DESeq2 Factor Error Message

After being sent some bad data, I observed what I think is a bug in which DESeq2 misdiagnoses user input containing factors. The error does no harm in and of itself, but if a user takes what seemss a reasonable step based on the error message, R's type system will silently invalidate the results. We will need a basic counts file to demonstrate and reproduce the issues, so we will recreate the one from the vignette:

TL;DNR

When using DESeq2 Passing a `data.frame` containing a column of type `factor` results in an error message stating the user has entered a column of type `character`. A `character` can be safely cast as an `integer`, but a `factor` cannot; R will silently change them to an ordered sequence, completely corrupting and invalidating the data (ie, the sequence 1,100,1000 is changed to 1,2,3). If a user forgets to set `stringsAsFactors = FALSE` and trusts the error message, they can essentially randomized thier results without R or DESeq2 warning them. The user is ultimately responsible for their types. However given that R promotes the accidental use of factors, has default behavior that corrupts such data silently, and that R users are often investigators from other disciplines who may not have experience coding, steps to reduce the chance of error may be helpful.

Steps to Re-create

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

# 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)

# 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)]
```

I found the error 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 be stacked on top of one another to make the counts file and there was an off-by-one error, I don't know. It can be reproduced like this:

```
ctsBad <- cts

ctsBad[8600, ] <- c("here", "there", "and", "everywhere", "yeah", "whoo!", "oops!")

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"
```

```
write.csv(ctsBad, "badCounts.csv")
```

When we write the data to a file, the well-know-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.

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

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
```

Though we're passing factors. we're told we are passing characters.

```
# 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"
```

```
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) { message(e)}
)
```

The real issue is that characters can be safely coerced to integers, but factors cannot. Moreover, they fail silently and return invalid results. We demonstrate below for the sake of completeness:

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

as.integer(x)
```

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

When you do that to a list of factors however, you get a deeply problematic and counter-intuitive result:

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

as.integer(x)
```

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

The output is effectively unrelated to the input. Given the error message, it's tempting to think one should just change the characters to integers, which is a normal and safe operation:

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

However, we don't actually have characters, we have factors, so we coerce our data to something only vaguely based on the actual values:

```
# observe results
message("original")
```

```
## original
```

```
cts %>% head(10)
```

```
##             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
## FBgn0000022         0        0        0          0          1          0
## FBgn0000024        10        7        5         10         11          3
## FBgn0000028         0        1        1          0          1          0
## FBgn0000032      1698      696      757       1446       1713        615
##             untreated4
## FBgn0000003          0
## FBgn0000008         70
## FBgn0000014          0
## FBgn0000015          2
## FBgn0000017       3150
## FBgn0000018        310
## FBgn0000022          0
## FBgn0000024          3
## FBgn0000028          0
## FBgn0000032        672
```

```
message("bad file")
```

```
## bad file
```

```
ctsBadFile %>% head(10)
```

```
##    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
## 7         0        0        0          0          1          0          0
## 8        10        7        5         10         11          3          3
## 9         0        1        1          0          1          0          0
## 10     1698      696      757       1446       1713        615        672
```

```
message("after coercion")
```

```
## after coercion
```

```
ctsBadFileAsInt %>% head(10)
```

```
##    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
## 7         1        1        1          1          2          1          1
## 8         3     2061     1824          3        142       1176       1237
## 9         1        2        2          1          2          1          1
## 10      767     2055     2213        489        809       1823       1999
```

DESeq2 has excellent documentation, and during routine QC check presented in the vignette, I identified the problem. In the case presented here, the plotMA function makes it clear that something is wrong, but on my initial dataset, the graphs looked passable.

Root Cause

Inspecting the source code of `DESeqDataSetFromMatrix` , we see that there is a coercion of input to type `matrix`, which is reasonable given that most users will pass a `dataframe`. However, passing a `factor` to `as.matrix` will results in another silent conversion of `factor` to `character`. Because DESeq2 checks the types *after* coercion of the inputs, the message given to users the current state after modification, not the nature of the input as it was given. This can be illustrated with a modification to `DESeq2` source code, presented below (my own comments are denoted `### --->` to distinguish them from those of the original author).

```
DESeqDataSetFromMatrixDebug <- function( countData, colData, design, tidy=FALSE, ignoreRank=FALSE, ... 
{

  ### ---> a function to observe types at a given time point
  printDebugOutput <- function(tag)
  {
    ### ---> so we can pass an identifier of some kind
    message(tag)
    ### ---> what is the class of the count data at this point?
    print(paste("class of countData: ", class(countData)))
    ### ---> what are the classes of each of the columns? (truncated so it isn't too long)
    print(sapply(countData, class)[1:10])
  }

  ### --->  inital types
  printDebugOutput("initial")

  if (tidy) {
    stopifnot(ncol(countData) > 1)
    rownms <- as.character(countData[,1])
    countData <- countData[,-1,drop=FALSE]
    rownames(countData) <- rownms
  }

  # check that these agree in number
  stopifnot(ncol(countData) == nrow(colData))

  ### ---> there is a coercion to type matrix
  # we expect a matrix of counts, which are non-negative integers
  countData <- as.matrix( countData )

  ### ---> reobserve the types
  printDebugOutput("after coercion to matrix")

  ### ---> the code is unchanged after this point



  if (is(colData,"data.frame"))
    colData <- as(colData, "DataFrame")
  # check if the rownames of colData are simply in different order
  # than the colnames of the countData, if so throw an error
  # as the user probably should investigate what's wrong
  if (!is.null(rownames(colData)) & !is.null(colnames(countData))) {
    if (all(sort(rownames(colData)) == sort(colnames(countData)))) {
      if (!all(rownames(colData) == colnames(countData))) {
        stop(paste("rownames of the colData:
  ",paste(rownames(colData),collapse=","),"
  are not in the same order as the colnames of the countData:
  ",paste(colnames(countData),collapse=",")))
      }
    }
  }
  if (is.null(rownames(colData)) & !is.null(colnames(countData))) {
    rownames(colData) <- colnames(countData)
  }

  se <- SummarizedExperiment(assays = SimpleList(counts=countData), colData = colData, ...)
  object <- DESeqDataSet(se, design = design, ignoreRank)

  return(object)
}
```

We can use this function to check the types:

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

```
## initial

## [1] "class of countData:  data.frame"
##    treated1    treated2    treated3 untreated1 untreated2 untreated3 untreated4
##    "factor"    "factor"    "factor"    "factor"    "factor"    "factor"    "factor"
##        <NA>        <NA>        <NA>
##          NA          NA          NA

## after coercion to matrix

## [1] "class of countData:  matrix"
##           0         140           4           1        6205         722
## "character" "character" "character" "character" "character" "character"
##           0          10           0        1698
## "character" "character" "character" "character"
```

This behavior can be confirmed in base R.

```
class(iris[,5])
```

```
## [1] "factor"
```

```r
class(as.matrix(iris)[,5])
```

```
## [1] "character"
```

The actual warning is raised in a separate function called `DESeqDataSet`, which is wrapped by `DESeqDataSetFromMatrix`. `DESeqDataSetFromMatrix`, however, could check for factors (or do whatever type-checking desired) before passing the input to `DESeqDataSet`. An example might look like this:

```r
DESeqDataSetFromMatrixDebugFactorSafe <- function( countData, colData, design, tidy=FALSE, ignoreRank=F
{

  ### ---> checking for factors initially
  if(length(which(sapply(countData, is.factor))) != 0){
    stop(
      paste("Error: inputs of type factor cannot be safely coerced to type integer.",
            "Please inspect input, and convert factors to characters before casting as integer")
    )
  }
  ### ---> the code is unchanged after this point

  if (tidy) {
    stopifnot(ncol(countData) > 1)
    rownms <- as.character(countData[,1])
    countData <- countData[,-1,drop=FALSE]
    rownames(countData) <- rownms
  }

  # check that these agree in number
  stopifnot(ncol(countData) == nrow(colData))

  # we expect a matrix of counts, which are non-negative integers
  countData <- as.matrix( countData )

  if (is(colData,"data.frame"))
    colData <- as(colData, "DataFrame")

  # check if the rownames of colData are simply in different order
  # than the colnames of the countData, if so throw an error
  # as the user probably should investigate what's wrong
  if (!is.null(rownames(colData)) & !is.null(colnames(countData))) {
    if (all(sort(rownames(colData)) == sort(colnames(countData)))) {
      if (!all(rownames(colData) == colnames(countData))) {
        stop(paste("rownames of the colData:
  ",paste(rownames(colData),collapse=","),"
  are not in the same order as the colnames of the countData:
  ",paste(colnames(countData),collapse=",")))
      }
    }
  }
  if (is.null(rownames(colData)) & !is.null(colnames(countData))) {
    rownames(colData) <- colnames(countData)
  }

  se <- SummarizedExperiment(assays = SimpleList(counts=countData), colData = colData, ...)
  object <- DESeqDataSet(se, design = design, ignoreRank)
  return(object)
}
```

This should prevent the error without interfering with the normal function of the program:

```r
# do we catch the bad one?
tryCatch(
  {
    # try to use the bad one
    dds <- DESeqDataSetFromMatrixDebugFactorSafe(countData = ctsBadFileAsInt,
                                                 colData = coldata, design = ~ condition)
  },
  error = function(e) { message(e)}
)

dds <- DESeqDataSetFromMatrixDebugFactorSafe(countData = cts,
                                             colData = coldata, design = ~ condition)

# do we allow the good one?
DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing

## class: DESeqDataSet
## dim: 14599 7
## metadata(1): version
## assays(4): counts mu H cooks
## rownames(14599): FBgn0000003 FBgn0000008 ... FBgn0261574 FBgn0261575
## rowData names(22): baseMean baseVar ... deviance maxCooks
## colnames(7): treated1 treated2 ... untreated3 untreated4
## colData names(3): condition type sizeFactor
```