A Case study in Silent Data Corruption in an RNA-Seq Experiment

A bad counts file, an incorrect error message, and counter-intuitive R behavior converge to create a plausible scenario for silent results corruption

Thadryan J. Sweeney

This report describes a series of small issues that can converge such that a `DESeq2` analysis runs from start to finish and raises no errors, but silently corrupts and invalidates the data. The results may or may not seem feasible enough to alert the analyst upon initial inspection. I observed it in practice and detected it using DESeq2's quality control functions. It seemed all-too-plausible that the situation could arise in other projects, so I've documented it for study. If nothing else, it can serve as a reminder of some easily forgotten but counter-intuitive and consequential default behavior of R.

TL;DNR

If a stray character exists in a file being used as a counts matrix, R will read the affected columns of the dataset as `factor`s instead of `integer`s or `numeric`s (in my case, a header row was duplicated several thousand rows into the file, affecting them all).

Though we're passing `factors`, `DESeq2` will raise an error stating that the user has passed `character`s and should be passing `numerics`.

A `character` can be safely cast as an `integer` or `numeric`, but a `factor` cannot; R will silently convert to a ranked sequence, completely invalidating the data (for example, the sequence 1, 100, 1000 is changed to 1, 2, 3).

This new, incorrect dataset will run without errors, but is basically unrelated to what the user intends to analyze.

Note: a bug report has been filed on Bioconductor and the error message updated. Click here for the link.

Steps to Re-create

We will re-create the issues using the sample analysis from the `DESeq2` vignette.

Obtain a counts file

This snippet simply loads a counts file in accordance with the documentation of DESeq2.

```r
library(DESeq2)    # go-to DE analysis package
library(pasilla)   # sample data used in tutorial
library(tidyverse) # data manipulation

# get the counts information from the pasilla package
pasCts <- system.file("extdata", "pasilla_gene_counts.tsv",
                      package = "pasilla", mustWork = TRUE)

# load the sample annotation file from the pasilla package
pasAnno <- system.file("extdata", "pasilla_sample_annotation.csv",
                       package = "pasilla", mustWork = TRUE)
```

```r
# create a matrix of counts
cts <- as.matrix(read.csv(pasCts, sep = "\t", row.names = "gene_id"))

# read in the sample data.
coldata <- read.csv(pasAnno, row.names = 1)

# select the desired features (just following the tutorial)
coldata <- coldata[, c("condition","type")]

# clean/standardize the rownames
rownames(coldata) <- sub("fb", "", rownames(coldata))

# sort/reorder the columns to match samples
cts <- cts[, rownames(coldata)]
```

Introduce a character

I found the issue after reading in a bad counts matrix file that was given to me. Somewhere in pre-processing a header of some kind had gotten duplicated and nestled a few thousand rows into the counts file. It looked something like this:

```
sampleName sampleName sampleName sampleName sampleName sampleName sampleName
```

Perhaps tables had been stacked on top of one another to make the counts file and there was an off-by-one error, I don't know, but it is an easy error to imagine making in Bioinformatics and data-related computing in general. It can be reproduced like this:

```r
ctsBad <- cts

# add a character row
ctsBad[8600, ] <- c("here", "there", "and", "everywhere", "yeah", "whoo!", "oops!")

# inspect
ctsBad[8597:8602, ]
```

```
##             treated1 treated2 treated3 untreated1   untreated2 untreated3
## FBgn0037427 "0"      "0"      "0"      "0"          "0"        "0"
## FBgn0037428 "2"      "0"      "0"      "0"          "0"        "0"
## FBgn0037429 "171"    "118"    "101"    "121"        "204"      "85"
## FBgn0037430 "here"   "there"  "and"    "everywhere" "yeah"     "whoo!"
## FBgn0037431 "9"      "2"      "1"      "3"          "7"        "1"
## FBgn0037432 "9"      "2"      "6"      "6"          "17"       "4"
##             untreated4
## FBgn0037427 "0"
## FBgn0037428 "0"
## FBgn0037429 "103"
## FBgn0037430 "oops!"
## FBgn0037431 "3"
## FBgn0037432 "7"
```

```r
# write to a file
write.csv(ctsBad, "badCounts.csv")
```

When we read the data from a file, the well-known-but-still-menacing factor default issues comes in to play. However, `DESeq2` appears to misdiagnose the types. To see how, we first read in the file with the bad row.

```
ctsBadFile <- read.delim("badCounts.csv", sep = ",")

# we're no longer getting integers
sapply(ctsBadFile, class)
```

```
##           X    treated1    treated2    treated3  untreated1  untreated2  untreated3
##    "factor"    "factor"    "factor"    "factor"    "factor"    "factor"    "factor"
##  untreated4
##    "factor"
```

```
ctsBadFile[8597:8602, ]
```

```
##                 X treated1 treated2 treated3 untreated1 untreated2 untreated3
## 8597 FBgn0037427        0        0        0          0          0          0
## 8598 FBgn0037428        2        0        0          0          0          0
## 8599 FBgn0037429      171      118      101        121        204         85
## 8600 FBgn0037430     here    there      and everywhere       yeah      whoo!
## 8601 FBgn0037431        9        2        1          3          7          1
## 8602 FBgn0037432        9        2        6          6         17          4
##      untreated4
## 8597          0
## 8598          0
## 8599        103
## 8600      oops!
## 8601          3
## 8602          7
```

This behavior is consistently tricky, but well-documented (and thankfully, changing). It becomes a more complex issue when mixed with a misleading error message and some more counter-intuitive bahavior.

Call DESeqDataSetFromMatrix()

We can now observe that passing `factor`s results in being warned we are passing `character`s.

```
# make the input the corrected sized matrix
ctsBadFile$X <- NULL

# demonstrate that we're passing factors
sapply(ctsBadFile, class)
```

```
##   treated1   treated2   treated3 untreated1 untreated2 untreated3 untreated4
##   "factor"   "factor"   "factor"   "factor"   "factor"   "factor"   "factor"
```

```
error <- tryCatch(
  {
    # try to use the bad one
    dds <- DESeqDataSetFromMatrix(countData = ctsBadFile,
                                  colData = coldata, design = ~ condition)
  },
  # "upon error 'e', use this function to show a message of 'e'"
  error = function(e) { return(e)}
)

# show the error messagge (just splitting it because it is long)
errorMessage <- unlist(str_split(error, ":"))
```

```
## Warning in stri_split_regex(string, pattern, n = n, simplify = simplify, :
```

```
## argument is not an atomic vector; coercing
```

```
cat("\n", errorMessage[2], errorMessage[3])
```

```
##
##   counts matrix should be numeric, currently it has mode  character
```

We see that `DESeq2` informs us we're passing `characters`, though we have actually passed `factors` (a detailed possible explanation of why this happens is presented in the analysis included with the bug report. Click here) to view it. The implied fix is to convert the dataset to `numerics` or `integers`. However, either of those steps result in silent data corruption because `factors` cannot be coerced to `integers` or `numerics` safely even though `characters` can. The problem is demonstrated below (an atomic, reproducible example of the related behavior is included at the bottom of the report).

```
# apply the fix insinutated by the error message
ctsBadFileAsInt <- ctsBadFile %>%
  mutate_all(as.integer)

ctsBadFileAsNumeric <- ctsBadFile %>%
  mutate_all(as.numeric)
```

We can then see the dataset before and after coercion. Before:

```
# inspect the original and factor versions
cts %>% head()
```

```
##             treated1 treated2 treated3 untreated1 untreated2 untreated3
## FBgn0000003        0        0        1          0          0          0
## FBgn0000008      140       88       70         92        161         76
## FBgn0000014        4        0        0          5          1          0
## FBgn0000015        1        0        0          0          2          1
## FBgn0000017     6205     3072     3334       4664       8714       3564
## FBgn0000018      722      299      308        583        761        245
##             untreated4
## FBgn0000003          0
## FBgn0000008         70
## FBgn0000014          0
## FBgn0000015          2
## FBgn0000017       3150
## FBgn0000018        310
```

```
ctsBadFile %>% head()
```

```
##   treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
## 1        0        0        1          0          0          0          0
## 2      140       88       70         92        161         76         70
## 3        4        0        0          5          1          0          0
## 4        1        0        0          0          2          1          2
## 5     6205     3072     3334       4664       8714       3564       3150
## 6      722      299      308        583        761        245        310
```

After:

```
ctsBadFileAsInt %>% head()
```

```
##   treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
## 1        1        1        2          1          1          1          1
## 2      485     2285     2139       2847        712       2011       2033
## 3     2133        1        1       2144          2          1          1
## 4        2         1         1           1         1046           2         813
## 5     2772      1281      1413        2066         3368        1343        1290
## 6     2966      1251      1326        2325         3172         978        1273
```

```
ctsBadFileAsNumeric %>% head()
```

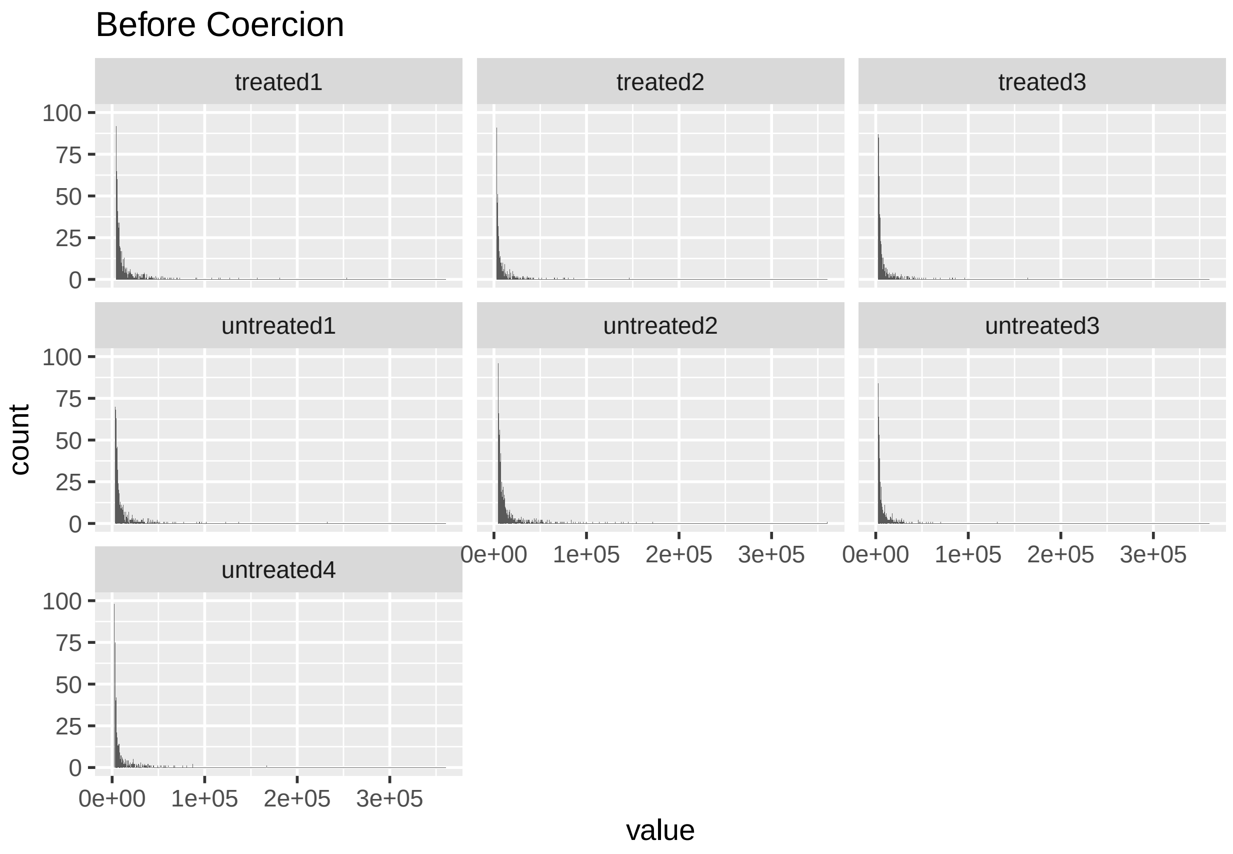
```
##   treated1 treated2 treated3 untreated1 untreated2 untreated3 untreated4
## 1        1         1         2           1            1           1           1
## 2      485      2285      2139        2847          712        2011        2033
## 3     2133         1         1        2144            2           1           1
## 4        2         1         1           1         1046           2         813
## 5     2772      1281      1413        2066         3368        1343        1290
## 6     2966      1251      1326        2325         3172         978        1273
```

We can get some intuition for the shift with a quick, if hacky, plot:

```
bin <- 1000

# the original dataset
ggplot(gather(data.frame(cts)), aes(value)) +
    geom_histogram(bins = bin) +
    facet_wrap(~key) +
    ylim(0, 100) +
    ggtitle("Before Coercion")
```

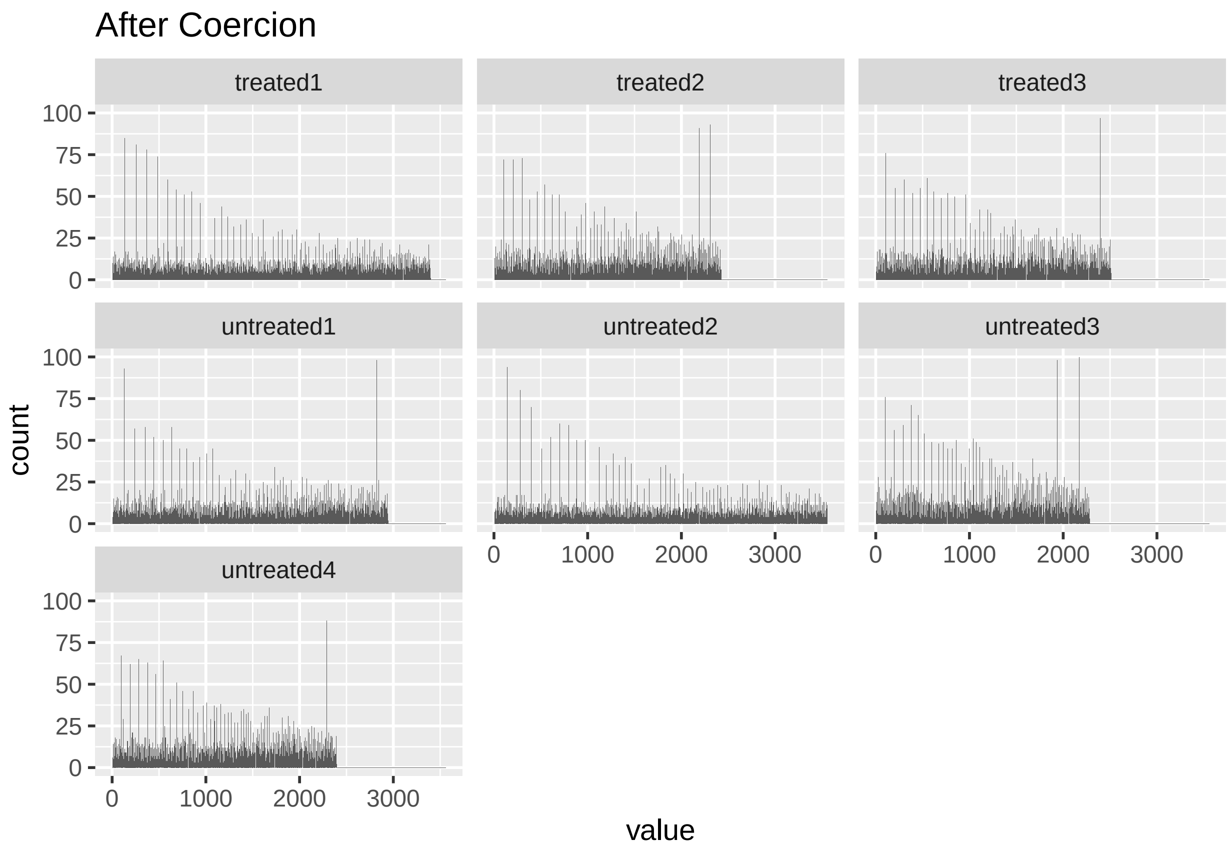
```
## Warning: Removed 67 rows containing missing values (geom_bar).
```



```
# the corrupted one
ggplot(gather(data.frame(ctsBadFileAsInt)), aes(value)) +
    geom_histogram(bins = bin) +
```

```r
    facet_wrap(~key) +
    ylim(0, 100) +
    ggtitle("After Coercion")
```

```
## Warning: Removed 63 rows containing missing values (geom_bar).
```



Either of these corrupted datasets will run without error:

```r
ddsBadInt <- DESeqDataSetFromMatrix(countData = ctsBadFileAsInt,
                                    colData = coldata, design = ~ condition)
```

This warns that the numerics are being converted to integer (a safe operation).

```r
ddsBadNumeric <- DESeqDataSetFromMatrix(countData = ctsBadFileAsNumeric,
                                        colData = coldata, design = ~ condition)
```

```
## converting counts to integer mode
```

Prevention

I have taken to including a line like this in my read-in scripts:

`stopifnot(is.null(names(Filter(is.factor, ctsBadFile))))`

```r
tryCatch(
  {
    stopifnot(is.null(names(Filter(is.factor, ctsBadFile))))
  },
  error = function(m){ print(m) }
)
```

```
## <simpleError: is.null(names(Filter(is.factor, ctsBadFile))) is not TRUE>
```

Comments

This is not a criticism of `DESeq2`. This is not to say that the analyst is not responsible for understanding their types. It is merely an effort to identify a potential source of error and reduce it. It's not hard to imagine a stray character row finding it's way into a counts file, and a hurried analyst not noticing and following a misleading error. I propose that it is especially important to be vigilant about type-related bugs given that many R users are investigators from other disciples who may not have experience programming, and even if they do, are likely to have it in a language where type-awareness is not necessarily required or promoted (as in R itself).

I am aware that the eternal confounder `stringsAsFactors=TRUE` was addressed in the newest release of R, which is progress. It makes this scenario markedly less likely, though not impossible. The other issues addressed are still worth refreshing on, especially the silent coercion issue.

Atomic Example of Factor Behavior

An atomic example of this troubling behavior may illuminate the issue. `character`s can be safely coerced to `integer`s, but `factor`s cannot. Moreover, they fail silently and return invalid results. We demonstrate below for the sake of completeness:

```r
# a vector of integers
x <- c("1", "10", "100", "1000")

as.integer(x)
```

```
## [1]    1   10  100 1000
```

When you do that to a vector of `factor`s however, you silently receive a counter-intuitive result:

```r
# the same, as a factor
x <- factor(c("1", "10", "100", "1000"))

as.integer(x)
```

```
## [1] 1 2 3 4
```

Session Info

```r
sessionInfo()
```

```
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Pop!_OS 19.10
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

```
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] forcats_0.5.0               stringr_1.4.0
##  [3] dplyr_0.8.5                 purrr_0.3.3
##  [5] readr_1.3.1                 tidyr_1.0.2
##  [7] tibble_3.0.0                ggplot2_3.3.0
##  [9] tidyverse_1.3.0             pasilla_1.14.0
## [11] DESeq2_1.26.0               SummarizedExperiment_1.16.1
## [13] DelayedArray_0.12.2         BiocParallel_1.20.1
## [15] matrixStats_0.56.0          Biobase_2.46.0
## [17] GenomicRanges_1.38.0        GenomeInfoDb_1.22.0
## [19] IRanges_2.20.2              S4Vectors_0.24.3
## [21] BiocGenerics_0.32.0
##
## loaded via a namespace (and not attached):
##  [1] colorspace_1.4-1       ellipsis_0.3.0         htmlTable_1.13.3
##  [4] XVector_0.26.0         base64enc_0.1-3        fs_1.4.1
##  [7] rstudioapi_0.11        farver_2.0.3           bit64_0.9-7
## [10] AnnotationDbi_1.48.0   fansi_0.4.1            lubridate_1.7.8
## [13] xml2_1.3.0             splines_3.6.1          geneplotter_1.64.0
## [16] knitr_1.28             Formula_1.2-3          jsonlite_1.6.1
## [19] broom_0.5.5            annotate_1.64.0        cluster_2.1.0
## [22] dbplyr_1.4.2           png_0.1-7              compiler_3.6.1
## [25] httr_1.4.1             backports_1.1.6        assertthat_0.2.1
## [28] Matrix_1.2-18          cli_2.0.2              formatR_1.7
## [31] acepack_1.4.1          htmltools_0.4.0        tools_3.6.1
## [34] gtable_0.3.0           glue_1.4.0             GenomeInfoDbData_1.2.2
## [37] Rcpp_1.0.4             cellranger_1.1.0       vctrs_0.2.4
## [40] nlme_3.1-144           xfun_0.12              rvest_0.3.5
## [43] lifecycle_0.2.0        XML_3.99-0.3           zlibbioc_1.32.0
## [46] scales_1.1.0           hms_0.5.3              RColorBrewer_1.1-2
## [49] yaml_2.2.1             memoise_1.1.0          gridExtra_2.3
## [52] rpart_4.1-15           latticeExtra_0.6-29    stringi_1.4.6
## [55] RSQLite_2.2.0          genefilter_1.68.0      checkmate_2.0.0
## [58] rlang_0.4.5            pkgconfig_2.0.3        bitops_1.0-6
## [61] evaluate_0.14          lattice_0.20-38        htmlwidgets_1.5.1
## [64] labeling_0.3           bit_1.1-15.2           tidyselect_1.0.0
## [67] magrittr_1.5           R6_2.4.1               generics_0.0.2
## [70] Hmisc_4.4-0            DBI_1.1.0              pillar_1.4.3
## [73] haven_2.2.0            foreign_0.8-75         withr_2.1.2
## [76] survival_3.1-11        RCurl_1.98-1.1         nnet_7.3-12
## [79] modelr_0.1.6           crayon_1.3.4           rmarkdown_2.1
## [82] jpeg_0.1-8.1           locfit_1.5-9.4         grid_3.6.1
## [85] readxl_1.3.1           data.table_1.12.8      blob_1.2.1
## [88] reprex_0.3.0           digest_0.6.25          xtable_1.8-4
## [91] munsell_0.5.0
```