How to Guide for Making Printable Geographic Locations for a Fire Lookout

Rought Draft 3

Travis Newton

2024-08-16

Table of contents

Chapter 1

Building Reference Location Tables

1.1 Introduction

1.1.1 About

Utilizing code in R can improve redundant operations in daily tasks as an employee in public service or a corporation. This tutorial series will focus on the United States State Forest Service (USFS) at the fire prevention level, specifically, a Fire Lookout Tower.

While the job of a lookout may seem straightforward, it is of utmost importance. A lookout's primary task is maintaining a constant 360-degree vigil, scanning for fire signs in known areas such as campsites, farming, and logging operations. Lighting is the most frequent cause of fires nowadays due to a Laissez-faire policy on forest management plans, reduced timber sales, and climate change.

In this first chapter, we explore making location cards for a Fire Lookout to use with his Osborne Fire Finder to quickly ID and triangulate smoke in the forgone distance. Depending on the region, lookouts come in many styles, and this covers a D-6 Cupola called the Dutchman Peak Lookout (circa.1927).

1.1.2 Task

The Dutchman Peak was fortunate to have past lookouts who wrote hand-drawn calligraphy cards displaying the same information the tutorial builds. Over time, the original cards have faded, weathered, and become unreadable. The tutorial will curate that data and replicate the original calligraphy as closely as possible.

Data and visual exploration were used to develop the improvements below. The `level` row will guide the user in placement after the cards have been laminated and individually cut (see Table 1.1).

1.1.3 Install packages and load data tables

```
# When installing packages for first time you will need to execute the options unlock on sharepoint computers
options("install.lock"=FALSE)
```

```
#Packages to be loaded
packages <- c("janitor", "tidyverse", "here", "readODS", "here", "showtext", "gdtools", "sysfonts", "png", "flextable")

#Check to see if packages exist, and load them. If they do not exist, they will be installed
package.check <- lapply(
  packages, FUN = function(p) {
    if (!require(p, character.only = TRUE)) {
      install.packages(p,dependencies = TRUE)
      library(p, character.only = TRUE)
    }
  })
```

1. Tibbles make better tables than data frames and are part of the tidyverse package. `clean_names` turns any column header into lowercase when reading in files. Start with a table of data with locations, elevation, degrees, distance, alternate names (used in future references), and a wall location.

```
#Read data into a tibble vs dataframe
maps <- as_tibble(read_ods(here("locationNames.ods"), col_names = TRUE)) %>% clean_names()

#arrange your point locations by azimuth
maps <- maps %>%
  arrange(deg_min)
head(maps, 3)
```

```
# A tibble: 3 x 7
  dir   name               deg_min miles  elev alt_name1 alt_name2
  <chr> <chr>                <dbl> <dbl> <dbl> <chr>     <chr>
1 <NA>  dutchman peak l.o.      0       0  7417 <NA>      <NA>
2 <NA>  point mtn             14      10  5136 <NA>      <NA>
3 <NA>  roxy ann              14.2    22  3576 <NA>      <NA>
```

```
c.org <- c("Geographic Name", "Degrees Minutes : Miles", "Elevation", NA, NA)
n.new <- c("Geographic Name", "Dir :: Degrees Minutes", "Miles", "Elevation",     "Level")

card.design <- data.frame("Original_Card" = c.org, "New_Card" = n.new)

ft <- flextable(card.design) %>%
  theme_vanilla() %>%
  autofit()
ft
```

Table 1.1: Card Design

Original_Card	New_Card
Geographic Name	Geographic Name
Degrees Minutes : Miles	Dir :: Degrees Minutes
Elevation	Miles
	Elevation
	Level

1.2 Cleaning Data

1. Create new columns with `mutate` from existing data in the table. A new column, *wall*, helps place it inside the cupola. The `between` function creates a range for the cardinal direction.

> Logical direction statement
>
> The north direction receives two `between` functions because it operates right and left at 0 degrees. The computer would not understand 320-50. Logically, you must state 320-360, then 0-50 degrees.

```
#Use the between method from dyplr as a shortcut with if_else to create a new column
maps <- maps %>%
  mutate(wall_hang = if_else(between(maps$deg_min, 320, 360) |
                               between(maps$deg_min, 0, 50), "North",
                             if_else(between(maps$deg_min, 50.1, 135), "East",
                                     if_else(between(maps$deg_min, 135.1, 225),"South",
                                             if_else(between(maps$deg_min, 225.1, 319.9), "West", NA)))))
head(maps, 3)
```

```
# A tibble: 3 x 8
  dir   name               deg_min miles  elev alt_name1 alt_name2 wall_hang
  <chr> <chr>                <dbl> <dbl> <dbl> <chr>     <chr>     <chr>
1 <NA>  dutchman peak l.o.     0       0  7417 <NA>      <NA>      North
2 <NA>  point mtn             14      10  5136 <NA>      <NA>      North
3 <NA>  roxy ann              14.2    22  3576 <NA>      <NA>      North
```

2. The same method is then applied to get the cardinal direction. The new *dir* column will combine into a label later.

```
#Create cardinal direction
maps <- maps %>%
  mutate(dir = if_else(between(maps$deg_min, 330, 360) | between(maps$deg_min, 0, 30), "N",
                       if_else(between(maps$deg_min, 30.1, 60), "NE",
                                if_else(between(maps$deg_min, 60.1, 120), "E",
                                        if_else(between(maps$deg_min, 120.1, 150), "E", if_else(between(maps$deg_min, 150.1, 210),
head(maps, 3)
```

```
# A tibble: 3 x 8
  dir   name               deg_min miles  elev alt_name1 alt_name2 wall_hang
  <chr> <chr>                <dbl> <dbl> <dbl> <chr>     <chr>     <chr>
1 N     dutchman peak l.o.     0       0  7417 <NA>      <NA>      North
2 N     point mtn             14      10  5136 <NA>      <NA>      North
3 N     roxy ann              14.2    22  3576 <NA>      <NA>      North
```

3. Instead of using the new `between` method, the below baseR would use greater and equal signs. The goal is to set four levels of tables, which can be used to designate different effects such as size, color, or font style (bold,italic).

Logical direction statement

The second part, using `mutate_if`, acts as an old `row_wise` technique that looks for character strings and provides them with the correct casing.

```
# mutate across rows to change data
maps <- maps %>%
  mutate(level = if_else(miles <= 10, 1,
                         if_else(miles > 10 & miles <=20, 2,
                                 if_else(miles > 20 & miles <=30, 3, 4 )))) %>%
  mutate_if(is.character,str_to_title) #what to predict, then function
head(maps, 3)
```

```
# A tibble: 3 x 9
  dir   name             deg_min miles  elev alt_name1 alt_name2 wall_hang level
  <chr> <chr>              <dbl> <dbl> <dbl> <chr>     <chr>     <chr>     <dbl>
1 N     Dutchman Peak L~     0       0  7417 <NA>      <NA>      North         1
2 N     Point Mtn           14      10  5136 <NA>      <NA>      North         1
3 N     Roxy Ann            14.2    22  3576 <NA>      <NA>      North         3
```

4. For calligraphy purposes, the degrees and minutes must be split into two separate columns. A warning will appear for missing values with NA, but will be corrected in the next block.

```
#separate the degrees and minutes
maps <- maps %>%
  separate(col = deg_min,
           remove = FALSE,
           into = c("degree", "minutes")) #these columns auto set as characters
head(maps,3)
```

```
# A tibble: 3 x 11
  dir   name    deg_min degree minutes miles  elev alt_name1 alt_name2 wall_hang
  <chr> <chr>     <dbl> <chr>  <chr>   <dbl> <dbl> <chr>     <chr>     <chr>
1 N     Dutchm~     0   0      <NA>        0  7417 <NA>      <NA>      North
2 N     Point ~    14   14     <NA>       10  5136 <NA>      <NA>      North
3 N     Roxy A~    14.2 14     15         22  3576 <NA>      <NA>      North
# i 1 more variable: level <dbl>
```

5. Using a Unix code abbreviation, the degree symbol can be added to the labels.

```
#count the number of characters and add correct numbers and abbreviation of deg or min
maps <- maps %>%
  mutate(minutes = if_else(is.na(minutes),
                           paste0("00", "'"),
                   if_else(str_length(minutes) == 1,
                           paste0(minutes, "0", "'"),
                           paste0(minutes,"'"))),
         degree = if_else(str_length(degree) == 1,
                          paste0(0, degree, "\u00B0"),
                          paste0(degree,"\u00B0"))) #symbol to create a degree `\u00B0`
head(maps, 3)
```

```
# A tibble: 3 x 11
  dir   name    deg_min degree minutes miles  elev alt_name1 alt_name2 wall_hang
  <chr> <chr>     <dbl> <chr>  <chr>   <dbl> <dbl> <chr>     <chr>     <chr>
1 N     Dutchm~     0   00°    00'         0  7417 <NA>      <NA>      North
2 N     Point ~    14   14°    00'        10  5136 <NA>      <NA>      North
3 N     Roxy A~    14.2 14°    15'        22  3576 <NA>      <NA>      North
# i 1 more variable: level <dbl>
```

6. New columns depicting the label information are created.

```
#create a text label, this time using the paste instead of paste0

maps <- maps %>%
  mutate(dir = str_to_upper(dir),  # fix the title case
         label_1 = paste(dir, degree, minutes),
         label_2 = paste(miles, "miles"),
         label_3 = paste(elev,"ft"),
         label_4 = paste(wall_hang, level))  #for easy reference when hanging
head(maps, 3)
```

```
# A tibble: 3 x 15
  dir   name     deg_min degree minutes miles  elev alt_name1 alt_name2 wall_hang
  <chr> <chr>      <dbl> <chr>  <chr>   <dbl> <dbl> <chr>     <chr>     <chr>
1 N     Dutchm~      0   00°    00'         0  7417 <NA>      <NA>      North
2 N     Point ~     14   14°    00'        10  5136 <NA>      <NA>      North
3 N     Roxy A~     14.2 14°    15'        22  3576 <NA>      <NA>      North
# i 5 more variables: level <dbl>, label_1 <chr>, label_2 <chr>, label_3 <chr>,
#   label_4 <chr>
```

> **Logical direction statement**
>
> As data manipulation continues, columns or rows can be lost. Creating a total card summary ensures accountability of all locations at the end of the process.

```
#create a reference to insure no location is left out when subsetting levels
lvl.count <- maps %>%
  group_by(level) %>%
  summarise(total = n())
lvl.count
```

```
# A tibble: 4 x 2
  level total
  <dbl> <int>
1     1    28
2     2    27
3     3    20
4     4    32
```

1.2.1 Creating Functions

1. The next step is creating a function to subset columns and `transpose` them like a `pivot_wider`. Using the transpose gives a mirror reflection of all table data.

```
func.level <- function(x,y) {
  f.level = dplyr::filter(.data = x, level == y)
  reduce = dplyr::select(.data = f.level, name, label_1, label_2, label_3, label_4)
  long.set = as_tibble(t(reduce))
  col.label = c(seq_along(long.set))
  row = set_names(long.set, paste0("col_",col.label))
  row1 = add_row(.data = row, .before = 1)
}
```

2. Call the functions by inputting the data source and level. The data is stored in a new frame.

```
level1 <- func.level(maps,1)
level2 <- func.level(maps,2)
level3 <- func.level(maps,3)
level4 <- func.level(maps,4)
```

3. Unlike the last summary check, there are now four tables to summarize. Using the `map` function against a `list` acts as a functional loop.

```
#check first function results
lvl.list <- list(level1, level2, level3, level4)
match.level <- map_int(.x = lvl.list, .f = length)
lvl.count2 <- lvl.count %>%
  mutate(fn1_count = match.level,
         missing_fn1 = total - fn1_count)
lvl.count2
```

```
# A tibble: 4 x 4
  level total fn1_count missing_fn1
  <dbl> <int>     <int>       <int>
1     1    28        28           0
2     2    27        27           0
3     3    20        20           0
4     4    32        32           0
```

4. In the second function, creative if else statments allow tables of varying lengths and columns falling on even or odd counts. Thus this one function can be run against all four of the current levels.

```
func.binding <- function(setL,inputD) {

  #......Set sequence pattern
  print("Initialize sequence pattern")
  start = seq(from = 1, to = length(inputD), by = setL)
```

```r
end = seq(from = setL, to = length(inputD), by = setL)
print(start)
print(end)

#......Fix sequence pattern
if (length(start) > length(end)) {
  print("......Fixing divisable sequence match")
  end.remainder = c(end, length(inputD))
  #Review range
  print("Adjusted Range Sequence")
  print(start)
  print(end.remainder)
}
else {
  print("......Columns are same length and divisable")
  end.remainder = end
}

#......Set loop conditions
counter = 1
set.col.names = c(paste0("col_", seq_along(1:setL)))
tbl.bind <- tibble(setNames(data.frame(matrix(ncol = setL, nrow = 0)), set.col.names))

#......Build binding table
print("......Counting Loop")
for (i in inputD) {
  if (counter < length(start)) {
    print(paste("Bind row = ", counter))
    index.range = select(inputD, start[counter]:end.remainder[counter])
    index.range = (set_names(index.range, set.col.names))
    tbl.bind = rbind(tbl.bind, index.range)
    counter = counter + 1
  }
  else if (counter == length(start)) {
    print(paste("Bind Final = ", counter))
    index.range = select(inputD,start[counter]:end.remainder[counter])
    index.range = (set_names(index.range, set.col.names[1:as.numeric(length(index.range))]))
    tbl.bind = plyr::rbind.fill(tbl.bind,index.range)
    counter = counter + 1
  }
}
#......Loop ends, add empty row
print("Dummy row added")
tbl.bind = add_row(.data = tbl.bind,.after = nrow(tbl.bind) + 1)
```

```
}
```

5. The two function inputs are the column numbers intended for output and the input table. While not necessary, adding print codes in functions can help find errors. In a production environment, print commands are typically removed as they are for testers. After beta testing, error codes (*try and exempt*) replace print commands.

> **Function inputs**
>
> The great thing about functions is that they can be adjusted repeatedly until the desired input is found. Test out the data using different column sizes.

```
#displayed with outputs
tbl.level1 <- func.binding(5, level1)
```

```
[1] "Initialize sequence pattern"
[1]  1  6 11 16 21 26
[1]  5 10 15 20 25
[1] "......Fixing divisable sequence match"
[1] "Adjusted Range Sequence"
[1]  1  6 11 16 21 26
[1]  5 10 15 20 25 28
[1] "......Counting Loop"
[1] "Bind row =  1"
[1] "Bind row =  2"
[1] "Bind row =  3"
[1] "Bind row =  4"
[1] "Bind row =  5"
[1] "Bind Final =  6"
[1] "Dummy row added"
```

1.3 Creating Flex tables

1.3.1 Fonts with Flextable

1. The required syntax between outputs is widely different between HTML, PDF, Word, Powerpoint, revealJS, and so forth. Google Fonts is a go-to for developers in the HTML space but will not translate through a LaTeX<space> render. Before Quartro, most report writers created tables using `kableextra`, with Rmarkdown and alot of LaTeX<space> syntax, which was finicky. `Flextable` is a new package that does the same thing and works well with Quartro.

2. The `extrafont` or `sysfonts` package are methods to pull existing fonts from the windows library. Google fonts can be imported with the `gdtools` or `showtext` package. `gdtools` is suitable for R shiny or markdown applications, in which quarto is a type of markdown. `Showtext` is used more in graphs, and the `extrafont` is more for pdf text blocks.

Google Fonts

Google Fonts can be displayed while viewing them in Rstudio IDE. But once rendered, the code breaks unless you hide the code from the render engine. A workaround is to write the table, hide the code, save the table as an image, import, and display.

1.3.2 Creating a Windows font table

1. The user would need to review all the windows fonts, curate a list of names and build a table in the similar manner as previous example. This has already been done, but this time as `.txt` file for practice.

```
#read in a .txt file with a list of favorite windows formats
win.tbl <- read_delim(here("windowsFonts.txt"), delim = ",", col_names = TRUE, trim_ws = TRUE) %>% clean_names()
head(win.tbl,3)
```

```
# A tibble: 3 x 3
  font                 type         usage
  <chr>                <chr>        <chr>
1 baskerville old face professional sans serif
2 bookman old style    professional sans serif
3 cambria              professional sans serif
```

2. Not all data is perfect when received. The example data comes in a mixture of letter casing. The cleaning will replace the lowercase with the title case and then uppercase the specific letters to match the Windows database.

```
win.tbl <- win.tbl %>%
  mutate(font = str_to_title(font)) %>%  # Title case
  mutate(font = str_replace(font, "Itc", str_to_upper)) %>% # Replace title case with uppercase
  mutate(font = str_replace(font, "Ms", str_to_upper)) %>%
  mutate(font = str_replace(font, "Clm", str_to_upper)) %>%
  mutate(font = str_replace(font, "Ui", str_to_upper)) %>%
  mutate(font = str_replace(font, "Gd", str_to_upper)) %>%
    mutate(font = str_replace(font, "Lt", str_to_upper)) %>%
  arrange(type, usage, font) %>%
  mutate(label = "Acorn Woman L.O. 282\u00B0 21.5 mi. 7055' elev") %>%
  mutate(item = 1:nrow(win.tbl), .before=font)
```

3. Index each column containing the font name, then set the column containing the string to change. Misspelled font names will not return the correct font, and the PDF render engine will fail.

```r
#create a flextable
pretty.font <- flextable(win.tbl) %>%
  border_outer(part = "all") %>% #add some border
  border_inner(part = "all") %>%
  bold(part = "header") %>%
  autofit()

#Set a different font for each associated column
pretty.font1 <- pretty.font %>%
  font(fontname = as.character(win.tbl[1,2]), i = 1, j = 5) %>%
  font(fontname = as.character(win.tbl[2,2]), i = 2, j = 5) %>%
  font(fontname = as.character(win.tbl[3,2]), i = 3, j = 5) %>%
  font(fontname = as.character(win.tbl[4,2]), i = 4, j = 5) %>%
  font(fontname = as.character(win.tbl[5,2]), i = 5, j = 5) %>%
  font(fontname = as.character(win.tbl[6,2]), i = 6, j = 5) %>%
  font(fontname = as.character(win.tbl[7,2]), i = 7, j = 5) %>%
  font(fontname = as.character(win.tbl[8,2]), i = 8, j = 5) %>%
  font(fontname = as.character(win.tbl[9,2]), i = 9, j = 5) %>%
  font(fontname = as.character(win.tbl[10,2]), i = 10, j = 5) %>%
  font(fontname = as.character(win.tbl[11,2]), i = 11, j = 5) %>%
  font(fontname = as.character(win.tbl[12,2]), i = 12, j = 5) %>%
  font(fontname = as.character(win.tbl[13,2]), i = 13, j = 5) %>%
  font(fontname = as.character(win.tbl[14,2]), i = 14, j = 5) %>%
  font(fontname = as.character(win.tbl[15,2]), i = 15, j = 5) %>%
  font(fontname = as.character(win.tbl[16,2]), i = 16, j = 5) %>%
  font(fontname = as.character(win.tbl[17,2]), i = 17, j = 5) %>%
  font(fontname = as.character(win.tbl[18,2]), i = 18, j = 5) %>%
  font(fontname = as.character(win.tbl[19,2]), i = 19, j = 5) %>%
  font(fontname = as.character(win.tbl[20,2]), i = 20, j = 5) %>%
  font(fontname = as.character(win.tbl[21,2]), i = 21, j = 5) %>%
  font(fontname = as.character(win.tbl[22,2]), i = 22, j = 5) %>%
  font(fontname = as.character(win.tbl[23,2]), i = 23, j = 5)

pretty.font2 <- theme_zebra(pretty.font1)
pretty.font2
```

Table 1.2

item	font	type	usage	label
1	Bradley Hand ITC	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055′ elev
2	Comic Sans MS	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055′ elev
3	Ink Free	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055′ elev

item	font	type	usage	label
4	Lucida Calligraphy	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
5	Papyrus	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
6	Segoe Print	art	handwrite	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
7	Maiandra GD	hybrid	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
8	Source Code Pro	hybrid	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
9	Sitka Small	hybrid	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
10	Arial	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
11	Avenir LT Pro	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
12	Century Gothic	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
13	Liberation Sans	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
14	Miriam Mono CLM	professional	sans	Acorn Woman L.O. 282⍰ 21.5 mi. 7055’ elev
15	Rubik	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
16	Segoe UI	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
17	Source Sans Pro	professional	sans	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
18	Baskerville Old Face	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
19	Bookman Old Style	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
20	Cambria	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
21	Century	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
22	Garamond	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev
23	Times New Roman	professional	sans serif	Acorn Woman L.O. 282° 21.5 mi. 7055’ elev

1.3.3 Combining Font Type, Color and Size

1. Create color combos using the USFS Agency color list.

> ⚠ Color Palette
>
> Using light colors is too hard to read in cupolas. Use dark colors such as forest green or black. Using a greyscale to save on print costs when printing in large quantities. For web design, ensure the colors are 508 compliant.

```
black <- "#000000"
brown <- "#6a5147"
green.forest <- "#005838"
green.mint <- "#62C1AC"
grey <- "#A9B1B6"
yellow <- "#ffd51d"
white <- "#FFFFFF"
```

2. Set the defaults for `flexible`, which reduces the length of the flexible code chunk.

```
init_flextable_defaults()

fontname <- as.character(win.tbl[7,2])
box.border <- fp_border_default(color = grey, width = 2, style = "solid")
```

3. The card size chosen (font size) will dictate how many rows can be displayed on one page. Automation can be accomplished with `lappy`, but lacks customization. Therefore, the user must subset the data and run as many tables as required to fit on a page.

```
#subset rows using slice
tbl.level1a <- tbl.level1 %>% slice(1:25)
tbl.level1b <- tbl.level1 %>% slice(25:37)

tbl.level2a <- tbl.level2 %>% slice(1:25)
tbl.level2b <- tbl.level2 %>% slice(25:37)

#no subset for tbl.level3

tbl.level4a <- tbl.level4 %>% slice(1:25)
tbl.level4b <- tbl.level4 %>% slice(25:43)
```

4. Create all the required flex tables. Flex table requires manually inputting the row numbers that must be changed for font sizes and styles.

```
ft <- flextable(tbl.level1a) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2, 8, 14, 20)) %>%
  fontsize(i =  c(2, 8, 14, 20), size = 15) %>%
  fontsize(i = c(3,4,5,9,10,11,15,16,17,21,22,23), size = 10) %>%
  fontsize(i = c(6, 12, 18, 24), size = 5) %>%
  color(color = brown) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex1a <- ft

ft <- flextable(tbl.level1b) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2,8)) %>%
  fontsize(i =  c(2,8), size = 15) %>%
  fontsize(i = c(3,4,5,9,10,11), size = 10) %>%
  fontsize(i = c(6,12), size = 5) %>%
  color(color = brown) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex1b <- ft
```

```
ft <- flextable(tbl.level2a) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2, 8, 14, 20)) %>%
  fontsize(i =  c(2, 8, 14, 20), size = 15) %>%
  fontsize(i = c(3,4,5,9,10,11,15,16,17,21,22,23), size = 10) %>%
  fontsize(i = c(6, 12, 18, 24), size = 5) %>%
  color(color = green.forest) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex2a <- ft

ft <- flextable(tbl.level2b) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2,8)) %>%
  fontsize(i =  c(2,8), size = 15) %>%
  fontsize(i = c(3,4,5,9,10,11), size = 10) %>%
  fontsize(i = c(6,12), size = 5) %>%
  color(color = green.forest) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex2b <- ft
```

```
ft <- flextable(tbl.level3) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2, 8, 14, 20)) %>%
  fontsize(i =  c(2, 8, 14, 20), size = 14) %>%
  fontsize(i = c(3,4,5,9,10,11,15,16,17,21,22,23), size = 9) %>%
  fontsize(i = c(6, 12, 18, 24), size = 5) %>%
  color(color = brown) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex3 <- ft
```

```
ft <- flextable(tbl.level4a) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2, 8, 14, 20)) %>%
  fontsize(i =  c(2, 8, 14, 20), size = 14) %>%
  fontsize(i = c(3,4,5,9,10,11,15,16,17,21,22,23), size = 9) %>%
  fontsize(i = c(6, 12, 18, 24), size = 5) %>%
  color(color = green.forest) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex4a <- ft

ft <- flextable(tbl.level4b) %>%
  delete_part("header") %>%
  border_inner_v(border = box.border, part = "all") %>%
  border_outer(border = box.border) %>%
  bold(i =  c(2,8,14)) %>%
  fontsize(i =  c(2,8,14), size = 14) %>%
  fontsize(i = c(3,4,5,9,10,11,15,16,17), size = 9) %>%
  fontsize(i = c(6,12,18), size = 5) %>%
  color(color = green.forest) %>%
  align_text_col(align = "center") %>%
  font(fontname = fontname, part = "all") %>%
  autofit()
flex4b <- ft
```

```
flex1a
```

Dutchman Peak L.o. N 00° 00' 0 miles 7417 ft North 1	Point Mtn N 14° 00' 10 miles 5136 ft North 1	Little Red Mtn N 21° 00' 1.5 miles 6654 ft North 1	Bald Mtn N 25° 00' 9 miles 5628 ft North 1	Wagner Butte NE 46° 30' 8.5 miles 7140 ft North 1
Big Red Mtn E 70° 00' 2.25 miles 7040 ft East 1	Mcdonald Peak E 70° 00' 7.5 miles 7225 ft East 1	Mt. Ashland E 73° 40' 9 miles 7533 ft East 1	Siskiyou Peak E 79° 00' 6 miles 7120 ft East 1	Sterling Mtn E 139° 00' 4 miles 6800 ft South 1
Observation Peak S 177° 00' 2 miles 7340 ft South 1	Dry Lake Mtn S 195° 30' 9.5 miles 6775 ft South 1	Big Rock S 207° 00' 6 miles 6852 ft South 1	Condrey Mtn SW 212° 00' 8.5 miles 7112 ft South 1	Donomore Peak SW 212° 50' 3 miles 6500 ft South 1
Scraggy Mtn SW 224° 30' 8 miles 7013 ft South 1	White Mtn SW 234° 10' 10 miles 6460 ft West 1	Lilly Mtn W 242° 00' 5.5 miles 5850 ft West 1	Silver Fork Gap W 267° 30' 1.5 miles 4900 ft West 1	Iron Knob W 270° 30' 9.5 miles 3722 ft West 1

Table 1.3Level 1a Print

```
flex2a
```

Pilot Rock E 93° 00’ 17 miles 5910 ft East 2	I-5 Mile Marker 1 E 107° 00’ 15 miles NA ft East 2	Cotton Wood Peak E 130° 30’ 16.5 miles 6607 ft East 2	Collins Baldy L.o. S 189° 50’ 19 miles 5493 ft South 2	Upper Devils Peak SW 237° 10’ 19 miles 6004 ft West 2
Kangaroo Mtn W 242° 00’ 17.5 miles 6694 ft West 2	Red Buttes W 243° 00’ 17 miles 6739 ft West 2	Rattlesnake Mtn W 246° 00’ 19.5 miles 6307 ft West 2	Preston Peak W 251° 00’ 13.5 miles 4934 ft West 2	Stein Butte W 260° 00’ 11 miles 4400 ft West 2
Whiskey Peak W 265° 00’ 19 miles 6497 ft West 2	Hartish Park W 271° 50’ 12.5 miles 1500 ft West 2	Collings Mtn W 272° 00’ 13 miles 3625 ft West 2	Steve Peak W 275° 50’ 19 miles 5835 ft West 2	Kinney Mtn W 281° 00’ 13 miles 4518 ft West 2
Boaz Mtn NW 307° 45’ 10.5 miles 3504 ft West 2	Tallow Box L.o. NW 308° 30’ 17 miles 5023 ft West 2	Burton Butte NW 311° 20’ 14 miles 4400 ft West 2	Mt. Baldy NW 313° 00’ 17 miles 4974 ft West 2	Ben Johnson Mtn NW 317° 00’ 15 miles 4395 ft West 2

Table 1.4Level 2a Print

```
flex1b
```

Deadmans Point W 278° 20' 2 miles 5500 ft West 1	Little Grey Back Mtn W 279° 30' 8 miles 5083 ft West 1	Acron Peak L.o. W 286° 15' 6 miles 4984 ft West 1	Mule Mtn W 288° 00' 10 miles 3532 ft West 1	Baldy Peak W 294° 00' 8.5 miles 4645 ft West 1
Yellow Jacket Mtn W 300° 00' 3 miles 6250 ft West 1	Cinnabar Mtn NW 315° 20' 8.5 miles 3918 ft West 1	Kenny Meadows N 335° 00' 5 miles 2556 ft North 1		

Table 1.5Level 1b Print

```
flex2b
```

Wellington Butte NW 324° 00' 19 miles 3705 ft North 2	Squires Peak N 330° 00' 14 miles 3316 ft North 2	Buncom N 330° 10' 10.5 miles NA ft North 2	Woodrat Mtn N 335° 10' 14 miles 4124 ft North 2	Nelson Mtn N 354° 30' 12 miles 3509 ft North 2
Anderson Butte N 358° 00' 11 miles 5197 ft North 2	Nine Mile Peak N 359° 00' 16.5 miles 4828 ft North 2			

Table 1.6Level 2b Print

```
flex3
```

Roxy Ann N 14° 15' 22 miles 3576 ft North 3	Grizzly Peak NE 42° 00' 21 miles 5922 ft North 3	Soda Mtn L.o. E 88° 00' 21 miles 6091 ft East 3	Black Mtn E 123° 30' 23 miles 5000 ft East 3	Badger Mtn E 140° 30' 21 miles 4980 ft South 3
Gunsight Peak S 164° 30' 22 miles 6146 ft South 3	Indian Creek Baldy S 171° 30' 21 miles 6275 ft South 3	Tom Martin Peak S 204° 30' 23 miles 7021 ft South 3	Lake Mtn SW 213° 00' 28 miles 6900 ft South 3	Buck Peak W 257° 45' 22 miles 7000 ft West 3
Pyramid Peak W 259° 45' 23.5 miles 6451 ft West 3	Lake Peak W 265° 00' 21 miles 6648 ft West 3	Bolan Mtn W 266° 30' 29 miles 6269 ft West 3	Swan Mtn W 268° 00' 24.5 miles 6272 ft West 3	Craggy Mtn W 271° 45' 24 miles 6300 ft West 3
Grayback Mtn W 282° 00' 21.5 miles 7055 ft West 3	Mt. Isabelle NW 328° 50' 21 miles 4494 ft North 3	Timber Mtn N 335° 00' 21.5 miles 4424 ft North 3	Lower Table Rock N 353° 45' 28 miles 2044 ft North 3	Upper Table Rock N 357° 40' 30 miles 2091 ft North 3

Table 1.7Level 3 Print

```
flex4a
```

Diamond Peak	N 20° 15’	109 miles	8744 ft	North 4
Mt. Bailey	N 23° 40’	84 miles	8366 ft	North 4
Mt. Theilson	N 28° 30’	87 miles	9182 ft	North 4
The Watchman L.o.	NE 32° 00’	72 miles	8013 ft	North 4
Rustler Peak L.o.	NE 35° 00’	48 miles	6248 ft	North 4
Mt. Scott L.o.	NE 36° 00’	75 miles	8929 ft	North 4
Mt. Mcgloughlin	NE 46° 30’	40.5 miles	9495 ft	North 4
Robinson Butte L.o.	NE 52° 45’	34 miles	5864 ft	East 4
Brown Mtn	NE 55° 00’	39 miles	7311 ft	East 4
Pelican Butte L.o.	E 65° 30’	47 miles	8208 ft	East 4
Hammaker Mtn	E 87° 40’	47 miles	6565 ft	East 4
Mount Dome	E 104° 15’	64 miles	6518 ft	East 4
Ball Mtn L.o.	E 114° 15’	42 miles	9000 ft	East 4
Willow Creek Mtn	E 115° 00’	37 miles	7845 ft	East 4
Goosenest Mtn	E 122° 50’	41 miles	8280 ft	East 4
Herd Peak L.o.	E 130° 00’	44 miles	7071 ft	East 4
Mt. Shasta	E 139° 00’	52 miles	14162 ft	South 4
Black Butte	E 149° 00’	50 miles	6325 ft	South 4
The Eddys	S 157° 00’	65 miles	9025 ft	South 4
China Mtn	S 160° 30’	48 miles	8542 ft	South 4

Table 1.8Level 4a Print

```
flex4b
```

South China S 161° 50' 49 miles 8206 ft South 4	Duzel Rock L.o. S 166° 30' 36 miles 6039 ft South 4	Bolivar L.o. S 174° 30' 54 miles 6900 ft South 4	Thompson Peak S 186° 30' 76 miles 9002 ft South 4	Boulder Peak S 198° 10' 40 miles 8299 ft South 4
Kings Castle NA 210° 05' 35 miles 7405 ft South 4	El Captian W 252° 00' 36 miles 6839 ft West 4	Persol Peak W 290° 00' 52 miles 5096 ft West 4	Onion Mtn L.o. NW 308° 00' 47 miles 4438 ft West 4	Sexton Mtn L.o. NW 328° 00' 46 miles 3834 ft North 4
King Mtn N 338° 40' 48 miles 4340 ft North 4	Round Top L.o. N 353° 00' 47 miles 4658 ft North 4			

Table 1.9Level 4b Print

1.4 Saving Data

1. Lots of great information has undergone the **data wrangling** process. The data should be saved for future uses. The data can be exported as .odt, .txt, .csv, .excel, and other formats. The easiest way to recall or import the data into other programs is the csv file.

```
write_csv(x = maps, file = here("data_output/maps.csv"),col_names = TRUE)
write_csv(x = win.tbl, file = here("data_output/winfonts.csv"),col_names = TRUE)
```