STA5092Z EDA Lecture 5-6 - Happiness Data

Table of contents

Checking your data - Checklist

- Formulate your question
- Check your data
- Automate your project workflow

We will go through these using the Boston, Ames, World Happiness Datasets. Before moving on, we will look at the code chunk options:

- `echo`: Show (TRUE) or hide (FALSE) the source code in output,
- `eval`: Execute (TRUE) or skip (FALSE) the code,

- `include`: Include both code and output in the rendered document (FALSE hides both),
- `warning`: Show (TRUE) or suppress (FALSE) warnings,
- `message`: Show (TRUE) or suppress (FALSE) messages,
- `results`: "markup" (default), "asis", "hide", "hold". Controls how results are displayed,
- `cache`: Cache results for faster re-rendering,
- `error`: Show errors in output (TRUE) or stop rendering (FALSE).

Libraries

```
if (!require("pacman")) install.packages("pacman")
```

```
Loading required package: pacman
```

```
pacman::p_load(here, data.table, purrrlyr, reshape2,
               tidyverse,  flextable, SmartEDA, DataExplorer, DT,
               inspectdf, lubridate, janitor, forcats,
               fastDummies, units, tsibble, feasts, fable,
               tmap, tmaptools, mapdeck, leaflet, leafgl,
               rgeoda, osmdata,
               exactextractr, geomerge, hereR, ggmap,
               kableExtra, knitr,
               colourvalues, viridis,
               readxl, rio, fst,
               tictoc, beepr,
               ggfortify, gganimate,
               grateful)
```

World Happiness Dataset

Download the dataset from the following link:

https://www.kaggle.com/unsdsn/world-happiness

You can see some basic characteristics of the dataset with the dim() and str() functions.

```
happiness2019 <- read.csv(here::here("Datasets", "WorldHappiness", "2019.csv"))

happiness2018 <- read.csv(here::here("Datasets", "WorldHappiness", "2018.csv"))
```

```r
happiness2017 <- read.csv(here::here("Datasets", "WorldHappiness", "2017.csv"))

happiness2016 <- read.csv(here::here("Datasets", "WorldHappiness", "2016.csv"))

happiness2015 <- read.csv(here::here("Datasets", "WorldHappiness", "2015.csv"))
```

Before we merge these years together in one file, we might want to create a year variable in each one.

```r
happiness2015 <- mutate(happiness2015, year = 2015)
str(happiness2015)
```

```
'data.frame':   158 obs. of  13 variables:
 $ Country                      : chr  "Switzerland" "Iceland" "Denmark" "Norway" ...
 $ Region                       : chr  "Western Europe" "Western Europe" "Western Europe" "We
 $ Happiness.Rank               : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Happiness.Score              : num  7.59 7.56 7.53 7.52 7.43 ...
 $ Standard.Error               : num  0.0341 0.0488 0.0333 0.0388 0.0355 ...
 $ Economy..GDP.per.Capita.     : num  1.4 1.3 1.33 1.46 1.33 ...
 $ Family                       : num  1.35 1.4 1.36 1.33 1.32 ...
 $ Health..Life.Expectancy.     : num  0.941 0.948 0.875 0.885 0.906 ...
 $ Freedom                      : num  0.666 0.629 0.649 0.67 0.633 ...
 $ Trust..Government.Corruption.: num  0.42 0.141 0.484 0.365 0.33 ...
 $ Generosity                   : num  0.297 0.436 0.341 0.347 0.458 ...
 $ Dystopia.Residual            : num  2.52 2.7 2.49 2.47 2.45 ...
 $ year                         : num  2015 2015 2015 2015 2015 ...
```

```r
happiness2016 <- mutate(happiness2016, year = 2016)
str(happiness2016)
```

```
'data.frame':   157 obs. of  14 variables:
 $ Country                      : chr  "Denmark" "Switzerland" "Iceland" "Norway" ...
 $ Region                       : chr  "Western Europe" "Western Europe" "Western Europe" "We
 $ Happiness.Rank               : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Happiness.Score              : num  7.53 7.51 7.5 7.5 7.41 ...
 $ Lower.Confidence.Interval    : num  7.46 7.43 7.33 7.42 7.35 ...
 $ Upper.Confidence.Interval    : num  7.59 7.59 7.67 7.58 7.47 ...
 $ Economy..GDP.per.Capita.     : num  1.44 1.53 1.43 1.58 1.41 ...
 $ Family                       : num  1.16 1.15 1.18 1.13 1.13 ...
 $ Health..Life.Expectancy.     : num  0.795 0.863 0.867 0.796 0.811 ...
```

```
 $ Freedom                        : num  0.579 0.586 0.566 0.596 0.571 ...
 $ Trust..Government.Corruption.: num  0.445 0.412 0.15 0.358 0.41 ...
 $ Generosity                     : num  0.362 0.281 0.477 0.379 0.255 ...
 $ Dystopia.Residual              : num  2.74 2.69 2.83 2.66 2.83 ...
 $ year                           : num  2016 2016 2016 2016 2016 ...
```

```
happiness2017 <- mutate(happiness2017, year = 2017)
str(happiness2017)
```

```
'data.frame':   155 obs. of  13 variables:
 $ Country                        : chr  "Norway" "Denmark" "Iceland" "Switzerland" ...
 $ Happiness.Rank                 : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Happiness.Score                : num  7.54 7.52 7.5 7.49 7.47 ...
 $ Whisker.high                   : num  7.59 7.58 7.62 7.56 7.53 ...
 $ Whisker.low                    : num  7.48 7.46 7.39 7.43 7.41 ...
 $ Economy..GDP.per.Capita.       : num  1.62 1.48 1.48 1.56 1.44 ...
 $ Family                         : num  1.53 1.55 1.61 1.52 1.54 ...
 $ Health..Life.Expectancy.       : num  0.797 0.793 0.834 0.858 0.809 ...
 $ Freedom                        : num  0.635 0.626 0.627 0.62 0.618 ...
 $ Generosity                     : num  0.362 0.355 0.476 0.291 0.245 ...
 $ Trust..Government.Corruption.: num  0.316 0.401 0.154 0.367 0.383 ...
 $ Dystopia.Residual              : num  2.28 2.31 2.32 2.28 2.43 ...
 $ year                           : num  2017 2017 2017 2017 2017 ...
```

```
happiness2018 <- mutate(happiness2018, year = 2018)
str(happiness2018)
```

```
'data.frame':   156 obs. of  10 variables:
 $ Overall.rank                 : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Country.or.region            : chr  "Finland" "Norway" "Denmark" "Iceland" ...
 $ Score                        : num  7.63 7.59 7.55 7.5 7.49 ...
 $ GDP.per.capita               : num  1.3 1.46 1.35 1.34 1.42 ...
 $ Social.support               : num  1.59 1.58 1.59 1.64 1.55 ...
 $ Healthy.life.expectancy      : num  0.874 0.861 0.868 0.914 0.927 0.878 0.896 0.876 0.913 0
 $ Freedom.to.make.life.choices: num  0.681 0.686 0.683 0.677 0.66 0.638 0.653 0.669 0.659 0
 $ Generosity                   : num  0.202 0.286 0.284 0.353 0.256 0.333 0.321 0.365 0.285 0
 $ Perceptions.of.corruption    : chr  "0.393" "0.340" "0.408" "0.138" ...
 $ year                         : num  2018 2018 2018 2018 2018 ...
```

```
happiness2019 <- mutate(happiness2019, year = 2019)
str(happiness2019)
```

```
'data.frame':   156 obs. of  10 variables:
 $ Overall.rank                : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Country.or.region           : chr  "Finland" "Denmark" "Norway" "Iceland" ...
 $ Score                       : num  7.77 7.6 7.55 7.49 7.49 ...
 $ GDP.per.capita              : num  1.34 1.38 1.49 1.38 1.4 ...
 $ Social.support              : num  1.59 1.57 1.58 1.62 1.52 ...
 $ Healthy.life.expectancy     : num  0.986 0.996 1.028 1.026 0.999 ...
 $ Freedom.to.make.life.choices: num  0.596 0.592 0.603 0.591 0.557 0.572 0.574 0.585 0.584 0
 $ Generosity                  : num  0.153 0.252 0.271 0.354 0.322 0.263 0.267 0.33 0.285 0
 $ Perceptions.of.corruption   : num  0.393 0.41 0.341 0.118 0.298 0.343 0.373 0.38 0.308 0.1
 $ year                        : num  2019 2019 2019 2019 2019 ...
```

`Join` functions

We have seen these functions before.

`Bind` functions

`Row bind`

`bind_rows()` function is similar to rbind, however this one matches the variable names and if there is no such similar variable in one of the files, it creates a variable with NAs.

Here in our example, simply

```
happiness15_19 <- bind_rows(happiness2015, happiness2016, happiness2017, happiness2018, happ
```

We get an error! What are the problems here?

- The names of the variables are not consistent such as:
- Country
- Economy..GDP.per.Capita. into GDP.per.capita
- Freedom.to.make.life.choices into Freedom
- Happiness.Score into

```
names(happiness2015)
```

```
 [1] "Country"                          "Region"
 [3] "Happiness.Rank"                   "Happiness.Score"
 [5] "Standard.Error"                   "Economy..GDP.per.Capita."
 [7] "Family"                           "Health..Life.Expectancy."
 [9] "Freedom"                          "Trust..Government.Corruption."
[11] "Generosity"                       "Dystopia.Residual"
[13] "year"
```

```
happiness2015 <- happiness2015 |> dplyr::select(c(1:4,6:11,13))
happiness2015 = happiness2015 |> rename(Rank = Happiness.Rank,
                                        Score = Happiness.Score,
                                        GDP.per.capita = Economy..GDP.per.Capita.,
                                        Life.expectancy = Health..Life.Expectancy.,
                                        Trust = Trust..Government.Corruption.)
names(happiness2015)
```

```
 [1] "Country"         "Region"          "Rank"            "Score"
 [5] "GDP.per.capita"  "Family"          "Life.expectancy" "Freedom"
 [9] "Trust"           "Generosity"      "year"
```

```
names(happiness2016)
```

```
 [1] "Country"                          "Region"
 [3] "Happiness.Rank"                   "Happiness.Score"
 [5] "Lower.Confidence.Interval"        "Upper.Confidence.Interval"
 [7] "Economy..GDP.per.Capita."         "Family"
 [9] "Health..Life.Expectancy."         "Freedom"
[11] "Trust..Government.Corruption."    "Generosity"
[13] "Dystopia.Residual"                "year"
```

```
happiness2016 <- happiness2016 |> dplyr::select(c(1:4,7:12,14))
happiness2016 = happiness2016 |> rename(Rank = Happiness.Rank,
                                        Score = Happiness.Score,
                                        GDP.per.capita = Economy..GDP.per.Capita.,
                                        Life.expectancy = Health..Life.Expectancy.,
                                        Trust = Trust..Government.Corruption.)
names(happiness2016)
```

```
 [1] "Country"         "Region"          "Rank"            "Score"
 [5] "GDP.per.capita"  "Family"          "Life.expectancy" "Freedom"
 [9] "Trust"           "Generosity"      "year"
```

```
names(happiness2017)
```

```
 [1] "Country"                        "Happiness.Rank"
 [3] "Happiness.Score"                "Whisker.high"
 [5] "Whisker.low"                    "Economy..GDP.per.Capita."
 [7] "Family"                         "Health..Life.Expectancy."
 [9] "Freedom"                        "Generosity"
[11] "Trust..Government.Corruption." "Dystopia.Residual"
[13] "year"
```

```
happiness2017 <- happiness2017 |> dplyr::select(c(1:3,6:11,13))
happiness2017 = happiness2017 |> rename(Rank = Happiness.Rank,
                                        Score = Happiness.Score,
                                        GDP.per.capita = Economy..GDP.per.Capita.,
                                        Life.expectancy = Health..Life.Expectancy.,
                                        Trust = Trust..Government.Corruption.)
names(happiness2017)
```

```
 [1] "Country"         "Rank"            "Score"           "GDP.per.capita"
 [5] "Family"          "Life.expectancy" "Freedom"         "Generosity"
 [9] "Trust"           "year"
```

```
names(happiness2018)
```

```
 [1] "Overall.rank"                "Country.or.region"
 [3] "Score"                       "GDP.per.capita"
 [5] "Social.support"              "Healthy.life.expectancy"
 [7] "Freedom.to.make.life.choices" "Generosity"
 [9] "Perceptions.of.corruption"   "year"
```

```
happiness2018 = happiness2018 |> rename(Country = Country.or.region,
                                        Rank = Overall.rank,
                                        Life.expectancy = Healthy.life.expectancy,
                                        Freedom = Freedom.to.make.life.choices,
                                        Trust = Perceptions.of.corruption)
names(happiness2018)
```

```
 [1] "Rank"            "Country"         "Score"           "GDP.per.capita"
 [5] "Social.support"  "Life.expectancy" "Freedom"         "Generosity"
 [9] "Trust"           "year"
```

```
names(happiness2019)
```

```
 [1] "Overall.rank"                "Country.or.region"
 [3] "Score"                       "GDP.per.capita"
 [5] "Social.support"              "Healthy.life.expectancy"
 [7] "Freedom.to.make.life.choices" "Generosity"
 [9] "Perceptions.of.corruption"   "year"
```

```
happiness2019 = happiness2019 |> rename(Country = Country.or.region,
                                        Rank = Overall.rank,
                                        Life.expectancy = Healthy.life.expectancy,
                                        Freedom = Freedom.to.make.life.choices,
                                        Trust = Perceptions.of.corruption)
names(happiness2019)
```

```
 [1] "Rank"            "Country"         "Score"           "GDP.per.capita"
 [5] "Social.support"  "Life.expectancy" "Freedom"         "Generosity"
 [9] "Trust"           "year"
```

- Perceptions.of.corruption (or Trust in the new name) in 2018 appears as a factor variable. We need to convert this into numeric.

```
str(happiness2018)
```

```
'data.frame':   156 obs. of  10 variables:
 $ Rank           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Country        : chr  "Finland" "Norway" "Denmark" "Iceland" ...
 $ Score          : num  7.63 7.59 7.55 7.5 7.49 ...
 $ GDP.per.capita : num  1.3 1.46 1.35 1.34 1.42 ...
 $ Social.support : num  1.59 1.58 1.59 1.64 1.55 ...
 $ Life.expectancy: num  0.874 0.861 0.868 0.914 0.927 0.878 0.896 0.876 0.913 0.91 ...
 $ Freedom        : num  0.681 0.686 0.683 0.677 0.66 0.638 0.653 0.669 0.659 0.647 ...
 $ Generosity     : num  0.202 0.286 0.284 0.353 0.256 0.333 0.321 0.365 0.285 0.361 ...
 $ Trust          : chr  "0.393" "0.340" "0.408" "0.138" ...
 $ year           : num  2018 2018 2018 2018 2018 ...
```

```
happiness2018 |> arrange(desc(Trust)) # to have an idea of what the problem is, sort your col
```

	Rank	Country	Score	GDP.per.capita	Social.support
1	20	United Arab Emirates	6.774	2.096	0.776
2	34	Singapore	6.343	1.529	1.451
3	151	Rwanda	3.408	0.332	0.896
4	3	Denmark	7.555	1.351	1.590
5	1	Finland	7.632	1.305	1.592
6	8	New Zealand	7.324	1.268	1.601
7	9	Sweden	7.314	1.355	1.501
8	5	Switzerland	7.487	1.420	1.549
9	2	Norway	7.594	1.456	1.582
10	17	Luxembourg	6.910	1.576	1.520
11	14	Ireland	6.977	1.448	1.583
12	10	Australia	7.272	1.340	1.573
13	6	Netherlands	7.441	1.361	1.488
14	7	Canada	7.328	1.330	1.532
15	76	Hong Kong	5.430	1.405	1.290
16	98	Somalia	4.982	0.000	0.712
17	15	Germany	6.965	1.340	1.474
18	19	Israel	6.814	1.301	1.559
19	44	Uzbekistan	6.096	0.719	1.584
20	16	Belgium	6.927	1.324	1.483
21	12	Austria	7.139	1.341	1.504
22	128	Georgia	4.340	0.853	0.592
23	130	Myanmar	4.308	0.682	1.174
24	23	France	6.489	1.293	1.466
25	87	Azerbaijan	5.201	1.024	1.161
26	63	Estonia	5.739	1.200	1.532
27	110	Laos	4.623	0.720	1.034
28	32	Qatar	6.374	1.649	1.303
29	97	Bhutan	5.082	0.796	1.335
30	123	Mozambique	4.417	0.198	0.902
31	31	Uruguay	6.379	1.093	1.459
32	58	Northern Cyprus	5.835	1.229	1.211
33	73	Belarus	5.483	1.039	1.498
34	54	Japan	5.915	1.294	1.462
35	70	Libya	5.566	0.985	1.350
36	127	Ethiopia	4.350	0.308	0.950
37	115	Bangladesh	4.500	0.532	0.850
38	150	Syria	3.462	0.689	0.382
39	22	Malta	6.627	1.270	1.525

40	4	Iceland	7.495	1.343	1.644
41	80	Lebanon	5.358	0.965	1.179
42	84	Algeria	5.295	0.979	1.154
43	90	Jordan	5.161	0.822	1.265
44	18	United States	6.886	1.398	1.471
45	106	Iran	4.707	1.059	0.771
46	41	Nicaragua	6.141	0.668	1.319
47	121	Burkina Faso	4.424	0.314	1.097
48	33	Saudi Arabia	6.371	1.379	1.331
49	43	Bahrain	6.105	1.338	1.366
50	60	Kazakhstan	5.790	1.143	1.516
51	48	Ecuador	5.973	0.889	1.330
52	75	Pakistan	5.472	0.652	0.810
53	141	Lesotho	3.808	0.472	1.215
54	74	Turkey	5.483	1.148	1.380
55	122	Egypt	4.419	0.885	1.025
56	45	Kuwait	6.083	1.474	1.301
57	83	Dominican Republic	5.302	0.982	1.441
58	154	South Sudan	3.254	0.337	0.608
59	71	Philippines	5.524	0.775	1.312
60	148	Haiti	3.582	0.315	0.714
61	86	China	5.246	0.989	1.142
62	107	Ivory Coast	4.671	0.541	0.872
63	13	Costa Rica	7.072	1.010	1.459
64	139	Togo	3.999	0.259	0.474
65	134	Niger	4.166	0.131	0.867
66	144	Zimbabwe	3.692	0.357	1.094
67	153	Tanzania	3.303	0.455	0.991
68	24	Mexico	6.488	1.038	1.252
69	117	Iraq	4.456	1.010	0.971
70	140	Guinea	3.964	0.344	0.792
71	126	Mauritania	4.356	0.557	1.245
72	133	India	4.190	0.721	0.747
73	146	Botswana	3.590	1.017	1.174
74	49	Belize	5.956	0.807	1.101
75	28	Brazil	6.419	0.986	1.474
76	11	United Kingdom	7.190	1.244	1.433
77	40	El Salvador	6.167	0.806	1.231
78	109	Senegal	4.631	0.429	1.117
79	125	Zambia	4.377	0.562	1.047
80	137	Sudan	4.139	0.605	1.240
81	81	Montenegro	5.347	1.017	1.279
82	147	Malawi	3.587	0.186	0.541

83	95	Vietnam	5.103	0.715	1.365
84	26	Taiwan	6.441	1.365	1.436
85	101	Nepal	4.880	0.425	1.228
86	114	Congo (Brazzaville)	4.559	0.682	0.811
87	104	Palestinian Territories	4.743	0.642	1.217
88	156	Burundi	2.905	0.091	0.627
89	36	Spain	6.310	1.251	1.538
90	64	Paraguay	5.681	0.835	1.522
91	72	Honduras	5.504	0.620	1.205
92	85	Morocco	5.254	0.779	0.797
93	30	Guatemala	6.382	0.781	1.268
94	136	Benin	4.141	0.378	0.372
95	53	Latvia	5.933	1.148	1.454
96	120	Cambodia	4.433	0.549	1.088
97	42	Poland	6.123	1.176	1.448
98	119	Namibia	4.441	0.874	1.281
99	152	Yemen	3.355	0.442	1.073
100	27	Panama	6.430	1.112	1.438
101	111	Tunisia	4.592	0.900	0.906
102	25	Chile	6.476	1.131	1.331
103	135	Uganda	4.161	0.322	1.090
104	142	Angola	3.795	0.730	1.125
105	131	Chad	4.301	0.358	0.907
106	35	Malaysia	6.322	1.161	1.258
107	105	South Africa	4.724	0.940	1.410
108	118	Mali	4.447	0.370	1.233
109	124	Kenya	4.410	0.493	1.048
110	29	Argentina	6.388	1.073	1.468
111	62	Bolivia	5.752	0.751	1.223
112	113	Sierra Leone	4.571	0.256	0.813
113	89	Macedonia	5.185	0.959	1.239
114	102	Venezuela	4.806	0.996	1.469
115	103	Gabon	4.758	1.036	1.164
116	132	Congo (Kinshasa)	4.245	0.069	1.136
117	51	Slovenia	5.948	1.219	1.506
118	57	South Korea	5.875	1.266	1.204
119	55	Mauritius	5.891	1.090	1.387
120	116	Sri Lanka	4.471	0.918	1.314
121	143	Madagascar	3.774	0.262	0.908
122	79	Greece	5.358	1.154	1.202
123	78	Serbia	5.398	0.975	1.369
124	99	Cameroon	4.975	0.535	0.891
125	37	Colombia	6.260	0.960	1.439

126	82	Croatia	5.321	1.115	1.161
127	155	Central African Republic	3.083	0.024	0.000
128	68	Turkmenistan	5.636	1.016	1.533
129	145	Afghanistan	3.632	0.332	0.537
130	61	Cyprus	5.762	1.229	1.191
131	92	Kyrgyzstan	5.131	0.530	1.416
132	21	Czech Republic	6.711	1.233	1.489
133	65	Peru	5.663	0.934	1.249
134	88	Tajikistan	5.199	0.474	1.166
135	91	Nigeria	5.155	0.689	1.172
136	94	Mongolia	5.125	0.914	1.517
137	112	Albania	4.586	0.916	0.817
138	56	Jamaica	5.890	0.819	1.493
139	149	Liberia	3.495	0.076	0.858
140	46	Thailand	6.072	1.016	1.417
141	108	Ghana	4.657	0.592	0.896
142	47	Italy	6.000	1.264	1.501
143	129	Armenia	4.321	0.816	0.990
144	59	Russia	5.810	1.151	1.479
145	66	Kosovo	5.662	0.855	1.230
146	69	Hungary	5.620	1.171	1.401
147	38	Trinidad & Tobago	6.192	1.223	1.492
148	96	Indonesia	5.093	0.899	1.215
149	77	Portugal	5.410	1.188	1.429
150	39	Slovakia	6.173	1.210	1.537
151	138	Ukraine	4.103	0.793	1.413
152	100	Bulgaria	4.933	1.054	1.515
153	50	Lithuania	5.952	1.197	1.527
154	52	Romania	5.945	1.116	1.219
155	67	Moldova	5.640	0.657	1.301
156	93	Bosnia and Herzegovina	5.129	0.915	1.078

	Life.expectancy	Freedom	Generosity	Trust	year
1	0.670	0.284	0.186	N/A	2018
2	1.008	0.631	0.261	0.457	2018
3	0.400	0.636	0.200	0.444	2018
4	0.868	0.683	0.284	0.408	2018
5	0.874	0.681	0.202	0.393	2018
6	0.876	0.669	0.365	0.389	2018
7	0.913	0.659	0.285	0.383	2018
8	0.927	0.660	0.256	0.357	2018
9	0.861	0.686	0.286	0.340	2018
10	0.896	0.632	0.196	0.321	2018
11	0.876	0.614	0.307	0.306	2018

12	0.910	0.647	0.361	0.302	2018
13	0.878	0.638	0.333	0.295	2018
14	0.896	0.653	0.321	0.291	2018
15	1.030	0.524	0.246	0.291	2018
16	0.115	0.674	0.238	0.282	2018
17	0.861	0.586	0.273	0.280	2018
18	0.883	0.533	0.354	0.272	2018
19	0.605	0.724	0.328	0.259	2018
20	0.894	0.583	0.188	0.240	2018
21	0.891	0.617	0.242	0.224	2018
22	0.643	0.375	0.038	0.215	2018
23	0.429	0.580	0.598	0.178	2018
24	0.908	0.520	0.098	0.176	2018
25	0.603	0.430	0.031	0.176	2018
26	0.737	0.553	0.086	0.174	2018
27	0.441	0.626	0.230	0.174	2018
28	0.748	0.654	0.256	0.171	2018
29	0.527	0.541	0.364	0.171	2018
30	0.173	0.531	0.206	0.158	2018
31	0.771	0.625	0.130	0.155	2018
32	0.909	0.495	0.179	0.154	2018
33	0.700	0.307	0.101	0.154	2018
34	0.988	0.553	0.079	0.150	2018
35	0.553	0.496	0.116	0.148	2018
36	0.391	0.452	0.220	0.146	2018
37	0.579	0.580	0.153	0.144	2018
38	0.539	0.088	0.376	0.144	2018
39	0.884	0.645	0.376	0.142	2018
40	0.914	0.677	0.353	0.138	2018
41	0.785	0.503	0.214	0.136	2018
42	0.687	0.077	0.055	0.135	2018
43	0.645	0.468	0.130	0.134	2018
44	0.819	0.547	0.291	0.133	2018
45	0.691	0.459	0.282	0.129	2018
46	0.700	0.527	0.208	0.128	2018
47	0.254	0.312	0.175	0.128	2018
48	0.633	0.509	0.098	0.127	2018
49	0.698	0.594	0.243	0.123	2018
50	0.631	0.454	0.148	0.121	2018
51	0.736	0.556	0.114	0.120	2018
52	0.424	0.334	0.216	0.113	2018
53	0.079	0.423	0.116	0.112	2018
54	0.686	0.324	0.106	0.109	2018

55	0.553	0.312	0.092	0.107	2018
56	0.675	0.554	0.167	0.106	2018
57	0.614	0.578	0.120	0.106	2018
58	0.177	0.112	0.224	0.106	2018
59	0.513	0.643	0.120	0.105	2018
60	0.289	0.025	0.392	0.104	2018
61	0.799	0.597	0.029	0.103	2018
62	0.080	0.467	0.146	0.103	2018
63	0.817	0.632	0.143	0.101	2018
64	0.253	0.434	0.158	0.101	2018
65	0.221	0.390	0.175	0.099	2018
66	0.248	0.406	0.132	0.099	2018
67	0.381	0.481	0.270	0.097	2018
68	0.761	0.479	0.069	0.095	2018
69	0.536	0.304	0.148	0.095	2018
70	0.211	0.394	0.185	0.094	2018
71	0.292	0.129	0.134	0.093	2018
72	0.485	0.539	0.172	0.093	2018
73	0.417	0.557	0.042	0.092	2018
74	0.474	0.593	0.183	0.089	2018
75	0.675	0.493	0.110	0.088	2018
76	0.888	0.464	0.262	0.082	2018
77	0.639	0.461	0.065	0.082	2018
78	0.433	0.406	0.138	0.082	2018
79	0.295	0.503	0.221	0.082	2018
80	0.312	0.016	0.134	0.082	2018
81	0.729	0.259	0.111	0.081	2018
82	0.306	0.531	0.210	0.080	2018
83	0.702	0.618	0.177	0.079	2018
84	0.857	0.418	0.151	0.078	2018
85	0.539	0.526	0.302	0.078	2018
86	0.343	0.514	0.091	0.077	2018
87	0.602	0.266	0.086	0.076	2018
88	0.145	0.065	0.149	0.076	2018
89	0.965	0.449	0.142	0.074	2018
90	0.615	0.541	0.162	0.074	2018
91	0.622	0.459	0.197	0.074	2018
92	0.669	0.460	0.026	0.074	2018
93	0.608	0.604	0.179	0.071	2018
94	0.240	0.440	0.163	0.067	2018
95	0.671	0.363	0.092	0.066	2018
96	0.457	0.696	0.256	0.065	2018
97	0.781	0.546	0.108	0.064	2018

98	0.365	0.519	0.051	0.064	2018
99	0.343	0.244	0.083	0.064	2018
100	0.759	0.597	0.125	0.063	2018
101	0.690	0.271	0.040	0.063	2018
102	0.808	0.431	0.197	0.061	2018
103	0.237	0.450	0.259	0.061	2018
104	0.269	0.000	0.079	0.061	2018
105	0.053	0.189	0.181	0.060	2018
106	0.669	0.356	0.311	0.059	2018
107	0.330	0.516	0.103	0.056	2018
108	0.152	0.367	0.139	0.056	2018
109	0.454	0.504	0.352	0.055	2018
110	0.744	0.570	0.062	0.054	2018
111	0.508	0.606	0.141	0.054	2018
112	0.000	0.355	0.238	0.053	2018
113	0.691	0.394	0.173	0.052	2018
114	0.657	0.133	0.056	0.052	2018
115	0.404	0.356	0.032	0.052	2018
116	0.204	0.312	0.197	0.052	2018
117	0.856	0.633	0.160	0.051	2018
118	0.955	0.244	0.175	0.051	2018
119	0.684	0.584	0.245	0.050	2018
120	0.672	0.585	0.307	0.050	2018
121	0.402	0.221	0.155	0.049	2018
122	0.879	0.131	0.000	0.044	2018
123	0.685	0.288	0.134	0.043	2018
124	0.182	0.454	0.183	0.043	2018
125	0.635	0.531	0.099	0.039	2018
126	0.737	0.380	0.120	0.039	2018
127	0.010	0.305	0.218	0.038	2018
128	0.517	0.417	0.199	0.037	2018
129	0.255	0.085	0.191	0.036	2018
130	0.909	0.423	0.202	0.035	2018
131	0.594	0.540	0.281	0.035	2018
132	0.854	0.543	0.064	0.034	2018
133	0.674	0.530	0.092	0.034	2018
134	0.598	0.292	0.187	0.034	2018
135	0.048	0.462	0.201	0.032	2018
136	0.575	0.395	0.253	0.032	2018
137	0.790	0.419	0.149	0.032	2018
138	0.693	0.575	0.096	0.031	2018
139	0.267	0.419	0.206	0.030	2018
140	0.707	0.637	0.364	0.029	2018

```
141           0.337   0.499      0.212 0.029 2018
142           0.946   0.281      0.137 0.028 2018
143           0.666   0.260      0.077 0.028 2018
144           0.599   0.399      0.065 0.025 2018
145           0.578   0.448      0.274 0.023 2018
146           0.732   0.259      0.061 0.022 2018
147           0.564   0.575      0.171 0.019 2018
148           0.522   0.538      0.484 0.018 2018
149           0.884   0.562      0.055 0.017 2018
150           0.776   0.354      0.118 0.014 2018
151           0.609   0.163      0.187 0.011 2018
152           0.712   0.359      0.064 0.009 2018
153           0.716   0.350      0.026 0.006 2018
154           0.726   0.528      0.088 0.001 2018
155           0.620   0.232      0.171 0.000 2018
156           0.758   0.280      0.216 0.000 2018
```

```
# we have seen that the 20th observation value is coded as N/A. We need to first change this

happiness2018$Trust[20]
```

```
[1] "N/A"
```

```
happiness2018$Trust[20] = NA

# let us check if all went well.
happiness2018 |> arrange(Trust)
```

```
    Rank                  Country Score GDP.per.capita Social.support
1     67                  Moldova 5.640          0.657          1.301
2     93   Bosnia and Herzegovina 5.129          0.915          1.078
3     52                  Romania 5.945          1.116          1.219
4     50                Lithuania 5.952          1.197          1.527
5    100                 Bulgaria 4.933          1.054          1.515
6    138                  Ukraine 4.103          0.793          1.413
7     39                 Slovakia 6.173          1.210          1.537
8     77                 Portugal 5.410          1.188          1.429
9     96                Indonesia 5.093          0.899          1.215
10    38        Trinidad & Tobago 6.192          1.223          1.492
11    69                  Hungary 5.620          1.171          1.401
12    66                   Kosovo 5.662          0.855          1.230
```

13	59	Russia	5.810	1.151	1.479
14	47	Italy	6.000	1.264	1.501
15	129	Armenia	4.321	0.816	0.990
16	46	Thailand	6.072	1.016	1.417
17	108	Ghana	4.657	0.592	0.896
18	149	Liberia	3.495	0.076	0.858
19	56	Jamaica	5.890	0.819	1.493
20	91	Nigeria	5.155	0.689	1.172
21	94	Mongolia	5.125	0.914	1.517
22	112	Albania	4.586	0.916	0.817
23	21	Czech Republic	6.711	1.233	1.489
24	65	Peru	5.663	0.934	1.249
25	88	Tajikistan	5.199	0.474	1.166
26	61	Cyprus	5.762	1.229	1.191
27	92	Kyrgyzstan	5.131	0.530	1.416
28	145	Afghanistan	3.632	0.332	0.537
29	68	Turkmenistan	5.636	1.016	1.533
30	155	Central African Republic	3.083	0.024	0.000
31	37	Colombia	6.260	0.960	1.439
32	82	Croatia	5.321	1.115	1.161
33	78	Serbia	5.398	0.975	1.369
34	99	Cameroon	4.975	0.535	0.891
35	79	Greece	5.358	1.154	1.202
36	143	Madagascar	3.774	0.262	0.908
37	55	Mauritius	5.891	1.090	1.387
38	116	Sri Lanka	4.471	0.918	1.314
39	51	Slovenia	5.948	1.219	1.506
40	57	South Korea	5.875	1.266	1.204
41	89	Macedonia	5.185	0.959	1.239
42	102	Venezuela	4.806	0.996	1.469
43	103	Gabon	4.758	1.036	1.164
44	132	Congo (Kinshasa)	4.245	0.069	1.136
45	113	Sierra Leone	4.571	0.256	0.813
46	29	Argentina	6.388	1.073	1.468
47	62	Bolivia	5.752	0.751	1.223
48	124	Kenya	4.410	0.493	1.048
49	105	South Africa	4.724	0.940	1.410
50	118	Mali	4.447	0.370	1.233
51	35	Malaysia	6.322	1.161	1.258
52	131	Chad	4.301	0.358	0.907
53	25	Chile	6.476	1.131	1.331
54	135	Uganda	4.161	0.322	1.090
55	142	Angola	3.795	0.730	1.125

56	27	Panama	6.430	1.112	1.438
57	111	Tunisia	4.592	0.900	0.906
58	42	Poland	6.123	1.176	1.448
59	119	Namibia	4.441	0.874	1.281
60	152	Yemen	3.355	0.442	1.073
61	120	Cambodia	4.433	0.549	1.088
62	53	Latvia	5.933	1.148	1.454
63	136	Benin	4.141	0.378	0.372
64	30	Guatemala	6.382	0.781	1.268
65	36	Spain	6.310	1.251	1.538
66	64	Paraguay	5.681	0.835	1.522
67	72	Honduras	5.504	0.620	1.205
68	85	Morocco	5.254	0.779	0.797
69	104	Palestinian Territories	4.743	0.642	1.217
70	156	Burundi	2.905	0.091	0.627
71	114	Congo (Brazzaville)	4.559	0.682	0.811
72	26	Taiwan	6.441	1.365	1.436
73	101	Nepal	4.880	0.425	1.228
74	95	Vietnam	5.103	0.715	1.365
75	147	Malawi	3.587	0.186	0.541
76	81	Montenegro	5.347	1.017	1.279
77	11	United Kingdom	7.190	1.244	1.433
78	40	El Salvador	6.167	0.806	1.231
79	109	Senegal	4.631	0.429	1.117
80	125	Zambia	4.377	0.562	1.047
81	137	Sudan	4.139	0.605	1.240
82	28	Brazil	6.419	0.986	1.474
83	49	Belize	5.956	0.807	1.101
84	146	Botswana	3.590	1.017	1.174
85	126	Mauritania	4.356	0.557	1.245
86	133	India	4.190	0.721	0.747
87	140	Guinea	3.964	0.344	0.792
88	24	Mexico	6.488	1.038	1.252
89	117	Iraq	4.456	1.010	0.971
90	153	Tanzania	3.303	0.455	0.991
91	134	Niger	4.166	0.131	0.867
92	144	Zimbabwe	3.692	0.357	1.094
93	13	Costa Rica	7.072	1.010	1.459
94	139	Togo	3.999	0.259	0.474
95	86	China	5.246	0.989	1.142
96	107	Ivory Coast	4.671	0.541	0.872
97	148	Haiti	3.582	0.315	0.714
98	71	Philippines	5.524	0.775	1.312

99	45	Kuwait	6.083	1.474	1.301
100	83	Dominican Republic	5.302	0.982	1.441
101	154	South Sudan	3.254	0.337	0.608
102	122	Egypt	4.419	0.885	1.025
103	74	Turkey	5.483	1.148	1.380
104	141	Lesotho	3.808	0.472	1.215
105	75	Pakistan	5.472	0.652	0.810
106	48	Ecuador	5.973	0.889	1.330
107	60	Kazakhstan	5.790	1.143	1.516
108	43	Bahrain	6.105	1.338	1.366
109	33	Saudi Arabia	6.371	1.379	1.331
110	41	Nicaragua	6.141	0.668	1.319
111	121	Burkina Faso	4.424	0.314	1.097
112	106	Iran	4.707	1.059	0.771
113	18	United States	6.886	1.398	1.471
114	90	Jordan	5.161	0.822	1.265
115	84	Algeria	5.295	0.979	1.154
116	80	Lebanon	5.358	0.965	1.179
117	4	Iceland	7.495	1.343	1.644
118	22	Malta	6.627	1.270	1.525
119	115	Bangladesh	4.500	0.532	0.850
120	150	Syria	3.462	0.689	0.382
121	127	Ethiopia	4.350	0.308	0.950
122	70	Libya	5.566	0.985	1.350
123	54	Japan	5.915	1.294	1.462
124	58	Northern Cyprus	5.835	1.229	1.211
125	73	Belarus	5.483	1.039	1.498
126	31	Uruguay	6.379	1.093	1.459
127	123	Mozambique	4.417	0.198	0.902
128	32	Qatar	6.374	1.649	1.303
129	97	Bhutan	5.082	0.796	1.335
130	63	Estonia	5.739	1.200	1.532
131	110	Laos	4.623	0.720	1.034
132	23	France	6.489	1.293	1.466
133	87	Azerbaijan	5.201	1.024	1.161
134	130	Myanmar	4.308	0.682	1.174
135	128	Georgia	4.340	0.853	0.592
136	12	Austria	7.139	1.341	1.504
137	16	Belgium	6.927	1.324	1.483
138	44	Uzbekistan	6.096	0.719	1.584
139	19	Israel	6.814	1.301	1.559
140	15	Germany	6.965	1.340	1.474
141	98	Somalia	4.982	0.000	0.712

```
142    7                  Canada 7.328          1.330          1.532
143   76               Hong Kong 5.430          1.405          1.290
144    6             Netherlands 7.441          1.361          1.488
145   10               Australia 7.272          1.340          1.573
146   14                 Ireland 6.977          1.448          1.583
147   17              Luxembourg 6.910          1.576          1.520
148    2                  Norway 7.594          1.456          1.582
149    5             Switzerland 7.487          1.420          1.549
150    9                  Sweden 7.314          1.355          1.501
151    8             New Zealand 7.324          1.268          1.601
152    1                 Finland 7.632          1.305          1.592
153    3                 Denmark 7.555          1.351          1.590
154  151                  Rwanda 3.408          0.332          0.896
155   34               Singapore 6.343          1.529          1.451
156   20    United Arab Emirates 6.774          2.096          0.776
    Life.expectancy Freedom Generosity Trust year
1             0.620   0.232      0.171 0.000 2018
2             0.758   0.280      0.216 0.000 2018
3             0.726   0.528      0.088 0.001 2018
4             0.716   0.350      0.026 0.006 2018
5             0.712   0.359      0.064 0.009 2018
6             0.609   0.163      0.187 0.011 2018
7             0.776   0.354      0.118 0.014 2018
8             0.884   0.562      0.055 0.017 2018
9             0.522   0.538      0.484 0.018 2018
10            0.564   0.575      0.171 0.019 2018
11            0.732   0.259      0.061 0.022 2018
12            0.578   0.448      0.274 0.023 2018
13            0.599   0.399      0.065 0.025 2018
14            0.946   0.281      0.137 0.028 2018
15            0.666   0.260      0.077 0.028 2018
16            0.707   0.637      0.364 0.029 2018
17            0.337   0.499      0.212 0.029 2018
18            0.267   0.419      0.206 0.030 2018
19            0.693   0.575      0.096 0.031 2018
20            0.048   0.462      0.201 0.032 2018
21            0.575   0.395      0.253 0.032 2018
22            0.790   0.419      0.149 0.032 2018
23            0.854   0.543      0.064 0.034 2018
24            0.674   0.530      0.092 0.034 2018
25            0.598   0.292      0.187 0.034 2018
26            0.909   0.423      0.202 0.035 2018
27            0.594   0.540      0.281 0.035 2018
```

28	0.255	0.085	0.191	0.036	2018
29	0.517	0.417	0.199	0.037	2018
30	0.010	0.305	0.218	0.038	2018
31	0.635	0.531	0.099	0.039	2018
32	0.737	0.380	0.120	0.039	2018
33	0.685	0.288	0.134	0.043	2018
34	0.182	0.454	0.183	0.043	2018
35	0.879	0.131	0.000	0.044	2018
36	0.402	0.221	0.155	0.049	2018
37	0.684	0.584	0.245	0.050	2018
38	0.672	0.585	0.307	0.050	2018
39	0.856	0.633	0.160	0.051	2018
40	0.955	0.244	0.175	0.051	2018
41	0.691	0.394	0.173	0.052	2018
42	0.657	0.133	0.056	0.052	2018
43	0.404	0.356	0.032	0.052	2018
44	0.204	0.312	0.197	0.052	2018
45	0.000	0.355	0.238	0.053	2018
46	0.744	0.570	0.062	0.054	2018
47	0.508	0.606	0.141	0.054	2018
48	0.454	0.504	0.352	0.055	2018
49	0.330	0.516	0.103	0.056	2018
50	0.152	0.367	0.139	0.056	2018
51	0.669	0.356	0.311	0.059	2018
52	0.053	0.189	0.181	0.060	2018
53	0.808	0.431	0.197	0.061	2018
54	0.237	0.450	0.259	0.061	2018
55	0.269	0.000	0.079	0.061	2018
56	0.759	0.597	0.125	0.063	2018
57	0.690	0.271	0.040	0.063	2018
58	0.781	0.546	0.108	0.064	2018
59	0.365	0.519	0.051	0.064	2018
60	0.343	0.244	0.083	0.064	2018
61	0.457	0.696	0.256	0.065	2018
62	0.671	0.363	0.092	0.066	2018
63	0.240	0.440	0.163	0.067	2018
64	0.608	0.604	0.179	0.071	2018
65	0.965	0.449	0.142	0.074	2018
66	0.615	0.541	0.162	0.074	2018
67	0.622	0.459	0.197	0.074	2018
68	0.669	0.460	0.026	0.074	2018
69	0.602	0.266	0.086	0.076	2018
70	0.145	0.065	0.149	0.076	2018

71	0.343	0.514	0.091	0.077	2018
72	0.857	0.418	0.151	0.078	2018
73	0.539	0.526	0.302	0.078	2018
74	0.702	0.618	0.177	0.079	2018
75	0.306	0.531	0.210	0.080	2018
76	0.729	0.259	0.111	0.081	2018
77	0.888	0.464	0.262	0.082	2018
78	0.639	0.461	0.065	0.082	2018
79	0.433	0.406	0.138	0.082	2018
80	0.295	0.503	0.221	0.082	2018
81	0.312	0.016	0.134	0.082	2018
82	0.675	0.493	0.110	0.088	2018
83	0.474	0.593	0.183	0.089	2018
84	0.417	0.557	0.042	0.092	2018
85	0.292	0.129	0.134	0.093	2018
86	0.485	0.539	0.172	0.093	2018
87	0.211	0.394	0.185	0.094	2018
88	0.761	0.479	0.069	0.095	2018
89	0.536	0.304	0.148	0.095	2018
90	0.381	0.481	0.270	0.097	2018
91	0.221	0.390	0.175	0.099	2018
92	0.248	0.406	0.132	0.099	2018
93	0.817	0.632	0.143	0.101	2018
94	0.253	0.434	0.158	0.101	2018
95	0.799	0.597	0.029	0.103	2018
96	0.080	0.467	0.146	0.103	2018
97	0.289	0.025	0.392	0.104	2018
98	0.513	0.643	0.120	0.105	2018
99	0.675	0.554	0.167	0.106	2018
100	0.614	0.578	0.120	0.106	2018
101	0.177	0.112	0.224	0.106	2018
102	0.553	0.312	0.092	0.107	2018
103	0.686	0.324	0.106	0.109	2018
104	0.079	0.423	0.116	0.112	2018
105	0.424	0.334	0.216	0.113	2018
106	0.736	0.556	0.114	0.120	2018
107	0.631	0.454	0.148	0.121	2018
108	0.698	0.594	0.243	0.123	2018
109	0.633	0.509	0.098	0.127	2018
110	0.700	0.527	0.208	0.128	2018
111	0.254	0.312	0.175	0.128	2018
112	0.691	0.459	0.282	0.129	2018
113	0.819	0.547	0.291	0.133	2018

114	0.645	0.468	0.130	0.134	2018
115	0.687	0.077	0.055	0.135	2018
116	0.785	0.503	0.214	0.136	2018
117	0.914	0.677	0.353	0.138	2018
118	0.884	0.645	0.376	0.142	2018
119	0.579	0.580	0.153	0.144	2018
120	0.539	0.088	0.376	0.144	2018
121	0.391	0.452	0.220	0.146	2018
122	0.553	0.496	0.116	0.148	2018
123	0.988	0.553	0.079	0.150	2018
124	0.909	0.495	0.179	0.154	2018
125	0.700	0.307	0.101	0.154	2018
126	0.771	0.625	0.130	0.155	2018
127	0.173	0.531	0.206	0.158	2018
128	0.748	0.654	0.256	0.171	2018
129	0.527	0.541	0.364	0.171	2018
130	0.737	0.553	0.086	0.174	2018
131	0.441	0.626	0.230	0.174	2018
132	0.908	0.520	0.098	0.176	2018
133	0.603	0.430	0.031	0.176	2018
134	0.429	0.580	0.598	0.178	2018
135	0.643	0.375	0.038	0.215	2018
136	0.891	0.617	0.242	0.224	2018
137	0.894	0.583	0.188	0.240	2018
138	0.605	0.724	0.328	0.259	2018
139	0.883	0.533	0.354	0.272	2018
140	0.861	0.586	0.273	0.280	2018
141	0.115	0.674	0.238	0.282	2018
142	0.896	0.653	0.321	0.291	2018
143	1.030	0.524	0.246	0.291	2018
144	0.878	0.638	0.333	0.295	2018
145	0.910	0.647	0.361	0.302	2018
146	0.876	0.614	0.307	0.306	2018
147	0.896	0.632	0.196	0.321	2018
148	0.861	0.686	0.286	0.340	2018
149	0.927	0.660	0.256	0.357	2018
150	0.913	0.659	0.285	0.383	2018
151	0.876	0.669	0.365	0.389	2018
152	0.874	0.681	0.202	0.393	2018
153	0.868	0.683	0.284	0.408	2018
154	0.400	0.636	0.200	0.444	2018
155	1.008	0.631	0.261	0.457	2018
156	0.670	0.284	0.186	<NA>	2018

```r
# however the variable is still a factor so we need to convert this into numeric.
happiness2018 = happiness2018 |> mutate(Trust = as.numeric(Trust))

# when we check again we see that all fine.
str(happiness2018)
```

```
'data.frame':   156 obs. of  10 variables:
 $ Rank           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Country        : chr  "Finland" "Norway" "Denmark" "Iceland" ...
 $ Score          : num  7.63 7.59 7.55 7.5 7.49 ...
 $ GDP.per.capita : num  1.3 1.46 1.35 1.34 1.42 ...
 $ Social.support : num  1.59 1.58 1.59 1.64 1.55 ...
 $ Life.expectancy: num  0.874 0.861 0.868 0.914 0.927 0.878 0.896 0.876 0.913 0.91 ...
 $ Freedom        : num  0.681 0.686 0.683 0.677 0.66 0.638 0.653 0.669 0.659 0.647 ...
 $ Generosity     : num  0.202 0.286 0.284 0.353 0.256 0.333 0.321 0.365 0.285 0.361 ...
 $ Trust          : num  0.393 0.34 0.408 0.138 0.357 0.295 0.291 0.389 0.383 0.302 ...
 $ year           : num  2018 2018 2018 2018 2018 ...
```

After solving the issues, we try the bind again:

```r
happiness15_19 <- bind_rows(happiness2015, happiness2016, happiness2017, happiness2018, happ
str(happiness15_19)
```

```
'data.frame':   782 obs. of  12 variables:
 $ Country        : chr  "Switzerland" "Iceland" "Denmark" "Norway" ...
 $ Region         : chr  "Western Europe" "Western Europe" "Western Europe" "Western Europe"
 $ Rank           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Score          : num  7.59 7.56 7.53 7.52 7.43 ...
 $ GDP.per.capita : num  1.4 1.3 1.33 1.46 1.33 ...
 $ Family         : num  1.35 1.4 1.36 1.33 1.32 ...
 $ Life.expectancy: num  0.941 0.948 0.875 0.885 0.906 ...
 $ Freedom        : num  0.666 0.629 0.649 0.67 0.633 ...
 $ Trust          : num  0.42 0.141 0.484 0.365 0.33 ...
 $ Generosity     : num  0.297 0.436 0.341 0.347 0.458 ...
 $ year           : num  2015 2015 2015 2015 2015 ...
 $ Social.support : num  NA NA NA NA NA NA NA NA NA NA ...
```

Now we can look at specific countries or years using filter() function. Make plots, summaries etc.

Question: How would you create a table for Country, Happiness Score and GDP per capita for only two countries (Afghanistan and Switzerland) for only 2016?

```r
happiness15_19 |>
  filter(Country %in% c("Afghanistan", "Switzerland"), year == 2016) |>
    dplyr::select(Country, GDP.per.capita, Score)
```

```
      Country GDP.per.capita Score
1 Switzerland        1.52733 7.509
2 Afghanistan        0.38227 3.360
```

Column bind

I would not use this if I were you. Instead use join functions.

`full_join` function

```r
h = full_join(happiness2015, happiness2016, by = "Country")
h = full_join(h, happiness2017, by = "Country")

h = happiness2015 |> full_join(happiness2016, by= "Country") |>
  full_join(happiness2017, by= "Country") |>
  full_join(happiness2018, by= "Country") |>
  full_join(happiness2019, by= "Country")

# h |>
#   tidyr::pivot_longer(
#     cols = everything(),
#     names_to = c(".value", "set"),
#     names_pattern = "(.)(.)"
#   )
# melt(h, id.vars = "Country")
```

The `match()` function

In some cases, you might have several variables, too many maybe, and you may only need to focus on a subset of these variables.

The `select()` function can be used to select columns of a data frame that you want to focus on or similarly omitting columns you don't need.

```r
happiness2019 |>
  dplyr::select(1) |>
  head(2)
```

```
  Rank
1    1
2    2
```

```
happiness2019 |>
  dplyr::select(1:3) |>
  head(2)
```

```
  Rank Country Score
1    1 Finland 7.769
2    2 Denmark 7.600
```

```
happiness2019 |>
  dplyr::select(c(1,3,5)) |>
  head(2)
```

```
  Rank Score Social.support
1    1 7.769          1.587
2    2 7.600          1.573
```

```
happiness2019 |>
  dplyr::select(-1) |>
  head(2)
```

```
  Country Score GDP.per.capita Social.support Life.expectancy Freedom
1 Finland 7.769          1.340          1.587           0.986   0.596
2 Denmark 7.600          1.383          1.573           0.996   0.592
  Generosity Trust year
1      0.153 0.393 2019
2      0.252 0.410 2019
```

You can also select columns based on specific criteria with:

starts_with() = Select columns that start with a character string ends_with() = Select columns that end with a character string contains() = Select columns that contain a character string matches() = Select columns that match a regular expression one_of() = Select columns names that are from a group of names or there is the grepl function!

```
happiness2019 |>
  dplyr::select(ends_with("m")) |>
  str()
```

```
'data.frame':   156 obs. of  1 variable:
 $ Freedom: num  0.596 0.592 0.603 0.591 0.557 0.572 0.574 0.585 0.584 0.532 ...
```

```
happiness2019 |>
  dplyr::select(starts_with("G")) |>
  str()
```

```
'data.frame':   156 obs. of  2 variables:
 $ GDP.per.capita: num  1.34 1.38 1.49 1.38 1.4 ...
 $ Generosity    : num  0.153 0.252 0.271 0.354 0.322 0.263 0.267 0.33 0.285 0.244 ...
```

The `filter()` function

Suppose we wanted to extract the rows of the happiness data frame where the levels of happiness (Score) are greater than 7, we could do:

```
str(happiness2019)
```

```
'data.frame':   156 obs. of  10 variables:
 $ Rank           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Country        : chr  "Finland" "Denmark" "Norway" "Iceland" ...
 $ Score          : num  7.77 7.6 7.55 7.49 7.49 ...
 $ GDP.per.capita : num  1.34 1.38 1.49 1.38 1.4 ...
 $ Social.support : num  1.59 1.57 1.58 1.62 1.52 ...
 $ Life.expectancy: num  0.986 0.996 1.028 1.026 0.999 ...
 $ Freedom        : num  0.596 0.592 0.603 0.591 0.557 0.572 0.574 0.585 0.584 0.532 ...
 $ Generosity     : num  0.153 0.252 0.271 0.354 0.322 0.263 0.267 0.33 0.285 0.244 ...
 $ Trust          : num  0.393 0.41 0.341 0.118 0.298 0.343 0.373 0.38 0.308 0.226 ...
 $ year           : num  2019 2019 2019 2019 2019 ...
```

```
happiness2019 |>
  filter(Score>7) |>
  head()
```

```
  Rank     Country Score GDP.per.capita Social.support Life.expectancy Freedom
1    1     Finland 7.769          1.340          1.587           0.986   0.596
2    2     Denmark 7.600          1.383          1.573           0.996   0.592
3    3      Norway 7.554          1.488          1.582           1.028   0.603
4    4     Iceland 7.494          1.380          1.624           1.026   0.591
5    5 Netherlands 7.488          1.396          1.522           0.999   0.557
```

```
6    6 Switzerland 7.480          1.452          1.526           1.052   0.572
  Generosity Trust year
1      0.153 0.393 2019
2      0.252 0.410 2019
3      0.271 0.341 2019
4      0.354 0.118 2019
5      0.322 0.298 2019
6      0.263 0.343 2019
```

We can place an arbitrarily complex logical sequence inside of filter(), say filtering observations where Score is greater than 7 and GDP.per.capita >1.5:

```
happiness2019 |>
  filter(Score > 7 & GDP.per.capita > 1.5) |>
  head()
```

```
  Rank    Country Score GDP.per.capita Social.support Life.expectancy Freedom
1   14 Luxembourg  7.09          1.609          1.479           1.012   0.526
  Generosity Trust year
1      0.194 0.316 2019
```

The `arrange()` function

Here we can order the rows of the data frame by GDP.per.capita, so that the first row is the least developed observation and the last row is the most developed observation. Currently the data is ordered according to Happiness Score.

```
# developed countries
happiness2019 |>
  arrange(GDP.per.capita) |>
  head()
```

```
  Rank                  Country Score GDP.per.capita Social.support
1  112                  Somalia 4.668          0.000          0.698
2  155 Central African Republic 3.083          0.026          0.000
3  145                  Burundi 3.775          0.046          0.447
4  141                  Liberia 3.975          0.073          0.922
5  127         Congo (Kinshasa) 4.418          0.094          1.125
6  114                    Niger 4.628          0.138          0.774
  Life.expectancy Freedom Generosity Trust year
1           0.268   0.559      0.243 0.270 2019
```

```
2           0.105   0.225      0.235 0.035 2019
3           0.380   0.220      0.176 0.180 2019
4           0.443   0.370      0.233 0.033 2019
5           0.357   0.269      0.212 0.053 2019
6           0.366   0.318      0.188 0.102 2019
```

```
# tail()
```

The `rename()` function

Renaming variable names can be difficult in R. For example, you may want to rename GDP.per.capita as gdppercapita, all in small letters and no dots in between.

```
happiness2019 |>
  rename(gdppercapita = GDP.per.capita) |>
  head()
```

```
  Rank     Country Score gdppercapita Social.support Life.expectancy Freedom
1    1     Finland 7.769        1.340          1.587           0.986   0.596
2    2     Denmark 7.600        1.383          1.573           0.996   0.592
3    3      Norway 7.554        1.488          1.582           1.028   0.603
4    4     Iceland 7.494        1.380          1.624           1.026   0.591
5    5 Netherlands 7.488        1.396          1.522           0.999   0.557
6    6 Switzerland 7.480        1.452          1.526           1.052   0.572
  Generosity Trust year
1      0.153 0.393 2019
2      0.252 0.410 2019
3      0.271 0.341 2019
4      0.354 0.118 2019
5      0.322 0.298 2019
6      0.263 0.343 2019
```

The `mutate()` function

The `mutate()` function exists to compute transformations of variables in a data frame. Often, you want to create new variables that are derived from existing variables and `mutate()` provides a clean interface for doing that.

Note: `mutate()` is order aware. So you can chain multiple mutates in a single call.

For example, you may want to standardize GDP.per.capita or apply a log transformation, or create a column for years:

```
happiness2019 = happiness2019 |>
  mutate(loggdppercapita = log10(GDP.per.capita)) |>
  head()

happiness2019 = happiness2019 |>
  mutate(year = 2019)|>
  head()
```

```
# Boolean, logical and conditional operators all work well with mutate() too.

happiness2019 <- happiness2019 |>
  mutate(happy = ifelse(Score > 5.5, "happy", "nothappy")) |>
  head(2)
```

The `Pipeline` Operator

The pipeline operator strings together multiple dplyr functions in a sequence.

```
> third(second(first(x)))
```

Instead of creating difficult to read and manage functions, we will use the pipeline:

```
> first(x) |> second |> third
```

Let us assume we wanted to select gdppercapita column, then filter those countries with less than 1.5, and log transform the values:

```
select(happiness2019, GDP.per.capita) |>
  mutate(loggdp = log10(GDP.per.capita)) |>
    filter(GDP.per.capita>1.5)
```

```
[1] GDP.per.capita loggdp
<0 rows> (or 0-length row.names)
```

The `group_by()` function

The `group_by()` function is used to generate summary statistics from the data frame within strata defined by a variable.

```
happiness2019 |>
  group_by(happy) |>
  summarise(mean_gdp = mean(GDP.per.capita, na.rm = T))
```

```
# A tibble: 1 x 2
  happy mean_gdp
  <chr>    <dbl>
1 happy     1.36
```

```
# count() and distinct(): Number and isolate unique observations.
happiness2019 |> count(happy)
```

```
  happy n
1 happy 2
```

```
happiness2019 |> distinct(happy)
```

```
  happy
1 happy
```

Family of join operations

Sometimes you may collect data from different environments and may need to merge the datasets according a particular variable.

One of the main advantages of dplyr is that you can join data frames easily. There are several functions one can use to merge, or join datasets. But the dplyr is the easiest and fastest.

```
#install.packages("nycflights13")
library(nycflights13)
```

```
Warning: package 'nycflights13' was built under R version 4.4.3
```

```
flights
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
```

```
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
# i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
#   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
planes
```

```
# A tibble: 3,322 x 9
   tailnum  year type              manufacturer model engines seats speed engine
   <chr>   <int> <chr>             <chr>        <chr>   <int> <int> <int> <chr>
 1 N10156   2004 Fixed wing multi~ EMBRAER      EMB-~       2    55    NA Turbo~
 2 N102UW   1998 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 3 N103US   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 4 N104UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 5 N10575   2002 Fixed wing multi~ EMBRAER      EMB-~       2    55    NA Turbo~
 6 N105UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 7 N107US   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 8 N108UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
 9 N109UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
10 N110UW   1999 Fixed wing multi~ AIRBUS INDU~ A320~       2   182    NA Turbo~
# i 3,312 more rows
```

inner_join

This join will take only the matching values based on the specified columns.

```
inner_join(flights, planes) |>
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
```

```
Joining with `by = join_by(year, tailnum)`
```

```
# A tibble: 4,630 x 10
    year month   day dep_time arr_time carrier flight tailnum type         model
   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>   <chr>        <chr>
 1  2013     1    18     1846     2156 UA        1292 N37465  Fixed wing ~ 737-~
 2  2013    10     1      647      744 B6         318 N355JB  Fixed wing ~ ERJ ~
```

```
 3  2013    10     1      652      921 UA        1439 N37471  Fixed wing ~ 737-~
 4  2013    10     1      755      954 9E        3538 N292PQ  Fixed wing ~ CL-6~
 5  2013    10     1      813     1050 B6         281 N354JB  Fixed wing ~ ERJ ~
 6  2013    10     1      925     1025 B6         116 N373JB  Fixed wing ~ ERJ ~
 7  2013    10     1     1113     1215 B6        1307 N374JB  Fixed wing ~ ERJ ~
 8  2013    10     1     1426     1535 B6         286 N368JB  Fixed wing ~ ERJ ~
 9  2013    10     1     1446     1635 US        1995 N156UW  Fixed wing ~ A321~
10  2013    10     1     1454     1751 B6         575 N374JB  Fixed wing ~ ERJ ~
# i 4,620 more rows
```

left_join

What do we mean by left join?

This join will take all of the values from the table we specify as left (e.g., the first one) and match them to records from the table on the right (e.g. the second one). If there isn't a match in the second table, then it will return NULL for the row in question.

```
left_join(flights, planes) |>
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
```

```
Joining with `by = join_by(year, tailnum)`
```

```
# A tibble: 336,776 x 10
    year month   day dep_time arr_time carrier flight tailnum type  model
   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>   <chr> <chr>
 1  2013     1     1      517      830 UA        1545 N14228  <NA>  <NA>
 2  2013     1     1      533      850 UA        1714 N24211  <NA>  <NA>
 3  2013     1     1      542      923 AA        1141 N619AA  <NA>  <NA>
 4  2013     1     1      544     1004 B6         725 N804JB  <NA>  <NA>
 5  2013     1     1      554      812 DL         461 N668DN  <NA>  <NA>
 6  2013     1     1      554      740 UA        1696 N39463  <NA>  <NA>
 7  2013     1     1      555      913 B6         507 N516JB  <NA>  <NA>
 8  2013     1     1      557      709 EV        5708 N829AS  <NA>  <NA>
 9  2013     1     1      557      838 B6          79 N593JB  <NA>  <NA>
10  2013     1     1      558      753 AA         301 N3ALAA  <NA>  <NA>
# i 336,766 more rows
```

If you do not provide the columns to be joined, dplyr will joing the datasets based on the columns that have the same name. At the end of the operator, it will tell us which columns are used to join.

```
Joining, by = c("year", "tailnum")
```

Now, what if the joining columns don't have same name, or the meaning of the same named columns is different, then you need to specify the joining columns with the by = argument:

```
left_join(flights, planes, by = c("tailnum", "year")) |>
  select(year, month, day, dep_time, arr_time, carrier, flight, tailnum, type, model)
```

```
# A tibble: 336,776 x 10
    year month   day dep_time arr_time carrier flight tailnum type  model
   <int> <int> <int>    <int>    <int> <chr>    <int> <chr>   <chr> <chr>
 1  2013     1     1      517      830 UA        1545 N14228  <NA>  <NA>
 2  2013     1     1      533      850 UA        1714 N24211  <NA>  <NA>
 3  2013     1     1      542      923 AA        1141 N619AA  <NA>  <NA>
 4  2013     1     1      544     1004 B6         725 N804JB  <NA>  <NA>
 5  2013     1     1      554      812 DL         461 N668DN  <NA>  <NA>
 6  2013     1     1      554      740 UA        1696 N39463  <NA>  <NA>
 7  2013     1     1      555      913 B6         507 N516JB  <NA>  <NA>
 8  2013     1     1      557      709 EV        5708 N829AS  <NA>  <NA>
 9  2013     1     1      557      838 B6          79 N593JB  <NA>  <NA>
10  2013     1     1      558      753 AA         301 N3ALAA  <NA>  <NA>
# i 336,766 more rows
```

right_join

Basically takes the second data frame's observations into account and the new joint will take all of its values from the second dataframe.

full_join

The full outer join returns all of the records in a new table, whether it matches on either the left or right tables. If the table rows match, then a join will be executed, otherwise it will return NULL in places where a matching row does not exist.

semi_join

A semi join creates a new table where it will return all rows from the first table where there is a corresponding matching value in second, but instead of the new table combining both the first and second tables, it only contains data from the first table.

Some Extras

- A very common `filter()` use case is identifying (or removing) missing data cases.

```
happiness2019 |>
  filter(is.na(Generosity))
```

```
 [1] Rank            Country         Score           GDP.per.capita
 [5] Social.support  Life.expectancy Freedom         Generosity
 [9] Trust           year            loggdppercapita happy
<0 rows> (or 0-length row.names)
```

How would we remove missing observations? Try this yourself.

- Arranging the data based on a variable (either numeric or categorical)

```
happiness2019 |>
  arrange(Generosity)
```

```
  Rank Country Score GDP.per.capita Social.support Life.expectancy Freedom
1    1 Finland 7.769          1.340          1.587           0.986   0.596
2    2 Denmark 7.600          1.383          1.573           0.996   0.592
  Generosity Trust year loggdppercapita happy
1      0.153 0.393 2019       0.1271048 happy
2      0.252 0.410 2019       0.1408222 happy
```

Arranging on a character-based column (i.e. strings) will sort alphabetically. Try this yourself by arranging according to the "Country.or.region" column.

- We can also arrange items in descending order using `arrange(desc())`.

```
happiness2019 |>
  arrange(desc(Generosity))
```

```
  Rank Country Score GDP.per.capita Social.support Life.expectancy Freedom
1    2 Denmark 7.600          1.383          1.573           0.996   0.592
2    1 Finland 7.769          1.340          1.587           0.986   0.596
  Generosity Trust year loggdppercapita happy
1      0.252 0.410 2019       0.1408222 happy
2      0.153 0.393 2019       0.1271048 happy
```

Once you learn the dplyr grammar there are a few additional benefits. `dplyr` can work with other data frame "backends" " such as SQL databases. There is an SQL interface for relational databases via the DBI package dplyr can be integrated with the data.table package for large fast tables

Managing Data Frames with `TIDYR` Package

Key `TIDYR` verbs

- `gather()`: Gather (or "melt") wide data into long format
- `spread()`: Spread (or "cast") long data into wide format.
- `separate()`: Separate (i.e. split) one column into multiple columns.
- `unite()`: Unite (i.e. combine) multiple columns into one.