matplotlib

import matplotlib.pyplot as plt
fig, ax = plt.subplot() **if add say (3,2) this would give us 3 plots vertical and 2 plots horizontal
plt.show()

Adding data to axes
example
ax.plot(df['col1'], df['col2'], marker='o', linestyle='—', color='r')
ax.set_xlabel('')
ax.set_ylabel('')
ax.set_title('')
plt.show()

Using twin axes
example for time series data (where two variables on y-axis have different scales)
ax.plot...
ax2 = ax.twinx()
ax2.plot(df.index, df['col'], color=)
ax2.set_ylabel('', color=)
*place different color for ax and ax2 so that each variable and its scale can easily be defined
#color ticks
ax2.tick_params('x', colors='')

A function that plots time-series
def plot_timeseries(axes, x, y, color, label, ylabel);
    axes.plot(x, y, color=color)
    axes.set_xlabel(xlabel)
    axes.set_ylabel(label, color=color)
    axes.tick_params('y', colors=color) *key see that it is colors, not color for the ticks, **but then equals color!

Annotating time-series data
example annotating on our climate change time series data the date where the temperature jumped 1 degree celsius
ax2.annotate('>1 degree', xy=(pd.Timestamp('2015-10-06'), 1), xytext=(pd.Timestamp('2008-10-06;), -0.2), arrowprops={'arrowstyle':'->', 'color':'gray'})

Bar charts
show us the values of one variable across different conditions
example
medals = pd.read_csv('medals_by_country_2016.csv', index_col=0
fig, ax = plt.subplots()
ax.bar(medals.index, medals['Gold']
plt.show()

Rotate the tick labels
ax.set_xticklabels(medal.index, rotation=90)

Stacked bar chart
example
ax.bar(medals.index, medals['Gold'], label='Gold')
ax.bar(medals.index, medals['Silver'], bottom=medals['Gold'], label='Silver')
ax.bar(medals.index, medals['Bronze'], bottom=medals['Gold'] + medals['Silver'], label='Bronze')
ax.legend
**added label and ax.legend to label and color code each stack

Histograms
shows us the entire distribution of values within a variable
example
fig, ax = plt.subplots()
ax.hist(mens_rowing['Height'], label='Rowing', bins=5)
ax.hist(mens_gymnastics['Height'], label='Gymnastics', bins=5)
ax.set_xlabel('Height (cm)')
ax.set_ylabel('# of observations')
plt.show()
**can also make boundaries for bins > for this example: bins=[150, 160, 170, 180, 190, 200, 210]

Adding error bars to bar charts
example
fig, ax = plt.subplots()
ax.bar('Rowing', mens_rowing['Height'].mean(), yerr=mens_rowing['Height'].std())
another example
ax.errorbar(seattle_weather['MONTH'], seattle_weather['MLY-TAVG-NORMAL'], yerr=seattle_weather['MLY-TAVG-STDDEV'])

Adding boxplots
fig, ax = plt.subplots()
ax.boxplot([mens_rowing['Height'], mens_gymnastics['Height']])

ax.set_xticklabels([‘Rowing’, ‘Gymnastics’])
ax.set_ylabel(‘Height (cm)’)
plt.show()
**box represents range 25-75% of data, line within box is the median, between the whiskers is 99% (1.5x outside 25% or 75%) of the data, dots outside of whiskers are outliers (the 1%)

Scatterplot
useful when you want to compare the values of different variables across observations (called a bivariate comparison)
example
fig, ax = plt.subplot()
ax.scatter(climate_change[‘co2’], climate_change[‘relative_temp’])
ax.set_xlabel(‘CO2 (ppm)’)
ax.set_ylabel(‘Relative temperature (Celsius)’)
plt.show()
another example
eighties = climate_change[’1980-01-01’:’1989-12-31’]
nineties = climate_change[’1990-01-01’:’1999-12-31’]
fig, ax = plt.subplots()
ax.scatter(eighties[‘co2’], eighties[‘relative_temp’], color=‘red’, label=‘eighties’)
ax.scatter(nineties[‘co2’], nineties[‘relative_temp’], color=‘blue’, label=’nineties’)
ax.legend()
ax.set_xlabel(‘’)
ax.set_ylabel(‘’)

Encoding a third variable by color
example
**we have a continuous variable denoting time stored in the DataFrame index; we can encode this as color
fig, ax = plt.subplots()
ax.scatter(climate_change[‘co2’], climate_change[‘relative_temp’], c=climate_change.index)
**time is now shown in the brightness of the data points

Choosing a style
example
plt.style.use(‘ggplot’)
fig, ax = plt.subplot()
ax.plot(df[], df[])
ax.plot(df2[], df2[])
**see matplotlib for an array of styles

Back to default
**will continue in this style until you change it
plt.style.use('default')

Guidelines for choosing plotting style
- dark backgrounds are usually less visible
- if color is important, consider choosing colorblind-friendly options like 'seaborn-colorblind' or 'tableau-colorblind10'

Saving the figure to file
**at the end
fig.savefig('gold_medals.png')
**in the Python shell can call the unix ls function, which will give us a listing of the files in the present working directory
**can also save as .jpg with same coding format
fig.savefig('gold_medals.jpg')
**jpg is higher quality than png but takes up more memory space (ie disk space or bandwidth)
jpg uses lossy compression which allows you to create figures that take up less space
fig.savefig('gold_medals.jpg', quality=50) **this number can be between 1 and 100
**svg file format will produce a vector graphics file where different elements can be edited using programs like Adobe
**dpi (dots per inch), the higher this number the more densely the image will be rendered
example
fig.savefig('gold_medals.png', dpi=300) **300 is relatively high resolution

Another aspect ration (ie size)
fig.set_size_inches([3,5])

Automating figures from data
Getting unique values of a column
example
sports = summer_2016_medals['Sport'].unique()
print(sports)
?say we didn't know how many sports were in the Olympics > we could devise a function to iterate over the DataFrame
example
fig, ax = plt.subplots()
for sport in sports:
    sport_df = summer_2016_medals[summer_2016_medals['Sport'] == sport]
    ax.bar(sport, sport_df['Height'].mean(),

yerr=sport_df[‘Height’].std())
ax.set_ylabel(‘’)
ax.set_xticklabels(‘’, rotation=)
plt.show()

Seaborn
example of a typical data analysis workflow
gather data> transform and clean> explore> analyze and build models> communicate results
works extremely well with pandas data structures
built on top of matplotlib
import seaborn as sns
**fun fact sns stands for Samuel Norman Seaborn a character from the West Wing television show

Create a count plot
import seaborn as sns
import matplotlib.pyplot as plt
gender = [‘Female’, ‘Female’, ‘Female’, ‘Female’, ‘Male’, ‘Male’, ‘Male’, ‘Male’}
sns.countplot(x=gender)
plt.show()
another example off of dataframe called “masculinity.csv” with alias df
sns.countplot(x=‘how_masculine’, data=df)
plt.show()

**key to clean seaborn charts is ‘tidy’ data
‘tidy’ data means each observation has its own row and each variable has its own column

Create a scatter plot
sns.scatterplot(x=‘total_bill’, y=‘tip’, data=tips, hue=’smoker’, hue_order=[‘yes’, ‘no’])
plt.show()
**using hue is using color to describe a third variable
**hue_order allows you to label the descriptors in the third variable
advancing the example - setting colors to our hue order
**create dictionary first
hue_colors = {‘yes’:’black’}, ‘no’:’red’}
sns.scatterplot(x=‘total_bill’, y=‘tip’, data=tips, hue=‘smoker’, palette=hue_colors)
plt.show()

Many questions in data science are centered around the relationship between two quantitative variables

Seaborn calls plots that visualize this relationship “relational plots”
relplot() stands for relational plot
enables you to visualize the relationship between two quantitative variables using either scatter or line plots
example using relplot instead of scatterplot
sns.relplot(x=’total_bill’, y=‘tip’, data=tips, kind=‘scatter’)
plt.show()
**this will make two plots
you can use col=‘’ to put them side by side or
row= to them on top of each other
can also use both if adding in an additional parameter
can also use col_wrap= to specify how many subplots you want per row
example
sns.relplot(x=‘total_bill’, y=‘tip’, data=tips, kind=‘scatter’, col=‘day’, col_wrap=2, col_order=[‘Thur’, ‘Fri’, ‘Sat’, ‘Sun’])
plt.show()

Customizations with seaborn scatter or relplots
showing relationship between two quantitative variables

- subplots (col and row)
- subgroups with color (hue)
- subgroups with point size and style
- changing point transparency

Subgroups with point size
sns.relplot(x=‘total_bill’,, y=‘tip’, data=tips, kind=’scatter’, size=‘size’, hue=’size’)
plt.show()
**in this example seaborn recognizes that size is a quantitative variable and automatically colors the points different shades of the same color instead of different colors per category value

Subgroups with point style
example
sns.relplot(x=‘total_bill’, y=‘tip’, data=tips, kind=‘scatter’, hue=‘smoker’, style=‘smoker’)
plt.show()
** this creates x for yes and dots for no

Subgroups with changing with point transparency
example
sns.relplot(x=’total_bill’, y=‘tip’, data=tips, kind=‘scatter’, alpha=0.4)

Line plots

another type of relational plots
difference between scatter and line
scatter - each plot point is an independent observation
line - each plot point represents the same ‘thing’, typically tracked over time
example
sns.relplot(x=‘hour’, y=‘NO_2_mean’, data=air_df_mean, kind=‘line’, style=‘location’, hue=‘location’, markers=True, dashes=False)
plt.show()
** if you don’t want the line styles to vary by subgroup, set the ‘dashes’ parameter to False
** line plots can also be used when you have more than one observation per x-value
in this example there is a row for each station that is taking a measurement every hour
the seaborn line plot aggregates them into a single summary measure (default is the mean)
will also automatically calculate a confidence interval for the mean (displayed by a shaded region)
Confidence intervals indicate the uncertainty we have about what the true mean is for the whole city
** we can visualize confidence intervals with parameter ‘ci’; ci=‘sd’ (ie shaded area is standard deviation)
standard deviation again shows the spread of the distribution of observations at each x value
** turn off confidence intervals by ci=None

Categorical plots
examples are count plots and bar plots and box plots and point plots
seaborn calls these two types of plots ‘categorical plots’
categorical plots involve a categorical variable, which is a variable that consists of a fixed, typically small number of possible values, or categories
these plots are commonly used when we want to make comparisons between different groups
count plot displays the number of observations in each category
we use catplot()
like relplot() gives us more flexibility then straight count or bar to create subplots

Count plots
example
sns.catplot(x=‘how_masculine’, data= masculinity _data, kind=‘count’)
plt.show()
** to create order make a list prior to calling catplot
example

category_order = [‘No answer’, ‘not at all’, ‘not very’, ‘somewhat’, ‘very’]
sns.catplot(x=, y=, data=, kind=, order=‘name of category list’

Bar plots
example
sns.catplot(x=‘day’, y=‘total_bill’, data=tips, kind=‘bar’)
plt.show()
** automatically shows confidence intervals

Box plots
example
sns.catplot(x=‘time’, y=‘total_bill’, data=tips, kind=‘box’, order=[‘Dinner’, ‘Lunch’], sym=“”)
plt.show()
** sym allows you to omit the outliers
** sym inside param is simply quotes
** by default remember that the whiskers extend out 1.5x the interquartile range
** you can change this standard by using the ’whis’ parameter
example
whis=2.0 (ie 2x the IQR)
or show the 5th and 95th percentiles whis=[5, 95]
or show min and max values whis=[0, 100]

Point plots
points show mean of quantitative variable
vertical lines show 95% confidence intervals
** differences between point plots and line plots
    -line plot has two quantitative variables (usually time on x-axis)
    -point plot has a categorical variable on x-axis
example
sns.catplot(x=‘age’, y=‘masculinity_important’, data=masculinity_data, hue=‘feel_masculine’, kind=‘point’, estimator=median, capsize=0.2, join=False)
plt.show()
** to display median over default mean
first make sure numpy is imported
remember median is usually a more accurate choice when there is a lot of outliers
** capsize customizes how the CIs are displayed; the number equals the width
** join removes the line joining each category

Seaborn customization
figure ’style’ includes background and axes
preset options: ‘white’, ‘dark’, ‘whitegrid’, ‘darkgrid’, ‘ticks’
these presets are set with sns.set_style()

**without grid just aiming for the audience to make high level observations
**grid may be useful if you want your audience to be able to determine the specific values of the plotted points

Changing the palette
figure 'palette' changes the color of the main elements of the plot
sns.set_palette()

Diverging palettes
great to use if your visualization deals with a scale where the two ends of the scale are opposites and there is a neutral midpoint
example of use to describe the survey of men who are grading the importance of masculinity
example
sns.set_palette('RdBu')
or for reverse
sns.set_palette('RdBu_r')

Sequential palettes
great for emphasizing a variable on a continuous scale
example of use depicting a car's horsepower and it's mpg where points grow larger and darker when the car has more cylinders
examples 'Greys', 'Blues', 'PuRd', 'GnBu'

Custom palettes
can create by making a list of color names or a list of hex color codes

Changing the scale
figure 'context' changes the scale of the plot elements and labels
sns.set_context()
scale options from smallest to largest: 'paper', 'notebook', 'talk' and 'poster'
default context is 'paper'

Adding titles and labels
**underlying mechanism of Seaborn in it's plot functions is to create two different types of objects: FacetGrid and AxesSubplots
**to figure out which type of object you're working with, first assign the plot output to a variable
example
g = sns.scatterplot(x='height', y='weight', data=df)
type(g)
this will return the object type

FacetGrid
consists of one or more AxesSubplots, which is how it supports subplots
relplot and catplot both support making subplots, which means that they are creating FacetGrid objects
in contrast single-type plot functions like scatterplot and count plot return a single AxesSubplot object

Adding a title to FacetGrid
g = sns.catplot()
g.fig.suptitle('New Title', y=)
** y adjust the title height; default=1

Titles for subplots
g.set_titles('This is {col_name}')
plt.show()

Adding axis labels
g.set(xlabel='New X Label', ylabel='New Y Label')
plt.show()

Rotating x-axis tick labels
plt.xticks(rotation=90)
plt.show()