# Level Difficulty in Candy Crush Saga

**Project Description**

[**Candy Crush Saga**](https://king.com/game/candycrush) is a hit mobile game developed by King (part of Activision|Blizzard) that is played by millions of people all around the world.

In this Project, you will get to work with a real Candy Crush dataset and use this data to estimate level difficulty.

This Project assumes you can manipulate data frames using dplyr and make simple plots using ggplot2. You can learn these skills in [**Introduction to the Tidyverse**](https://www.datacamp.com/courses/introduction-to-the-tidyverse).

**Task 1: Instructions**

Load in the packages you're going to need for the project:

* readr
* dplyr
* ggplot2

Tip: Instead of loading in these packages separately you can load the tidyverse *meta-package* which will load in a bunch of useful packages, including readr, dplyr and ggplot2.

**Good to know**

This project assumes you have used the dplyr and ggplot2 packages and that you are familiar with the pipe operator (%>%). Before taking on this project we recommend that you have completed the following courses:

* [**Introduction to the Tidyverse**](https://www.datacamp.com/courses/introduction-to-the-tidyverse)
* [**Data Visualization with ggplot2 (Part 2)**](https://www.datacamp.com/courses/data-visualization-with-ggplot2-2)

RStudio has created some very helpful cheat sheets, including two that will be helpful for this project: [**Data Wrangling**](https://cdn.rawgit.com/rstudio/cheatsheets/85b5a5e2/data-transformation.pdf) and [**Data Visualization with ggplot2**](https://cdn.rawgit.com/rstudio/cheatsheets/85b5a5e2/data-visualization-2.1.pdf). We recommend that you keep them open in a separate tab to make it easy to refer to them.

Take Hint

**Task 2: Instructions**

Load in the dataset and display the first couple of rows.

* Load the csv file located at datasets/candy\_crush.csv using read\_csv and assign it to the variable data.
* Display the head of the data.

Make sure to use read\_csv (with an *underscore*) to read in the data. The read.csv function, which is built into R, has a number of problems which the new read\_csv function avoids.

We define the **granularity** of a dataset as the *lowest level of detail* of the observations. Here that means the combination of level, player\_id, and dt. The rest of the columns are the *facts* that happened at that *level of detail*. That is, what happened for a given player, at a given day, at a given level. Sometimes we refer to the two types of columns as **id** columns (level, player\_id, dt) and **variable** columns (num\_attempts, num\_success).

Take Hint

**Task 3: Instructions**

Count how many players are in the dataset and how many days it spans.

* Count the number of unique players included in the data.
* Compute the period for which we have data.

Remember that there might be several rows for each player, but here you should calculate the number of unique players.

One of the nice features of the function read\_csv from the readr package is that it uses a heuristic to figure out the types of your columns. That's why the dt column has already been parsed as Date. This is useful because there are many functions that work with Dates, for example range().

**HINT**

If you have got a vector of, say, strings such as data$player\_id, you can use the unique() function to return the set of distinct values and then take length() of that vector to get the number of unique values.

x <- c('a','a','b')

# subset of unique values, prints 'a' 'b'

unique(x)

# Number of unique values, prints 2

length(unique(x))

You can find the range of days using the range() function, or the min() and max() functions.

**Task 4: Instructions**

Calculate the probability of winning a level in a single attempt for each level.

* Group the dataset by level.
* Compute the total number of attempts and wins for each level by using the summarise function.
* Compute the probability to win as the number of wins divided by the number of attempts. Assign the result to a new column called p\_win using mutate.
* The resulting data frame should be assigned to difficulty and printed out.

Modeling the probability of winning a level (*pwin*) as a Bernoulli process is, of course, a simplification. In reality, this probability will also depend on the skill of each player and the player could learn from past attempts and play better every time. But to include those assumptions is a refinement that we will leave for another occasion.

**HINT**

Here is some code to get you started:

difficulty <- data %>%

group\_by(....) %>%

summarise(attempts = sum(....), wins = sum(....)) %>%

mutate(p\_win = .... / ....)

**Task 5: Instructions**

Plot a line graph with the difficulty for each level.

* Use ggplot to plot a line graph with p\_win on the Y-axis and level on the X-axis.
* Set the breaks of the X-axis to show a tick mark for every level.
* Set the Y-axis labels to a nicely formatted percentage using the scales package.

For how to set the tick marks of a continuous scale see [**the ggplot2 documentation for scale\_x\_continuous**](http://ggplot2.tidyverse.org/reference/scale_continuous.html) (the examples could also be useful to look at).

If you don't know how to percent format the Y-axis check out [**this stack overflow question**](https://stackoverflow.com/q/27433798).

**HINT**

Line graphs can be generated by ggplot with geom\_line():

ggplot(aes(x = level, y = p\_win)) +

geom\_line() +

You can modify the default x axis breaks by setting them explicitly:

scale\_x\_continuous(breaks = 1:15)

The scales package provides a set of functions to format axes. For example, you can change the Y-axis to a percent format:

scale\_y\_continuous(label = scales::percent)

**Task 6: Instructions**

Add points to the plot and a horizontal dashed line at the 10% value.

* Copy and paste the solution from task 5.
* In addition to the lines *between* the datapoints, add a point *at* each datapoint.
* Add a horizontal *dashed* line to the plot at Y-axis value 10%.

Check out [**the documentation for geom\_point**](http://ggplot2.tidyverse.org/reference/geom_point.html) for how to add points to a plot.

Check out [**the documentation for geom\_hline**](http://ggplot2.tidyverse.org/reference/geom_abline.html) for how to annotate a plot with horizontal lines.

**HINT**

You can draw horizontal and vertical lines in ggplot with geom\_vline() and geom\_hline(). To add a dashed horizontal line at 50% you could use:

geom\_hline(yintercept = 0.5, linetype = 'dashed')

The threshold for *pwin* is 10% but remember that the y values are defined over the interval [0, 1] so the threshold should be written as 0.1 .

**Task 7: Instructions**

Compute the standard error of the difficulty for each level using the given formula.

* Add the column error to difficulty which should contain the standard error of *pwin* using the formula error = sqrt(p\_win \* (1 - p\_win) / attempts).

There are many ways we could calculate the uncertainty around the difficulty estimates. We could, for example, have used bootstrap estimation or Bayesian modeling. However, calculating standard errors is a very quick way of getting uncertainty estimates that in many cases are good enough.

Take Hint

**Task 8: Instructions**

Add error bars to the difficulty profile plot.

* Copy and paste the ggplot code used to generate the plot in task 6.
* Use geom\_errorbar to add error bars that range from p\_win - error to p\_win + error for each level.

ggplot2 has many nice ways of showing vertical intervals: lines, crossbars and error bars. Here you will use error bars. Check out [**the documentation for geom\_errorbar**](http://ggplot2.tidyverse.org/reference/geom_linerange.html) for how to do this.

**HINT**

You can add error bars in ggplot2 by using the geom\_errorbar function. The required arguments are ymin and ymax which set the top and bottom positions of the error bar. Here is some code to get you started:

geom\_errorbar(aes(ymin = ...., ymax = ....))

**Task 9: Instructions**

Calculate how likely is it that a player will complete all the levels in the first attempt.

* Calculate the probability of the average player completing every level in the first attempt. Assign the result to p.

The probability of two independent events happening is simply the product of the individual probabilities. So the probability of winning both level 1 *and* level 2 on the first attempt would be

p\_win[1] \* p\_win[2]

To extend this to all the 15 levels in the episode you can use the prod function which multiplies all the numbers in a vector together (that is, takes the *product* of all the vector elements).

**HINT**

In R one can compute the product of all elements of a vector by using the prod function:

prod(difficulty$p\_win)

**Task 10: Instructions**

Should our level designer worry that a lot of players will complete the episode in one attempt?

* Set should\_the\_designer\_worry to TRUE or FALSE to indicate your answer.

**If you want to know more**

Now that you've analyzed some Candy Crush data, maybe you want to try out the actual game? Perhaps you can think of other level metrics you would want to calculate?

**[](https://king.com/game/candycrush)**