# Project 1

*Chris Finkle*

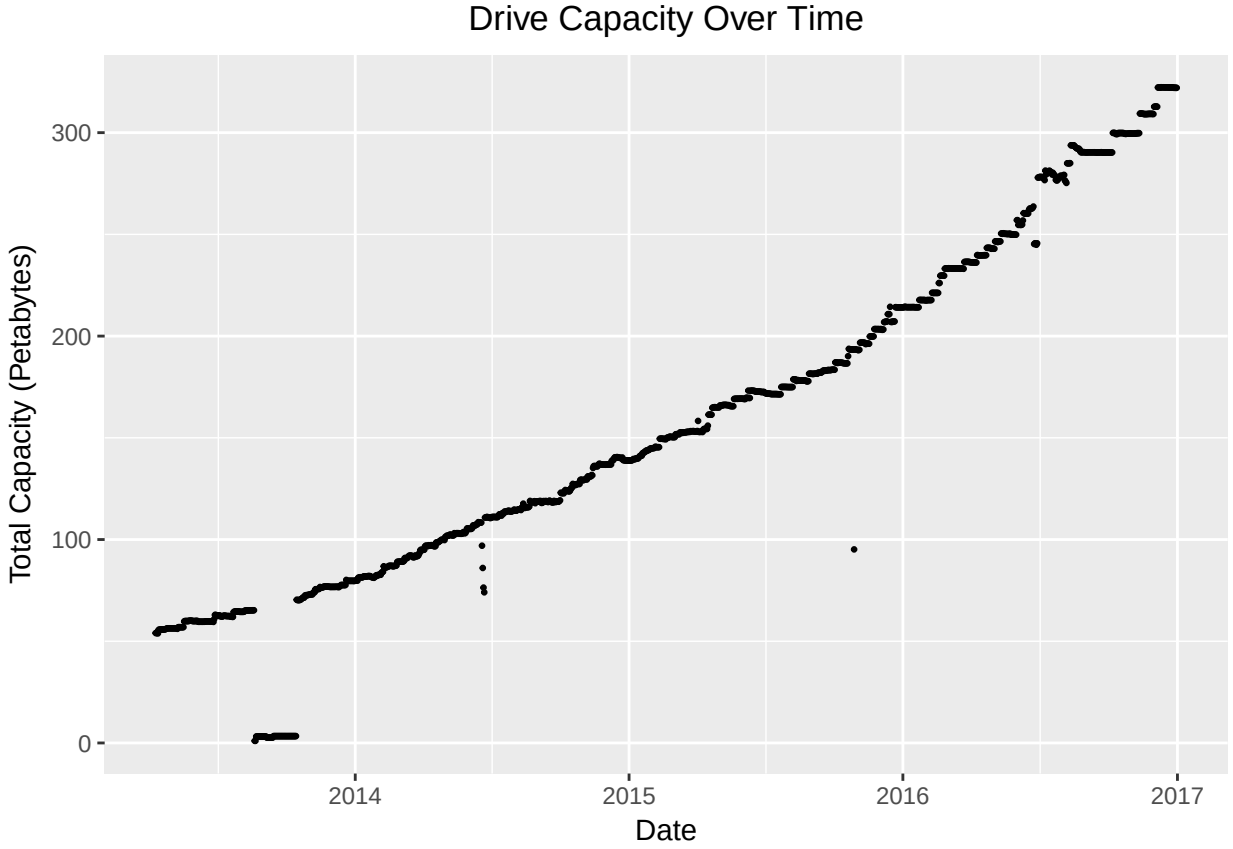
*2/27/2017*

## Backblaze

Backblaze is a service for Windows and Mac users who wish to back up data in a remote location - one might think of it as part 'the cloud'. But like all things in the cloud, it relies on all too terrestrial hardware in the end.

### Hard Drives

Specifically, Backblaze relies on racks and racks of off-the-shelf hard drives organized into StoragePods. In order to keep up with the increasing demands of a customer base whose machines are hooked into the exponential growth of Moore's Law, Backblaze has consistently added drives at an exponential rate since 2013.

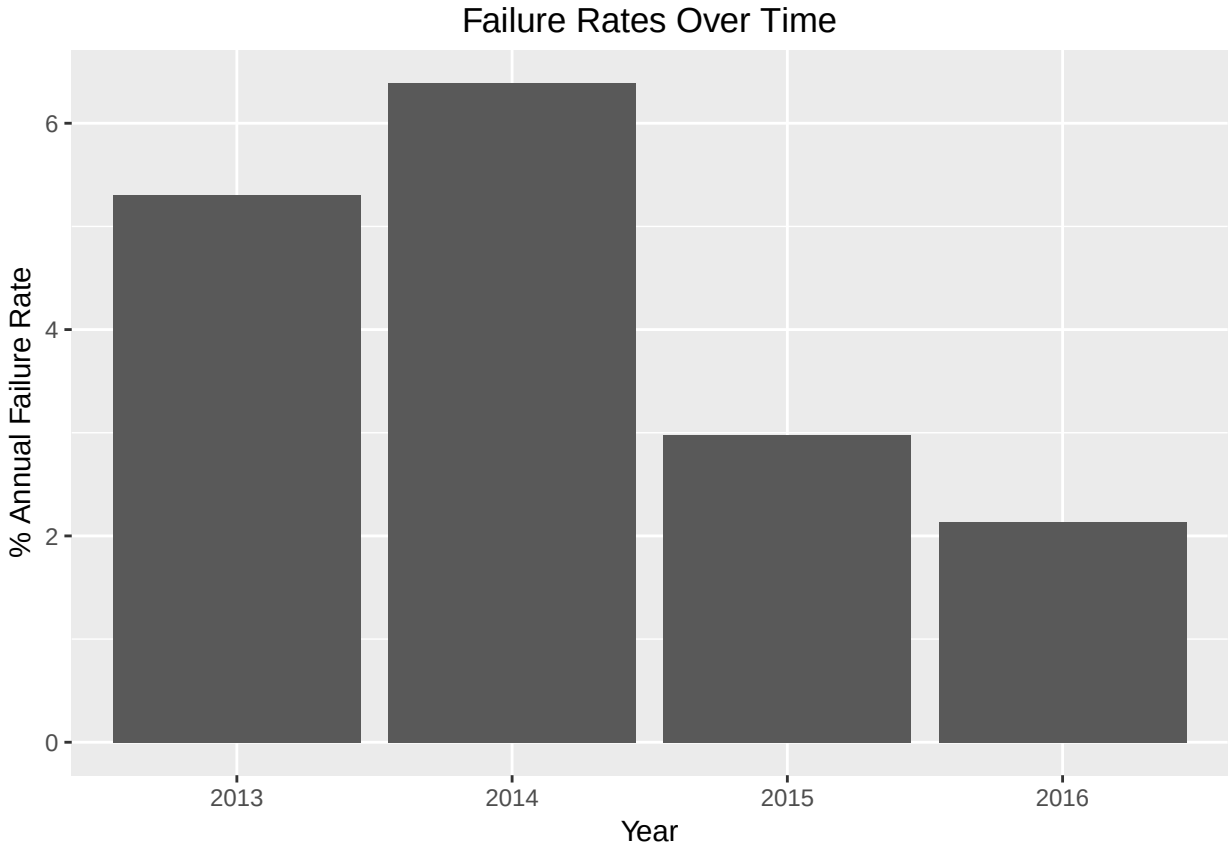



From an initial capacity just over 50 PB, the company's collective capacity exceeded 300 PB in 2016.

### Drive Failure

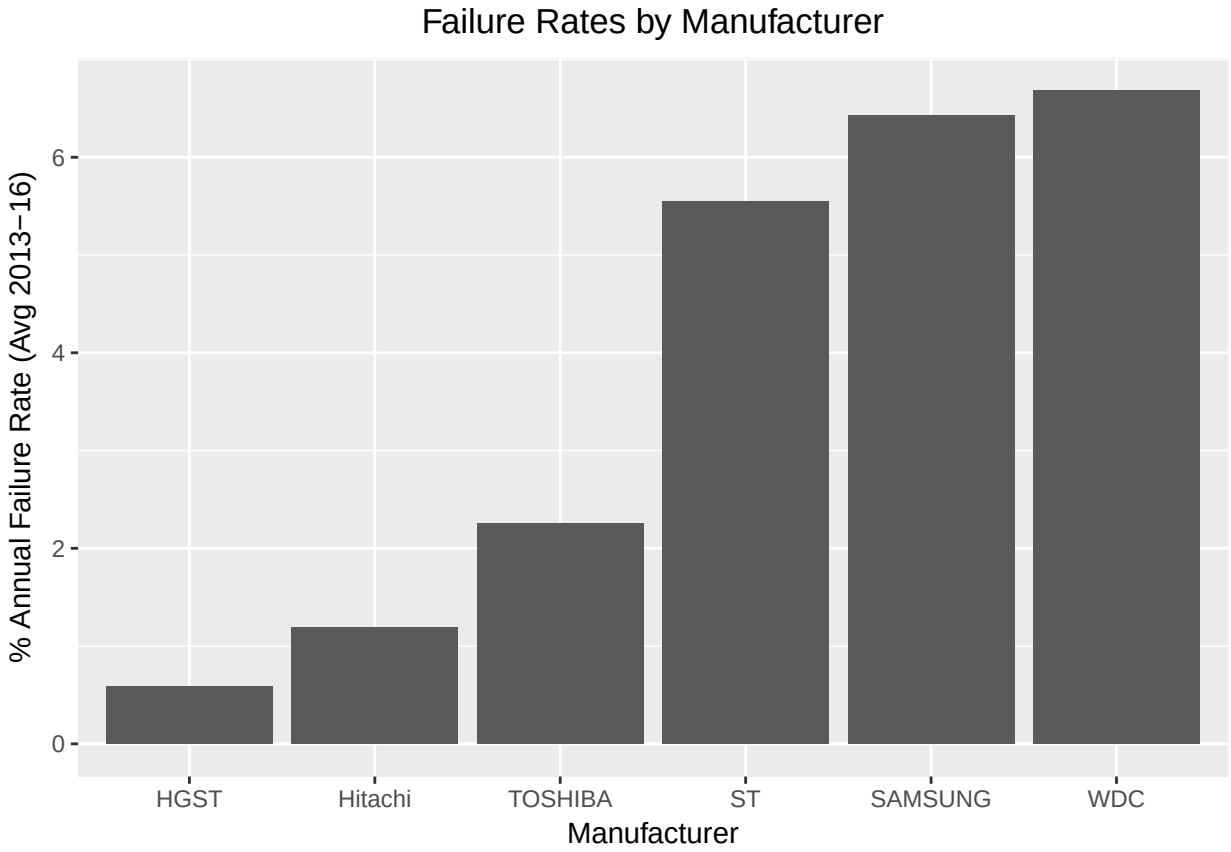
With so many thousands of drives running, a certain number of failures is inevitable. The Annual Failure Rate (AFR) is a way of understanding what proportion of the drives can be expected to fail during a drive

year. A failure rate of 5% indicates that if we ran 20 drives for a year, we would expect one of them to die during that time (for a very simplified example).




In 2014, Backblaze's failure rate hit its highest level, cresting 6%. Something changed in 2015 however, and their AFR has been much smaller in subsequent years. Part of this may be due to more effective data analysis.

For example, using drive feedback it's possible to determine which drive manufacturers fail more often than others so that they can be avoided in the future.

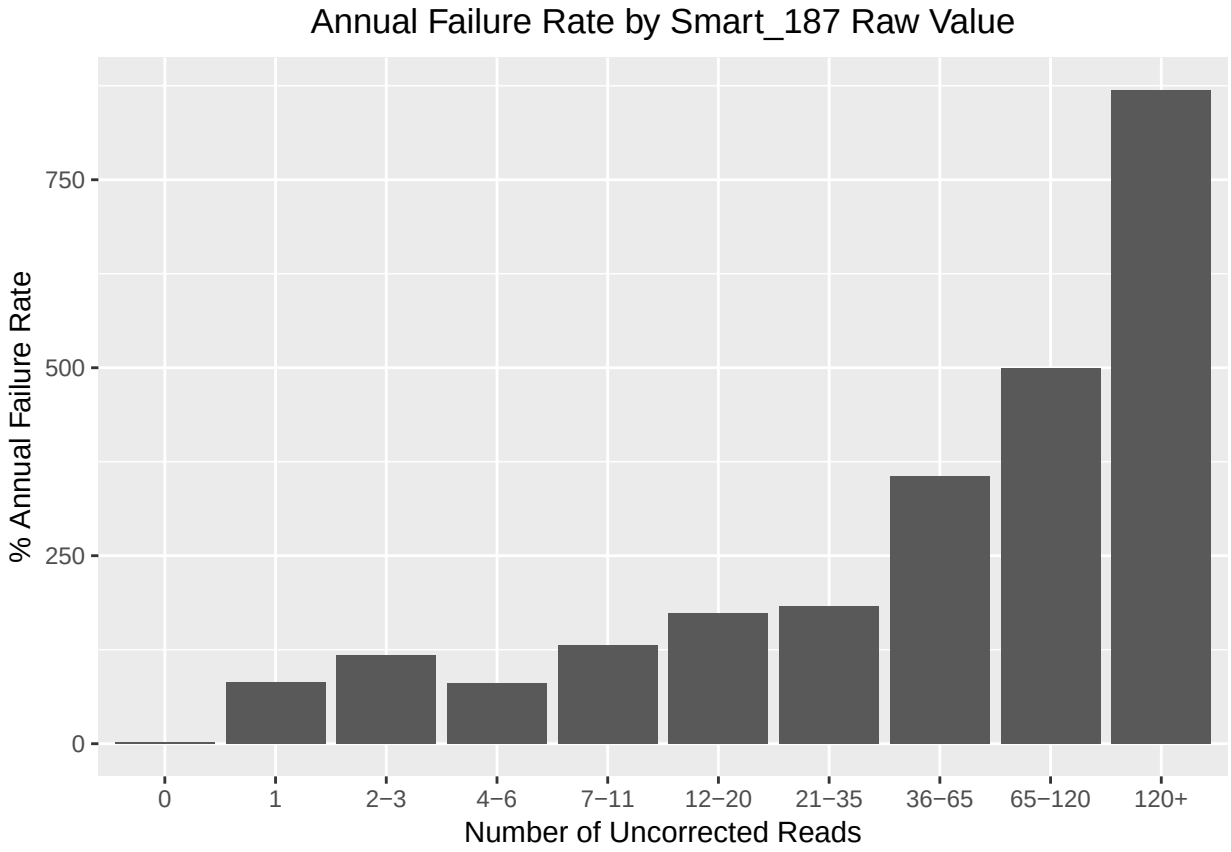



Western Digital (WDC) is the worst offender of the bunch. Along with Samsung, their drives have an AFR of over six percent. Curiously, WDC subsidiary HGST is the most consistent, with a miniscule AFR of less than one percent.

Knowledge of failure rates by manufacturer is important from a big-picture level ("Where should we buy our next thousand drives?") but this type of population-level understanding does little good once the drives are bought and installed. In order to respond efficiently and effectively to individual drive failures, more granular data is required.

### SMART Statistics

Each individual drive is capable of providing dozens of metrics known as SMART stats which describe certain behaviors, some of which can be troubling. Among the most dire is "smart_187", which counts the number of uncorrected read errors a drive suffers during the day.




As can be seen in the graph, AFR goes up along with the value of Smart_187. Even a single uncorrected read error is enough to send failures spiking more than tenfold. Having a non-zero value for Smart_187 certainly seems like a good proxy for imminent drive failure.

Just to be sure, let's perform a Two Sample t-test on failing and non-failing drives to make sure that they have significantly different Smart_187 values on average.

```
    Welch Two Sample t-test

data:  Non_Failing_Drives$num_err and Failing_Drives$num_err
t = -3.5779, df = 3801, p-value = 0.0003507
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -98.10045 -28.64623
sample estimates:
 mean of x  mean of y
 0.7224008 64.0957391
```

The mean Smart_187 value for non-failing drives was less than 1, while for failing drives it was 64. The samples are large enough that we can safely conclude a strong correlation between failure and high Smart_187 ($p<0.0005$). The instant an indvidual drive reports an uncorrected read error, it's time to schedule it for replacement. Indeed, this has been Backblaze's policy since at least late 2014; it may have contributed to their decreased aggregate AFR in subsequent years.

**Notes**

The raw data collected from Backblaze's hard drives was dozens of columns wide and distributed across hundreds of daily .csv files. In order to collect the five stats required for these analyses into a single file, they were each routed into this python script:

```
#!/usr/bin/python
import sys

for arg in sys.argv:
    if arg.split(".")[-1]=="csv": #necessary to prevent program from reading itself
        f = open(arg)
        o = open("drive.csv", 'r+') #output file; must already exist in working directory
        o.seek(0,2) #position write at end of existing file
        line = f.readline()
        vals = line.split(",")
        sm = 4 #needs to equal something
        for x in range(len(vals)):
            if vals[x] == "smart_187_raw":
                sm = x
        idxs = (0, 2, 3, 4, sm)
        line = f.readline()
        while line:
            vals = line.split(",")
            keep = [vals[x] for x in idxs]
            nrow = ",".join(keep)
            o.write(nrow+'\n')
            line = f.readline()
        f.close()
        o.close()
```

The routing was automated (and the column names inserted) by a bash script:

```
#!/bin/bash
PAR="/bigdata/data/backblaze/"
echo "Date,Model,Capacity,Failure,Smart_187_raw" > drive.csv
for year in `ls $PAR`; do
    for x in `ls $PAR/$year`; do
        echo $x
        ./file_reader.py $PAR/$year/$x;
    done
done
```

This concatenation took approximately 5 minutes and resulted in an unwieldy 2.5 GB text file of nearly 60 million lines. For problems an order of magnitude or more larger than this one, such techniques will no longer be practical. However, their performance was satisfactory in this situation.