More on nested data frames

Data Wrangling and Husbandry

03/22/2020

Remember about subsetting lists

- `x[1]` is a list
  - Something list `x[1:4]` is legitimate and gives a (sub-)list of 4 elements
  - Can also be done by name: ‘x[“a”]
- `x[[1]]` is a component of a list and removes the top hierarchical level
  - `x[[1:4]]` is not legitimate
- `x$a` is the same as `x[["a"]]`

split-apply-combine

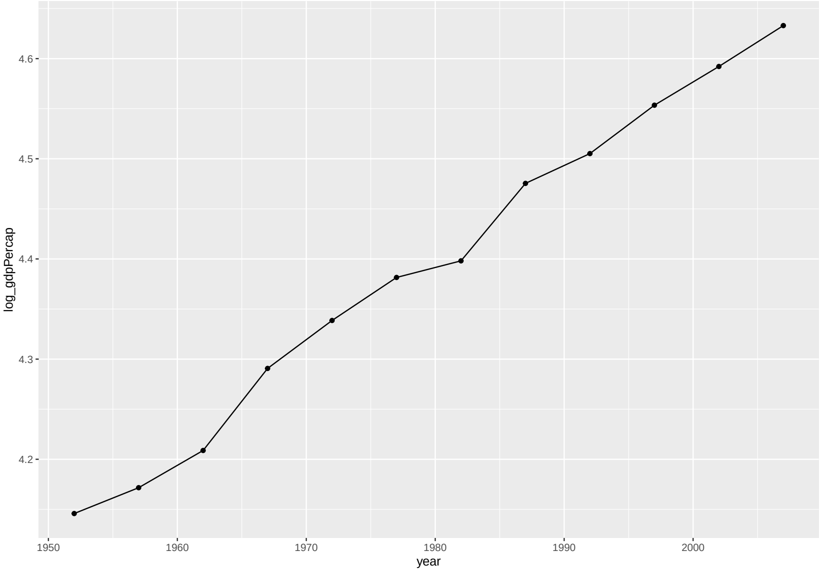
Many tasks in R require the steps of split, apply, and combine

- split data into pieces,
- apply some function to each piece
- combine the results back together

A small example taking advantage of the `modelr` package. We did a similar analysis back in Week 4

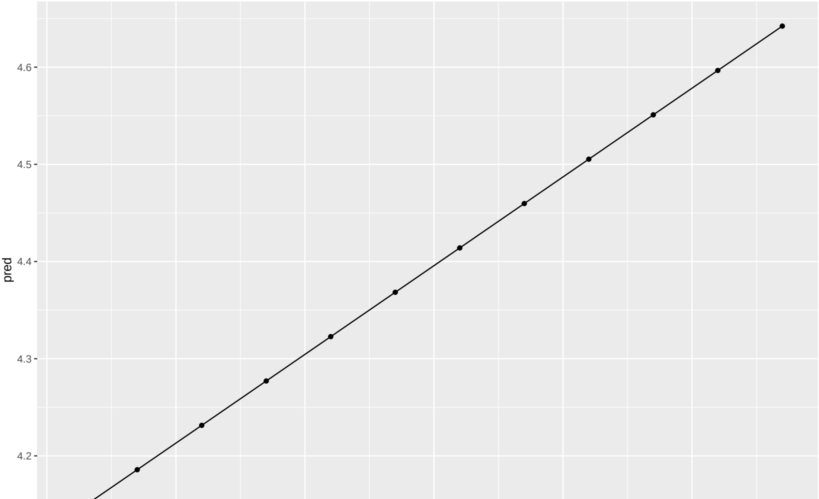
```
library(gapminder)
library(modelr)
gap_us <- gapminder %>% filter(country == "United States")
  mutate(log_gdpPercap = log10(gdpPercap))
gap_us_lm <- lm(log_gdpPercap ~ year, data = gap_us)
```

```r
(gap_us_full <- gap_us %>% ggplot(aes(year, log_gdpPercap))
```

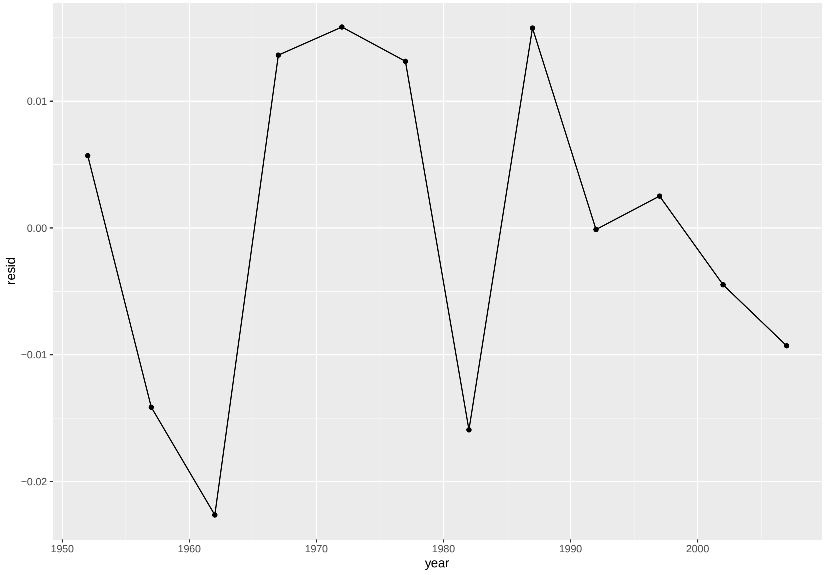


The function add_predictions() is from the modeler package. You can get something similar using broom::augment() as we've done in the past.

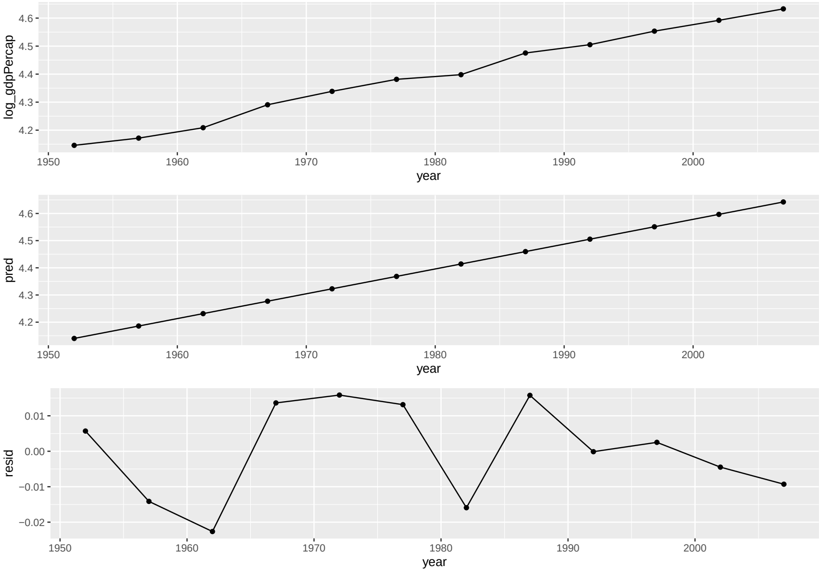
```
(gap_us_trend <- add_predictions(gap_us, gap_us_lm) %>%
   ggplot(aes(year, pred)) + geom_line() + geom_point() )
```



```r
(gap_us_resid <- add_residuals(gap_us, gap_us_lm) %>%
   ggplot(aes(year, resid)) + geom_line() + geom_point() )
```



```
library(gridExtra)
## the package patchwork also works nicely to combine plots
grid.arrange(gap_us_full, gap_us_trend, gap_us_resid, nrow
```



There are several ways we could apply this sort of analysis to each country, but there is a fairly elegant way using the concept of nested data frames. As we've seen, since a data frame is a collection of lists, all of the same length, a column of a data frame could itself contain data frames. This is easily implemented using the `tidyr` package's `nest()` function.

```r
gap_nested <-   gapminder %>%
    group_by(country, continent) %>%
    nest()
gap_nested[1:3, ]
```

```
## # A tibble: 3 x 3
## # Groups:   country, continent [710]
##   country     continent data
##   <fct>       <fct>     <list>
## 1 Afghanistan Asia      <tibble [12 x 4]>
## 2 Albania     Europe    <tibble [12 x 4]>
## 3 Algeria     Africa    <tibble [12 x 4]>
```

```r
gap_nested[1, "data"]
```

```
## # A tibble: 1 x 1
##   data
##   <list>
## 1 <tibble [12 x 4]>
```

```r
gap_nested$data[[1]]
```

```
## # A tibble: 12 x 4
##     year lifeExp      pop gdpPercap
##    <int>   <dbl>    <int>     <dbl>
##  1  1952    28.8  8425333      779.
##  2  1957    30.3  9240934      821.
##  3  1962    32.0 10267083      853.
##  4  1967    34.0 11537966      836.
##  5  1972    36.1 13079460      740.
##  6  1977    38.4 14880372      786.
##  7  1982    39.9 12881816      978.
##  8  1987    40.8 13867957      852.
##  9  1992    41.7 16317921      649.
## 10  1997    41.8 22227415      635.
## 11  2002    42.1 25268405      727.
## 12  2007    43.8 31889923      975.
```

```
gap_nested_lm <- gapminder %>% mutate(log_gdpPercap = log1(
  group_by(continent, country) %>% nest()
gap_nested_lm[1:3, ]
```

```
## # A tibble: 3 x 3
## # Groups:   country, continent [710]
##   country     continent data
##   <fct>       <fct>     <list>
## 1 Afghanistan Asia      <tibble [12 x 5]>
## 2 Albania     Europe    <tibble [12 x 5]>
## 3 Algeria     Africa    <tibble [12 x 5]>
```

```
gap_nested_lm[1, "data"]
```

```
## # A tibble: 1 x 1
##   data
##   <list>
## 1 <tibble [12 x 5]>
```

```r
gap_nested_lm$data[[1]] %>% glimpse()
```

```
## Observations: 12
## Variables: 5
## $ year          <int> 1952, 1957, 1962, 1967, 1972, 197
## $ lifeExp       <dbl> 28.801, 30.332, 31.997, 34.020, 3
## $ pop           <int> 8425333, 9240934, 10267083, 11537
## $ gdpPercap     <dbl> 779.4453, 820.8530, 853.1007, 836
## $ log_gdpPercap <dbl> 2.891786, 2.914265, 2.931000, 2.9
```

We can define a function to fit a regression model per country

```
gap_lm <- function(df){
  lm(log_gdpPercap ~ year, data = df)
}
```

We could then use the `map()` function from the `purrr` library to fit a model per country, since `gap_nested_lm$data` is a list:

```
class(gap_nested_lm$data)
```

```
## [1] "list"
```

```
gap_all_lm <- gap_nested_lm$data %>% map(gap_lm)
```

```r
class(gap_all_lm)
```

```
## [1] "list"
```

```r
length(gap_all_lm)
```

```
## [1] 142
```

```r
gap_all_lm[[1]]
```

```
##
## Call:
## lm(formula = log_gdpPercap ~ year, data = df)
##
## Coefficients:
## (Intercept)         year
##   3.6963846   -0.0004019
```

Instead, however, let's put the model right back into the nested data frame:

```
gap_nested_lm <- gap_nested_lm %>%
  mutate(lm_fit = map(data, gap_lm))

gap_nested_lm
```

```
## # A tibble: 142 x 4
## # Groups:   country, continent [710]
##    country     continent data              lm_fit
##    <fct>       <fct>     <list>            <list>
##  1 Afghanistan Asia      <tibble [12 x 5]> <lm>
##  2 Albania     Europe    <tibble [12 x 5]> <lm>
##  3 Algeria     Africa    <tibble [12 x 5]> <lm>
##  4 Angola      Africa    <tibble [12 x 5]> <lm>
##  5 Argentina   Americas  <tibble [12 x 5]> <lm>
##  6 Australia   Oceania   <tibble [12 x 5]> <lm>
##  7 Austria     Europe    <tibble [12 x 5]> <lm>
##  8 Bahrain     Asia      <tibble [12 x 5]> <lm>
##  9 Bangladesh  Asia      <tibble [12 x 5]> <lm>
```

Because this is a data frame, we can still do things like `filter()` and `arrange()`. Or `mutate()`.

```
library(broom)
gap_nested_lm <- gap_nested_lm %>%
  mutate(lm_glance = map(lm_fit, glance))
```

Consider

```
gap_nested_lm <- gap_nested_lm %>%
  mutate(resid = map2(data, lm_fit, add_residuals))
## Notice map2, not map, because two arguments are indexed

head(gap_nested_lm)
```

```
## # A tibble: 6 x 6
## # Groups:   country, continent [710]
##   country     continent data                lm_fit lm_glanc
##   <fct>       <fct>     <list>              <list> <list>
## 1 Afghanistan Asia      <tibble [12 x 5~ <lm>   <tibble
## 2 Albania     Europe    <tibble [12 x 5~ <lm>   <tibble
## 3 Algeria     Africa    <tibble [12 x 5~ <lm>   <tibble
## 4 Angola      Africa    <tibble [12 x 5~ <lm>   <tibble
## 5 Argentina   Americas  <tibble [12 x 5~ <lm>   <tibble
## 6 Australia   Oceania   <tibble [12 x 5~ <lm>   <tibble
```

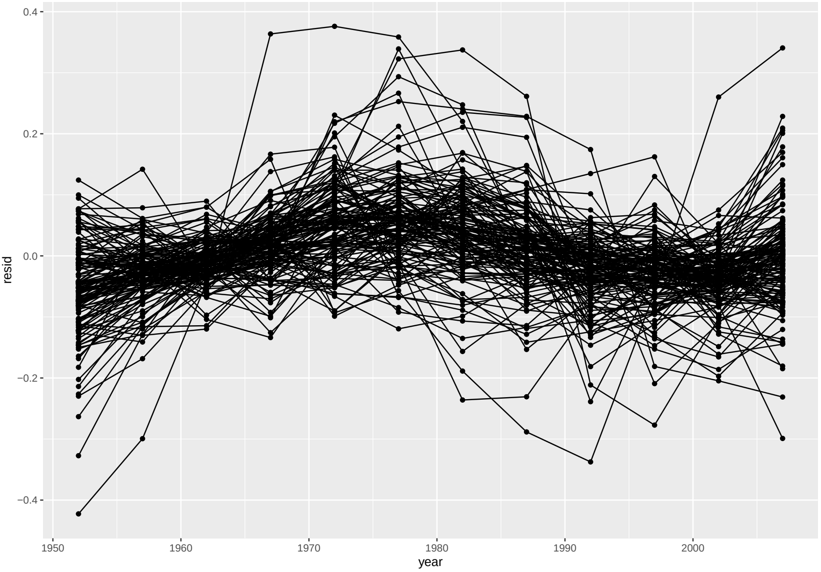
At this point, maybe we would like to plot the residuals, or otherwise use them, going back to the original observations. The function `unnest()` will take care of this for us.

```
gap_unnested <- unnest(gap_nested_lm, resid)
select(gap_unnested, country, year, resid)
```

```
## Adding missing grouping variables: `continent`
```

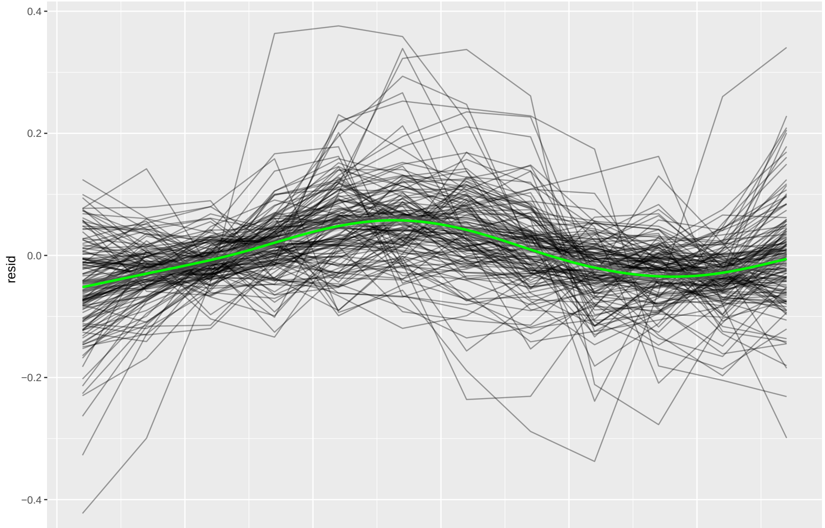
```
## # A tibble: 1,704 x 4
## # Groups:   country, continent [710]
##    continent country      year    resid
##    <fct>     <fct>       <int>    <dbl>
##  1 Asia      Afghanistan  1952 -0.0202
##  2 Asia      Afghanistan  1957  0.00433
##  3 Asia      Afghanistan  1962  0.0231
##  4 Asia      Afghanistan  1967  0.0164
##  5 Asia      Afghanistan  1972 -0.0347
##  6 Asia      Afghanistan  1977 -0.00641
##  7 Asia      Afghanistan  1982  0.0905
##  8 Asia      Afghanistan  1987  0.0328
```

```r
gap_unnested %>% ggplot(aes(year, resid, group = country))
  geom_line() +
  geom_point()
```



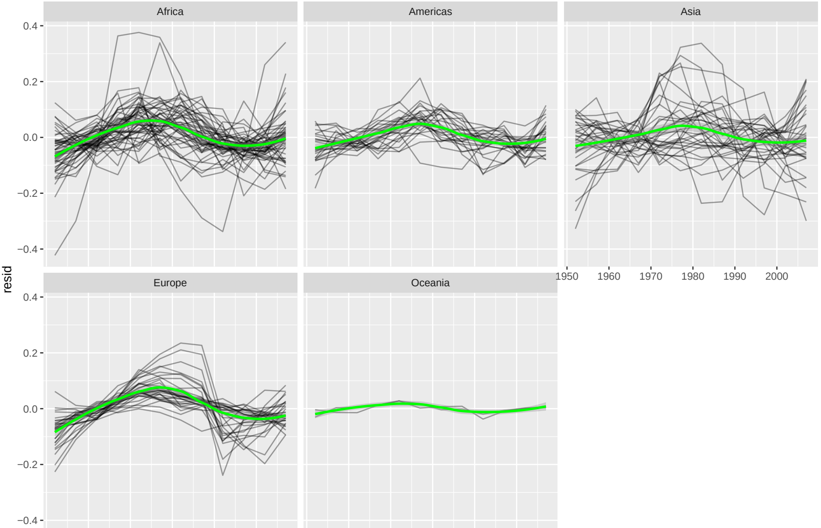
```r
gap_unnested %>% ggplot(aes(year, resid)) +
  geom_line(alpha = 0.375, aes(group = country)) +
  geom_smooth(color = "green")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(
```



```r
gap_unnested %>% ggplot(aes(year, resid)) +
  geom_line(alpha = 0.375, aes(group = country)) +
  geom_smooth(color = "green") + facet_wrap(~ continent)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~
```



An aside on using `walk()` to save lots of files

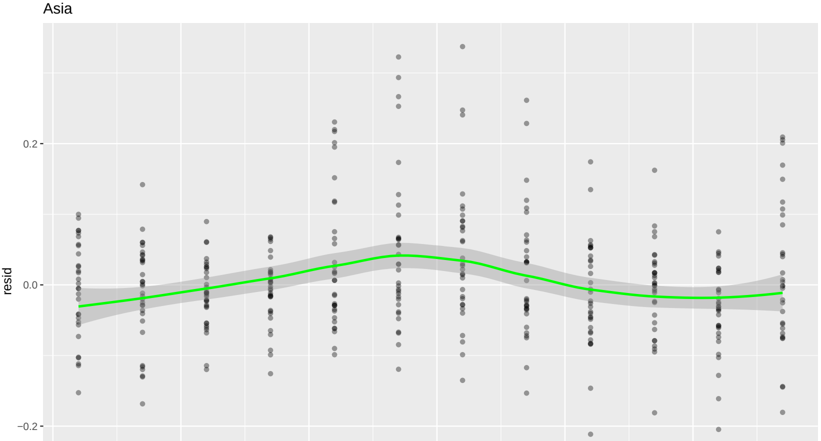
We could make a set of distinct plots of residuals by continent using `map()`:

```
continent_residuals <- map(unique(gap_unnested$continent),
    ~ gap_unnested %>%
      filter(continent == .x) %>%
      ggplot(aes(year, resid)) +
        geom_point(alpha = 0.375) +
        geom_smooth(color = "green"))
```

Notice that this is a list of plots. I'm going to add titles using map2():

```
continent_residuals <- map2(unique(gap_unnested$continent),

continent_residuals[[1]]
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~
```



I did this in preparation to saving the files

```
walk2(
  paste0(unique(gap_unnested$continent), "-residuals.pdf"),
  continent_residuals,
  ggsave,
  width = 8, height = 6)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~
## `geom_smooth()` using method = 'loess' and formula 'y ~
## `geom_smooth()` using method = 'loess' and formula 'y ~
## `geom_smooth()` using method = 'loess' and formula 'y ~
## `geom_smooth()` using method = 'loess' and formula 'y ~
```

That code writes out the files

```
Africa-residuals.pdf
Americas-residuals.pdf
Asia-residuals.pdf
Europe-residuals.pdf
Oceania-residuals.pdf
```

Let's return to the use of glance(). We can also unnest() with the lm_glance column (which is a column of row vectors). The use of .drop = TRUE drops additional columns that would otherwise give duplicated rows.

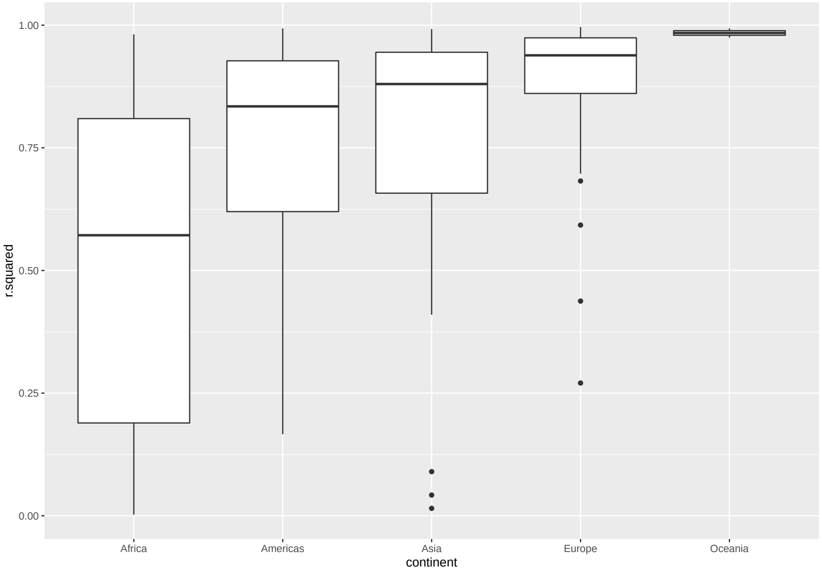
```
gap_glance <- gap_nested_lm %>%
  unnest(lm_glance, .drop = TRUE)
```

```
## Warning: The `.drop` argument of `unnest()` is deprecate
## All list-columns are now preserved.
## This warning is displayed once per session.
## Call `lifecycle::last_warnings()` to see where this warn
```

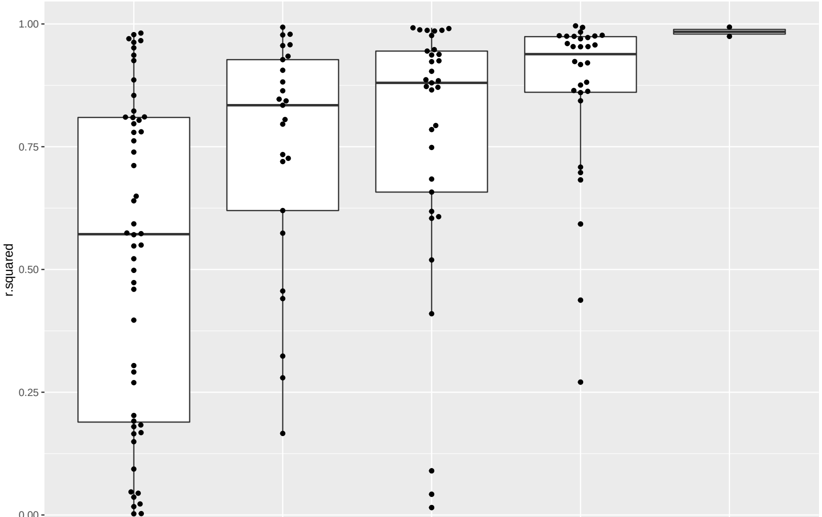
```
print(head(gap_glance), width = Inf)
```

```
## # A tibble: 6 x 16
## # Groups:   country, continent [710]
##   country     continent data              lm_fit r.squar
##   <fct>       <fct>     <list>            <list>     <db
## 1 Afghanistan Asia      <tibble [12 x 5]> <lm>       0.0
## 2 Albania     Europe    <tibble [12 x 5]> <lm>       0.7
```

```r
gap_glance %>%
  ggplot(aes(continent, r.squared)) +
  geom_boxplot()
```

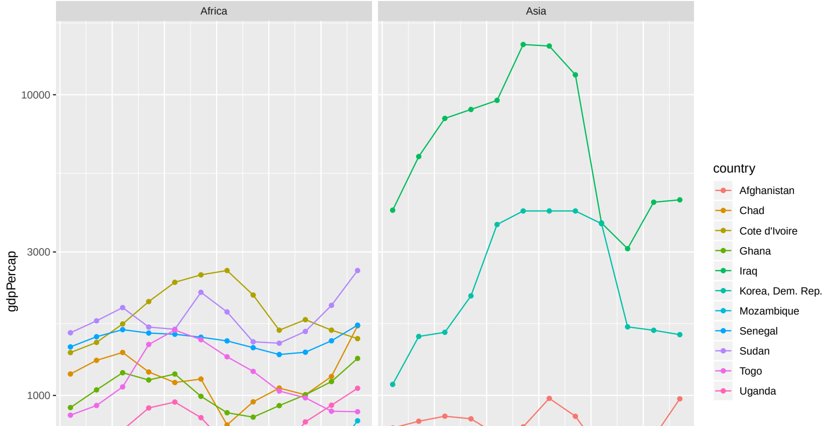


```r
library(ggbeeswarm)
gap_glance %>%
  ggplot(aes(continent, r.squared)) +
  geom_boxplot() +
  geom_beeswarm()
```

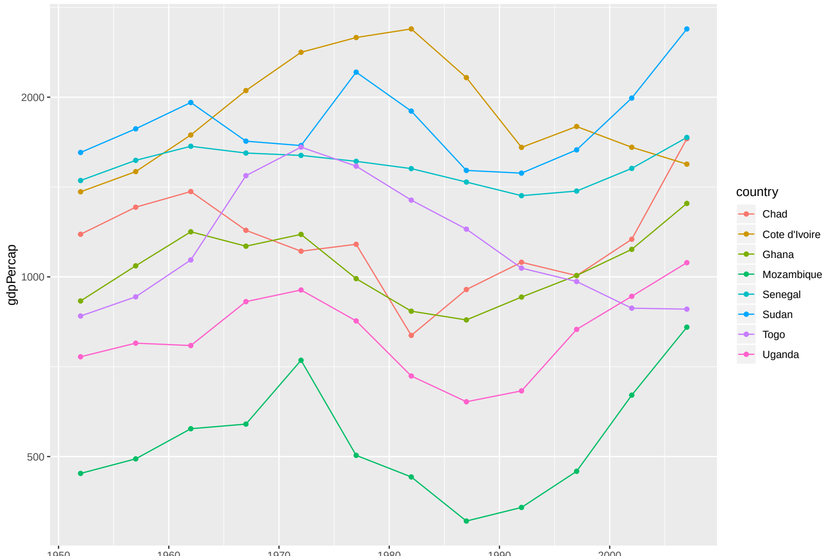


Let's look at the worst fitting countries

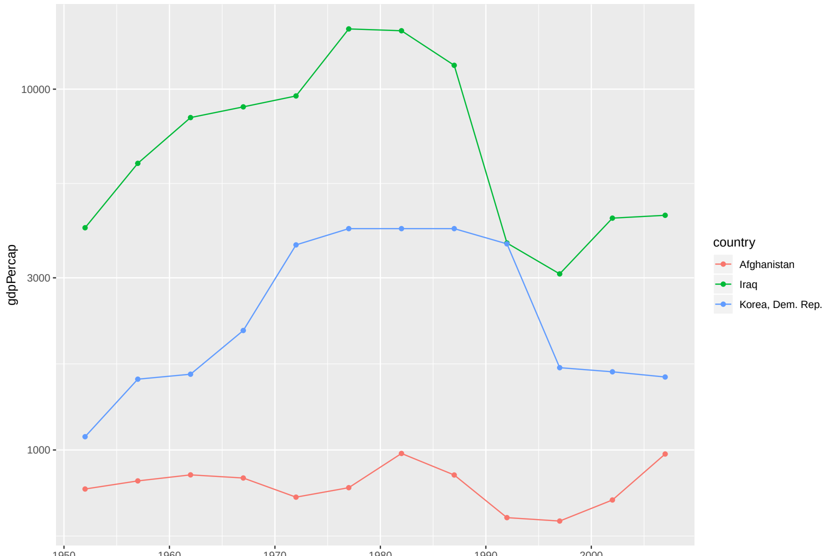
```r
gap_bad_fit <- filter(gap_glance, r.squared < 0.125)

gapminder %>% semi_join(gap_bad_fit, by = "country") %>%
  ggplot(aes(year, gdpPercap, colour = country)) +
    geom_line() + geom_point() +
  scale_y_log10() +  facet_wrap(~ continent)
```



```r
gapminder %>% semi_join(gap_bad_fit, by = "country") %>%
  filter(continent == "Africa") %>%
  ggplot(aes(year, gdpPercap, colour = country)) +
    geom_line() + geom_point() + scale_y_log10()
```



```r
gapminder %>% semi_join(gap_bad_fit, by = "country") %>%
  filter(continent == "Asia") %>%
  ggplot(aes(year, gdpPercap, colour = country)) +
    geom_line() + geom_point() + scale_y_log10()
```



High level view

- If all you need to do is calculate a single number per group, then `group_by()` and `summarize()` should do the trick
- The use of `nest()` is needed for more general group-wise computation
- Typically do not want list-columns as the final product, since they are rather awkward
- The results here could have been done by other approaches, but this approach is both simple and powerful

In class exercise

- load the repurrrsive package and run this code:

```
library(repurrrsive)
```

```
##
## Attaching package: 'repurrrsive'

## The following object is masked _by_ '.GlobalEnv':
##
##     gap_nested
```

```
GoT <- tibble(
  name = got_chars %>% map_chr("name"),
  aliases = got_chars %>% map("aliases"),
  allegiances = got_chars %>% map("allegiances")
)
```

- create a new logical variable in the tibble, corresponding to whether or not "Lannister" is in the allegiances column
- Keep only those rows for which the new variable is TRUE, drop

An example for cross-validation

```
iris.cv <- iris %>%
  crossv_kfold(k = 10)
head(iris.cv)
```

```
## # A tibble: 6 x 3
##   train         test          .id
##   <named list> <named list> <chr>
## 1 <resample>   <resample>   01
## 2 <resample>   <resample>   02
## 3 <resample>   <resample>   03
## 4 <resample>   <resample>   04
## 5 <resample>   <resample>   05
## 6 <resample>   <resample>   06
```

`crossv_kfold()` is part of the `modelr` package

```r
iris.cv <- iris.cv %>%
  mutate(model = map(train, ~lm(Sepal.Length ~ Petal.Length
head(iris.cv)
```

```
## # A tibble: 6 x 4
##   train         test          .id   model
##   <named list> <named list> <chr> <named list>
## 1 <resample>   <resample>   01    <lm>
## 2 <resample>   <resample>   02    <lm>
## 3 <resample>   <resample>   03    <lm>
## 4 <resample>   <resample>   04    <lm>
## 5 <resample>   <resample>   05    <lm>
## 6 <resample>   <resample>   06    <lm>
```

```r
library(magrittr) ## we want to use %$%
iris.cv %$%
  map2_dbl(model, test, rmse)
```

```
##         1         2         3         4         5 
## 0.4166689 0.5002604 0.4606129 0.4023813 0.3836226 0.360
##         9        10 
## 0.4708731 0.4418846
```

```r
# rmse is part of the modelr package
```

We could have done it all in a single (extended) line

```
iris %>%
  crossv_kfold(10) %>%
  mutate(model = map(train, ~lm(Sepal.Length ~ Petal.Lengt
  map2_dbl(model, test, rmse)
```

```
##         1         2         3         4         5
## 0.5746148 0.3584097 0.5395497 0.4414599 0.3083885 0.303
##         9        10
## 0.3660640 0.3769929
```

Jenny Bryan has an edifying and amusing slide deck on this subject