IE 3301 - 004

ENGINEERING PROBABILITY

# REAL-WORLD DATA: ANALYSIS

*Part II*

December 6, 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 part of the project $\chi^2$ Goodness-of-Fit test was performed on each of the following datasets previously assessed using descriptive analysis in Part I.
I chose to collect a set of resistance values from a batch of 100 1k$\Omega \pm 5\%$ resistors ($k\Omega$ is the measurement of resistance, in units of Kilo Ohms), for the first set, this was purchased on Amazon.com. For the second set, I compiled login timestamps for users of a compute cluster that I manage in the Physics department. The log was a slice from the first week of November. I also 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.

## CHI-SQUARE GOODNESS-OF-FIT TEST

Our previous analysis helped us determine by inspection how well the data followed the suspected distributions. With Chi-Squared, we can assess this objectively. The Chi-Squared 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-Squared 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 with the next lowest percentage and combine them. It will also combine the ranges as well as their respective bin counts. The code to do this is included in the appendix code section.
The results of the test are discused for each dataset more below.

### Set One

First, we must define our hypotheses. We define $H_0$ to be the case that Set One data follows a normal distribution. And $H_1$ to be that it does not follow the distribution. We reject $H_0$ if our $\chi^2$ value is greater than $\chi^2_{\alpha,k-1}$, which in our case is 14.407 based on Table A.5 (Walpole, 740) given $\alpha = 0.05$ and $k = 8$ (chosen since the reduction of classes resulted in eight remaining). In order to test the hypothesis that Set One followed a normal distribution, the $\chi^2$ value was calculated through the Chi-Squared Goodness-of-Fit test. After merging the classes together using the algorithm included in appendix IV.A, a table with two fewer classeses of observations emerged. The following table was generated result:

| Class | Observed | Class Probability | Expected | $\chi^2$ 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

The Observed column shows the same data as that of Appendix III.A except with two fewer classes. The class probabilities values were calculated using the cdf for the normal distribution, for each respective class. And the expected frequencies is simply the same values scaled to the number of total observations in Set One. The Chi Squared value for each row, was calculated by $(observed_i - expected_i)^2/expected$ and them summed $\forall i$ to a total of 5.99508, which, since it is lower than 14.407, we fail to reject our hypothesis $H_0$. This is a weak conclusion, but leads us to conclude that it does indeed follow a normal distribution.

## Set Two

For Set Two, we follow the same procedure as that for Set One with the difference being we use the cdf function for the exponential distribution. This was handled automatically by the program, as it would pass a reference to the the correct cdf function when switching data files. Again, we define our hypothesis $H_0$ to be the case that Set Two data follows an exponential distribution. And $H_1$ to be the case that it does not follow the exponential distribution. The degrees of freedom for the recombined table (due to 5% for $\chi^2$) resulted in five separate classes, thus, in order to reject $H_0$ we must calculate a $\chi^2_{\alpha,k-1} = 9.488$ where $\alpha = 0.05$ and $k = 5$.

| Class | Observed | Class Probability | Expected | $\chi^2$ Component |
|---|---|---|---|---|
| $0.00000 \leq X \leq 8375.8$ | 91 | 0.417017 | 50.042 | 33.5229 |
| $8375.80 < X \leq 16750.6$ | 10 | 0.244639 | 29.3567 | 12.7631 |
| $16750.6 < X \leq 25125.4$ | 5 | 0.14198 | 17.0377 | 8.50499 |
| $25125.4 < X \leq 33500.2$ | 5 | 0.0824006 | 9.88807 | 2.41637 |
| $33500.2 < X$ | 9 | 0.1141385 | 13.6966 | 1.6104 |
| Total | 120 | 1.000000 | 120.0000 | 58.8176 |

**Table 2:** Set Two: Chi Square table

After computing the new observed values based on the reduction of the degrees of freedom, and calculating the class probabilities and in turn the expected frequencies- we repeat the calculation for $\chi^2$, which results in 58.8176 after summing the last column. Based on this calculated value, we must reject our null hypothesis $H_0$. This is a strong conclusion for the data not following an exponential distribution.

## CONCLUSION

After performing $\chi^2$ Goodness-of-Fit test on each set of data, it became clear that although I felt confident in my Part I conclusions made by visual inspection- according to the tests those are not the case (with knowledge that one conclusion was weak). In Part I, I concluded that Set One did not follow a normal distribution since the histogram lacked the symmetry that I had to come to expect from normal distributions, where the tails approach 0. It is a weak conclusion, but over the confidence interval it must be concluded that we fail to reject $H_0$.

For Set Two, I was much more more surprised by the results of the $\chi^2$ test, since the conclusion then was that it does not follow an exponential distribution when by visual inspection of histogram it appeared to follow the expected curve well with the exception of some high outliers (which was not the reason why $H_0$ was rejected). $H_0$ could have been rejected based on the value from the first class alone.

In all, the methods employed in Part II were helpful for making known another method of verifying distributions of data, I would have liked for the conclusions to agree with my initial assesement, but after rechecking my process and code I could not identify a source of error- based on that, I conclude the findings of my $\chi^2$ analysis to be true, yet unsatisfying.

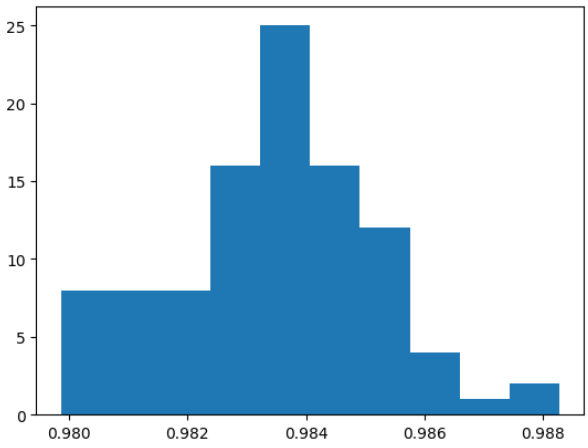


**Figure 1:** Set One: Histogram

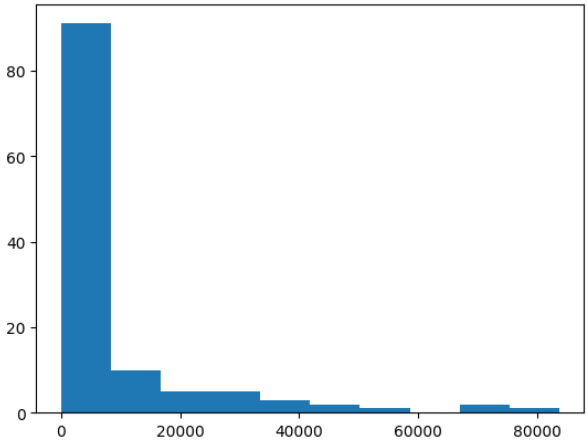


**Figure 2:** Set One: Histogram

# 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 |
|---|---|
| $0.00000 \leq X \leq 0.98071$ | 8 |
| $0.98071 < X \leq 0.98155$ | 8 |
| $0.98155 < X \leq 0.98239$ | 8 |
| $0.98239 < X \leq 0.98407$ | 16 |
| $0.98407 < X \leq 0.98491$ | 25 |
| $0.98491 < X \leq 0.98575$ | 17 |
| $0.98575 < X \leq 0.98659$ | 12 |
| $0.98659 < X \leq 0.98743$ | 4 |
| $0.98743 < X \leq 0.98827$ | 1 |
| $0.98827 < X$ | 2 |

**Table 3:** Set One: Frequency table

### B - Set Two: Frequency table

| Value Range | Count |
|---|---|
| $0.00000 \leq X \leq 8375.00$ | 91 |
| $8375.00 < X \leq 16750.0$ | 10 |
| $16750.0 < X \leq 25125.4$ | 5 |
| $25125.4 < X \leq 33500.2$ | 5 |
| $33500.2 < X \leq 41875.0$ | 3 |
| $41875.0 < X \leq 50249.8$ | 2 |
| $50249.8 < X \leq 58624.6$ | 1 |
| $58624.6 < X \leq 66999.4$ | 0 |
| $66999.4 < X \leq 75374.2$ | 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 3:** Set One: Fluke 289 w/ resistors

## REFERENCES

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