Seaborn

* It’s a statistical plotting library built on matplotlib
* Imported with ‘import seaborn as sns’
* Use %matplotlib in line to help display the plots in line while coding.
* Seaborn comes by default with some data sets that you can load e.g., sns.load\_dataset(a) where ‘a’ is the string representation of the data set name.

# Distribution plots

* The distribution (dist) plot allows you show the distribution of a univariate set of observation.
* You can create a distplot with sns.distplot(a) where a is the column title for the data to be plotted in the form data frame [‘column’]
* You can remove the kernel density estimation (KDE) by passing the argument kde=False into the distplot call i.e., sns.distplot(a, kde=False)
* You can also set the bin size to plot a specific number of bars i.e., sns.distplot(a, kde=False, bin=d) where d is an integer.

## Joint plots

* A joint plot is used to plot bivariate data on the same axis.
* You can create a joint plot with sns.jointplot(a, b, data=’c’, kind=’d’) where a and b are the data sets for the x and y axes respectively, c is a string of the name of the data frame, and d is the type of plot.
* Type of plot include scatter (which is the default), reg (regression line), KDE (kernel density estimation), hex (for hexagons) etc.

## Pair plots

* Pair plot is an easy and fast way to plot every possible combination of numerical data in the data frame.
* You can create a pair plot with sns.pairplot(a) where a is the name of the data frame. Since its going to automatically plot every possible combination, there is no need to specify beyond the data frame name.
* You can add in a hue and palette arguments respectively e.g., sns.pairplot(a, hue=’b’, palette = ‘c’) where b is the column title of a non-numerical (or categorical) data in the data frame and c is a choice palette.
* Note that it returns a histogram for any combination of a column with itself.

## Rug plots

* Rug plots are similar to a distplot but differ simply in that, the rug plot uses dash lines instead of bars to plot.
* You can call a rug plot with sns.rugplot(a[‘b’]) where a is the data frame name and b is the column title.

## Kernel Density Estimation (kde) plots

* KDE plots replace every single observation with a normal distribution (Gaussian) curve around it.
* NOT UNDERSTOOD

# Categorical plots

* Categorical plots are used to plot categorical data against one another or against numeric data.
* Categorical data are categories that don’t overlap such as Gender (male, female) etc.

## Bar plots

* You can use a bar plot to plot a categorical data with sns.barplot(x=’a’, y= ‘b’, data = c) where a is the column for the x axis, b is the column for y axis, c is the data frame name.
* Bar plots return aggregate of each category and by default, the mean is returned.
* To change the aggregate function being used, you need to import numpy and call an aggregate (e.g., std, etc.) from the numpy library. E.g., sns.barplot(x=’a’, y= ‘b’, data = c, estimator = np.std) estimator is kwarg for aggregator.

## Count plots

* A count plot is simply a plot of the number of occurrences of a categorical data.
* You can call it with the sns.counplot(x=’a’, data = ‘b’ ) were a is the column name of the categorical data set, and b is the data frame name.

## Box plots

* Box plot is used to visualize distribution of categorical data in a way that facilitates comparison among the variables.
* Box plots are also known as a box and whisker plot.
* You can create a box plot with sns.boxplot(x = ‘a’, y = ‘b’, data = ‘c’) where a and b are the column names respectively for the plot and c is the data frame title.
* The box plot essentially reveals the quartiles and outlier for every plotted category.
* You can further split the categories by yet other categories using the hue argument e.g., sns.boxplot(x = ‘a’, y = ‘b’, data = ‘c’, hue = ‘d’) where d is another categorical data column name.
* It is advised to always have the categorical data on the x axis and the numerical data on the y axis.
* The points outside the whiskers are outliers

## Violin plots

* A violin plot like a box plot also plots the distribution of categorical data and takes the exact arguments as a box plot.
* You can create it with sns.violinplot(x = ‘a’, y = ‘b’, data = ‘c’) where is a categorical data column, b is the numeric column name and c is the data frame title.
* You can also call a hue argument in a violin plot with sns.violinplot(x = ‘a’, y = ‘b’, data = ‘c’, hue = ‘d’) where d is another categorical column of data sets.
* Additionally, and this differs from box plot, you can call the split argument to combine both halves of the hue argument to enable comparison further with sns.violinplot(x= 'a, y= 'b', data = c, split = True). True is Boolean.
* Violin plot does have the draw down of requiring a lot more knowledge and possibly time to understand its visualization compared to a box plot.

## Strip plots

* Strip plots draw a scatter plot of the given data set where one of the data sets is categorical data.
* You can create a strip plot with sns.stripplot(x = ‘a’, y = ‘b’, data = ‘c’) where a, b and c are the categorical data column name, numeric data column name and data frame name respectively.
* Due to the usual concentration of dots in a point when using strip plots, it is usual to add an extra argument called jitters to slant the dots with (x = ‘a’, y = ‘b’, data = ‘c’, jitters = True). Jitters is Boolean.
* Strip plots can also take hue, split arguments e.g., (x = ‘a’, y = ‘b’, data = ‘c’, jitters = True, hue = ‘d’, split = True) where d is the column name of the categorical data set to be hued on.
* It is important to note that strip clubs are more difficult to interpret than box plots.

## Swam plots

* A swam plot is a combination of the violin and strip plots.
* It is not scalable and should not be used for very large data sets.
* It essentially creates a violin plot with the dots of a strip plot thereby avoiding stacking dots.
* You can create it with sns.swarmplot(x = ‘a’, y = ‘b’ , data = c) where a, b and c are the categorical data column name, numeric data column name and data frame name respectively.
* You can stack a violin plot on the swarm plot to improve interpretation by calling a violin plot for the same data sets in the same cell and setting the swarm plot color to a specific color which lets the swarm plot be clear and uniform across the various violin plot colors.
* To specify color for swarm plots, pass the argument color e.g., sns.swarmplot(x = ‘a’, y = ‘b’ , data = c, color=’d’) where d is the color such as black.

Note: swarm, violin and strip plots are more suitable for exploration over presentation especially when the audience are not data analysts or scientists.

## Factor plots

* Factor plot is a general use plot. It plots any of the other categorical plots by taking the arguments necessary i.e., x, y, data etc. and an extra argument for the plot type such as violin, bar, strip, box etc.
* You can create a factor plot with sns.factorplot(x = ‘a’, y = ‘b’ , data = c, kind = ’e’) where a, b, c and d are the categorical data column name, numeric data column name, data frame name and plot type respectively.

# Matrix plots

## Heat maps

* Heat maps are used to visualize data in the form of a matrix. Your data should already be in matrix form to make sure it works well.
* Data being in matrix form implies that the data has both a relevant column and row title respectively. The default row index will not be suitable.
* The best ways to transform your data into a matrix from a data frame is usually correlation or pivot table.
* To call correlation data, use a.corr() where ‘a’ is data frame name.
* You can call a heatmap using sns.heatmap(a) where ‘a’ is variable name for transformed data.
* You can add arguments for color and annotation e.g., sns.heatmap(a, annot = True, cmap = ‘c’) where b and c are a Boolean and a color scheme (such as ‘coolwarm’) respectively.
* You can transform a data into a pivot table with a.pivot\_table(index = ‘b’, columns = ‘c’, values = ‘d’) where a is data frame name, b is column name to be used as index, column name to be used as columns and d is the values in the new data frame.
* You can use the linewidths and line color arguments to customize the heat maps.

## Cluster maps

* Cluster maps use clustering (putting together similar data sets in a data frame) to create a heat map.
* Cluster maps will not necessarily be in order
* You can create a cluster map with sns.clustermap(a) where a is the variable name of the transformed data.
* You can add cmap argument as well
* You can normalize the scale with a max. min is always 0 by default by passing the standard scale argument e.g., sns.clustermap(fp, cmap = 'coolwarm', standard\_scale=t) where t is the scale max in integer.

# Grids

## Pair Grid

* Pair grid is a type of pair plot with a lot more control over things that pair plot does automatically.
* Pair grid creates an empty grid of pair plots that can then be slowly filled
* To start with, create the pair grid and assign it to a variable e.g., pg = sns.PairGrid(a) where pg is assigned variable name, a is the data frame name.
* You can then begin plotting with pg.map(plt.b) where b is the preferred type of plot such as scatter for a single plot type for all the plots.
* The set of plots are divided into upper, diagonal and lower which are those above the diagonal, those on the diagonal and those below the diagonal. This allows you to specify a plot type for each of those groups.
* To specify plot type for a set of the pair grids, use pg.map\_upper(a), pg.map\_diag(a) and pg.map\_lower(a) to specify the upper diagonal and lower plot types respectively. Where ‘a’ is the preferred plot type for that group of plots.
* Pair grid is simply a pair plot with a lot more control.

## Facet Grid

* Facet grid needs to be assigned to a variable just like pair grid.
* It is used to separate data plots based on specific arguments. It sort of allows for clarification of plots using two data sets.
* You can create a facet grid with fg = sns.FacetGrid(data = ’a’, col = ‘b’, row = ‘c’) where a, b and c are data frame name, column name for facet grid column and rows respectively.
* You then pass in the data to be mapped with fg.map(d, e) or fg.map(d, ‘e’, ‘f’) where d is plot type command such as plt.scatter, e and f are the columns to be plotted depending on if the plot type requires only one data set or two.
* The primary use is to customize plots even further.
* Note that the plot type command is not called with its usual accompanying brackets.

# Regression plots

## Linear model plots (lmplot)

* Lmplots allow you to display linear models with seaborn. It creates a scatter plot with a linear model stacked on it.
* You can create a lmplot with sns.lmplot(x = ’a’, y = ’b’, data = c) where ‘a’ is the column to be plotted on the x axis, ‘b’ is the column to be plotted on the y axis and c is the assigned variable name of the data frame.
* You can make it separate based on some categorial feature with the hue argument e.g., sns.lmplot(x = ’a’, y = ’b’, data = c, hue = ‘d’) where ‘d’ is the column name of the categorical data to be hued on.
* You can customize the marker type just like in matplotlib e.g., sns.lmplot(x = 'total\_bill', y = 'tip', data=tips, hue = ‘sex’, markers = [‘o’, ‘v’]). Note: the marker object is being passed as a list because the plot has been hued implying two plots on it.
* You can usually make a direct call to matplotlib from seaborn since seaborn is built on matplotlib. For example, scatter\_kws is a matplotlib keyword argument that can be used to change the marker size of a lmplot in seaborn e.g., sns.lmplot(x = 'total\_bill', y = 'tip', data=tips, hue = ‘sex’, markers = [‘o’, ‘v’], scatter\_kws = {‘s’: b}) where is represents marker size argument and b is the integer for marker size.
* Instead of the hue argument, you can differentiate a plot with Facet Grid arguments such as adding columns and rows arguments to create rows or column with the data separated according to the passed argument e.g., sns.lmplot(x = 'total\_bill', y = 'tip', data=tips, col = ‘e’, row = ‘f’).
* You can still add hue after columns and rows arguments if necessary.
* If the plots become cluttered due to the number of hue and columns or rows, you can customize the figure size and aspect (which state the ratio of the height to the width).
* You can customize the figure size and aspect ratio with the arguments sns.lmplot(x = 'total\_bill', y = 'tip', data=tips, col = ‘e’, row = ‘f’ size=’g’, aspect=’h’).
* Size argument has been renamed height as it primarily affects the figure’s height.

# Styles and color

## Styles

* You can set a style that then affects whatever plot you create subsequently with sns.set\_style(‘a’) where a can be darkgrid, whitegrid, white, ticks or dark respectively.
* You can remove the plot spines with sns.despine(). This automatically removes the top and right spines.
* To remove either or both of the left and bottom spine, you need to pass in the argument as True because they are by default False (only the top and right are True by default)
* You can use matplotlib figsize, height and width to customize for seaborn since seaborn is simply calling matplotlib in effect. For instance, plt.figure(figsize=(a, b)) will work correctly on seaborn plots where a and b are the width and height respectively.

## Scale and context

* You can set the context of a figure to override all the styles and give you a figure that suits your context with sns.set\_context(‘a’) where a is the context and can be paper, notebook, talk poster or a dictionary respectively.
* You can also pass in the font scale argument to customize the font with sns.set\_context(‘a’, font\_scale = b) where b is an integer that specifies the multiplier for the default font size you would prefer. This means it simply scales or multiplies the default font size of the context by the integer passed.

## Palettes and colors

* You can choose a palette for a seaborn plot by passing the palette argument e.g., sns.lmplot(x=’a’, y=’b’, data=c, hue=’d’, palette=’e’) where ‘e’ is a palette name in strings.
* You can find different palette name in the matplotlib colour map page.