IE 3301 - 004

ENGINEERING PROBABILITY

REAL-WORLD DATA: ANALYSIS

Part I & II

December 3, 2017

Joe Cloud

Student ID: 1000921236

University of Texas at Arlington

I, Joe Cloud, did not give or receive any assistance on this project, and the report submitted is wholly my own.

Contents

Introduction

The aim of this project, as stated in the project brief is to analyze real-world data using techniques covered in the introductory engineering probability course. This includes some basic descriptive analysis of the data, as well as performing the chi-square good-of-fit test in order to determine how well conclusions of the analysis conforms with the expected values for select distributions. In this project, data collection was performed on two independent data sets: one is a sample of a continuous random variable that is suspected to be normally distributed. And the second dataset is a measure of the time interval between events, which is suspected to follow an exponential distribution. Each dataset is statistically summarized. For my project, I chose to collect a set of resistance values from a batch of 100 1kΩ $\pm$ 5% resistors ($k\Omega$ is the measurement of resistance, in units of Kilo Ohms), for the first set. For the second set, I compiled login timestamps for users from a compute cluster and wrote preprocessing scripts to extract the login intervals. The result was a dataset of 120 events containing the login interval (in seconds) between user connections.

Data

Set One

Data collection began with the process of measuring the resistance of each resistor and recording its value. The measurements were conducted with a calibrated 4.5 digit digital multimeter (see Appendix V for equipment information). To perform this measurement, a digital multimeter was placed in 'resistance' measuring mode, and with probes attached to the correct ports, each resistor was attached to the end of the probe on either side of the filament. Once attached the resistive value would be flashed on the display of the digital multimeter. Each measurement was entered into a spreadsheet and later outputed to a CSV file for further processing by data analysis scripts.
Before data analysis was performed, 0.33Ω was subtracted from each value, this is done to account for the resistance added by the multimeter probes. Although this minor offset did not impact the distribution of measurements for the purposes of the project, it did provide us with results that more accurately represent the true values. The values are offset by approximately 0.03%.

In an ideal world, the resistors would measure identically to the 1kΩ label. In reality, the cost to refine manufacturing to attain such level of accuracy is expensive. Tolerances guarantee a range of possible [random] values which are acceptable for most electronics. This dataset was created to explore the distribution of these values.

Set Two

The data collection for Set Two revolved around login access to a compute cluster and will be difficult to replicate without access to a system with comprehensive logging and login activity. Though, it is possible for the reader to perform similar analysis with data from a personal machine's logged data. On a Unix-like operating system this can be performed with the use of the `last` command. One of the benefits of computer-triggered collection is that it is much less susceptible to timing errors as opposed to that of a 'human polling' based approach.
I included the source code necessary to collect the data in Appendix IV.C. The process is largely computational. After the raw data was collected with the `last` command, the data was parsed through a preprocessing script

written to convert the timestamps in HH:MM:SS format to that of total seconds. Then, each event was subtracted from the next, yielding interlogin times. The negative sign is discarded as only the magnitude is significant. As a side note, all source code written for this project are included in the appendix.

Descriptive Analysis

This section details the results of analysis performed on both datasets One and Two. To analyze the data, python scripts were written to perform the requested calculations and output the desired plots for each dataset. The scripts are included as part of the Code section of Appendix IV.A.

Set One

The first step in analysis for Set One was calculating the mean $\bar{X}_1$, which is 0.98349 $k\Omega$, between a minimum value of $0.97987k\Omega$ and a maximum value of $0.98827k\Omega$. The next step was calculating the standard deviation, σ_1. This was done using `numpy` as opposed to doing it by hand with $\sigma_1 = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \bar{X}_1)^2}$. The result for σ_1 was 0.001739, which would indicate that the spread from the mean is fairly low, considering the significance of the value. The last calculation necessary for box-and-whisker plot was the quartiles, in which $Q1 = 0.98257k\Omega$, $Q2 = 0.98659k\Omega$, $Q3 = 0.98467k\Omega$.

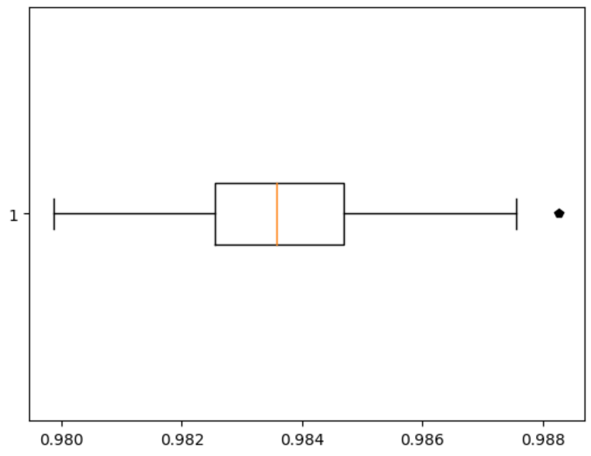

Figure 1: Set One: Box-and-whisker plot

The box-and-whisker plot is a helpful method of visualizing spread within a dataset. The vertical line segments end on either side of the box represent the minimum and maximum values within the sample space, while the dot to the right represents outliers within the data. The box itself is useful for showing the quartiles (first being the left edge of the box, with third being the right). In this set, the data appears to be dispersed somewhat evenly. The whiskers are similar in length, with only a slight bias towards the left. This would indicate that the resistant of the 1kΩ hold an average lower than the median resistance. (x-axis is resistance in kΩ)

The frequency table (Appendix III.A) was created first by determining a bin size. The range of the data was divided into 10 equal parts. Each bin would contain the tally of the values within a particular slice. For the resistor values, which is a continuous random variable that we suspect to be normally distributed, it is promising to see that the tallies peak where the center-most values are located. The right-most extreme values do appear to decrease quite significantly, while the lower end appears to contain more tallies than we would expect for a proper normal distribution.

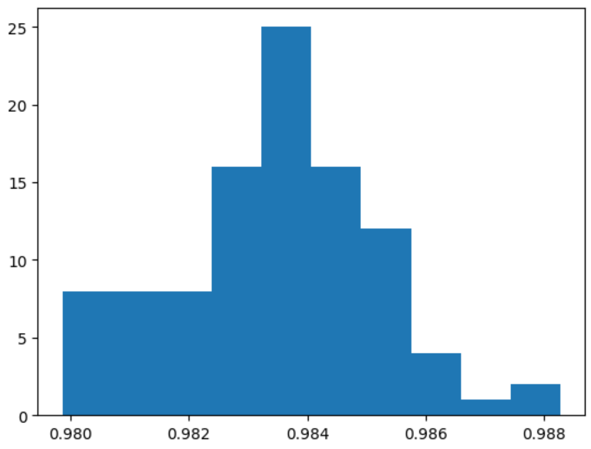

Figure 2: Set One: Histogram

The histogram represents the distribution of the resistive values across 10 bins, where the y-axis is the number of values tallied within the slice, and x-axis is the measured resistance in units of kΩ. Based on the histogram and previous data analysis, the sample of resistors do not follow a normal distribution due to a lack of symmetry.

Set Two

The first step in analysis for Set Two was calculating the mean $\bar{X}_2$, which is 8445.05 seconds (s), between a minimum value of 1.0s (restricted due to measurement resolution) and a maximum value of 83749s. The next step was calculating the standard deviation, σ_2, which was 15392.12s. This indicates that spread from the mean is high. The last calculation necessary for box-and-whisker plot was the quartiles, in which $Q1 = 251.75s, Q2 = 1865.0s, Q3 = 8052.25s$.

In this set, the data appears to be heavily right skewed, with the majority of the values located on the lower end of the scale (in seconds) and a few large outliers to the right. This foreshadows an interesting frequency distribution and histogram. Since this interval data is extracted from timestamps, we suspect the data to follow an exponential distribution. It is based on a continuous random variable (through the time stamp), but the values are discretized due to the limits of logging precision.

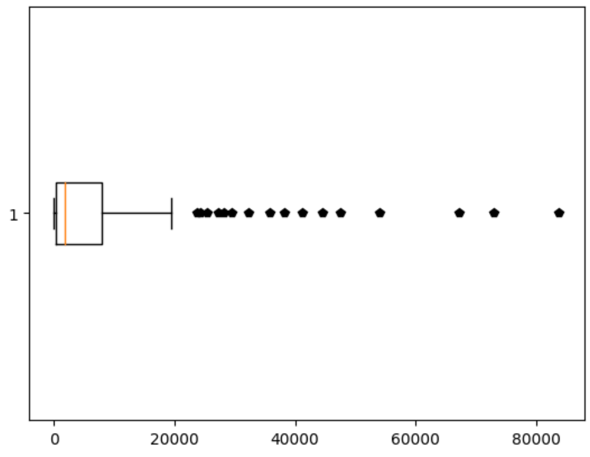

Figure 3: Set Two: Box-and-whisker plot

Similar to Set One, the frequency table (Appendix III.B) was created first by determining a bin size. The range of the data was divided into 10 equal parts as well. Each bin contained the tally of the values within a particular slice. When tabled, the values for the login intervals showed that the vast majority of tallies were entered into the first slice, and a significant decrease in each successive step. The second through the last bin combined makes up less than a third of the tallies of the first bin.

The histogram to the right represents the distribution of the login interval values across 10 bins. Where the y-axis is the number of values tallied within the slice, and x-axis is time between successive logins in seconds. Based on the histogram and previous data analysis, I conclude that the data follows an exponential distribution. As interval increases there is an exponential decrease in the values recorded, with some outliers to the extreme but with more samples the overall distribution would continue to show strong resemblance to that of the standard exponential distribution.

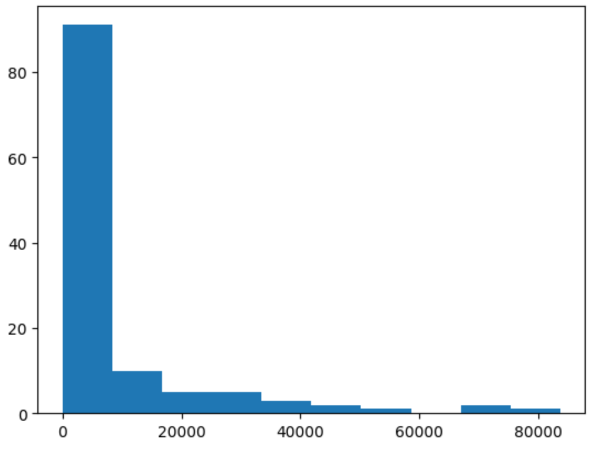

Figure 4: Set Two: Histogram

CHI-SQUARE GOODNESS-OF-FIT TEST

Our previous analysis helped us determine how well we thought the data followed the suspected distributions. With Chi-Square, we can assess this more objectively. The Chi-Square Goodness-of-Fit Test measures the difference between the actual distribution of data and that which we should expect given the sample mean, and standard deviation of each dataset. The process is essentially recomputing a frequency table strictly based on the cumulative distribution function (cdf) given the previous mean and standard deviation. Then sum the square of the subtraction between the actual counts and expected frequency. There is a caveat using this method though. Chi-Square does not work well with expected frequencies below 5%. The solution to this is combining classes to increase the percentanges in the bins below the threshold. To do this practically, effort was put into a python function that would find expected frequencies that were under 5% and select an adjacent class that made most sense and combined them, it will also combine the ranges as well as their respective bin counts. The results of the test are discused for each dataset more below.

Set One

In order to test the hypothesis that set one followed a normal distribution, the χ^2 value was calculated through the Chi-Square Goodness-of-Fit test. After merging the classes together using the algorithm developed in appendix IV.A, a table with two fewer classeses of observations emerged. The following table was generated by the calculations performed in the program:

Class	Observed	Class Probability	Expected	χ^2 Component
$0.00000 \leq X \leq 0.98071$	8	0.0559182	5.59182	1.03711
$0.98071 < X \leq 0.98155$	8	0.0779745	7.79745	0.00526152
$0.98155 < X \leq 0.98239$	8	0.131646	13.1646	2.02615
$0.98239 < X \leq 0.98323$	16	0.176996	17.6996	0.163204
$0.98323 < X \leq 0.98407$	25	0.189509	18.9509	1.93089
$0.98407 < X \leq 0.98491$	16	0.161588	16.1588	0.00156048
$0.98491 < X \leq 0.98575$	12	0.109723	10.9723	0.0962668
$0.98575 < X$	7	0.0966457	9.66457	0.734636
Total	100	1.000000	100.000	5.99508

Table 1: Set One: Chi Square table

More blah

Set Two

Class	Observed	Class Probability	Expected	χ^2 Component
$0.00000 \le X \le 8375.8$	91	0.417017	50.042	33.5229
$8375.80 < X \le 16750.6$	10	0.244639	29.3567	12.7631
$16750.6 < X \le 25125.4$	5	0.14198	17.0377	8.50499
$25125.4 < X \le 33500.2$	5	0.0824006	9.88807	2.41637
$33500.2 < X \le 41875$	3	0.0478225	5.7387	1.307
$41875.0 < X$	6	0.0661399	7.93679	0.47263
Total	120	1.000000	120.0000	58.9870

Table 2: Set Two: Chi Square table

CONCLUSION

This is my conclusion for the paper.

APPENDIX

I - Dataset: One

Measured resistance of each resistor in the batch of 100.

0.98227	0.98467	0.98377	0.98277	0.97987	0.98467	0.98437	0.98337	0.98317	0.98507
0.98027	0.98077	0.98487	0.98037	0.98457	0.98237	0.98387	0.98367	0.98257	0.98487
0.98377	0.98217	0.98017	0.98487	0.98447	0.98737	0.98317	0.98287	0.98477	0.98417
0.98207	0.98387	0.98757	0.98427	0.98477	0.98257	0.98537	0.98517	0.98037	0.98007
0.98527	0.98617	0.98397	0.98127	0.98357	0.98367	0.98387	0.98097	0.98357	0.98077
0.98287	0.98087	0.98627	0.98107	0.98377	0.98327	0.98537	0.98357	0.98577	0.98547
0.98247	0.98507	0.98337	0.98367	0.98527	0.98347	0.98827	0.98207	0.98337	0.98297
0.98297	0.98097	0.98387	0.98287	0.98467	0.98327	0.98157	0.98037	0.98487	0.98117
0.98547	0.98397	0.98437	0.98337	0.98317	0.98287	0.98507	0.98397	0.98287	0.98327
0.98167	0.98197	0.98317	0.98547	0.98377	0.98637	0.98057	0.98277	0.98547	0.98447

II - Dataset: Two

The interlogin time for users on the compute cluster, in seconds. 120 interlogin times recorded.

1796	67183	7901	13437	6645	916	41159	8269	2684	38237	3263	47470
6397	24371	28258	12257	1348	18621	6675	2691	25385	86	10	18647
68	31	19	2202	124	508	3	72894	5630	15758	7980	431
3918	3541	11982	35755	6654	2526	15853	3533	507	367	1404	2306
2237	23714	887	141	5493	520	83749	3516	29573	27200	2	32295
2197	3579	4556	166	864	56	113	98	5	1645	3	7368
1846	2136	2214	213	535	632	1440	268	243	303	1182	54091
1	19565	690	243	83	135	84	3346	6641	1373	457	393
19	187	376	9292	1214	589	567	1240	1884	207	423	5
905	44505	71	11948	10404	2151	2413	227	10287	196	25	8480

III - Tables

A - Set One: Frequency table

Value Range	Count
$0.00000 \le X \le 0.98071$	8
$0.98071 < X \le 0.98155$	8
$0.98155 < X \le 0.98239$	8
$0.98239 < X \le 0.98407$	16
$0.98407 < X \le 0.98491$	25
$0.98491 < X \le 0.98575$	17
$0.98575 < X \le 0.98659$	12
$0.98659 < X \le 0.98743$	4
$0.98743 < X \le 0.98827$	1
$0.98827 < X$	2

Table 3: Set One: Frequency table

B - Set Two: Frequency table

Value Range	Count
$0.00000 \le X \le 8375.00$	91
$8375.00 < X \le 16750.0$	10
$16750.0 < X \le 25125.4$	5
$25125.4 < X \le 33500.2$	5
$33500.2 < X \le 41875.0$	3
$41875.0 < X \le 50249.8$	2
$50249.8 < X \le 58624.6$	1
$58624.6 < X \le 66999.4$	0
$66999.4 < X \le 75374.2$	2
$75374.2 < X$	1

Table 4: Set Two: Frequency table

IV - Code

A - Descriptive Analysis

```
#AUTHOR: Joe Cloud
#PURPOSE: Perform analysis for Probability for Engineers, project
#UTA FALL 2017


import numpy as np
from scipy.stats import norm, expon, chisquare
import sys
import matplotlib.pyplot as plt
import tabulate

DATA_FILE = "../set_one/resistor_vals_offset.csv" # Set to default list
QUARTILES = [25, 50, 75]
EXPECT_NORMAL = True

if len(sys.argv) > 1:
   DATA_FILE = sys.argv[1]
   EXPECT_NORMAL = False

OUTPUT_FILE = "results/" + DATA_FILE.split('/')[-1].split('vals')[0]

def main():


   sample_vals = np.genfromtxt(DATA_FILE, delimiter=',')
   print(sample_vals)
```

```
n = len(sample_vals)
print("There are %d samples" % n)

print("Min value is: %f" % min(sample_vals))
print("Max value is: %f" % max(sample_vals))

sample_mean = np.mean(sample_vals)
print("Mean value is: %f" % sample_mean)

sample_std = np.std(sample_vals)
print("STD value is: %f" % sample_std)

# Calculate quartiles
sample_quarts = []
for quart in QUARTILES:
   sample_quarts.append(np.percentile(sample_vals, quart))

print("Quartiles: ", *sample_quarts, sep=', ')

#generateTable(sample_vals)

# Construct box-and-whisker plot, a.k.a. boxplot
fig = plt.figure()
ax = plt.subplot(111)
ax.boxplot(sample_vals, 0, 'kp', 0)
fig.savefig(OUTPUT_FILE + 'boxplot.png', bbox_inches='tight')
fig.clf()

num_bins = 10

# Frequency table
counts, ranges = np.histogram(sample_vals, bins=num_bins)
print("histogram:", counts, ranges)

# Histogram data
fig = plt.figure()
ax = plt.subplot(111)
ax.hist(sample_vals, bins=num_bins)
fig.savefig(OUTPUT_FILE + 'histogram.png', bbox_inches='tight')
fig.clf()

# Part 2

new_counts, new_ranges, new_expp = expected_frequencies(counts, ranges, sample_mean,
    sample_std, n)
```

```
  print(counts, new_counts)
  print(new_expp)

  print(sum(new_expp))
  print(counts)

  print(ranges, new_ranges)

  print(counts, new_counts)
  cS = chisquare(new_counts, new_expp)
  print(cS)

def expected_frequencies(counts, ranges, sample_mean, sample_std, n):

  counts = np.copy(counts)
  ranges = np.copy(ranges)

  STATFUNC = norm.cdf
  if EXPECT_NORMAL == False:
    STATFUNC = expon.cdf

  expp = []
  expp_sum = 0



  for i in range(0, len(ranges)):

    upper = 1
    if i < len(ranges) - 1:
      upper = STATFUNC(ranges[i+1], sample_mean, sample_std)

    lower = STATFUNC(ranges[i], sample_mean, sample_std)
    if i == 0:
      lower = 0

    expp.append(upper - lower)

    expp_sum += expp[i]
    print(i, expp_sum, upper, lower)

  print(expp)

  if EXPECT_NORMAL == True:
    expp[-2] += expp[-1]
    expp = expp[:-1]
```

```
else:
   expp[1] += expp[0]
   expp = expp[1:]
print("Sum should equal 1, is %f" % sum(expp)) # CHECK
i = 0
clean_pass = False
found_dirt = False
while(not clean_pass):

   found_dirt = False

   if expp[i]*n < 5.0: # Generally we don't want valus to be below 5%
      print("Found dirt")
      found_dirt = True
      print(expp[i])

      if i <= len(expp)/2:

         if i == 0:
            expp[1] += expp[0]
            counts[1] += counts[0]
            expp = expp[1:]
            counts = counts[1:]
            ranges = ranges[1:]
         else:
            if expp[i - 1] < expp[i + 1]:
               expp[i] += expp[i - 1]
               counts[i] += counts[i - 1]
               expp = np.delete(expp, i - 1)
               counts = np.delete(counts, i - 1)
               ranges = np.delete(ranges, i - 1)
            else:
               expp[i] += expp[i + 1]
               counts[i] += counts[i + 1]
               expp = np.delete(expp, i + 1)
               counts = np.delete(counts, i + 1)
               ranges = np.delete(ranges, i + 1)
      else:
         maxv = len(expp) - 1
         maxc = len(counts) -1
         cdelta =0 # maxv - maxc

         print("MAX", maxv, maxc)
         if i == maxv:
            expp[maxv - 1] += expp[maxv]
            counts[maxc - 1] += counts[maxc]
```

```
                expp = expp[:-1]
                counts = counts[:-1]
                ranges = ranges[:-1]
            else:
                if expp[i - 1] < expp[i + 1]:
                    expp[i] += expp[i - 1]
                    counts[i - cdelta] += counts[i - 1 - cdelta]
                    expp = np.delete(expp, i - 1)
                    counts = np.delete(counts, i - 1 - cdelta)
                    ranges = np.delete(ranges, i - 1)
                else:
                    expp[i] += expp[i + 1]
                    counts[i - cdelta] += counts[i + 1 - cdelta]
                    expp = np.delete(expp, i + 1)
                    counts = np.delete(counts, i + 1 - cdelta)
                    ranges = np.delete(ranges, i + 1)

        if (i == len(expp) - 1) and found_dirt == False:
            clean_pass = True

        i = (i + 1) % (len(expp))

        print("EXPP:", expp,"\nCOUNTS: ", counts,"\nRANGES:", ranges)

    return counts, ranges, expp*n



def generateTable(data):

    data_c = data.reshape(10, int(len(data)/10))
    print(data_c.shape)

    gen_table = tabulate.tabulate(data_c, tablefmt="latex")
    print(gen_table)

    outfile = open(OUTPUT_FILE + 'latex_gen.txt', 'w')
    outfile.write("\n\n\n")
    outfile.write("%s\n" % gen_table)


if __name__ == "__main__":
    main()
```

B - Data Processing: Set One

```
#!/usr/bin/env python3

# This simply offsets each value by 0.33

FILE_NAME="resistor_vals.csv"

def main():

   f = open(FILE_NAME, 'r')

   data = f.readlines()

   print(data)

   data_output = []
   for val in data:
      data_output.append(float(val) - 0.00033)

   outfile = open("resistor_vals_offset.csv", 'w')

   for val in data_output:
      outfile.write("%s\n" % val)


if __name__ == "__main__":
   main()
```

C - Data Processing: Set Two

```
#!/usr/bin/env python3

# This performs necessary pre-processing for raw data from 'last' command

FILE_NAME = "stripped_batch_three.log"

def main():

   f = open(FILE_NAME, 'r')
   data = f.readlines()
   data_conved = convToSeconds(data)
   print(data_conved)
```

```
  data_interval = calcInterval(data_conved)
  print(data_interval)

  outfile = open("logininterval_vals.txt", 'w')

  for val in data_interval:
    outfile.write("%s\n" % val)


def convToSeconds(data):

  in_seconds = []

  for login in data:
    # Takes value in format HH:MM:SS and tokenizes, adds up each delimited element
    # w/ respective multiplier.
    temp_tok = login.split(':')

    in_seconds.append(int(temp_tok[2])+int(temp_tok[1])*60+int(temp_tok[0])*3600)

  return in_seconds


def calcInterval(data):
  in_intervals = []
  # Go through and get the difference between T1 and T0 to determine login intervals
  # Must abs values bc for ex, T0 may be 23:34:46 and T1 00:12:49 where the difference would
      be negative
  for i in range(0, len(data) - 1):
    in_intervals.append(abs(data[i] - data[i+1]))

  return in_intervals


if __name__ == "__main__":
  main()
```

```
#!/usr/bin/env bash

last -Fwx > last_batch_$1.log
cat last_batch_$1 | awk '{ print $7 }' > stripped_batch_$1.txt
```

V - Equipment Used

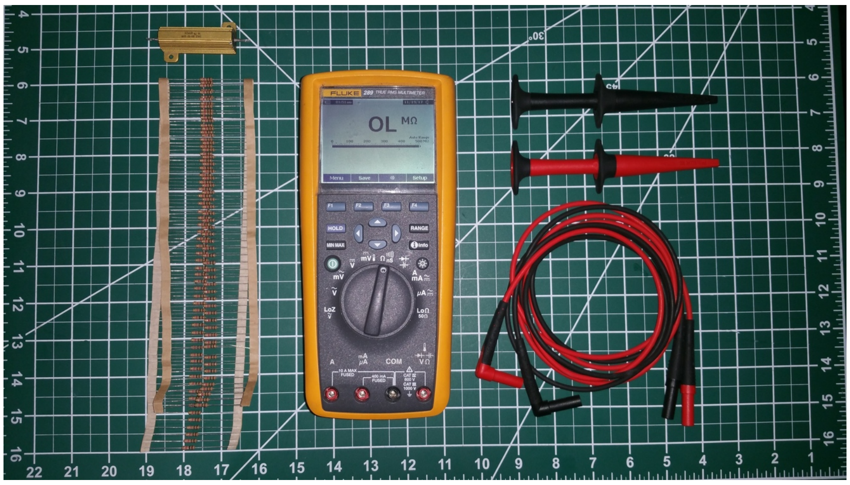

Figure 5: Set One: Fluke 289 w/ resistors

References

Walpole, R.E., 2016. *Probability and Statistics for Engineers and Scientists.* Prentice Hall.