Workout 03: R Package `binomial`

Stat 133, Spring 2019

The purpose of this assignment is to create an R package that implements functions for calculating probabilities of a Binomial random variable, and related calculations such as the probability distribution, the expected value, variance, etc.

Here's a list of resources that may help you complete this assignment:

- **Pack YouR Code**: www.gastonsanchez.com/packyourcode
- **Example package `"cointoss"`**: github.com/gastonstat/cointoss
- **Package development cheat-sheet**: packages-cheatsheet.pdf
- **R Packages**: r-pkgs.had.co.nz

Binomial Distribution

The Binomial distribution is perhaps the most famous probability distribution among discrete random variables. This is the theoretical probability model that we use when calculating probabilities about the number of successes in a fixed number of random trials performed under identical conditions (assuming a constant probability of success on each trial).

A classic example of a binomial random variable X involves the number of Heads (or Tails) that you get when tossing a coin $n \geq 0$ times. Say you are interested in finding the probability of getting three heads in four tosses of a fair coin: P(X = 3 heads in 4 tosses). To find the answer, we use the formula of the binomial probability:

$$Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

where:

- n is the number of (fixed) trials ($n \geq 0$)
- p is the probability of success on each trial ($0 \leq p \leq 1$)
- $1 - p$ is the probability of failure on each trial
- k is a variable that represents the number of successes out of n trials ($0 \leq k \leq n$)
- the first term in parenthesis is NOT a fraction, it is the number of combinations in which k success can occur in n trials

So, what is the probability of three heads in four tosses? Assuming that we flip a fair coin (50% chance of heads), P(X = 3) is:

$$Pr(X = 3) = \binom{4}{3} 0.5^3 (1 - 0.5)^{4-3} = 0.25$$

If, instead of a fair coin, you have a loaded coin with a 75% chance of heads, then P(X = 3) is:

$$Pr(X = 3) = \binom{4}{3} 0.75^3 (1 - 0.75)^{4-3} \approx 0.4218$$

Mean and Variance

When X is a random variable that has a binomial distribution with n trials and probability of success p, we write $X \sim Bin(n, p)$. As with most random variables, it is useful to know about the various summary measures of X.

The expected value or mean of a binomial distribution is: np. This is the expected number of successes in n trials.

The variance is given by: $np(1 - p)$. Consequently, the standard deviation is simply the square root of the variance, that is: $\sqrt{np(1 - p)}$

Mode

For $0 < p < 1$, the most likely number of success in n independent trials with probability p of success on each trial is m, the greater integer less than or equal to $np + p$:

$$m = \text{int}(np + p)$$

where int denotes the integer part function. If $np + p$ is an integer, as in the case $p = 0.5$ and n odd, then there are two most likely numbers, m and $m - 1$. Otherwise, there is a unique mode.

Skewness and Kurtosis

Other two additional measures are skewness and kurtosis. Skewness is a measure of the asymmetry of the probability distribution of a random variable about its mean. The skewness value can be positive or negative, or undefined. The skewness of a binomial random variable can be calculated as:

$$\text{skewness} = \frac{1 - 2p}{\sqrt{np(1 - p)}}$$

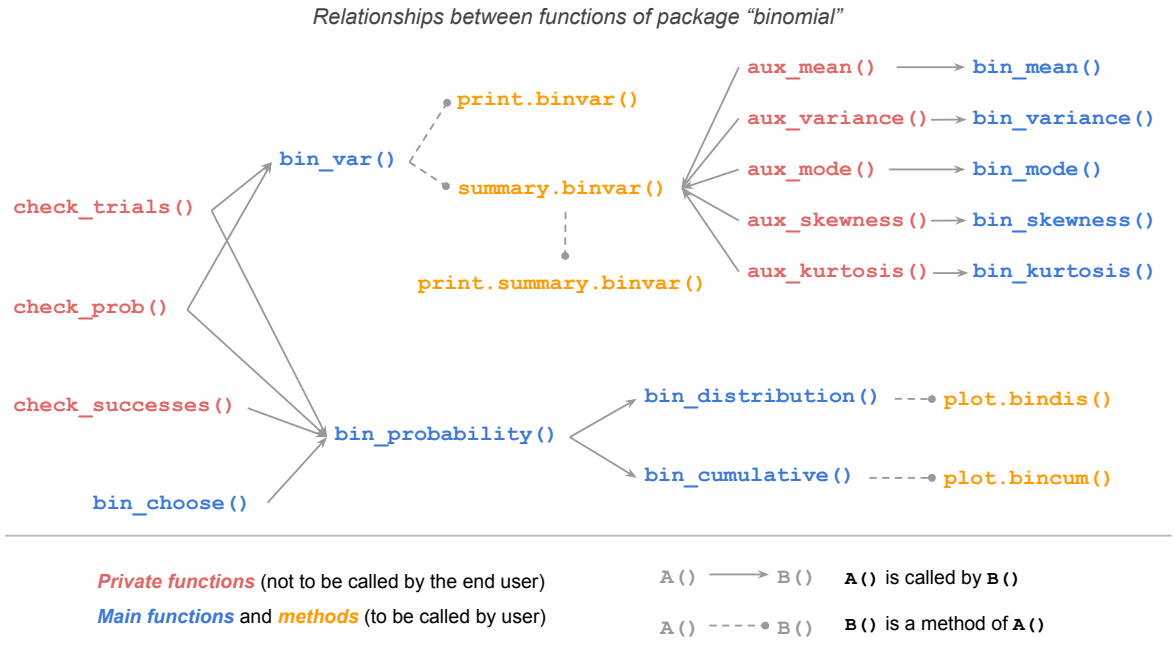
The Kurtosis (from greek *kurtos*, meaning "curved, arching") is a measure of the "tailedness" of the probability distribution of a random variable. In a similar way to the concept of skewness, kurtosis is a descriptor of the shape of a probability distribution. For a binomial random variable, its kurtosis can be obtained as:

$$\text{kurtosis} = \frac{1 - 6p(1-p)}{np(1-p)}$$

---

1) R Functions

In order to create the package `"binomial"`, the first stage involves writing all the functions depicted in the following diagram (more explanations below):



As you can tell from the diagram above, there are three types of functions:

- **private**: these are auxiliary functions not intended to be called by the user (in red color).
- **main**: these are the main functions that the user is expected to invoke (in blue color).
- **methods**: these are also public functions (in yellow color) that support classes of objects generated by some of the *main* functions.

More details about these functions is given in the following section.

Some Restrictions

R comes with built-in functions that allow you to compute binomial probabilities, as well as factorial, and combinations. However, we want to make things a bit challenging and interesting. Therefore, you are **NOT allowed** to use base R functions such as:

- `choose()`
- `dbinom()`
- `pbinom()`
- `qbinom()`
- `rbinom()`

Checklist

Here's a list for each of the mandatory functions in your `"binomial"` package.

- private checker functions:
  - `check_prob()`
  - `check_trials()`
  - `check_success()`
- private auxiliary functions for summary measures:
  - `aux_mean()`
  - `aux_variance()`
  - `aux_mode()`
  - `aux_skewness()`
  - `aux_kurtosis()`
- main functions; and methods:
  - `bin_choose()`
  - `bin_probability()`
  - `bin_distribution()`; and `plot.bindis()`
  - `bin_cumulative()`; and `plot.bincum()`
  - `bin_var()`; and `print.binvar()`, `summary.binvar()`, `print.summary.binvar()`
  - `bin_mean()`, `bin_variance()`, `bin_mode()`, `bin_skewness()`, `bin_kurtosis()`

1.1) Private Checker Functions

Function `check_prob()`

Write a *private* auxiliary function `check_prob()` to test if an input `prob` is a valid probability value (i.e. $0 \leq p \leq 1$).

- `check_prob()` takes one input: `prob`.
- If `prob` is valid, then `check_prob()` should return `TRUE`.

- If `prob` is invalid, then `check_prob()` should `stop()` execution with an error—e.g. something like `'invalid prob value'`, or `'p has to be a number betwen 0 and 1'`.
- Since this is a *private* function, use simple R comments to write its documentation (don't use Roxygen comments).

Function `check_trials()`

Write a *private* auxiliary function `check_trials()` to test if an input `trials` is a valid value for number of trials (i.e. n is a non-negative integer).

- `check_trials()` takes one input: `trials`.
- If `trials` is valid, then `check_trials()` should return `TRUE`.
- If `trials` is invalid, then `check_trials()` should `stop()` execution with an error—e.g. something like `'invalid trials value'`.
- Since this is a *private* function, use simple R comments to write its documentation (don't use Roxygen comments).

Function `check_success()`

Write a *private* auxiliary function `check_success()` to test if an input `success` is a valid value for number of successes (i.e. $0 \leq k \leq n$).

- `check_success()` takes two inputs: `success` and `trials`.
- `success` should be a vector of non-negative integer(s) less than or equal to `trials`.
- Notice that `success` can be of length greater than 1 (i.e. multiple successes).
- If `success` is valid, then `check_success()` should return `TRUE`.
- If `success` is invalid, then `check_success()` should `stop()` execution with an error—e.g. something like `'invalid success value'` or if $k > n$ then `'success cannot be greater than trials'`.
- Since this is a *private* function, use simple R comments to write its documentation (don't use Roxygen comments).

1.2) Private Auxiliary Functions

Write the following *private* auxiliary functions:

- `aux_mean()`
- `aux_variance()`
- `aux_mode()`
- `aux_skewness()`
- `aux_kurtosis()`

All these functions take two arguments: `trials` and `prob`. And return the corresponding value from the computed summary measure (formulas provided in the introduction of this

document).

- Because these are *private* functions, there's no need to use the *checker* functions `check_trials()` and `check_prob()` inside the auxiliary functions.
- Each of these functions will be invoked by their corresponding *main* functions e.g. `bin_mean()`, `bin_variance()`, etc.
- Since these are *private* functions, use simple R comments to write the documentation (don't use Roxygen comments).

Here's how you should be able to invoke these functions:

```
aux_mean(10, 0.3)
```

```
## [1] 3
```

```
aux_variance(10, 0.3)
```

```
## [1] 2.1
```

```
aux_mode(10, 0.3)
```

```
## [1] 3
```

```
aux_skewness(10, 0.3)
```

```
## [1] 0.2760262
```

```
aux_kurtosis(10, 0.3)
```

```
## [1] -0.1238095
```

1.3) Function `bin_choose()`

Use `factorial()` to write a *main* function `bin_choose()` that calculates the number of combinations in which k successes can occur in n trials.

Recall that the number of combinations "n choose k" is given by:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

For instance, the number of combinations in which $k = 2$ successes can occur in $n = 5$ trials is:

$$\binom{n=5}{k=2} = \frac{5!}{2!(5-2)!} = 10$$

- Your function should have arguments `n` and `k`.
- Try to write `bin_choose()` with vectorized code (i.e. avoid loops).
- If $k > n$, then `bin_choose()` should `stop()` execution returning an error message like `"k cannot be greater than n"`
- Keep in mind that you are NOT allowed to use the R function `choose()`.
- Document your function with roxygen comments: (e.g. `@title`, `@description`, `@param`, `@return`, `@export`, `@examples`).

Here's how you should be able to invoke `bin_choose()`

```
bin_choose(n = 5, k = 2)
bin_choose(5, 0)
bin_choose(5, 1:3)
```

1.4) Function `bin_probability()`

Use `bin_choose()` to create a *main* function `bin_probability()`.

- `bin_probability()` should take three arguments: `success`, `trials`, and `prob`.
- Use `check_trials()` to check that `trials` is valid
- Use `check_prob()` to check that `prob` is valid
- Use `check_success()` to check that `success` is valid
- If any of `trials`, `prob` or `success` is invalid, then `bin_probability()` should raise an error—triggered by `stop()`—e.g. something like `'invalid trials value'` or `'invalid success value'`.
- Document your function with roxygen comments: (e.g. `@title`, `@description`, `@param`, `@return`, `@export`, `@examples`).

Here's how you should be able to invoke `bin_probability()`:

```
# probability of getting 2 successes in 5 trials
# (assuming prob of success = 0.5)
bin_probability(success = 2, trials = 5, prob = 0.5)
```

```
## [1] 0.3125
```

```
# probabilities of getting 2 or less successes in 5 trials
# (assuming prob of success = 0.5)
bin_probability(success = 0:2, trials = 5, prob = 0.5)
```

```
## [1] 0.03125 0.15625 0.31250
```

```
# 55 heads in 100 tosses of a loaded coin with 45% chance of heads
bin_probability(success = 55, trials = 100, prob = 0.45)
```

```
## [1] 0.01075277
```

1.5) Function `bin_distribution()`

Use `bin_probability()` to create a *main* function `bin_distribution()`.

- Your function should have two arguments `trials`, and `prob`.
- The returned output should be a `data.frame` with two classes: `c("bindis", "data.frame")`
- In other words, the primary class is `"bindis"` indicating that this is an object of class *binomial distribution*. Additionally, to keep this object as a data frame, we still need to include a class `"data.frame"`.
- This function should return a data frame with the probability distribution: *sucesses* in the first column, *probability* in the second column.
- Document your function with roxygen comments: (e.g. `@title`, `@description`, `@param`, `@return`, `@export`, `@examples`).

Here's how you should be able to invoke `bin_distribution()`

```
# binomial probability distribution
bin_distribution(trials = 5, prob = 0.5)
```

```
##   success probability
## 1       0     0.03125
## 2       1     0.15625
## 3       2     0.31250
## 4       3     0.31250
## 5       4     0.15625
## 6       5     0.03125
```

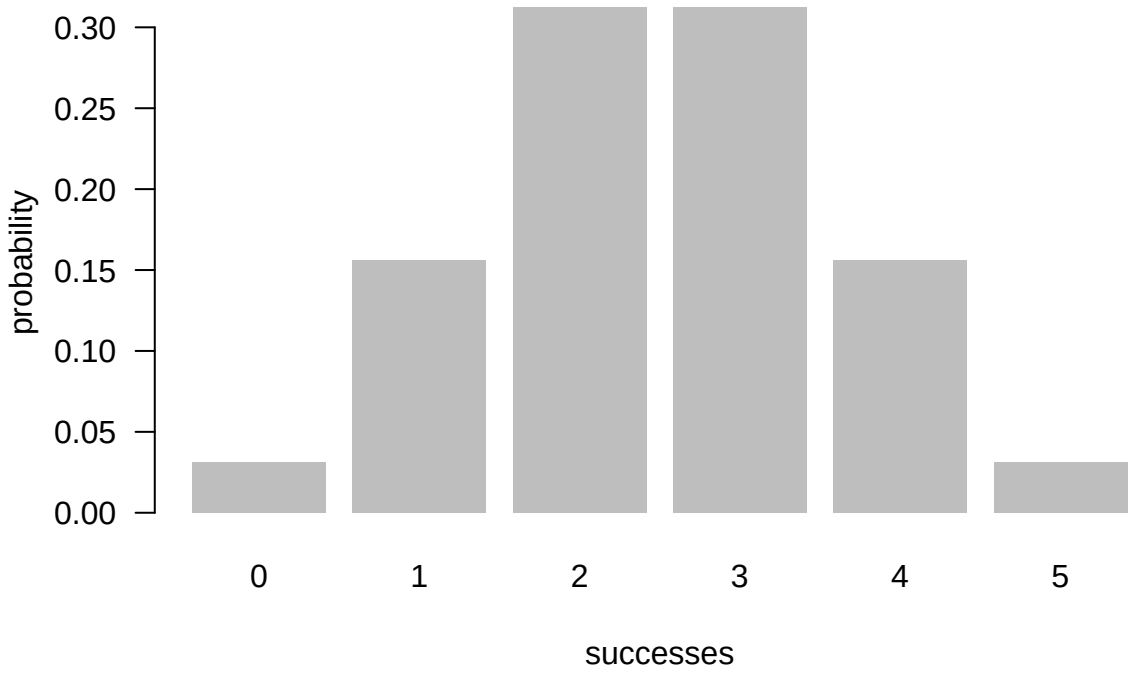
Function `plot.bindis()`

Write a plotting method (i.e. a function) `plot.bindis()` that graphs a barplot to display the probability histogram of a binomial distribution object `"bindis"`.

- Since this is a method, you just need to document your function with the roxygen comment: `@export`.

Here's an example of how to invoke the plot method. You are welcome to plot a graph with more visual appeal than the image depicted below.

```
# plotting binomial probability distribution
dis1 <- bin_distribution(trials = 5, prob = 0.5)
plot(dis1)
```



1.6) Function `bin_cumulative()`

Use `bin_cumulative()` to create a *main* function `bin_cumulative()`.

- Your function should have two arguments `trials`, and `prob`.
- The returned output should be a `data.frame` with two classes: `c("bincum", "data.frame")`
- In other words, the primary class is `"bincum"` indicating that this is an object of class *binomial cumulative distribution*. Additionally, to keep this object as a data frame, we still need to include a class `"data.frame"`.
- This function should return a data frame with both the probability distribution and the cumulative probabilities: *sucesses* in the first column, *probability* in the second column, and *cumulative* in the third column.
- Document your function with roxygen comments: (e.g. `@title`, `@description`, `@param`, `@return`, `@export`, `@examples`).

Here's how you should be able to invoke `bin_cumulative()`

```
# binomial cumulative distribution
bin_cumulative(trials = 5, prob = 0.5)
```

```
##   success probability cumulative
## 1       0     0.03125    0.03125
## 2       1     0.15625    0.18750
## 3       2     0.31250    0.50000
```

```
## 4       3     0.31250    0.81250
## 5       4     0.15625    0.96875
## 6       5     0.03125    1.00000
```

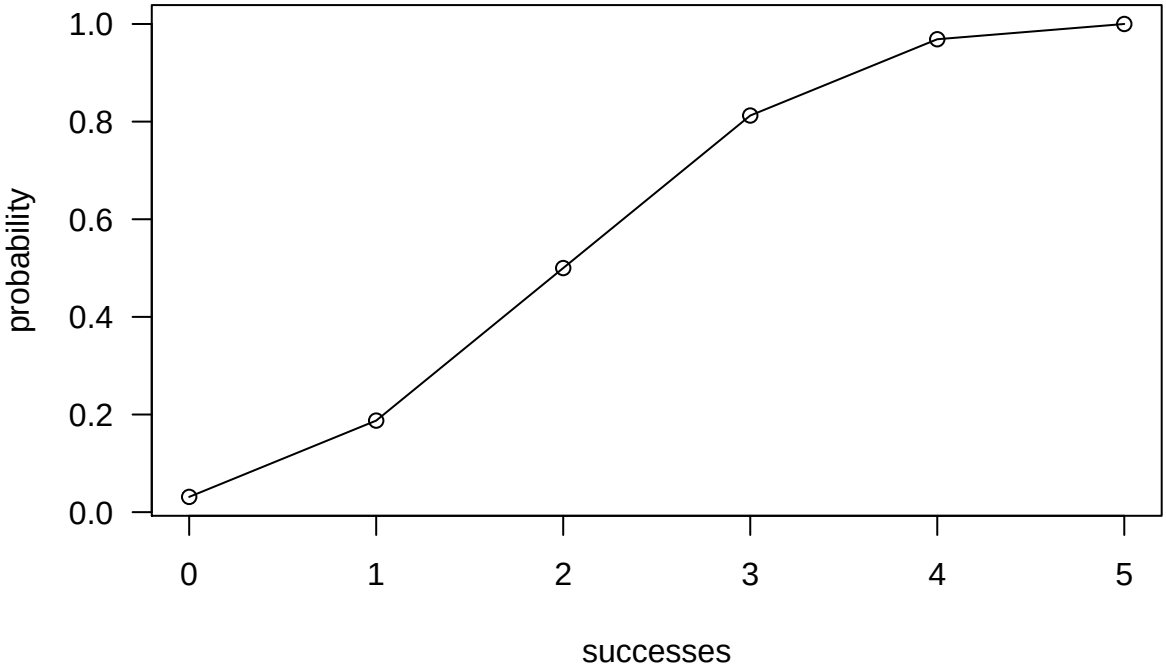
Function `plot.bincum()`

Write a plotting method (i.e. a function) `plot.bincum()` that graphs the cumulative distribution in ab object `"bincum"`.

- Since this is a method, you just need to document your function with the roxygen comment: `@export`.

Here's an example of how to invoke the plot method. You are welcome to plot a graph with more visual appeal than the image depicted below.

```
# plotting binomial cumulative distribution
dis2 <- bin_cumulative(trials = 5, prob = 0.5)
plot(dis2)
```



1.7) Function `bin_variable()`

Another function to include in your `"binomial"` package is `bin_variable()`.

- This is a *main* function that takes two arguments: `trials` and `prob`
- This function should return an object of class `"binvar"`, that is, a *binomial random variable* object.

- This function should invoke `check_trials()` and `check_prob()`
- The returned object should be a list with named elements:
  - `trials`: number of trials
  - `prob`: probability of success
- Document your function with roxygen comments: (e.g. `@title`, `@description`, `@param`, `@return`, `@export`, `@examples`).

Because `bin_variable()` will return an object of class `"binvar"`, you will also need to implement methods: `print.binvar()`, `summary.binvar()`, and `print.summary.binvar()`.

Method `print.binvar()`

Create a method function `print.binvar()` to be able to nicely print the content of an object `"binvar"`. See example below.

- Since this is a method, you just need to document your function with the roxygen comment: `@export`.

Here's an example of how you should be able to invoke `bin_variable()`, and its printed output:

```
bin1 <- bin_variable(trials = 10, p = 0.3)
bin1
```

```
"Binomial variable"

Paramaters
- number of trials: 10
- prob of success : 0.3
```

Methods `summary.binvar()` and `print.summary.binvar()`

To get a full summary description of an object `"binvar"`, you need to create a function `summary.binvar()`.

- This function takes an object of class `"binvar"`
- The returned output is a list of class `"summary.binvar"` containing named elements:
  - `trials`: number of trials
  - `prob`: probability of success
  - `mean`: mean or expected value
  - `variance`: variance
  - `mode`: mode
  - `skewness`: skewness
  - `kurtosis`: kurtosis

- Use the *private* auxiliary functions: `aux_mean()`, `aux_variance()`, etc. to compute the summary measures.
- Since this is a method, you just need to document your function with the roxygen comment: `@export`.

In order to nicely print the contents of an object `"summary.binvar"`, you also need to write a print method: `print.summary.binvar()`. See example below:

```
bin1 <- bin_variable(trials = 10, p = 0.3)
binsum1 <- summary(bin1)
binsum1
```

```
"Summary Binomial"

Paramaters
- number of trials: 10
- prob of success : 0.3

Measures
- mean    : 3
- variance: 2.1
- mode    : 3
- skewness: 0.2760262
- kurtosis: -0.1238095
```

1.8) Functions of measures

Finally, your `"binomial"` package should also contain *main* functions for each of the summary measures: e.g. `bin_mean()`, `bin_variance()`, etc.

- These are *main* functions that take two arguments: `trials` and `prob`
- Use `check_trials()` to check that `trials` is valid
- Use `check_prob()` to check that `prob` is valid
- Invoke your *auxiliary* functions to do the corresponding calculation. For instance: `aux_mean()` gets called by `bin_mean()`.

Here's how you should be able to invoke these functions:

```
bin_mean(10, 0.3)
```

```
## [1] 3
```

```
bin_variance(10, 0.3)
```

```
## [1] 2.1
```

```
bin_mode(10, 0.3)
```

```
## [1] 3
```

```
bin_skewness(10, 0.3)
```

```
## [1] 0.2760262
```

```
bin_kurtosis(10, 0.3)
```

```
## [1] -0.1238095
```

---

2) Tests

Your `"binomial"` package should include tests—via the package `"testthat"`—for the following functions, and their *contexts*:

- Context for checkers:
  - `check_prob()`
  - `check_trials()`
  - `check_success()`
- Context for summary measures:
  - `aux_mean()`
  - `aux_variance()`
  - `aux_mode()`
  - `aux_skewness()`
  - `aux_kurtosis()`
- Context for binomial:
  - `bin_choose()`
  - `bin_probability()`
  - `bin_distribution()`
  - `bin_cumulative()`

You will need to come up with at least three *expectations* for each of the above functions. For example, say you are testing `check_prob()`, you may want to use expectations to confirm that:

- `prob` is a number between 0 and 1
- `prob` is of length 1
- getting an error if `prob` is invalid

Recall that, when writing test for functions of an R package, you need to create a subdirectory `tests/` containing a subdirectory `testthat/` which will contain the R scripts for each *context*.

3) Package Creation

Carefully check the example package `"cointoss"` to get some hints and inspiration. We expect that you write your own code, with your consistent style (avoid the temptation of copy/plagiarism):

https://github.com/gastonstat/cointoss

Vignette

Your package should also include an introductory vignette that shows the user how to utilize the various functionalities of the `"binomial"` package.

Package Structure

After completion, your package `"binomial"` should have the following filestructure:

```
binomial/
   .Rbuildignore
   binomial.Rproj
   devtools-flow.R
   DESCRIPTION
   NAMESPACE
   README.md
   R/
   man/
   tests/
   vignettes/
```

Submission

- Create a folder (i.e. subdirectory) `binomial/` in your github classroom repository. This will be the folder of your package.
- The due date is May-01 (before midnight). You will have to show your package to your GSI during lab, either May-02 or May-03. We will only grade committed work pushed to your classroom repo before the deadline.