Announcements

HW#2 Due 9/19 at 11:59 PM
HW#1 Q3 – Error in Answer Key

Agenda Today

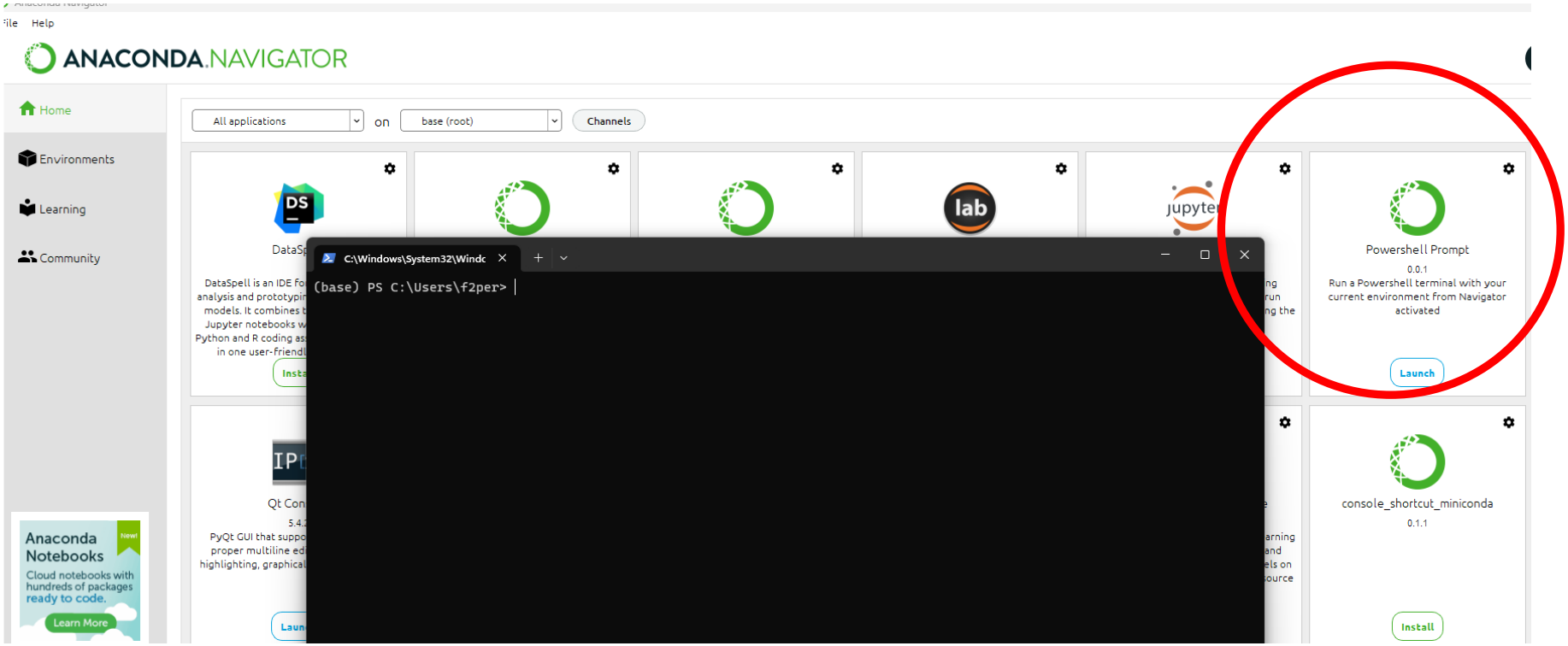
Lab 4 - Exploratory Data Analysis
Seaborn

Seaborn

4.2 Visual Distribution of Two Categorical Variables

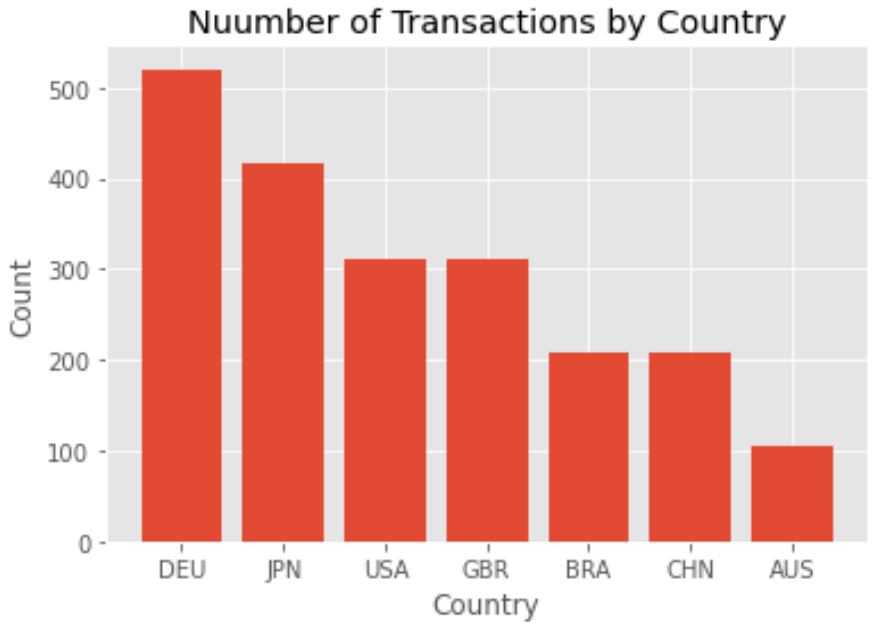
- In this section, we will utilze the `seaborn` data visualization package to create our plots
- We will also switch to the `tips` dataset that comes with `seaborn` , so there is no need to load the data from a csv file
- `seaborn` methods builds on top of `matplotlib` functionality
- We could have down the plots below using `matplotlib` but we will have to write more code
- Note that `seaborn` does the crosstabulation for the categorical variable (unlike `matplotlib`)
- Note that we could have also used the `seaborn` methods to visualize One Categorical variable

Install Squarify



```
plt.bar(country_sales.index, country_sales)
plt.title('Number of Transactions by Country')
plt.xlabel('Country')
plt.ylabel('Count')

plt.show()
```



seaborn.color_palette

seaborn.**color_palette**(*palette=None, n_colors=None, desat=None, as_cmap=False*)

Return a list of colors or continuous colormap defining a palette.

Parameters: **palette** : *None, string, or sequence, optional*

Name of palette or None to return current palette. If a sequence, input colors are used but possibly cycled and desaturated.

n_colors : *int, optional*

Number of colors in the palette. If `None`, the default will depend on how `palette` is specified. Named palettes default to 6 colors, but grabbing the current palette or passing in a list of colors will not change the number of colors unless this is specified. Asking for more colors than exist in the palette will cause it to cycle. Ignored when `as_cmap` is True.

desat : *float, optional*

Proportion to desaturate each color by.

Calling with no arguments returns all colors from the current default color cycle:

```
sns.color_palette()
```

Other variants on the seaborn categorical color palette can be referenced by name:

```
sns.color_palette("pastel")
```

Return a specified number of evenly spaced hues in the "HUSL" system:

```
sns.color_palette("husl", 9)
```

Return all unique colors in a categorical Color Brewer palette:

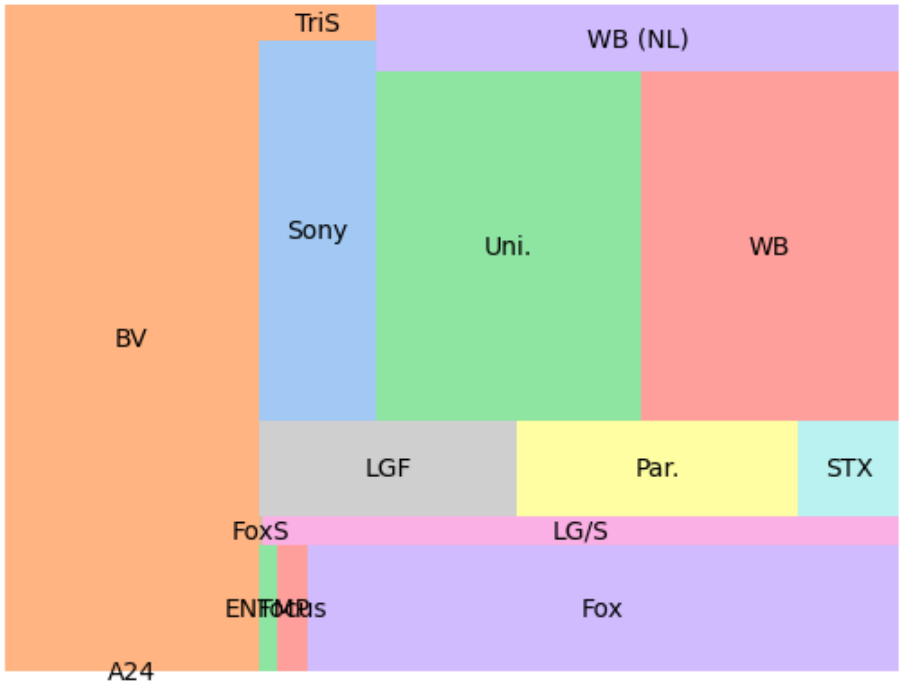
```
sns.color_palette("Set2")
```

Return a diverging Color Brewer palette as a continuous colormap:

https://seaborn.pydata.org/generated/seaborn.color_palette.html

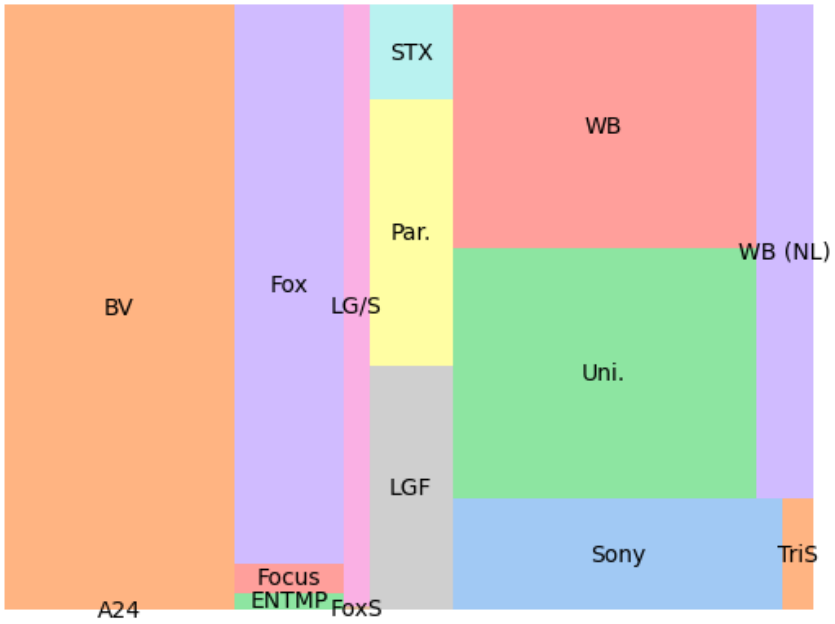
In [25]:
```
box_rev=pd.read_excel('BoxOffice.xlsx')
box_rev
studio_rev=box_rev.groupby('Studio').Opening.sum()
studio_rev
squarify.plot(sizes=studio_rev, label = studio_rev.index, color=sns.color_palette('pastel',len(studio_rev.index)))
plt.axis('off')
```

Out[25]:
```
(0.0, 100.0, 0.0, 100.0)
```

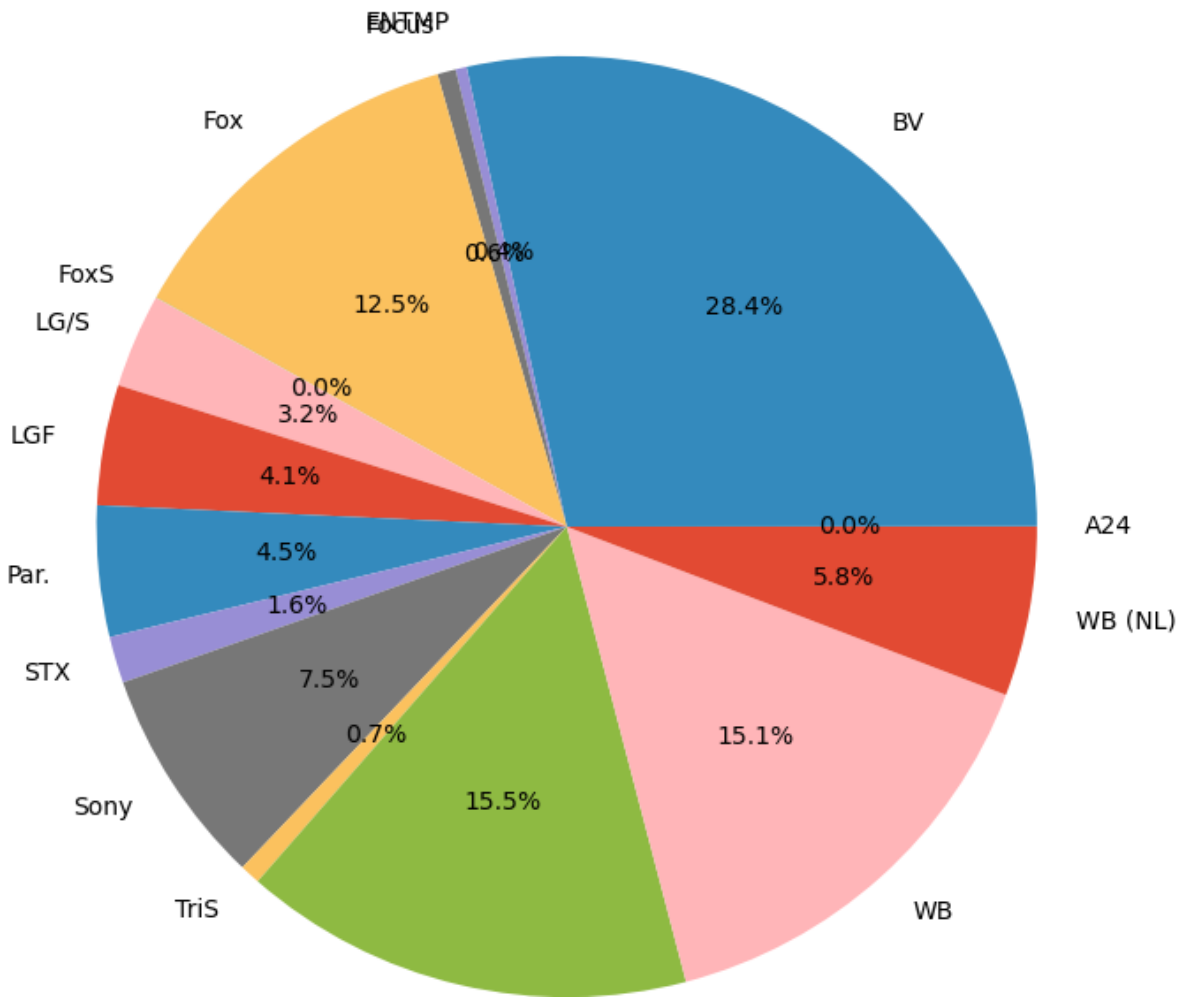


In [32]:
```
box_rev=pd.read_excel('BoxOffice.xlsx')
box_rev
studio_rev=box_rev.groupby('Studio').Opening.sum()
studio_rev
squarify.plot(sizes=studio_rev, label = studio_rev.index, norm_x=300, norm_y=150,color=sns.color_palette('pastel',len(studic
plt.axis('off')
```

Out[32]:
```
(0.0, 300.0, 0.0, 150.0)
```



In [24]:
```
plt.pie(studio_rev,labels=studio_rev.index, radius=1.8,autopct='%.1f%%')
plt.show()
```



numpy.where

numpy.where(*condition*, [*x*, *y*,]/)

Return elements chosen from *x* or *y* depending on *condition*.

> **Note**
>
> When only *condition* is provided, this function is a shorthand for `np.asarray(condition).nonzero()`. Using `nonzero` directly should be preferred, as it behaves correctly for subclasses. The rest of this documentation covers only the case where all three arguments are provided.

Parameters:

condition : *array_like, bool*

Where True, yield *x*, otherwise yield *y*.

x, y : *array_like*

Values from which to choose. *x*, *y* and *condition* need to be broadcastable to some shape.

Returns:

out : *ndarray*

An array with elements from *x* where *condition* is True, and elements from *y* elsewhere.

seaborn.catplot

```
seaborn.catplot(data=None, *, x=None, y=None, hue=None, row=None, col=None, col_wrap=None,
estimator='mean', errorbar=('ci', 95), n_boot=1000, units=None, seed=None, order=None,
hue_order=None, row_order=None, col_order=None, height=5, aspect=1, kind='strip',
native_scale=False, formatter=None, orient=None, color=None, palette=None, hue_norm=None,
legend='auto', legend_out=True, sharex=True, sharey=True, margin_titles=False, facet_kws=None,
ci='deprecated', **kwargs)
```

Figure-level interface for drawing categorical plots onto a FacetGrid.

This function provides access to several axes-level functions that show the relationship between a numerical and one or more categorical variables using one of several visual representations. The `kind` parameter selects the underlying axes-level function to use:

Categorical scatterplots:

- `stripplot()` (with `kind="strip"`; the default)
- `swarmplot()` (with `kind="swarm"`)

Categorical distribution plots:

- `boxplot()` (with `kind="box"`)
- `violinplot()` (with `kind="violin"`)
- `boxenplot()` (with `kind="boxen"`)

Categorical estimate plots:

- `pointplot()` (with `kind="point"`)
- `barplot()` (with `kind="bar"`)
- `countplot()` (with `kind="count"`)

Mosaic plots show relationships

A mosaic plot is a special type of stacked bar chart. For two variables, the width of the columns is proportional to the number of observations in each level of the variable plotted on the horizontal axis. The vertical length of the bars is proportional to the number of observations in the second variable within each level of the first variable.

Mosaic plots help show relationships and give a visual way to compare groups. Figure 1 shows a mosaic plot for data from a clinical trial. The goal is to compare the distribution of patients over 65 years in both the placebo and study drug treatment groups. Ideally, the clinical trial should have about the same percentage of elderly patients in each treatment group.

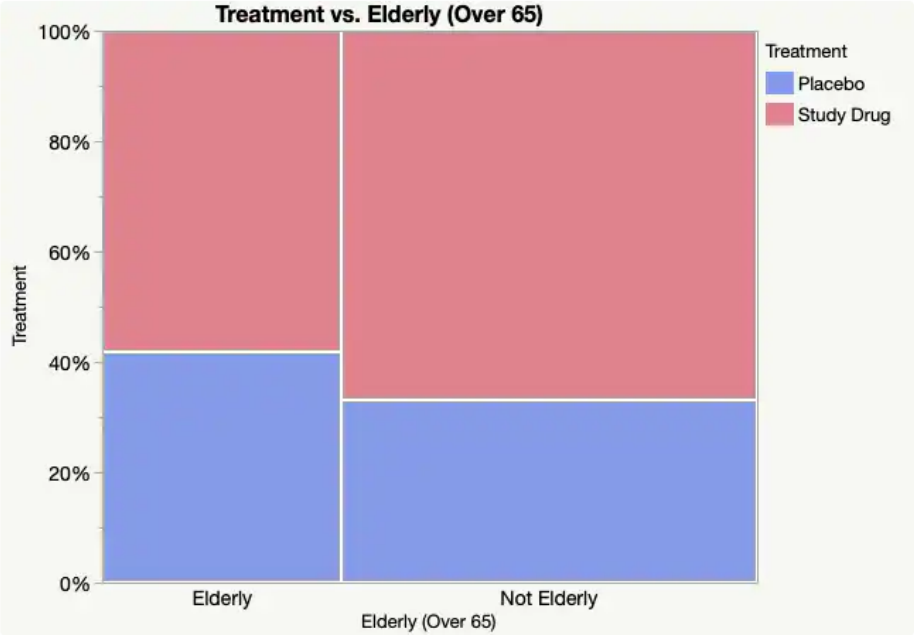


Figure 1: Mosaic plot comparing the distribution of elderly subjects across two treatment groups

statsmodels.graphics.mosaicplot.mosaic

```
statsmodels.graphics.mosaicplot.mosaic(
    data,
    index=None,
    ax=None,
    horizontal=True,
    gap=0.005,
    properties=<function <lambda>>,
    labelizer=None,
    title='',
    statistic=False,
    axes_label=True,
    label_rotation=0.0
)
```

data : {`dict`, `Series`, `ndarray`, `DataFrame`}

The contingency table that contains the data. Each category should contain a non-negative number with a tuple as index. It expects that all the combination of keys to be represents; if that is not true, will automatically consider the missing values as 0. The order of the keys will be the same as the one of insertion. If a dict of a Series (or any other dict like object) is used, it will take the keys as labels. If a np.ndarray is provided, it will generate a simple numerical labels.

index : `list`, `optional`

Gives the preferred order for the category ordering. If not specified will default to the given order. It does not support named indexes for hierarchical Series. If a DataFrame is provided, it expects a list with the name of the columns.

ax : `Axes`, `optional`

The graph where display the mosaic. If not given, will create a new figure

horizontal : bool, `optional`

The starting direction of the split (by default along the horizontal axis)

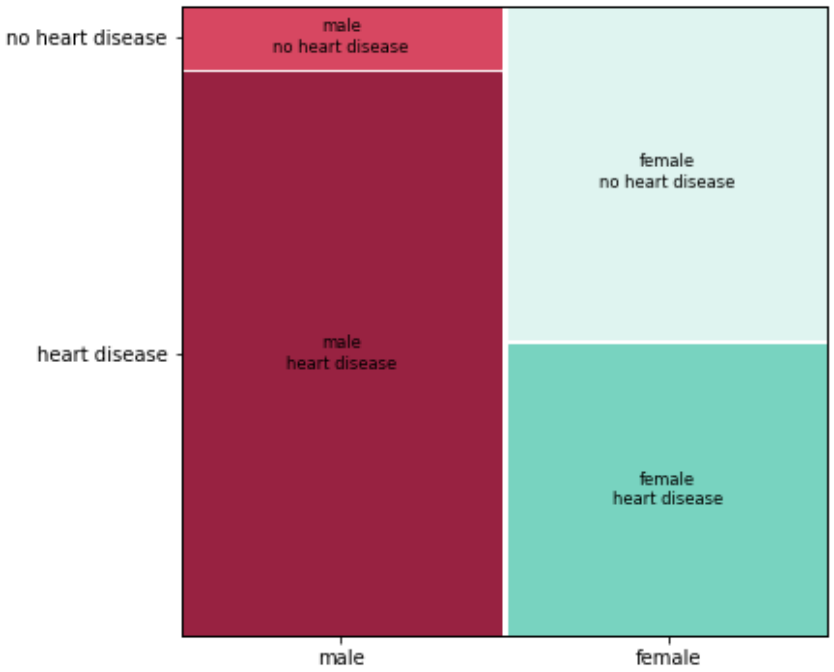
gap : {`float`, sequence[`float`]}

https://www.statsmodels.org/devel/generated/statsmodels.graphics.mosaicplot.mosaic.html

To change the color you need to provide a mapping that matches the names. Don't think you can change the label color easily:

```
from statsmodels.graphics.mosaicplot import mosaic
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1,figsize=(6,6))
gender = np.repeat(['male','female'],30)
heart_disease = [np.random.choice(['heart disease','no heart disease'],30,p=[0.8,0.2
                 np.random.choice(['heart disease','no heart disease'],30,p=[0.5,0.5
data = pd.DataFrame({'gender': gender, 'heart disease': np.array(heart_disease).flat
cols = {('male', 'heart disease'):'#9a1f40',('male', 'no heart disease'):'#d9455f',
        ('female','heart disease' ):'#74d4c0', ('female', 'no heart disease'):'#def4
x = mosaic(data,['gender','heart disease'],
           properties = lambda key: {'color': cols[key]} ,
           ax=ax,gap=0.01)
```



seaborn.displot

```
seaborn.displot(data=None, *, x=None, y=None, hue=None, row=None, col=None, weights=None,
kind='hist', rug=False, rug_kws=None, log_scale=None, legend=True, palette=None,
hue_order=None, hue_norm=None, color=None, col_wrap=None, row_order=None, col_order=None,
height=5, aspect=1, facet_kws=None, **kwargs)
```

Figure-level interface for drawing distribution plots onto a FacetGrid.

This function provides access to several approaches for visualizing the univariate or bivariate distribution of data, including subsets of data defined by semantic mapping and faceting across multiple subplots. The `kind` parameter selects the approach to use:

- `histplot()` (with `kind="hist"`; the default)
- `kdeplot()` (with `kind="kde"`)
- `ecdfplot()` (with `kind="ecdf"`; univariate-only)

Additionally, a `rugplot()` can be added to any kind of plot to show individual observations.

Extra keyword arguments are passed to the underlying function, so you should refer to the documentation for each to understand the complete set of options for making plots with this interface.

See the distribution plots tutorial for a more in-depth discussion of the relative strengths and weaknesses of each approach. The distinction between figure-level and axes-level functions is explained further in the user guide.

`andas.DataFrame`, `numpy.ndarray`, *mapping, or sequence*

t data structure. Either a long-form collection of vectors that can be assigned to ed variables or a wide-form dataset that will be internally reshaped.

ctors or keys in `data`

ables that specify positions on the x and y axes.

ctor or key in `data`

antic variable that is mapped to determine the color of plot elements.

: *vectors or keys in* `data`

Variables that define subsets to plot on different facets.

kind : *{"hist", "kde", "ecdf"}*

Approach for visualizing the data. Selects the underlying plotting function and determines the additional set of valid parameters.

rug : *bool*

If True, show each observation with marginal ticks (as in `rugplot()`).

rug_kws : *dict*

Parameters to control the appearance of the rug plot.

log_scale : *bool or number, or pair of bools or numbers*

Set axis scale(s) to log. A single value sets the data axis for univariate distributions and both axes for bivariate distributions. A pair of values sets each axis independently. Numeric values are interpreted as the desired base (default 10). If `False`, defer to the existing Axes scale.

legend : *bool*

If False, suppress the legend for semantic variables.

palette : *string, list, dict, or* `matplotlib.colors.Colormap`

Method for choosing the colors to use when mapping the `hue` semantic. String values

seaborn.pairplot

```
seaborn.pairplot(data, *, hue=None, hue_order=None, palette=None, vars=None, x_vars=None,
y_vars=None, kind='scatter', diag_kind='auto', markers=None, height=2.5, aspect=1,
corner=False, dropna=False, plot_kws=None, diag_kws=None, grid_kws=None, size=None)
```

Plot pairwise relationships in a dataset.

By default, this function will create a grid of Axes such that each numeric variable in `data` will by shared across the y-axes across a single row and the x-axes across a single column. The diagonal plots are treated differently: a univariate distribution plot is drawn to show the marginal distribution of the data in each column.

It is also possible to show a subset of variables or plot different variables on the rows and columns.

This is a high-level interface for `PairGrid` that is intended to make it easy to drav should use `PairGrid` directly if you need more flexibility.

Parameters:

data : *`pandas.DataFrame`*
Tidy (long-form) dataframe where each column is a variable and each row is an observation.

hue : *name of variable in `data`*
Variable in `data` to map plot aspects to different colors.

hue_order : *list of strings*
Order for the levels of the hue variable in the palette

palette : *dict or seaborn color palette*
Set of colors for mapping the `hue` variable. If a dict, keys should be values in the `hue` variable.

vars : *list of variable names*
Variables within `data` to use, otherwise use every column with a numeric datatype.

{x, y}_vars : *lists of variable names*
Variables within `data` to use separately for the rows and columns of the figure; i.e. to make a non-square plot.

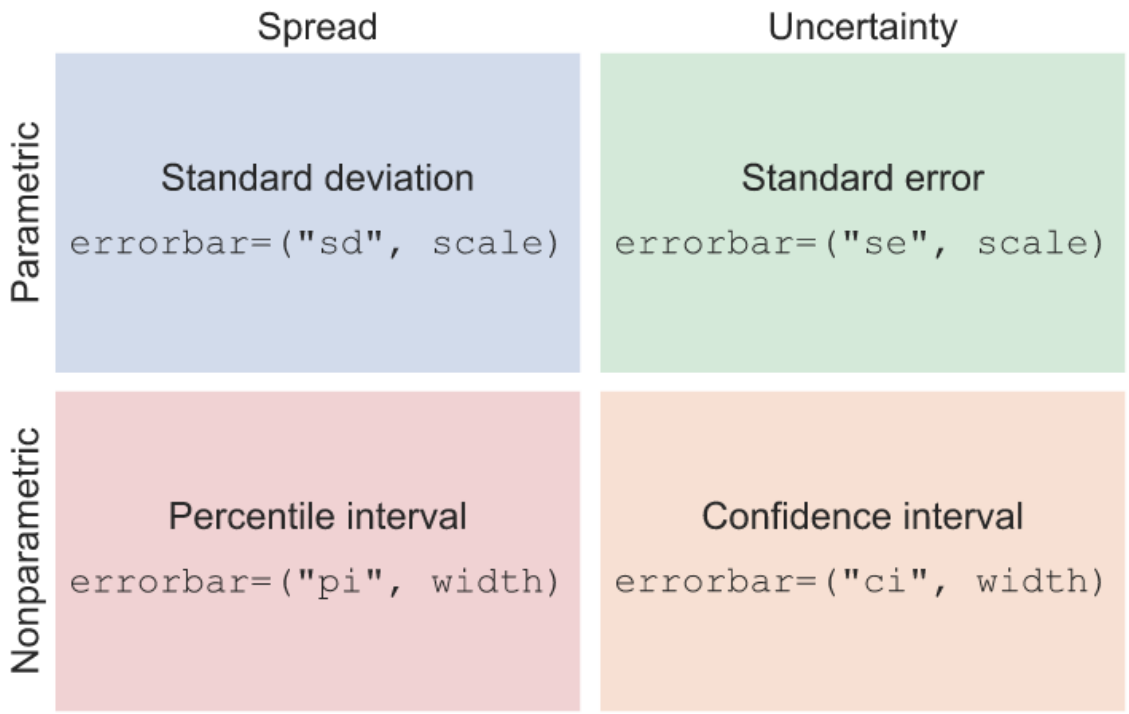
kind : *{'scatter', 'kde', 'hist', 'reg'}*
Kind of plot to make.

diag_kind : *{'auto', 'hist', 'kde', None}*
Kind of plot for the diagonal subplots. If 'auto', choose based on whether or not `hue` is used.

markers : *single matplotlib marker code or list*
Either the marker to use for all scatterplot points or a list of markers with a length the same as the number of levels in the hue variable so that differently colored points will

In seaborn, there are two approaches for constructing each kind of error bar. One approach is parametric, using a formula that relies on assumptions about the shape of the distribution. The other approach is nonparametric, using only the data that you provide.

Your choice is made with the `errorbar` parameter, which exists for each function that does estimation as part of plotting. This parameter accepts the name of the method to use and, optionally, a parameter that controls the size of the interval. The choices can be defined in a 2D taxonomy that depends on what is shown and how it is constructed:



https://seaborn.pydata.org/tutorial/error_bars.html

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.

https://seaborn.pydata.org/generated/seaborn.heatmap.html

seaborn.scatterplot

```
seaborn.scatterplot(data=None, *, x=None, y=None, hue=None, size=None, style=None,
palette=None, hue_order=None, hue_norm=None, sizes=None, size_order=None, size_norm=None,
markers=True, style_order=None, legend='auto', ax=None, **kwargs)
```

Draw a scatter plot with possibility of several semantic groupings.

Parameters:

data : `pandas.DataFrame`, `numpy.ndarray`, *mapping, or sequence*

Input data structure. Either a long-form collection of vectors that can be assigned to named variables or a wide-form dataset that will be internally reshaped.

x, y : *vectors or keys in* `data`

Variables that specify positions on the x and y axes.

hue : *vector or key in* `data`

Grouping variable that will produce points with different colors. Can be either categorical or numeric, although color mapping will behave differently in latter case.

size : *vector or key in* `data`

Grouping variable that will produce points with different sizes. Can be either categorical or numeric, although size mapping will behave differently in latter case.

style : *vector or key in* `data`

Grouping variable that will produce points with different markers. Can have a numeric dtype but will always be treated as categorical.

palette : *string, list, dict, or* `matplotlib.colors.Colormap`

Method for choosing the colors to use when mapping the `hue` semantic. String values are passed to `color_palette()`. List or dict values imply categorical mapping, while a colormap object implies numeric mapping.

hue_order : *vector of strings*

Specify the order of processing and plotting for categorical levels of the `hue` semantic.

hue_norm : *tuple or* `matplotlib.colors.Normalize`

Either a pair of values that set the normalization range in data units or an object that will map from data units into a [0, 1] interval. Usage implies numeric mapping.

sizes : *list, dict, or tuple*

An object that determines how sizes are chosen when `size` is used. List or dict arguments should provide a size for each unique data value, which forces a categorical

seaborn.FacetGrid

class seaborn.**FacetGrid**(*data*, ***, *row=None*, *col=None*, *hue=None*, *col_wrap=None*, *sharex=True*, *sharey=True*, *height=3*, *aspect=1*, *palette=None*, *row_order=None*, *col_order=None*, *hue_order=None*, *hue_kws=None*, *dropna=False*, *legend_out=True*, *despine=True*, *margin_titles=False*, *xlim=None*, *ylim=None*, *subplot_kws=None*, *gridspec_kws=None*)

Multi-plot grid for plotting conditional relationships.

__init__(*data*, ***, *row=None*, *col=None*, *hue=None*, *col_wrap=None*, *sharex=True*, *sharey=True*, *height=3*, *aspect=1*, *palette=None*, *row_order=None*, *col_order=None*, *hue_order=None*, *hue_kws=None*, *dropna=False*, *legend_out=True*, *despine=True*, *margin_titles=False*, *xlim=None*, *ylim=None*, *subplot_kws=None*, *gridspec_kws=None*)

Parameters: **data** : *DataFrame*

Tidy ("long-form") dataframe where each column is a variable and each row is an observation.

row, col, hue : *strings*

Variables that define subsets of the data, which will be drawn on separate facets in the grid. See the `{var}_order` parameters to control the order of levels of this variable.

col_wrap : *int*

"Wrap" the column variable at this width, so that the column facets span multiple rows. Incompatible with a `row` facet.

share{x,y} : *bool, 'col', or 'row' optional*

If true, the facets will share y axes across columns and/or x axes across rows.

height : *scalar*

Height (in inches) of each facet. See also: `aspect`.

aspect : *scalar*

Aspect ratio of each facet, so that `aspect * height` gives the width of each facet in inches.

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.

{row,col,hue}_order : *lists*

Order for the levels of the faceting variables. By default, this will be the order that the levels appear in `data` or, if the variables are pandas categoricals, the category order.

hue_kws : *dictionary of param -> list of values mapping*

Other keyword arguments to insert into the plotting call to let other plot attributes

In [30]:
```
mpg['mpg'].corr(mpg['weight'])
```

Out[30]:
```
-0.8317409332443351
```

In [32]:
```
mpg.corr(numeric_only=True)
```

Out[32]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

In [34]:
```
mpg_matrix=mpg.corr(numeric_only=True)
sns.heatmap(mpg_matrix,annot=True,cmap='Blues')
```

Out[34]: <Axes: >