seaborn.barplot

```
seaborn.barplot(data=None, *, x=None, y=None, hue=None, order=None, hue_order=None,
estimator='mean', errorbar=('ci', 95), n_boot=1000, units=None, seed=None, orient=None,
color=None, palette=None, saturation=0.75, width=0.8, errcolor='.26', errwidth=None,
capsize=None, dodge=True, ci='deprecated', ax=None, **kwargs)
```

Show point estimates and errors as rectangular bars.

A bar plot represents an estimate of central tendency for a numeric variable with the height of each rectangle and provides some indication of the uncertainty around that estimate using error bars. Bar plots include 0 in the quantitative axis range, and they are a good choice when 0 is a meaningful value for the quantitative variable, and you want to make comparisons against it.

For datasets where 0 is not a meaningful value, a point plot will allow you to focus on differences between levels of one or more categorical variables.

It is also important to keep in mind that a bar plot shows only the mean (or other estimator) value, but in many cases it may be more informative to show the distribution of values at each level of the categorical variables. In that case, other approaches such as a box or violin plot may be more appropriate.

> **Note**
>
> This function always treats one of the variables as categorical and draws data at ordinal positions (0, 1, … n) on the relevant axis, even when the data has a numeric or date type.

See the tutorial for more information.

Parameters:

data : *DataFrame, array, or list of arrays, optional*

Dataset for plotting. If `x` and `y` are absent, this is interpreted as wide-form. Otherwise it is expected to be long-form.

x, y, hue : *names of variables in `data` or vector data, optional*

Inputs for plotting long-form data. See examples for interpretation.

order, hue_order : *lists of strings, optional*

Order to plot the categorical levels in; otherwise the levels are inferred from the data objects.

estimator : *string or callable that maps vector -> scalar, optional*

Statistical function to estimate within each categorical bin.

errorbar : *string, (string, number) tuple, callable or None*

Name of errorbar method (either “ci”, “pi”, “se”, or “sd”), or a tuple with a method name and a level parameter, or a function that maps from a vector to a (min, max) interval, or None to hide errorbar.

n_boot : *int, optional*

Number of bootstrap samples used to compute confidence intervals.

units : *name of variable in `data` or vector data, optional*

Identifier of sampling units, which will be used to perform a multilevel bootstrap and account for repeated measures design.

seed : *int, numpy.random.Generator, or numpy.random.RandomState, optional*

Seed or random number generator for reproducible bootstrapping.

orient : *“v” | “h”, optional*

Orientation of the plot (vertical or horizontal). This is usually inferred based on the type of the input variables, but it can be used to resolve ambiguity when both `x` and `y` are numeric or when plotting wide-form data.

color : *matplotlib color, optional*

Single color for the elements in the plot.

palette : *palette name, list, or dict*

Colors to use for the different levels of the `hue` variable. Should be something that can be interpreted by `color_palette()`, or a dictionary mapping hue levels to matplotlib colors.

saturation : *float, optional*

Proportion of the original saturation to draw colors at. Large patches often look better with slightly desaturated colors, but set this to `1` if you want the plot colors to perfectly match the input color.

width : *float, optional*

Width of a full element when not using hue nesting, or width of all the elements for one level of the major grouping variable.

errcolor : *matplotlib color*

Color used for the error bar lines.

errwidth : *float, optional*

Thickness of error bar lines (and caps).

capsize : *float, optional*

Width of the “caps” on error bars.

dodge : *bool, optional*

When hue nesting is used, whether elements should be shifted along the categorical axis.

ax : *matplotlib Axes, optional*

Axes object to draw the plot onto, otherwise uses the current Axes.

kwargs : *key, value mappings*

Other keyword arguments are passed through to `matplotlib.axes.Axes.bar()`.

Returns:

ax : *matplotlib Axes*

Returns the Axes object with the plot drawn onto it.

See also

`countplot`
Show the counts of observations in each categorical bin.

`pointplot`
Show point estimates and confidence intervals using scatterplot glyphs.

`catplot`

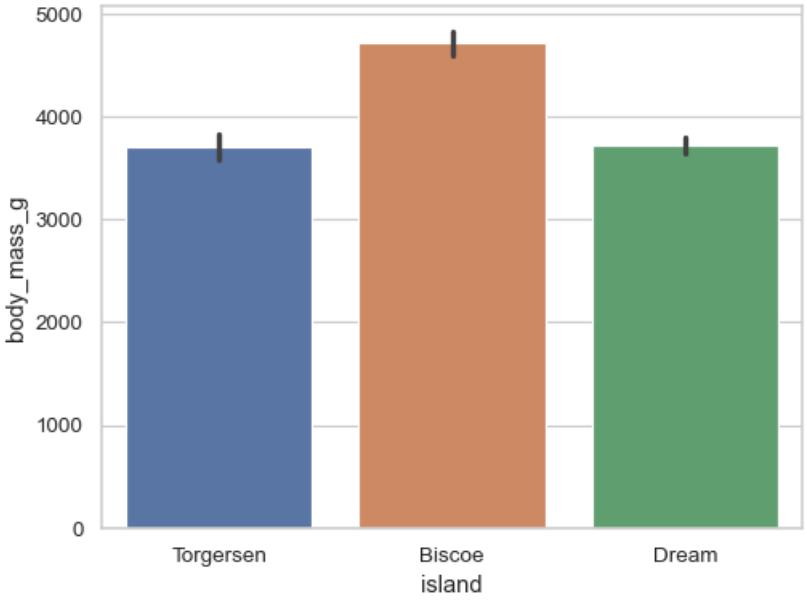
`catplot`

Combine a categorical plot with a `FacetGrid`.

Examples

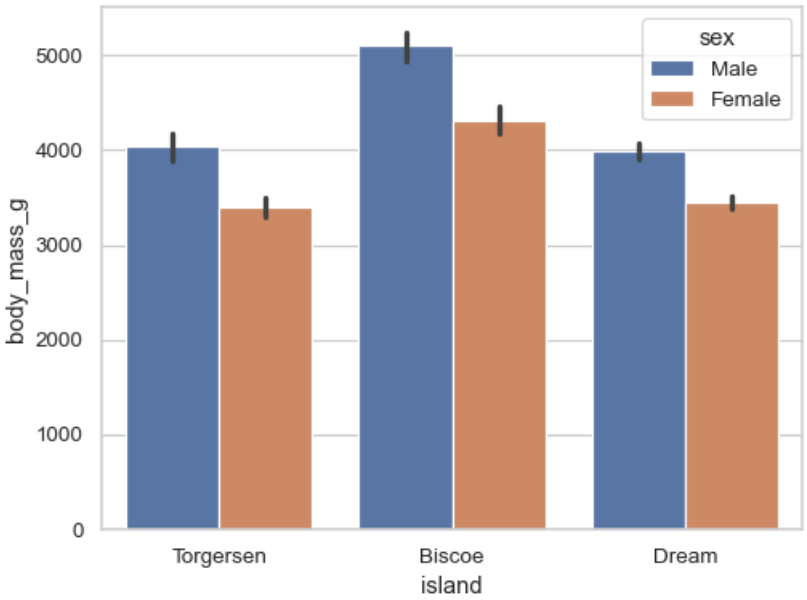
Group by a categorical varaible and plot aggregated values, with confidence intervals:

```
df = sns.load_dataset("penguins")
sns.barplot(data=df, x="island", y="body_mass_g")
```



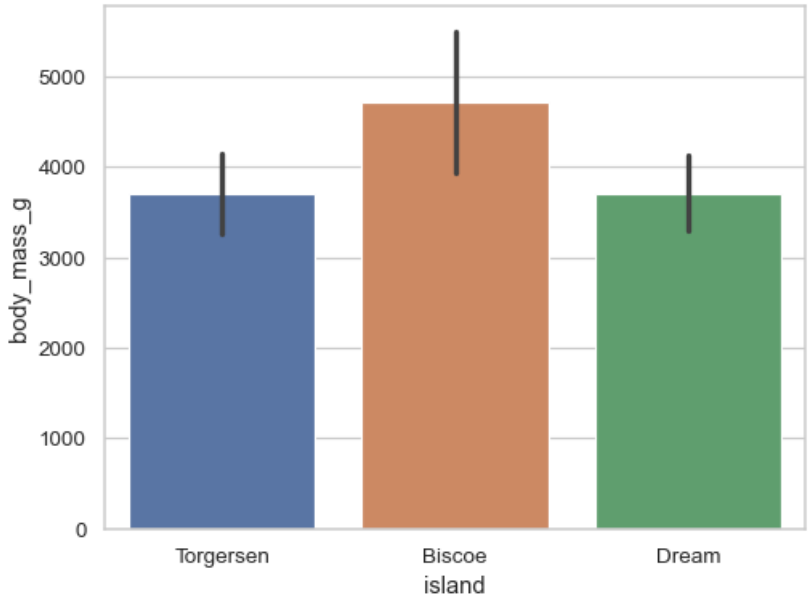
Add a second layer of grouping:

```
sns.barplot(data=df, x="island", y="body_mass_g", hue="sex")
```



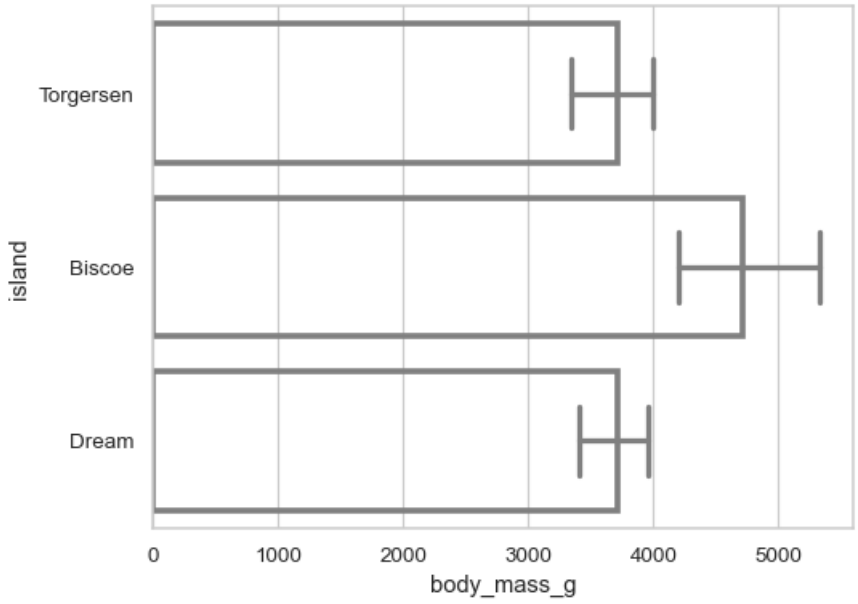
Use the error bars to show the standard deviation rather than a confidence interval:

```
sns.barplot(data=df, x="island", y="body_mass_g", errorbar="sd")
```



Customize the appearance of the plot:

```
sns.barplot(
    data=df, x="body_mass_g", y="island",
    errorbar=("pi", 50), capsize=.4, errcolor=".5",
    linewidth=3, edgecolor=".5", facecolor=(0, 0, 0, 0),
)
```



seaborn.heatmap

`seaborn.heatmap(data, *, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs)`

Plot rectangular data as a color-encoded matrix.

This is an Axes-level function and will draw the heatmap into the currently-active Axes if none is provided to the `ax` argument. Part of this Axes space will be taken and used to plot a colormap, unless `cbar` is False or a separate Axes is provided to `cbar_ax`.

Parameters:

data : *rectangular dataset*

2D dataset that can be coerced into an ndarray. If a Pandas DataFrame is provided, the index/column information will be used to label the columns and rows.

vmin, vmax : *floats, optional*

Values to anchor the colormap, otherwise they are inferred from the data and other keyword arguments.

cmap : *matplotlib colormap name or object, or list of colors, optional*

The mapping from data values to color space. If not provided, the default will depend on whether `center` is set.

center : *float, optional*

The value at which to center the colormap when plotting divergent data. Using this parameter will change the default `cmap` if none is specified.

robust : *bool, optional*

If True and `vmin` or `vmax` are absent, the colormap range is computed with robust quantiles instead of the extreme values.

annot : *bool or rectangular dataset, optional*

If True, write the data value in each cell. If an array-like with the same shape as `data`, then use this to annotate the heatmap instead of the data. Note that DataFrames will match on position, not index.

fmt : *str, optional*

String formatting code to use when adding annotations.

annot_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.axes.Axes.text()` when `annot` is True.

linewidths : *float, optional*

Width of the lines that will divide each cell.

linecolor : *color, optional*

Color of the lines that will divide each cell.

cbar : *bool, optional*

Whether to draw a colorbar.

cbar_kws : *dict of key, value mappings, optional*

Keyword arguments for `matplotlib.figure.Figure.colorbar()`.

cbar_ax : *matplotlib Axes, optional*

Axes in which to draw the colorbar, otherwise take space from the main Axes.

square : *bool, optional*

If True, set the Axes aspect to “equal” so each cell will be square-shaped.

xticklabels, yticklabels : *“auto”, bool, list-like, or int, optional*

If True, plot the column names of the dataframe. If False, don’t plot the column names. If list-like, plot these alternate labels as the xticklabels. If an integer, use the column names but plot only every n label. If “auto”, try to densely plot non-overlapping labels.

mask : *bool array or DataFrame, optional*

If passed, data will not be shown in cells where `mask` is True. Cells with missing values are automatically masked.

ax : *matplotlib Axes, optional*

Axes in which to draw the plot, otherwise use the currently-active Axes.

kwargs : *other keyword arguments*

All other keyword arguments are passed to `matplotlib.axes.Axes.pcolormesh()`.

Returns:

ax : *matplotlib Axes*

Axes object with the heatmap.

See also

`clustermap`

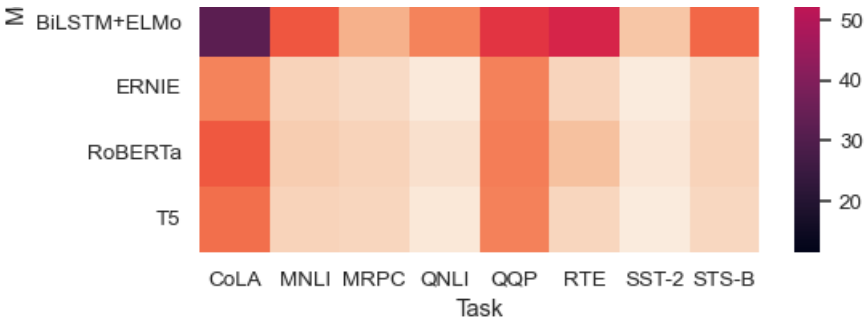
Plot a matrix using hierarchical clustering to arrange the rows and columns.

Examples

Pass a `DataFrame` to plot with indices as row/column labels:

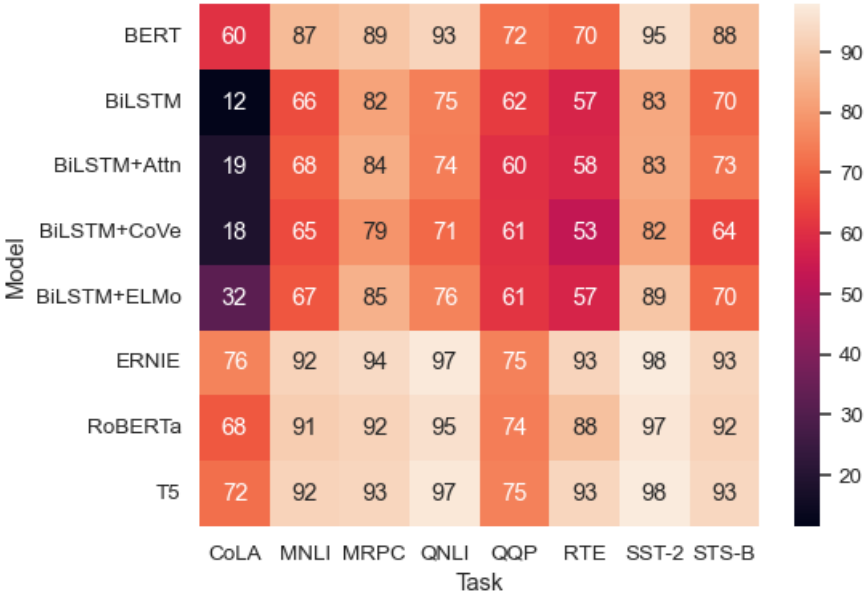
```
glue = sns.load_dataset("glue").pivot("Model", "Task", "Score")
sns.heatmap(glue)
```

```
/var/folders/qk/cdrdfhfn5g554pnb30pp4ylr0000gn/T/ipykernel_77613/2862412127.py:1: FutureWarning: I
  glue = sns.load_dataset("glue").pivot("Model", "Task", "Score")
```



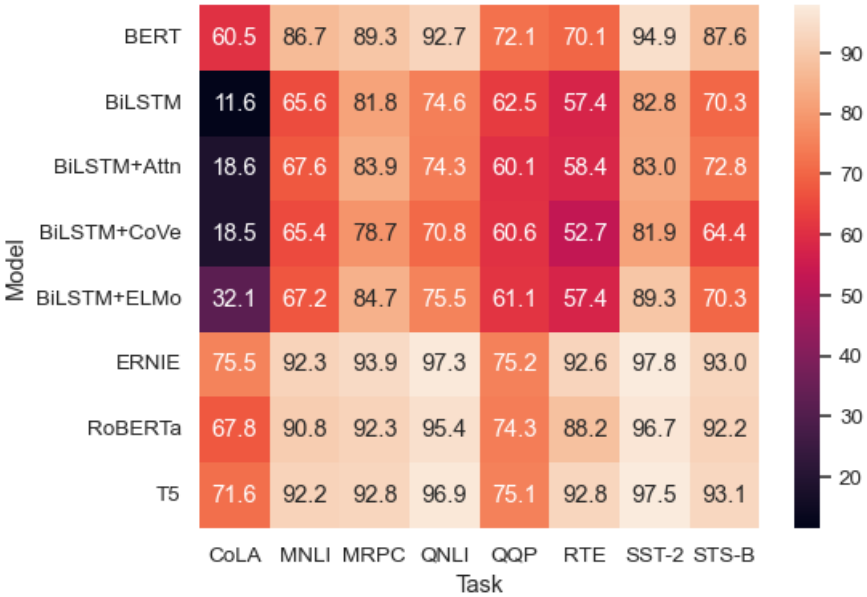
Use `annot` to represent the cell values with text:

```
sns.heatmap(glue, annot=True)
```



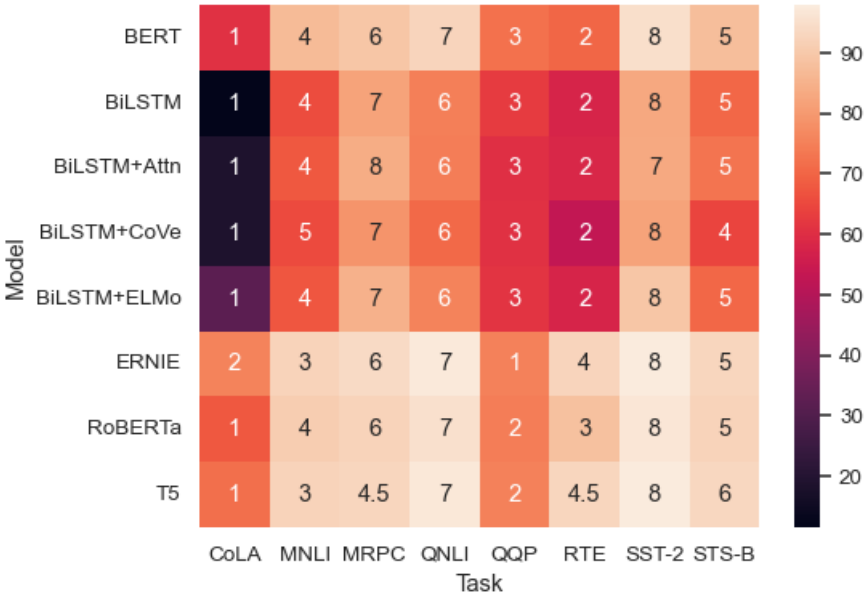
Control the annotations with a formatting string:

```
sns.heatmap(glue, annot=True, fmt=".1f")
```



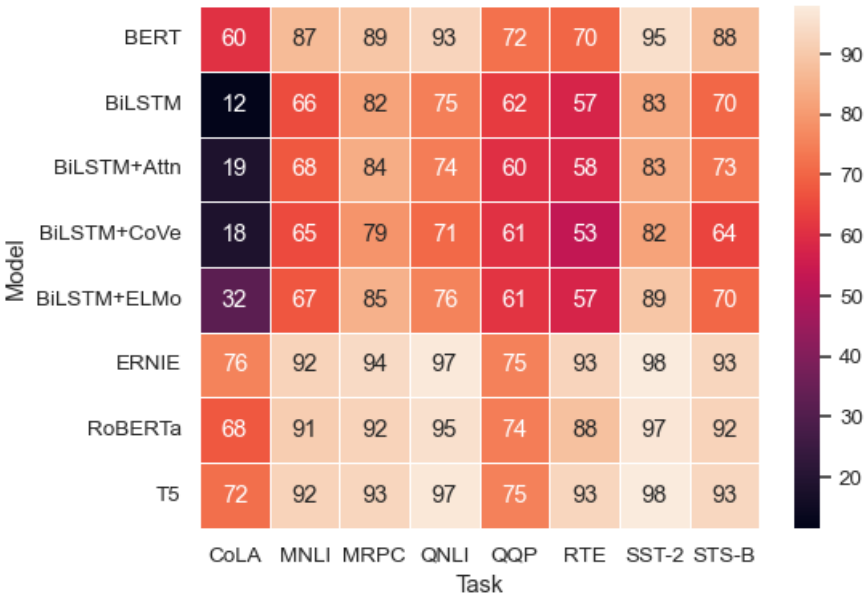
Use a separate dataframe for the annotations:

```
sns.heatmap(glue, annot=glue.rank(axis="columns"))
```



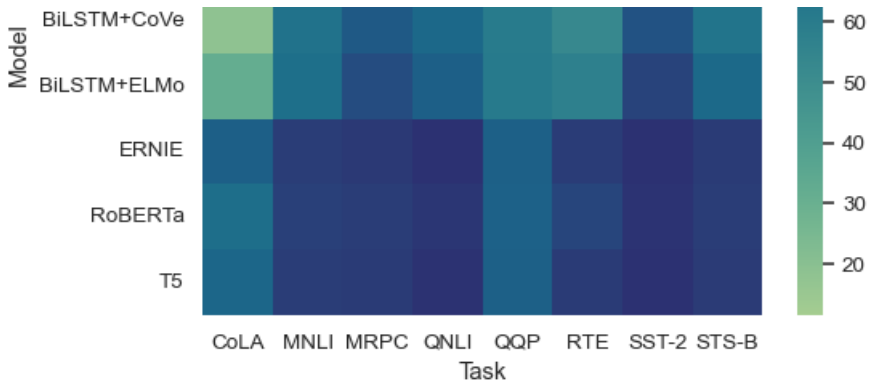
Add lines between cells:

```
sns.heatmap(glue, annot=True, linewidth=.5)
```



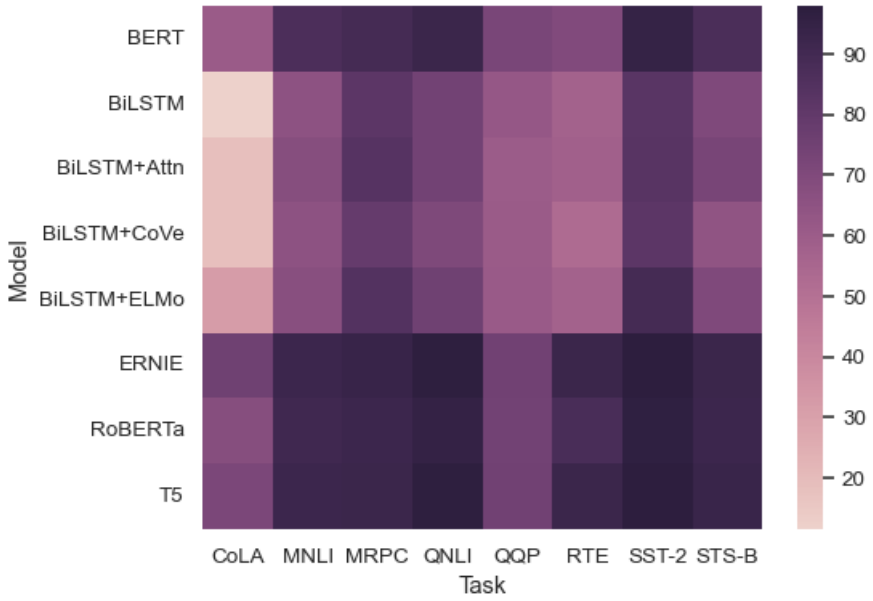
Select a different colormap by name:

```
sns.heatmap(glue, cmap="crest")
```



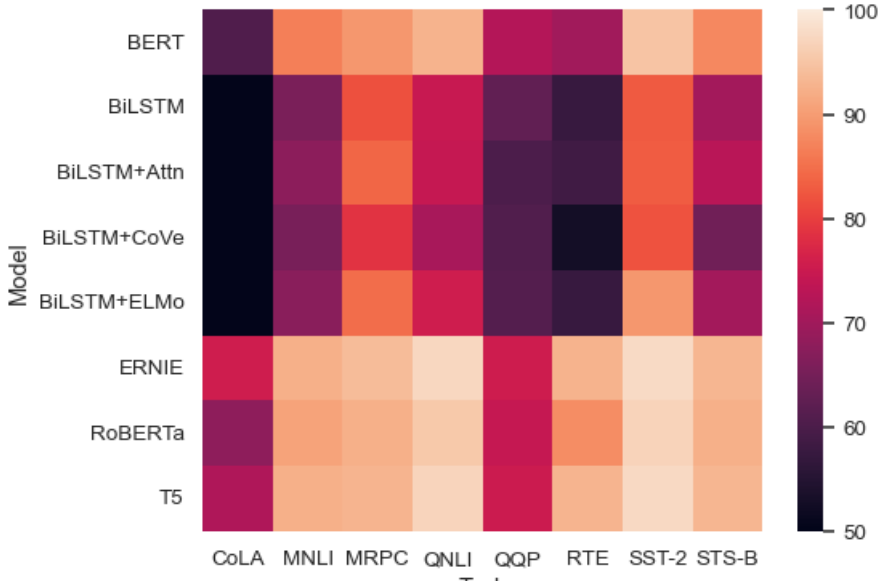
Or pass a colormap object:

```
sns.heatmap(glue, cmap=sns.cubehelix_palette(as_cmap=True))
```



Set the colormap norm (data values corresponding to minimum and maximum points):

```
sns.heatmap(glue, vmin=50, vmax=100)
```



lask

Use methods on the `matplotlib.axes.Axes` object to tweak the plot:

```
ax = sns.heatmap(glue, annot=True)
ax.set(xlabel="", ylabel="")
ax.xaxis.tick_top()
```