Basic Statistics and Simulation

Owen Ward

November 15, 2018

Introduction

Why do we need statistics?

- Raw data is hard to make sense of. Simple statistics are used to give easy to understand summaries of the data.
- Data can take lots of forms and so different methods needed to do this for different data.

Data Types

- One of the simplest things that needs to be done is understand the different types of data.
- Can be basically be broken down into numerical and categorical.

Questions?

- What questions to we want to answer with descriptive statistics?
- Generally, want to know average values of the data we have, and how they compare to the average?
- How spread out the values are from this average?
- Measure of average values are the mean and the median.
- Are two distinct variables related? For example, is there a relationship between height and weight?

Mean and Median and Mode

- These two definitions have subtle differences.
- Mean is the average of the data.
- Median is the number such that half the data is above the number and half below it.
- Mode the middle number in the data.

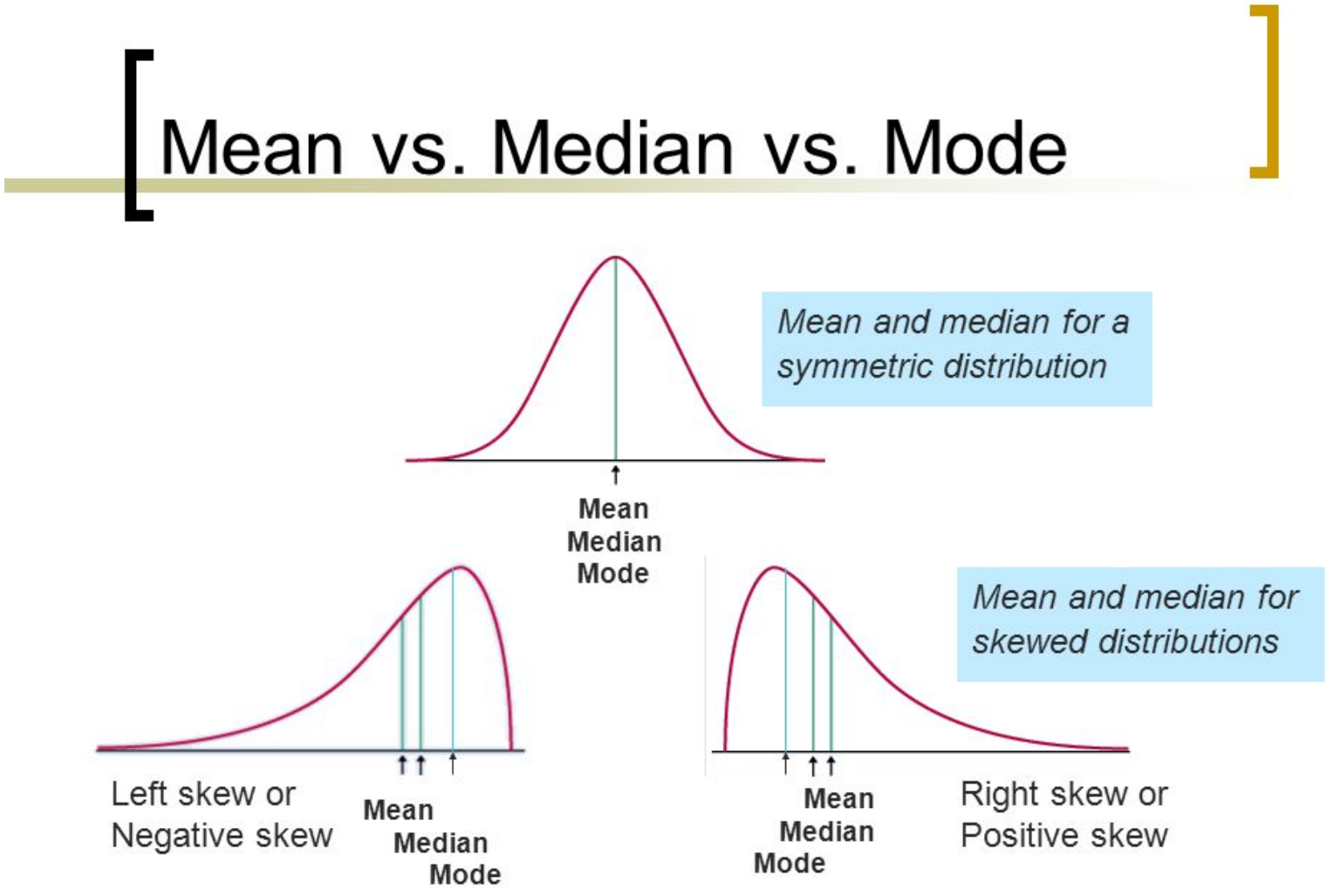


Figure 1: An example of the difference

How are these statistics related?

- The mean and the median are closely related. But choosing one over the other can be misleading.
- For example, when studying income, reporting the mean will give a very different result to reporting the median.

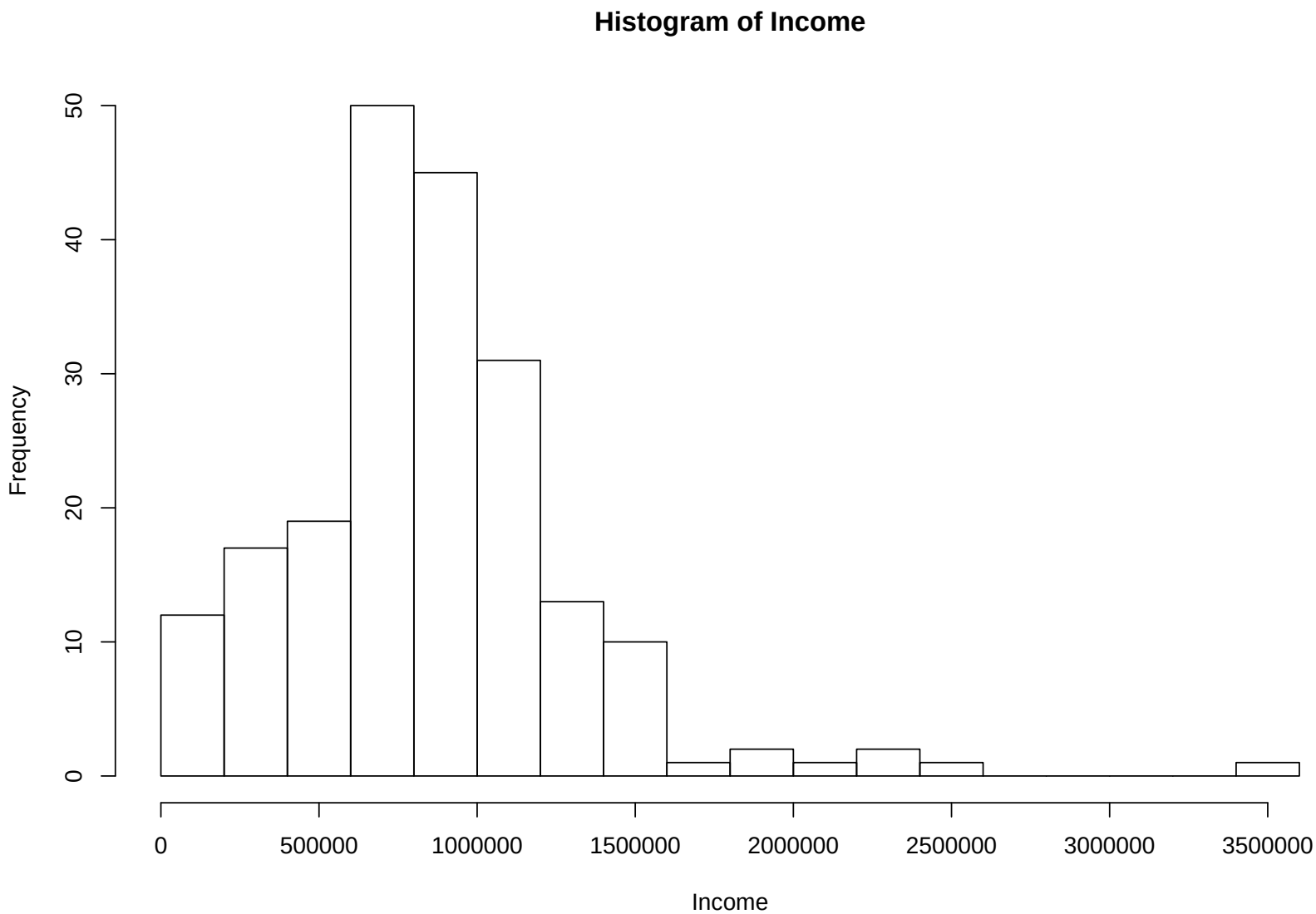
```r
library(SemiPar)
data("age.income")
head(age.income)
```

```
##   age log.income
## 1  21    11.1563
## 2  22    12.8131
## 3  22    13.0960
## 4  22    11.6952
## 5  22    11.5327
## 6  22    12.7657
```

```r
Income = exp(age.income$log.income)
# this income given on the log scale
summary(Income)
```

```
##     Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    70003  600009  810981  849773 1060022 3464920
```

```
hist(Income,breaks = 15)
```



What does this mean?

- In a sense, you can "lie" with statistics, or even, hide the truth in data.
- https://opendata.cityofnewyork.us/ gives data about NYC. If interested in a project or exploring what you could find out this contains lots of data about NYC.

Simulation

Why simulation is useful

- Simulation is an extremely powerful method which can be used to help solve lots of difficult problems and also to give insights which would not be available otherwise.
- Statistics is the study of randomness and therefore it seems reasonable to incorporate randomness into solving problems in statistics and data science.

Today

- Today we will look at examples of using simulation and and some of the properties which help make simulation based methods work.

When to use simulation

- Sometimes questions of interest are simply to difficult to answer exactly. However, using simulation we can often (quite easily) get approximate answers which are close to the true answer.
- These can be mathematical problems (high dimensional integrals) or more straightforward problems such as

  Suppose an elevator cable will break if there are more than 1600 pounds in the elevator. If 8 people get in, what is the probability the cable will snap?

- Simulation can also be used to check and validate models, and can demonstrate problems with your methods or data.
- Later, we will also see how adding randomness into models can lead to improved performance.

More robust predictions

One simple example where simulation can be used is in making better predictions. For example, FiveThirtyEight, which has provided some of the best U.S election predictions in the past decade uses simulation to give their final predictions (and add the required uncertainty).

- See for example https://projects.fivethirtyeight.com/2018-nba-predictions/
- Simulate several thousand games to give win projections.

Generating random numbers

How?

- To do simulations we need to create randomness and to do this random numbers are needed.
- How can we generate random numbers? What if thousands of numbers are needed?
- Computers can't really generate truly random numbers and so computer generated random numbers are "pseudo-random", numbers that look random but actually aren't.

In R

- R most easily generates random numbers in the interval $[0, 1]$. To generate from a different range it does this and then transforms them as appropriate.

```
runif(1) # generates 1 random number on [0,1]
```

```
## [1] 0.07469617
```

```
runif(3, min=-5,max = 5)
```

```
## [1] -2.253179 -1.080094  2.060659
```

```r
runif(5, min =0, max=2)
```

```
## [1] 0.1112149 1.8238522 1.5443645 0.1176106 0.7340822
```

```r
# which is basically the same method as
2*runif(5,min=0,max = 1)
```

```
## [1] 1.1409463 1.9795601 1.9136876 0.2603921 1.8296372
```

More generally

- Similarly we can generate from many other statistical distributions

```
rnorm(n=3,mean = 1, sd = 2)
```

```
## [1] 5.073980 1.444847 2.392173
```

```
rpois(n = 3, lambda = 2)
```

```
## [1] 2 2 1
```

A brief history of Simulation

- In the past, people have used different methods to generate random numbers. Examples include dice, coin flips, and the digits of π.
- Simulating random things is effectively impossible on a computer.
- Generally uses a method related to number theory called the Mersenne Twister.
- This uses complicated number theory methods to produce numbers that "look" random and have properties of random numbers.
- One way to get "truly" random numbers is to look at recordings of atmospheric noise. No idea how it is generated.

The elevator problem

Suppose an elevator cable will break if there are more than 1600 pounds in the elevator. If 8 people get in, what is the probability the cable will snap?

- Well known that weights are almost normally distributed. Lets say approximately has mean 180 pounds with standard deviation 20 pounds.
- So then we just want a simulation where, many times, we pick 8 random people and see how much they weigh.

```
weights = rnorm(n=8, mean = 180, sd = 20)
sum(weights)
```

```
## [1] 1539.395
```

```r
greater = c(rep(0,1000))
for(i in 1:1000){
  weights = rnorm(n=8, mean = 180, sd = 20)
  if(sum(weights) > 1600){
    greater[i] = 1
  }
}
sum(greater)/1000
```

```
## [1] 0.003
```

For such small probabilities might need to do more simulations to get an accurate answer.

```
greater = c(rep(0,10000))
for(i in 1:10000){
  weights = rnorm(n=8, mean = 180, sd = 20)
  if(sum(weights) > 1600){
    greater[i] = 1
  }
}
sum(greater)/10000
```

```
## [1] 0.0025
```

Another example

Suppose you have two machines which make screws. In machine 1 the lengths are normal with mean 3 *inches and standard deviation* 0.5 *inches. In machine 2 they have mean* 2.5 *inches and standard deviation* 0.75 *inches. If you pick one from each, what is the probability the screw from machine 1 is longer?*

- If you know something about normal distributions this is straightforward. But can still be solved using only simulation.

```r
n = 1000
m1 = rnorm(n, mean = 3, sd = 0.5)
m2 = rnorm(n, mean = 2.5, sd = 0.75 )
length(which(m1>m2))/n
```

```
## [1] 0.716
```

The Monty Hall Problem

This is a nice example of a problem which might appear difficult at first, but can be easily solved with some simulation.

> *Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?*

Think about it for a few minutes?

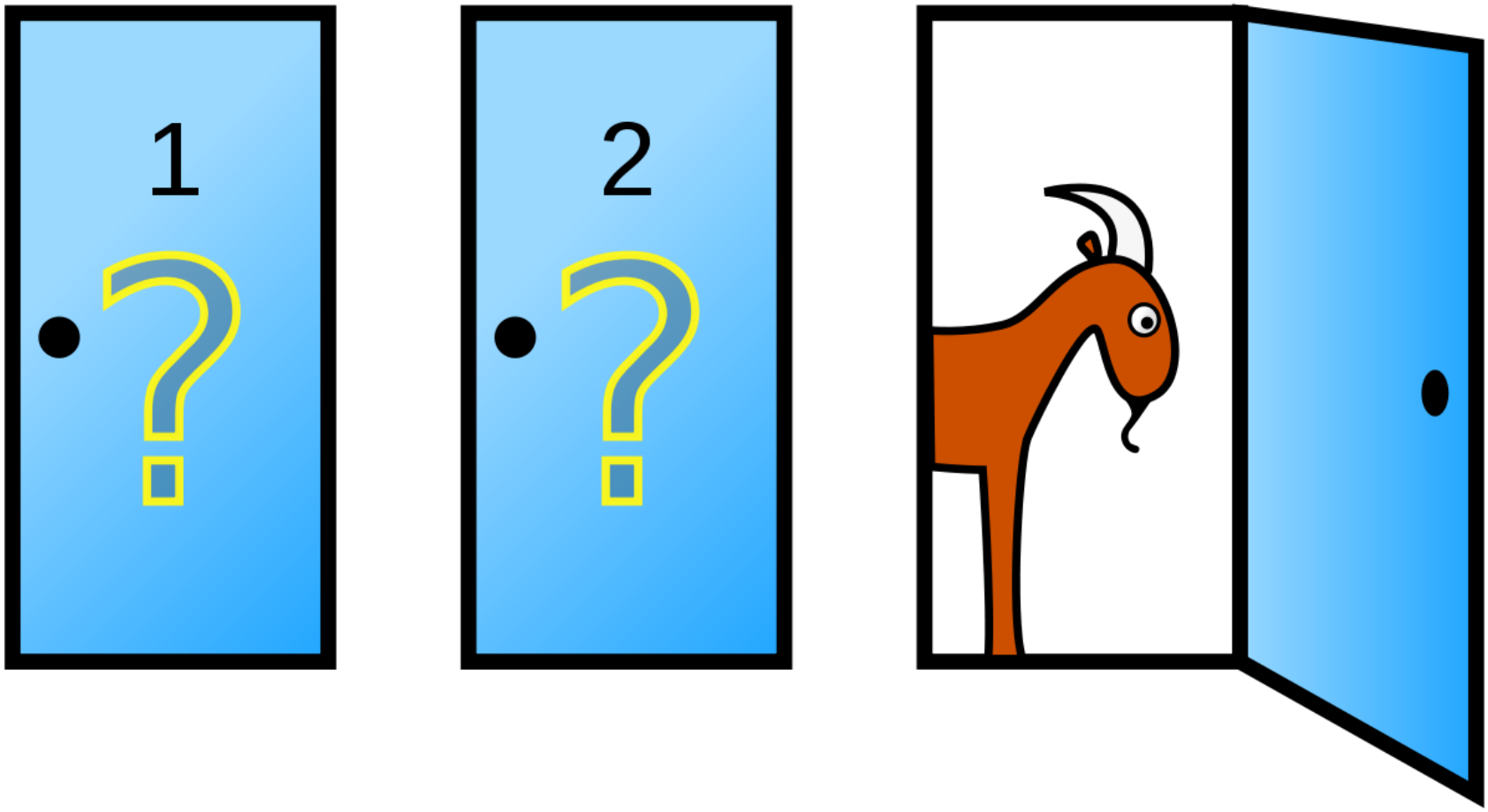


Figure 2: The Monty Hall Problem

The solution

- There are many methods to get to the solution here, including conditional probability and explicitly working through all the solutions. However an approximate solution can also be obtained quite easily by simulation.

R code

```
doors <- c("A","B","C")
outcome = c()
for(i in 1:100){
  prize <- sample(doors,1)
  pick <- sample(doors,1)
  open <- sample(doors[which(doors != pick &
                               doors != prize)],1)
  switch_outcome <- doors[which(doors != pick &
                                  doors != open)]
  if(pick == prize){
    outcome = c(outcome,"NoSwitchWin" )
  }
  if(switch_outcome == prize){
    outcome = c(outcome,"SwitchWin")
  }
}
summary(as.factor(outcome))
```

More simulations

Try $n = 1000, 100000$.

```
## NoSwitchWin   SwitchWin
##         340         660
```

```
## NoSwitchWin   SwitchWin
##       33223       66777
```

- Getting closer and closer to the true value, with win probability of $2/3$ if you switch.

Fake data simulation

- This is a really simple way to check your model fits the data well.
- Fit your model to the data and use the model to predict new "fake" data.
- Compare this fake data to the true data and see if they look inherently different.
- If they do could indicate a poor model which needs to be updated.

Estimating Pi

Accuracy

- How do we know that, as we take more and more simulations, an estimate is guaranteed to get closer and closer to the true answer?
- In many settings, without many assumptions, it can be shown that these simulation estimates do get close to the true value.
- Simularly, this is a statistical method, so it is interesting to see what type of distribution the estimate converges to.
- This can also be proved in general, by the Law of Large Numbers and the Central Limit Theorem.

The Law of Large numbers and the Central Limit Theorem

The bootstrap and Cross Validation

The Bootstrap

- When we just have one (small) sample of data it can be hard to understand properties, or the estimates might be uncertain.
- Want a method which can be used to give uncertainty estimates for statistics of interest.
- Want a simple method that can give estimates of standard errors and confidence intervals for any quantity.

The Bootstrap

- Was invented by Brad Efron in the 70's.
- Is a simple yet flexible method to quantify uncertainty of estimates.
- Can almost always be used.

The bootstrap algorithm

- Have n data points $x_1, \ldots, x_n$ and want to estimate some function of the data $f(x_1, \ldots, x_n)$, such as the sample mean or variance.
- take T samples of size n from your data *with replacement*, where T is large, say $T = 1000$ or even $T = 100000$.
- Each of these t=1,. . . ,T$ samples could contain same of the original data repeated.
- For each of these bootstrap samples calculate f. This will give T estimates of the statistic, $f(t_1), \ldots, f(t_T)$.

Using Bootstrap samples

- Much like the original data can be used to construct a histogram of the random variable, the T estimates of f can be used to give a histogram of that statistic.
- Can also be used to construct confidence intervals for the statistic. What is a confidence interval? Look up the definition?
- Help us gain insight about the uncertainty in our estimate.
- Will do a quick example.

Cross Validation

- Hard to go into detail, but is a method commonly used in statistics and machine learning.
- Often, when fitting a model, need to come up with a method to determine the "best" model, while not overfitting to the data you have.
- Cross validation is a way to do this.

How it works

- For K fold CV, the data is broken up into K randomly chosen equally sized groups.
- For each of these folds at a time, hold it out and fit your chosen model on the other $K - 1$.
- Use this model to predict on the held out fold, giving you a measure of the error of your model on your held out data.
- Repeat this K times, each time holding out a different fold and average these errors. This is a more robust way of finding the best model, can prevent overfitting.

Connections to more advanced machine learning methods

Bayesian Statistics

Many modern machine learning methods use advanced Bayesian models. You will likely see more about Bayesian statistics in other classes, but in brief, the difficulty with these methods is often computing a normalising constant, generally a very high dimensional integral. These can be solved using a simulation method known as Monte Carlo Simulation

Monte Carlo Simulation

- Suppose we didn't know how to compute the integral

$$\int_0^1 e^x \, dx$$

- We can solve this using Monte Carlo

Integrals

- Simulate numbers randomly over the range of x, here $[0, 1]$, calling them $x_1, \ldots, x_n$.
- At each of these points we compute e^x
- Take the average of these values as our estimate $T_n = \frac{1}{n} \sum_{i=1}^{n} e^{x_i}$.
- As n increases then by the Law of Large numbers T_n will get closer to the true value of the integral $(e - 1)$.
- Will show this in a few minutes.

High Dimensions

- While this is a simple example the exact same method works in higher dimensions, although the number of simulations needed to get a good estimate increases.
- Much research is devoted to developing better ways to do these approximations.