Intermediate Seaborn
by datacamp

sns.displot(df['col'], kde=True, bins=10)
kde is the kernel distribution error (ie the line that flows over the bins of the histogram)

rug plot is an alternative way to view the distribution of data by including small tick marks along the x axis
function that use displot function
sns.displot(df['col'], kde=True, rug=True, fill=True)
'fill' fills in under the kde curve

ecdfplot also uses the displot function
shows the cumulative distribution of the data
sns.distplot(df['col'], kind='ecdf'
** gives a stair-ish curve showing that cumulation of the data

Above analysis is univariate
For bivariate analysis like regression analysis
Regression plots
data and x and y must be defined
sns.regplot(data=df, x='alcohol', y='pH')
**like kde and rug plot act as building blocks for the displot, we can see a similar relationship with regression plots
**regplot() is considered a low level analysis while lmplot() is considered high level

lmplot has more features
organize data by colors (hue)
organize data by columns (col)
***the use of plotting multiple graphs while changing a single variable is often called faceting

Setting styles
sns.set()
a nice way to see which style may benefit your needs

```
for style in ['white', 'dark', 'whitegrid', 'dark grid', 'ticks']:
    sns.set_style(style)
    sns.displot(df['col'])
    plt.show()
```

the lines around the axes are called ‘spines’
to remove
sns.despine(left=True)
default is to remove top and right but can also remove left and bottom

Colors
seaborn supports assigning colors to plots using matplotlib color codes
sns.set(color_codes=True)
sns.displot(df[’Tuition’], color=‘g’)

Palettes
seaborn uses the set_palette() function to define a palette
6 default palettes
palettes = [‘deep’, ‘muted’, ‘pastel’, ‘bright’, ‘dark’, ‘colorblind’]
for p in palettes:
    sns.set_palette(p)
    sns.displot(df[‘col’])

sns.palplot() display color swatches in a Jupiter notebook
sns.color_palette() returns the current palette

Displaying palettes
palettes = [see above]
for p in palettes:
    sns.set_palette(p)
    sns.plaplot(xns.color_palette())
    plt.show()
**this will display the six default palettes

**Defining custom palettes
circular colors for when the dat is not ordered
sns.palplot(sns.color_palette(‘Paired’, 12))
**paired colors; 12 of them

sequential colors for when the data has a consistent range from high to low
sns.palplot(sns.color_palette(‘Blues’, 12))
** gives a range of 12 blues from light to dark

diverging colors when both the low and high values are interesting
sns.palplot(sns.color_palette(‘BrBG’, 12))
** give a range of 12 of half brown and half blue green

Customizations with Matplotlib axes

axes can be passed to seaborn functions
example
fig, ax =plt.subplots()
sns.histplot(df['Tuition'], ax=ax)
ax.set(xlabel='Tuition 2013-14', ylabel=, xlim=)
title=

Combining plots
example
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(7,4))
sns.histplot(df['Tuition'], stat='density', ax=ax0)
sns.histplot(df.query('State == 'MN'')['Tuition'], stat='density', ax=ax1)
ax1.set(xlabel='Tuition (MN)', xlim=(0, 70000))
ax1.axvline(x=20000, label='My Budget', linestyle='—')
ax1.legend()

Categorical plot types
categorical data is data which includes a limited or fixed number of values and is most useful when combined with numeric data
3 subgroups for categorical plots
First show all of the individual observations on the plot
stripplot() and swarmplot()
Second shows an abstract representation of the categorical data
boxplot() and violinplot() and boxenplot()
Third show statistical estimates of the categorical variables
barplot(), countplot(), and pointplot()

Stripplot
sns.stripplot(data=df, y='', x='', jitter=True)

Swarmplot
sns.swarmplot(data=df, y='', x='')
** places observations in a manner where they do not overlap
** downside not good for datasets with lots of observations

Boxplot
sns.boxplot(data=df, y='', x='')
shows several measures related to the distribution of the data

Violinplots
sns.violinplot(data=df, y='', x='')
uses a kernel density calculation
it does not show all data points

useful for displaying large datasets
computationally intensive to create

Boxenplot
enhanced boxplot
sns.boxenplot(data=df, y='', x='')
scales more effectively to large datasets
hybrid between box and violin

Barplot
sns.barplot(data=df, y='', x='', hue='')
shows an estimate of the value as well as a confidence interval

Pointplot
sns.pointplot(data=df, y='', x='', hue='')
shows summary measure and confidence interval
useful for observing how values change across categorical values

Countplot
sns.countplot(data=df, y='', hue='')
displays the number of instances of each variable

Evaluating regression with residplot()
useful for evaluating the fit of a model
sns.residplot(data=df, x='temp', y='total_rentals')

Polynomial regression with order parameter
sns.regplot(data=df, x='', y='', order=2)

**can use x_jitter parameter, may make it easier to see the individual distribution
example x_jitter=0.1
**x_estimator can be useful for highlighting trends
example x_estimator=np.mean
**can also divide data into discrete bins
example x_bins=4

Matrix plots
heat map is the most common
help to quickly see trends in data
sns.heatmap() requires data to be in a grid format
pandas crosstab() is frequently used to manipulate the data
example
sns.heatmap(pd.crosstab(df['month'], df['weekday'], values=df['total_rentals'],

aggfunc='mean').round(0))

Customize a heat map
annot=True will turn annotations on in individual cells
fmt='d' ensures that the results are displayed as integers
cmap='YlGnBu' in this case will change the shading off these three colors
cbar=False then then color bar is not displayed
linewidths=.5 puts some spacing between the cells
center=df_crosstab.loc[9,6] can center off index

Using a correlation matrix with heat maps
sns.heatmap(df[cols].corr(), cmap='YlGnBu')

Using Facetgrid
advantages of analyzing multiple plots of data
useful with data with many variables
allows you to quickly identify trends
referred to as a trellis or lattice plot
also in data science often called faceting
data needs to be tidy
ie one observation per row of data

FacetGrid
foundational for many data aware grids
allows the user to control how data is distributed across columns, rows, and hue

once a FacetGrid is created, the plot type must be mapped to the grid
example of this
g = sns.FacetGrid(df, col='HIGHDEG')
g.map(sns.boxplot, 'Tuition', order=['1', '2', '3', '4'])

catplot()
is a shortcut to creating FacetGrids
combines the faceting and mapping process into 1 function
example
sns.catplot(x='Tuition', data=df, col='HIGHDEG', kind='box')

FacetGrid for regression
can be used for scatter or regression plots
g = sns.FacetGrid(df, col='HIGHDEG')
g.map(plt.scatter, 'Tuition', 'SAT_AVG_ALL')

lmplot() creates shortcut similar to catplot but for regression

sns.lmplot(data=df, x='Tuition', y='SAT_AVG_ALL', col="HIGHDEG', fit_reg=False)
fit_reg disables regression lines when set to False

PairGrid and pair plot
allow us to see interactions across different columns of data
we only have to define the columns of data we want to compare

PairGrid
shows us pairwise relationships between data elements
like FacetGrid you have to apply it then map it
g = sns.PairGrid(df, vars=['Fair_Mrkt_Rent', 'Median_Income'])
g = g.map(sns.scatterplot)

Customizing PairGrid
can map to on or off diagonal
g = g.map_diag(sns.scatterplot)
g = g.map_offdiag(sns.scatterplot)

Pairplot
shortcut for the PairGrid
sns.pairplot(df, vars=['Fair_Mrkt_Rent', 'Median_Income'], kind='reg',
diag_kind='hist')

Customizing a pair plot
sns.pairplot(df.query('BEDRMS < 3'), vars=['Fair_Mrkt_Rent', 'Median_Income',
'Utility'], hue='BEDRMS', palette='husl', plot_kws={'alpha': 0.5})

JointGrid
allows us to compare the distribution of data between two variables
uses scatter plots, kdes, histograms, regression lines, distribution plots to give us insights into our data
input is an x and y variable
in the center is a scatter plot
the x and y axis show the distribution of the data for each variable
like the other grids you need to define the grid than map onto the grid
example
g = sns.JointGrid(data=df, x='Tuition', y='ADM_RATE_ALL')
g.plot(sns.regplot, sns.histplot)

Advanced JointGrid
g = sns.JointGrid(data=df, x='Tuition', y='ADM_RATE_ALL')
g = g.plot_joint(sns.kdeplot)
g= g.plot_marginals(sns.kdeplot, shade=True)

here plot_joint specifies that a kde plot should be included in the center
plot_marginals defines kdeplots on the margins

jointplot()
easier to use but less customizations than JointGrid
sns.jointplot(data=df, x='Tuition', y='ADM_RATE_ALL', kind='hex')
example gives us a hex plot in the center
supports scatter, hex, residual, regression, and kde plots
**can also add overlay plots to enhance the final output
another example
g = (sns.jointplot(x='Tuition', y='ADM_RATE_ALL', kind='scatter', xlim=(0, 25000), data=df.query('UG < 2500 & Ownership == 'Public'')).plot_joint(sns.kdeplot))

How to select your Seaborn plot
first step in analyzing numerical data is looking at its distribution
distplot() is the best place to start for this analysis
rugplot, kdeplot, and ecdfplot can be useful alternatives

Regression Analysis
regression plots show the relationship between two variables
lmplot performs regression analysis and supports faceting
lmplot is often the best function to use for determining linear relationships between data
once again faceting is creating multiple subplots based on subsets of the data
allows you to break down your data into categories and display them in panels or facets