GIS in R: custom legends

Learning Objectives

After completing this tutorial, you will be able to:

- Add a custom legend to a map in `R`.
- Plot a vector dataset by attributes in `R`.

What you need

You will need a computer with internet access to complete this lesson and the data for week 5 of the course.

Download week 5 Data (~500 MB){:data-proofer-ignore=" .btn }

Plot Lines by Attribute Value

To plot vector data with the color of each objected determined by it's associated attribute values, the attribute values must be class = `factor`. A **factor** is similar to a category - you can group vector objects by a particular category value - for example you can group all lines of `TYPE=footpath`. However, in `R`, a factor can also have a determined *order*.

By default, `R` will import spatial object attributes as `factors`.

Data Tip: If our data attribute values are not read in as factors, we can convert the categorical attribute values using `as.factor()`. {: .notice}

```
# load libraries
library(raster)
library(rgdal)
options(stringsAsFactors = F)
```

Next, import and explore the data.

```
# import roads
sjer_roads <- readOGR("data/week5/california/madera-county-roads",
                      "tl_2013_06039_roads")
## OGR data source with driver: ESRI Shapefile
## Source: "data/week5/california/madera-county-roads", layer: "tl_2013_06039_roads"
## with 9640 features
## It has 4 fields
# view the original class of the TYPE column
class(sjer_roads$RTTYP)
## [1] "character"
unique(sjer_roads$RTTYP)
## [1] "M" NA  "S" "C"
```

It looks like we have some missing values in our road types. We want to plot all road types even those that are NA. Let's change the roads with an `RTTYP` attribute of NA to "unknown".

Following, we can convert the road attribute to a factor.

```
# set all NA values to "unknown" so they still plot
sjer_roads$RTTYP[is.na(sjer_roads$RTTYP)] <- "Unknown"
unique(sjer_roads$RTTYP)
```

```
## [1] "M"       "Unknown" "S"       "C"

# view levels or categories - note that there are no categories yet in our data!
# the attributes are just read as a list of character elements.
levels(sjer_roads$RTTYP)
## NULL

# Convert the TYPE attribute into a factor
# Only do this IF the data do not import as a factor!
sjer_roads$RTTYP <- as.factor(sjer_roads$RTTYP)
class(sjer_roads$RTTYP)
## [1] "factor"
levels(sjer_roads$RTTYP)
## [1] "C"       "M"       "S"       "Unknown"

# how many features are in each category or level?
summary(sjer_roads$RTTYP)
##       C       M       S Unknown
##      10    4456      25    5149
```

When we use `plot()`, we can specify the colors to use for each attribute using the `col=` element. To ensure that `R` renders each feature by it's associated factor / attribute value, we need to create a `vector` or colors - one for each feature, according to its associated attribute value / `factor` value.

To create this vector we can use the following syntax:

```
c("colorOne", "colorTwo", "colorThree")[object$factor]
```

Note in the above example we have

1. A vector of colors - one for each factor value (unique attribute value)
2. The attribute itself (`[object$factor]`) of class `factor`.

Let's give this a try.

```
# count the number of unique values or levels
length(levels(sjer_roads$RTTYP))
## [1] 4

# create a color palette of 4 colors - one for each factor level
roadPalette <- c("blue", "green", "grey", "purple")
roadPalette
## [1] "blue"   "green"  "grey"   "purple"
# create a vector of colors - one for each feature in our vector object
# according to its attribute value
roadColors <- c("blue", "green", "grey", "purple")[sjer_roads$RTTYP]
head(roadColors)
## [1] "green" "green" "green" "green" "green" "green"

# plot the lines data, apply a diff color to each factor level)
plot(sjer_roads,
     col=roadColors,
     lwd=2,
     main="Madera County Roads")
```

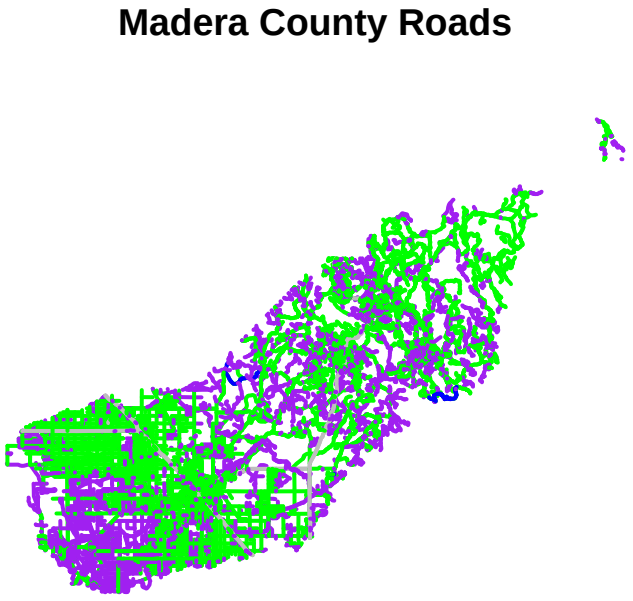

Figure 1: Adjust colors on map by creating a palette.

Adjust Line Width

We can also adjust the width of our plot lines using `lwd`. We can set all lines to be thicker or thinner using `lwd=`.

```
# make all lines thicker
plot(sjer_roads,
     col=roadColors,
     main="Madera County Roads\n All Lines Thickness=6",
     lwd=6)
```

Adjust Line Width by Attribute

If we want a unique line width for each factor level or attribute category in our spatial object, we can use the same syntax that we used for colors, above.

`lwd=c("widthOne", "widthTwo","widthThree")[object$factor]`

Note that this requires the attribute to be of class `factor`. Let's give it a try.

```
class(sjer_roads$RTTYP)
## [1] "factor"
levels(sjer_roads$RTTYP)
## [1] "C"       "M"       "S"       "Unknown"
# create vector of line widths
lineWidths <- (c(1, 2, 3, 4))[sjer_roads$RTTYP]
# adjust line width by level
# in this case, boardwalk (the first level) is the widest.
plot(sjer_roads,
     col=roadColors,
     main="Madera County Roads \n Line width varies by TYPE Attribute Value",
     lwd=lineWidths)
```

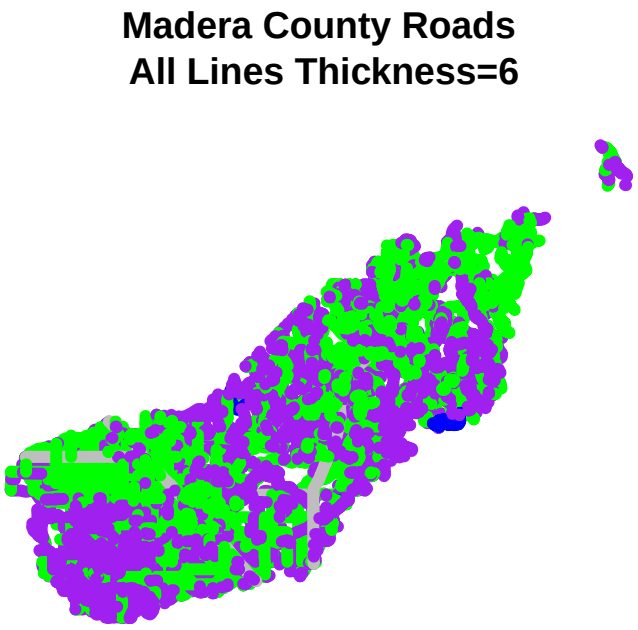

Figure 2: map of madera roads

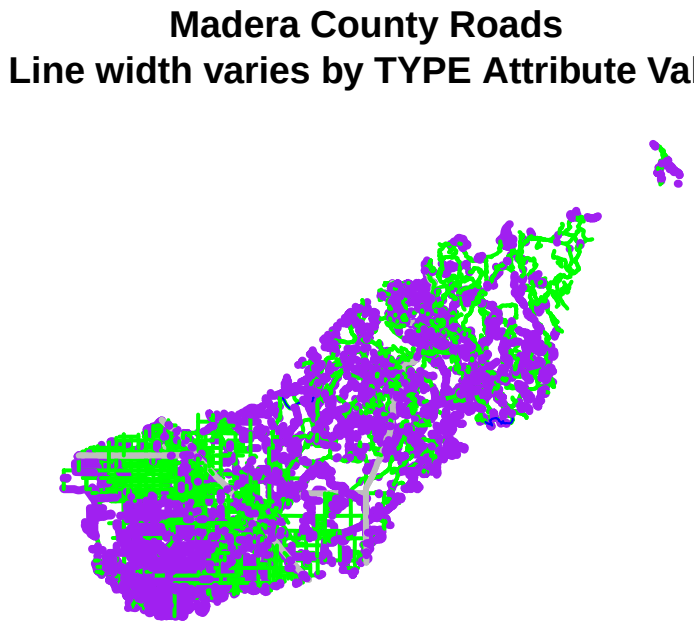

Figure 3: Map with legend that shows unique line widths.

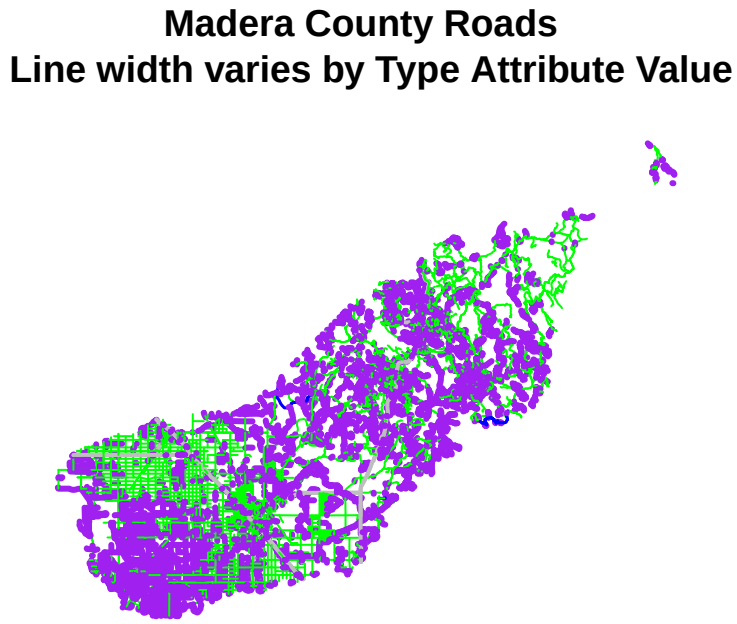

Figure 4: roads map modified

Optional challenge: Plot line width by attribute

We can customize the width of each line, according to specific attribute value, too. To do this, we create a vector of line width values, and map that vector to the factor levels - using the same syntax that we used above for colors. HINT: `lwd=(vector of line width thicknesses)[spatialObject$factorAttribute]`

Create a plot of roads using the following line thicknesses:

1. **unknown** lwd = 3
2. **M** lwd = 1
3. **S** lwd = 2
4. **C** lwd = 1.5

Data Tip: Given we have a factor with 4 levels, we can create an vector of numbers, each of which specifies the thickness of each feature in our `SpatialLinesDataFrame` by factor level (category): `c(6,4,1,2)[sjer_roads$RTTYP]` {: .notice}

Add Plot Legend

We can add a legend to our plot too. When we add a legend, we use the following elements to specify labels and colors:

- **location**: we can specify an x and Y location of the plot Or generally specify the location e.g. 'bottomright' keyword. We could also use `top`, `topright`, etc.
- `levels(objectName$attributeName)`: Label the **legend elements** using the categories of `levels` in an attribute (e.g., levels(sjer_roads$RTTYP) means use the levels C, S, footpath, etc).
- `fill=`: apply unique **colors** to the boxes in our legend. `palette()` is the default set of colors that `R` applies to all plots.

Let's add a legend to our plot.

```
# add legend to plot
plot(sjer_roads,
```

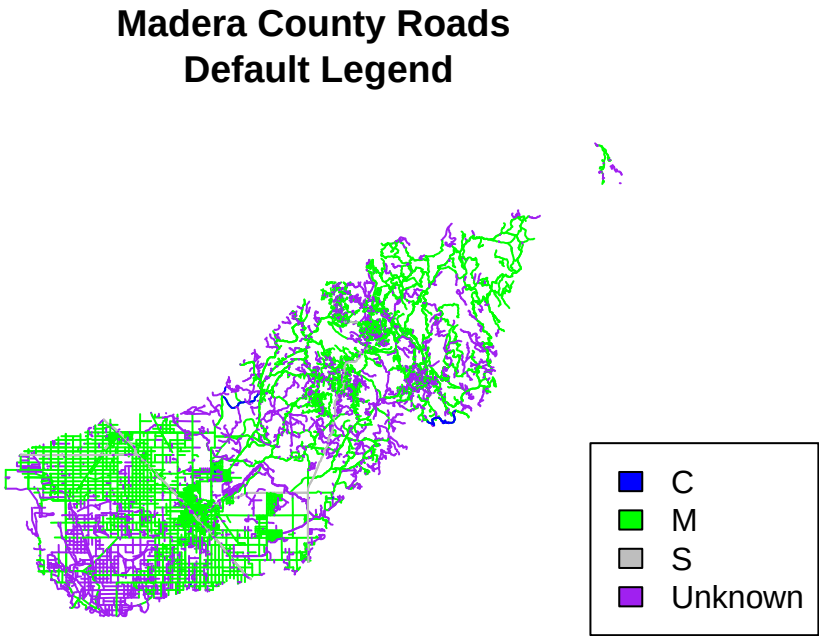

Figure 5: SJER roads map with custom legend.

```
     col=roadColors,
     main="Madera County Roads\n Default Legend")

# we can use the color object that we created above to color the legend objects
roadPalette
## [1] "blue"   "green"  "grey"   "purple"

# add a legend to our map
legend("bottomright",   # location of legend
      legend=levels(sjer_roads$RTTYP), # categories or elements to render in
             # the legend
      fill=roadPalette) # color palette to use to fill objects in legend.
```

We can tweak the appearance of our legend too.

- `bty=n`: turn off the legend BORDER
- `cex`: change the font size

Let's try it out.

```
# adjust legend
plot(sjer_roads,
     col=roadColors,
     main="Madera County Roads \n Modified Legend - smaller font and no border")
# add a legend to our map
legend("bottomright",
       legend=levels(sjer_roads$RTTYP),
       fill=roadPalette,
       bty="n", # turn off the legend border
       cex=.8) # decrease the font / legend size
```

We can modify the colors used to plot our lines by creating a new color vector, directly in the plot code too

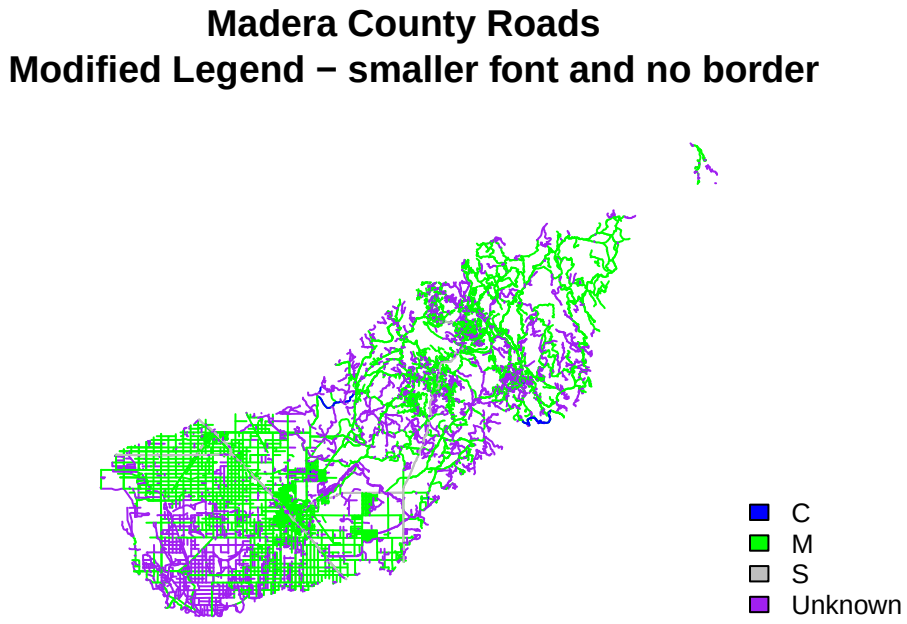

Figure 6: modified custom legend

rather than creating a separate object.

`col=(newColors)[sjer_roads$RTTYP]`

Let's try it!

```
# manually set the colors for the plot!
newColors <- c("springgreen", "blue", "magenta", "orange")
newColors
## [1] "springgreen" "blue"        "magenta"     "orange"

# plot using new colors
plot(sjer_roads,
     col=(newColors)[sjer_roads$RTTYP],
     main="Madera County Roads \n Pretty Colors")

# add a legend to our map
legend("bottomright",
       levels(sjer_roads$RTTYP),
       fill=newColors,
       bty="n", cex=.8)
```

Data Tip: You can modify the defaul R color palette using the palette method. For example `palette(rainbow(6))` or `palette(terrain.colors(6))`. You can reset the palette colors using `palette("default")`! {: .notice}

Plot Lines by Attribute

Create a plot that emphasizes only roads designated as C or S (County or State). To emphasize these types of roads, make the lines that are C or S, THICKER than the other lines. NOTE: this attribute information is located in the `sjer_roads$RTTYP` attribute.

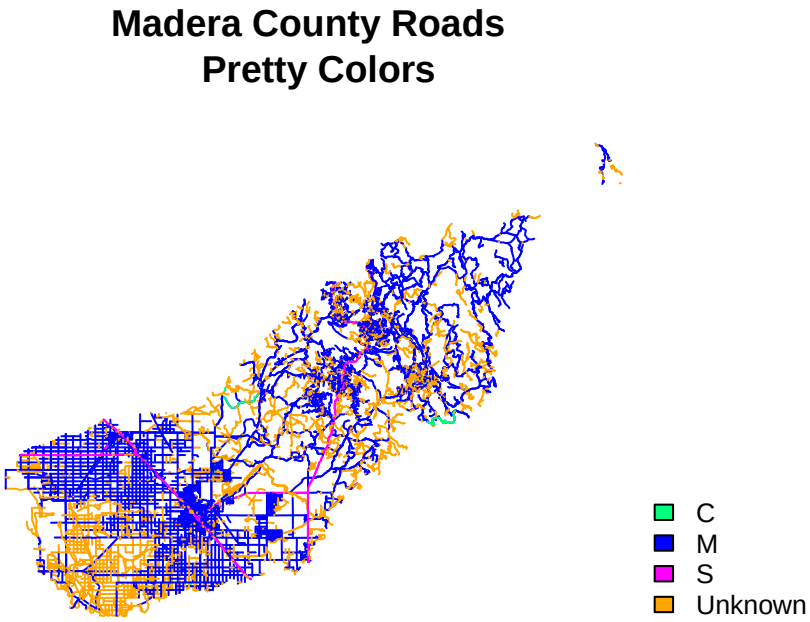

Figure 7: adjust colors

Be sure to add a title and legend to your map! You might consider a color palette that has all County and State roads displayed in a bright color. All other lines can be grey.

```
# view levels
levels(sjer_roads$RTTYP)
## [1] "C"       "M"       "S"       "Unknown"
# make sure the attribute is of class "factor"
class(sjer_roads$RTTYP)
## [1] "factor"

# convert to factor if necessary
sjer_roads$RTTYP <- as.factor(sjer_roads$RTTYP)
levels(sjer_roads$RTTYP)
## [1] "C"       "M"       "S"       "Unknown"

# count factor levels
length(levels(sjer_roads$RTTYP))
## [1] 4
# set colors so only the allowed roads are magenta
# note there are 3 levels so we need 3 colors
challengeColors <- c("magenta","grey","magenta","grey")
challengeColors
## [1] "magenta" "grey"    "magenta" "grey"

# plot using new colors
plot(sjer_roads,
     col=(challengeColors)[sjer_roads$RTTYP],
     lwd=c(4,1,1,1)[sjer_roads$RTTYP],
     main="SJER Roads")

# add a legend to our map
legend("bottomright",
```

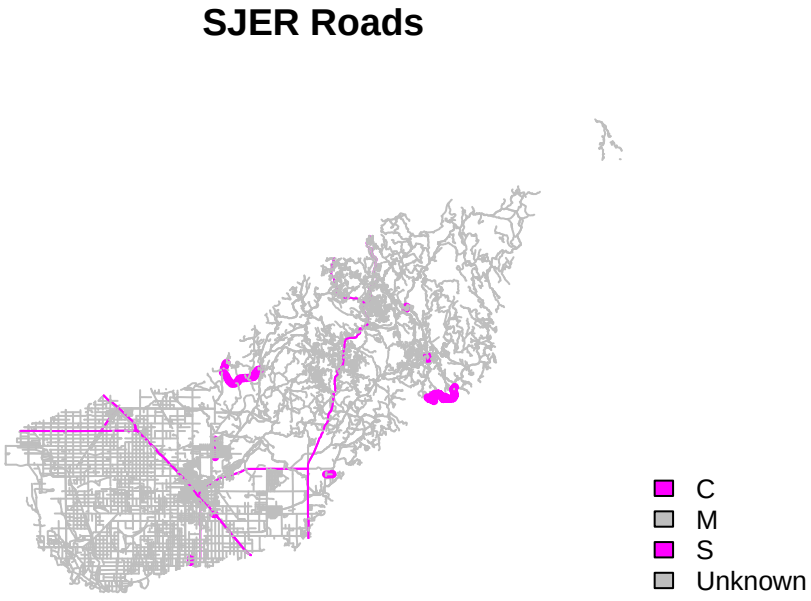

Figure 8: emphasize some attributes

```
       levels(sjer_roads$RTTYP),
       fill=challengeColors,
       bty="n", # turn off border
       cex=.8) # adjust font size
```

Finall, let's adjust the legend. We want the legend SYMBOLS to represent the actual symbology in the map - which contains lines, not polygons.

```
# plot using new colors
plot(sjer_roads,
     col=(challengeColors)[sjer_roads$RTTYP],
     lwd=c(4,1,2,1)[sjer_roads$RTTYP], # color each line in the map by attribute
     main="Madera County Roads\n County and State recognized roads")

# add a legend to our map
legend("bottomright",
       levels(sjer_roads$RTTYP),
       lty=c(1,1,1,1), # tell are which objects to be drawn as a line in the legend.
       lwd=c(4,1,2,1),  # set the WIDTH of each legend line
       col=challengeColors, # set the color of each legend line
       bty="n", # turn off border
       cex=.8) # adjust font size
```

Adding point and lines to a legend

The last step in customizing a legend is adding different types of symbols to the plot. In the example above, we just added lines. But what if we wanted to add some POINTS too? We will do that next.

In the data below, we've create a custom legend where each symbol type and color is defined using a vector. We have 3 levels: grass, soil and trees. Thus we need to define 3 symbols and 3 colors for our legend and our plot.

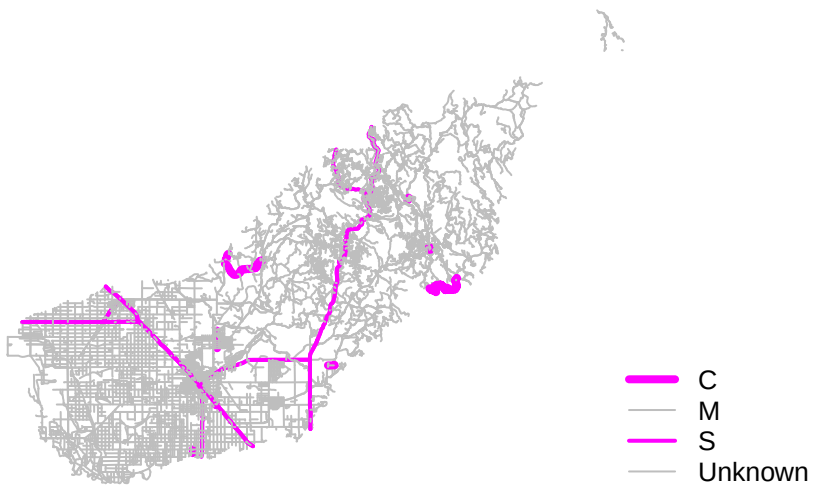

Figure 9: Custom legend with lines

```
pch=c(8,18,8)
```

```
plot_colors <- c("chartreuse4", "burlywood4", "darkgreen")
```

```
# import points layer
sjer_plots <- readOGR("data/week5/california/SJER/vector_data",
                      "SJER_plot_centroids")
## OGR data source with driver: ESRI Shapefile
## Source: "data/week5/california/SJER/vector_data", layer: "SJER_plot_centroids"
## with 18 features
## It has 5 fields

sjer_plots$plot_type <- as.factor(sjer_plots$plot_type)
levels(sjer_plots$plot_type)
## [1] "grass" "soil"  "trees"
# grass, soil trees
plot_colors <- c("chartreuse4", "burlywood4", "darkgreen")

# plot using new colors
plot(sjer_plots,
     col=(plot_colors)[sjer_plots$plot_type],
     pch=8,
     main="Madera County Roads\n County and State recognized roads")


# add a legend to our map
legend("bottomright",
       legend = levels(sjer_plots$plot_type),
       pch=c(8,18,8),  # set the WIDTH of each legend line
       col=plot_colors, # set the color of each legend line
       bty="n", # turn off border
       cex=.9) # adjust legend font size
```

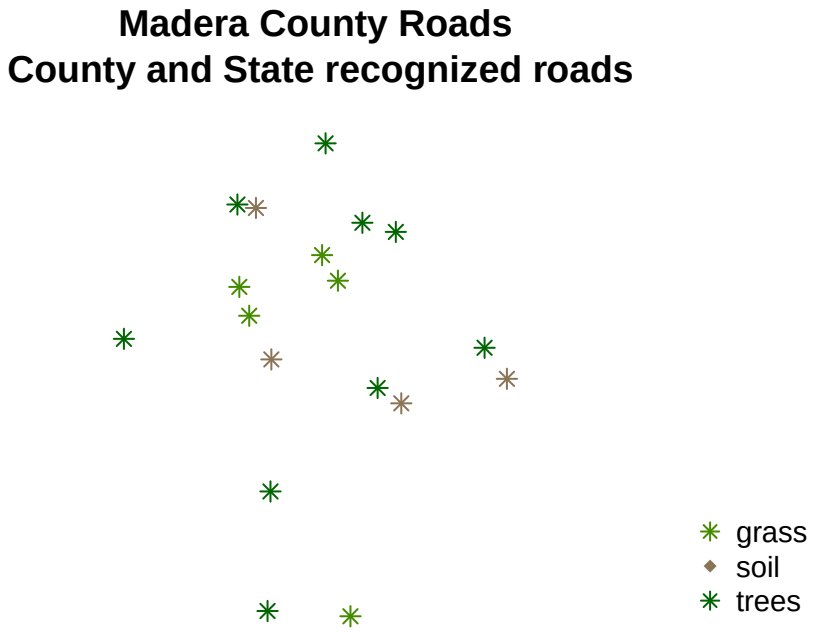

Figure 10: plot legend with points and lines

Next, let's try to plot our roads on top of the plot locations. Then let's create a custom legend that contains both lines and points. NOTE: in this example i've fixed the projection for the roads layer and cropped it! You will have to do the same before this code will work.

```
## OGR data source with driver: ESRI Shapefile
## Source: "data/week5/california/SJER/vector_data", layer: "SJER_crop"
## with 1 features
## It has 1 fields
```

When we create a legend, we will have to add the labels for both the points layer and the lines layer.

```
c(levels(sjer_plots$plot_type), levels(sjer_roads$RTTYP))
```

```
# view all elements in legend
c(levels(sjer_plots$plot_type), levels(sjer_roads$RTTYP))
## [1] "grass"   "soil"    "trees"   "C"       "M"       "S"       "Unknown"
```

```
# plot using new colors
plot(sjer_plots,
     col=(plot_colors)[sjer_plots$plot_type],
     pch=8,
     main="Madera County Roads and plot locations")

# plot using new colors
plot(sjer_roads_utm,
     col=(challengeColors)[sjer_plots$plot_type],
     pch=8,
     add=T)

# add a legend to our map
legend("bottomright",
       legend = c(levels(sjer_plots$plot_type), levels(sjer_roads$RTTYP)),
       pch=c(8,18,8),  # set the WIDTH of each legend line
```

Madera County Roads and plot locations

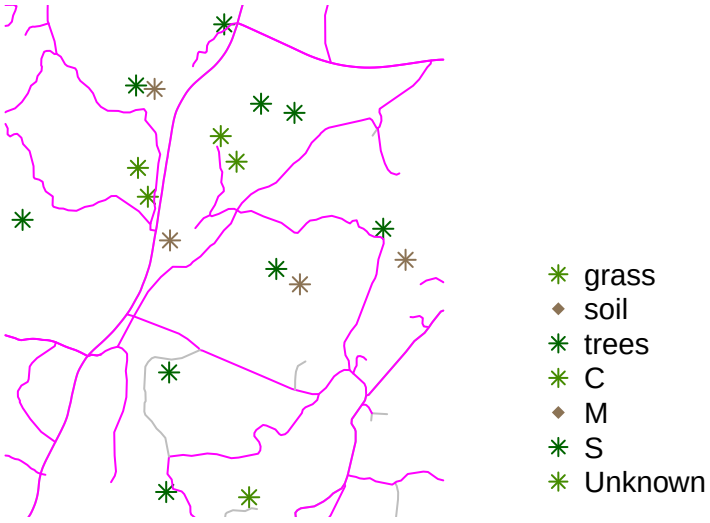

Figure 11: final plot custom legend.

```
       col=plot_colors, # set the color of each legend line
       bty="n", # turn off border
       cex=.9) # adjust legend font size
```

Next we have to tell R, which symbols are lines and which are point symbols. We can do that using the lty argument. We have 3 unique point symbols and 4 unique line symbols. We can include a NA for each element that should not be a line in the lty argument:

`lty=c(NA,NA, NA, 1, 1, 1, 1)`

And we include a NA value for each element that should not be a symbol in the pch argument:

pch=c(8,18,8, NA, NA, NA, NA)

```
# plot using new colors
plot(sjer_plots,
     col=(plot_colors)[sjer_plots$plot_type],
     pch=8,
     main="Madera County Roads and plot locations")

# plot using new colors
plot(sjer_roads_utm,
     col=(challengeColors)[sjer_plots$plot_type],
     pch=8,
     add=T)

# add a legend to our map
legend("bottomright",
       legend = c(levels(sjer_plots$plot_type), levels(sjer_roads$RTTYP)),
       pch=c(8,18,8, NA, NA, NA, NA),  # set the symbol for each point
       lty=c(NA,NA, NA, 1, 1, 1, 1),
       col=c(plot_colors, challengeColors), # set the color of each legend line
       bty="n", # turn off border
```

Madera County Roads and plot locations

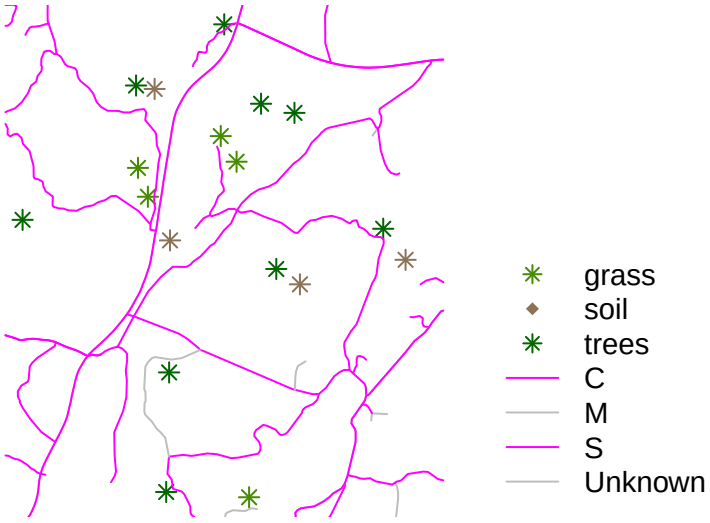

Figure 12: Plot with points and lines customized.

```
       cex=.9) # adjust legend font size
```

Force the legend to plot next to your plot

Refining the look of your plots takes a bit of patience in R, but it can be done! Play with the code below to see if you can make your legend plot NEXT TO rather than on top of your plot.

The steps are:

1. Place your legend on the OUTSIDE of the plot extent by grabbing the `xmax` and `ymax` values from one of the objects that you are plotting's `extent()`. This allows you to be precise in your legend placement.
2. Set the `xpd=T` argument in your legend to enforce plotting outside of the plot extent and
3. OPTIONAL: adjust the plot **PAR**ameters using `par()`. You can set the **mar**gin of your plot using `mar=`. This provides extra space on the right (if you'd like your legend to plot on the right) for your legend and avoids things being "chopped off". Provide the `mar=` argument in the format: `c(bottom, left, top, right)`. The code below is telling r to add 7 units of padding on the RIGHT hand side of our plot. The default units are INCHES.

IMPORTANT: be cautious with margins. Sometimes they can cause problems when you knit - particularly if they are too large.

Let's give this a try. First, we grab the northwest corner location x and y. We will use this to place our legend.

```
# figure out where the upper RIGHT hand corner of our plot extent is
the_plot_extent <- extent(sjer_aoi)

# grab the upper right hand corner coordinates
furthest_pt_east <- the_plot_extent@xmax
furthest_pt_north <- the_plot_extent@ymax
# view values
furthest_pt_east
```

Madera County Roads and plot locations

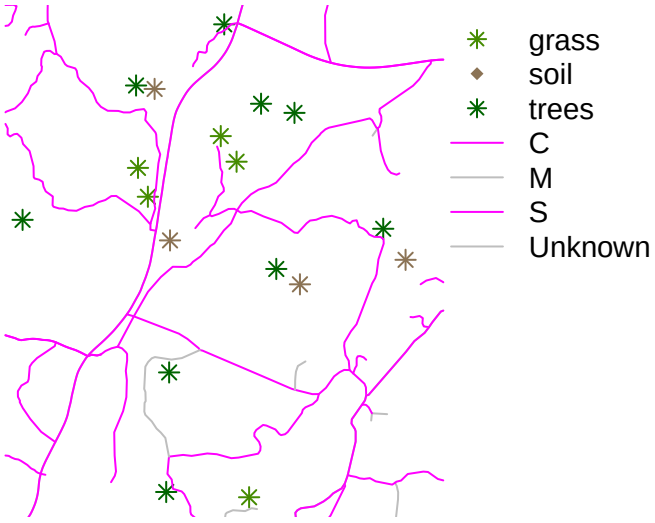

Figure 13: plot with fixed legend

```
## [1] 258867.4
furthest_pt_north
## [1] 4112362

# plot using new colors
plot(sjer_plots,
     col=(plot_colors)[sjer_plots$plot_type],
     pch=8,
     main="Madera County Roads and plot locations")

# plot using new colors
plot(sjer_roads_utm,
     col=(challengeColors)[sjer_plots$plot_type],
     pch=8,
     add=T)

# add a legend to our map
legend(x=furthest_pt_east, y=furthest_pt_north,
       legend = c(levels(sjer_plots$plot_type), levels(sjer_roads$RTTYP)),
       pch=c(8,18,8, NA, NA, NA, NA),  # set the symbol for each point
       lty=c(NA,NA, NA, 1, 1, 1, 1) ,
       col=c(plot_colors, challengeColors), # set the color of each legend line
       bty="n", # turn off border
       cex=.9, # adjust legend font size
       xpd=T) # force the legend to plot outside of your extent
```

Let's use the margin parameter to clean things up. Also notice i'm using the AOI extent layer to create a "box" around my plot. Now things are starting to look much cleaner!

I've also added some "fake" legend elements to create subheadings like we might add to a map legend in QGIS or ArcGIS.

Madera County Roads and plot locations

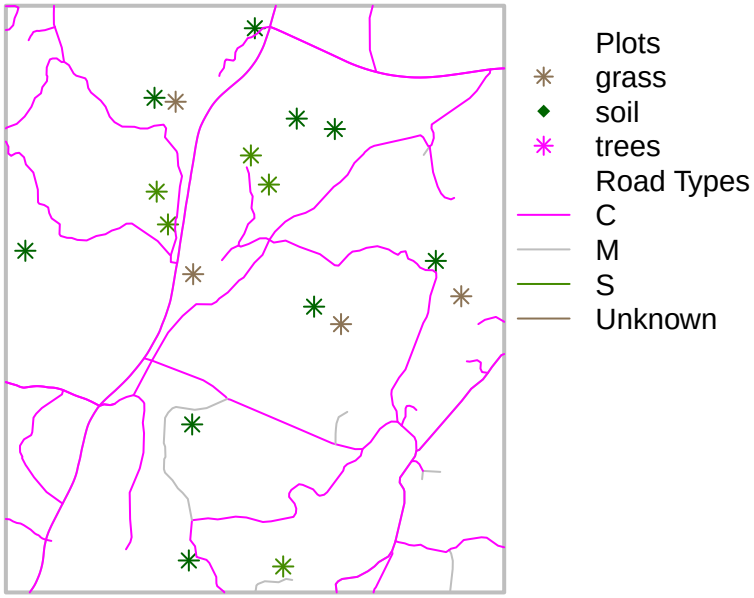

Figure 14: final legend with points and lines customized 2ß.

```
legend = c("Plots", levels(sjer_plots$plot_type), "Road Types", levels(sjer_roads$RTTYP))
```

```
# adjust margin to make room for the legend
par(mar=c(2, 2, 4, 7))
# plot using new colors
plot(sjer_aoi,
     border="grey",
     lwd=2,
     main="Madera County Roads and plot locations")
plot(sjer_plots,
     col=(plot_colors)[sjer_plots$plot_type],
     add=T,
     pch=8)
# plot using new colors
plot(sjer_roads_utm,
     col=(challengeColors)[sjer_plots$plot_type],
     pch=8,
     add=T)

# add a legend to our map
legend(x=(furthest_pt_east+50), y=(furthest_pt_north-15),
       legend = c("Plots", levels(sjer_plots$plot_type), "Road Types", levels(sjer_roads$RTTYP)),
       pch=c(NA, 8, 18, 8, NA, NA, NA, NA, NA),  # set the symbol for each point
       lty=c(NA,NA,NA, NA, NA, 1, 1, 1, 1),
       col=c(plot_colors, challengeColors), # set the color of each legend line
       bty="n", # turn off border
       cex=.9, # adjust legend font size
       xpd=T)
```

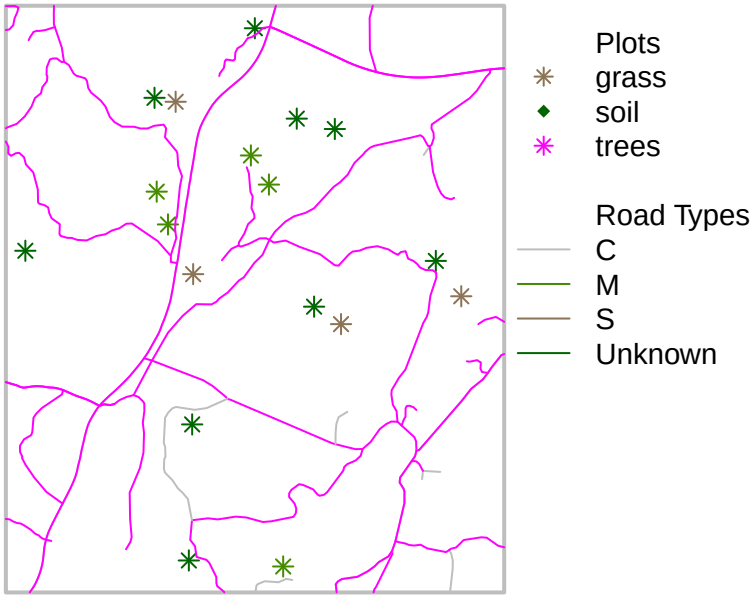

Figure 15: final legend with points and lines customized.

Now, if you want to move the legend out a bit further, what would you do?

BONUS!

Any idea how I added a space to the legend below to create "sections"?

```
# important: once you are done, reset par which resets plot margins
# otherwise margins will carry over to your next plot and cause problems!
dev.off()
```