An Incomplete Solutions Guide to the NIST/SEMATECH e-Handbook of Statistical Methods

examples and case studies using the tidyverse and ggplot2

Ray Hoobler

2018-04-19

Contents

Preface

Exploratory Data Analysis (EDA) is a philosophy on how to work with data, and for many applications, the workflow is better suited for most working scientist and engineers. As a scientist, we are trained to formulate a hypothesis and design a series of experiments that will allow us to test the hypothesis effectively. Unfortunately, most data doesn't from carefully controlled trials, but from observations. Statisticians will readily jump into describing the difference in as much detail as you would like.

For most of us, we need tools to characterize an instrument or a process. The philosophy of EDA provides the framework to do this work.

Unfortunately, most textbooks still focus on traditional statistical techniques and even while it is essential to understand the underlying assumptions and fundamentals, I would argue that most of the work we do as scientist and engineers are not well suited for rigorous statistical analysis. In many cases, the need to disseminate information to a broad audience is best served by the methods espoused by EDA. The NIST e-Handbook Engineering Statistics is a welcome deviation from the norm.

In the Spring of 2018, I adopted this text as the basis of a one-semester, graduate course that focused applied statistical techniques. The audience for this course were working scientist, and the course was a core course in a Professional Science Master's (PSM).

Unfortunately, the one drawback of the NIST Handbook is the use of Dataplot as the primary software package for analysis. The authors have provided examples using the R statistical language; however, most–if not all–of these scripts are written using base R which is unfortunate. Modern R now incorporates many packages for streamlining the EDA process. This book attempts to capture my efforts to use these methods and share them with students in the course. The two packages that I primarily used were **tidyverse** and **ggplot2**.

Before going further, I should clarify one thing—I'm a hack. I classify learning as three levels: novice, hack, expert.

Novice: basic knowledge of how to use a tool with a desire to learn. Hack: Basic to intermediate knowledge of how to use a tool accompanied by resources to produce a finished product. Expert: Extensive knowledge of how to use a tool; can produce a finished product with few outside resources.

I'm sure other factors can be added to each category, but these capture the spirit of how I approach learning.

The number of resources available to learn R is numerous, and the first I would strongly recommend is R for Data Science. This text is an introduction to the tidyverse. The tidyverse is not just a collection of R packages, but a philosophy on how to work with data. It makes data analysis almost fun!

The other primary resource available for EDA is ggplot2. Like the tidyverse, ggplot2 is not just a package of tools, but a philosophy built around the Grammar of Graphics.

I encourage the reader to explore the references related to these two packages and their underlying design philosophies.

This book will show how I have worked through the exercises and case studies presented in the NIST handbook using methods found in the tidyverse and ggplot2. I have found this framework to be incredibly satisfying and one I was eager to share beyond my class.

If you find this material useful, please send me an email.

Structure of the book

Content was built around the e-book NIST/SEMATECH e-Handbook of Statistical Methods.

At the begining of each exercise or case study, I've included a link back to the specific page of the e-Handbook. The e-Handbook can be downloaded in full from the NIST site. The compressed file is over 100Mb (not 43Mb) as stated.

Software information and conventsions

Follow "best practices" of the *tityverse*

The R session information for this book is shown below:

```
sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.4
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] magrittr_1.5    bindrcpp_0.2.2  forcats_0.3.0   stringr_1.3.0
##  [5] dplyr_0.7.4     purrr_0.2.4     readr_1.1.1     tidyr_0.8.0
##  [9] tibble_1.4.2    ggplot2_2.2.1   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.4 xfun_0.1         reshape2_1.4.3   haven_1.1.1
##  [5] lattice_0.20-35  colorspace_1.3-2 htmltools_0.3.6  yaml_2.1.18
##  [9] utf8_1.1.3       rlang_0.2.0      pillar_1.2.1     foreign_0.8-69
## [13] glue_1.2.0       modelr_0.1.1     readxl_1.0.0     bindr_0.1.1
## [17] plyr_1.8.4       munsell_0.4.3    gtable_0.2.0     cellranger_1.1.0
## [21] rvest_0.3.2      psych_1.8.3.3    evaluate_0.10.1  labeling_0.3
## [25] knitr_1.20       parallel_3.4.4   highr_0.6        broom_0.4.4
## [29] Rcpp_0.12.16     backports_1.1.2  scales_0.5.0     jsonlite_1.5
## [33] mnormt_1.5-5     hms_0.4.2        digest_0.6.15    stringi_1.1.7
## [37] bookdown_0.7     grid_3.4.4       rprojroot_1.3-2  cli_1.0.0
## [41] tools_3.4.4      lazyeval_0.2.1   crayon_1.3.4     pkgconfig_2.0.1
## [45] xml2_1.2.0       lubridate_1.7.4  assertthat_0.2.0 rmarkdown_1.9
## [49] httr_1.3.1       rstudioapi_0.7   R6_2.2.2         nlme_3.1-137
## [53] compiler_3.4.4
```

Acknowledgements

Current and former students

Former coleagues

Ray James Hoobler Salt Lake City, Utah

0.1 Simple Programing Examples

```
x_cube <- function (x) {
  x^3
}

x_cube(3)
```

```
## [1] 27
```

```
library(tidyverse)
set.seed(5234)
n_darts <- 1e6
x_test <- runif(n_darts, -1, 1)
y_test <- runif(n_darts, -1, 1)

random_xy_table <- tibble(n = 1:n_darts, x = x_test, y = y_test) %>%
  mutate(r = sqrt(x^2 + y^2),
  dart_circle = case_when(
    r <= 1 ~ 1,
    r > 1 ~ 0
  )
  )

random_xy_table
```

```
## # A tibble: 1,000,000 x 5
##         n       x      y     r dart_circle
##     <int>   <dbl>  <dbl> <dbl>       <dbl>
##   1     1  0.739  -0.999 1.24           0.
##   2     2  0.385  -0.560 0.680          1.
##   3     3 -0.448  -0.627 0.771          1.
##   4     4  0.446   0.740 0.864          1.
##   5     5  0.0918 -0.521 0.529          1.
##   6     6  0.848   0.484 0.977          1.
##   7     7 -0.623   0.443 0.764          1.
##   8     8 -0.379   0.319 0.496          1.
##   9     9  0.303  -0.694 0.757          1.
## 10    10  0.863   0.966 1.30           0.
## # ... with 999,990 more rows
```

```
pi_results <- random_xy_table %>%
  summarise(pi_est = mean(dart_circle)*4) %>%
  mutate(pi, accuracy = (pi_est - pi)/pi * 100)

pi_results
```

```
## # A tibble: 1 x 3
##   pi_est    pi accuracy
##    <dbl> <dbl>    <dbl>
## 1   3.14  3.14  0.00609
```

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN:

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 1

Exploratory Data Analysis

1.1 A EDA Example

An EDA/Graphics Example

The Anscombe dataset is an excelent place to start as it will allow us to start using R immediately. The anscombe dataset is part of the **datasets** package and is automatically loaded with RStudio.

```
anscombe
```

```
##    x1 x2 x3 x4    y1   y2    y3    y4
## 1  10 10 10  8  8.04 9.14  7.46  6.58
## 2   8  8  8  8  6.95 8.14  6.77  5.76
## 3  13 13 13  8  7.58 8.74 12.74  7.71
## 4   9  9  9  8  8.81 8.77  7.11  8.84
## 5  11 11 11  8  8.33 9.26  7.81  8.47
## 6  14 14 14  8  9.96 8.10  8.84  7.04
## 7   6  6  6  8  7.24 6.13  6.08  5.25
## 8   4  4  4 19  4.26 3.10  5.39 12.50
## 9  12 12 12  8 10.84 9.13  8.15  5.56
## 10  7  7  7  8  4.82 7.26  6.42  7.91
## 11  5  5  5  8  5.68 4.74  5.73  6.89
```

1.2 But first... let's start working in the tidyverse

The tidyverse is discribed as

> an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

You can install the **tidyverse** package with

```
install.packages("tidyverse")
```

Once installed, simply load the package:

```
library(tidyverse)
```

Additional details can be found at tidyverse.org

If you have only created charts and graphs using spreadsheets, you will assume the data is ready to plot. It might be nice to have the x1 and y1 values closer together in the table, but we could still select the individual columns and plot the datasets.

We're going to jump right in with with the idea of *tidy data*. That each row should be a single observation.

As mentioned in the introduction, this text assumes a basic knowledge of the tidyverse. In this example, we will selct the x data from the data frame, rename the colunn labels, use the `gather()` function to tidy the data. We will then repeat the process for the y data, removing the group names from the data set. The last step is to combine these two data frames into a single data frame we will use for plotting. I'm sure there are more efficient ways to do this; however, the code used to do this manipulation is typical when working with non-tidy data. An added benifit is that hte code is readable.

```
x_anscombe <- anscombe %>%  # results will be storred into a new object x_anscombe; we start with the o
  select(x1, x2, x3, x4) %>%  # select the columns we want to work with
  rename(group1 = x1, group2 = x2, group3 = x3, group4 = x4) %>% # rename the values using a generic he
  gather(key = group, value = x_values, group1, group2, group3, group4) # gather the columns into rows

x_anscombe
```

```
##     group x_values
## 1  group1       10
## 2  group1        8
## 3  group1       13
## 4  group1        9
## 5  group1       11
## 6  group1       14
## 7  group1        6
## 8  group1        4
## 9  group1       12
## 10 group1        7
## 11 group1        5
## 12 group2       10
## 13 group2        8
## 14 group2       13
## 15 group2        9
## 16 group2       11
## 17 group2       14
## 18 group2        6
## 19 group2        4
## 20 group2       12
## 21 group2        7
## 22 group2        5
## 23 group3       10
## 24 group3        8
## 25 group3       13
## 26 group3        9
## 27 group3       11
## 28 group3       14
## 29 group3        6
## 30 group3        4
## 31 group3       12
## 32 group3        7
## 33 group3        5
## 34 group4        8
## 35 group4        8
## 36 group4        8
## 37 group4        8
## 38 group4        8
## 39 group4        8
## 40 group4        8
## 41 group4       19
## 42 group4        8
## 43 group4        8
## 44 group4        8
```

```
y_anscombe <- anscombe %>%
  select(y1, y2, y3, y4) %>%
  gather(key = group, value = y_values, y1, y2, y3, y4) %>% # I don't need to rename the columns as I w
  select(y_values)

y_anscombe
```

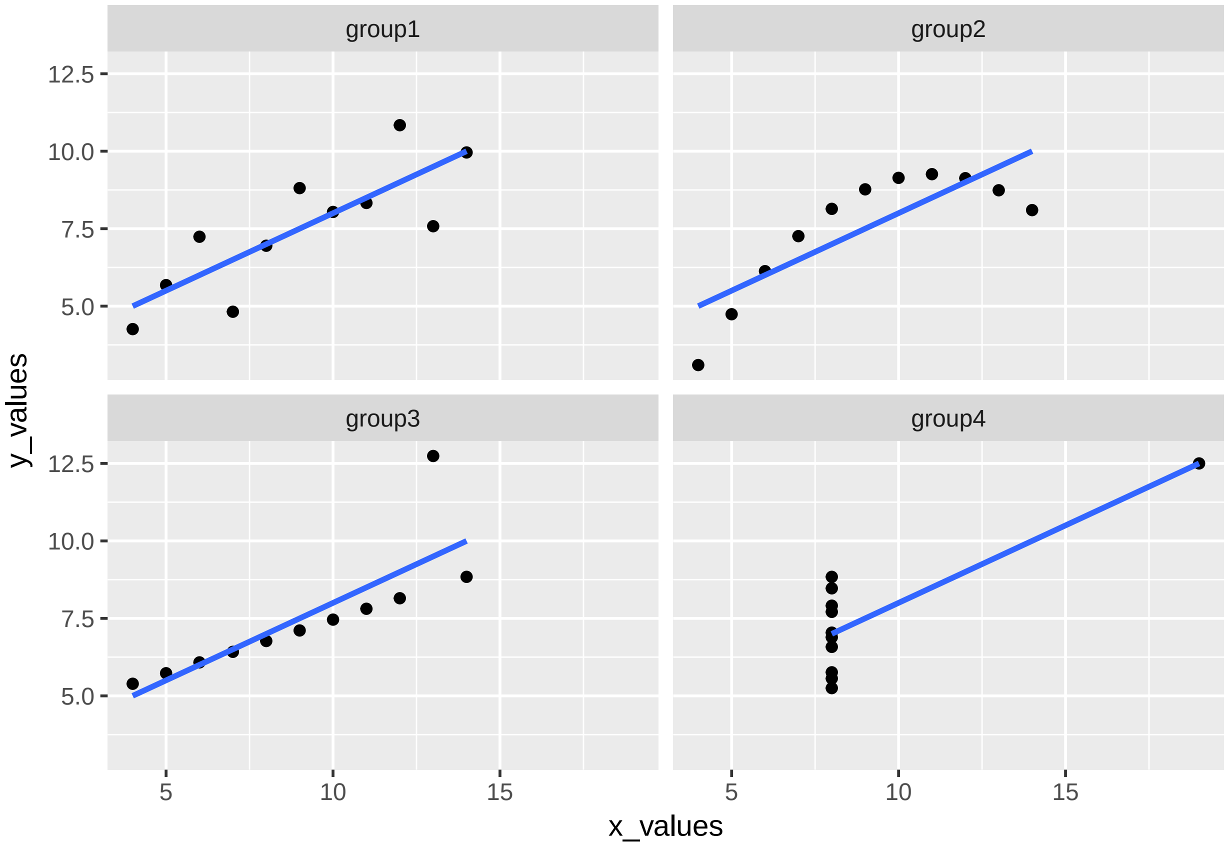
```
##    y_values
## 1      8.04
## 2      6.95
## 3      7.58
## 4      8.81
## 5      8.33
## 6      9.96
## 7      7.24
## 8      4.26
## 9     10.84
## 10     4.82
## 11     5.68
## 12     9.14
## 13     8.14
## 14     8.74
## 15     8.77
## 16     9.26
## 17     8.10
## 18     6.13
## 19     3.10
## 20     9.13
## 21     7.26
## 22     4.74
## 23     7.46
## 24     6.77
## 25    12.74
## 26     7.11
## 27     7.81
## 28     8.84
## 29     6.08
## 30     5.39
## 31     8.15
## 32     6.42
## 33     5.73
## 34     6.58
## 35     5.76
## 36     7.71
## 37     8.84
## 38     8.47
## 39     7.04
## 40     5.25
## 41    12.50
## 42     5.56
## 43     7.91
## 44     6.89
```

```
anscombe_tidy <- bind_cols(x_anscombe, y_anscombe)
anscombe_tidy
```

```
##     group x_values y_values
## 1  group1       10     8.04
## 2  group1        8     6.95
## 3  group1       13     7.58
## 4  group1        9     8.81
## 5  group1       11     8.33
## 6  group1       14     9.96
## 7  group1        6     7.24
## 8  group1        4     4.26
## 9  group1       12    10.84
## 10 group1        7     4.82
## 11 group1        5     5.68
## 12 group2       10     9.14
## 13 group2        8     8.14
## 14 group2       13     8.74
## 15 group2        9     8.77
## 16 group2       11     9.26
## 17 group2       14     8.10
## 18 group2        6     6.13
## 19 group2        4     3.10
## 20 group2       12     9.13
## 21 group2        7     7.26
## 22 group2        5     4.74
## 23 group3       10     7.46
## 24 group3        8     6.77
## 25 group3       13    12.74
## 26 group3        9     7.11
## 27 group3       11     7.81
## 28 group3       14     8.84
## 29 group3        6     6.08
## 30 group3        4     5.39
## 31 group3       12     8.15
## 32 group3        7     6.42
## 33 group3        5     5.73
## 34 group4        8     6.58
## 35 group4        8     5.76
## 36 group4        8     7.71
## 37 group4        8     8.84
## 38 group4        8     8.47
## 39 group4        8     7.04
## 40 group4        8     5.25
## 41 group4       19    12.50
## 42 group4        8     5.56
## 43 group4        8     7.91
## 44 group4        8     6.89
```

While this may seem like a lot of work to make a new table—which is much harder to read—this method allows us to exploit the **gramar of graphics** used by the **ggplot2** package.

```
ggplot(anscombe_tidy, aes(x_values, y_values)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~group)
```



It may not be immediately obvious from the plots, but the slope and intercept for each line are identical. We can calculate these values for each dataset using the linear model function, `lm()`.

```
lm(y1 ~ x1, data = anscombe)
```

```
##
## Call:
## lm(formula = y1 ~ x1, data = anscombe)
##
## Coefficients:
## (Intercept)           x1
##      3.0001       0.5001
```

```
lm(y2 ~ x2, data = anscombe)
```

```
##
## Call:
## lm(formula = y2 ~ x2, data = anscombe)
##
## Coefficients:
```

```
## (Intercept)           x2
##       3.001        0.500
```

```
lm(y3 ~ x3, data = anscombe)
```

```
##
## Call:
## lm(formula = y3 ~ x3, data = anscombe)
##
## Coefficients:
## (Intercept)           x3
##      3.0025       0.4997
```

```
lm(y4 ~ x4, data = anscombe)
```

```
##
## Call:
## lm(formula = y4 ~ x4, data = anscombe)
##
## Coefficients:
## (Intercept)           x4
##      3.0017       0.4999
```

The calculated slope and intercept are the same (at least to three significant figures); the use of EDA allows us to differentiate the data quickly.

1.3 Common graphical analysis used in the e-Handbook

Four techniques are routinely used in the e-Handbook for preliminary EDA. These four charts are routinely displayed as a "4-plot." Each technique will be presented in the following sub-sections.

- Run sequence plot
- Lag plot
- Histogram
- Normal probility plot

1.4 Case studies from chapter 1 of the NIST/SEMATECH e-Handbook

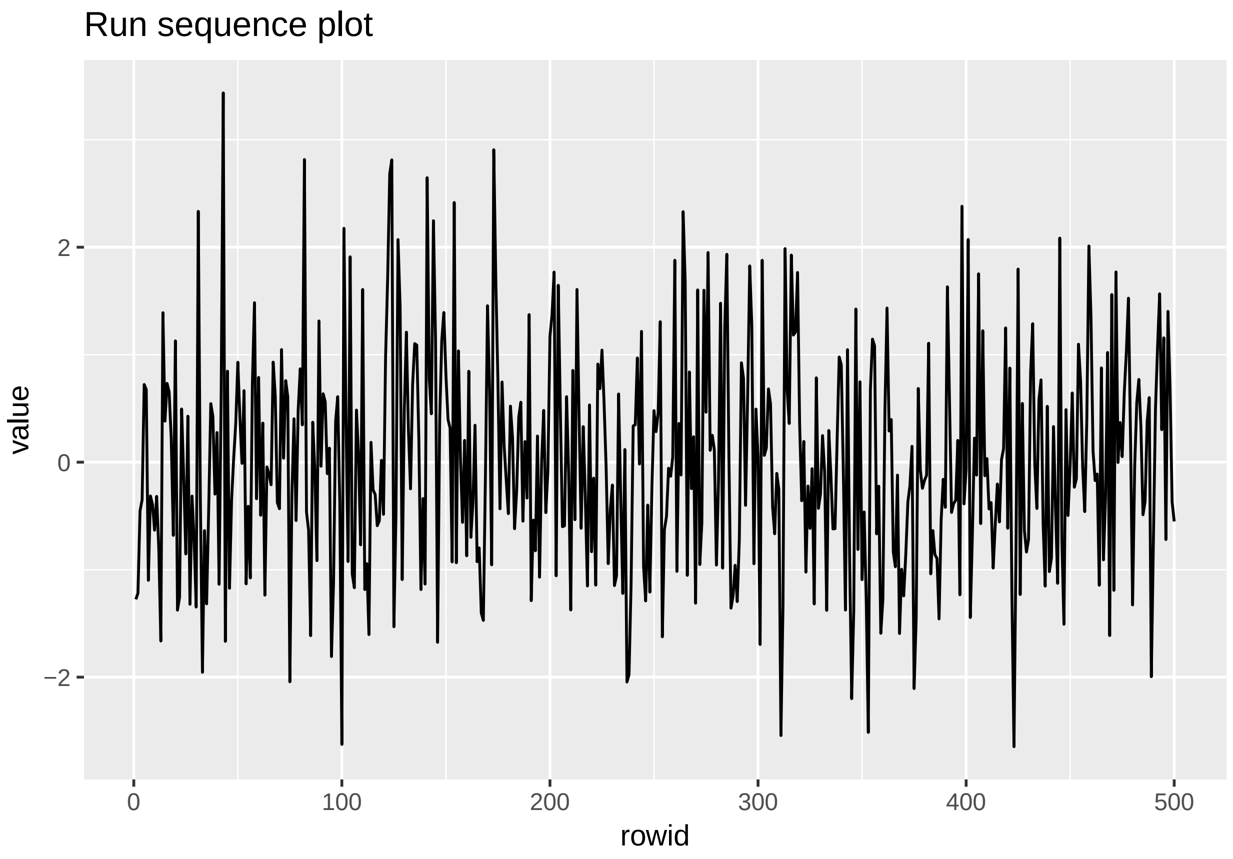
1.4.1 Normal random numbers

Normal Random Numbers

```
normal_random_numbers <- scan("NIST data/RANDN.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

normal_random_numbers
```

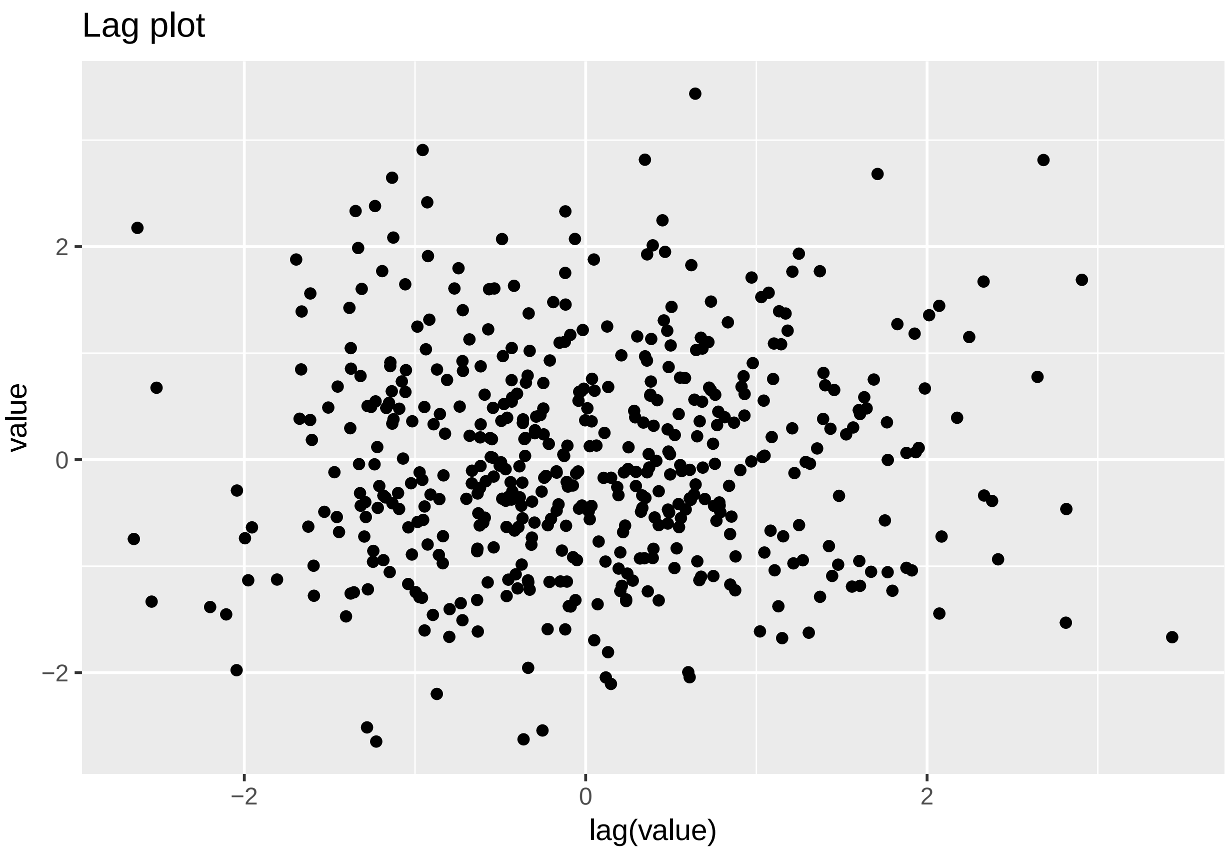
```
## # A tibble: 500 x 2
##    rowid  value
##    <int>  <dbl>
##  1     1 -1.28
##  2     2 -1.22
##  3     3 -0.453
##  4     4 -0.350
##  5     5  0.723
##  6     6  0.676
##  7     7 -1.10
##  8     8 -0.314
##  9     9 -0.394
## 10    10 -0.633
## # ... with 490 more rows
```

```
ggplot(normal_random_numbers, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



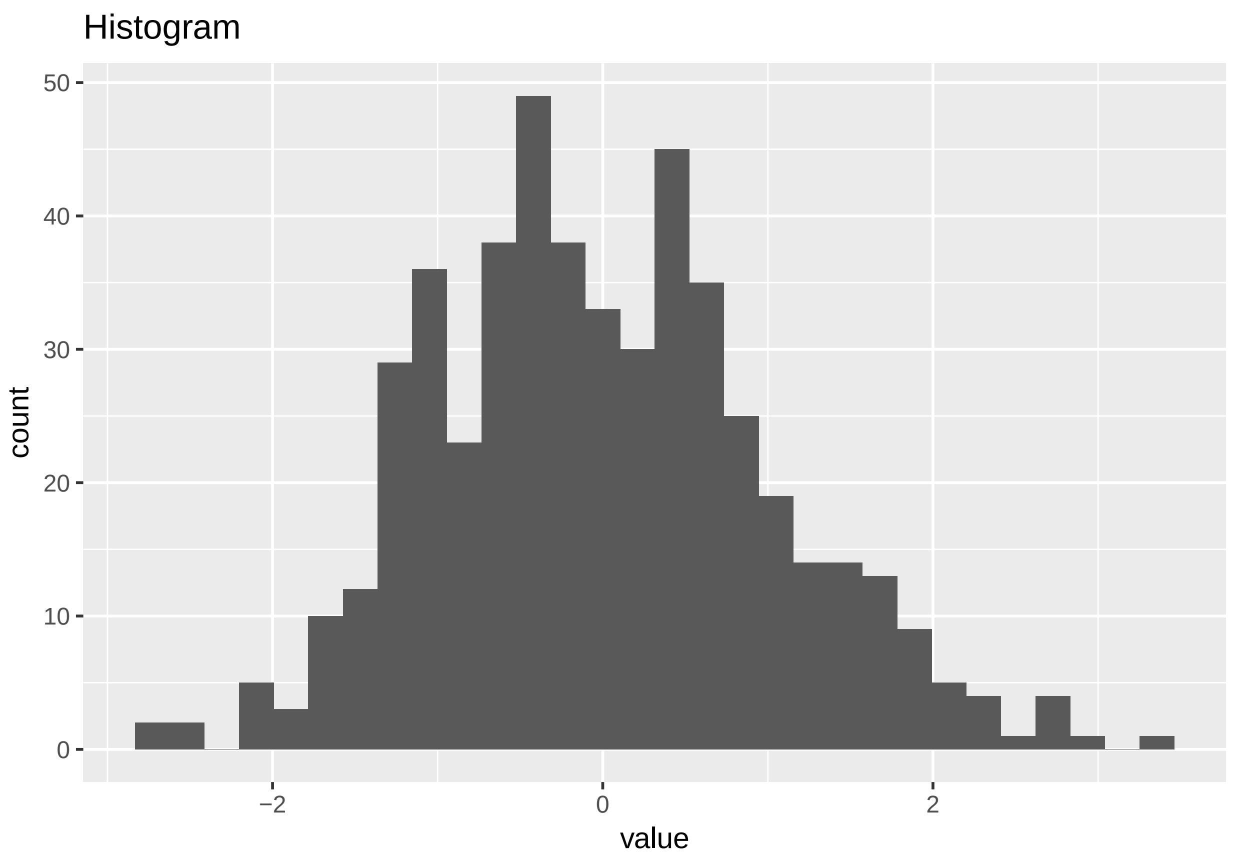
```
ggplot(normal_random_numbers, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

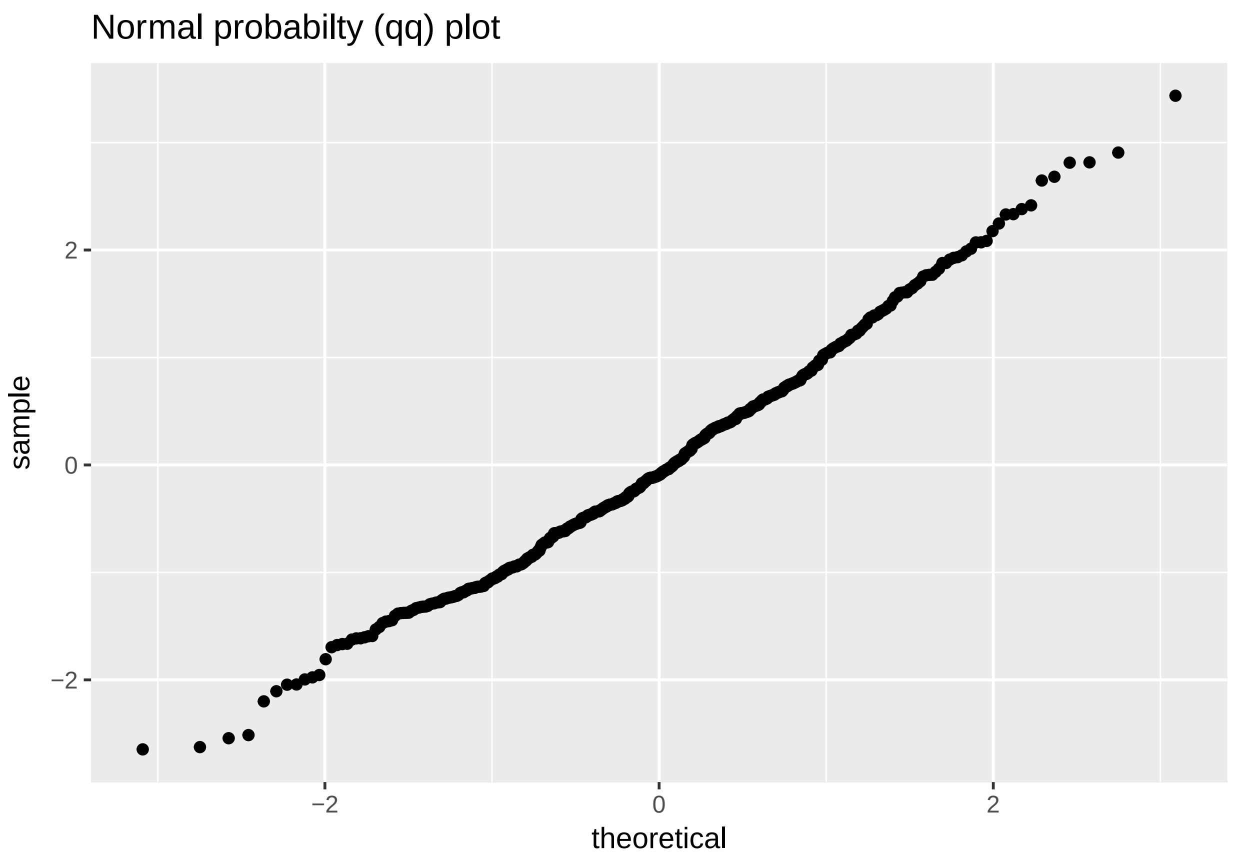


```
ggplot(normal_random_numbers, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(normal_random_numbers, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



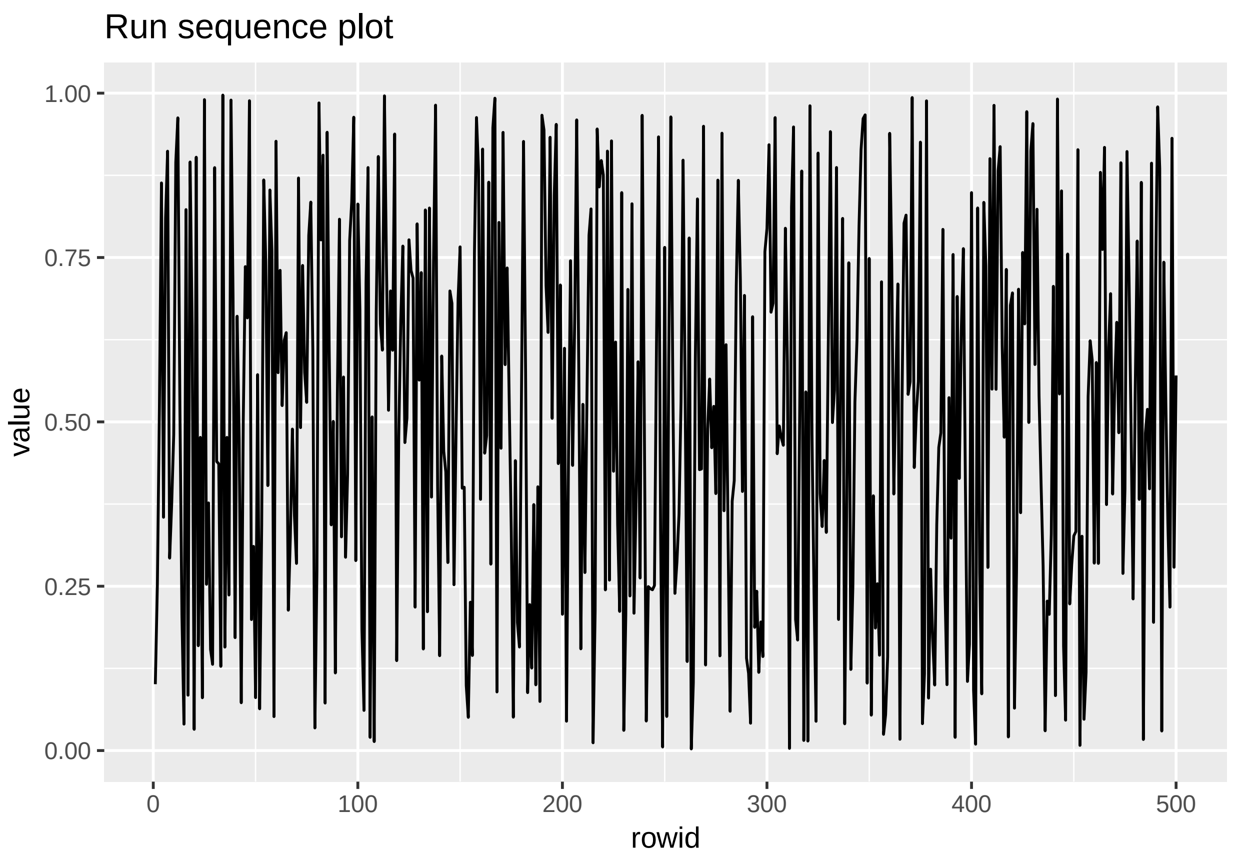
1.4.2 Uniform random numbers

Uniform Random Numbers

```r
uniform_random_numbers <- scan("NIST data/RANDU.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

uniform_random_numbers
```

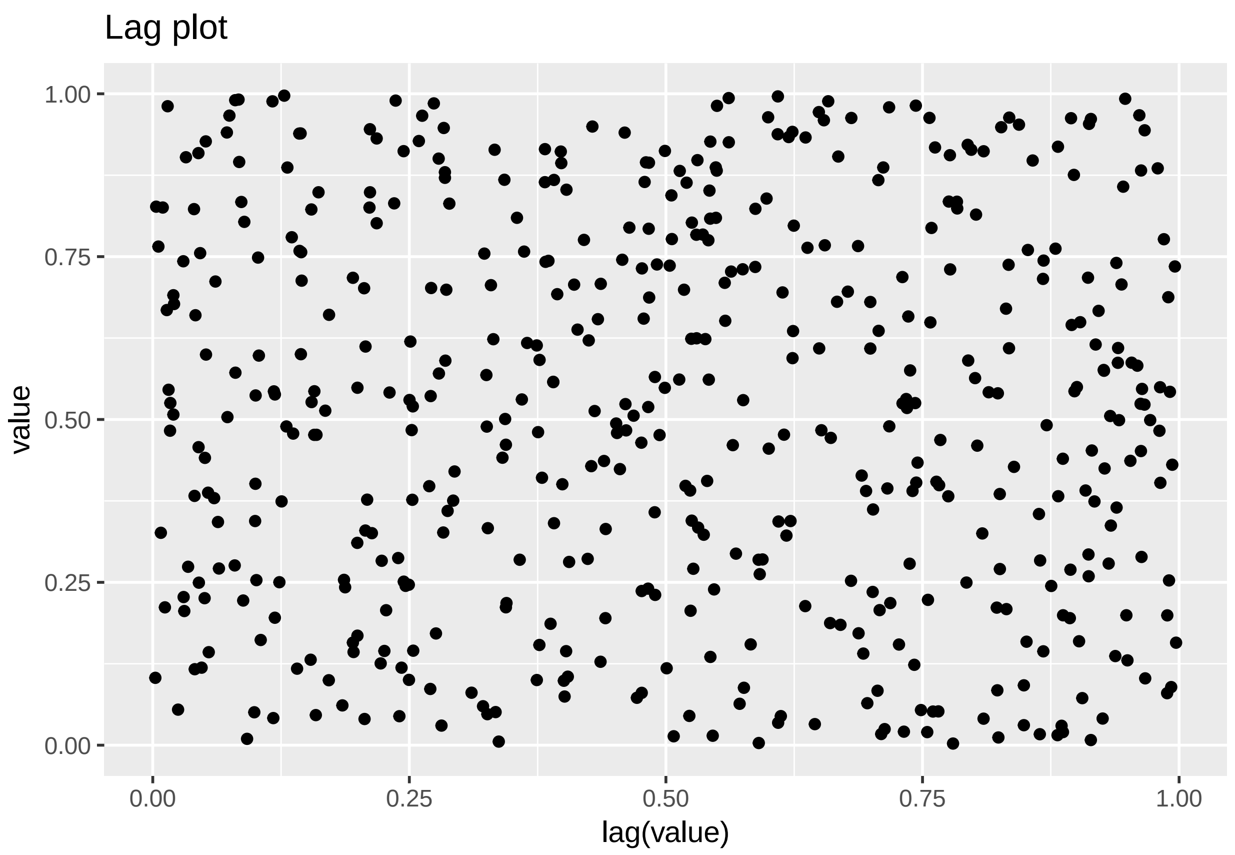
```
## # A tibble: 500 x 2
##    rowid value
##    <int> <dbl>
##  1     1 0.101
##  2     2 0.253
##  3     3 0.520
##  4     4 0.863
##  5     5 0.355
##  6     6 0.810
##  7     7 0.912
##  8     8 0.293
##  9     9 0.375
## 10    10 0.481
## # ... with 490 more rows
```

```r
ggplot(uniform_random_numbers, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



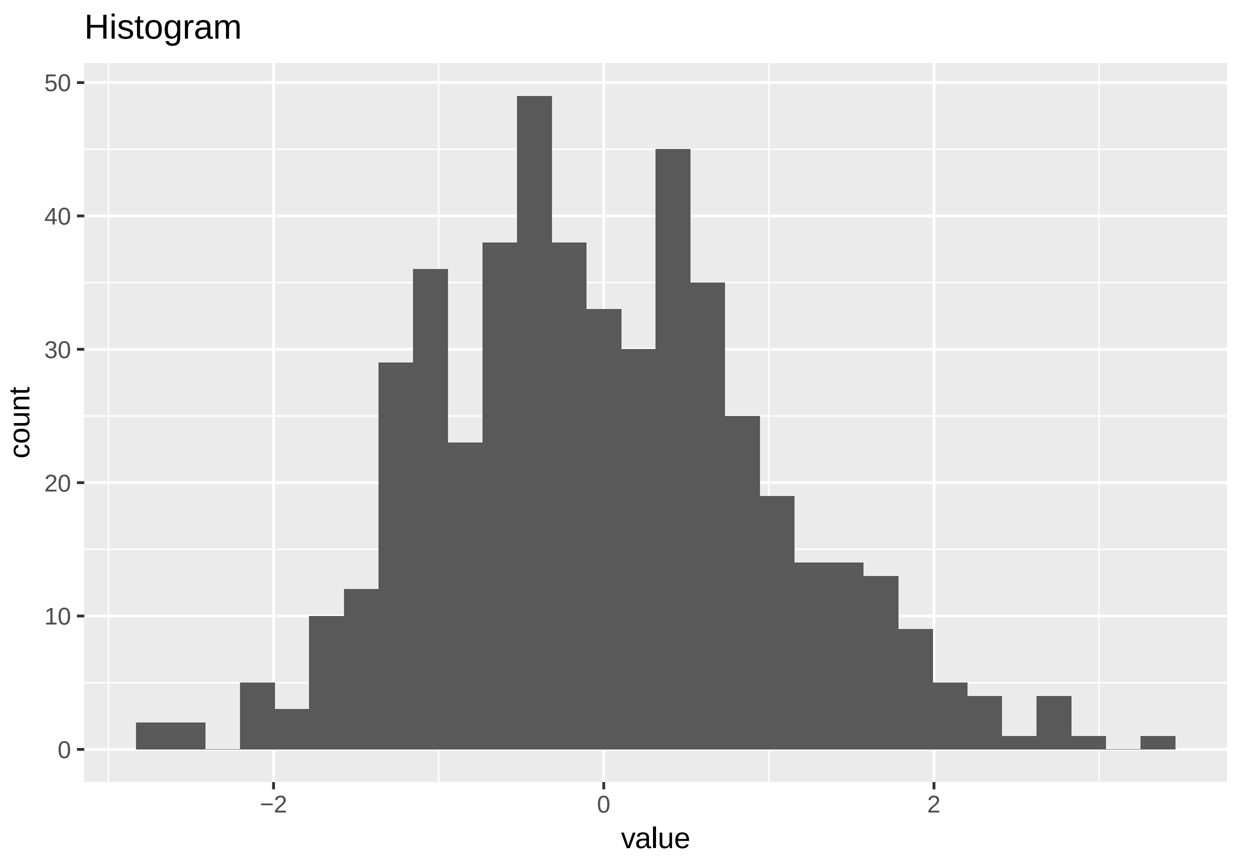
```
ggplot(uniform_random_numbers, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

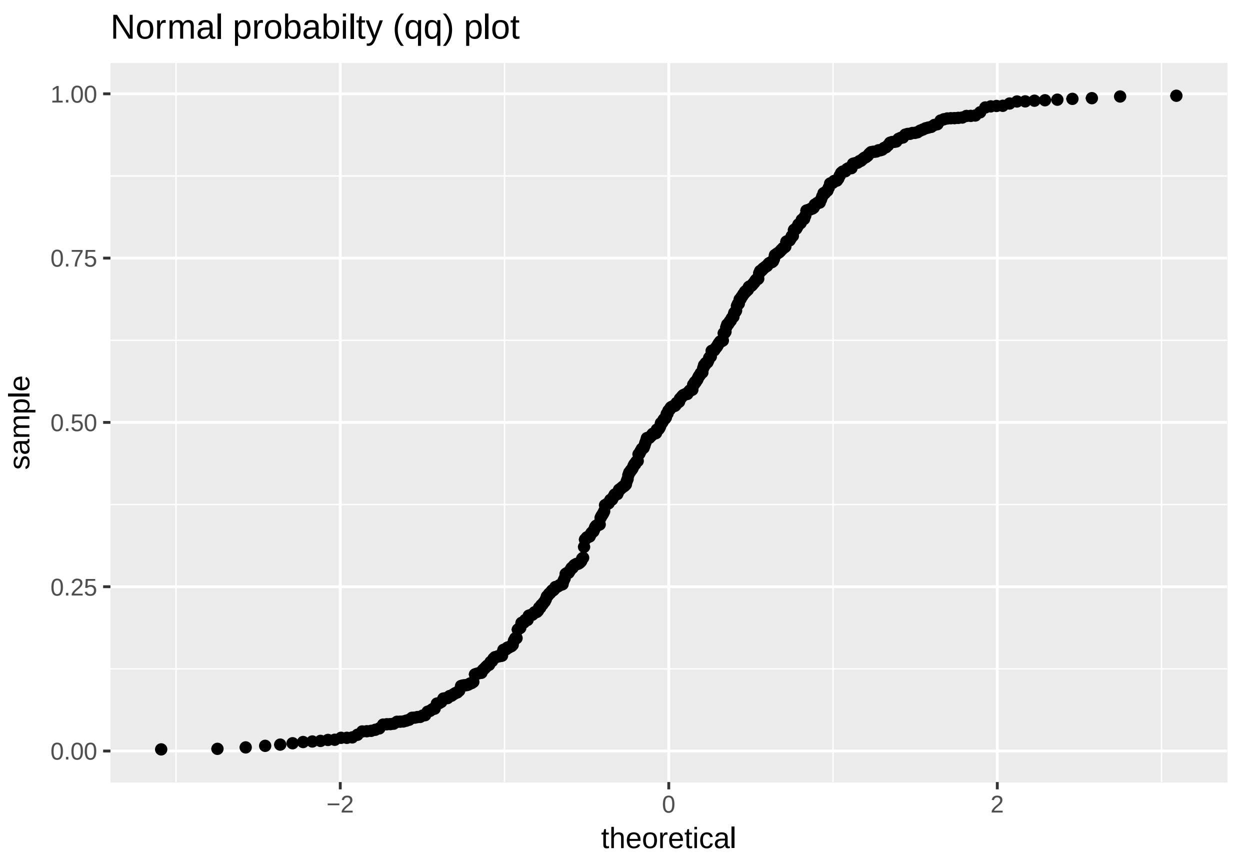


```
ggplot(normal_random_numbers, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(uniform_random_numbers, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



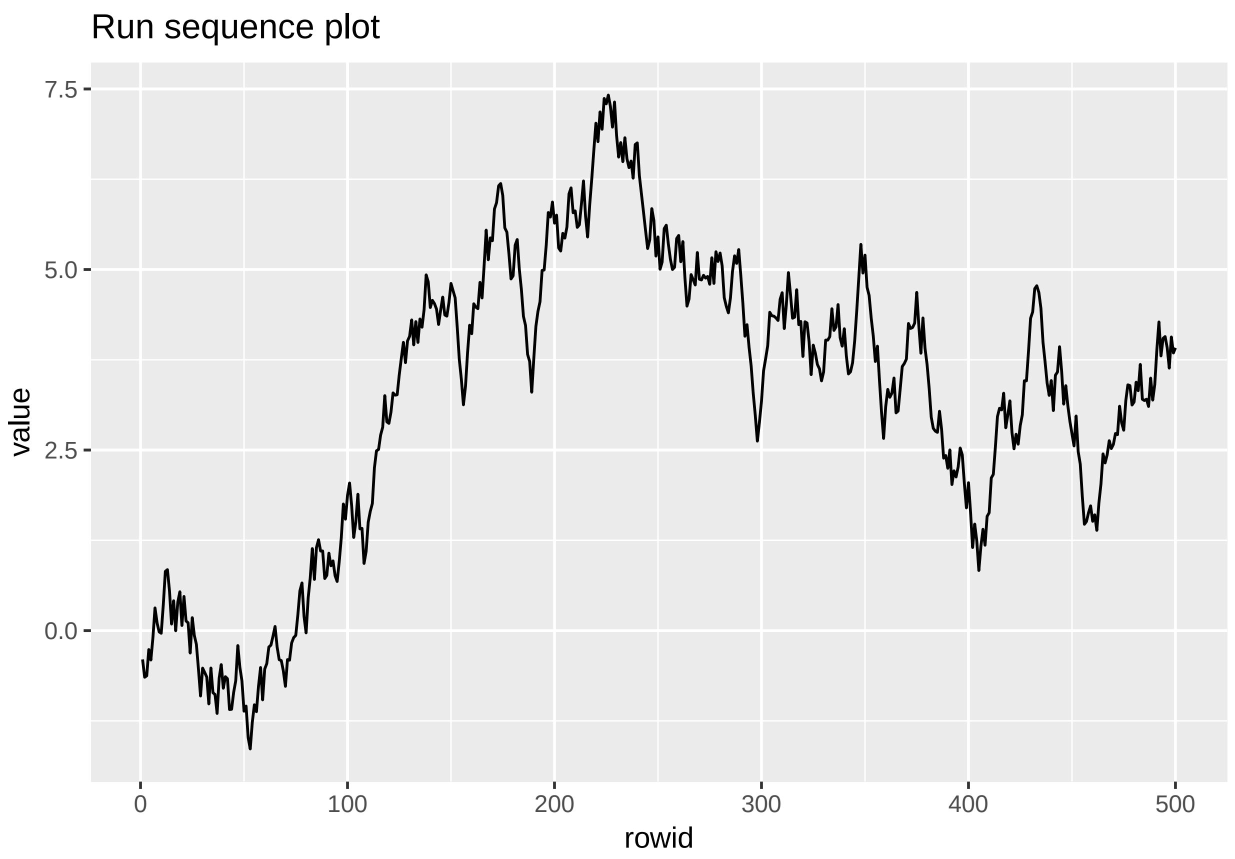
1.4.3 Random walk

Random Walk

```
random_walk <- scan("NIST data/RANDWALK.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

random_walk
```

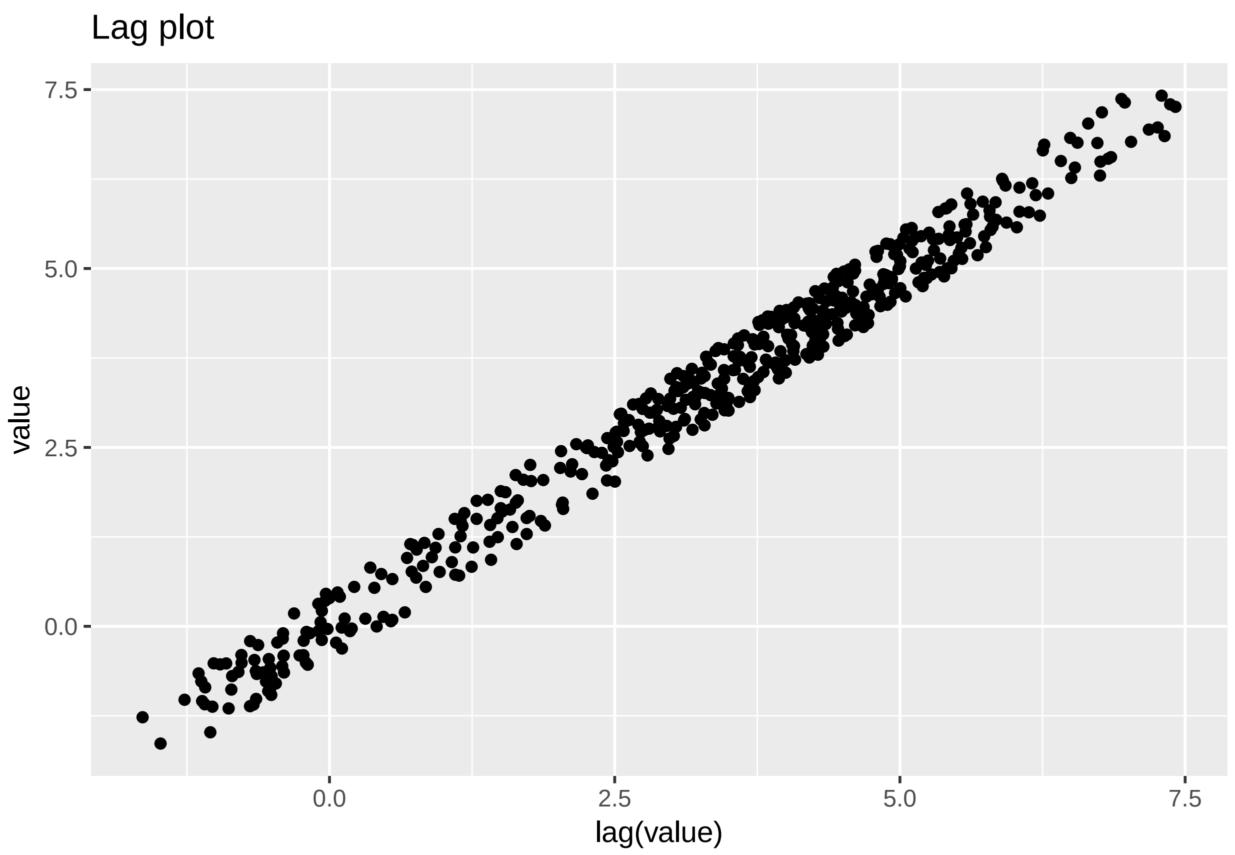
```
## # A tibble: 500 x 2
##    rowid   value
##    <int>   <dbl>
##  1     1 -0.399
##  2     2 -0.646
##  3     3 -0.626
##  4     4 -0.262
##  5     5 -0.407
##  6     6 -0.0976
##  7     7  0.314
##  8     8  0.107
##  9     9 -0.0177
## 10    10 -0.0371
## # ... with 490 more rows
```

```
ggplot(random_walk, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



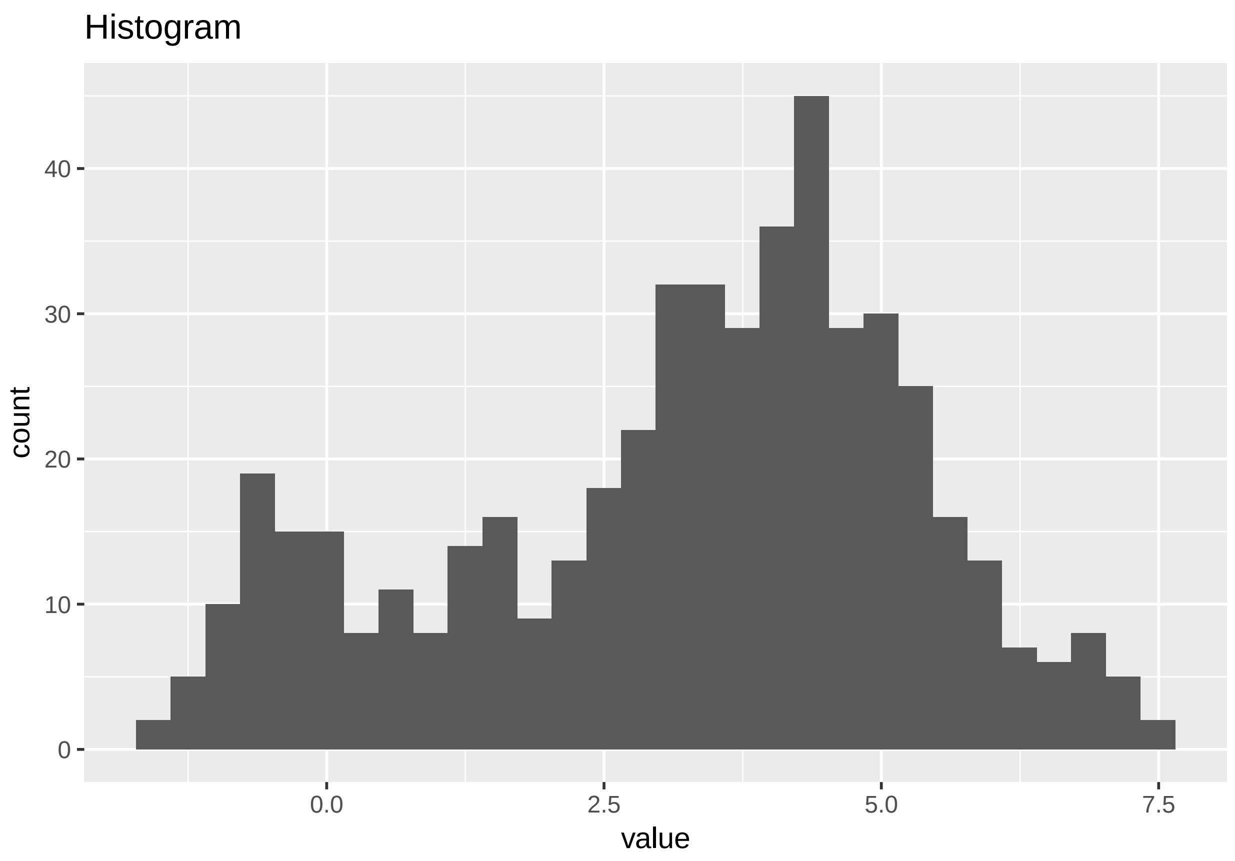
```
ggplot(random_walk, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

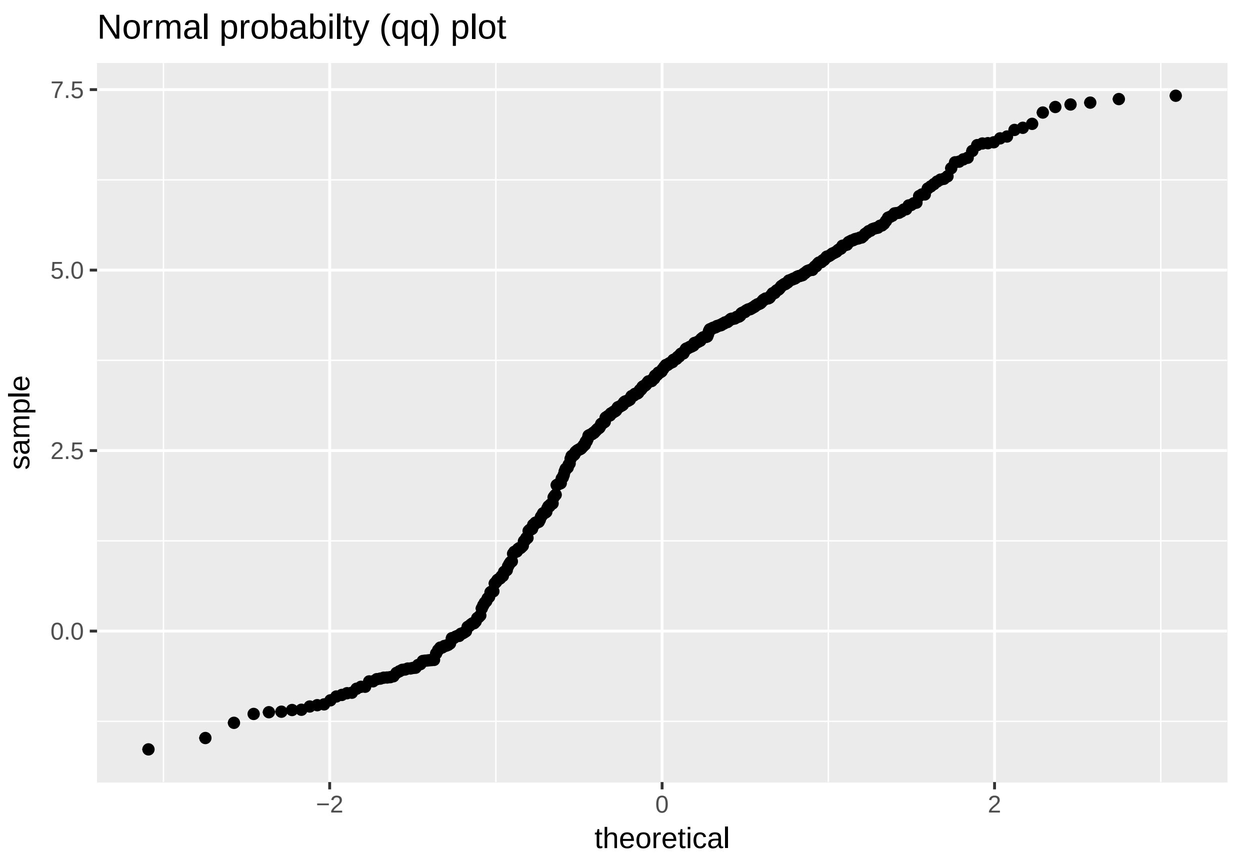


```r
ggplot(random_walk, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(random_walk, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



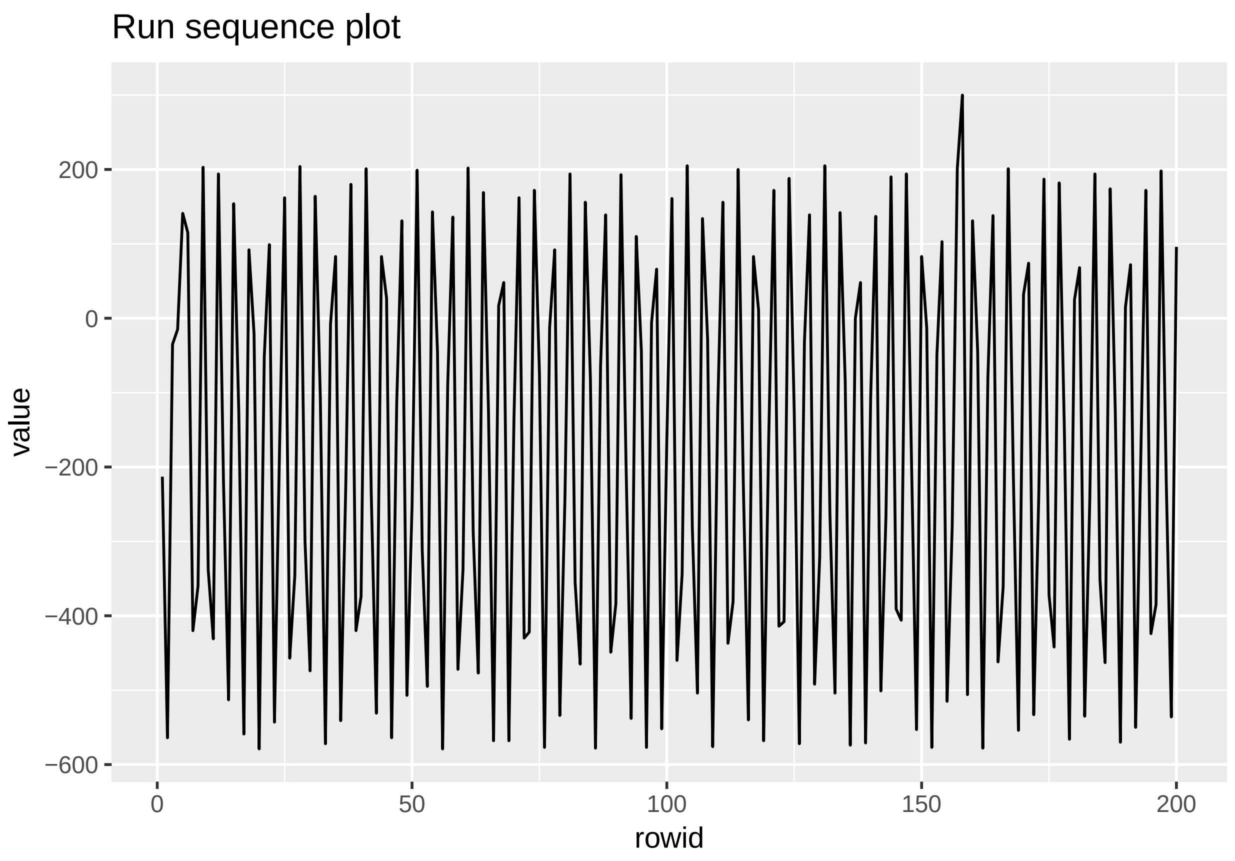
1.4.4 Beam deflections

Beam Deflections

```
beam_deflections <- scan("NIST data/LEW.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

beam_deflections
```

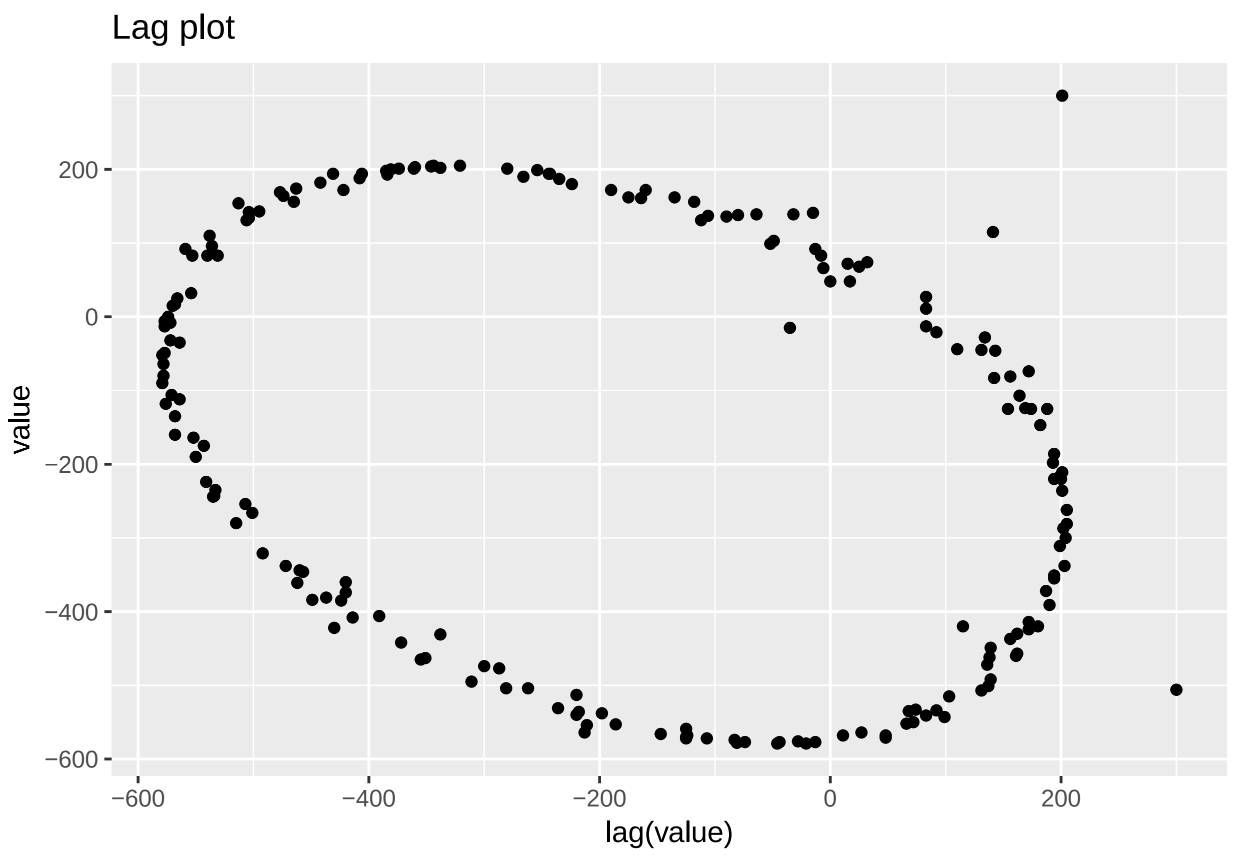
```
## # A tibble: 200 x 2
##    rowid value
##    <int> <dbl>
##  1     1 -213.
##  2     2 -564.
##  3     3  -35.
##  4     4  -15.
##  5     5  141.
##  6     6  115.
##  7     7 -420.
##  8     8 -360.
##  9     9  203.
## 10    10 -338.
## # ... with 190 more rows
```

```
ggplot(beam_deflections, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



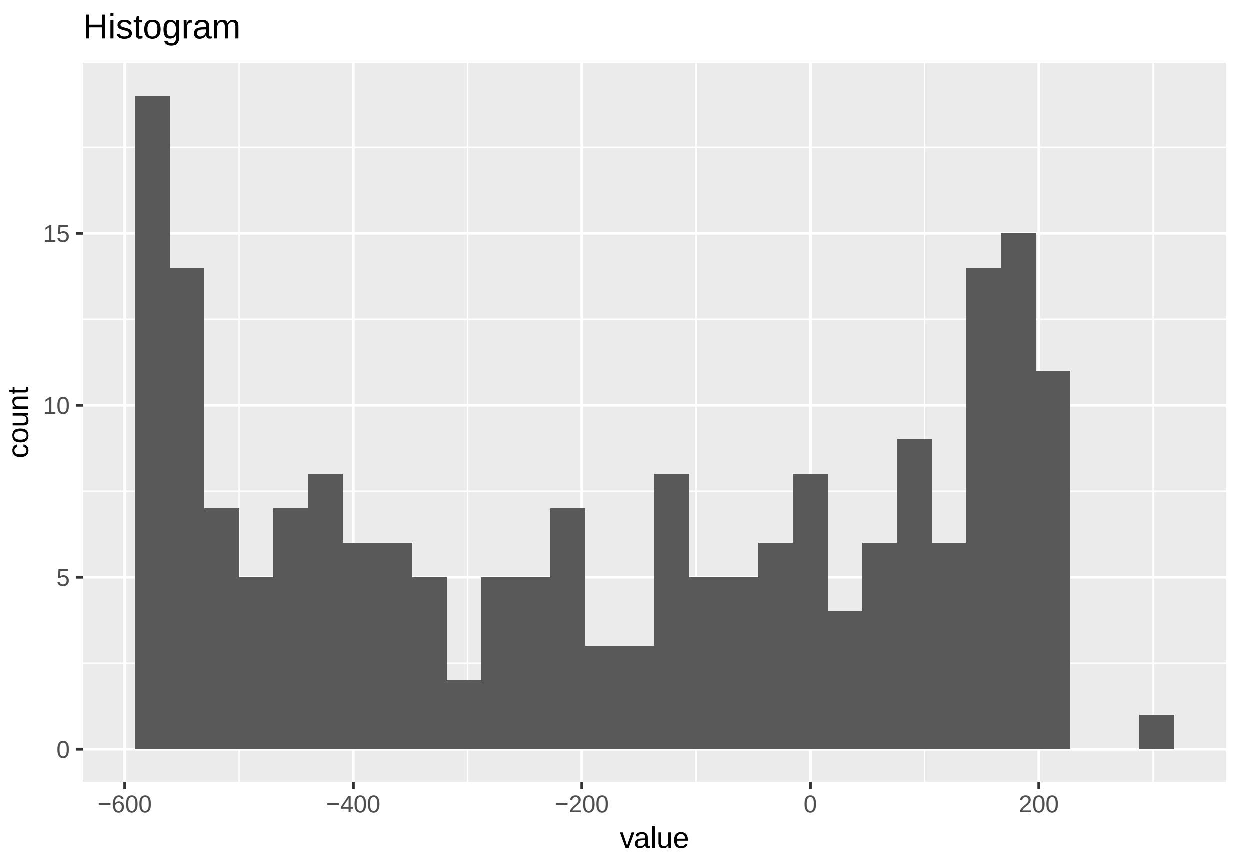
```
ggplot(beam_deflections, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

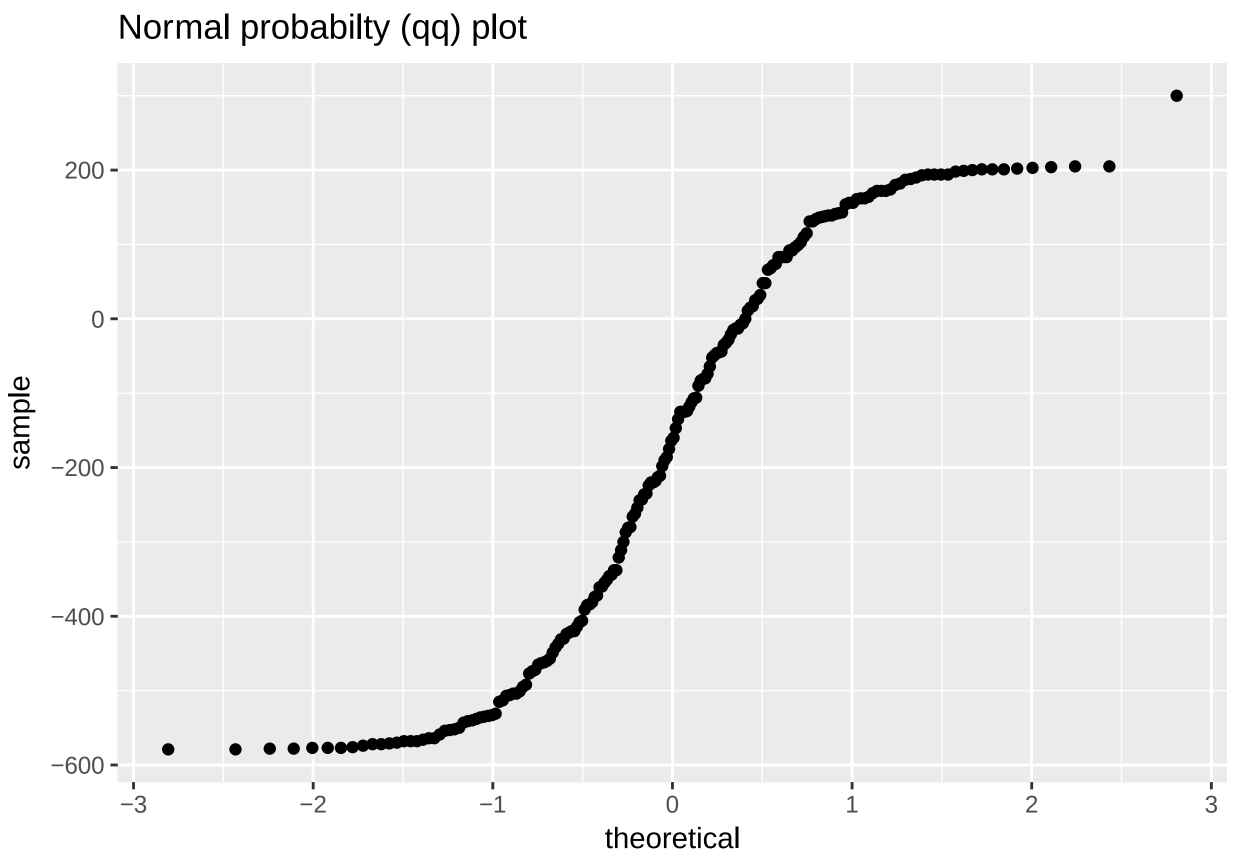


```
ggplot(beam_deflections, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(beam_deflections, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



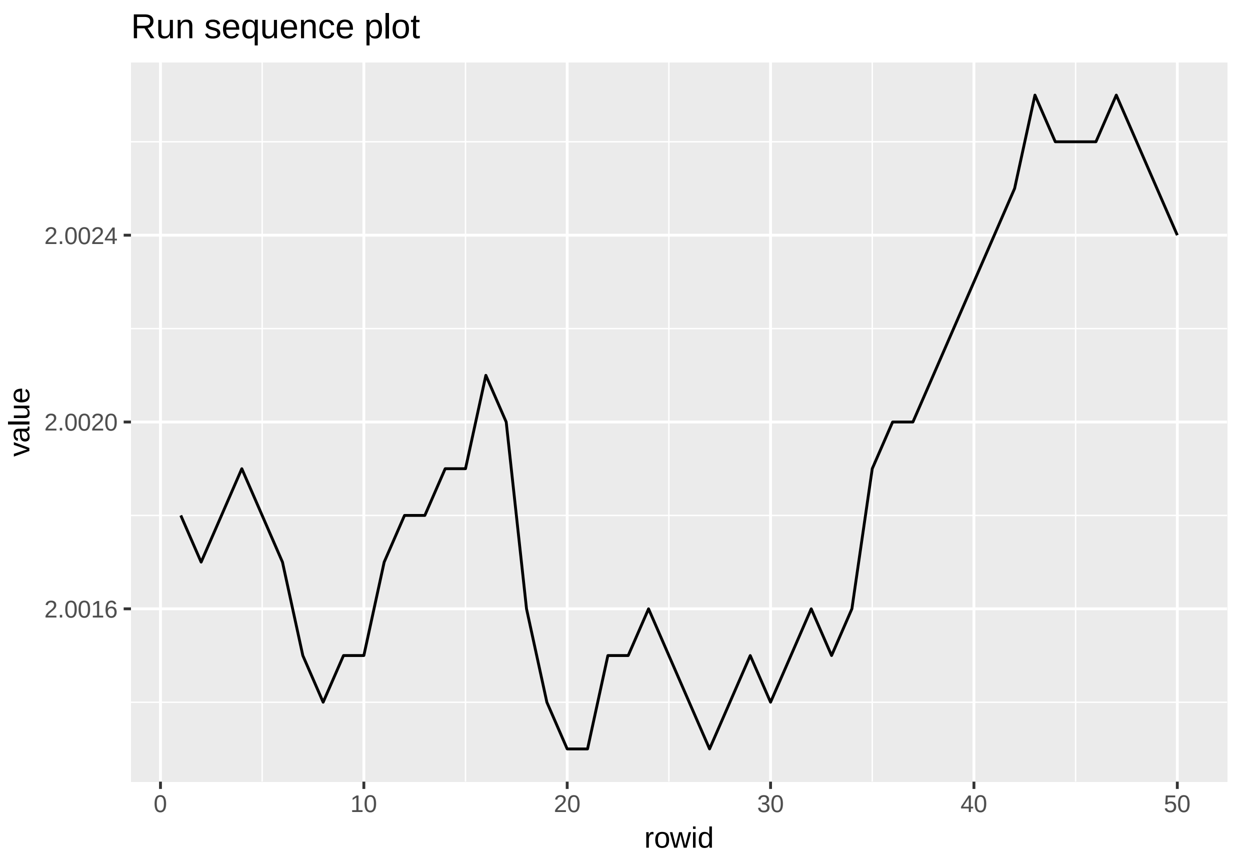
1.4.5 Filter transmitance

Filter Transmitance

```
filter_transmitance <- scan("NIST data/MAVRO.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

filter_transmitance
```

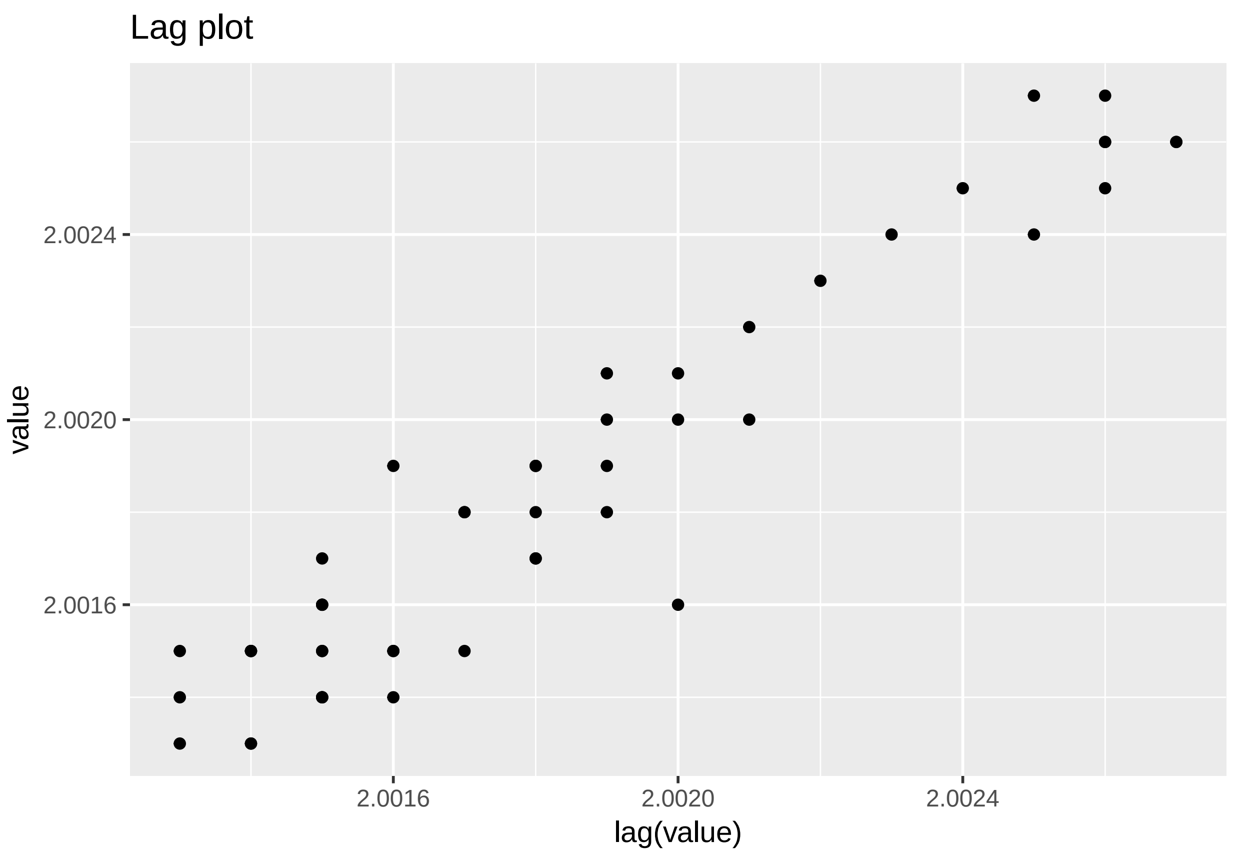
```
## # A tibble: 50 x 2
##    rowid value
##    <int> <dbl>
##  1     1  2.00
##  2     2  2.00
##  3     3  2.00
##  4     4  2.00
##  5     5  2.00
##  6     6  2.00
##  7     7  2.00
##  8     8  2.00
##  9     9  2.00
## 10    10  2.00
## # ... with 40 more rows
```

```
ggplot(filter_transmitance, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



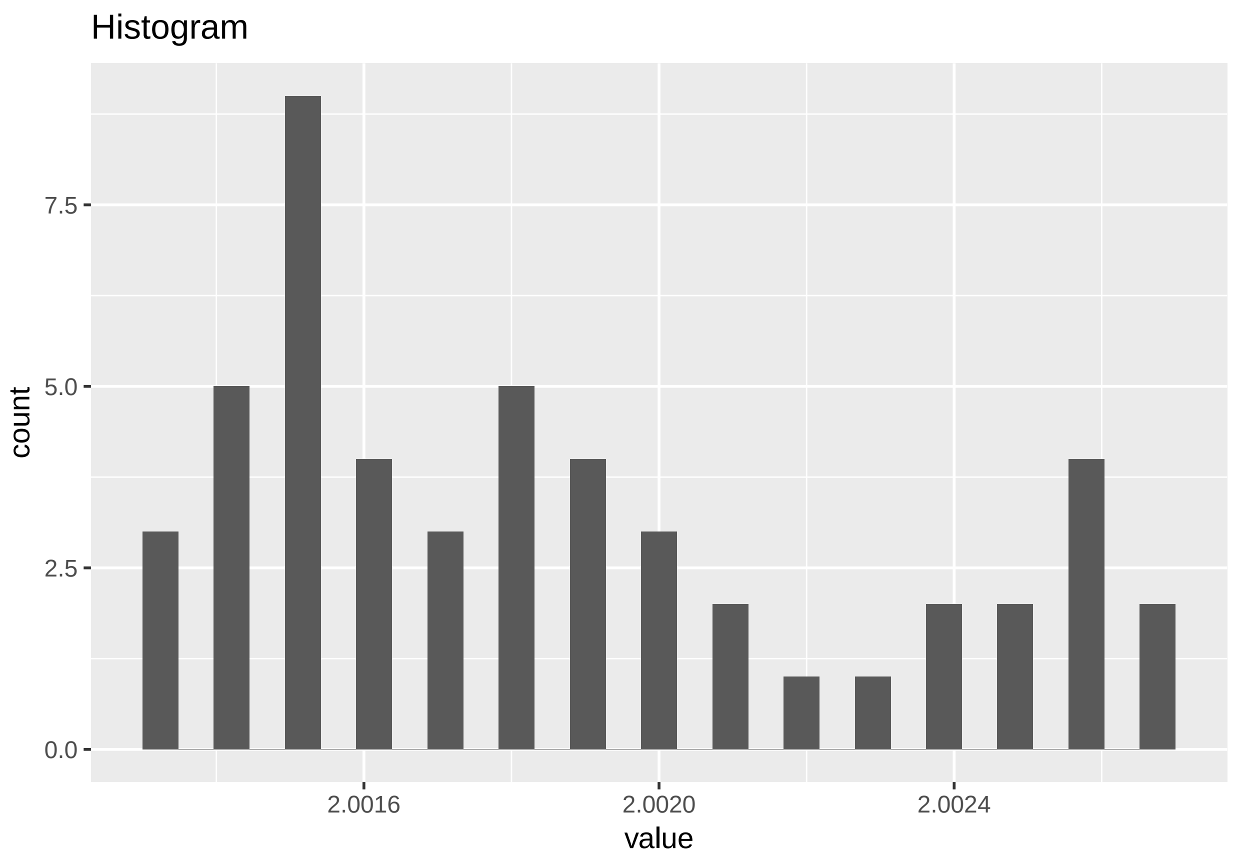
```
ggplot(filter_transmitance, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

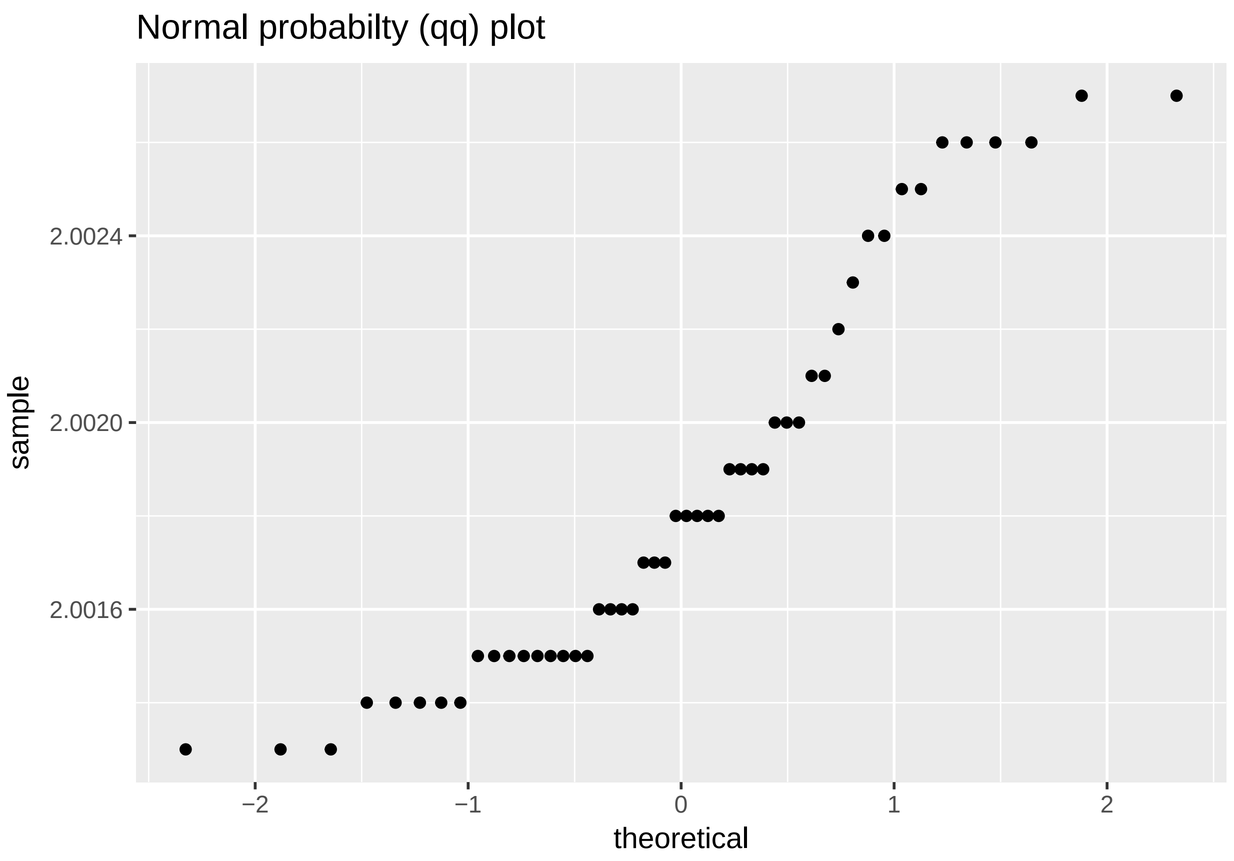


```r
ggplot(filter_transmitance, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(filter_transmitance, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



1.4.6 Standard resistor

Standard Resistor

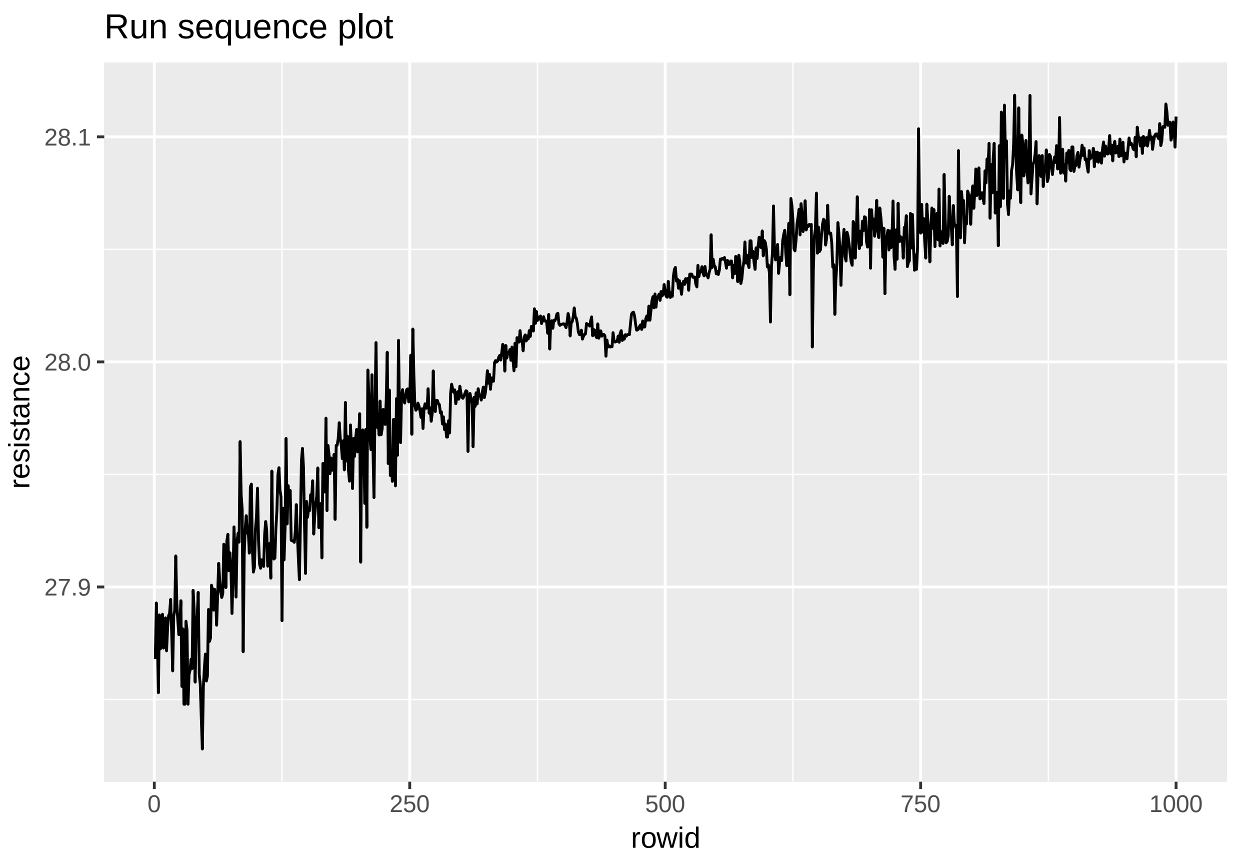
```
standard_resistor <- read_table2("NIST data/DZIUBA1.DAT", skip = 25, col_names = FALSE) %>%
  rowid_to_column() %>%
  rename(month = X1, day = X2, year = X3, resistance = X4)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_character(),
##   X3 = col_integer(),
##   X4 = col_double()
## )
```

```
standard_resistor
```

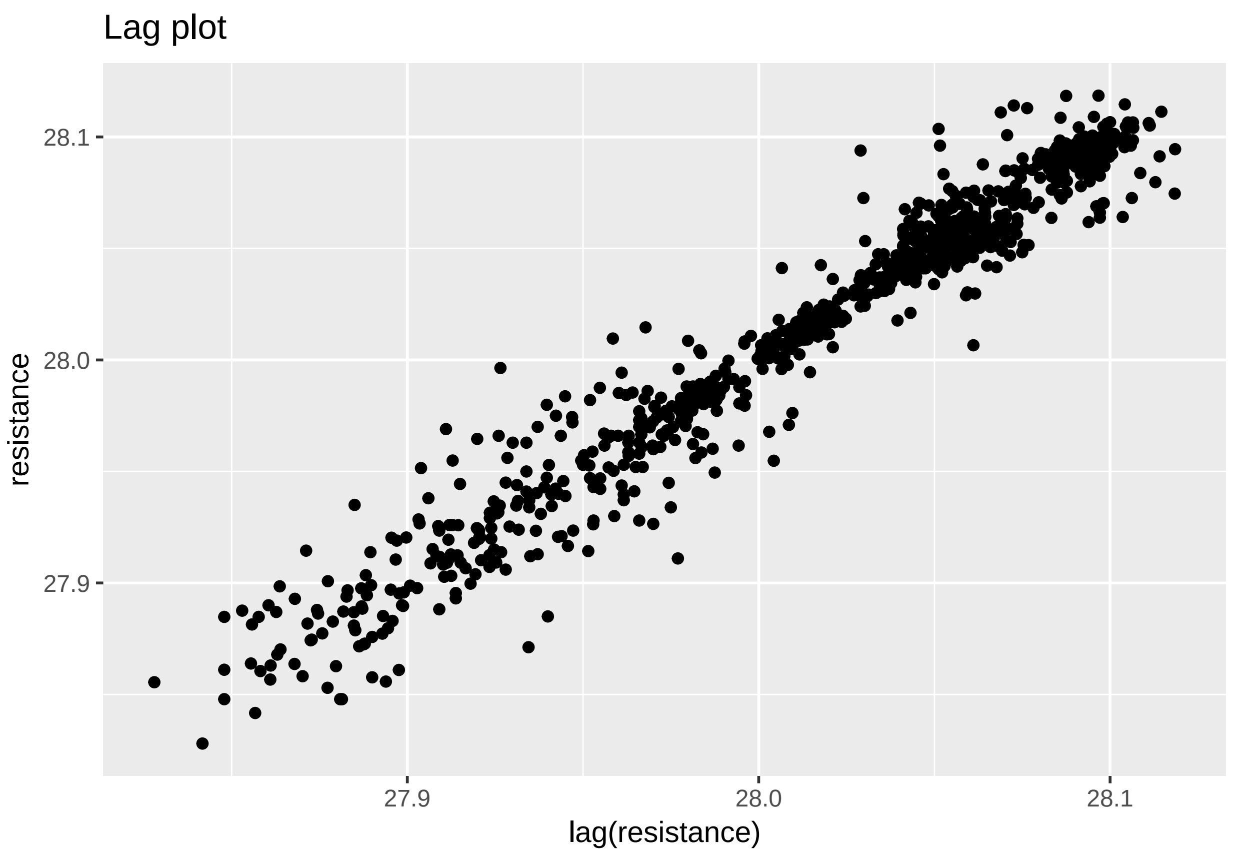
```
## # A tibble: 1,000 x 5
##    rowid month day    year resistance
##    <int> <chr> <chr> <int>      <dbl>
##  1     1 2     5        80       27.9
##  2     2 2     12       80       27.9
##  3     3 2     13       80       27.9
##  4     4 2     14       80       27.9
##  5     5 2     28       80       27.9
##  6     6 2     28       80       27.9
##  7      7 3     21       80       27.9
##  8      8 3     24       80       27.9
##  9      9 4     3        80       27.9
## 10     10 4     3        80       27.9
## # ... with 990 more rows
```

```
ggplot(standard_resistor, aes(rowid, resistance)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



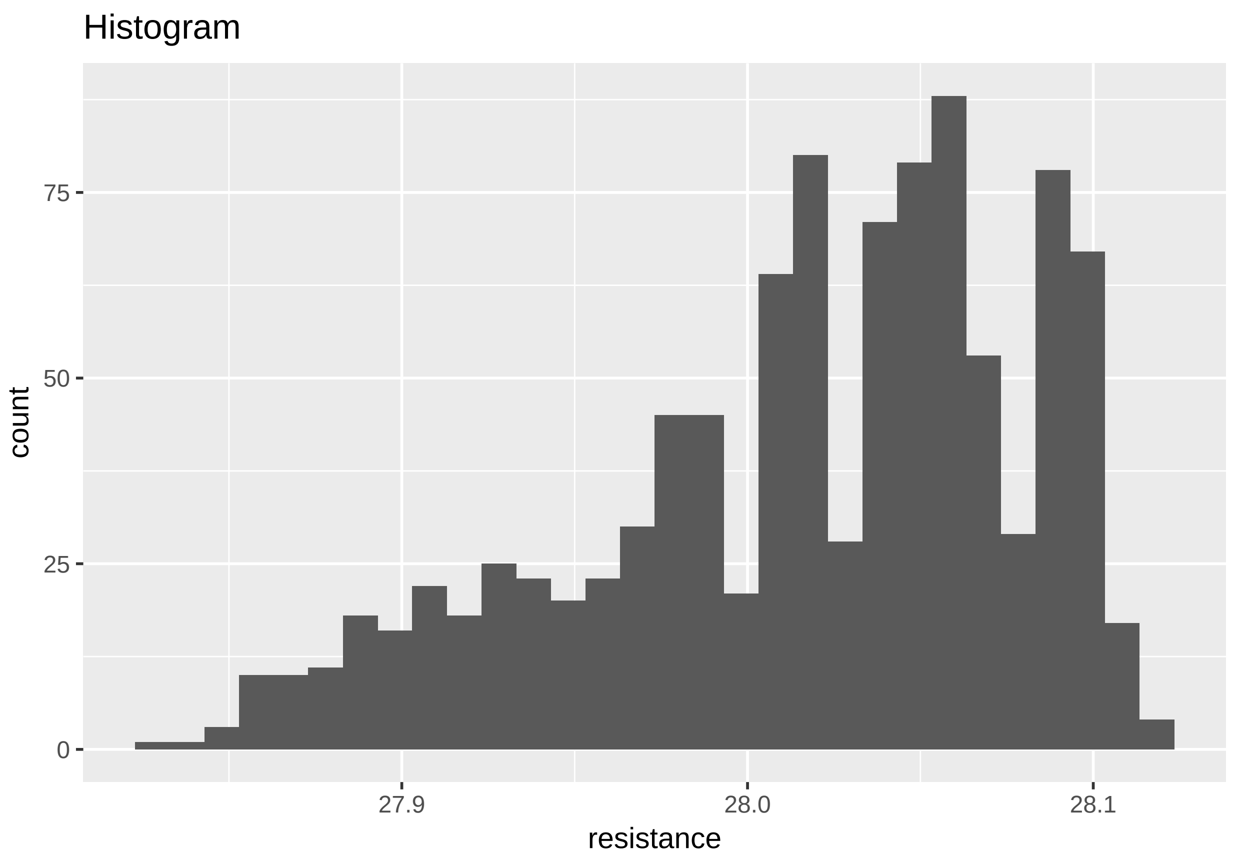
```
ggplot(standard_resistor, aes(lag(resistance), resistance)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

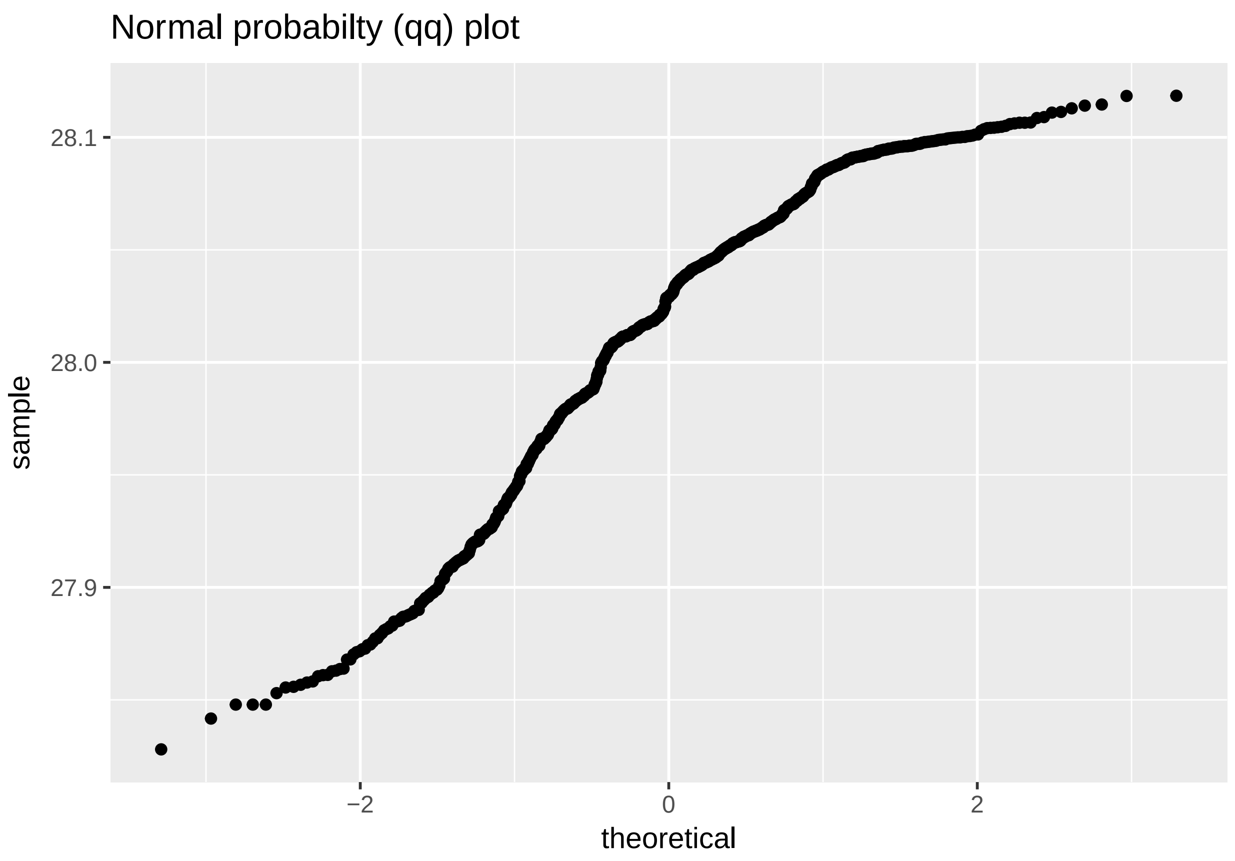


```r
ggplot(standard_resistor, aes(resistance)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(standard_resistor, aes(sample = resistance)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



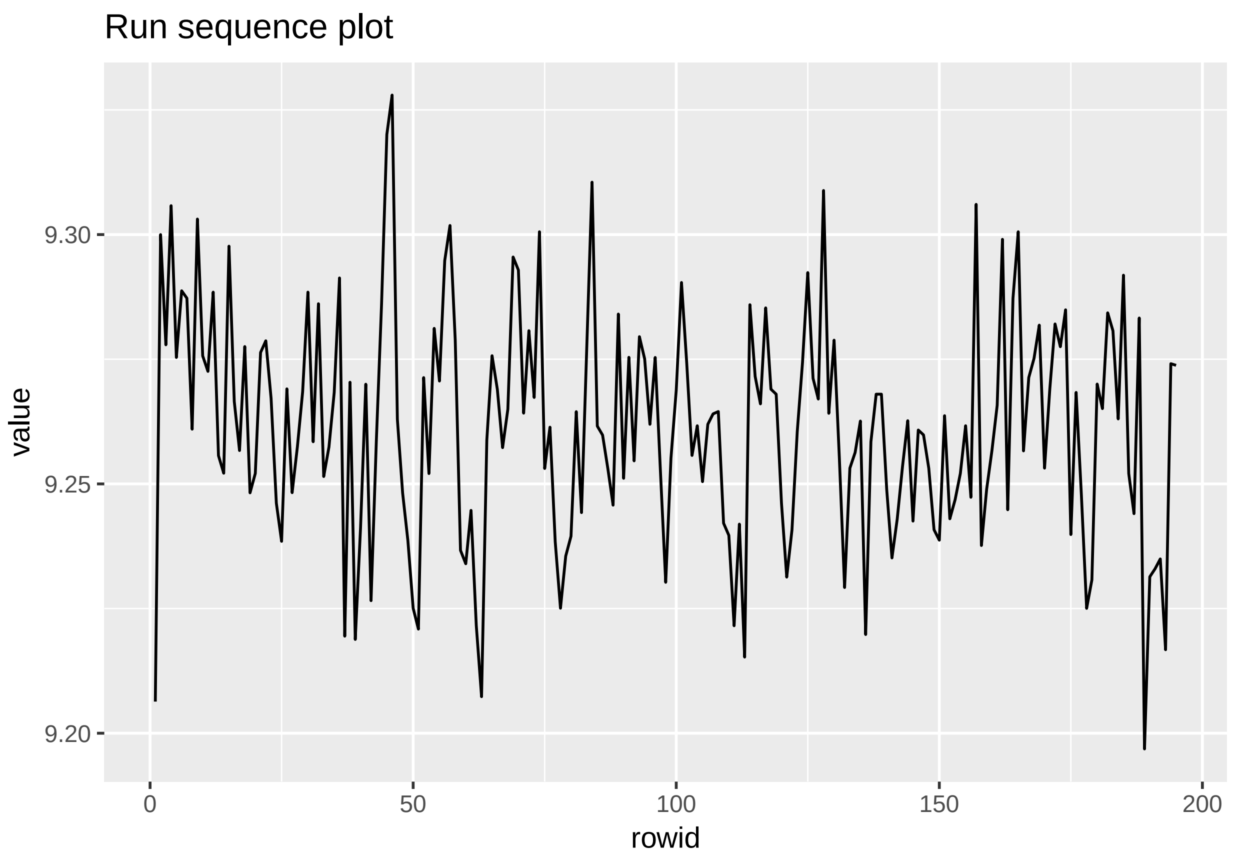
1.4.7 Heat flow meter 1

Heat Flow Meter 1

```
heat_flow_meter1 <- scan("NIST data/ZARR13.DAT", skip = 25) %>%
  as.tibble() %>%
  rowid_to_column()

heat_flow_meter1
```

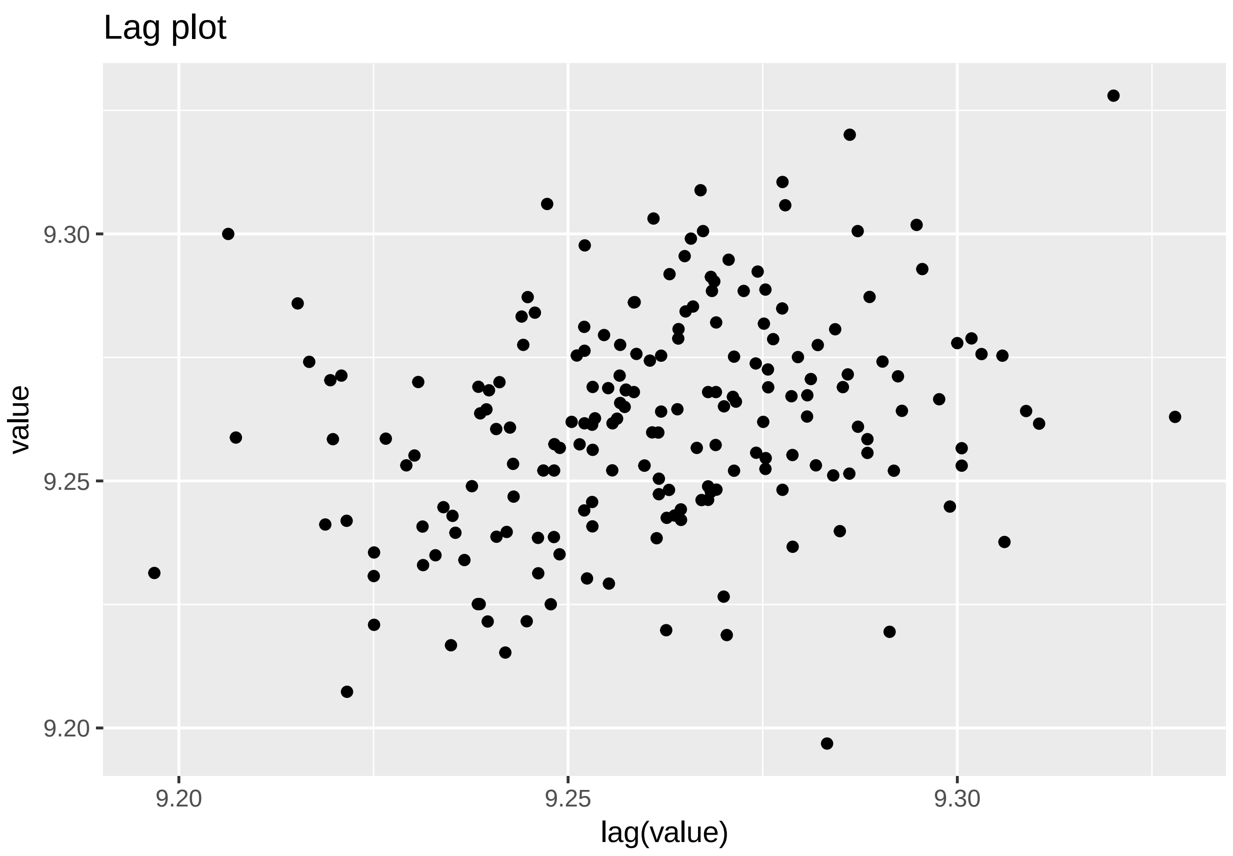
```
## # A tibble: 195 x 2
##    rowid value
##    <int> <dbl>
##  1     1  9.21
##  2     2  9.30
##  3     3  9.28
##  4     4  9.31
##  5     5  9.28
##  6     6  9.29
##  7     7  9.29
##  8     8  9.26
##  9     9  9.30
## 10    10  9.28
## # ... with 185 more rows
```

```
ggplot(heat_flow_meter1, aes(rowid, value)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



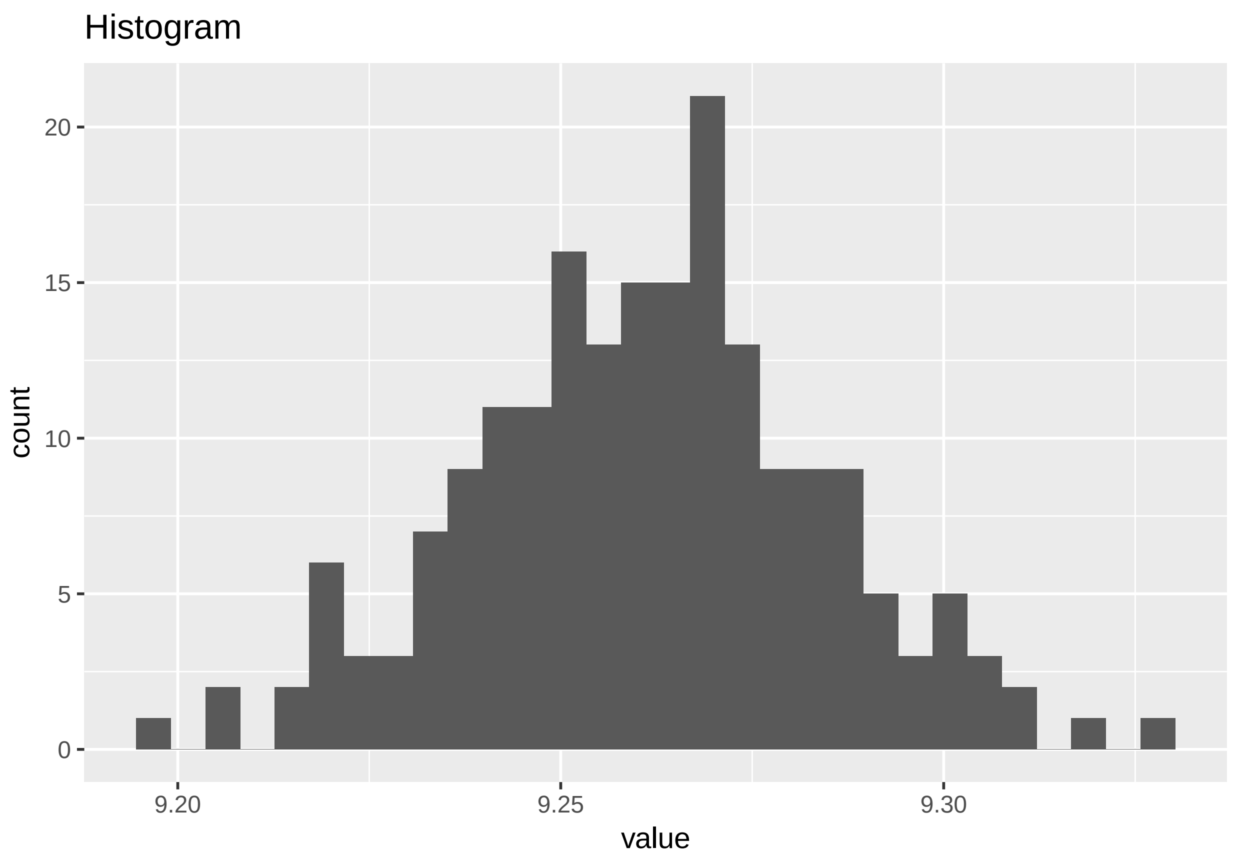
```
ggplot(heat_flow_meter1, aes(lag(value), value)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

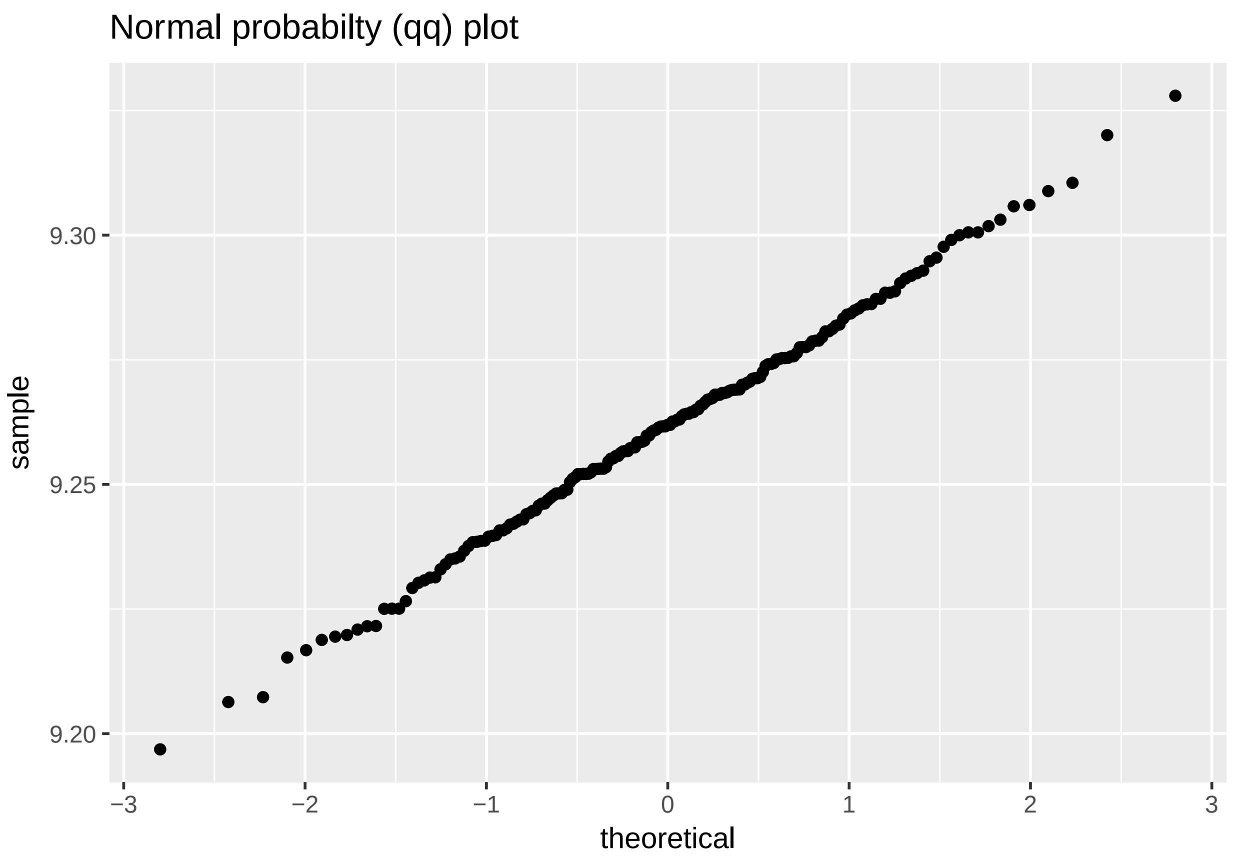


```
ggplot(heat_flow_meter1, aes(value)) +
  geom_histogram() +
  labs(title = "Histogram")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
ggplot(heat_flow_meter1, aes(sample = value)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



1.4.8 Ceramic strength

Ceramic Strength

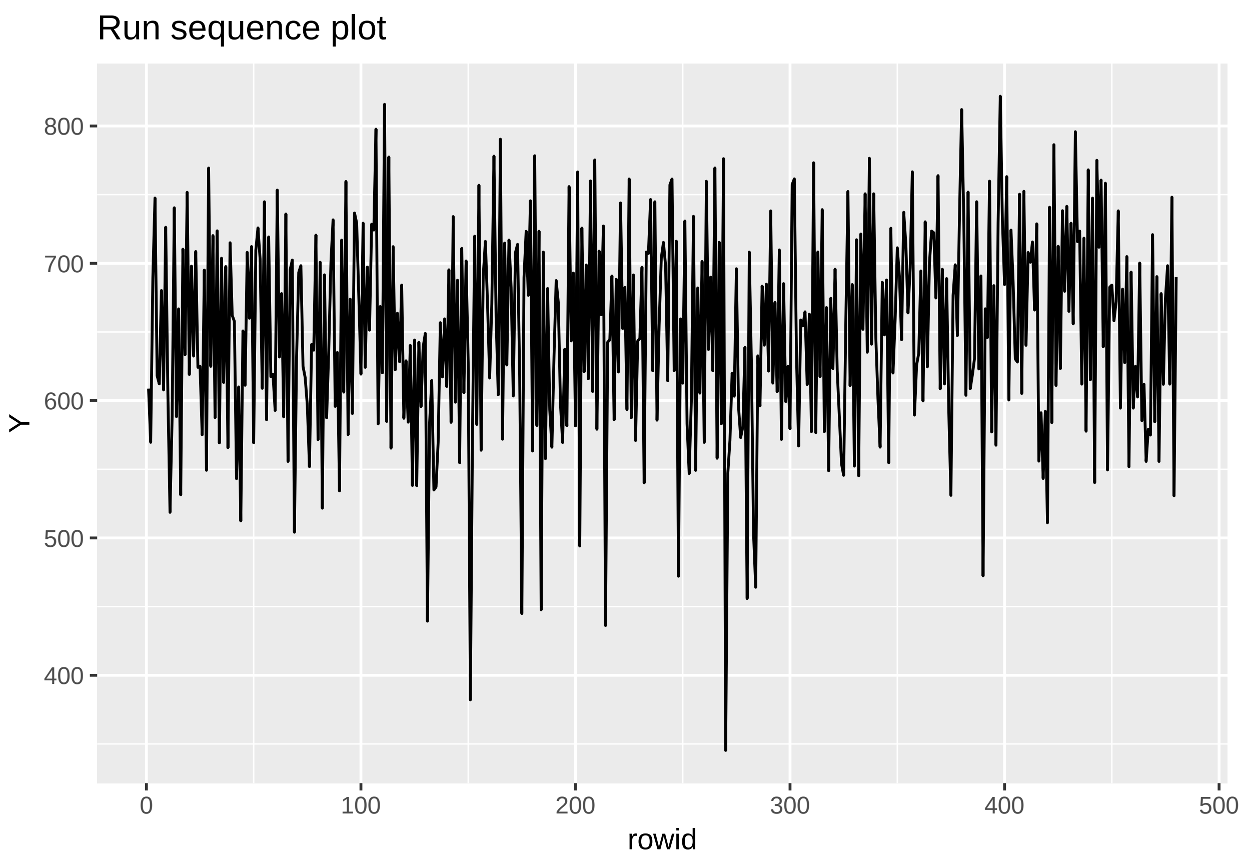
```
ceramic_strength <- read_table2("NIST data/JAHANMI2.DAT", skip = 48, col_names = TRUE) %>%
  filter(Lab >= 1) %>%
  rowid_to_column()
```

```
## Parsed with column specification:
## cols(
##   Id = col_character(),
##   Lab = col_integer(),
##   Num = col_integer(),
##   Test = col_integer(),
##   Y = col_double(),
##   X1 = col_integer(),
##   X2 = col_integer(),
##   X3 = col_integer(),
##   X4 = col_integer(),
##   Trt = col_integer(),
##   Set = col_integer(),
##   Llab = col_double(),
##   Rep = col_integer(),
##   Bat = col_integer(),
##   Sblab = col_double(),
##   Set2 = col_integer()
## )

## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 2)

## Warning: 1 parsing failure.
## row # A tibble: 1 x 5 col     row col   expected   actual   file                 expected   <int> <chr> <chr>
```

```
ceramic_strength
```

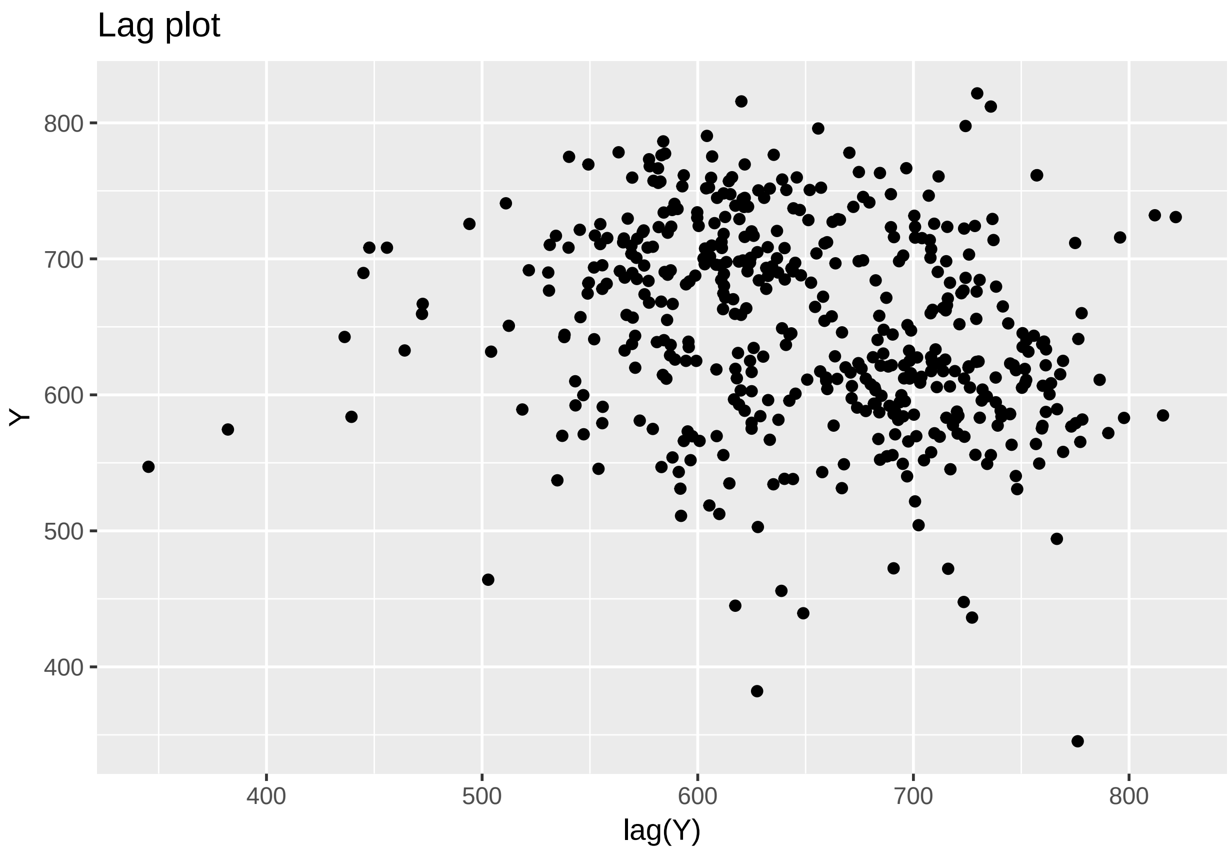
```
## # A tibble: 480 x 17
##    rowid Id      Lab   Num  Test     Y    X1    X2    X3    X4   Trt   Set
##    <int> <chr> <int> <int> <int> <dbl> <int> <int> <int> <int> <int> <int>
##  1     1 1         1     1     1  609.    -1    -1    -1    -1     1     1
##  2     2 2         1     2     1  570.    -1    -1    -1    -1     1     1
##  3     3 3         1     3     1  690.    -1    -1    -1    -1     1     1
##  4     4 4         1     4     1  748.    -1    -1    -1    -1     1     1
##  5     5 5         1     5     1  618.    -1    -1    -1    -1     1     1
##  6     6 6         1     6     1  612.    -1    -1    -1    -1     1     1
##  7     7 7         1     7     1  680.    -1    -1    -1    -1     1     1
##  8     8 8         1     8     1  608.    -1    -1    -1    -1     1     1
##  9     9 9         1     9     1  726.    -1    -1    -1    -1     1     1
## 10    10 10        1    10     1  605.    -1    -1    -1    -1     1     1
## # ... with 470 more rows, and 5 more variables: Llab <dbl>, Rep <int>,
## #   Bat <int>, Sblab <dbl>, Set2 <int>
```

```
ggplot(ceramic_strength, aes(rowid, Y)) +
  geom_line() +
  labs(title = "Run sequence plot")
```



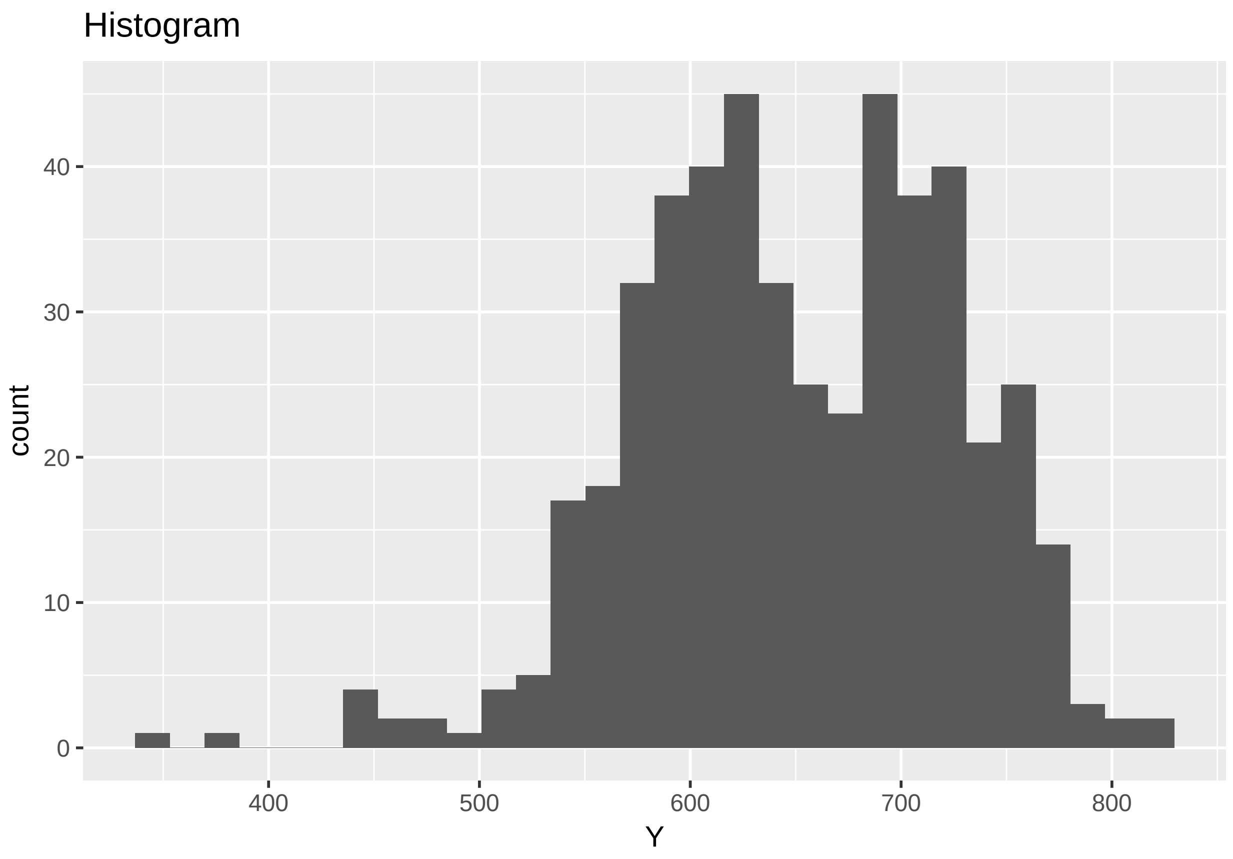
```
ggplot(ceramic_strength, aes(lag(Y), Y)) +
  geom_point() +
  labs(title = "Lag plot")
```

```
## Warning: Removed 1 rows containing missing values (geom_point).
```

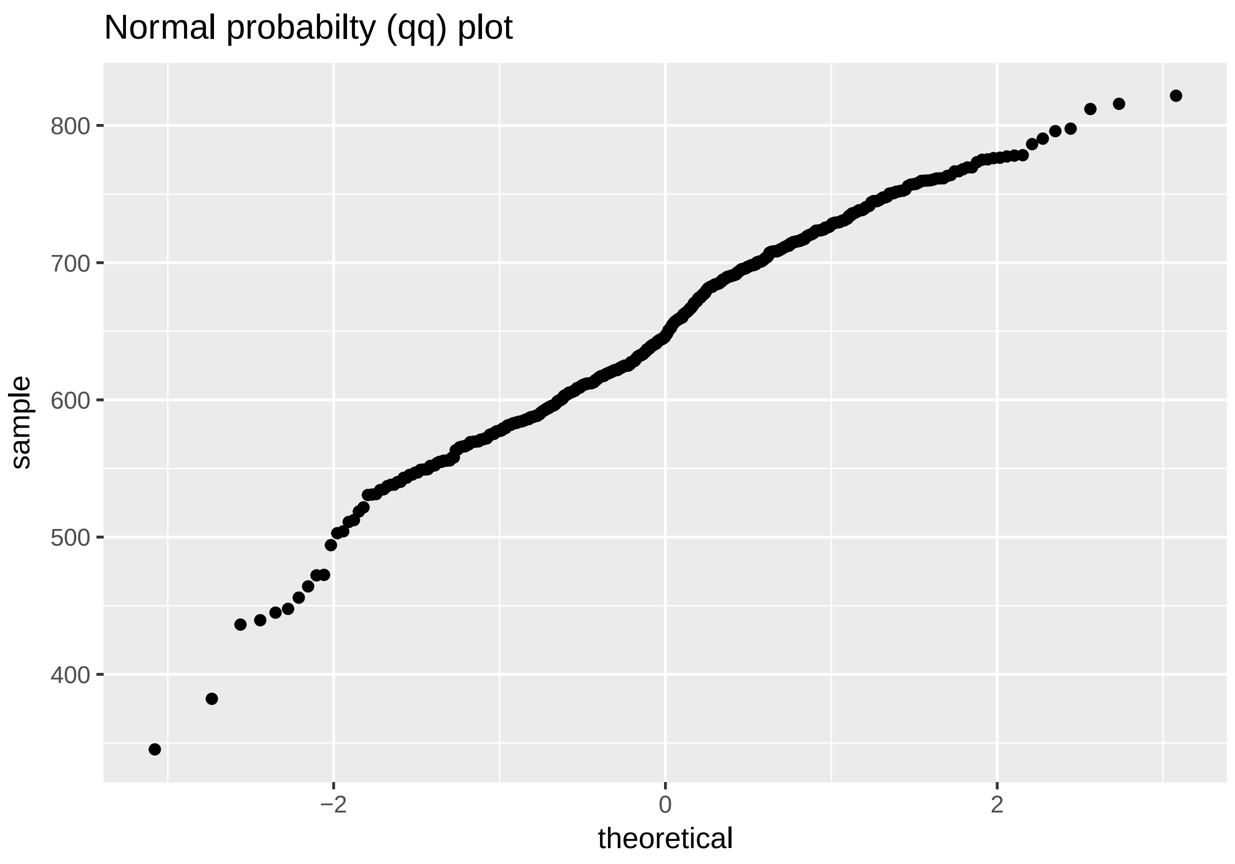


```
ggplot(ceramic_strength, aes(Y)) +
  geom_histogram() +
  labs(title = "Histogram")
```

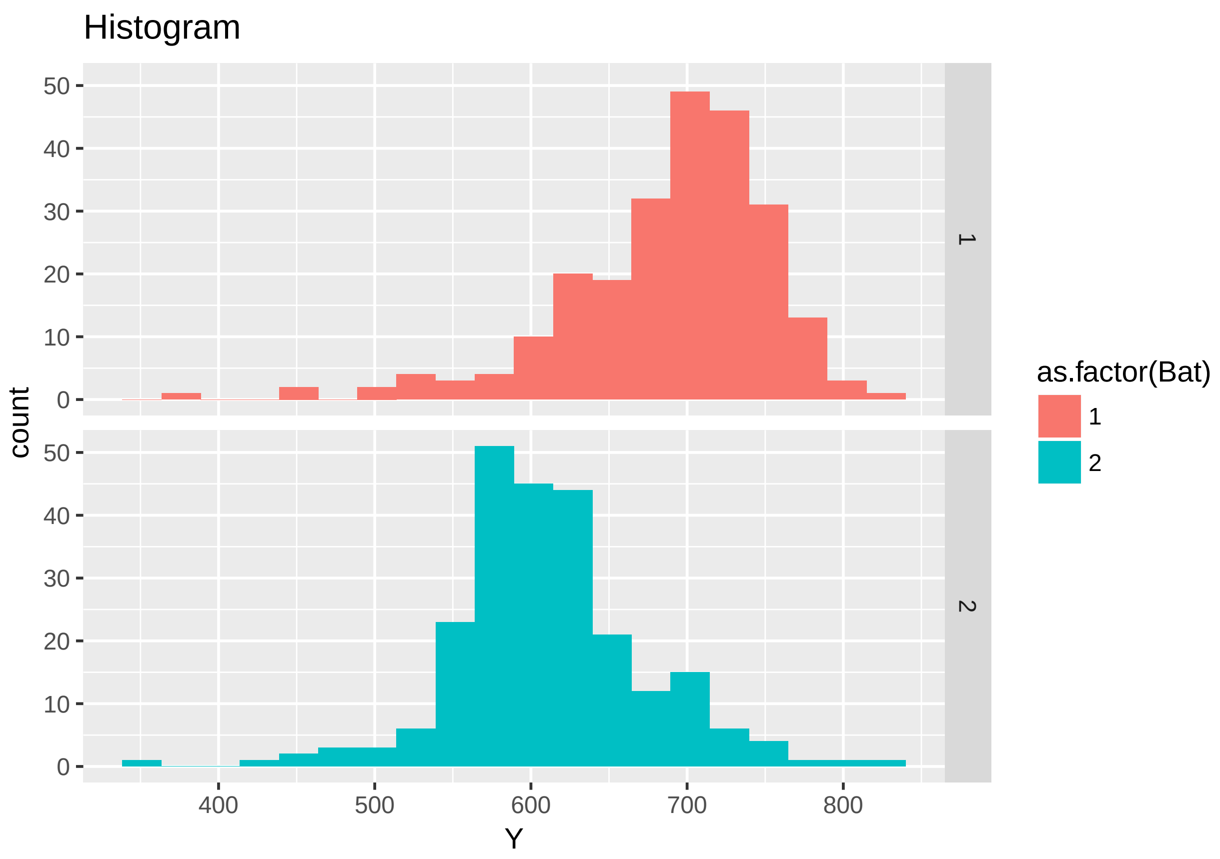
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



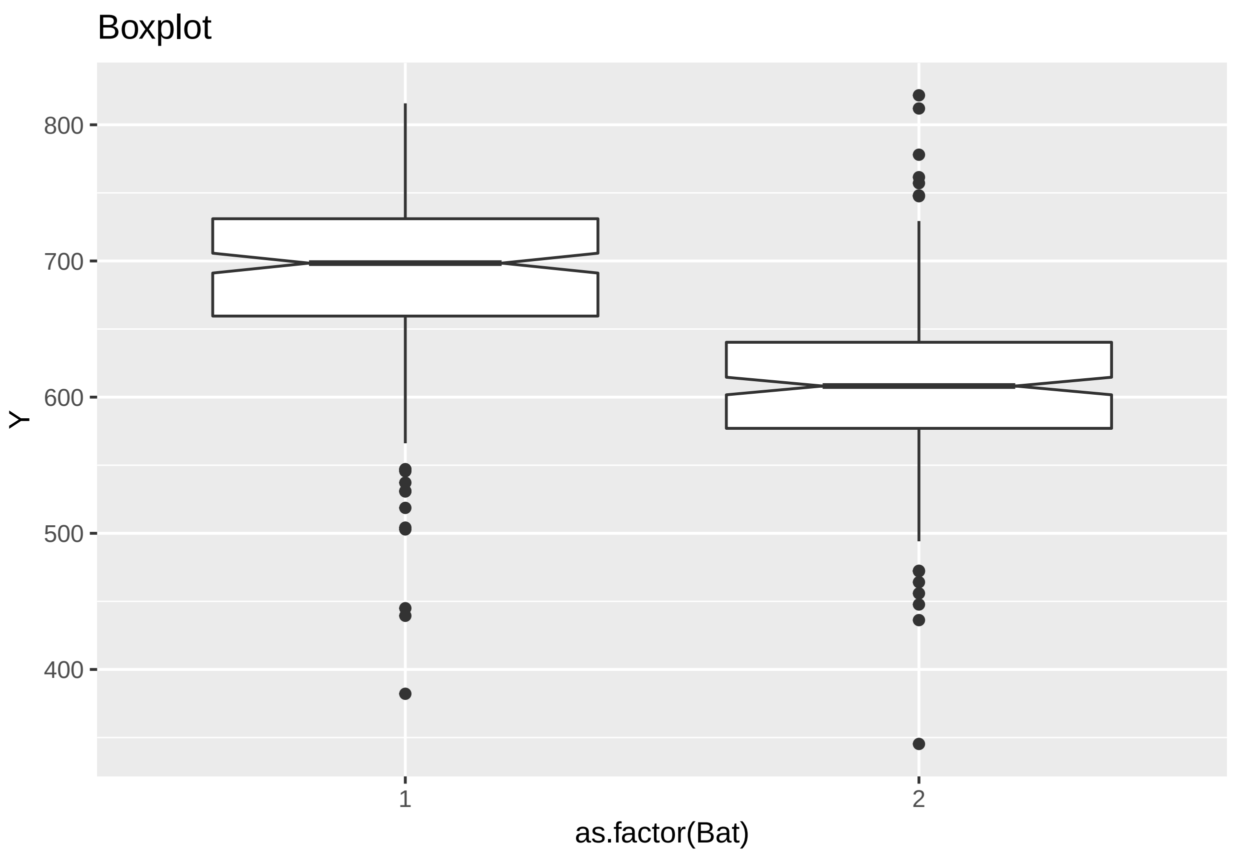
```r
ggplot(ceramic_strength, aes(sample = Y)) +
  geom_qq() +
  labs(title = "Normal probabilty (qq) plot")
```



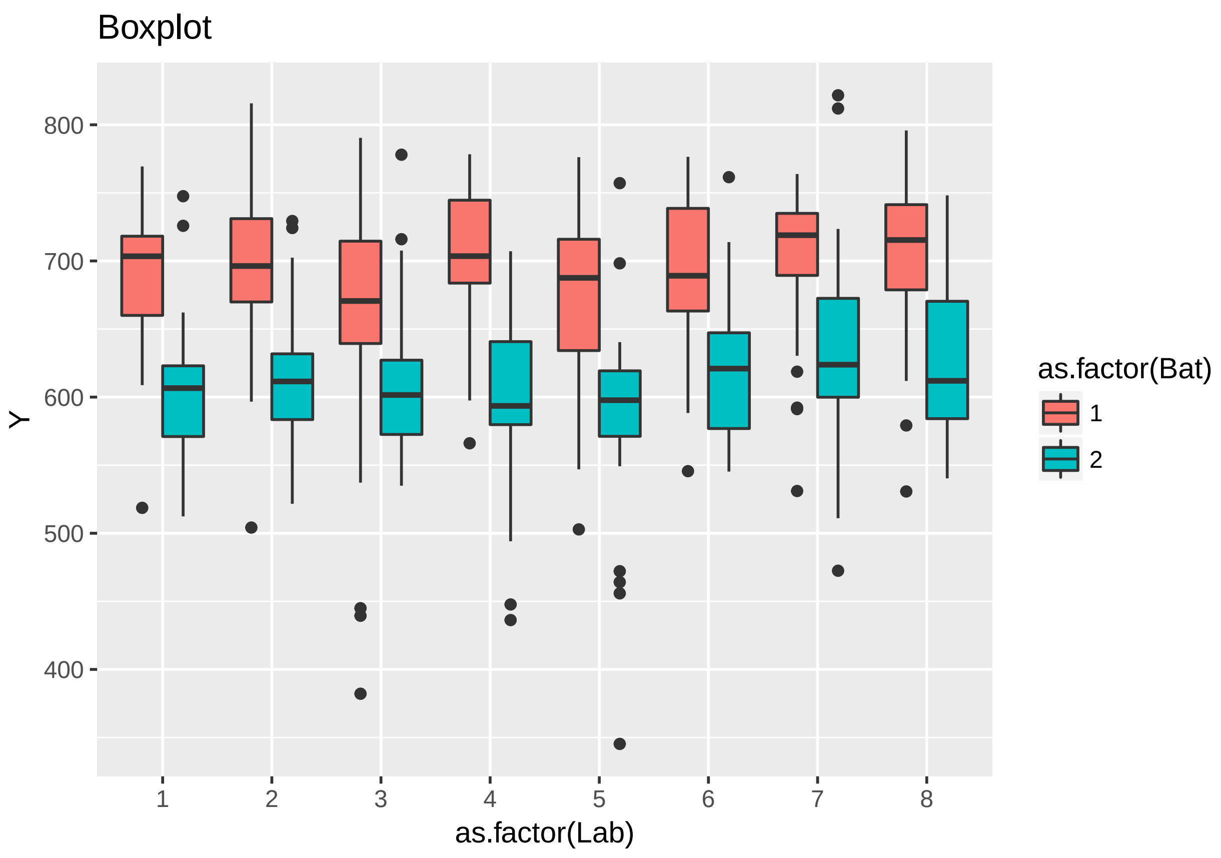
```r
ggplot(ceramic_strength, aes(Y)) +
  geom_histogram(aes(fill = as.factor(Bat)), bins = 20) +
  facet_grid(Bat ~ .) +
  labs(title = "Histogram")
```



```r
ggplot(ceramic_strength, aes(as.factor(Bat), Y)) +
  geom_boxplot(notch = TRUE) +
  labs(title = "Boxplot")
```



```
ggplot(ceramic_strength, aes(as.factor(Lab), Y)) +
  geom_boxplot(aes(fill = as.factor(Bat))) +
  labs(title = "Boxplot")
```



You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter **??**. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter 3.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 1.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

I'd like to add a reference (Hoobler et al., 1993).

```
library(magrittr)
library(tidyverse)
```

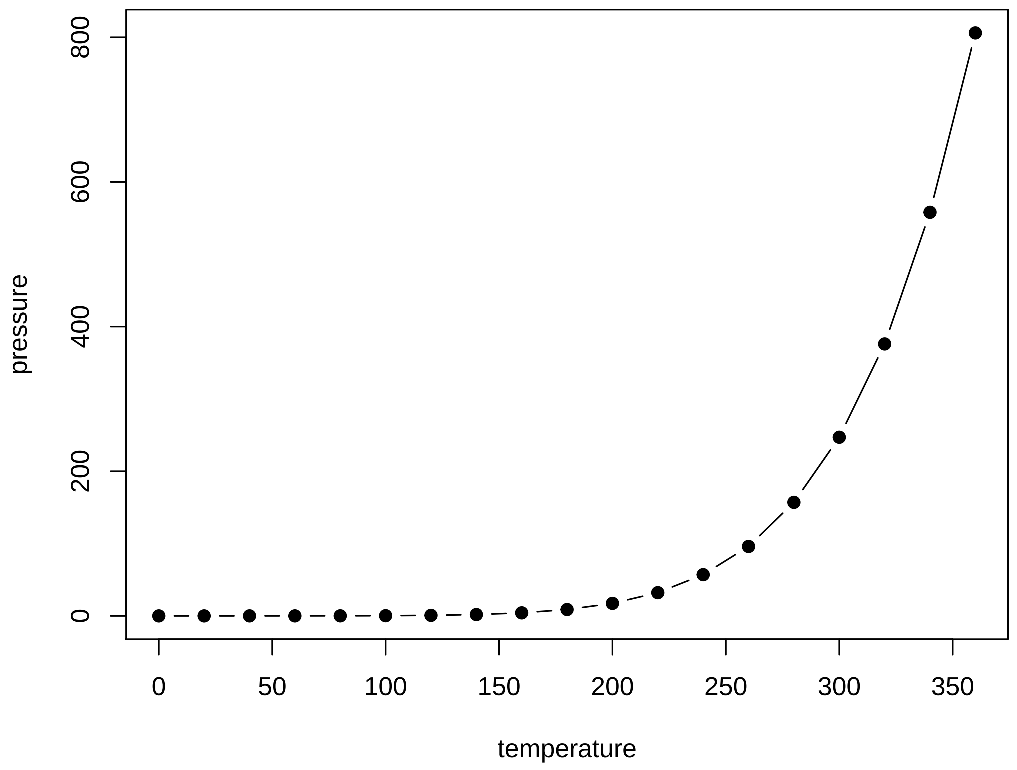


Figure 1.1: Here is a nice figure!

Table 1.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 2

Measurement Process Characterization

2.1 Characterization

2.2 Gauge R & R studies

2.3 Case Studies

2.3.1 Check standard

2.3.1.1 Background and data

The measurements on the check standard duplicate certification measurements that were being made, during the same time period, on individual wafers from crystal #51939. For the check standard there were:

- J = 6 repetitions at the center of the wafer on each day
- K = 25 days

Check standard for resistivity measurement

2.3.1.2 Reading the dataset

```
check_standard <- read_table2("NIST data/MPC62_clean.DAT", col_names = TRUE) %>%
  rowid_to_column()
```

```
## Parsed with column specification:
## cols(
##   Crystal_ID = col_integer(),
##   Wafer_ID = col_integer(),
##   Month = col_character(),
##   Day = col_character(),
##   Hour = col_character(),
##   Minute = col_character(),
##   Operator = col_integer(),
##   Humidity = col_integer(),
##   Probe_ID = col_integer(),
##   Temperature = col_double(),
##   Resistance = col_double(),
##   Stdev = col_double(),
##   Df = col_integer()
## )
```

```
check_standard
```

```
## # A tibble: 25 x 14
##    rowid Crystal_ID Wafer_ID Month Day   Hour  Minute Operator Humidity
##    <int>      <int>    <int> <chr> <chr> <chr> <chr>     <int>    <int>
##  1     1      51939      137 03    24    18    01            1       42
##  2     2      51939      137 03    25    12    41            1       35
##  3     3      51939      137 03    25    15    57            1       33
##  4     4      51939      137 03    28    10    10            2       47
##  5     5      51939      137 03    28    13    31            2       44
##  6     6      51939      137 03    28    17    33            1       43
##  7     7      51939      137 03    29    14    40            1       36
##  8     8      51939      137 03    29    16    33            1       35
##  9     9      51939      137 03    30    05    45            2       32
## 10    10      51939      137 03    30    09    26            2       33
## # ... with 15 more rows, and 5 more variables: Probe_ID <int>,
## #   Temperature <dbl>, Resistance <dbl>, Stdev <dbl>, Df <int>
```

2.3.1.3 Level-1 standard deviation

Measurements for J runs over K days for L runs are:

$$Y_{lkj}(l = 1, \ldots, L, \; k = 1, \ldots, K, \; j = 1, \ldots, J)$$

The level-1 repeatability (short term precision) is calcuated from the pooled standard deviation over days and runs

$$s_{1lk} = \sqrt{\frac{1}{J-1}\sum_{j=1}^{J}(Y_{lkj} - \overline{Y}_{lk\bullet})^2}$$

with

$$\overline{Y}_{lk\bullet} = \frac{1}{J}\sum_{j=1}^{J}\overline{Y}_{lkj}$$

As stated in the e-Handbook: >An individual short-term standard deviation will not be a reliable estimate of precision if the degrees of freedom is less than ten, but the individual estimates can be pooled over the K days to obtain a more reliable estimate.

The pooled level-1 standard deviation estimate with v = K(J - 1) degrees of freedom is

$$s_1 = \sqrt{\frac{1}{K}\sum_{k=1}^{K} s_k^2}$$

```r
s1_chkstd <- check_standard %>%
  mutate(Stdev_sq = Stdev^2) %$%
  mean(Stdev_sq) %>%
  sqrt()

s1_chkstd
```

```
## [1] 0.06138795
```

Several comments on the code above. I've introduced the `%$%` operator. This allows me to use indivdual columns from my data frame and is useful for preforming mathematical operations on a specific column of data. It is from the **magrittr** package.

I find this type of code easy to read and understand. Describing the operations is simple, I'm just working from inside out of the equation:

- creating a new column of data that is $(Stdev)^2$
- finding the mean of that new column
- taking the square root of that number to give s_1.

2.3.1.4 Level-2 standard deviation (reproducibility)

$$s_{chkstd} = s_2 = \sqrt{\frac{1}{K-1}\sum_{k=1}^{K}\left(\overline{Y}_{k\bullet} - \overline{Y}_{\bullet\bullet}\right)^2}$$

with

$$\overline{Y}_{\bullet\bullet} = \frac{1}{K}\sum_{k=1}^{K}\overline{Y}_{k\bullet}$$

Which is simply the standard deviation of the daily measuremnts

```r
s2_chkstd <- check_standard %$%
  sd(Resistance)

s2_chkstd
```

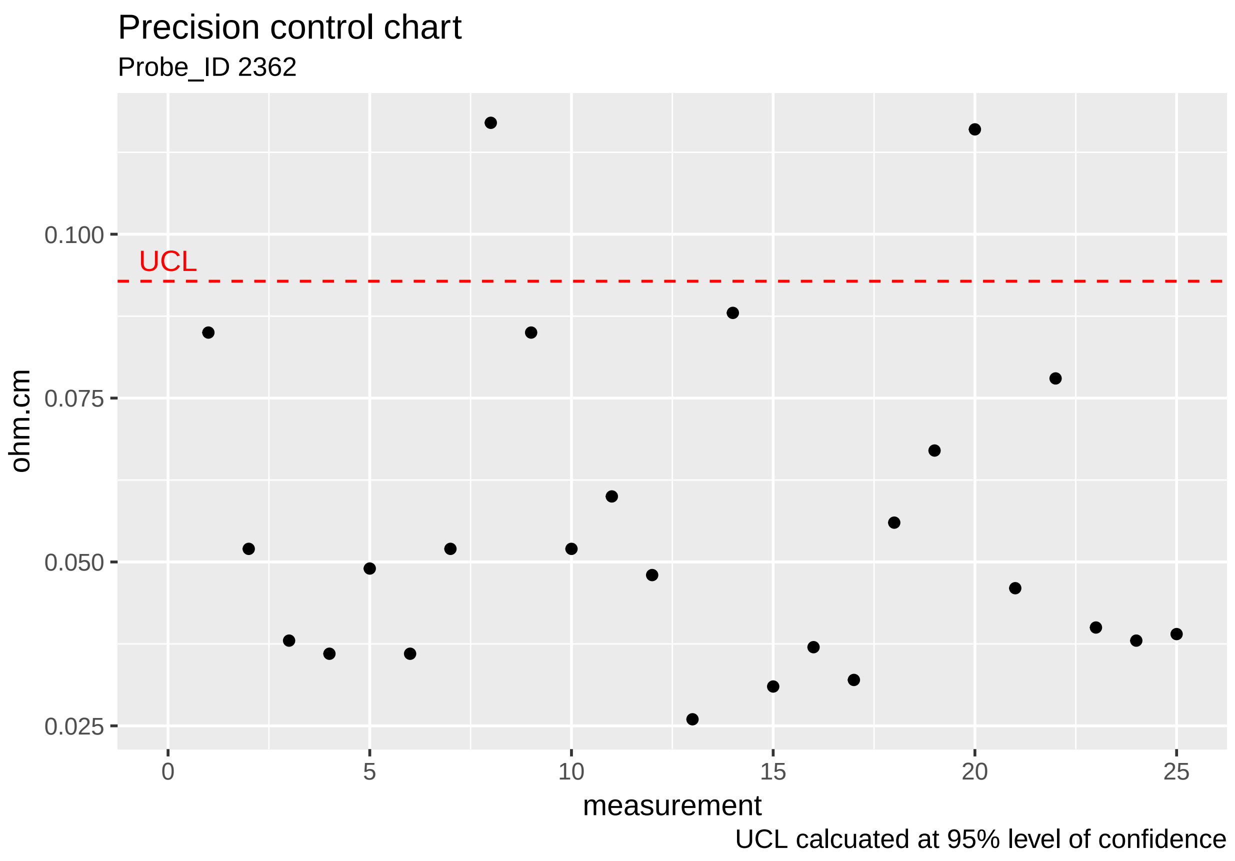
```
## [1] 0.02679813
```

2.3.1.5 Control chart for standard deviation - Precision

```r
UCL_precision_ckkstd <- s1_chkstd*sqrt(qf(0.95, 5, 125))
UCL_precision_ckkstd
```

```
## [1] 0.0928313
```

```r
ggplot(check_standard) +
  geom_point(aes(rowid, Stdev)) +
  geom_hline(aes(yintercept = UCL_precision_ckkstd), colour = "red", linetype = "dashed") +
  labs(title =  "Precision control chart", subtitle = "Probe_ID 2362", x = "measurement", y = "ohm.cm",
  annotate("text", x = 0, y = 0.096, label = "UCL", colour = "red")
```

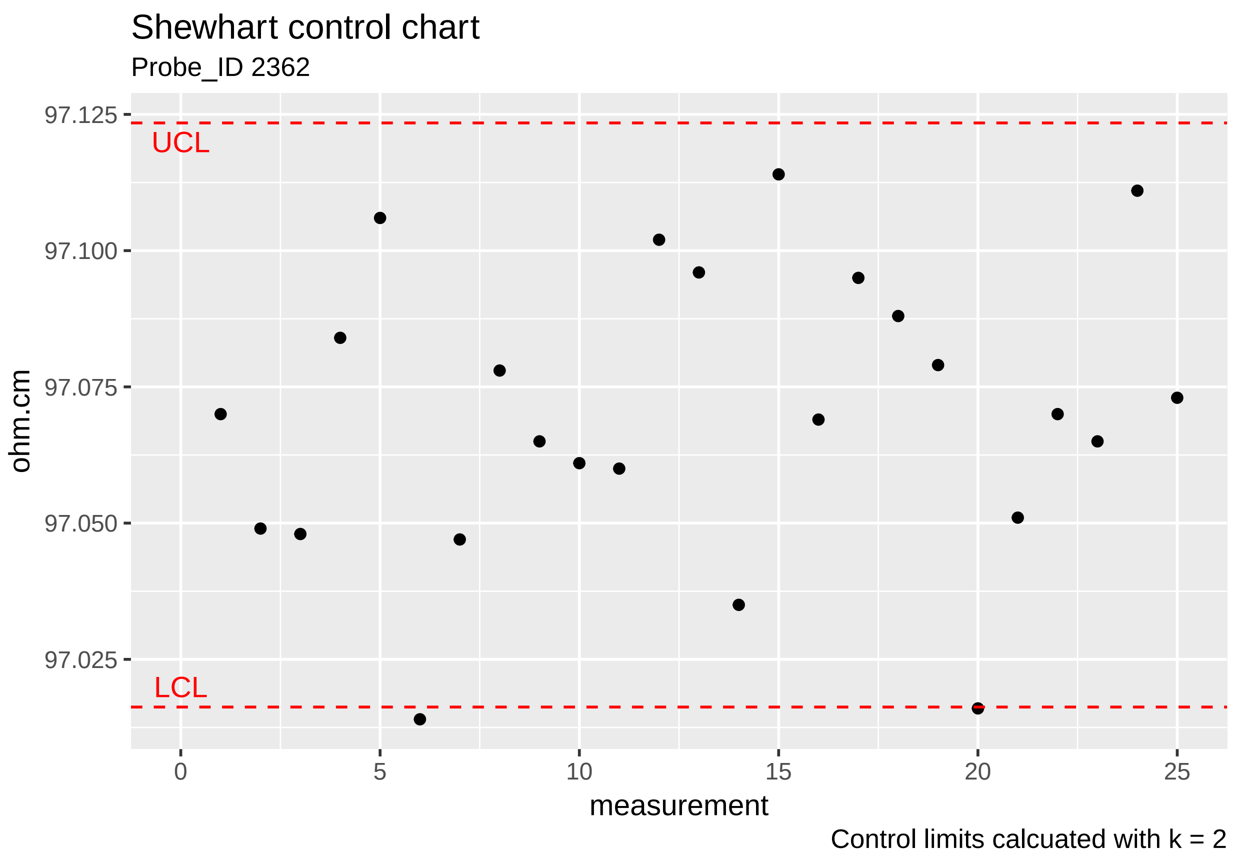


2.3.1.6 Control chart for measurement bias and variability

The control limits for monitoring the bias and long-term variability of resistivity with a Shewhart control chart are given by

$$UCL = \text{Average} + 2 \cdot s_2 Centerline = \text{Average} LCL = \text{Average} 2 \cdot s_2$$

```r
ggplot(check_standard) +
  geom_point(aes(rowid, Resistance)) +
  geom_hline(aes(yintercept = (mean(Resistance) + 2*s2_chkstd)), colour = "red", linetype = "dashed") +
  geom_hline(aes(yintercept = (mean(Resistance) - 2*s2_chkstd)), colour = "red", linetype = "dashed") +
  labs(title =  "Shewhart control chart", subtitle = "Probe_ID 2362", x = "measurement", y = "ohm.cm", 
  annotate("text", x = 0, y = 97.12, label = "UCL", colour = "red") +
  annotate("text", x = 0, y = 97.02, label = "LCL", colour = "red")
```



2.3.2 Gauge study

2.3.2.1 Background and data

Measurements on the check standards are used to estimate repeatability, day effect, and run effect The effect of operator was not considered to be significant for this study; therefore, 'day' replaces 'operator' as a factor in the nested design. Averages and standard deviations from J = 6 measurements at the center of each wafer are shown in the table.

- J = 6 measurements at the center of the wafer per day
- K = 6 days (one operator) per repetition
- L = 2 runs (complete)
- Q = 5 wafers (check standards 138, 139, 140, 141, 142)
- R = 5 probes (1, 281, 283, 2062, 2362)

Gauge study of resistivity probes

```
gauge_study <- read_table2("NIST data/MPC61_clean.DAT", col_names = TRUE) %>%
  rowid_to_column()
```

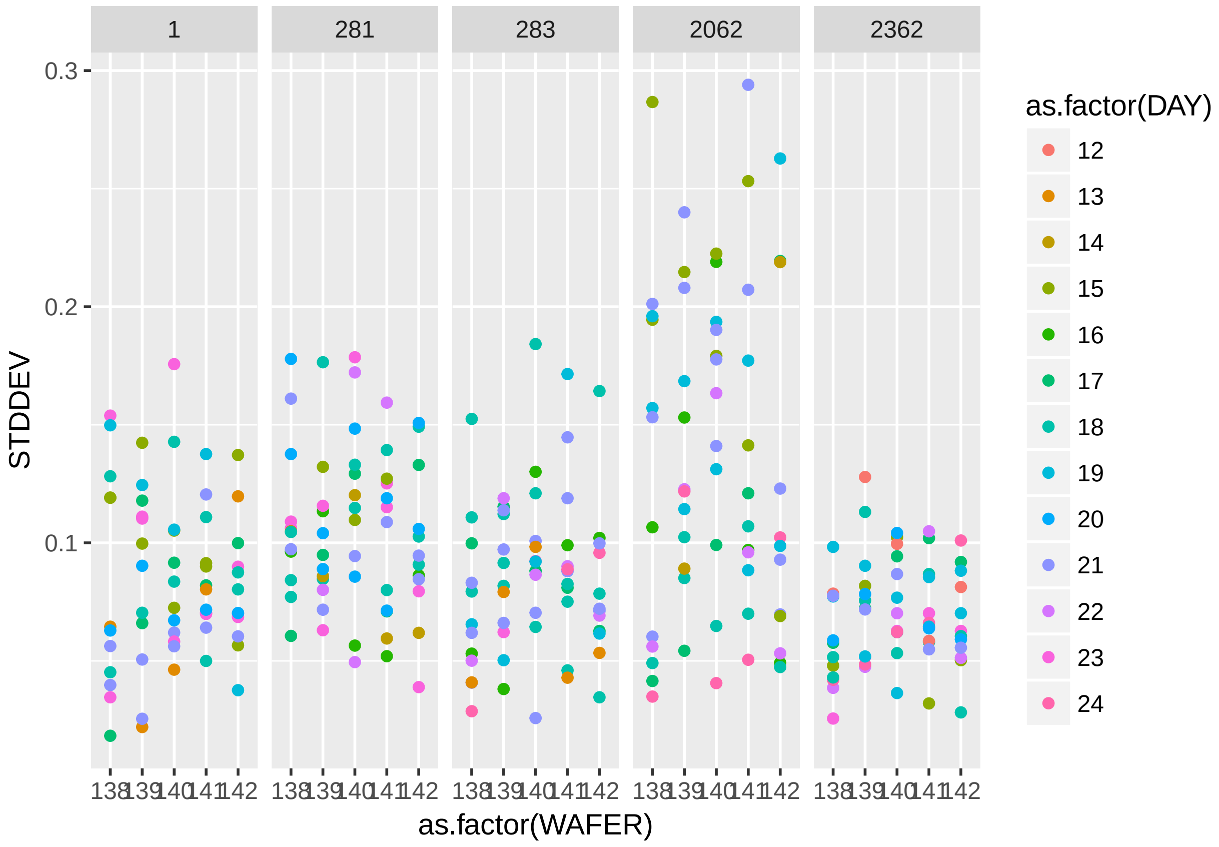
```
## Parsed with column specification:
## cols(
##   RUN = col_integer(),
##   WAFER = col_double(),
##   PROBE = col_double(),
##   MONTH = col_double(),
##   DAY = col_double(),
##   OP = col_double(),
##   TEMP = col_double(),
##   AVERAGE = col_double(),
##   STDDEV = col_double()
## )
```

```
gauge_study
```

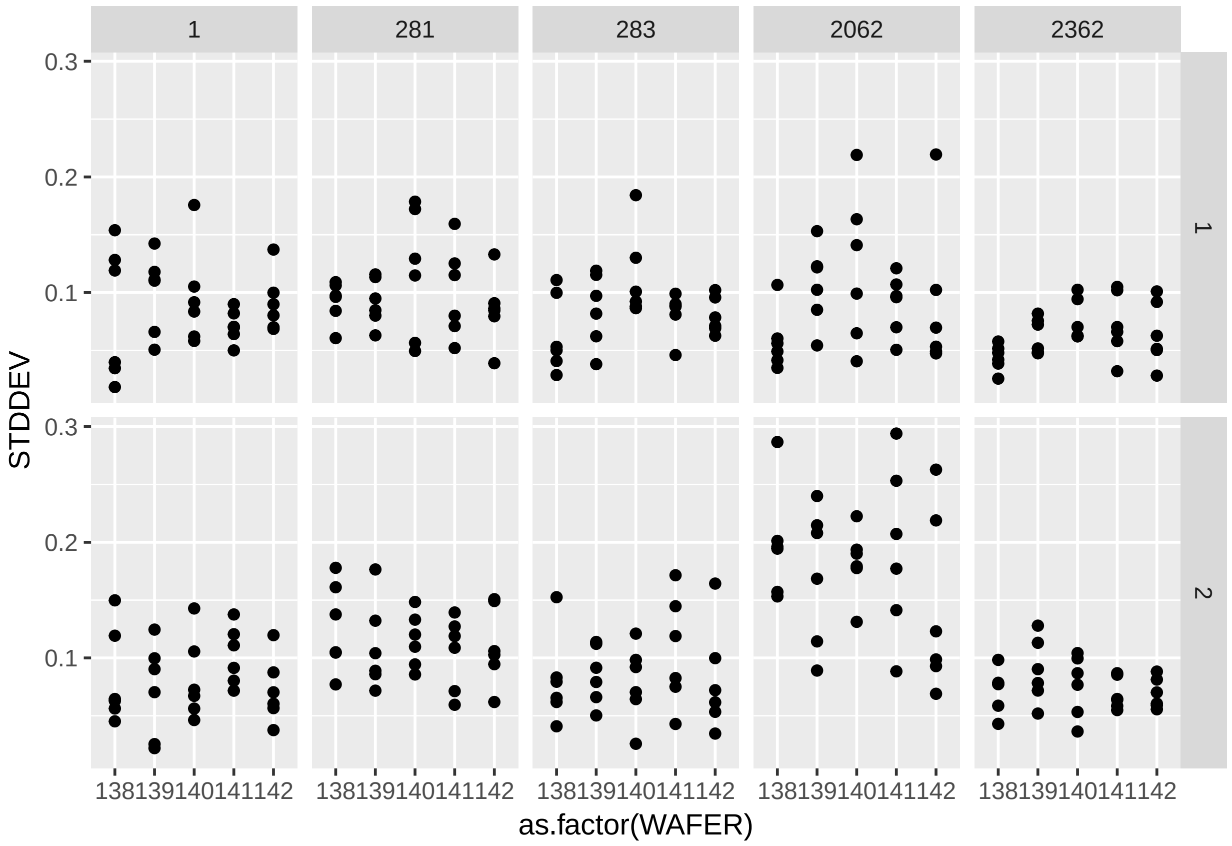
```
## # A tibble: 300 x 10
##    rowid   RUN WAFER PROBE MONTH   DAY    OP  TEMP AVERAGE STDDEV
##    <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>  <dbl>
##  1     1     1  138.    1.    3.   15.    1.  23.0    95.2 0.119
##  2     2     1  138.    1.    3.   17.    1.  23.0    95.2 0.0183
##  3     3     1  138.    1.    3.   18.    1.  22.8    95.2 0.128
##  4     4     1  138.    1.    3.   21.    1.  23.2    95.2 0.0398
##  5     5     1  138.    1.    3.   23.    2.  23.2    95.1 0.0346
##  6     6     1  138.    1.    3.   23.    1.  23.2    95.1 0.154
##  7     7     1  138.  281.    3.   16.    1.  23.0    95.2 0.0963
##  8     8     1  138.  281.    3.   17.    1.  23.0    95.1 0.0606
##  9     9     1  138.  281.    3.   18.    1.  22.8    95.1 0.0842
## 10    10     1  138.  281.    3.   21.    1.  23.3    95.1 0.0973
## # ... with 290 more rows
```

2.3.2.2 Repeatability standard deviations

```
ggplot(gauge_study) +
  geom_point(aes(as.factor(WAFER), STDDEV, colour = as.factor(DAY))) +
  facet_wrap(~ as.factor(PROBE), nrow = 1)
```

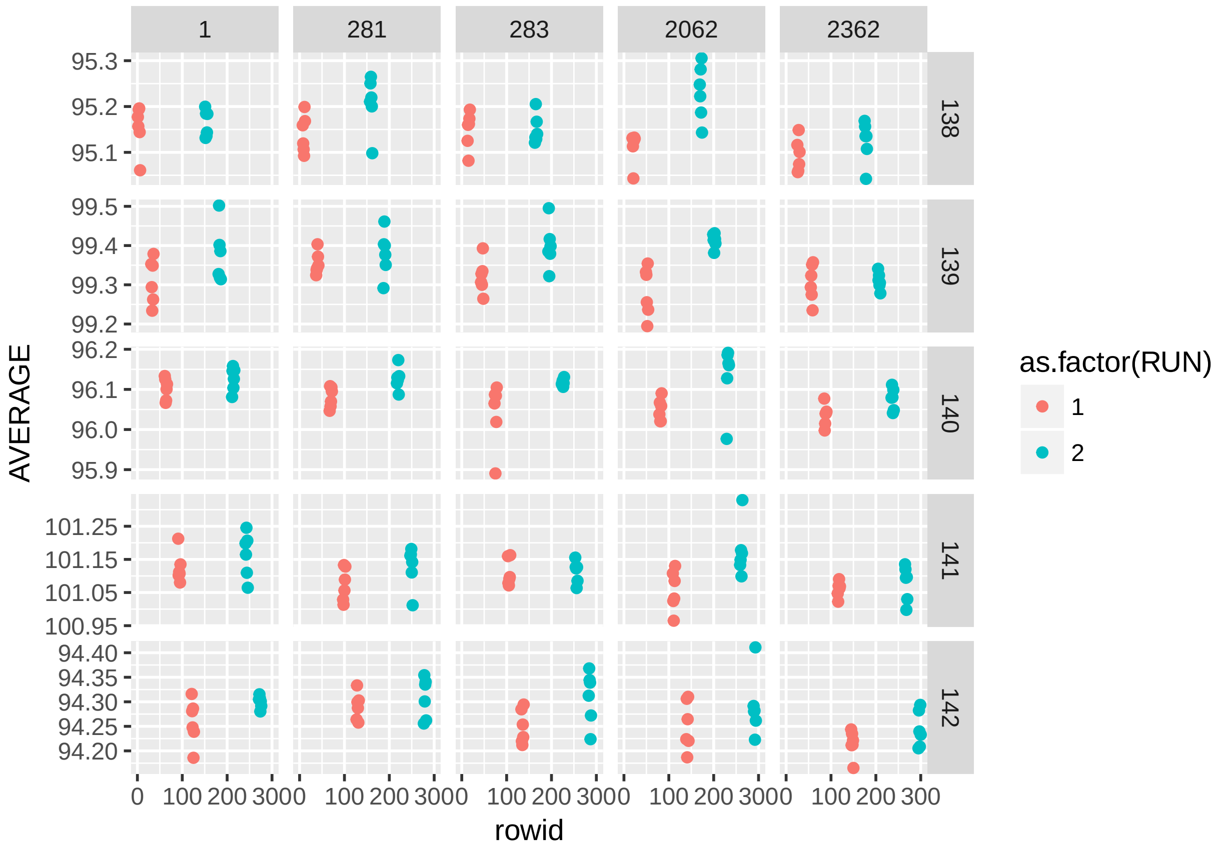


```r
ggplot(gauge_study) +
  geom_point(aes(as.factor(WAFER), STDDEV)) +
  facet_grid(as.factor(RUN) ~ as.factor(PROBE))
```



2.3.2.3 Effects of days and long-term stability

```r
ggplot(gauge_study) +
  geom_point(aes(rowid, AVERAGE, colour = as.factor(RUN))) +
  facet_grid(as.factor(WAFER) ~ as.factor(PROBE), scales = "free_y")
```



2.3.2.4 Differences among 5 probes

```
probe_means_run <- gauge_study %>%
  group_by(PROBE, WAFER, RUN) %>%
  summarise(n = n(), probe_mean = mean(AVERAGE)) %>%
  unite(join_id, WAFER, RUN, sep = "_", remove = FALSE) %>%
  ungroup()

probe_means_run
```

```
## # A tibble: 50 x 6
##    PROBE join_id WAFER   RUN     n probe_mean
##    <dbl> <chr>   <dbl> <int> <int>      <dbl>
##  1    1. 138_1    138.     1     6       95.2
##  2    1. 138_2    138.     2     6       95.2
##  3    1. 139_1    139.     1     6       99.3
##  4    1. 139_2    139.     2     6       99.4
##  5    1. 140_1    140.     1     6       96.1
##  6    1. 140_2    140.     2     6       96.1
##  7    1. 141_1    141.     1     6      101.
##  8    1. 141_2    141.     2     6      101.
##  9    1. 142_1    142.     1     6       94.3
## 10    1. 142_2    142.     2     6       94.3
## # ... with 40 more rows
```

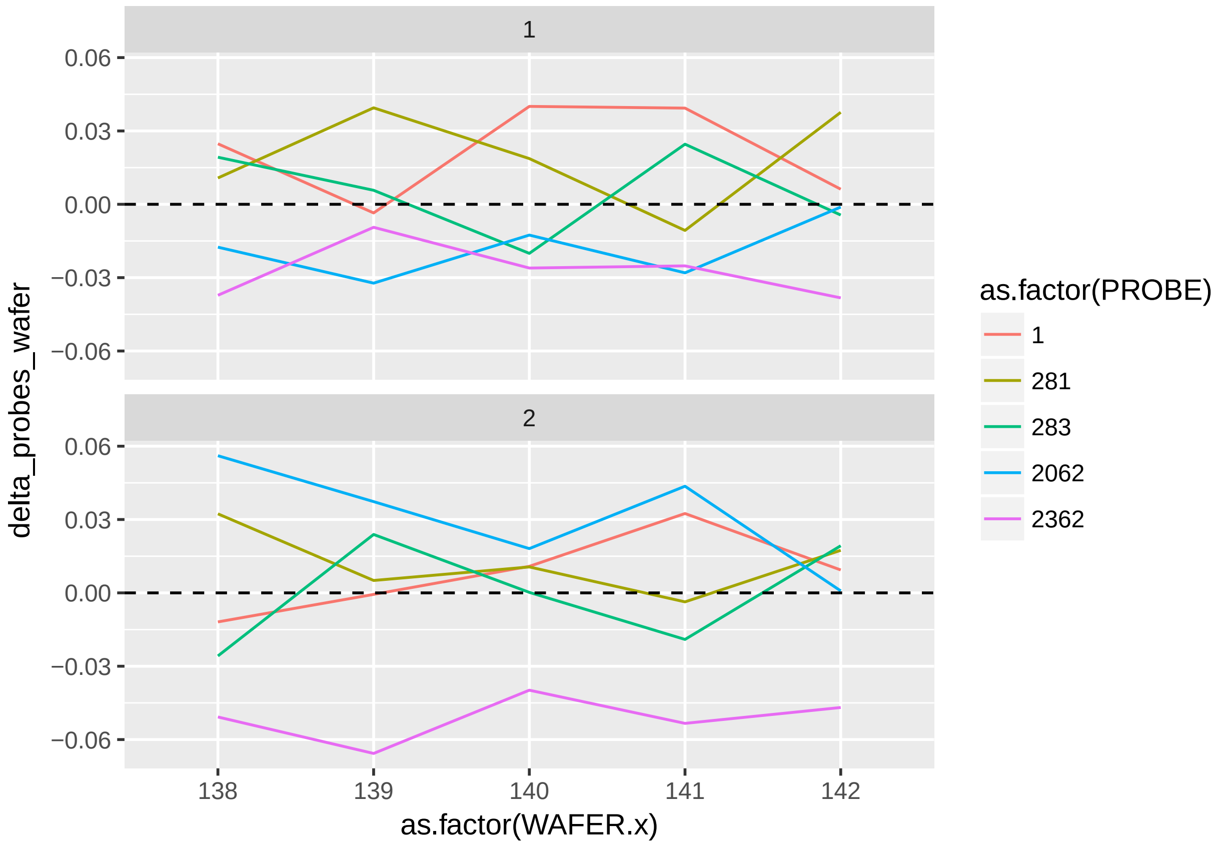
```
wafer_means_run <- gauge_study %>%
  group_by(WAFER, RUN) %>%
  summarise(n = n(), wafer_means = mean(AVERAGE)) %>%
  unite(join_id, WAFER, RUN, sep = "_", remove = FALSE) %>%
  ungroup()

wafer_means_run
```

```
## # A tibble: 10 x 5
##    join_id WAFER   RUN     n wafer_means
##    <chr>   <dbl> <int> <int>       <dbl>
##  1 138_1    138.     1    30        95.1
##  2 138_2    138.     2    30        95.2
##  3 139_1    139.     1    30        99.3
##  4 139_2    139.     2    30        99.4
##  5 140_1    140.     1    30        96.1
##  6 140_2    140.     2    30        96.1
##  7 141_1    141.     1    30       101.
##  8 141_2    141.     2    30       101.
##  9 142_1    142.     1    30        94.3
## 10 142_2    142.     2    30        94.3
```

```
delta_probes <- left_join(probe_means_run, wafer_means_run, by = "join_id") %>%
  mutate(delta_probes_wafer = probe_mean - wafer_means)

delta_probes
```

```
## # A tibble: 50 x 11
##    PROBE join_id WAFER.x RUN.x   n.x probe_mean WAFER.y RUN.y   n.y
##    <dbl> <chr>     <dbl> <int> <int>      <dbl>   <dbl> <int> <int>
##  1    1. 138_1      138.     1     6       95.2    138.     1    30
##  2    1. 138_2      138.     2     6       95.2    138.     2    30
##  3    1. 139_1      139.     1     6       99.3    139.     1    30
##  4    1. 139_2      139.     2     6       99.4    139.     2    30
##  5    1. 140_1      140.     1     6       96.1    140.     1    30
##  6    1. 140_2      140.     2     6       96.1    140.     2    30
##  7    1. 141_1      141.     1     6      101.     141.     1    30
##  8    1. 141_2      141.     2     6      101.     141.     2    30
##  9    1. 142_1      142.     1     6       94.3    142.     1    30
## 10    1. 142_2      142.     2     6       94.3    142.     2    30
## # ... with 40 more rows, and 2 more variables: wafer_means <dbl>,
## #   delta_probes_wafer <dbl>
```

```
ggplot(delta_probes) +
  geom_line(aes(as.factor(WAFER.x), delta_probes_wafer, group = as.factor(PROBE), colour = as.factor(PR
  geom_hline(aes(yintercept = 0), linetype = "dashed") +
  facet_wrap(~ as.factor(RUN.x), ncol = 1)
```



2.3.2.5 Analysis and interpretation

Table of estimates for probe #2362

> A graphical analysis shows repeatability standard deviations plotted by wafer and probe... The plots show that for both runs the precision of this probe is better than for the other probes.
>
> Probe #2362, because of its superior precision, was chosen as the tool for measuring all 100 ohm.cm resistivity wafers at NIST. Therefore, the remainder of the analysis focuses on this probe.

2.3.2.6 probe #2362

```
probe_2362 <- gauge_study %>%
  filter(PROBE == 2362)

probe_2362
```

```
## # A tibble: 60 x 10
##    rowid   RUN WAFER PROBE MONTH   DAY    OP  TEMP AVERAGE STDDEV
##    <int> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>   <dbl>  <dbl>
##  1    25     1  138. 2362.    3.   15.    1.  23.1    95.1 0.0480
##  2    26     1  138. 2362.    3.   17.    1.  23.0    95.1 0.0577
##  3    27     1  138. 2362.    3.   18.    1.  23.0    95.1 0.0516
##  4    28     1  138. 2362.    3.   22.    1.  23.2    95.1 0.0386
##  5    29     1  138. 2362.    3.   23.    2.  23.3    95.1 0.0256
##  6    30     1  138. 2362.    3.    24.    2.  23.1    95.1 0.0420
##  7    55     1  139. 2362.    3.    15.    1.  23.1    99.3 0.0818
##  8    56     1  139. 2362.    3.    17.    1.  23.0    99.3 0.0723
##  9    57     1  139. 2362.    3.    18.    1.  22.9    99.3 0.0756
## 10    58     1  139. 2362.    3.    22.    1.  23.3    99.4 0.0475
## # ... with 50 more rows
```

Pooled level-1 standard deviations (ohm.cm)

```
s1_2362_1 <- probe_2362 %>%
  filter(RUN == 1) %>%
  mutate(Stdev_sq = STDDEV^2) %$%
  mean(Stdev_sq) %>%
  sqrt()

s1_2362_1
```

```
## [1] 0.06750898
```

```
s1_2362_2 <- probe_2362 %>%
  filter(RUN == 2) %>%
  mutate(Stdev_sq = STDDEV^2) %$%
  mean(Stdev_sq) %>%
  sqrt()

s1_2362_2
```

```
## [1] 0.07785664
```

```
s1_2362 <- probe_2362 %>%
  mutate(Stdev_sq = STDDEV^2) %$%
  mean(Stdev_sq) %>%
  sqrt()

s1_2362
```

```
## [1] 0.07286673
```

Level-2 standard deviations (ohm.cm) for 5 wafers

```
s2_2362 <- gauge_study %>%
  group_by(PROBE, WAFER, RUN) %>%
  filter(PROBE == 2362) %>%
  summarise(df = n()-1, probe_mean = mean(AVERAGE), probe_stdev = sd(AVERAGE), probe_stdev_sq = probe_s
  group_by(RUN) %>%
  summarise(s2_run = sqrt(mean(probe_stdev_sq)))

s2_2362
```

```
## # A tibble: 2 x 2
##     RUN s2_run
##   <int>  <dbl>
## 1     1 0.0333
## 2     2 0.0388
```

Over both runs

```
s2_2352_all <- s2_2362 %>%
  mutate(s2_run_sq = s2_run^2) %$%
  mean(s2_run_sq) %>%
  sqrt()

s2_2352_all
```

```
## [1] 0.03616824
```

```
sd_2362_wafer <- gauge_study %>%
  group_by(PROBE, WAFER, RUN) %>%
  filter(PROBE == 2362) %>%
  summarise(probe_mean = mean(AVERAGE)) %>%
  mutate(
    run_number = case_when(
      RUN == 1 ~ "Run1",
      RUN == 2 ~ "Run2"
    )
  ) %>%
  select(PROBE, WAFER, probe_mean, run_number) %>%
  group_by(WAFER) %>%
  summarise(sd_wafer = sd(probe_mean))


sd_2362_wafer
```

```
## # A tibble: 5 x 2
##   WAFER sd_wafer
##   <dbl>    <dbl>
## 1  138.  0.0222
## 2  139.  0.00271
## 3  140.  0.0288
## 4  141.  0.0133
## 5  142.  0.0205
```

```
s3_2362 <- sd_2362_wafer %>%
  mutate(sd_wafer_sq = sd_wafer^2) %$%
  mean(sd_wafer_sq) %>%
  sqrt()

s3_2362
```

```
## [1] 0.01964524
```

Chapter 3

Methods

We describe our methods in this chapter.

```
"code"
```

```
## [1] "code"
```

Chapter 4

Applications

Some *significant* applications are demonstrated in this chapter.

4.1 Example one

4.2 Example two

Chapter 5

Final Words

We have finished a nice book.

Chapter 6

Final Words

We have finished a nice book.

Chapter 7

Final Words

We have finished a nice book.

Chapter 8

Final Words

We have finished a nice book.

Bibliography

Hoobler, R. J., Hutton, M. A., Dillard, M. M., Castellani, M. P., Rheingold, A. L., Rieger, A. L., Rieger, P. H., Richards, T. C., and Geiger, W. E. (1993). Synthesis, characterization, and crystal structure of the chromium complex (.eta.5-C5Ph5)Cr(CO)3 radical. *Organometallics*, 12(1):116–123.

Xie, Y. (2015). *Dynamic Documents with R and knitr.* Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown.* R package version 0.7.