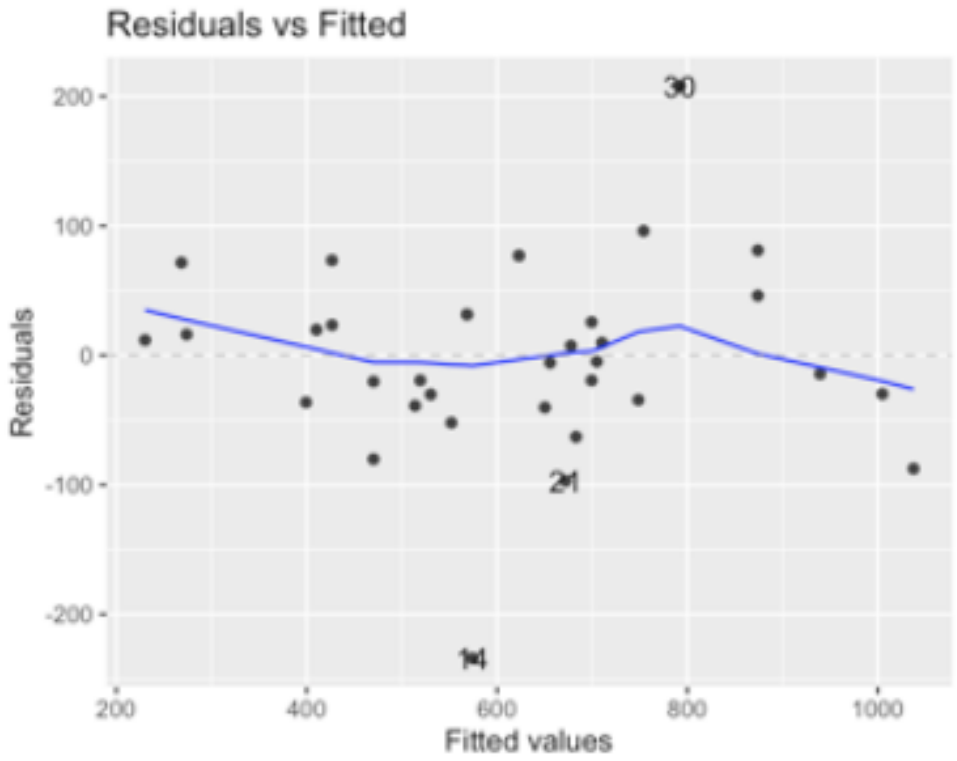
Manually calculating using R:

```
bream %>%
    mutate(
        residuals_sq = residuals(mdl_bream) ^ 2
    ) %>%
    summarize(
        resid_sum_of_sq = sum(residuals_sq),
        deg_freedom = n() -2,
```

rse = sqrt(resid_sum_of_sq / deg_freedom))
#degrees of freedom is the number of observations minus the number of model coefficients
for our pull and our manual calculation on RSE for bream was approx 74g
what this means? >
that typically the difference between predicted bream masses and observed masses is about 74g

Root-mean-square error (RMSE)
same as RSE but at the end you do not subtract the number of coefficients from n()
used to quantify how inaccurate the model predictions are
however it is worse at comparisons between models
*RSE is typically better and used more often

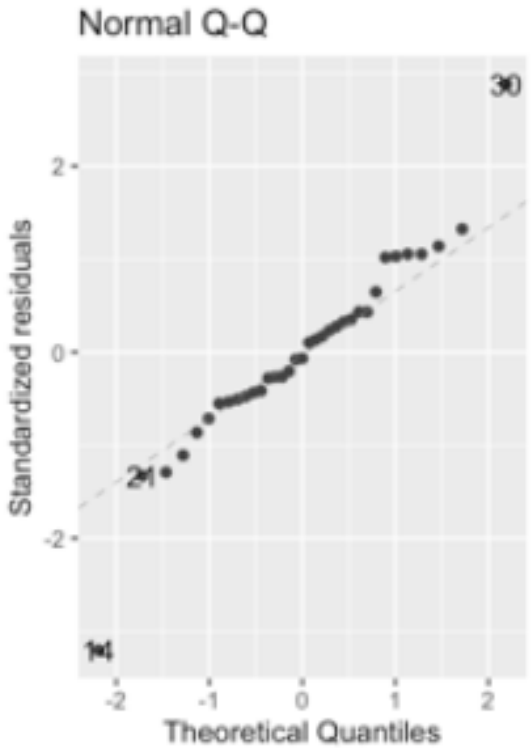
Visualizing model fit
if a linear regression model is a good fit, then the residuals are approximatedly normally distributed, with mean zero
we can use diagnostic plots to help us with this
first is >
plot residuals vs. fitted values



the blue line is a LOESS trend line
if residuals met the assumption that they are normally distributed with mean zero,

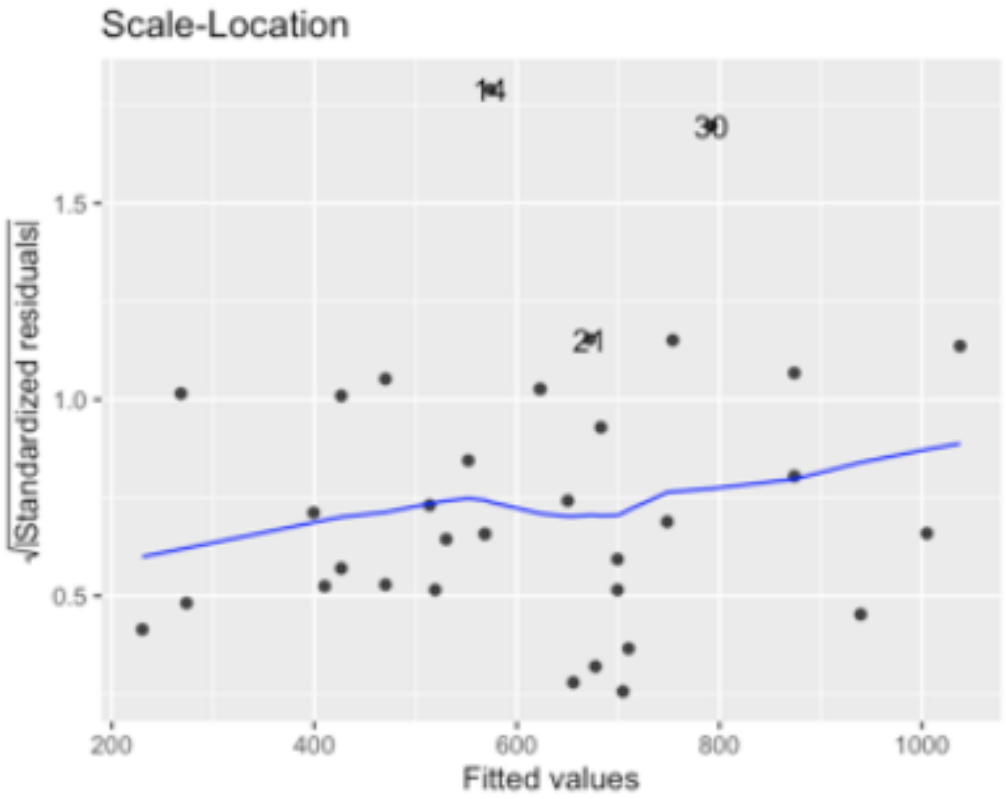
then the trend line (like above) will closely follow the y = 0 line on the plot

Another diagnostic tool is the Q-Q plot



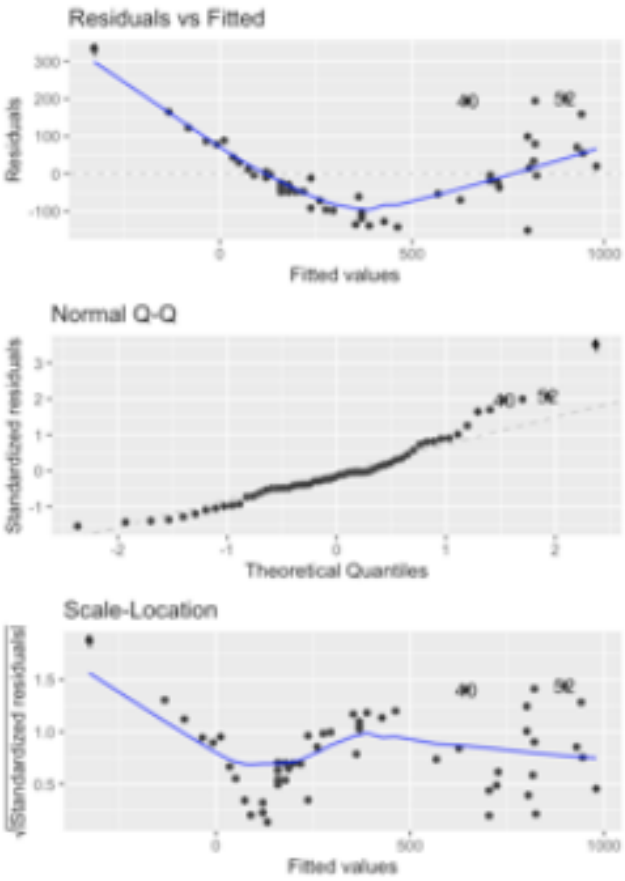
this Q-Q plot follows a normal distribution
the x-axis represents quantiles
y-axis represents standarized residuals (this is the residuals divided by their standard deviation)
if the points track along the dotted line then they are normally distributed
above we have two outliers which are labeled 14 and 30 (numbers corresponding to respecitve rows)

Scale-location plot



shows the square root of the standardized residuals versus the fitted values
shows whether the size of the residuals gets bigger or smaller

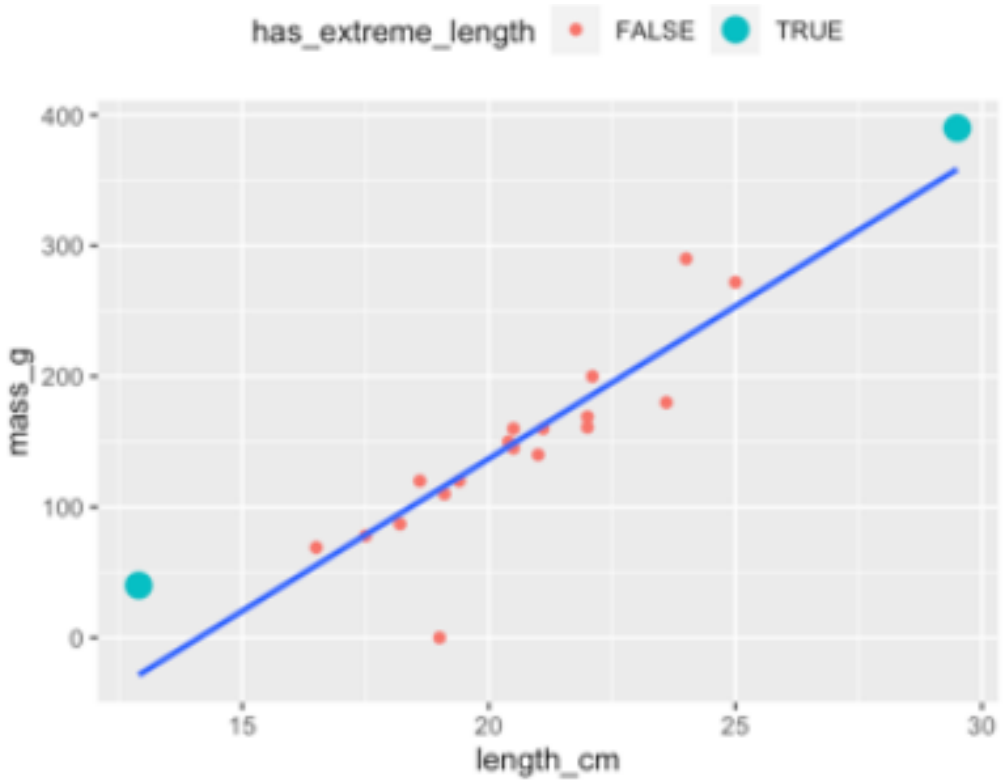
Using R with these features
library(ggplot2)
library(ggfortify)
#this is where things get a little complicated, the which argument uses numbers instead of descriptive names
#which = 1 for residuals vs fitted, 2 for q-q plot, and 3 for scale-location
#nice thing is you can draw all three at once
autoplot(model_perch, which = 1:3, nrow=3, ncol=1)
output>



Outliers, leverage, and influence
using R for extreme explanatory values

```
roach %>%
    mutate(
        has_extreme_length = length_cm < 15 | length_cm > 26) %>%
    ggplot(aes(length_cm, mass_g)) +
    geom_point(aes(color = has_extreme_length)) +
    geom_smooth(method = 'lm', se = FALSE)
```

ouput>



Leverage
is a measure of how extreme the explanatory variable values are
for one explanatory variable extremes can be found with filtering
with many explanatory variables the math becomes complicated and requires a model object
example

```
mdl_roach <- lm(mass_g ~ length_cm, data = roach)
hatvalues(mdl_roach)
```

#leverage for historical reasons is called hatvalues
returns a numeric vector with as many values as there are observations

Finding values with leverage

```
mdl_roach %>%
    augment() %>%
    select(mass_g, length_cm, leverage = .hat) %>%
    arrange(desc(leverage)) %>%
    head()
```

#we can find leverage within the augment method
#'select' chooses the columns of interest (mass_g, length_cm, .hat (renamed in our argument as leverage)
#'arrange' in descending order to get the largest leveraged values at the top

Influence
measures how much the model would change if you left the observation out of the dataset when modeling
called a ‘leave one out’ metric
resembles torque
the influence of each observation is based on the size of the residuals and the leverage

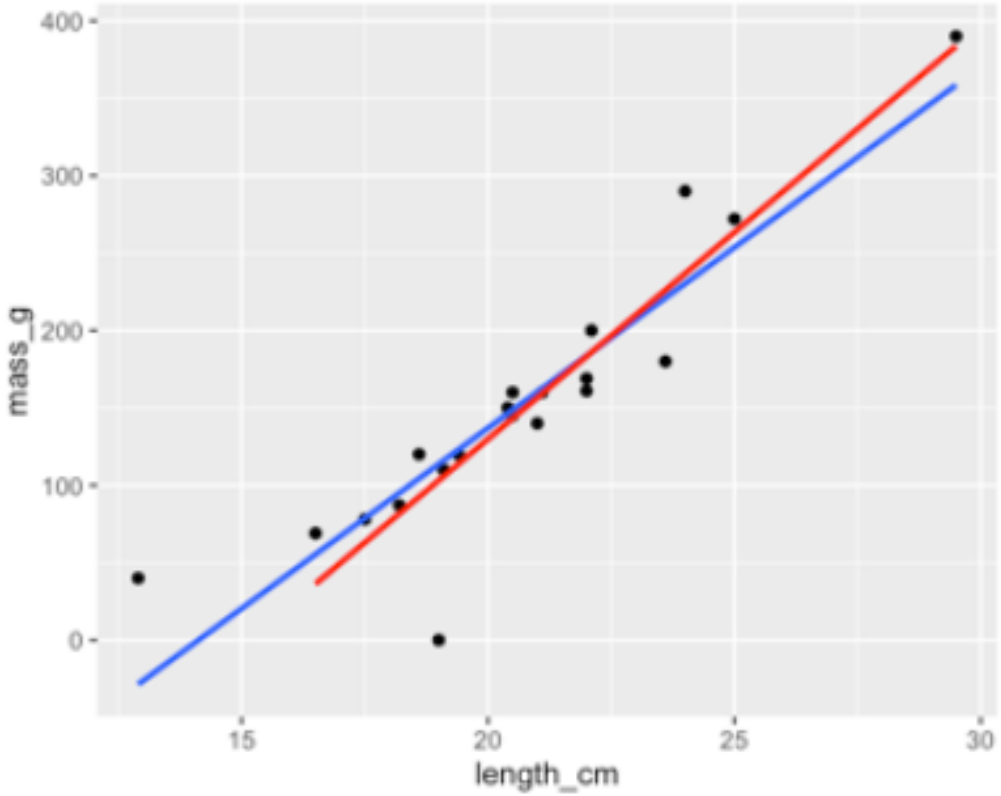
Cook’s distance
the most common measure of influence
based on the size of the residuals and the leverage
*bigger number denotes more influence for the observation
cooks.distance(mdl_roach)
returns values as a vector
R example (just like above)

```
mdl_roach %>%
    augment() >%>
    select(mass_g, length_cm, cooks = .cooksd) %>%
    arrange(desc(cooks)) %>%
    head()
```

Removing the most influential roach

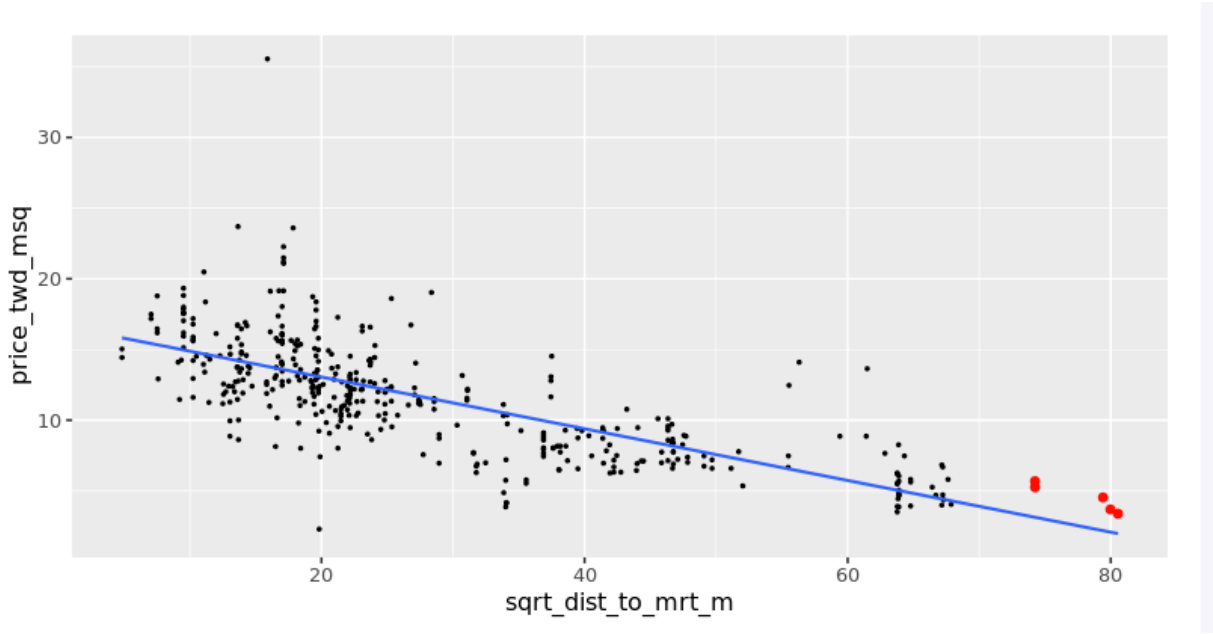
```
roach_not_short <- roach %>%
    filter(length != 12.9)
ggplot(roach, aes(length_cm, mass_g)) +
    geom_point() +
    geom_smooth(method = ‘lm’, se = FALSE) +
    geom_smooth(method = ‘lm, se = FALSE, data = roach_not_short, color = ‘red’)
```

output>



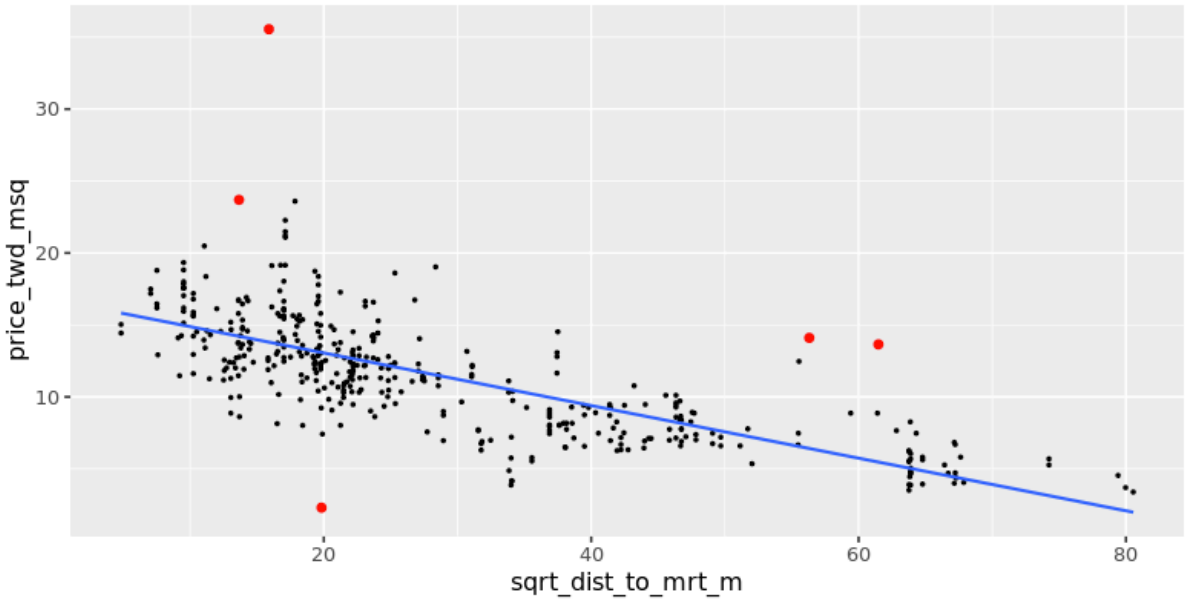
show us the change in regression line if we were to remove the outlier roach of 12.9cm
using autoplot for influence:
#which argument 4 = Cook's distance, 5 = Residuals vs Leverage, 6 = Cook's dist vs Leverage
autoplot(mdl_roach, which = 4:6, nrow=3, ncol=1)
these really just show clearly the labels of the most influential observations

High leverage points in red



Observations with a large distance to the nearest MRT station have the highest leverage, because most of the observations have a short distance, so long distances are more extreme.

High influence points in red

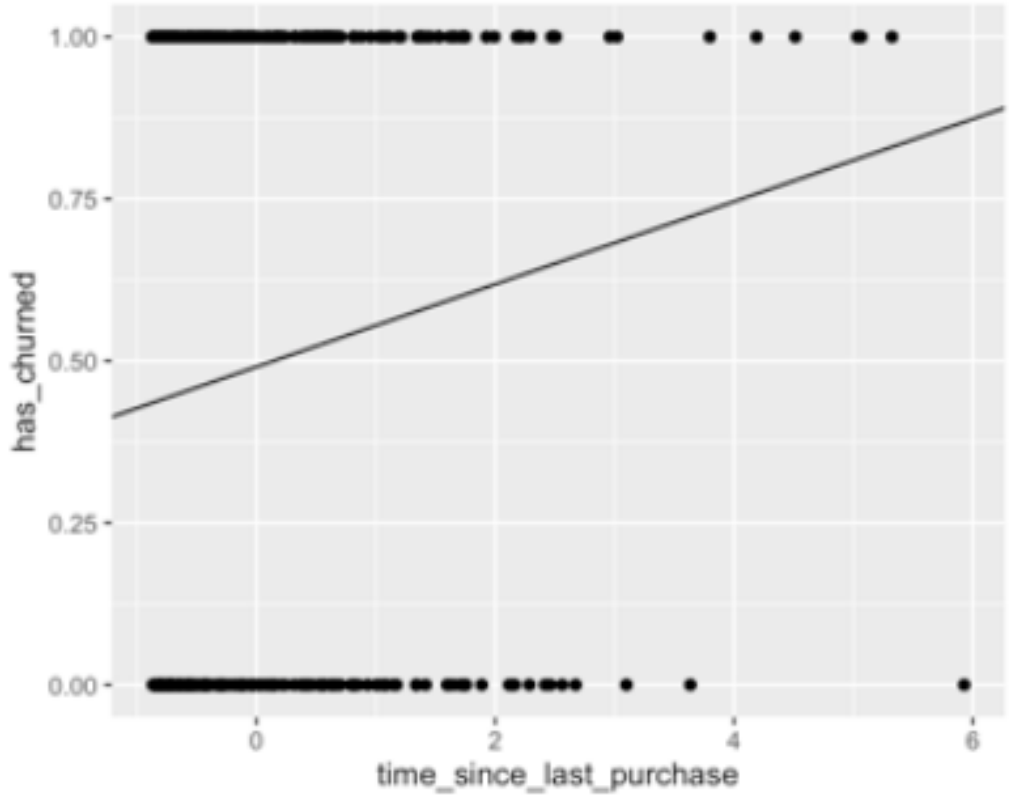


Observations with predictions far away from the trend line have high influence, because they have large residuals and are far away from other observations

Logistic Regression
example with churn dataset assessing open and closed banking accounts
has_churned values 0 or 1 with churn = 1, not churned = 0

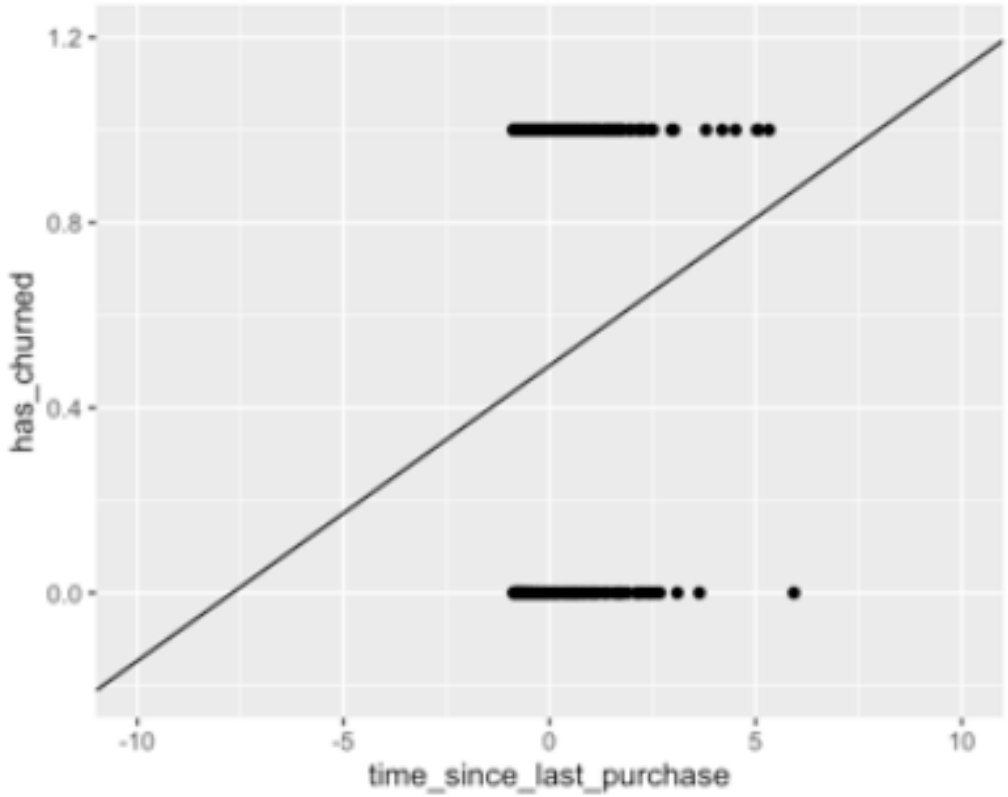
```
mdl_churn_vs_recency_lm <- lm(has_churned ~ time_since_last_purchase, data = churn)
coeffs <- coefficients(mdl_churn_vs_recency_lm)
intercept <- coeffs[1]
slope <- coeffs[2]
#visualize
ggplot(churn, aes(time_since_last_purchase, has_churned)) +
      geom_point() +
      geom_abline(intercept = intercept, slope = slope)
```

#*key use geom_abline over geom_smooth s that the line isn't limited to the extent of the data



*predictions are probabilities of churn, not amounts of churn
model predictions are fractional here

What happens if we use a linear model here?



*with a linear model we will get negative probabilities or probabilities greater than 1 > both of which are impossible
this is a job for logistic regression

Logistic regression
is another type of generalized linear model
used when the response variable is logical
*responses follow logistic (S-shaped) curve
with R:
mdl_recency_glm <- glm(has_churned ~ time_since_last_purchase, data = churn, family = binomial)
#run glm (stands for generalized linear model)
#family argument gives you several options such as gaussian
#for logistic regression we want the family argument to be set to 'binomial'
this will give us a report specifying coefficients
visualize (add to above code)
+
geom_smooth(method = 'glm', se = FALSE, method.args = list(family = binomial))
#this adds a logistic regression trend line
#method.args argument contains a list of other arguments passed to glm
*now our logistic (S) curve never goes below 0 or 1

*what this example tells us is that closer customers are to a purchase the less likely they are to churn

Example

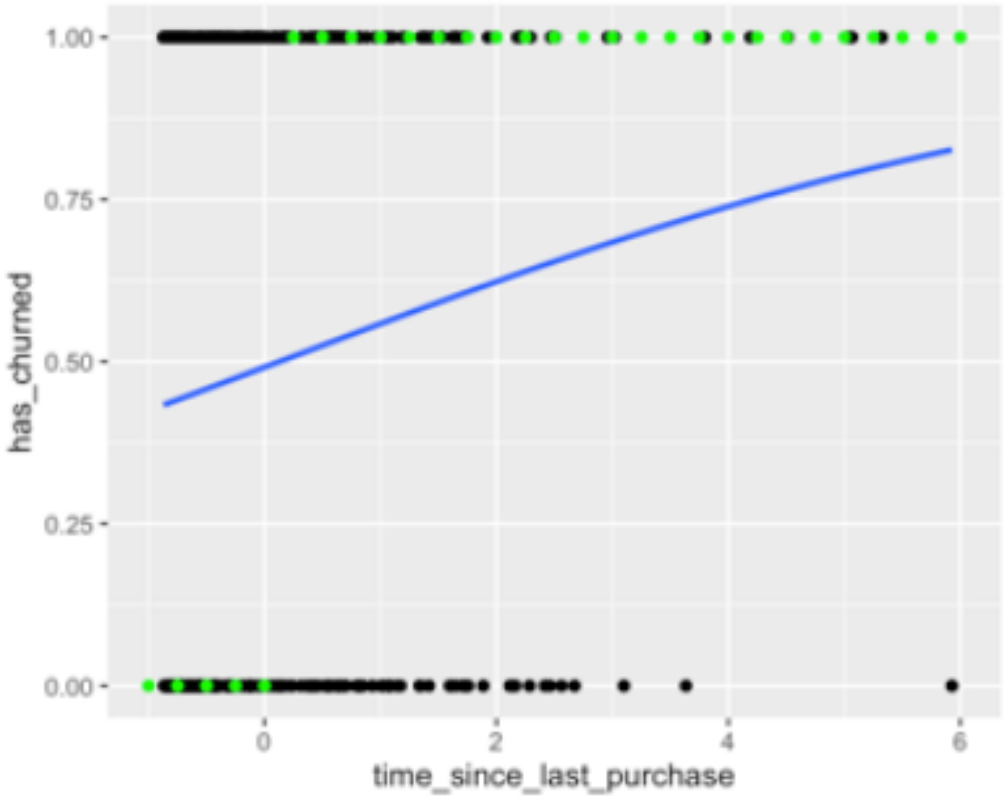
```
# Using churn, plot time_since_last_purchase
ggplot(churn, aes(time_since_last_purchase)) +
  # as a histogram with binwidth 0.25
  geom_histogram(binwidth = 0.25) +
  # faceted in a grid with has_churned on each row
  facet_grid(rows = vars(has_churned))
```

Predictions and odds ratio
getting the most likely outcome
that is, if the probability of churning is less than 0.5, the most likely outcome is that they won't churn
if probability is greater than 0.5, it's more likely that they will churn
we get this by rounding the predicted probabilities

```
prediction_data <- explanatory_data %>%
        mutate(has_churned = predict(mdl_recency, explanatory_data, type =
'respones),
        most_likely_outcome = round(has_churned))
#'type' argument set to 'response' gets us the probability of churning
```

Visualizing:

```
plt_churn_vs_recency_base +
        geom_point(
                aes(y = most_likely_outcome),
                data = prediction_data,
                color = 'green')
output>
```



we can see clearly where the prediction points lie and the switch happens at probability of 0.5

Odds ratios
is the probability of something happening divided by the probability that it doesn't
odds ratio = probability / (1 -probability)
example
0.25 / (1-0.25) = 1/3

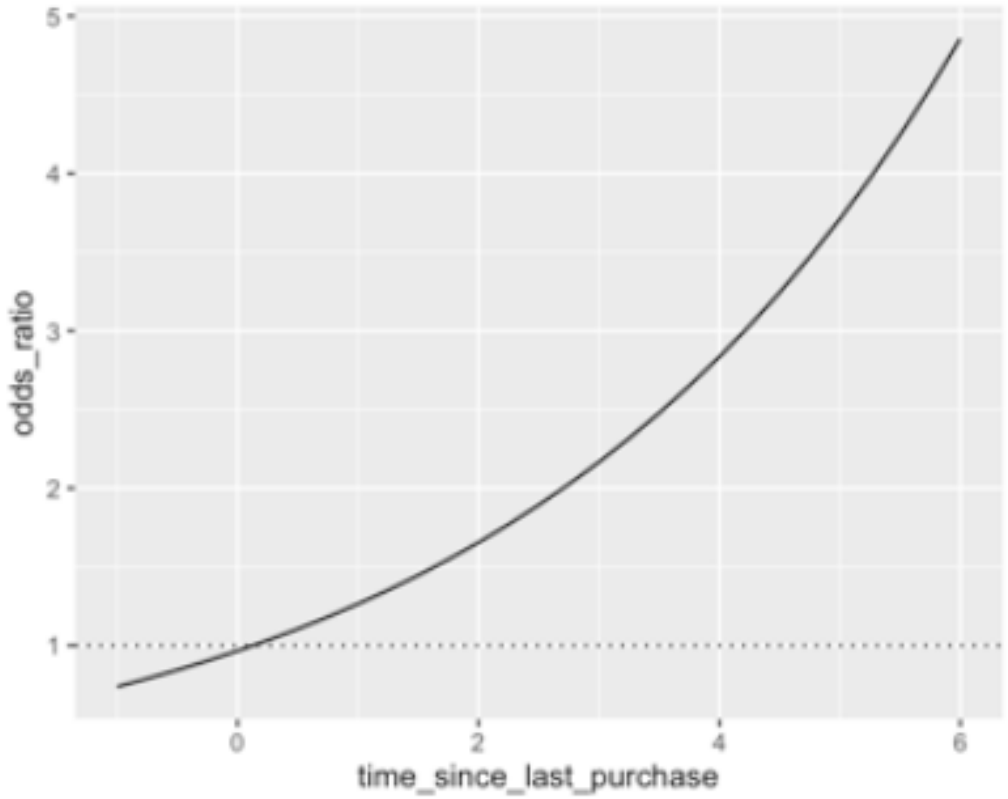
Caluculating with R

```
prediction_data <- explanatory_data %>%
        mutate(has_churned = predict(mdl_recency, explanatory_data, type =
'respones),
        most_likely_outcome = round(has_churned),
        odds_ratio = has_churned / (1 - has_churned))
```

#here we are dividing the predicted probability by 1 minus the predicted probability
Visualizing:

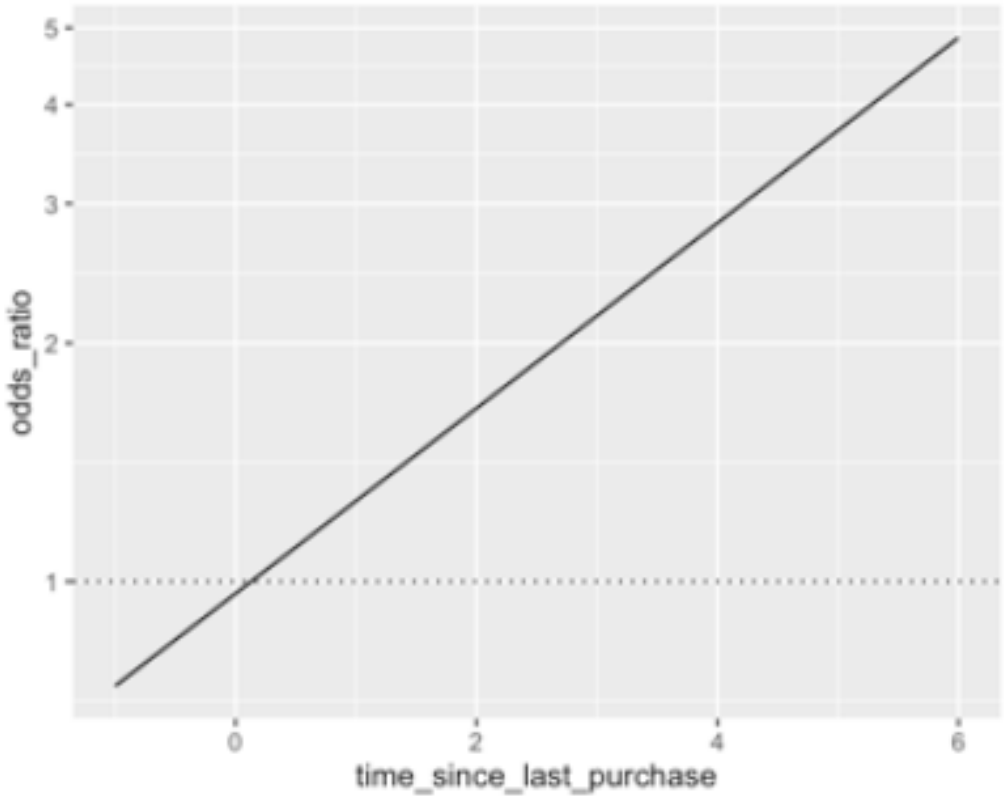
```
ggplot(
        prediction_data,
        aes(time_since_last_purchase, odds_ratio)) +
        geom_line() +
        geom_hline(yintercept = 1, linetype = 'dotted')
```

output>



here the odds ratio is 1 indicating that churning is just as likely as not churning
below 1 the chance of churning is less than the chance of not churning
above 1 the chance of churning is greater than the chance of not churning
near the right upper corner the chance is about 5 times more for the chance of churning than not churning

Visualization cont'd:
*nice property of logistic regression odds ratios is that on a log-scale, they change linearly with the explanatory variable
above code +
#created data
scale_y_log10()
output>



Calculating log odds ratio cont'd:

```
prediction_data <- explanatory_data %>%
      mutate(has_churned = predict(mdl_recency, explanatory_data, type =
'response'),
      most_likely_outcome = round(has_churned),
      odds_ratio = has_churned / (1 - has_churned),
      log_odds_ratio = log(odds_ratio),
      log_odds_ratio1 = predict(mdl_recency, explanatory_data))
```

#logarithm od odds ratios is another common way of describing logistic regression predictions

*such a common way that predict function by default will return the log odds ratio

these pieces of code are equivalent:

```
log_odds_ratio = log(odds_ratio),
log_odds_ratio1 = predict(mdl_recency, explanatory_data))
```

Comparing scales

Scale	Are values easy to interpret?	Are changes easy to interpret?	Is precise?
Probability	✔	✘	✔
Most likely outcome	✔✔	✔	✘
Odds ratio	✔	✘	✔
Log odds ratio	✘	✔	✔

most likely outcome is easiest to understand but lacks precision
odds ratio falters in non-linear relationships as it becomes challenging to reason about how changes in the explanatory variable will change the response
log odds is difficult to interpret for individual values, but the linear relationship that it creates with the explanatory variables makes it easy to reason about change

Example

```
# From previous step
prediction_data <- explanatory_data %>%
  mutate(
    has_churned = predict(mdl_churn_vs_relationship, explanatory_data, type =
"response"),
    odds_ratio = has_churned / (1 - has_churned)
  )

# Using prediction_data, plot odds_ratio vs. time_since_first_purchase
ggplot(prediction_data, aes(time_since_first_purchase, odds_ratio)) +
  # Make it a line plot
  geom_line() +
  # Add a dotted horizontal line at y = 1
  geom_hline(yintercept=1, linetype='dotted')
```

Example

```
# Update the data frame
prediction_data <- explanatory_data %>%
  mutate(
    has_churned = predict(mdl_churn_vs_relationship, explanatory_data, type =
"response"),
    odds_ratio = has_churned / (1 - has_churned),
    log_odds_ratio = log(odds_ratio)
  )

# Update the plot
ggplot(prediction_data, aes(time_since_first_purchase, odds_ratio)) +
```

```
  geom_line() +
  geom_hline(yintercept = 1, linetype = "dotted") +
  # Use a logarithmic y-scale
  scale_y_log10()
```

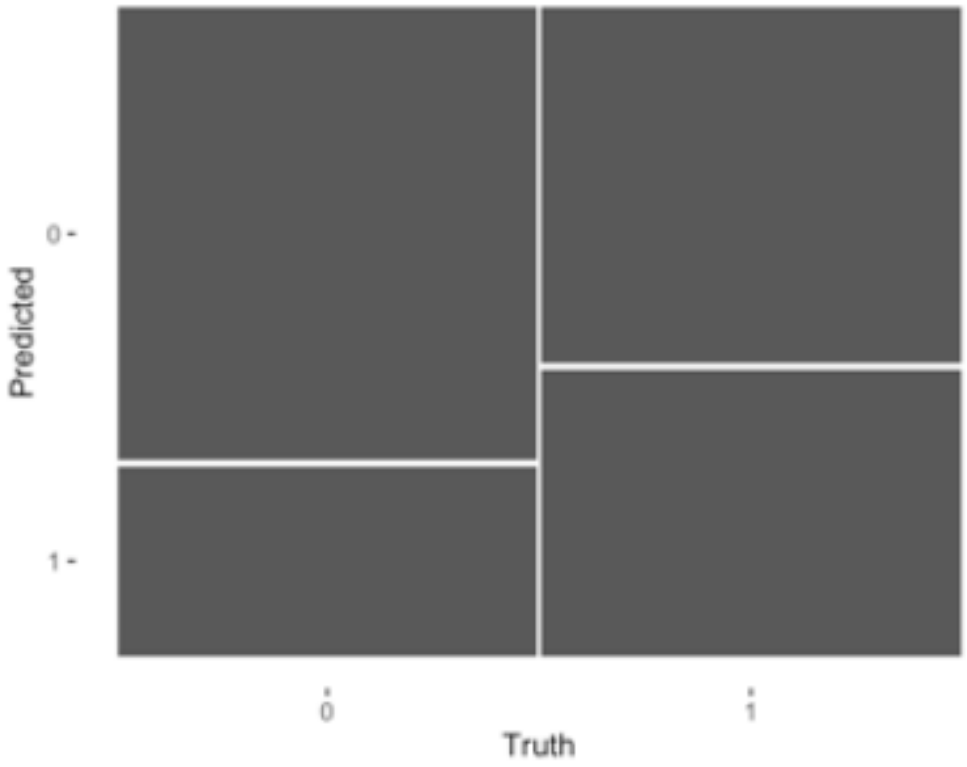
Quantifying logisitic regression fit
we assess this not with diagnositc plots but with confusion matrices

	actual false	actual true
predicted false	correct	false negative
predicted true	false positive	correct

confusion matrix one of the few R tasks done better in base code than in the tidyverse
example
madl_recency <- glm(has_churned ~ time_since_last_purchase, data = churn, family = 'binomial')
#get actual responses from has_churned column
actual_response <- churn$has_churned
#get predicted responses from the model
#'fitted' returns the predicted values of each observation in the dataset
#these fitted values are probabilities
#get the most likely outcome round the values to 0 or 1
predicted_response < round(fitted(mdl_recency))
#use 'table' to get the counts of each combination of values
outcomes <- table(predicted_response, actual_response)
output>

```
                  actual_response
predicted_response   0   1
                 0 141 111
                 1  59  89
```

Visualize:
library(yardstick)
confusion <- conf_mat(outcomes)
autoplot(confusion)
output>



Performance metrics
#'event_level' represents the responses which in this example are in the second column
summary(confusion, event_level = 'second')

Accuracy
the proportion of correct predictions
accuracy = TN + TP / TN + FN + FP + TP
in R:
summary(confusion) %>%
    slice(1)

Sensitivity
the proportion of true positives
sensitivity = TP / FN + TP
in R:
summary(confusion) %>%
    slice(3)
higher sensitivity is better

Specificity
the proportion of true negatives
specificity = TN / TN + FP
higher specificity is better
*however tradeoff where improving specificity will decrease sensitivity, or increasing sensitivity will decrease specificity