# Tackling Formatted Tabular Data from Excel

*10th July 2024*

---

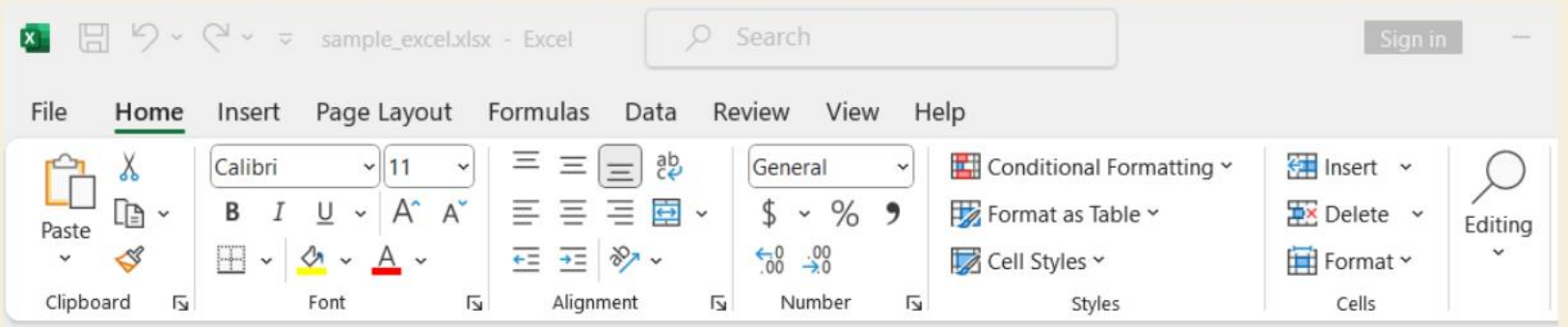
Jeremy Selva
@JauntyJJS
*https://jeremy-selva.netlify.app*
For useR! 2024

# Formatted Cell in Excel

# Formatted Cell in Excel

Formatted cells are useful for clinicians to make highlight important information. They tend to be well received by people but not so for software.

Here is one called sample_excel.xlsx.

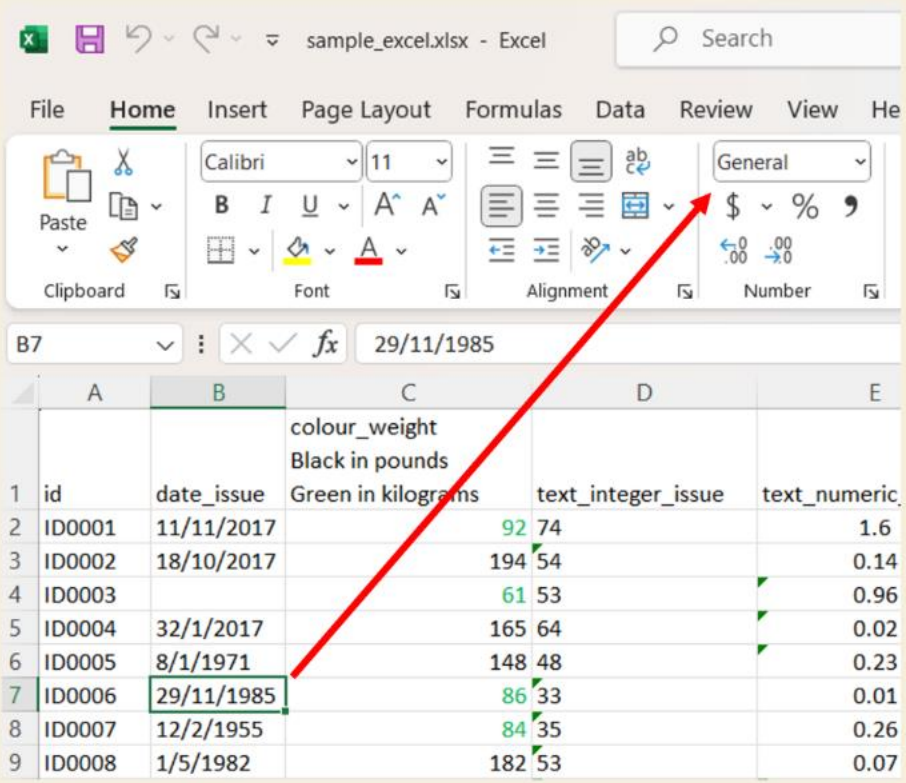



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | id | date_issue | colour_weight<br>Black in pounds<br>Green in kilograms | text_integer_issue | text_numeric_issue | numeric_integer_issue | one_or_zero_issue |
| 2 | ID0001 | 11/11/2017 | 92 | 74 | 1.6 | 1 | |
| 3 | ID0002 | 18/10/2017 | 194 | 54 | 0.14 | 55 | |
| 4 | ID0003 | | 61 | 53 | 0.96 | 9 | |
| 5 | ID0004 | 32/1/2017 | 165 | 64 | 0.02 | 2 | |
| 6 | ID0005 | 8/1/1971 | 148 | 48 | 0.23 | 3 | |
| 7 | ID0006 | 29/11/1985 | 86 | 33 | 0.01 | 7 | |
| 8 | ID0007 | 12/2/1955 | 84 | 35 | 0.26 | 1 | |
| 9 | ID0008 | 1/5/1982 | 182 | 53 | 0.07 | 3 | |
| 10 | ID0009 | 20/4/1969 | 80 | 187 | 0.06 | 75 | |
| 11 | ID0010 | 21/11/1962 | 78 | 141 | 0.01 | 23 | |

# Formatted Cell in Excel

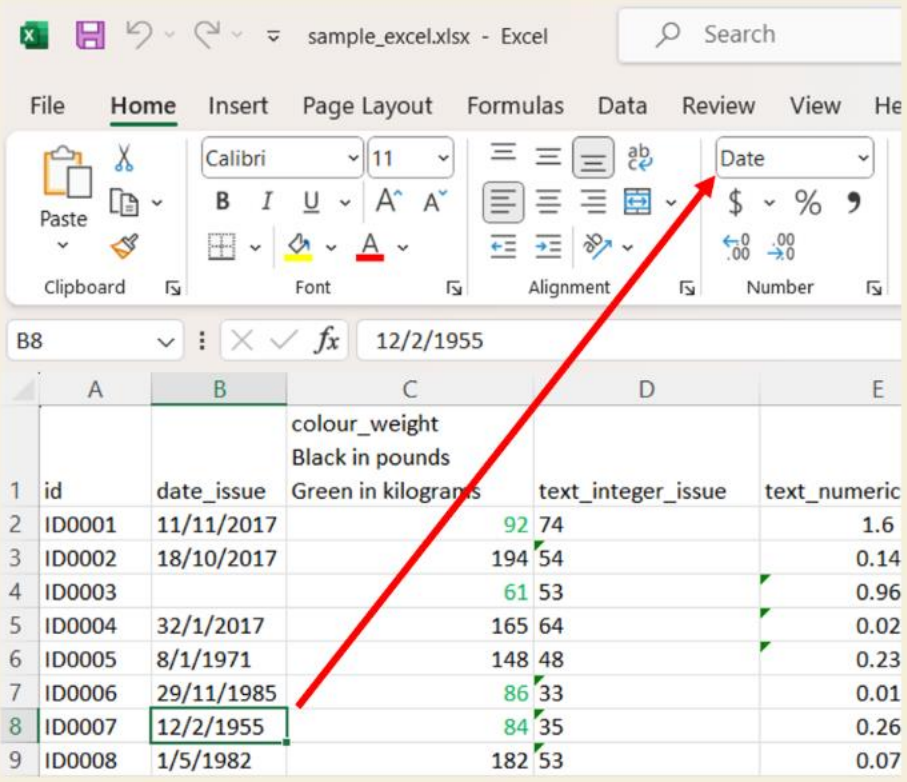
The column `date_issue` has two format.

One format is in **General**




| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric |
| 2 | ID0001 | 11/11/2017 | 92 | 74 | 1.6 |
| 3 | ID0002 | 18/10/2017 | 194 | 54 | 0.14 |
| 4 | ID0003 | | 61 | 53 | 0.96 |
| 5 | ID0004 | 32/1/2017 | 165 | 64 | 0.02 |
| 6 | ID0005 | 8/1/1971 | 148 | 48 | 0.23 |
| 7 | ID0006 | 29/11/1985 | 86 | 33 | 0.01 |
| 8 | ID0007 | 12/2/1955 | 84 | 35 | 0.26 |
| 9 | ID0008 | 1/5/1982 | 182 | 53 | 0.07 |

The other format is in **Date**

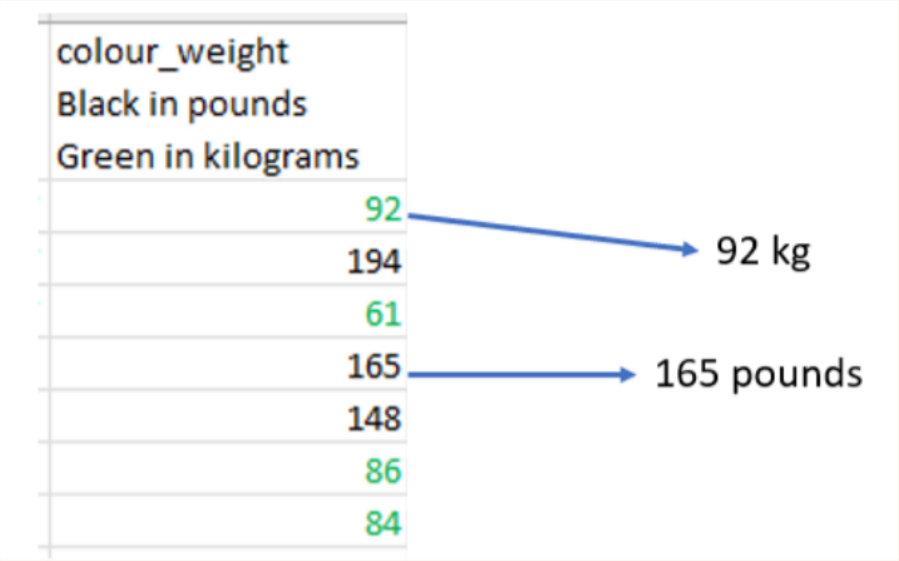



| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric |
| 2 | ID0001 | 11/11/2017 | 92 | 74 | 1.6 |
| 3 | ID0002 | 18/10/2017 | 194 | 54 | 0.14 |
| 4 | ID0003 | | 61 | 53 | 0.96 |
| 5 | ID0004 | 32/1/2017 | 165 | 64 | 0.02 |
| 6 | ID0005 | 8/1/1971 | 148 | 48 | 0.23 |
| 7 | ID0006 | 29/11/1985 | 86 | 33 | 0.01 |
| 8 | ID0007 | 12/2/1955 | 84 | 35 | 0.26 |
| 9 | ID0008 | 1/5/1982 | 182 | 53 | 0.07 |

# Formatted Cell in Excel

The column `colour_weight` has two colour format.

- Cells in **black** are weight in pounds
- Cells in **green** are weight in kilogram




The columns `text_integer_issue` and `text_numeric_issue` are numeric columns but some cells were formatted as text. These cells are indicated by the **green** triangle.

| text_integer_issue | text_numeric_issue |
|---|---|
| 74 | 1.6 |
| 54 | 0.14 |
| 53 | 0.96 |
| 64 | 0.02 |
| 48 | 0.23 |
| 33 | 0.01 |
| 35 | 0.26 |
| 53 | 0.07 |
| 187 | 0.06 |

# Formatted Cell in Excel

The columns `numeric_integer_issue` and `one_or_zero_issue` are numeric columns.

- `numeric_integer_issue` has only positive integer values
- `one_or_zero_issue` has only values 0 and 1 but has missing values on the first few hundred rows

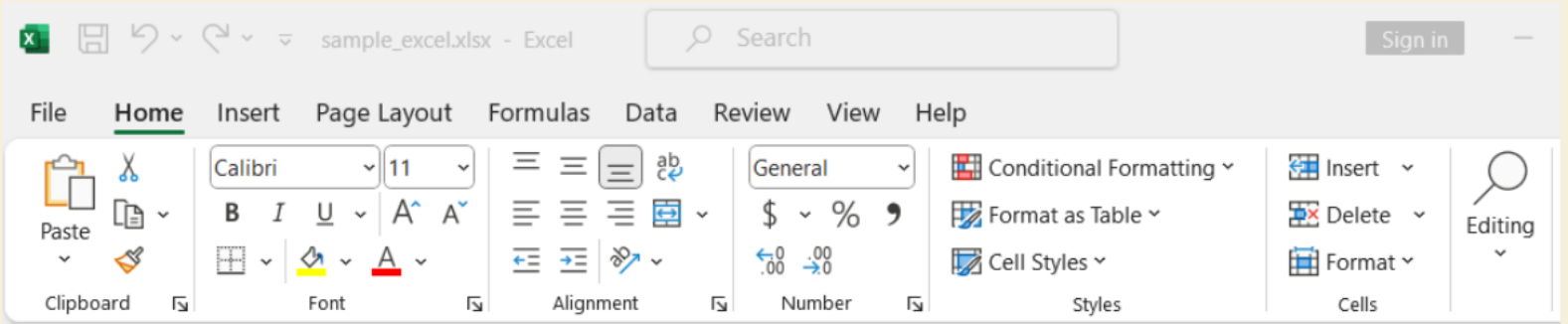
| | A | F | G |
|---|---|---|---|
| 1 | id | numeric_integer_issue | one_or_zero_issue |
| 1046 | ID1045 | 96 | |
| 1047 | ID1046 | 95 | |
| 1048 | ID1047 | 34 | 1 |
| 1049 | ID1048 | 57 | 0 |
| 1050 | ID1049 | 37 | 1 |
| 1051 | ID1050 | 83 | 1 |
| 1052 | ID1051 | 11 | 1 |
| 1053 | ID1052 | 28 | 0 |
| 1054 | ID1053 | 32 | 0 |

# Read Data Attempts

# Read Data Attempt 1

Tried to read the data using *readxl::read_excel*. No warning was provided but…

```
1 sample_excel_attempt_1 <- readxl::read_excel(
2   path = here::here("sample_excel.xlsx"),
3   sheet = "Sheet1"
4 )
```




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | id | date_issue | colour_weight<br>Black in pounds<br>Green in kilograms | text_integer_issue | text_numeric_issue | numeric_integer_issue | one_or_zero_issue |
| 2 | ID0001 | 11/11/2017 | 92 | 74 | 1.6 | 1 | |
| 3 | ID0002 | 18/10/2017 | 194 | 54 | 0.14 | 55 | |
| 4 | ID0003 | | 61 | 53 | 0.96 | 9 | |
| 5 | ID0004 | 32/1/2017 | 165 | 64 | 0.02 | 2 | |
| 6 | ID0005 | 8/1/1971 | 148 | 48 | 0.23 | 3 | |
| 7 | ID0006 | 29/11/1985 | 86 | 33 | 0.01 | 7 | |
| 8 | ID0007 | 12/2/1955 | 84 | 35 | 0.26 | 1 | |
| 9 | ID0008 | 1/5/1982 | 182 | 53 | 0.07 | 3 | |
| 10 | ID0009 | 20/4/1969 | 80 | 187 | 0.06 | 75 | |
| 11 | ID0010 | 21/11/1962 | 78 | 141 | 0.01 | 23 | |

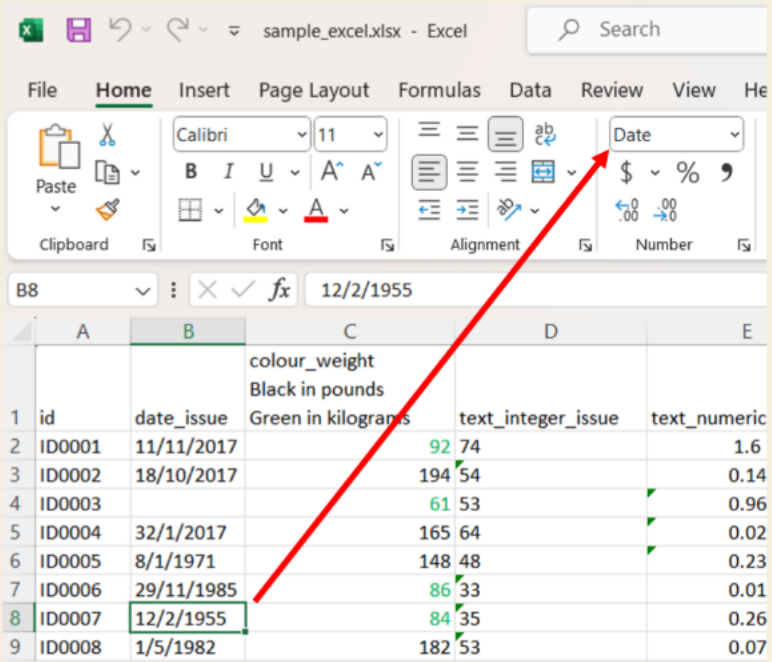
# Read Data Attempt 1

Here is the output

| id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric_issue |
|---|---|---|---|---|
| ID0001 | 11/11/2017 | 92 | 74 | 1.6 |
| ID0002 | 18/10/2017 | 194 | 54 | 0.14000000000000001 |
| ID0003 | | 61 | 53 | 0.96 |
| ID0004 | 32/1/2017 | 165 | 64 | 0.02 |
| ID0005 | 25941 | 148 | 48 | 0.23 |
| ID0006 | 29/11/1985 | 86 | 33 | 0.01 |
| ID0007 | 20132 | 84 | 35 | 0.26 |
| ID0008 | 30072 | 182 | 53 | 7.0000000000000007E-2 |

1–8 of 1053 rows Previous 1 of 132 Next

- `date_issue`: those formatted as Date have been turned to numbers
- `colour_weight`: different colour inputs not differentiated



# Read Data Attempt 1

```
1 str(sample_excel_attempt_1)
```

```
tibble [1,053 × 7] (S3: tbl_df/tbl/data.frame)
 $ id                                                :
chr [1:1053] "ID0001" "ID0002" "ID0003" "ID0004" ...
 $ date_issue                                        :
chr [1:1053] "11/11/2017" "18/10/2017" NA "32/1/2017" ...
 $ colour_weight
Black in pounds
Green in kilograms: num [1:1053] 92 194 61 165 148 86 84 182
80 78 ...
 $ text_integer_issue                                :
chr [1:1053] "74" "54" "53" "64" ...
 $ text_numeric_issue                                :
chr [1:1053] "1.6" "0.14000000000000001" "0.96" "0.02" ...
 $ numeric_integer_issue                             :
num [1:1053] 1 55 9 2 3 7 1 3 75 23 ...
 $ one_or_zero_issue                                 :
logi [1:1053] NA NA NA NA NA NA ...
```

## Good news

- `numeric_integer_issue`: column is read correctly as numeric

## Bad news

- `text_integer_issue`: column turned to text
- `text_numeric_issue`: column turned to text
- `one_or_zero_issue`: column turned to logical

# Read Data Attempt 2

When I read the formatted data indicating the column types, it gives intimidating warnings.

- `id` as “text”
- `date_issue` as “date”
- `colour_weight` as “numeric”
- `text_integer_issue` and `text_numeric_issue` as “numeric”
- `numeric_integer_issue` and `one_or_zero_issue` as “numeric”

```
sample_excel_attempt_2 <- readxl::read_excel(
  path = here::here("sample_excel.xlsx"),
  sheet = "Sheet1",
  col_types = c("text" , "date",
                "numeric", "numeric",
                "numeric", "numeric",
                "numeric")
)
```

```
Warning: Expecting date in B2 / R2C2: got '11/11/2017'
Warning: Expecting date in B3 / R3C2: got '18/10/2017'
Warning: Coercing text to numeric in D3 / R3C4: '54'
Warning: Coercing text to numeric in E4 / R4C5: '0.96'
Warning: Expecting date in B5 / R5C2: got '32/1/2017'
Warning: Coercing text to numeric in E5 / R5C5: '0.02'
Warning: Coercing text to numeric in E6 / R6C5: '0.23'
Warning: Expecting date in B7 / R7C2: got '29/11/1985'
Warning: Coercing text to numeric in D7 / R7C4: '33'
Warning: Coercing text to numeric in D8 / R8C4: '35'
Warning: Coercing text to numeric in D9 / R9C4: '53'
Warning: Coercing text to numeric in D10 / R10C4: '187'
Warning: Coercing text to numeric in E10 / R10C5: '0.06'
Warning: Expecting date in B11 / R11C2: got '21/11/1962'
Warning: Expecting date in B12 / R12C2: got '30/11/2021'
Warning: Expecting date in B14 / R14C2: got '14/11/2016'
Warning: Expecting date in B15 / R15C2: got '26/09/2016'
Warning: Expecting date in B16 / R16C2: got '15/12/1962'
Warning: Expecting date in B17 / R17C2: got '10/08/2016'
Warning: Expecting date in B18 / R18C2: got '18/07/2016'
Warning: Expecting date in B19 / R19C2: got '19/01/2019'
```

# Read Data Attempt 2

Here is the output

| id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric_issue |
|---|---|---|---|---|
| ID0001 | | 92 | 74 | 1.6 |
| ID0002 | | 194 | 54 | 0.14 |
| ID0003 | | 61 | 53 | 0.96 |
| ID0004 | | 165 | 64 | 0.02 |
| ID0005 | 1971-01-08T00:00:00Z | 148 | 48 | 0.23 |

1–5 of 1053 rows Previous 1 of 211 Next

## Good news

- `numeric_integer_issue`: column is read correctly as numeric
- `one_or_zero_issue`: column is read correctly as numeric

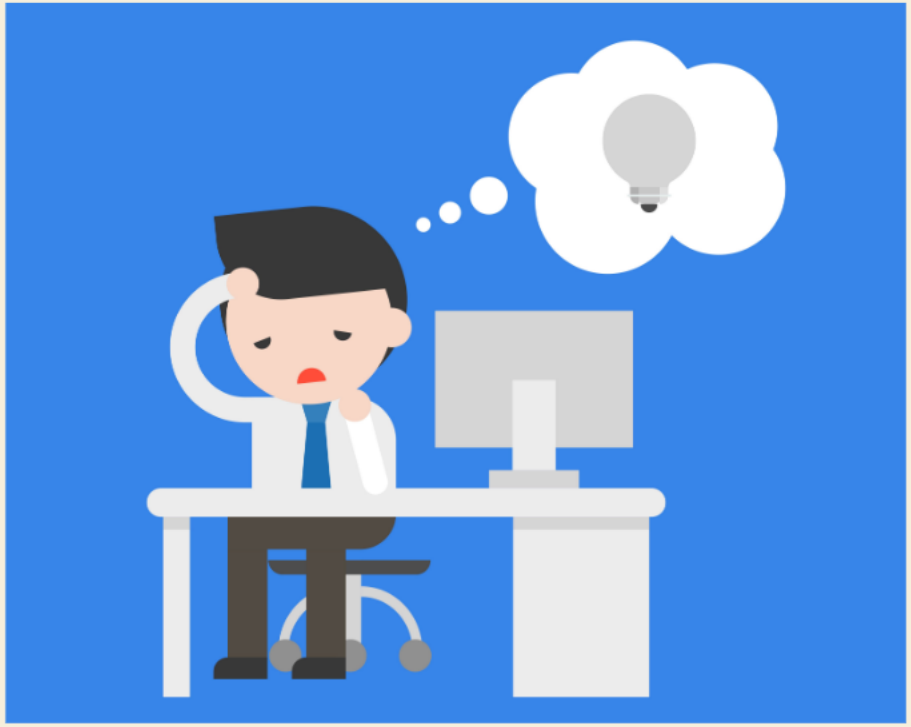
## Bad news

- `date_issue`: many rows turn to blank
- `colour_weight`: different colour inputs not differentiated

# Reflection

From these previous failed attempt, I start to ask these questions and lose confidence in R.

- Does the `id` column only have unique values ?
- Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?
- Does column `numeric_integer_issue` only have integer values ?
- Does column `one_or_zero_issue` only have values 0 or 1 ?
- Can we fix `date_issue` ?
- Can we fix `colour_weight` ?

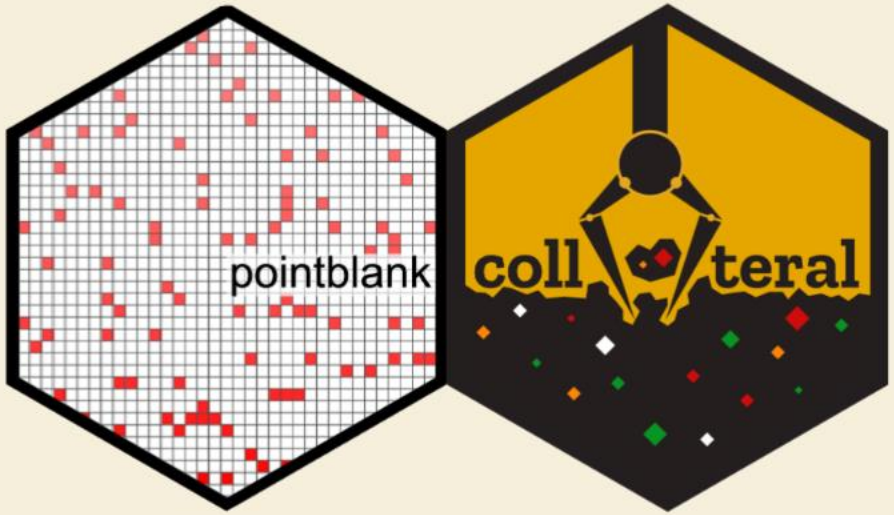


*Businessman sitting at office and feeling dull because lack of idea by Amonrat Rungreangfangsai*

# Reflection

Resort to manual checking/fixing of formatted cells in excel sheets. However, I realise that this approach is not sustainable.

Thankfully, there are some R packages (*pointblank*, *collateral*, *tidyxl*) that can help




tidyxl

tidyxl imports non-tabular data from Excel files into R. It exposes cell content, position, formatting and comments in a tidy structure for further manipulation, especially by the unpivotr package. It supports the xml-based file formats '.xlsx' and '.xlsm' via the embedded RapidXML C++ library. It does not support the binary file formats '.xlsb' or '.xls'.

*Wind-up businessman image by Amonrat Rungreangfangsai*

*https://jauntyjjs.github.io/useR-2024*

# Read Data Attempt 3

Read the excel sheet again with the following `col_type` and deal with each question one at a time.

- `id` as “text”
- `date_issue` as “list”
- ~~`colour_weight \r\nBlack in pounds\r\nGreen in kilograms`~~ as “text”
- `text_integer_issue` and `text_numeric_issue` as “text”
- `numeric_integer_issue` and `one_or_zero_issue` as “numeric”

```
sample_excel_attempt_3 <- readxl::read_excel(
  path = here::here("sample_excel.xlsx"),
  sheet = "Sheet1",
  col_types = c("text" , "list",
                "text", "text", "text",
                "numeric", "numeric")
)
```

# Does the `id` column only have unique values ?

# Does the `id` column only have unique values ?

Use *pointblank::rows_distinct* to validate columns that needs to have unique values.

```
data.frame(id = c("ID_01", "ID_02")) |>
  pointblank::rows_distinct(columns = "id" )
```

```
    id
1 ID_01
2 ID_02
```

```
data.frame(id = c("ID_01", "ID_02", "ID_01")) |>
  pointblank::rows_distinct(columns = "id" )
```

```
Error: Exceedance of failed test units where there weren't
distinct rows across all columns.
The `rows_distinct()` validation failed beyond the absolute
threshold level (1).
* failure level (2) >= failure threshold (1)
```

```
id_check <- sample_excel_attempt_3 |>
  dplyr::select("id") |>
  pointblank::rows_distinct(columns = "id" )
```

| id |
|---|
| ID0001 |
| ID0002 |
| ID0003 |
| ID0004 |
| ID0005 |

1–5 of 1053 rows  Previous  1  of 211  Next

Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

# Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

In sample_excel_attempt_2, when I read `text_integer_issue` as a numeric column, I received some warning message..

- Warning: Coercing text to numeric in D3 / R3C4: '54'
- Warning: Coercing text to numeric in D7 / R7C4: '33'

The warnings inform the user that it sees "54" and "33" in cells D3 and D7 respectively as "text" and it is forced to be converted to numeric

| | D |
|---|---|
| 1 | text_integer_issue |
| 2 | 74 |
| 3 | 54 |
| 4 | 53 |
| 5 | 64 |
| 6 | 48 |
| 7 | 33 |
| 8 | 35 |
| 9 | 53 |
| 10 | 187 |
| 11 | 141 |

**Important**

It may be safer to check if the column truly contain only positive integers even though they are in text, rather than relying on long warning messages.

# Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

Use *pointblank::col_vals_regex* and *pointblank::col_vals_gt* to ensure that text in the column `text_integer_issue` are positive integers.

```r
data.frame(integer_data = c("1", "2.0000", "59", NA)) |>
  pointblank::col_vals_regex(
    columns = c("integer_data"),
    regex = "^[1-9]([0-9]+)?(.[0]+)?$",
    na_pass = TRUE,
  )
```

```
  integer_data
1            1
2       2.0000
3           59
4         <NA>
```

```r
data.frame(integer_data = c(1, 2, 3, NA)) |>
  pointblank::col_vals_gt(
    columns = c("integer_data"),
    value = 0,
    na_pass = TRUE,
  )
```

```
  integer_data
1            1
2            2
3            3
4           NA
```

```r
data.frame(integer_data = c("1", "2.0000", "2.1")) |>
  pointblank::col_vals_regex(
    columns = c("integer_data"),
    regex = "^[1-9]([0-9]+)?(.[0]+)?$",
    na_pass = TRUE,
  )
```

```
Error: Exceedance of failed test units where values in
`integer_data` should have matched the regular expression:
`^[1-9]([0-9]+)?(.[0]+)?$`.
The `col_vals_regex()` validation failed beyond the absolute
threshold level (1).
* failure level (1) >= failure threshold (1)
```

```r
data.frame(integer_data = c(-1, 0, 1, 2)) |>
  pointblank::col_vals_gt(
    columns = c("integer_data"),
    value = 0,
    na_pass = TRUE,
  )
```

```
Error: Exceedance of failed test units where values in
`integer_data` should have been > `0`.
The `col_vals_gt()` validation failed beyond the absolute
threshold level (1).
* failure level (2) >= failure threshold (1)
```

# Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

Similarly, I can use the same functions *pointblank::col_vals_regex* and *pointblank::col_vals_gt* to ensure that the text in the column `text_numeric_issue` are positive numbers.

```r
data.frame(numeric_data = c("0.140", "7.07E-2", "2", NA)) |>
  pointblank::col_vals_regex(
    columns = c("numeric_data"),
    regex = "^[0-9]+((.[0-9]+)?(E(-)?[0-9]+)?)?$",
    na_pass = TRUE,
  )
```

```
  numeric_data
1        0.140
2      7.07E-2
3            2
4         <NA>
```

```r
data.frame(numeric_data = c("not numeric", FALSE, "", 2)) |>
  pointblank::col_vals_regex(
    columns = c("numeric_data"),
    regex = "^[0-9]+((.[0-9]+)?(E(-)?[0-9]+)?)?$",
    na_pass = TRUE,
  )
```

```
Error: Exceedance of failed test units where values in `numeric_data` should have matched the regular expression: `^[0-9]+((.
[0-9]+)?(E(-)?[0-9]+)?)?$`.
The `col_vals_regex()` validation failed beyond the absolute threshold level (1).
* failure level (3) >= failure threshold (1)
```

# Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

Continue with `sample_excel_attempt_3` which reads `text_integer_issue` and `text_numeric_issue` column as text.

*text_integer_issue* | *text_numeric_issue*

```
integer_check_from_text <- sample_excel_attempt_3 |>
  dplyr::select(c("id","text_integer_issue")) |>
  pointblank::col_vals_regex(
    columns = c("text_integer_issue"),
    regex = "^[1-9]([0-9]+)?(.[0]+)?$",
    na_pass = TRUE,
  ) |>
  dplyr::mutate(
    text_integer_issue = as.integer(.data[["text_integer_
  ) |>
  pointblank::col_vals_gt(
    columns = c("text_integer_issue"),
    value = 0,
    na_pass = TRUE,
  ) |>
  dplyr::rename(
    text_integer_verified = "text_integer_issue"
  )
```

| *id* | *text_integer_verified* |
|---|---|
| *ID0001* | *74* |
| *ID0002* | *54* |
| *ID0003* | *53* |
| *ID0004* | *64* |
| *ID0005* | *48* |

*1–5 of 1053 rows* Previous 1 *of 211* Next

# Are numbers in characters from columns `text_integer_issue` and `text_numeric_issue` read correctly ?

Continue with `sample_excel_attempt_3` which reads `text_integer_issue` and `text_numeric_issue` column as text.

*text_integer_issue* | *text_numeric_issue*

```
numeric_check <- sample_excel_attempt_3 |>
  dplyr::select(c("id","text_numeric_issue")) |>
  pointblank::col_vals_regex(
    columns = c("text_numeric_issue"),
    regex = "^[0-9]+((.[0-9]+)?(E(-)?[0-9]+)?)?$",
    na_pass = TRUE,
  ) |>
  dplyr::mutate(
    text_numeric_issue = as.numeric(.data[["text_numeric_
  ) |>
  pointblank::col_vals_gt(
    columns = c("text_numeric_issue"),
    value = 0,
    na_pass = TRUE,
  ) |>
  dplyr::rename(
    text_numeric_verified = "text_numeric_issue"
  )
```

| *id* | *text_numeric_verified* |
|---|---|
| *ID0001* | *1.6* |
| *ID0002* | *0.14* |
| *ID0003* | *0.96* |
| *ID0004* | *0.02* |
| *ID0005* | *0.23* |

*1–5 of 1053 rows* Previous 1 *of 211* Next

Does the column `numeric_integer_issue` only have integer values ?

# Does the column `numeric_integer_issue` only have integer values ?

It may be necessary to check if a numeric column only has integers.

However, I cannot use *pointblank::col_vals_regex* because the column is not read in text.

Create the function `is_integer_vector` that returns **FALSE** when at least one of its element is not an integer.

```r
is_integer_value <- function(input_value,
                             allow_na = FALSE) {

  boolean_result <- FALSE

  # When input value is NA
  if (is.na(input_value)) {
    if (isTRUE(allow_na)) {
      boolean_result <- TRUE
      return(boolean_result)
    } else {
      return(boolean_result)
    }
  }

  # When input value is not numeric
  if (isTRUE(!is.numeric(input_value))) {
    return(boolean_result)
  }

  # When input value is numeric
  boolean_result <- isTRUE(input_value %% 1 == 0)

  return(boolean_result)
}


is_integer_vector <- function(input_vector,
```

# Does the column `numeric_integer_issue` only have integer values ?

It may be necessary to check if a numeric column only has integers.

However, I cannot use *pointblank::col_vals_regex* because the column is not read in text.

Create the function `is_integer_vector` that returns `FALSE` when at least one of its element is not an integer.

```r
      return(boolean_result)
    } else {
      return(boolean_result)
    }
  }

  # When input value is not numeric
  if (isTRUE(!is.numeric(input_value))) {
    return(boolean_result)
  }

  # When input value is numeric
  boolean_result <- isTRUE(input_value %% 1 == 0)

  return(boolean_result)
}


is_integer_vector <- function(input_vector,
                              allow_na = FALSE) {

  boolean_results <- input_vector |>
    purrr::map_lgl(
      .f = is_integer_value,
      allow_na = allow_na
    )
  return(boolean_results)
}
```

# Does the column `numeric_integer_issue` only have integer values ?

Similarly, I can use *pointblank::col_vals_expr* to ensure that the numeric column has only integer using the self-made `is_integer_vector` function.

```r
integer_data <- data.frame(
  integer_col = c(-1, 0, NA, 2.0000, 3)
)

integer_data |>
  pointblank::col_vals_expr(
    expr = ~is_integer_vector(
      input_vector = integer_data[["integer_col"]],
      allow_na = TRUE)
  )
```

```
  integer_col
1          -1
2           0
3          NA
4           2
5           3
```

```r
non_integer_data <- data.frame(
  non_integer_col = c(-1, 0, NA, 2.0000,
                      3.010, pi, exp(1)
                     )
)

non_integer_data |>
  pointblank::col_vals_expr(
    expr = ~ is_integer_vector(
      input_vector = non_integer_data[["non_integer_col"]
      allow_na = TRUE)
  )
```

```
Error: The `col_vals_expr()` validation failed beyond the
absolute threshold level (1).
* failure level (3) >= failure threshold (1)
```

# Does the column `numeric_integer_issue` only have integer values ?

Going back to `sample_excel_attempt_3` which reads `numeric_integer_issue` column as numeric, I apply the `is_integer_vector` function on the `numeric_integer_issue` column before converting the column to an integer column.

```
integer_check_from_numeric <- sample_excel_attempt_3 |>
  dplyr::select(c("id","numeric_integer_issue")) |>
  pointblank::col_vals_expr(
    expr = ~ is_integer_vector(
      input_vector = sample_excel_attempt_3[["numeric_int
      allow_na = TRUE)
  ) |>
  dplyr::mutate(
    numeric_integer_issue = as.integer(.data[["numeric_ir
  ) |>
  dplyr::rename(
    numeric_integer_verified = "numeric_integer_issue"
  )
```

| id | numeric_integer_verified |
|---|---|
| ID0001 | 1 |
| ID0002 | 55 |
| ID0003 | 9 |
| ID0004 | 2 |
| ID0005 | 3 |

1–5 of 1053 rows  Previous 1 of 211 Next

Does the column `one_or_zero_issue` only have values 0 or 1 ?

# Does the column `one_or_zero_issue` only have values 0 or 1 ?

Use *pointblank::col_vals_in_set* to ensure that the column only contains values from a user-defined set.

```
data.frame(one_or_zero_data = c(0, NA, 1)) |>
  pointblank::col_vals_in_set(
    columns = c("one_or_zero_data"),
    set = c(NA, 0, 1)
  )
```

```
  one_or_zero_data
1                0
2               NA
3                1
```

```
data.frame(one_or_zero_data = c(0, NA, 1, 2)) |>
  pointblank::col_vals_in_set(
    columns = c("one_or_zero_data"),
    set = c(NA, 0, 1)
  )
```

```
Error: Exceedance of failed test units where values in
`one_or_zero_data` should have been in the set of `NA`, `0`,
`1`.
The `col_vals_in_set()` validation failed beyond the
absolute threshold level (1).
* failure level (1) >= failure threshold (1)
```

```
one_or_zero_check <- sample_excel_attempt_3 |>
  dplyr::select(c("id","one_or_zero_issue")) |>
  pointblank::col_vals_in_set(
    columns = c("one_or_zero_issue"),
    set = c(NA, 0, 1)
  ) |>
  dplyr::rename(
    one_or_zero_verified = "one_or_zero_issue"
  )
```

| id | one_or_zero_verified |
|---|---|
| ID0001 | |
| ID0002 | |
| ID0003 | |
| ID0004 | |
| ID0005 | |

1–5 of 1053 rows  Previous  1  of 211  Next

# Can we fix `date_issue` ?

# Can we fix `date_issue` ?

First, convert the date columns into a list of character, `Date` and logical vectors so that data in both Excel General and Date format are preserved.

```r
sample_excel_attempt_3 <- readxl::read_excel(
  path = here::here("sample_excel.xlsx"),
  sheet = "Sheet1",
  col_types = c("text" , "list",
                "text", "text", "text",
                "numeric", "numeric")
) |>
  pointblank::rows_distinct(columns = "id" )


str(head(sample_excel_attempt_3$date_issue))
```

```
List of 6
 $ : chr "11/11/2017"
 $ : chr "18/10/2017"
 $ : logi NA
 $ : chr "32/1/2017"
 $ : POSIXct[1:1], format: "1971-01-08"
 $ : chr "29/11/1985"
```

| id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric_issue |
|---|---|---|---|---|
| ID0001 | 11/11/2017 | 92 | 74 | 1.6 |
| ID0002 | 18/10/2017 | 194 | 54 | 0.14000000000000001 |
| ID0003 | | 61 | 53 | 0.96 |
| ID0004 | 32/1/2017 | 165 | 64 | 0.02 |
| ID0005 | 1971-01-08T00:00:00Z | 148 | 48 | 0.23 |

1–5 of 1053 rows Previous 1 of 211 Next

# Can we fix `date_issue` ?

Next, create a function that convert dates in character vectors into `Date` objects, convert logical vector to `NA` and convert `Date` vectors into the date format that I want.

```
convert_dmy_text_to_date <- function(input) {
  if (length(class(input)) == 1) {
    if (class(input) == "character") {
      return(as.Date.character(lubridate::dmy(input)))
    } else if (class(input) == "logical") {
      return(NA)
    }
  }
  return(lubridate::as_date(lubridate::ymd(input)))
}
```

> **Tip**
>
> However, creating function can lead to unexpected warnings and errors. To view these issues, I use some functions from the *collateral* R package.

- *collateral::map_peacefully*
- *collateral::has_warnings* and *collateral::has_errors*

# Can we fix `date_issue` ?

Use *collateral::map_peacefully* to capture function side effects using both *purrr::safely()* and *purrr::quietly()*.

```
fixed_date <- sample_excel_attempt_3 |>
  dplyr::select(c("id","date_issue")) |>
  dplyr::mutate(
    converted_date_log = collateral::map_peacefully(
      .x = .data[["date_issue"]],
      .f = convert_dmy_text_to_date
    ),
    converted_date = purrr::map_vec(
      .x = .data[["converted_date_log"]],
      .f = "result"
    )
  )

print(head(fixed_date))
```

```
# A tibble: 6 × 4
  id     date_issue converted_date_log converted_date
  <chr>  <list>     <collat>           <date>
1 ID0001 <chr [1]>  R _ _ _ _          2017-11-11
2 ID0002 <chr [1]>  R _ _ _ _          2017-10-18
3 ID0003 <lgl [1]>  R _ _ _ _          NA
4 ID0004 <chr [1]>  R _ _ W _          NA
5 ID0005 <dttm [1]> R _ _ _ _          1971-01-08
6 ID0006 <chr [1]>  R _ _ _ _          1985-11-29
```

# Can we fix `date_issue` ?

Use *collateral::has_warnings* and *collateral::has_errors* to create logical columns which gives TRUE when there are warning or error messages. Use *pointblank::test_col_vals_in_set* to obtain a single logical value.

```
fixed_date <- fixed_date |>
  dplyr::mutate(
    warning_check = collateral::has_warnings(.data[["converted_date_log"]]),
    error_check = collateral::has_errors(.data[["converted_date_log"]])
  )

print(head(fixed_date))
```

```
# A tibble: 6 × 6
  id     date_issue converted_date_log converted_date warning_check error_check
  <chr>  <list>     <collat>           <date>         <lgl>         <lgl>
1 ID0001 <chr [1]>  R _ _ _ _          2017-11-11     FALSE         FALSE
2 ID0002 <chr [1]>  R _ _ _ _          2017-10-18     FALSE         FALSE
3 ID0003 <lgl [1]>  R _ _ _ _          NA             FALSE         FALSE
4 ID0004 <chr [1]>  R _ _ W _          NA             TRUE          FALSE
5 ID0005 <dttm [1]> R _ _ _ _          1971-01-08     FALSE         FALSE
6 ID0006 <chr [1]>  R _ _ _ _          1985-11-29     FALSE         FALSE
```

```
no_issue <- fixed_date |>
  pointblank::test_col_vals_in_set(
    columns = c("warning_check", "error_check"),
    set = c(FALSE)
  )

print(no_issue)
```

```
[1] FALSE
```

# Can we fix `date_issue` ?

Isolate rows with issues and output the warning and error messages.

```r
if (!isTRUE(no_issue)) {
  fixed_date |>
    dplyr::filter(
      warning_check == TRUE | error_check == TRUE
    ) |>
    dplyr::mutate(
      warning_log = purrr::map(
        .x = .data[["converted_date_log"]],
        .f = "warnings",
        .null = NA),
      error_log = purrr::map(
        .x = .data[["converted_date_log"]],
        .f = "errors",
        .null = NA)
    ) |>
    reactable::reactable(
      style = list(fontSize = "1rem")
    )
}
```

| id | date_issue | converted_date_log | converted_date | warning_check | e |
|---|---|---|---|---|---|
| ID0004 | 32/1/2017 | [object Object] | | true | f |

# Can we fix `date_issue` ?

Correct the invalid dates accordingly and rerun everything. We just assume that 32/1/2017 was supposed to be 31/1/2017.

```
fixed_date <- sample_excel_attempt_3 |>
  dplyr::select(c("id","date_issue")) |>
  dplyr::mutate(
    date_issue = dplyr::case_when(
      (.data[["id"]] == "ID0004" &
       .data[["date_issue"]] == "32/1/2017"
      ) ~ list(c("31/1/2017")),
      .default = .data[["date_issue"]]
    )
  ) |>
  dplyr::mutate(
    converted_date_log = collateral::map_peacefully(
      .x = .data[["date_issue"]],
      .f = convert_dmy_text_to_date
    ),
    converted_date = purrr::map_vec(
      .x = .data[["converted_date_log"]],
      .f = "result"
    ),
    warning_check = collateral::has_warnings(.data[["co
    error_check = collateral::has_errors(.data[["conve

  ) |>
  pointblank::col_vals_in_set(
    columns = c("warning_check", "error_check"),
    set = c(FALSE)
  ) |>
```

| id | date_fixed_yyyy_mm_dd |
|---|---|
| ID0001 | 2017-11-11 |
| ID0002 | 2017-10-18 |
| ID0003 | |
| ID0004 | 2017-01-31 |
| ID0005 | 1971-01-08 |

1–5 of 1053 rows  Previous 1 of 211 Next

# Can we fix `colour_weight` ?

# Can we fix `colour_weight` ?

Use *tidyxl::xlsx_cells* to read the excel file in cells.

```
1  cells <- tidyxl::xlsx_cells(
2    path = here::here("sample_excel.xlsx"),
3    sheet = "Sheet1",
4    include_blank_cells = TRUE)
```

Observe that there is no indication of which row is **green** or **black**.

I am only provided with the `local_format_id` labelled 1 to 15 at the last column.

| sheet | address | row | col | is_blan |
|---|---|---|---|---|
| Sheet1 | A1 | 1 | 1 | false |
| Sheet1 | B1 | 1 | 2 | false |
| Sheet1 | C1 | 1 | 3 | false |
| Sheet1 | D1 | 1 | 4 | false |
| Sheet1 | E1 | 1 | 5 | false |

1–5 of 6335 rows Previous 1 of 1267 Next

# Can we fix `colour_weight` ?

Use *tidyxl::xlsx_cells* to read the excel file in cells.

```
cells <- tidyxl::xlsx_cells(
  path = here::here("sample_excel.xlsx"),
  sheet = "Sheet1",
  include_blank_cells = TRUE)
```

Observe that there is no indication of which row is **green** or **black**.

I am only provided with the `local_format_id` labelled 1 to 15 at the last column.

| width | row_outline_level | col_outline_level | style_format | local_format_id |
|---|---|---|---|---|
| 8.38 | 1 | 1 | Normal | 1 |
| 6640625 | 1 | 1 | Normal | 1 |
| 5546875 | 1 | 1 | Normal | 8 |
| 3671875 | 1 | 1 | Normal | 1 |
| 6640625 | 1 | 1 | Normal | 1 |

1–5 of 6335 rows Previous 1 of 1267 Next

# Can we fix `colour_weight` ?

Use *tidyxl::xlsx_formats* to obtain the format information of the excel file in a list.

```
formats <- tidyxl::xlsx_formats(
  path = here::here("sample_excel.xlsx")
)
```

Here is a way to view all colours in Hex8 used for all 15 `local_format_id`

```
print(formats$local$font$color$rgb)
```

```
 [1] "FF000000" "FF000000" "FF00B050" "FF000000" "FF000000"
"FF000000"
 [7] "FF000000" "FF000000" "FF000000" "FF000000" NA
NA
[13] NA         "FF00B050" "FF000000"
```

```
unique(formats$local$font$color$rgb)
```

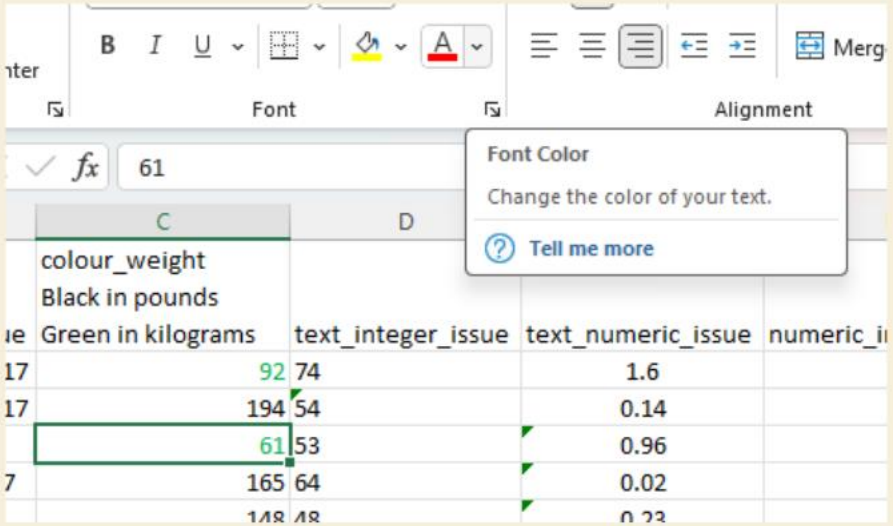
```
[1] "FF000000" "FF00B050" NA
```

Need to identify which one is **black** and **green**.

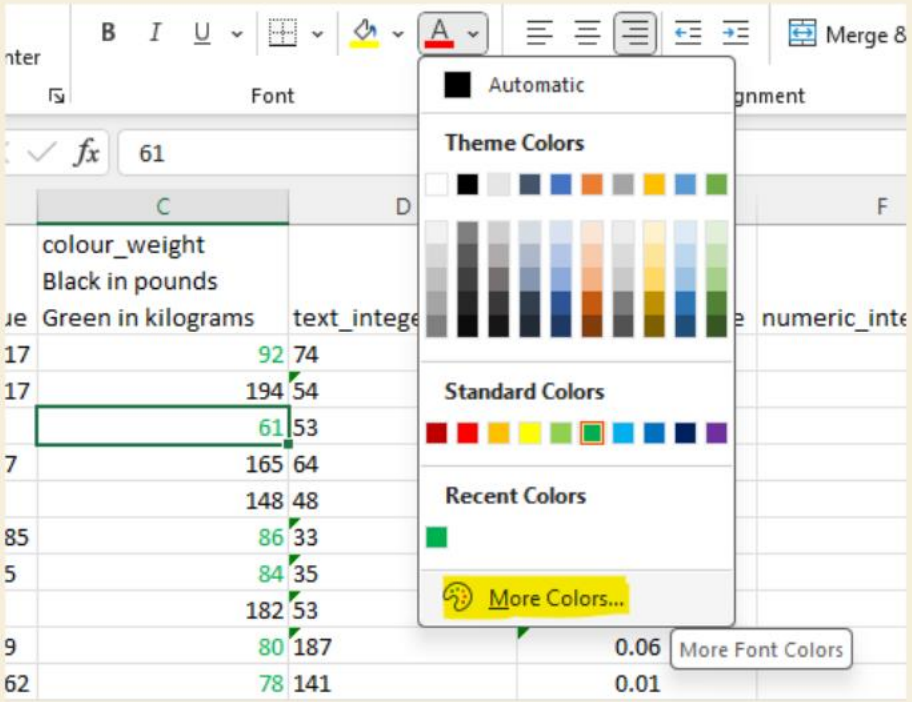
```
$local
$local$numFmt
 [1] "General"  "mm-dd-yy" "General"  "General"  "General"
"mm-dd-yy"
 [7] "@"        "General"  "General"  "General"  "mm-dd-
yy" "General"
[13] "General"  "General"  "General"

$local$font
$local$font$bold
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
[13] FALSE FALSE FALSE

$local$font$italic
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
[13] FALSE FALSE FALSE

$local$font$underline
 [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

$local$font$strike
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
FALSE FALSE FALSE
[13] FALSE FALSE FALSE

$local$font$vertAlign
```

# Can we fix `colour_weight` ?

Using green as the running example, first in excel, click on a cell with **green** font. Next, click on the drop down button beside the font colour button.

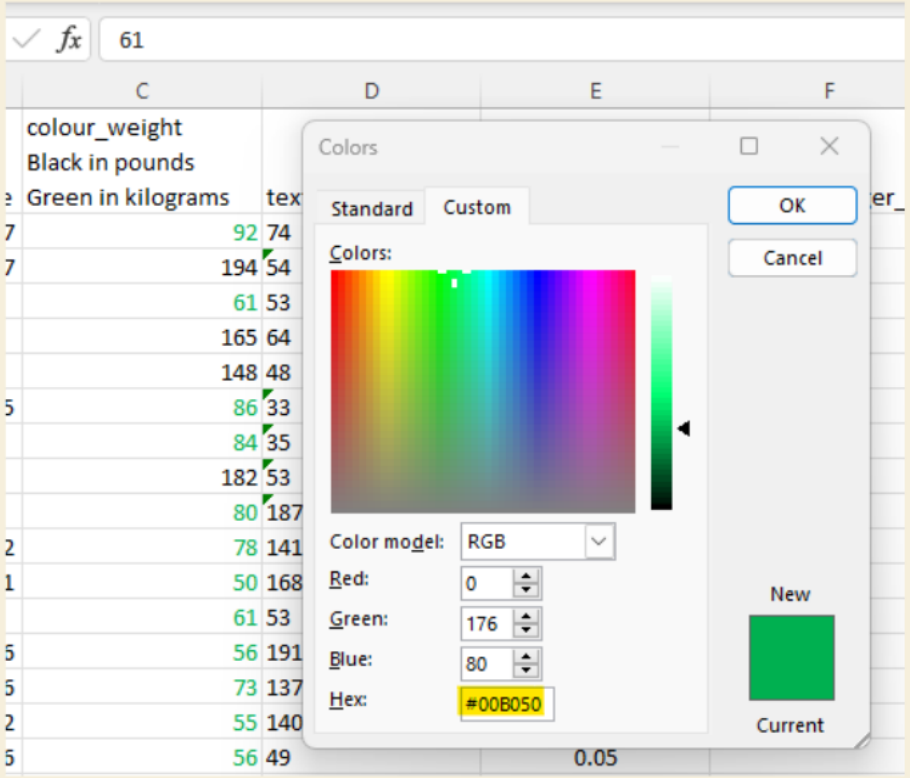



This will give the following output. Next click on More Colors…

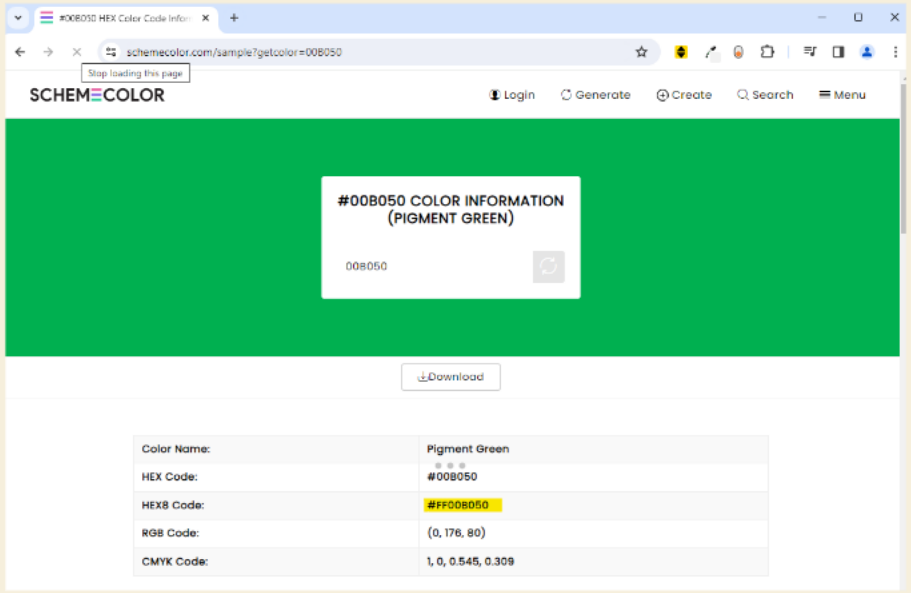


# Can we fix `colour_weight` ?

Go to the Custom tab and extract the hex code saying #00B050 for **green**.




Next, use *https://www.schemecolor.com/?getcolor={hex code}* (*https://www.schemecolor.com/sample?getcolor=00B050* in our running example) to find out what the Hex8 code is for the **green** font.



# Can we fix `colour_weight` ?

Identify the `local_format_id` accordingly with the **black** and **green** Hex8 code as #**FF000000** and **#FF00B050** respectively.

```
1 green_font_local_format_id <- which(formats$local$font$color$rgb == "FF00B050")
2 green_font_local_format_id
```

```
[1]  3 14
```

```
1 black_font_local_format_id <- which(formats$local$font$color$rgb == "FF000000")
2 black_font_local_format_id
```

```
 [1]  1  2  4  5  6  7  8  9 10 15
```

Identify the column index of `colour_weight_black_in_pounds_green_in_kilograms`. *pointblank::row_count_match* is used to ensure we have only one row left after filtering.

```
1 weight_column_index <- cells |>
2   dplyr::filter(
3     .data[["character"]] == "colour_weight \r\nBlack in pounds\r\nGreen in kilograms"
4   ) |>
5   pointblank::row_count_match(count = 1) |>
6   dplyr::pull(.data[["col"]])
7
8 weight_column_index
```

```
[1] 3
```

# Can we fix `colour_weight` ?

With the column index and `local_format_id` identified, we can filter the `cells` data to isolate cells which contain the weight in pounds.

| sheet | address | row | col | is_blank | |
|---|---|---|---|---|---|
| | | | | | |
| Sheet1 | A1 | 1 | 1 | false | |
| Sheet1 | B1 | 1 | 2 | false | |
| Sheet1 | C1 | 1 | 3 | false | |
| Sheet1 | D1 | 1 | 4 | false | |
| Sheet1 | E1 | 1 | 5 | false | |

1–5 of 6335 rows Previous 1 of 1267 Next

```
weight_in_pounds <- cells |>
  dplyr::filter(.data[["row"]] != 1) |>
  dplyr::filter(.data[["col"]] == weight_column_index) |>
  dplyr::filter(.data[["local_format_id"]] %in% black_for
  pointblank::col_vals_in_set(columns = c("data_type"), s
  dplyr::filter(.data[["data_type"]] == "numeric") |>
  dplyr::select(c("row", "numeric")) |>
  dplyr::rename(weight_pounds = "numeric") |>
  dplyr::mutate(
    weight_kg_converted = janitor::round_half_up(.data[["
  )
```

| row | weight_pounds | weight_kg_converted |
|---|---|---|
| | | |
| 3 | 194 | 88 |
| 5 | 165 | 75 |
| 6 | 148 | 67 |
| 9 | 182 | 83 |
| 36 | 191 | 87 |

1–5 of 602 rows Previous 1 of 121 Next

# Can we fix `colour_weight` ?

With the column index and `local_format_id` identified, we can filter the `cells` data to isolate cells which contain the weight in kilogram.

| sheet | address | row | col | is_blank |
|---|---|---|---|---|
| Sheet1 | A1 | 1 | 1 | false |
| Sheet1 | B1 | 1 | 2 | false |
| Sheet1 | C1 | 1 | 3 | false |
| Sheet1 | D1 | 1 | 4 | false |
| Sheet1 | E1 | 1 | 5 | false |

1–5 of 6335 rows Previous 1 of 1267 Next

```
weight_in_kg <- cells |>
  dplyr::filter(.data[["row"]] != 1) |>
  dplyr::filter(.data[["col"]] == weight_column_index) |>
  dplyr::filter(.data[["local_format_id"]] %in% green_for
  pointblank::col_vals_in_set(
    columns = c("data_type"),
    set = c("numeric")
  ) |>
  dplyr::filter(.data[["data_type"]] == "numeric") |>
  dplyr::select(c("row", "numeric")) |>
  dplyr::rename(weight_kg = "numeric")
```

| row | weight_kg |
|---|---|
| 2 | 92 |
| 4 | 61 |
| 7 | 86 |
| 8 | 84 |
| 10 | 80 |

1–5 of 451 rows Previous 1 of 91 Next

# Can we fix `colour_weight` ?

Need to extract the `id` column from `cells`

| sheet | address | row | col | is_blank | |
|---|---|---|---|---|---|
| Sheet1 | A1 | 1 | 1 | false | |
| Sheet1 | B1 | 1 | 2 | false | |
| Sheet1 | C1 | 1 | 3 | false | |
| Sheet1 | D1 | 1 | 4 | false | |
| Sheet1 | E1 | 1 | 5 | false | |

1–5 of 6335 rows Previous 1 of 1267 Next

```
id_column_index <- which(
  colnames(sample_excel_attempt_3) == "id"
)

id_cells <- cells |>
  dplyr::filter(.data[["row"]] != 1) |>
  dplyr::filter(.data[["col"]] == id_column_index) |>
  pointblank::col_vals_in_set(
    columns = c("data_type"),
    set = c("character")
  ) |>
  dplyr::select(c("row", "character")) |>
  dplyr::rename(id = "character")
```

| row | id |
|---|---|
| 2 | ID0001 |
| 3 | ID0002 |
| 4 | ID0003 |
| 5 | ID0004 |
| 6 | ID0005 |

1–5 of 1053 rows Previous 1 of 211 Next

# Can we fix `colour_weight` ?

Combine the weight data together

```
fixed_weight <- id_cells |>
  dplyr::left_join(weight_in_pounds,
                   by = dplyr::join_by("row"),
                   unmatched = "error",
                   relationship = "one-to-one") |>
  dplyr::left_join(weight_in_kg,
                   by = dplyr::join_by("row"),
                   unmatched = "error",
                   relationship = "one-to-one") |>
  tidyr::unite(
    col = "weight_fixed_kg",
    c("weight_kg_converted",
      "weight_kg"),
    remove = TRUE,
    na.rm = TRUE) |>
  dplyr::select(c("id", "weight_fixed_kg"))
```

| id | weight_fixed_kg |
|---|---|
| ID0001 | 92 |
| ID0002 | 88 |
| ID0003 | 61 |
| ID0004 | 75 |
| ID0005 | 67 |

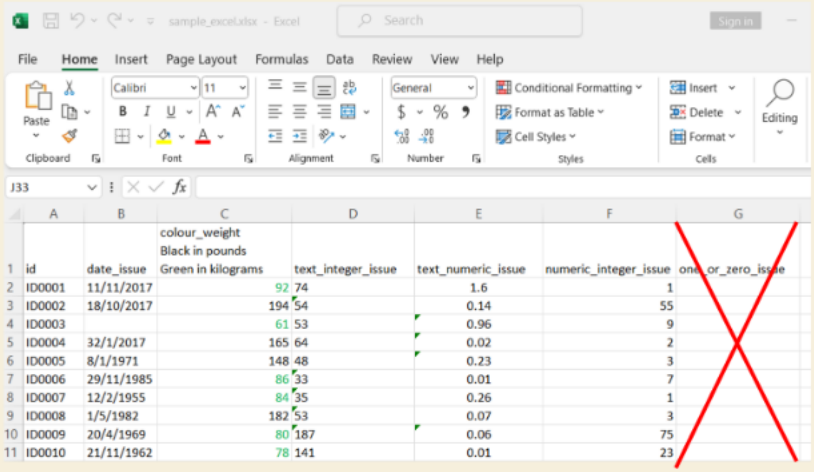
1–5 of 1053 rows Previous 1 of 211 Next

# Can we fix `colour_weight` (alternative) ?

Another approach is to use *unheadr::annotate_mf_all* but the last column `one_or_zero_issue` must be removed. Here is the file required: (sample_excel_remove_last.xlsx).

```
sample_excel_attempt_4 <- unheadr::annotate_mf_all
  xlfilepath = here::here(
    "sample_excel.xlsx"
  )
)
```

```
Error in unheadr::annotate_mf_all(xlfilepath =
here::here("sample_excel.xlsx")): Check spreadsheet
for blank cells in seemingly empty rows
```




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric_issue | numeric_integer_issue | one_or_zero_issue |
| 2 | ID0001 | 11/11/2017 | 92 | 74 | 1.6 | 1 | |
| 3 | ID0002 | 18/10/2017 | 194 | 54 | 0.14 | 55 | |
| 4 | ID0003 | | 61 | 53 | 0.96 | 9 | |
| 5 | ID0004 | 32/1/2017 | 165 | 64 | 0.02 | 2 | |
| 6 | ID0005 | 8/1/1971 | 148 | 48 | 0.23 | 3 | |
| 7 | ID0006 | 29/11/1985 | 86 | 33 | 0.01 | 7 | |
| 8 | ID0007 | 12/2/1955 | 84 | 35 | 0.26 | 1 | |
| 9 | ID0008 | 1/5/1982 | 182 | 53 | 0.07 | 3 | |
| 10 | ID0009 | 20/4/1969 | 80 | 187 | 0.06 | 75 | |
| 11 | ID0010 | 21/11/1962 | 78 | 141 | 0.01 | 23 | |

```
sample_excel_attempt_4 <- unheadr::annotate_mf_all(
  xlfilepath = here::here("sample_excel_remove_last.xlsx")
)
```

| id | date_issue | colour_weight Black in pounds Green in kilograms | text_integer_issue | text_numeric_issue | numeric_ir r_issue |
|---|---|---|---|---|---|
| | | | | | |
| (color-FF000000) ID0001 | (color-FF000000) 11/11/2017 | (color-FF00B050) 92 | (color-FF000000) 74 | (color-FF000000) 1.6 | (color-FF00 1 |
| (color-FF000000) ID0002 | (color-FF000000) 18/10/2017 | (color-FF000000) 194 | (color-FF000000) 54 | (color-FF000000) 0.14000000000000001 | (color-FF00 55 |
| (color-FF000000) ID0003 | (color-FF000000) NA | (color-FF00B050) 61 | (color-FF000000) 53 | (color-FF000000) 0.96 | (color-FF00 9 |
| (color-FF000000) ID0004 | (color-FF000000) 32/1/2017 | (color-FF000000) 165 | (color-FF000000) 64 | (color-FF000000) 0.02 | (color-FF00 2 |
| (color-FF000000) ID0005 | (color-FF000000) 25941 | (color-FF000000) 148 | (color-FF000000) 48 | (color-FF000000) 0.23 | (color-FF00 3 |

1–5 of 1053 rows Previous 1 of 211 Next

# Wrapping up

# Wrapping up

Combine all fixed and verified columns together

| | id | date_issue | colour_weight<br>Black in pounds<br>Green in kilograms | text_integer_issue |
|---|---|---|---|---|
| 1 | id | date_issue | colour_weight<br>Black in pounds<br>Green in kilograms | text_integer_issue |
| 2 | ID0001 | 11/11/2017 | 92 | 74 |
| 3 | ID0002 | 18/10/2017 | 194 | 54 |
| 4 | ID0003 | | 61 | 53 |
| 5 | ID0004 | 32/1/2017 | 165 | 64 |
| 6 | ID0005 | 8/1/1971 | 148 | 48 |
| 7 | ID0006 | 29/11/1985 | 86 | 33 |
| 8 | ID0007 | 12/2/1955 | 84 | 35 |
| 9 | ID0008 | 1/5/1982 | 182 | 53 |
| 10 | ID0009 | 20/4/1969 | 80 | 187 |
| 11 | ID0010 | 21/11/1962 | 78 | 141 |

| | id | text_numeric_issue | numeric_integer_issue | one_or_zero_issue |
|---|---|---|---|---|
| 1 | id | text_numeric_issue | numeric_integer_issue | one_or_zero_issue |
| 2 | ID0001 | 1.6 | 1 | |
| 3 | ID0002 | 0.14 | 55 | |
| 4 | ID0003 | 0.96 | 9 | |
| 5 | ID0004 | 0.02 | 2 | |
| 6 | ID0005 | 0.23 | 3 | |
| 7 | ID0006 | 0.01 | 7 | |
| 8 | ID0007 | 0.26 | 1 | |
| 9 | ID0008 | 0.07 | 3 | |
| 10 | ID0009 | 0.06 | 75 | |
| 11 | ID0010 | 0.01 | 23 | |

▼ Code

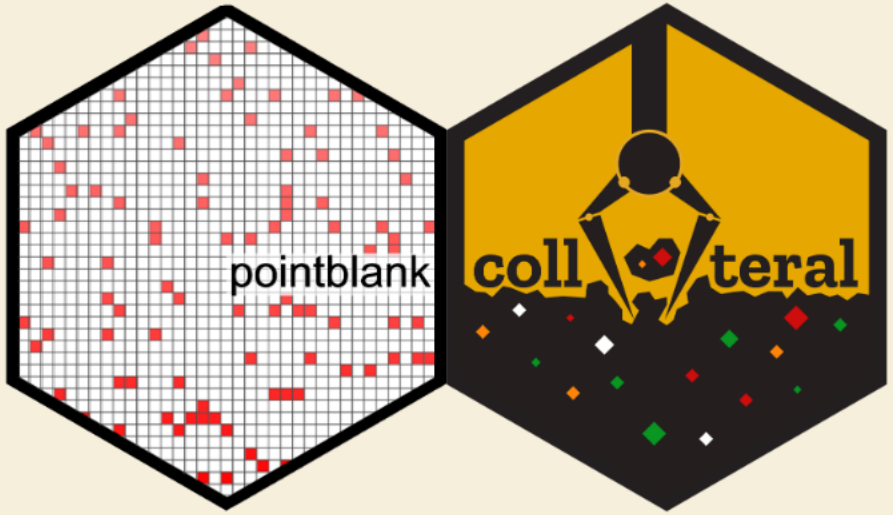
```
cleaned_data <- sample_excel_attempt_3 |>
  dplyr::select("id") |>
  dplyr::left_join(fixed_date,
                   by = dplyr::join_by("id"),
                   unmatched = "error",
                   relationship = "one-to-one") |>
  dplyr::left_join(fixed_weight,
                   by = dplyr::join_by("id"),
                   unmatched = "error",
                   relationship = "one-to-one") |>
  dplyr::left_join(integer_check_from_text,
```

| id | date_fixed_yyyy_mm_dd | weight_fixed_kg | text_integer_verified | text_numeric_verified |
|---|---|---|---|---|
| | | | | |
| ID0001 | 2017-11-11 | 92 | 74 | 1.6 |
| ID0002 | 2017-10-18 | 88 | 54 | 0.14 |
| ID0003 | | 61 | 53 | 0.96 |
| ID0004 | 2017-01-31 | 75 | 64 | 0.02 |
| ID0005 | 1971-01-08 | 67 | 48 | 0.23 |

# Wrapping up

R packages (*pointblank*, *collateral*, *tidyxl*) can help to validate and tackle some problematic formatted columns in Excel, without resorting to too much manual work.

However, we can see that tidying up formatted Excel files remains challenging, even with *R*. Hope that this presentation can encourage others to persevere and strive to find/share alternative ways.




tidyxl

tidyxl imports non-tabular data from Excel files into R. It exposes cell content, position, formatting and comments in a tidy structure for further manipulation, especially by the unpivotr package. It supports the xml-based file formats '.xlsx' and '.xlsm' via the embedded RapidXML C++ library. It does not support the binary file formats '.xlsb' or '.xls'.


*Businessman sitting at stack of books, read a book and dreaming about success by Amonrat Rungreangfangsai*

*https://jauntyjjs.github.io/useR-2024*