Introduction to ggplot

Max Griswold

2024-02-12

Data for this introduction

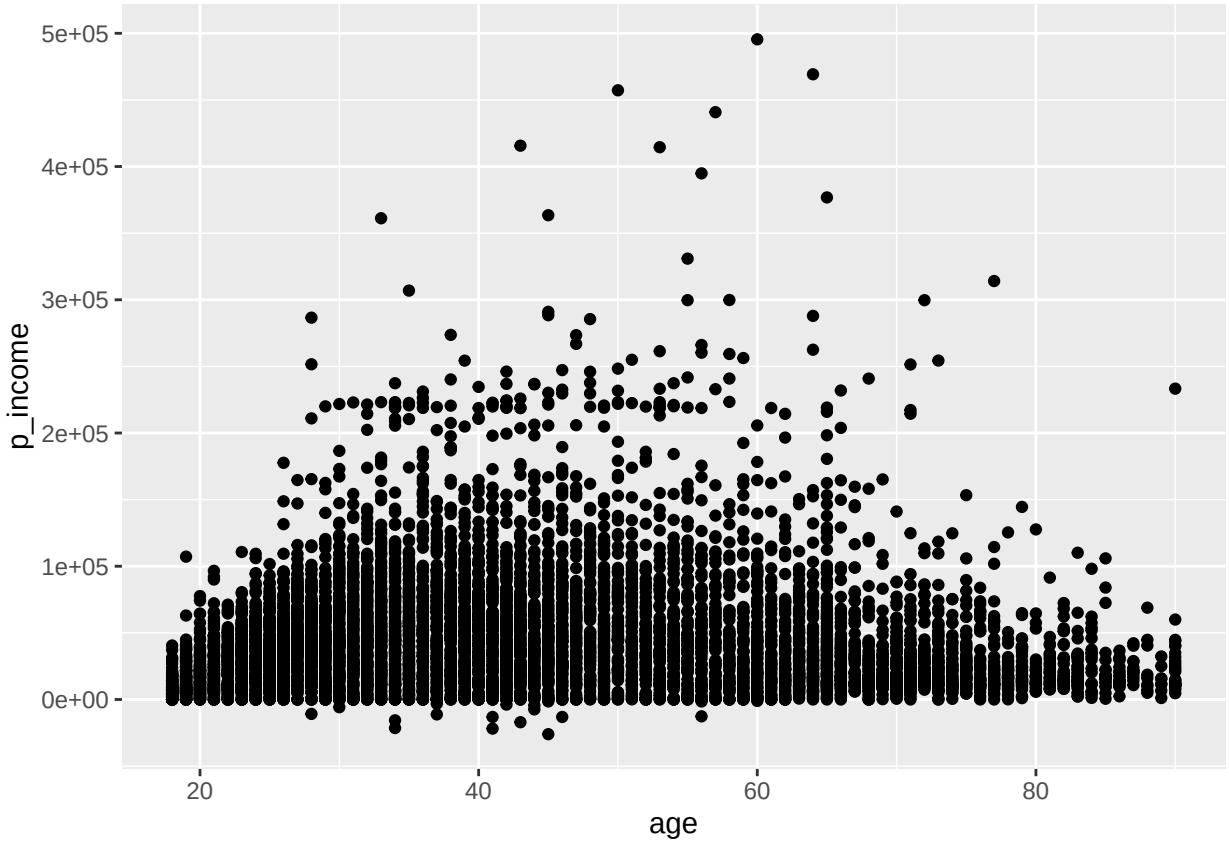
I'm a reusing the 5% sample of CPS between 1980 and 2023 for the examples in this notebook. Please see the code from week 4 to better understand the decision-making behind the constructed variables in this dataset.

Basics of ggplot

As discussed in the lecture, ggplot is based on the grammar of graphics. This framework to graphics uses the idea of layering graphical components to build plots. In ggplot, we accomplish this by using a "+" sign between lines of code to indicate successive layering of components. Note that the order of layers can matter! If you add points to a figure, then lines, the lines will be layered over the points.

I typically start building a figure by first calling my dataset, mapping columns to axes, then creating a specific geometry. To map data to axes, we use the "aes" function within other ggplot functions, like "ggplot" (which starts a figure) or "geom_point" (which adds point geometry)

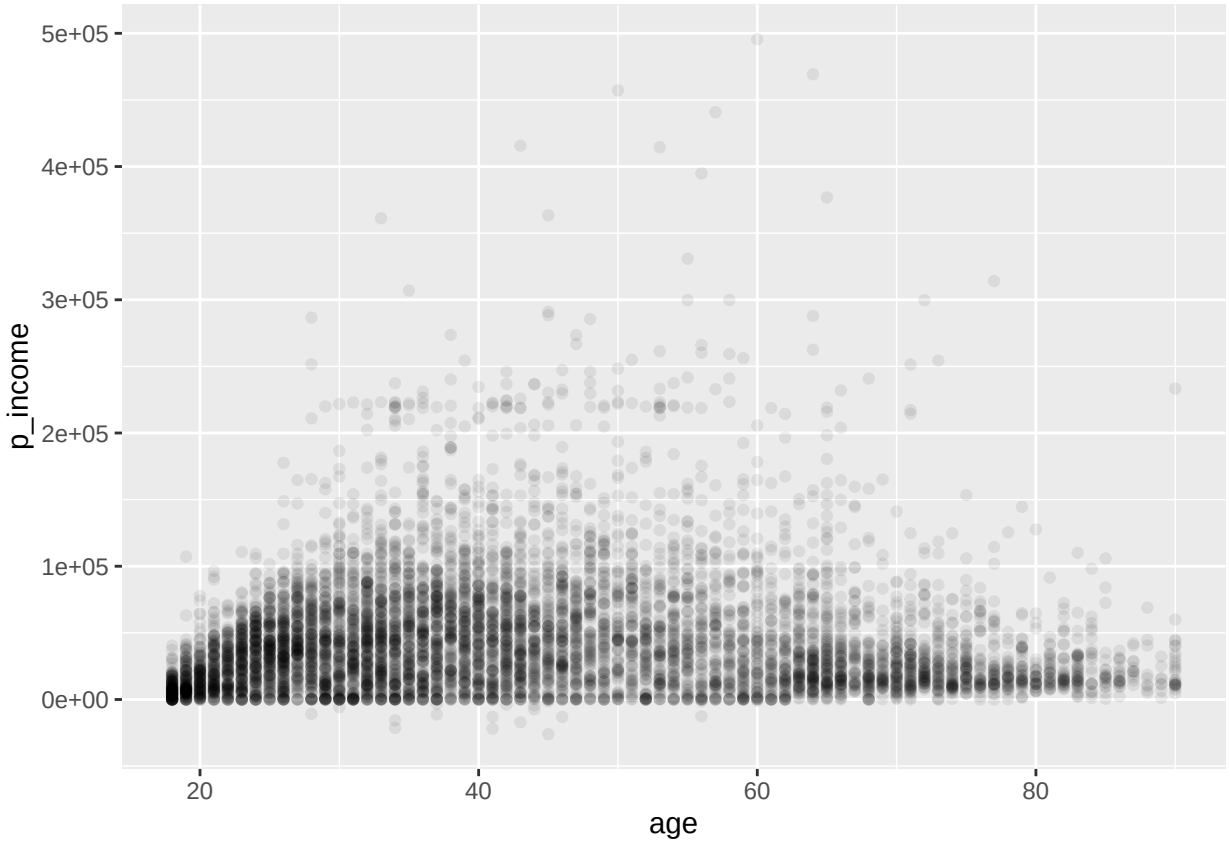
```
library(ggplot2)
library(data.table)

df <- fread("./data/cps/cps_5%_sample_processed.csv")

# Plot income v. age in 1990:
df_plot <- df[year == 1990,]

ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point()
```



Scatterplots

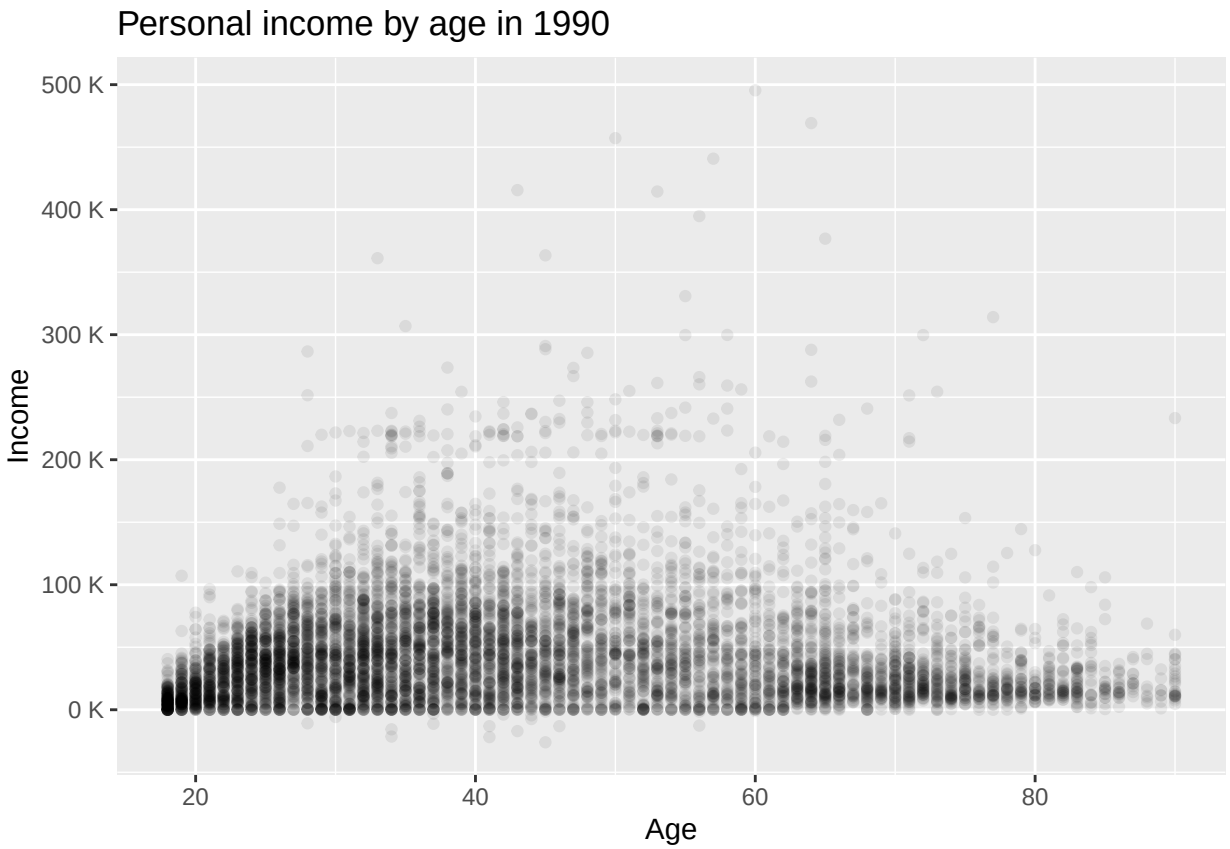
This figure is difficult to read due to the number of points. We can use a component like transparency to better display the relative amount of data we have within the scatter. In ggplot, this is called "alpha" and it is another aesthetic component we can map to a data value if we wanted, by including this within the "aes" function. However, I will instead call this within the geometry itself and set a single value for all points.

```
ggplot(data = df_plot, aes(x = age, y = p_income)) +

  # I chose the alpha value by playing around with potential amounts between 0 & 1,
  # finding one so that points were still visible but the amount was clearer.
  geom_point(alpha = 0.07)
```



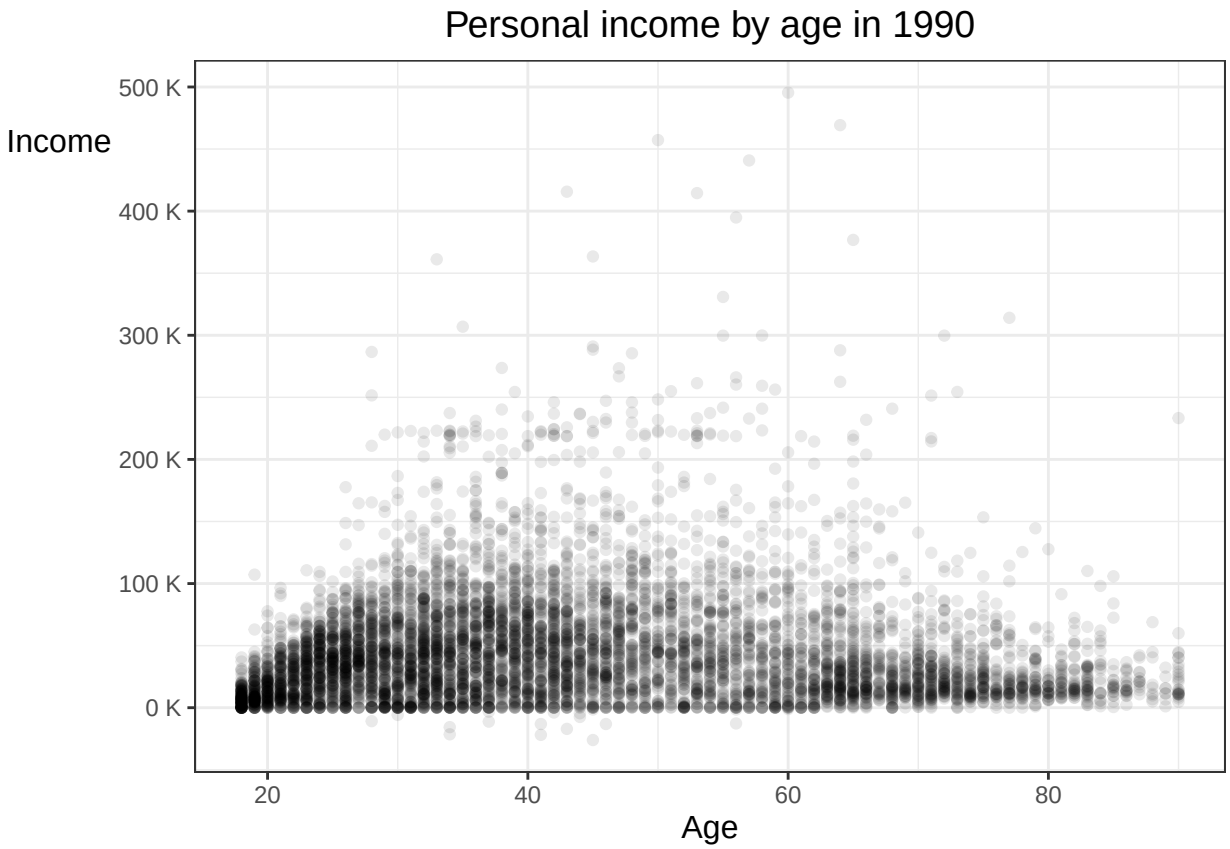
We can also modify axis labels and scales to better display the data. I personally do not like using scientific notation for public facing figures so I'll use a scale component to modify how the y-axis is displayed (scale_y_continuous). I will also use a label component (labs) to modify the axis titles and the plot title:

```
# Calling the scales package to modify the y-axis labels into thousands:
library(scales)

ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point(alpha = 0.07) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3))
```



ggplot also provides robust themes to change the background of the plot, the tick marks, the position, font, and size of text, etc. We can access these by providing ggplot a theme template or modifying specific "theme" components. I do both in the next plot. You can find a library of ggplot themes that are built into the package here. There is also the package ggthemes which provides lots of alternative styles which match other software packages or newsprint styles (like the economist or 538).

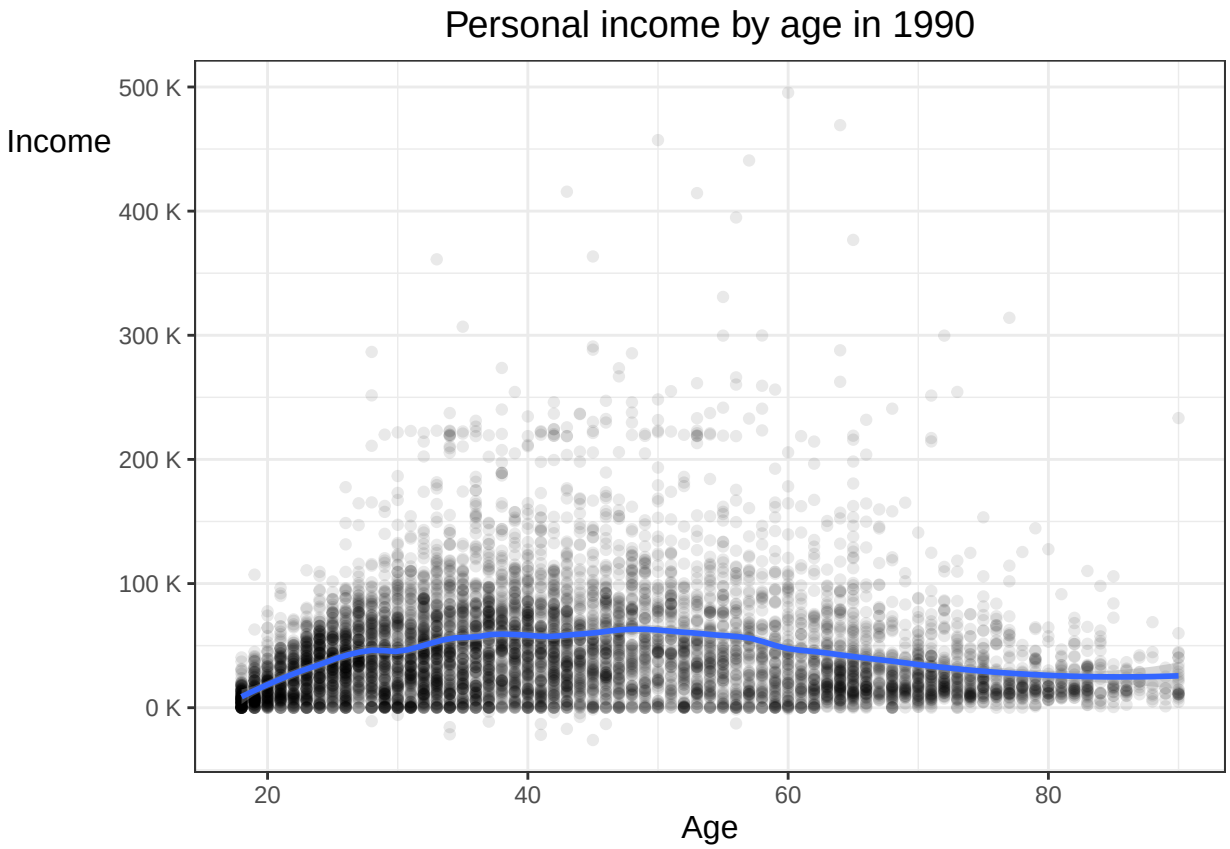
```
ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point(alpha = 0.09) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
  theme_bw() +
  # You can see an exhaustive list of theme modifiers by calling "help"
  # on "theme". Most modifiers use an "element_" function which provides
  # even more granular control
  theme(plot.title = element_text(hjust = 0.5, size = 14),
        axis.title.y = element_text(angle = 0, vjust = 0.9, size = 12),
        axis.title.x = element_text(size = 12))
```



We can also add a summary statistic to the figure to better detect trends. Here, I'm using a nonlinear smoothing function called LOESS to better capture the nonlinearity within the data. The "span" parameter in LOESS controls the smoothness of the estimated curve, controlling the bandwidth of a local window used to estimate LOESS (e.g., setting span at 0.2 below means for a given x-value, 20% of data points are being used to estimate a weighted average for that x).

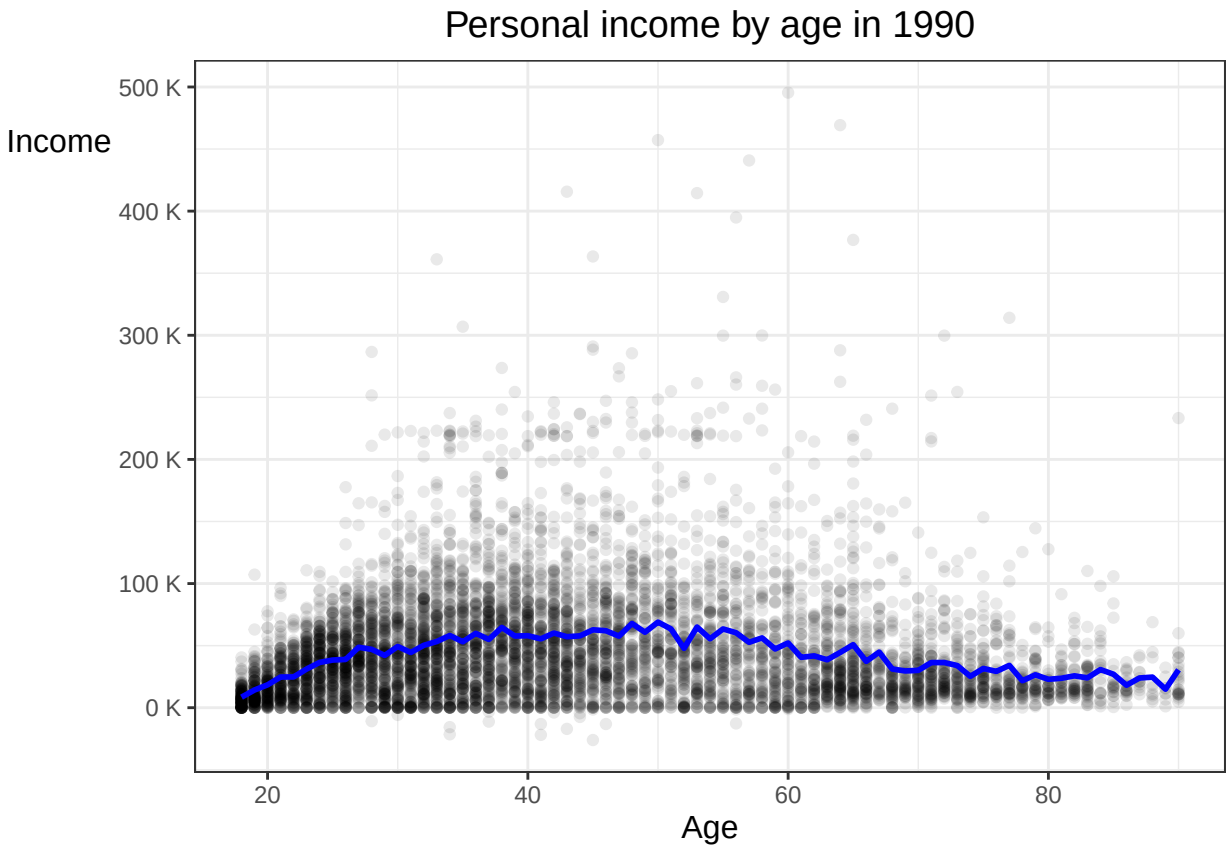
```
df_plot[, age_average := mean(.SD$p_income), by = "age"]

ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point(alpha = 0.09) +
  stat_smooth(method = "loess", span = 0.2) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5, size = 14),
        axis.title.y = element_text(angle = 0, vjust = 0.9, size = 12),
        axis.title.x = element_text(size = 12))
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



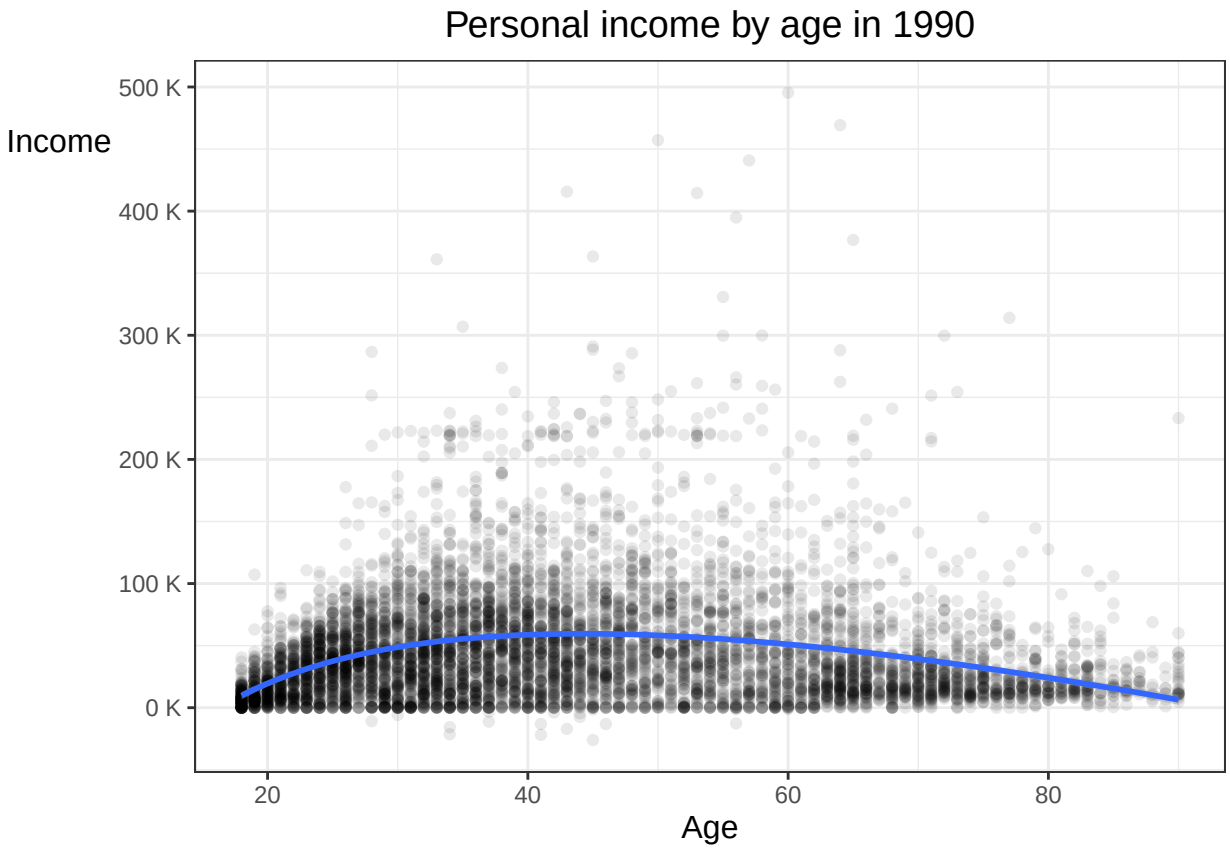
However, in applied work, I typically use models or plot population averages as a line to better display summary trends. I'm further modifying the line by modifying the size and color parameter for this geom.

```
df_plot[, age_average := mean(.SD$p_income), by = "age"]

ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point(alpha = 0.09) +

  # This line won't be as smooth as LOESS but still captures the basic idea.
  geom_line(aes(y = age_average), color = "blue", size = 1) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5, size = 14),
        axis.title.y = element_text(angle = 0, vjust = 0.9, size = 12),
        axis.title.x = element_text(size = 12))
```



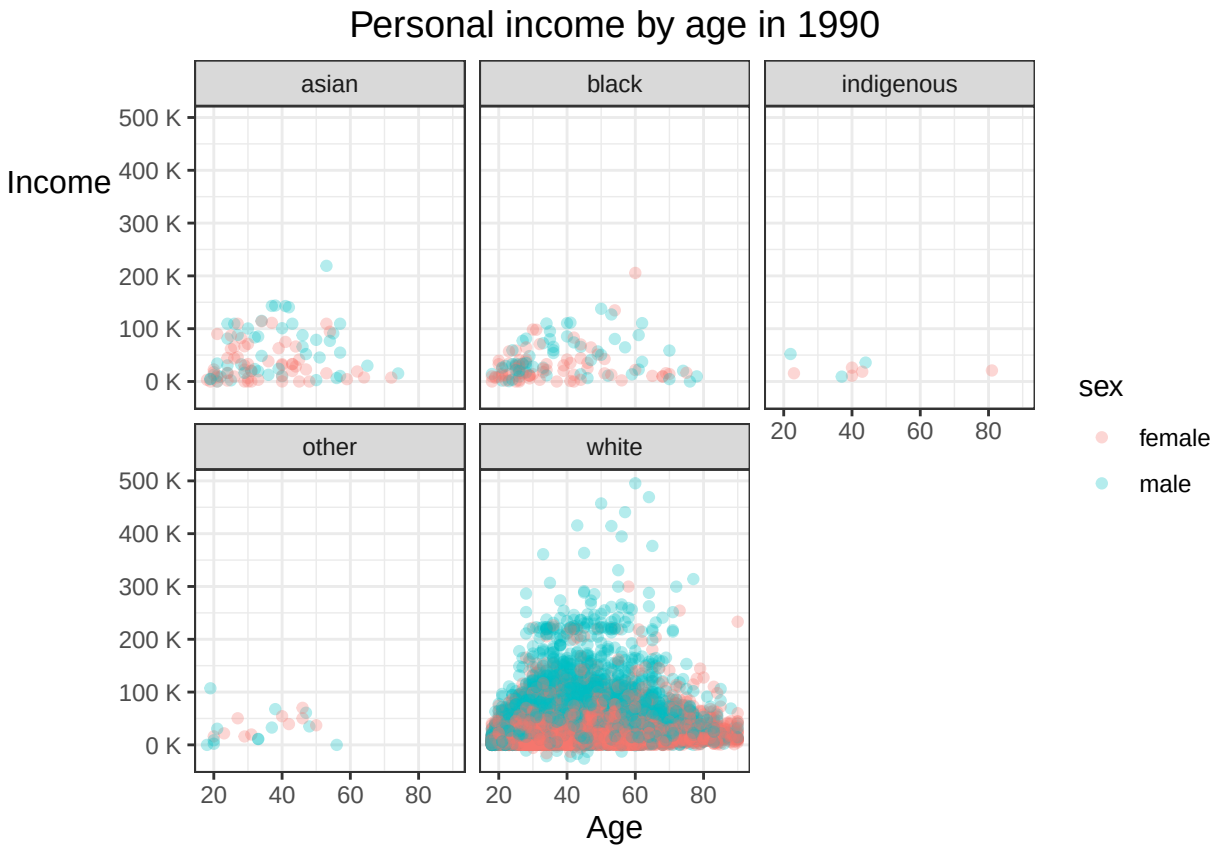
And alternatively, using a model:

```
ggplot(data = df_plot, aes(x = age, y = p_income)) +
  geom_point(alpha = 0.09) +
  stat_smooth(method = "lm", formula = y ~ x + log(x)) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5, size = 14),
        axis.title.y = element_text(angle = 0, vjust = 0.9, size = 12),
        axis.title.x = element_text(size = 12))
```



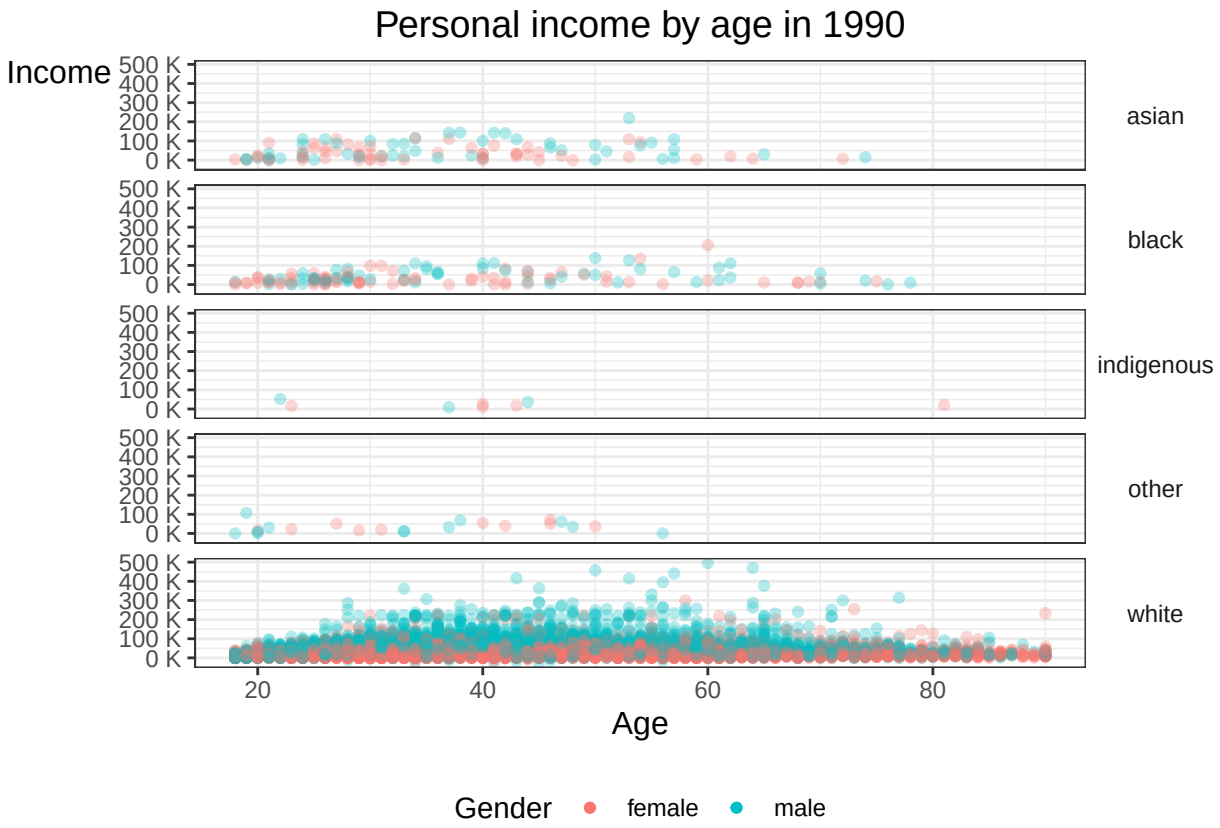
One of the most powerful aspects of ggplot and the grammar of graphics comes from layering multiple scaled aesthetics with small multiples using "facets" in ggplot. In the below plot, I create small multiples by race and use color to show differences by sex:

```
# To do this, all I needed to do was add "color = race" to the original
# aes call, along with adding "facet_wrap".

ggplot(data = df_plot, aes(x = age, y = p_income, color = sex)) +
  geom_point(alpha = 0.3) +
  facet_wrap(~race) +
  labs(x = "Age",
       y = "Income",
       title = "Personal income by age in 1990") +
  scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5, size = 14),
        axis.title.y = element_text(angle = 0, vjust = 0.9, size = 12),
        axis.title.x = element_text(size = 12))
```



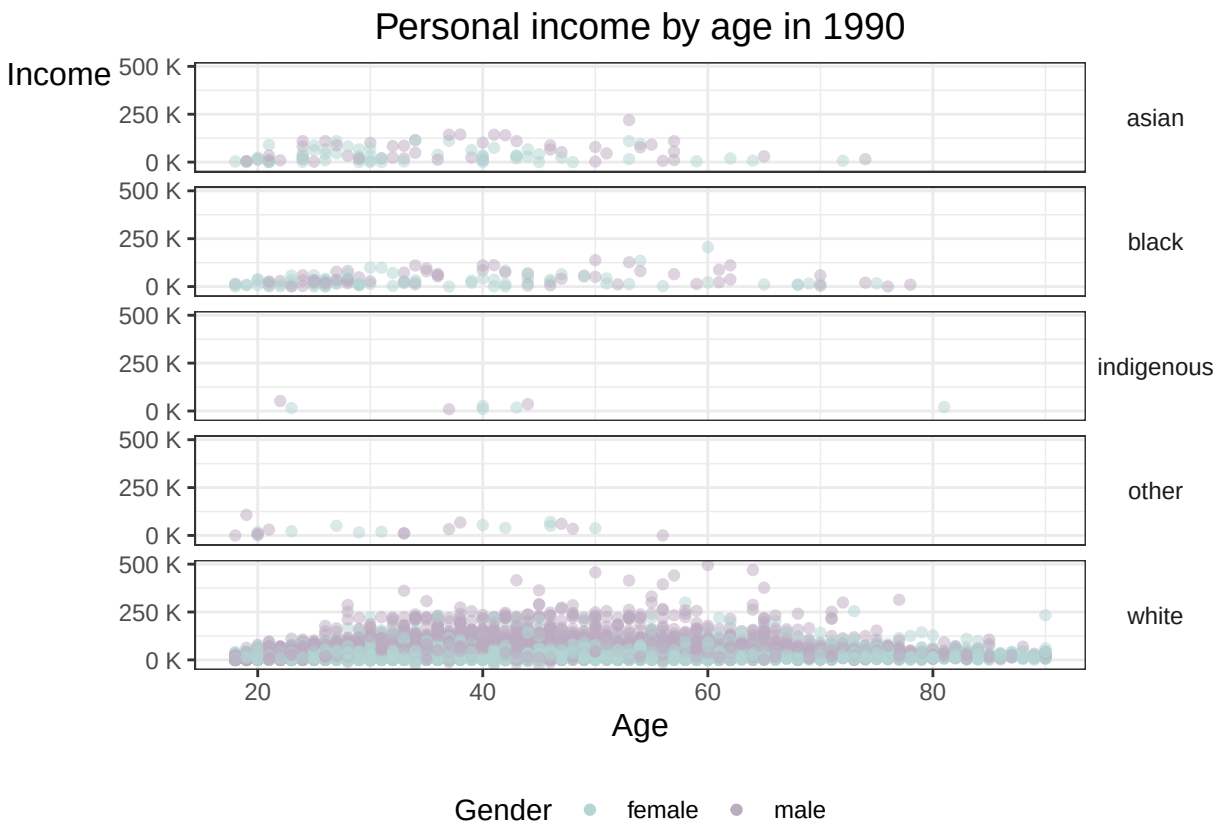
We can improve this plot a little by further modifying the position of the legend within the theme and override the aesthetic options in the plot for the legend (like the use of alpha). We can also change the ordering of the facets and change the placement of the facet labels within the theme function.

```
ggplot(data = df_plot, aes(x = age, y = p_income, color = sex)) +
        geom_point(alpha = 0.3) +
        facet_wrap(~race, nrow = 5, strip.position = "right") +
        labs(x = "Age",
             y = "Income",
             title = "Personal income by age in 1990",
             color = "Gender") +
        scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3)) +
        guides(color = guide_legend(override.aes = list(alpha = 1))) +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5, size = 14),
              axis.title.y = element_text(angle = 0, vjust = 1, size = 12),
              axis.title.x = element_text(size = 12),
              strip.background = element_blank(),
              strip.text.y = element_text(angle = 0),
              legend.position = "bottom")
```



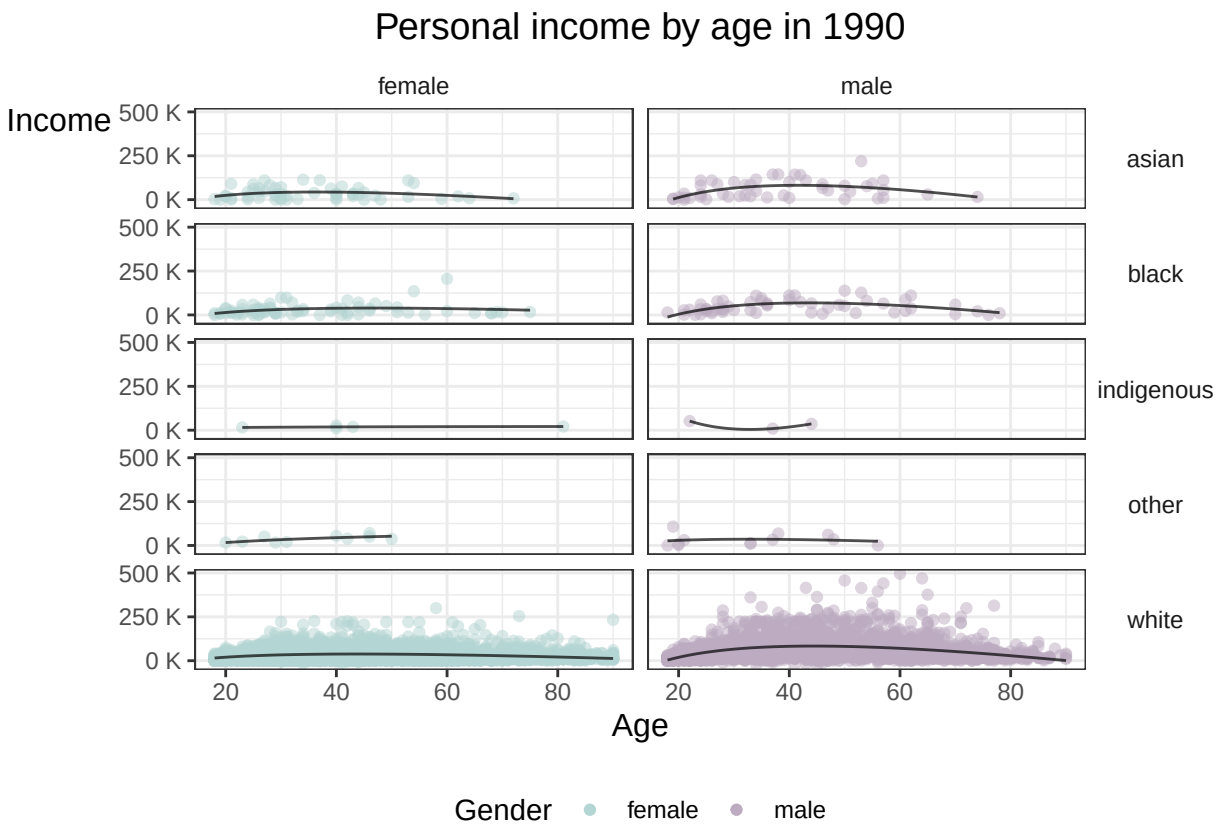
Lastly, we can modify the breaks displayed in the figure so it looks a little bit more reasonable by further using the scales functions and change the colors of the plot to be more aesthetically pleasing. I commonly use the colorbrewer tool to find color palettes that work well in ggplot, along with making my own color palettes.

```
ggplot(data = df_plot, aes(x = age, y = p_income, color = sex)) +
        geom_point(alpha = 0.5) +
        facet_wrap(~race, nrow = 5, strip.position = "right") +
        labs(x = "Age",
             y = "Income",
             title = "Personal income by age in 1990",
             color = "Gender") +
        scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3),
                           breaks = c(0, 2.5e5, 5e5)) +
        scale_color_manual(values = c("#b3d5d3", "#bcabc1")) +
        guides(color = guide_legend(override.aes = list(alpha = 1))) +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5, size = 14),
              axis.title.y = element_text(angle = 0, vjust = 1, size = 12),
              axis.title.x = element_text(size = 12),
              strip.background = element_blank(),
              strip.text.y = element_text(angle = 0),
              legend.position = "bottom")
```



A scatter plot is likely not doing justice to the difference in income status between these two groups. Instead, let's use additional small multiples for gender and include a summary measure

```
ggplot(data = df_plot, aes(x = age, y = p_income, color = sex)) +
        geom_point(alpha = 0.5) +
        # Note that I'm using facet_grid rather than facet_wrap now:
        facet_grid(race~sex) +
        stat_smooth(method = 'lm', formula = y ~ x + log(x), se = F,
                    color = 'black', alpha = 0.7, geom ='line') +
        labs(x = "Age",
             y = "Income",
             title = "Personal income by age in 1990",
             color = "Gender") +
        scale_y_continuous(labels = unit_format(unit = "K", scale = 1e-3),
                           breaks = c(0e5, 2.5e5, 5e5)) +
        scale_color_manual(values = c("#b3d5d3", "#bcabc1")) +
        guides(color = guide_legend(override.aes = list(alpha = 1))) +
        theme_bw() +
        theme(plot.title = element_text(hjust = 0.5, size = 14),
              axis.title.y = element_text(angle = 0, vjust = 1, size = 12),
              axis.title.x = element_text(size = 12),
              strip.background = element_blank(),
              strip.text.y = element_text(angle = 0),
              legend.position = "bottom")
```



Bar charts

What if we're interested in the distribution of income by age? ggplot also makes it easy to construct a bar chart to display this information. We'll need to do a little preprocessing on our data first to prepare our data for this figure.

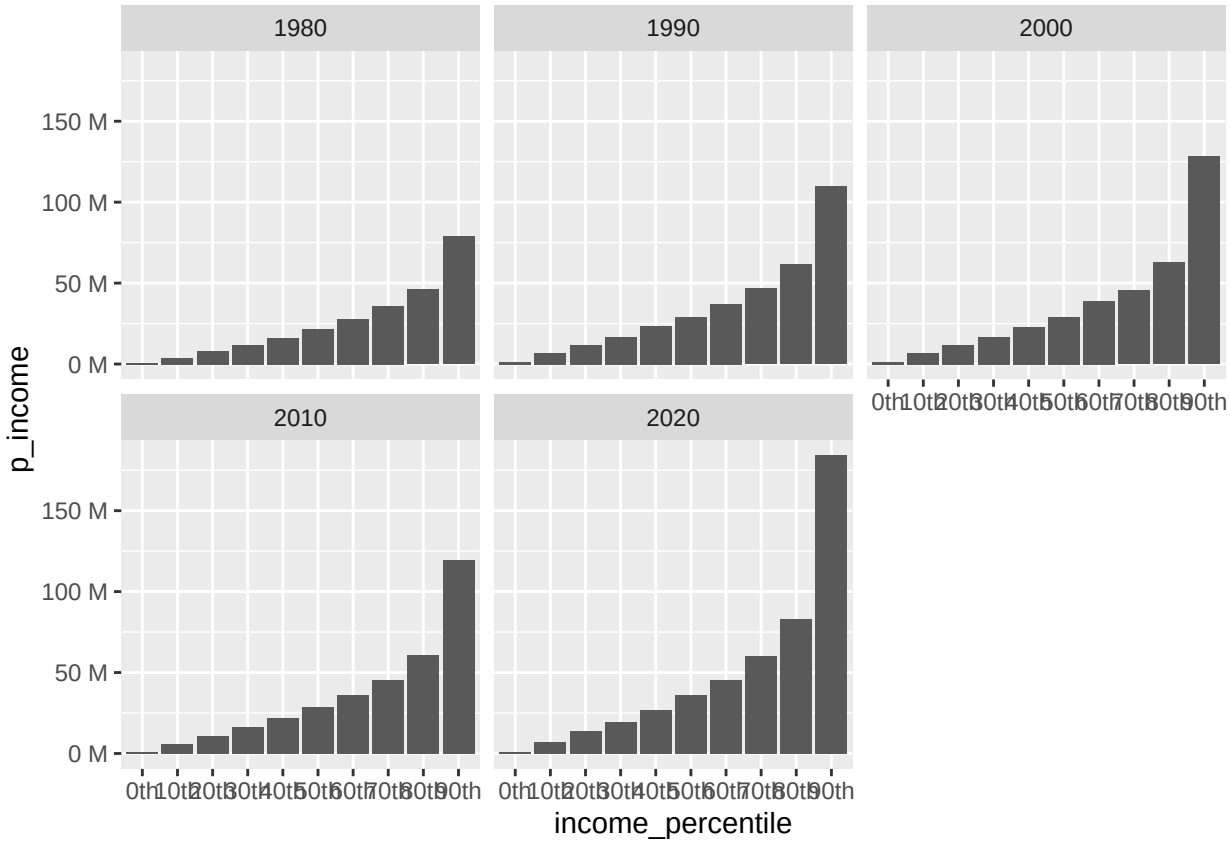
```
# Construct 5-year age groups

df_bar <- df[, .(year, p_income, p_income_rank, age, sex, marital_status, race)]

income_groups <- paste0(seq(0, 90, 10), "th")
df_bar[, income_percentile := cut(p_income_rank, breaks = seq(0, 1, 0.1),
                                  labels = income_groups, include.lowest = T,
                                  right = F)]

age_groups <- c("[18 - 25)", paste0("[", seq(25, 70, 5), ", ",
                                    c(seq(30, 70, 5), "75+"), ")"))

df_bar[, age_groups := cut(age, breaks = c(18, seq(25, 70, 5), 100),
                           labels = age_groups, include.lowest = T,
                           right = F)]

head(df_bar)
```

```
##    year    p_income p_income_rank age    sex marital_status  race
## 1: 1980 17399.28861   0.374334547  85 female         former white
## 2: 1980  4382.54164   0.143317456  20 female        married white
```

```
## 3: 1980 29802.77890   0.545110675  23 female        married white
## 4: 1980 10993.74780   0.248108714  62 female        married white
## 5: 1980     7.47874   0.045530961  57 female        married white
## 6: 1980     0.00000   0.003642477  54 female         single white
##    income_percentile age_groups
## 1:              30th  [70, 75+)
## 2:              10th  [18 - 25)
## 3:              50th  [18 - 25)
## 4:              20th   [60, 65)
## 5:               0th   [55, 60)
## 6:               0th   [50, 55)
```

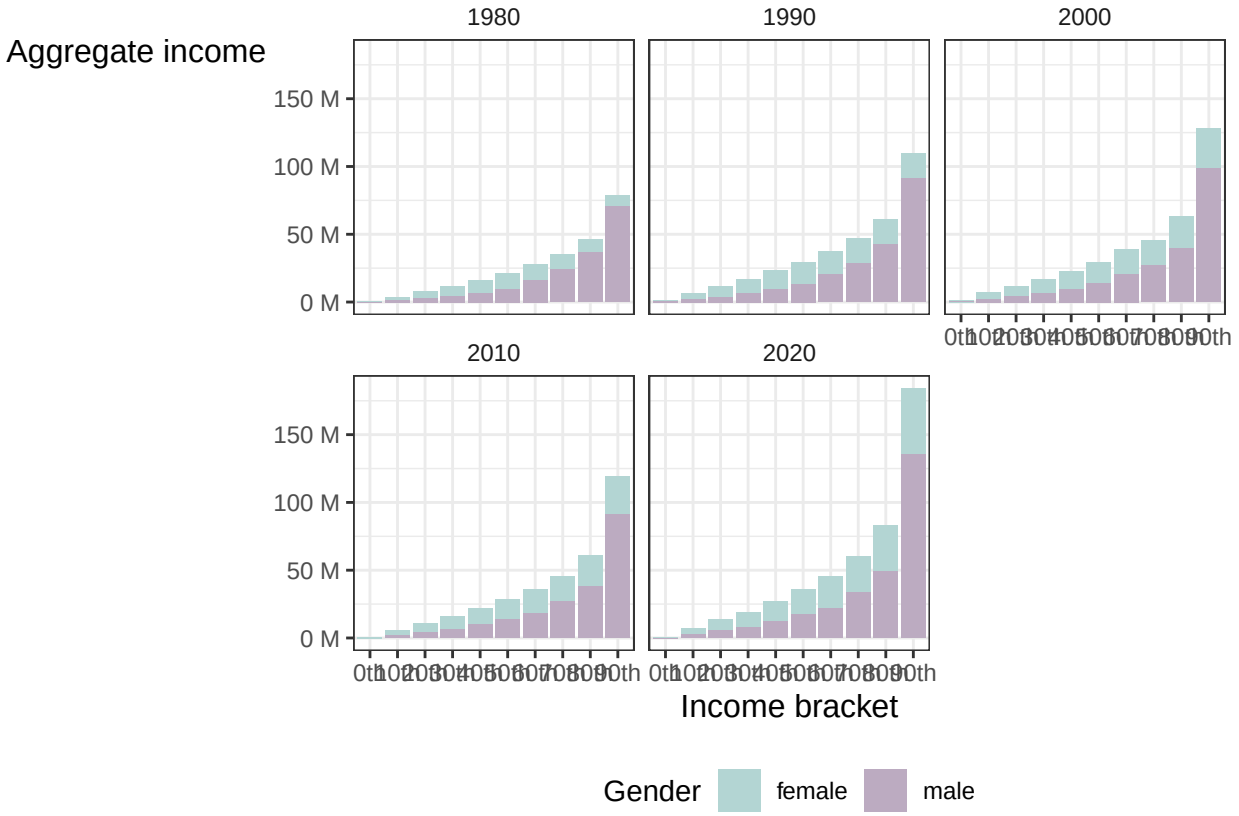
Let's do a naive bar chart plotting income amounts by 10 years:

```
# I'm being a little tricky here and using the modulo operator to capture
# years by 10
df_bar <- df_bar[year %% 10 == 0,]

# Note what the below plot is displaying: For each income percentile and year,
# geom col is adding up personal income within that bracket to generate the plot.
# Since we're using percentiles as groups, we are guaranteeing equivalent sample
# size across income percentiles.

ggplot(df_bar, aes(y = p_income, x = income_percentile)) +
  geom_col() +
  scale_y_continuous(labels = unit_format(unit = "M", scale = 1e-6)) +
  facet_wrap(~year)
```



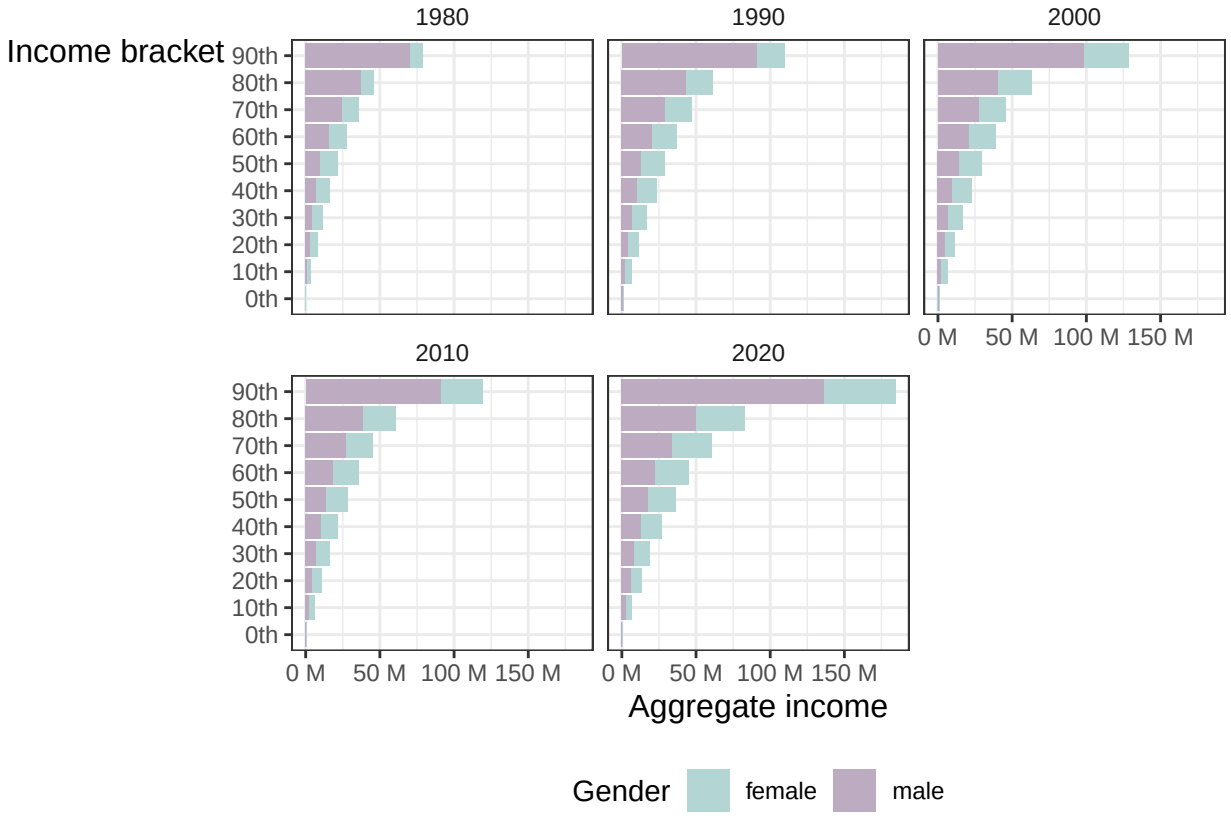
Let's modify the aesthetics a little bit and use "fill" to display the portion of yearly income earned by men and women.

```
ggplot(df_bar, aes(y = p_income, x = income_percentile, fill = sex)) +
  geom_col() +
  scale_y_continuous(labels = unit_format(unit = "M", scale = 1e-6)) +
  facet_wrap(~year) +
  labs(x = "Income bracket", y = "Aggregate income", fill = "Gender") +
  scale_fill_manual(values = c("#b3d5d3", "#bcabc1")) +
  theme_bw() +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0),
        axis.title = element_text(size = 12),
        strip.background = element_blank())
```



Using coordinate references, we can flip this figure to make it a little more readable:

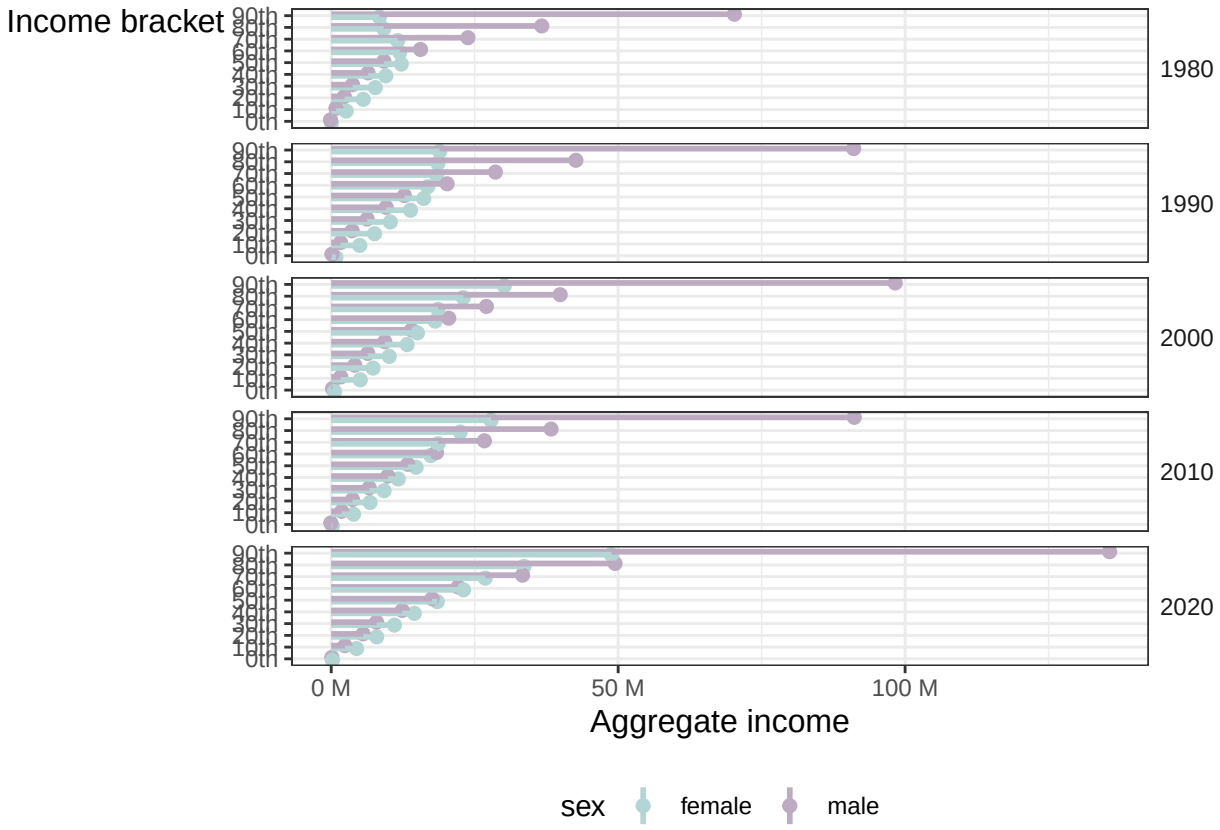
```
ggplot(df_bar, aes(y = p_income, x = income_percentile, fill = sex)) +
  geom_col() +
  scale_y_continuous(labels = unit_format(unit = "M", scale = 1e-6)) +
  facet_wrap(~year) +
  coord_flip() +
  labs(x = "Income bracket", y = "Aggregate income", fill = "Gender") +
  scale_fill_manual(values = c("#b3d5d3", "#bcabc1")) +
  theme_bw() +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0),
        axis.title = element_text(size = 12),
        strip.background = element_blank())
```



Rather than using bars, we could instead use points and thin lines (lollipop chart)

```
# I find it easiest to calculate the statistic I need using the dataset rather
# than attempt to use stat_summary in ggplot. So, let's calculate aggregate income
# by age and sex, then map this geoms:

df_agg <- copy(df_bar)
df_agg[, agg_income := sum(.SD$p_income), by = c("year", "income_percentile", "sex")]

df_agg <- unique(df_agg[, .(agg_income, income_percentile, sex, year)])

# I'm using position_dodge to separate points and segments by sex
ggplot(df_agg, aes(y = agg_income, x = income_percentile, color = sex)) +
  geom_point(position = position_dodge(width = 0.5), size = 2) +
  geom_linerange(aes(ymin = 0, ymax = agg_income,
                     xmin = income_percentile, xmax = income_percentile),
                 position = position_dodge(width = 0.5), size = 1) +
  scale_y_continuous(labels = unit_format(unit = "M", scale = 1e-6)) +
  facet_wrap(~year, nrow = 5, strip.position = "right") +
  coord_flip() +
  labs(x = "Income bracket", y = "Aggregate income", fill = "Gender") +
  scale_color_manual(values = c("#b3d5d3", "#bcabc1")) +
  theme_bw() +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0),
        axis.title = element_text(size = 12),
```

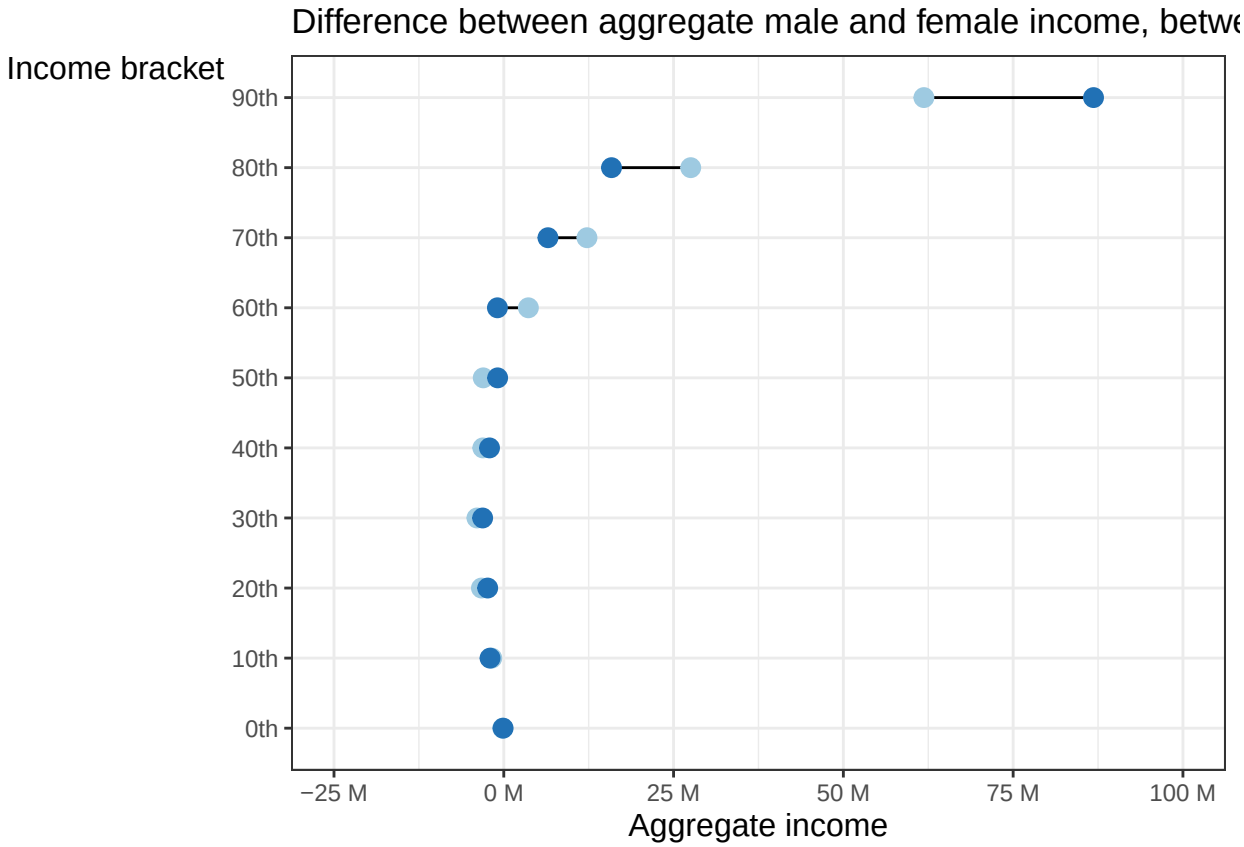
```
        strip.background = element_blank(),
        strip.text.y = element_text(angle = 0))
```



Plot above is difficult to read. We could instead follow the suggestion of Schwabish and look at differences instead, calculating this between 1980 and 2020:

```
df_agg <- dcast(df_agg, formula = income_percentile ~ sex + year, value.var = "agg_income")

df_agg[, diff_1980 := male_1980 - female_1980]
df_agg[, diff_2020 := male_2020 - female_2020]

df_agg <- df_agg[, .(income_percentile, diff_1980, diff_2020)]

# I'm using two separate geom_points and manually setting the colors:
# Within the title, I'm using html to add color based on ggtext package
library(ggtext)

ggplot(df_agg, aes(x = income_percentile)) +
  geom_linerange(aes(ymin = diff_1980, ymax = diff_2020,
                     xmin = income_percentile, xmax = income_percentile)) +
  geom_point(aes(y = diff_1980), color = "#9ecae1", size = 3) +
  geom_point(aes(y = diff_2020), color = "#2171b5", size = 3) +
  scale_y_continuous(labels = unit_format(unit = "M", scale = 1e-6),
                     breaks = c(-2.5e7, 0, 2.5e7, 5e7, 7.5e7, 1e8),
                     limits = c(-2.5e7, 1e8)) +
  coord_flip() +
```

```
  labs(x = "Income bracket", y = "Aggregate income",
       fill = "Gender", title = "Difference between aggregate male and female income, between <span sty
  theme_bw() +
  theme(legend.position = "bottom",
        axis.title.y = element_text(angle = 0),
        axis.title = element_text(size = 12),
        strip.background = element_blank(),
        strip.text.y = element_text(angle = 0),
        plot.title = element_markdown())
```



Maps

To make maps, I'll need to add geographic indicators to the CPS dataset. To do this, I'm merging on geography shapefiles produced by the Census called "TIGER shapefiles". These are available for all Census geographical units and are useful as boundary maps. You can find these files by year on the census website.

```
# I like using the sf package to manipulate spatial dataframes.
# There is a cheatsheet on the class page for this package which covers the basic
# functions.

# Alternatively, there is the terra package for raster data and the sp package
# for spatial methods.

library(sf)
```

```
## Linking to GEOS 3.9.3, GDAL 3.5.2, PROJ 8.2.1; sf_use_s2() is TRUE
```

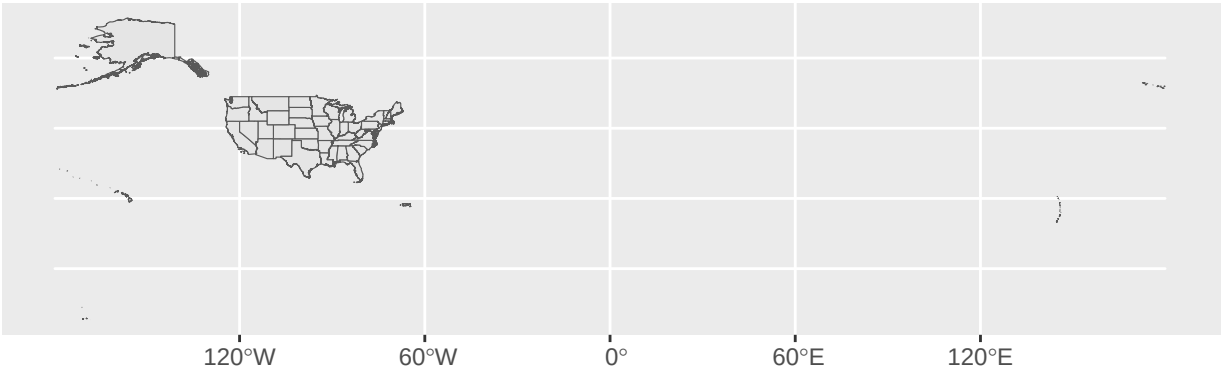
```
state_bounds <- st_read("./data/helper/state_bounds/")
```

```
## Reading layer 'us_state' from data source
##   'C:\Users\griswold\Documents\GitHub\Quantitative-Analysis-Practicum-2024\data\helper\state_bounds'
##   using driver 'ESRI Shapefile'
## Simple feature collection with 56 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -179.1489 ymin: -14.5487 xmax: 179.7785 ymax: 71.36516
## Geodetic CRS:  NAD83
```

```
print(state_bounds)
```

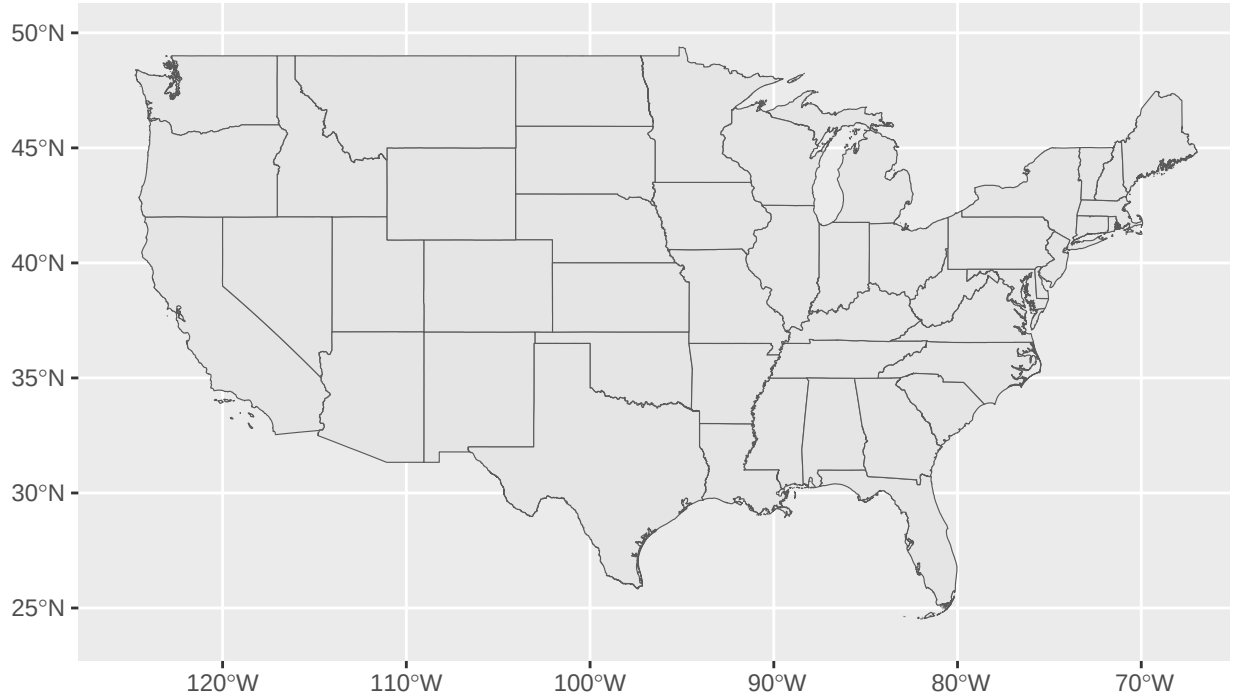
```
## Simple feature collection with 56 features and 9 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -179.1489 ymin: -14.5487 xmax: 179.7785 ymax: 71.36516
## Geodetic CRS:  NAD83
## First 10 features:
##    STATEFP  STATENS    AFFGEOID GEOID STUSPS           NAME LSAD        ALAND
## 1       28 01779790 0400000US28    28     MS    Mississippi   00 121533519481
## 2       37 01027616 0400000US37    37     NC North Carolina   00 125923656064
## 3       40 01102857 0400000US40    40     OK       Oklahoma   00 177662925723
## 4       51 01779803 0400000US51    51     VA       Virginia   00 102257717110
## 5       54 01779805 0400000US54    54     WV  West Virginia   00  62266474513
## 6       22 01629543 0400000US22    22     LA      Louisiana   00 111897594374
## 7       26 01779789 0400000US26    26     MI       Michigan   00 146600952990
## 8       25 00606926 0400000US25    25     MA  Massachusetts   00  20205125364
## 9       16 01779783 0400000US16    16     ID          Idaho   00 214049787659
## 10      12 00294478 0400000US12    12     FL        Florida   00 138949136250
##          AWATER                       geometry
## 1    3926919758 MULTIPOLYGON (((-88.50297 3...
## 2   13466071395 MULTIPOLYGON (((-75.72681 3...
## 3    3374587997 MULTIPOLYGON (((-103.0026 3...
## 4    8528531774 MULTIPOLYGON (((-75.74241 3...
## 5     489028543 MULTIPOLYGON (((-82.6432 38...
## 6   23753621895 MULTIPOLYGON (((-88.8677 29...
## 7  103885855702 MULTIPOLYGON (((-83.19159 4...
## 8    7129925486 MULTIPOLYGON (((-70.23405 4...
## 9    2391722557 MULTIPOLYGON (((-117.2427 4...
## 10  31361101223 MULTIPOLYGON (((-80.17628 2...
```

```
# ggplot makes it easy to produce maps. At the most basic, we can
# plot just the geography using geom_sf

ggplot(data = state_bounds) +
  geom_sf()
```



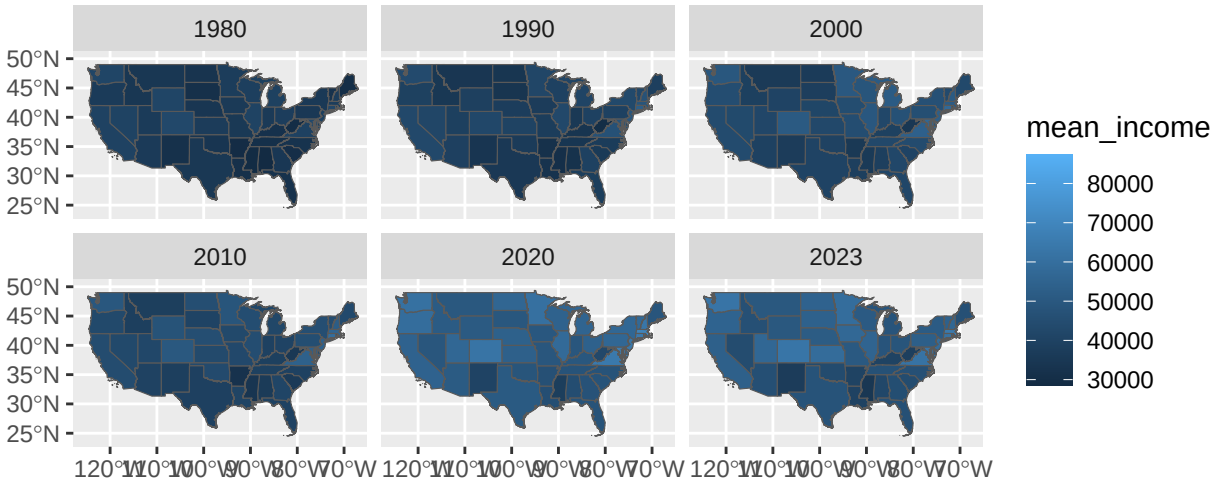
The shapefile from Census includes geography for territories, along with a large Alaska and a distant Hawaii. For applied work, I would try to find a shapefile that shows Alaska and Hawaii as map insets (or make insets myself). For now, to keep things simple, I'll crop out these states and only display the 48 contiguous states.

```
# We can use lims to bound the map. I chose the limits by eyeballing the coordinates

ggplot(data = state_bounds) +
  geom_sf() +
  lims(x = c(-125, -68), y = c(24, 50))
```



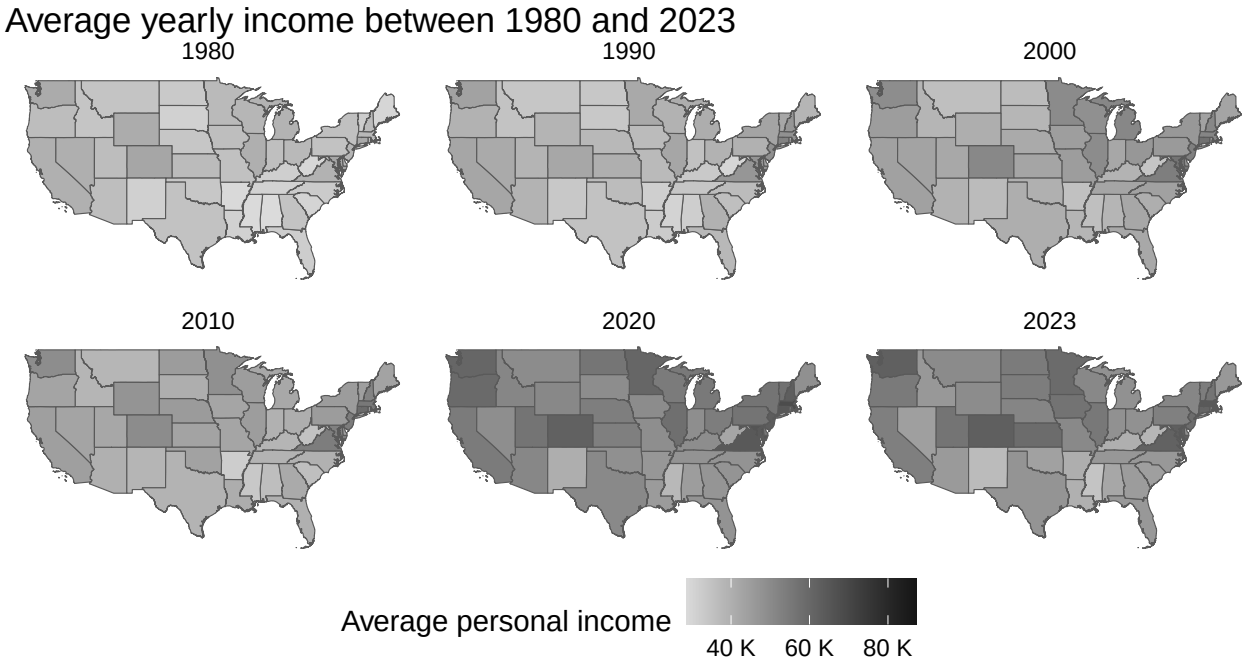
Let's merge the state bounds with the cps data.

```
# I calculated mean income within each state by year using the full CPS
# sample. You can find this dataset in the data folder.

df_map <- fread("./data/cps/cps_state_year_mean_income.csv")

# Let's only hold onto 10 year increments, along with 2023:
df_map <- df_map[year %% 10 == 0 | year == 2023,]

# Merge on state boundaries
df_map <- merge(df_map, state_bounds, by.x = "state", by.y = "NAME", all.y = F)

# Now need to transform this dataset back into a spatial dataframe so we can plot it.
df_map <- st_as_sf(df_map)

# To display the data, I'm using the fill aesthetic.
ggplot(df_map, aes(fill = mean_income)) +
  geom_sf() +
  lims(x = c(-125, -68), y = c(24, 50)) +
  facet_wrap(~year)
```



Similar to the scatterplots and bar charts, we can modify the aesthetic elements to improve the visual fidelity of the maps. I like to also modify the theme to remove coordinates on the axes. I'm also using scale_fill_gradient to create a custom color gradient.

```
# To display the data, I'm using the fill aesthetic.
ggplot(df_map, aes(fill = mean_income)) +
  geom_sf() +
  lims(x = c(-125, -68), y = c(24, 50)) +
  labs(fill = "Average personal income",
       title = "Average yearly income between 1980 and 2023") +
  facet_wrap(~year) +
  scale_fill_gradient(low = "#dbdbdb", high = "#141414",
                      labels = unit_format(unit = "K", scale = 1e-3),
                      breaks = seq(2e4, 1e5, 2e4)) +
  theme_void() +
  theme(legend.position = "bottom",
        strip.background = element_blank())
```



Lots to modify further at this point to improve the graphic's legibility, such as revising how the legend is displayed or how much of each state is filled, perhaps based on sample size! But this seems like a good stopping point.