Intermediate Data Science

Visualization

Joanna Bieri DATA201

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours take place in Duke 209 – Office Hours Schedule
- Class Website
- Syllabus

Plotting and Visualization

In Data101 we learned about the plotly package. Here are some resources to help remind you about visualizing data in plotly:

Plotting and Visualization

- Visualizing Numerical and Categorical Data
- Recoding and Visualizing Data
- Effective Visualization

Plotting and Visualization

Plotly is a great software package that produces interactive plots with lots of customization. But it is not the only way to create outstanding graphics.

Plotting and Visualization

We will see three new plotting packages today:

Matplotlib - link

Plotting and Visualization

We will see three new plotting packages today:

Matplotlib - link

Seaborn - link

Plotting and Visualization

We will see three new plotting packages today:

Matplotlib - link

Seaborn - link

Bokeh - link

Plotting and Visualization

I usually start with either matplotlib or plotly and then switch to other methods if needed as I start creating images for production or publication.

Install the packages as needed

```
!conda install -y seaborn
!conda install -y bokeh
```

Matplotlib

It is standared to import matplotlib.pyplot as plt. Try looking at the plt. packages to see what all is available!

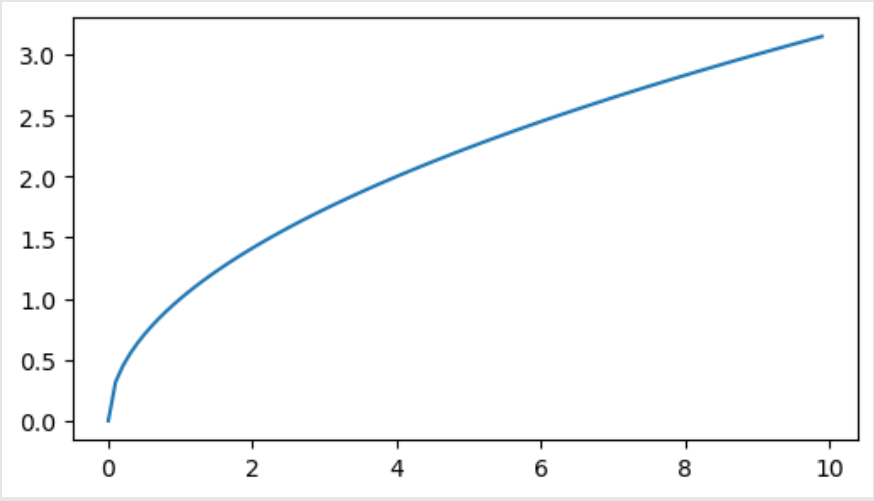
Matplotlib

Here is a minimal example:

```
# First create some data
x = np.arange(0,10,.1) # Choose your x-values
y = np.sqrt(x) # Get the y values - use a function

# Create the plot
plt.plot(x,y)
plt.show()
```

Matplotlib

Here is a minimal example:



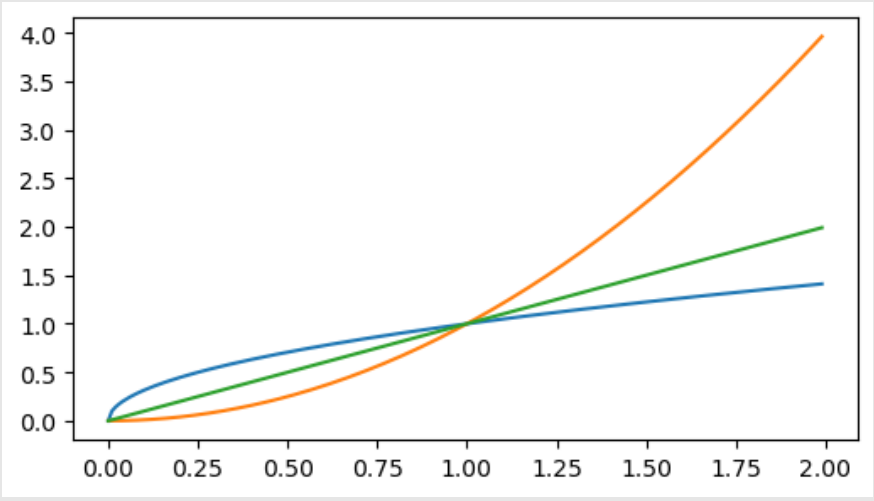
Matplotlib

An example with multiple lines

```
# First create some data
x = np.arange(0,2,.01) # Choose your x-values
y1 = np.sqrt(x) # Get the y values - use a function
y2 = x**2
y3 = x

# Create the plot
# Notice the lines are added to the same figure
# The colors and line style are automatic
plt.plot(x,y1)
plt.plot(x,y2)
plt.plot(x,y3)
plt.show()
```

Matplotlib

An example with multiple lines



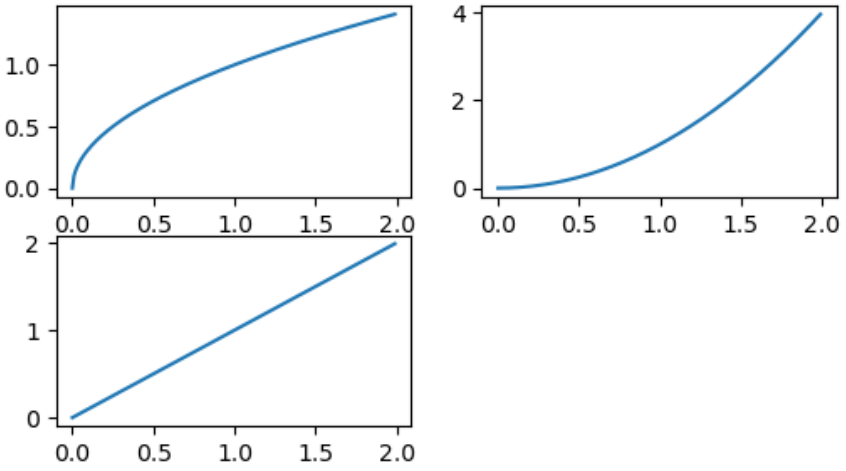
Matplotlib

Subplots - multiple plots in one figure

```
# Now create the figure object
fig = plt.figure()
# Now add some subplots
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
# Now put the data into the subplots
ax1.plot(x,y1)
ax2.plot(x,y2)
ax3.plot(x,y3)
plt.show()
```

Matplotlib

Subplots - multiple plots in one figure



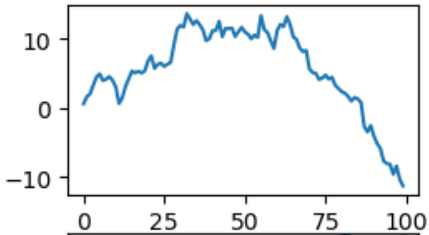
Matplotlib

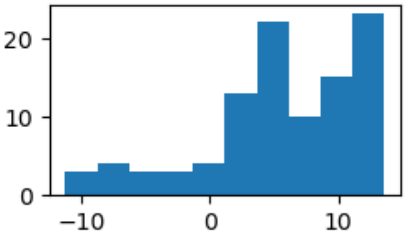
Other plot types

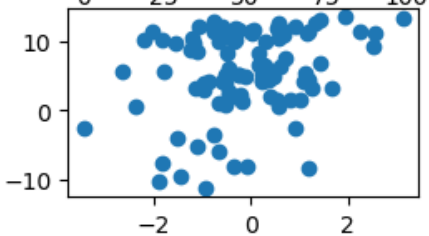
```
# Now create the figure object
fig = plt.figure()
# Now add some subplots
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)
# Now put the data into the subplots
# For some functions you don't need x and y
ax1.plot(y)
# Some functions only accept one input value to bin
ax2.hist(y)
# Some function require both x and y.
ax3.scatter(x,y)
plt.show()
```

Matplotlib

Other plot types







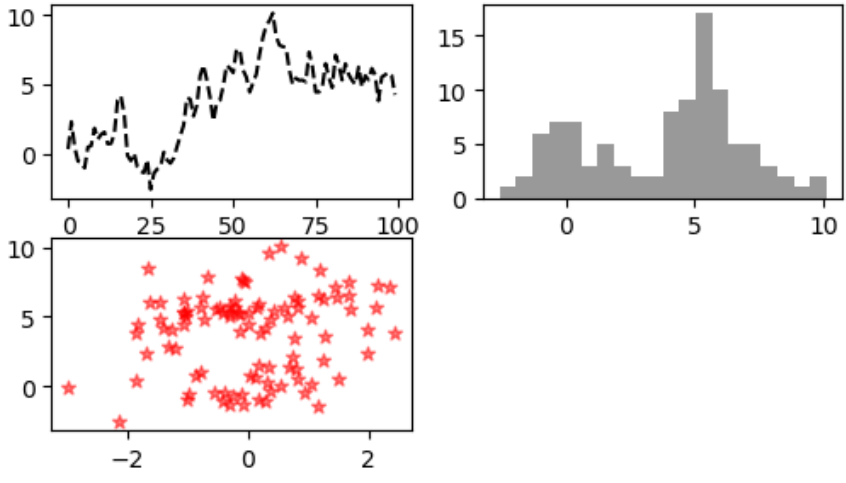
Matplotlib

Adding some line styles

```
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1) # This says 2x2 grid in location1
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3) # The subfigures go left to right
# Update the color and add dasked line
ax1.plot(y,color='black', linestyle='dashed')
# choose number of bins and make less opaque
ax2.hist(y,color='black',bins=20,alpha=0.4)
# change color and marker, make less opaque
ax3.scatter(x,y,color='red',marker='*',alpha=.5)
plt.show()
```

Matplotlib

Adding some line styles



Matplotlib

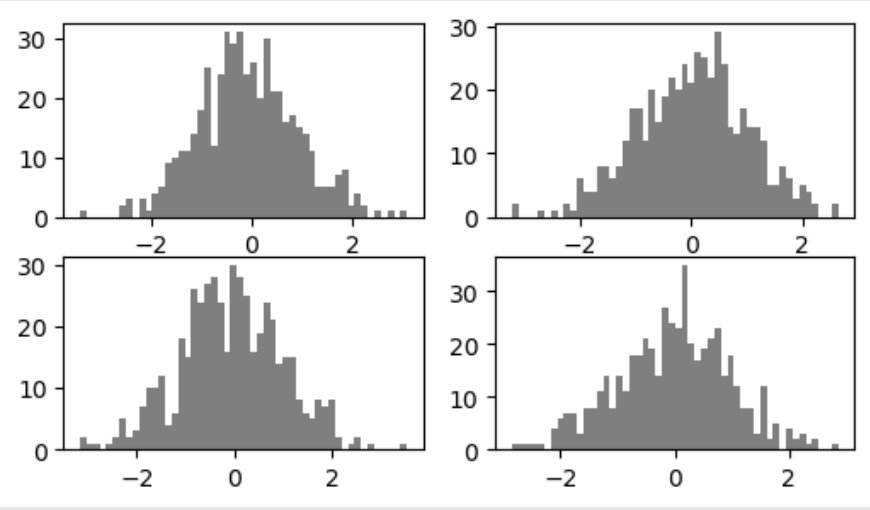
In this example we will see how to create subplots using the plt.subplots command, instead of specifying the axes independently. This is really useful when creating plots in a for loop.

More advanced subplots

```
# This command automatically creates all the axes
# We will do a 2x2 grid.
# Then add to each plot by calling axes[1,1], axes[1,2], ...
fig, axes = plt.subplots(2, 2)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.standard_normal(500), bins=5
                        color="black", alpha=0.5)
```

Matplotlib

More advanced subplots



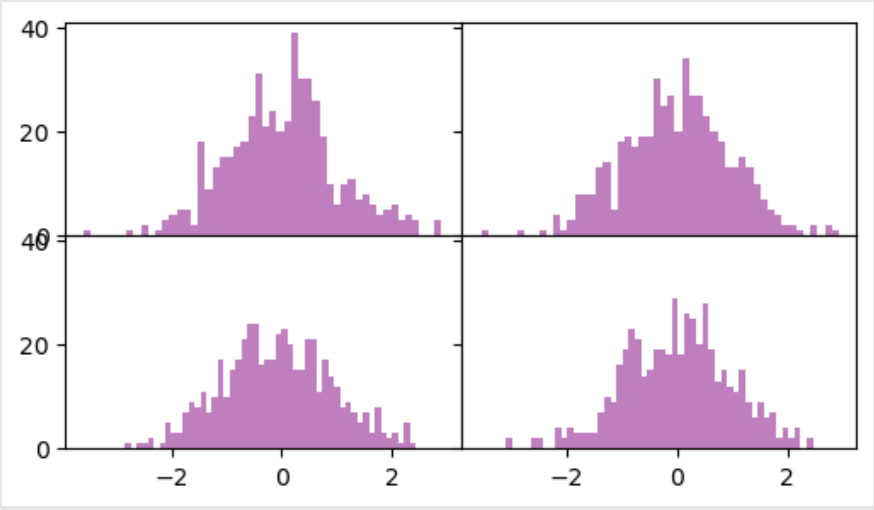
Matplotlib

Reduce White Space

```
# A small change to the code above reduces the white space
# and has all the plots use the same x and y-axis
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)
for i in range(2):
    for j in range(2):
        axes[i, j].hist(np.random.standard_normal(500), bins=5
                        color="purple", alpha=0.5)

# Remove white space
fig.subplots_adjust(wspace=0, hspace=0)
```

Matplotlib

Reduce White Space



Matplotlib - linestyles, markers, and colors

Here is a quick overview of the options available in matplot lib:

Matplotlib - linestyles, markers, and colors

Common Colors (Short Codes)

Code	Color
`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
`'m'`	magenta
`'y'`	yellow
`'k'`	black
`'w'`	white

Matplotlib - linestyles, markers, and colors

Full Named Colors

Matplotlib also supports full names like: - `'blue'`, `'green'`, `'red'`, `'orange'`, `'purple'`, `'brown'`, `'pink'`, `'gray'`, `'olive'`, `'navy'`, etc.
You can also use **hex codes**:

```
color = '#1f77b4'  # Matplotlib's default blue
```

Matplotlib - linestyles, markers, and colors

Line Styles

Code	Description
`'-'`	Solid line
`'--'`	Dashed line
`'-.'`	Dash-dot line
`':'`	Dotted line
`''` or `' '`	No line (useful for markers only)

Matplotlib - linestyles, markers, and colors

Marker Styles

Code	Marker
`'o'`	Circle
`'^'`	Triangle up
`'v'`	Triangle down
`'s'`	Square
`'D'`	Diamond
`'x'`	X
`'+'`	Plus
`'*'`	Star
`'.'`	Point

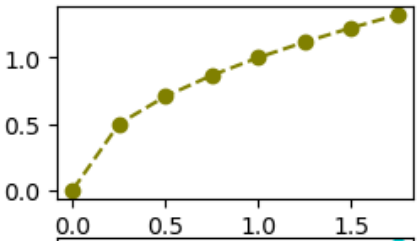
Matplotlib - linestyles, markers, and colors

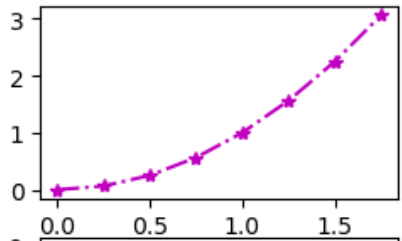
An example with colors!

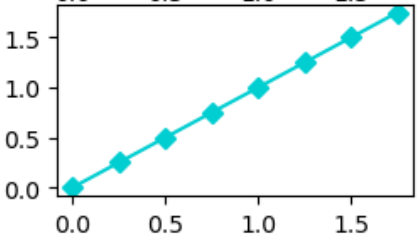
```
fig, axes = plt.subplots(2,2)
# Now put the data into the subplots
# Each one demonstrates a different way to add colors, lines,
axes[0,0].plot(x,y1,color='olive',linestyle='--', marker='o')
axes[0,1].plot(x,y2,'m-.*')
axes[1,0].plot(x,y3,color='#00CED1',marker='D')
axes[1,1].plot(x,y4,':')

plt.show()
```

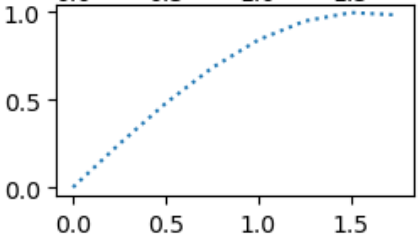
Matplotlib - linestyles, markers, and colors

An example with colors!



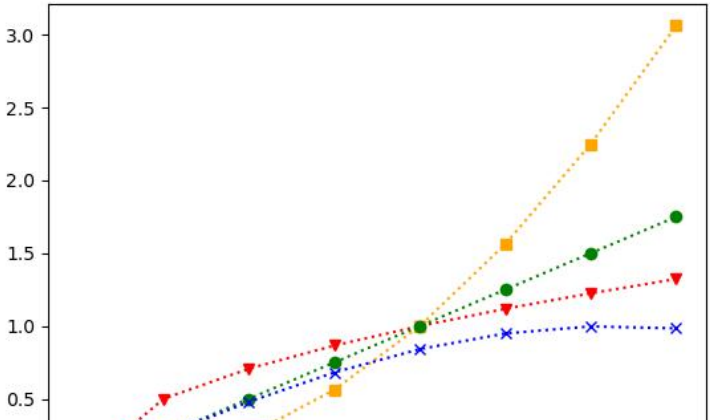






You Try

See if you can recreate the plot below. The functions used are the same as above.



Matplotlib - Ticks, Labels, and Legends

As you can see in the plots above, it becomes important to be able to add labels and legends to your plots. Matplotlib allows you to create plots with legends and other more fancy features!

Matplotlib - Ticks, Labels, and Legends

- You can use the `label=` command to label each item.

Matplotlib - Ticks, Labels, and Legends

- You can use the `label=` command to label each item.
- You can use `.grid()` to add a background grid to the plots

Matplotlib - Ticks, Labels, and Legends

- You can use the `label=` command to label each item.
- You can use `.grid()` to add a background grid to the plots
- The command `.legend()` adds the legend to each plot

Matplotlib - Ticks, Labels, and Legends

- You can use the `label=` command to label each item.
- You can use `.grid()` to add a background grid to the plots
- The command `.legend()` adds the legend to each plot
- You can set the ranges on the x and y-axes using `.xlim()` and `.ylim()`

Matplotlib - Ticks, Labels, and Legends

- You can use the `label=` command to label each item.
- You can use `.grid()` to add a background grid to the plots
- The command `.legend()` adds the legend to each plot
- You can set the ranges on the x and y-axes using `.xlim()` and `.ylim()`
- The commands `xticks()` and `yticks()` updates the markers on the x and y-axes

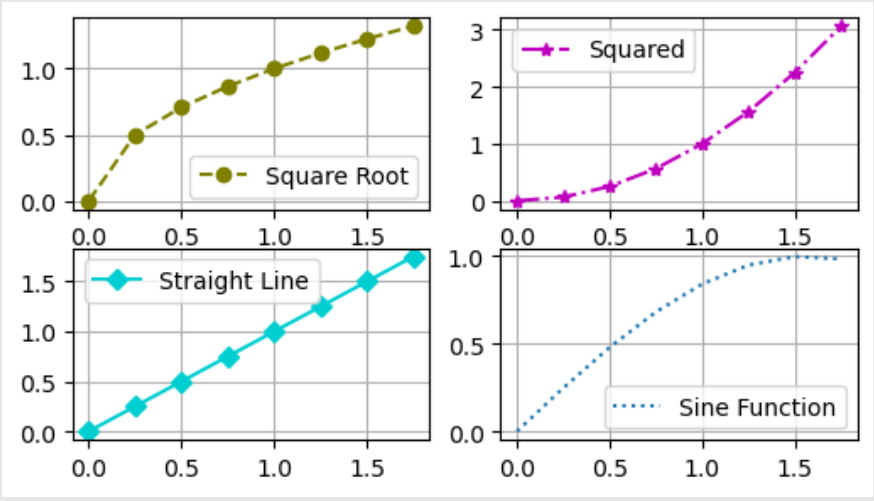
Matplotlib - Ticks, Labels, and Legends

Example

```
fig, axes = plt.subplots(2,2)
# The only change here is to add a label to each line
axes[0,0].plot(x,y1,color='olive',linestyle='--', marker='o',l
axes[0,1].plot(x,y2,'m-.*',label='Squared')
axes[1,0].plot(x,y3,color='#00CED1',marker='D',label='Straight
axes[1,1].plot(x,y4,':',label='Sine Function')

# Then add the legend and grid in a for loop
# Here axes.flat is a 1D iterator over all the subplot Axes ob
for ax in axes.flat:
    ax.grid()
    ax.legend()
```

Matplotlib - Ticks, Labels, and Legends

Example



Matplotlib - Ticks, Labels, and Legends

Example

```
fig, axes = plt.subplots(2,2)
# The only change here is to add a label to each line
axes[0,0].plot(x,y1,color='olive',linestyle='--',
               marker='o',label='Square Root')
axes[0,1].plot(x,y2,'m-.*',label='Squared')
axes[1,0].plot(x,y3,color='#00CED1',marker='D',
               label='Straight Line')
axes[1,1].plot(x,y4,':',label='Sine Function')

# Then add the legend and grid in a for loop
# Here axes.flat is a 1D iterator over all the subplot Axes ob
for ax in axes.flat:
    ax.grid()
    ax.legend()
```

Matplotlib - Ticks, Labels, and Legends

Example

```
plt.plot(x,y1,'m-o')
# Change the limits
plt.xlim([0,2])
plt.ylim([0,2])
# Add a grid
plt.grid()
```

Matplotlib - Ticks, Labels, and Legends

Example - continued

```
# Change what is on the axes
xtick_positions = [0, 0.5, 1, 1.5, 2]
xtick_labels = ['zero', 'half', 'one', 'one & half', 'two']
plt.xticks(xtick_positions, xtick_labels,
           rotation=30,fontsize=10)

ytick_positions = [0,0.75,1.50]
ytick_labels=['min','middle','max']
plt.yticks(ytick_positions,ytick_labels,fontsize=8)
```

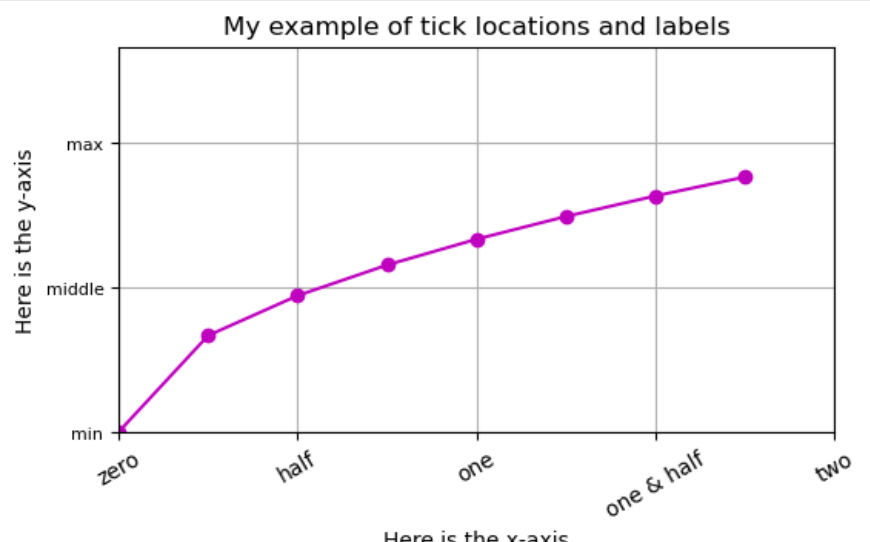
Matplotlib - Ticks, Labels, and Legends

Example - continued

```
# Add a title and labels
plt.title('My example of tick locations and labels')
plt.xlabel('Here is the x-axis')
plt.ylabel('Here is the y-axis')
```

Matplotlib - Ticks, Labels, and Legends

Example - continued



Matplotlib - Adding Annotations

Sometimes you want to add text to your plot that helps you point out important aspects of the data. This can be done by adding annotations. This example will walk us through a few new ideas:

- Using `datetime` objects in python. These represent a specific point in time — including the year, month, day, hour, minute, second, microsecond, and optionally a time zone.
- Calling `.plot` directly on a pandas series object
- Adding annotations from a list

Matplotlib - Adding Annotations

Example with annotations

```
from datetime import datetime

# Read in the data using pandas
data = pd.read_csv("data/spx.csv",
                index_col=0, parse_dates=True)
# Get just the SPX column - this is a series object
spx = data["SPX"]

# Call .plot on this object and send in optional commands
spx.plot(color="red",linewidth=.5)
```

Matplotlib - Adding Annotations

Example with annotations - continued

```
# Now we will hard code some events that take place
# over time
# datetime tells python that this is a data
# and should be ordered that way
# This is a list of tuples
crisis_data = [
    (datetime(2007, 10, 11), "Peak of bull market"),
    (datetime(2008, 3, 12), "Bear Stearns Fails"),
    (datetime(2008, 9, 15), "Lehman Bankruptcy")
]
```

Matplotlib - Adding Annotations

Example with annotations - continued

```
# Now cycle through the events
for date, label in crisis_data:
    # Add an annotation for each
    # label is the words you want to add
    # xy= is the (x,y) location of pointer end
    # xytext= is the (x,y) location of the words
    # arrowprops= lets you set arrow properties
    plt.annotate(label, xy=(date, spx.asof(date) + 75),
        xytext=(date, spx.asof(date) + 225),
        arrowprops=dict(facecolor="black",
                        headwidth=4, width=1,
                        headlength=4),
        horizontalalignment="left", verticalalignment="top")
```

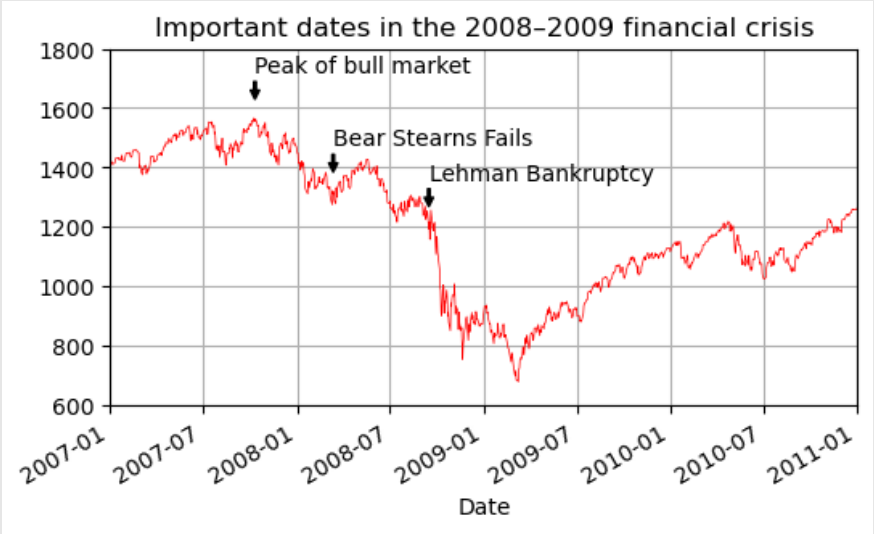
Matplotlib - Adding Annotations

Example with annotations - continued

```
# Set the x and y limits to zoom in on 2007-2010
plt.xlim(["1/1/2007", "1/1/2011"])
plt.ylim([600, 1800])

plt.title("Important dates in the 2008-
2009 financial crisis")
plt.grid()

plt.show()
```

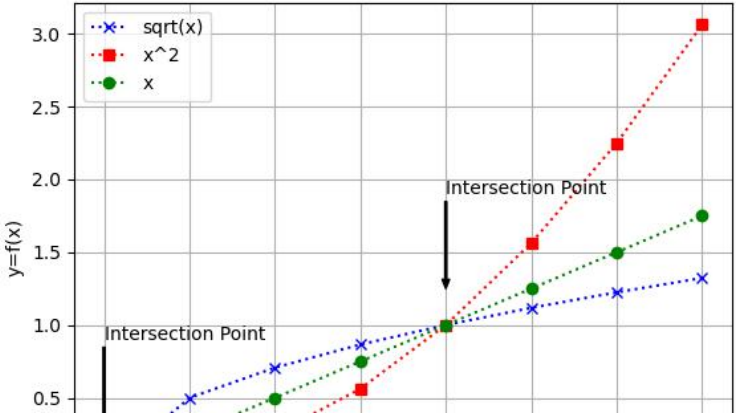
Matplotlib - Adding Annotations

Example with annotations



You Try

Now using what you know about annotations and labels. See if you can recreate the plot with the data given below.



Matplotlib - bar plot

Here is an example of a bar plot. What I want you to learn here is that the basic syntax is always the same! Once you know the structure of matplotlib you can explore all sorts of plots.

Example bar plot

```
# Example data
categories =
    ["Math", "Science", "History", "English", "Art"]
# Create x positions for bars
# This creates x = [0,1,2,3,4]
# as a place holder for the x-labels
x = np.arange(len(categories))
yvalues = [85, 92, 78, 88, 95]
```

Matplotlib - bar plot

Example bar plot - continued

```
plt.bar(
    x,
    yvalues,
    color="skyblue",    # change bar color
    edgecolor="black", # add edge color
    linewidth=1.5,     # thickness of edges
    hatch="/",         # pattern fill
    alpha=0.8,         # transparency
                       # (0=transparent, 1=opaque)
    width=0.6,         # width of bars
    align="center"     # alignment: 'center'
)
```

Matplotlib - bar plot

Example bar plot - continued

```
# Add labels, title, and ticks
plt.title("Student Test Scores by Subject",
            fontsize=16,
            fontweight="bold")
plt.xlabel("Subjects", fontsize=12)
plt.ylabel("Scores", fontsize=12)

# Change the ticks and categories on the axis
plt.xticks(x, categories, rotation=30, fontsize=10)
# Have y-ticks be every 10
plt.yticks(np.arange(0, 101, 10))
```

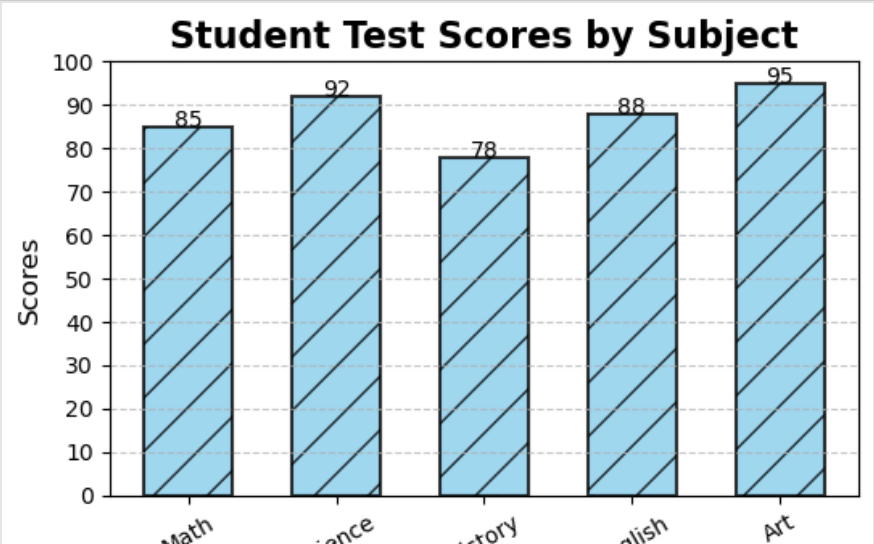
Matplotlib - bar plot

Example bar plot - continued

```
# Add grid lines to only the y-axis
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Annotate
for i, v in enumerate(yvalues):
    plt.annotate(str(yvalues[i]), xy=(x[i], v + 4),
                 horizontalalignment='center',
                 verticalalignment="top")

plt.show()
```

Matplotlib - bar plot

Example bar plot



Matplotlib - more crazy examples

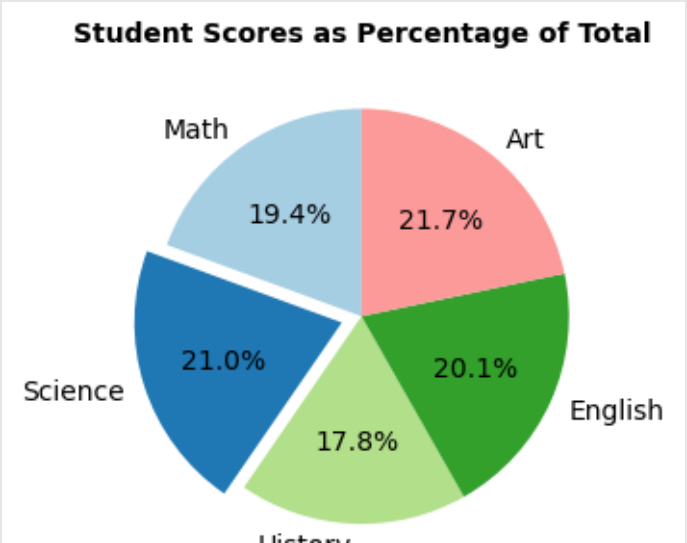
Just for fun!

Example Pie

```
plt.pie(
    yvalues, labels=categories,
    autopct="%1.1f%%", startangle=90,
    colors=plt.cm.Paired.colors,
    explode=[0, 0.1, 0, 0, 0]  # emphasize Science
)
plt.title("Student Scores as Percentage of Total",
            fontsize=14,
            fontweight="bold")
plt.show()
```

Matplotlib - more crazy examples

Example Pie



Matplotlib - more crazy examples

Example Spider

```
# LOOP THE DATA
from math import pi

# For this type of plot you need to deal with angles
# We do this in radians
N = len(categories)
# Make the values loop back around so the first
# and last are the same
values_loop = yvalues + [yvalues[0]]
# Create the right number of angles to match
# the number of values
# Then add zero on the end to loop back around
angles = [n / float(N) * 2 * pi for n in range(N)] + [0]
```

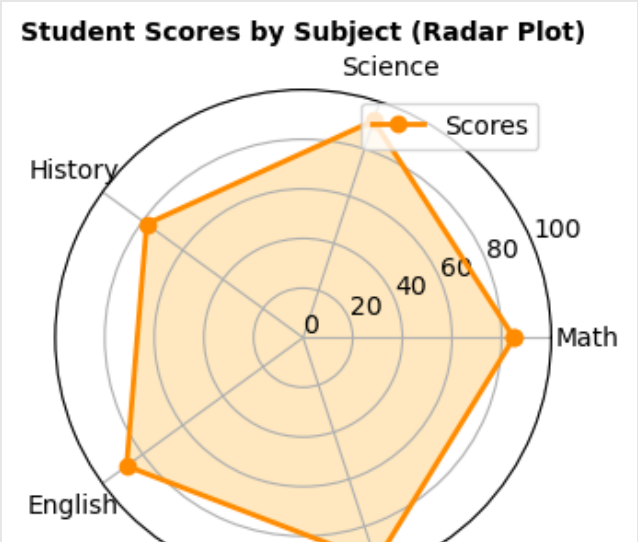
Matplotlib - more crazy examples

Example Spider - continued

```
plt.polar(angles, values_loop, "o-", linewidth=2,
            label="Scores", color="darkorange")
# Fill inside the lines
plt.fill(angles, values_loop, alpha=0.25, color="orange")
# Update the ticks
plt.xticks(angles[:-1], categories)
plt.yticks(range(0, 101, 20))
plt.title("Student Scores by Subject (Radar Plot)",
            fontsize=14, fontweight="bold")
# Move the legend
plt.legend(loc="upper right")
plt.show()
```

Matplotlib - more crazy examples

Example Spider



Matplotlib - Saving and Configuration

If you want to save a figure that you have created you need to add

```
plt.savefig('figurename.jpg')
```

BEFORE you do plt.show().

Matplotlib - Saving and Configuration

You can customize the size of your plot

```
plt.rc('figure', figsize=(10,10))
```

And to go back to default

```
plt.rcdefaults()
```

Matplotlib - Saving and Configuration

There are LOTS of other options that you can take advantage of!

Pandas Plotting

There are default plotting options that leverage matplotlib as part of the pandas package. You can see our book pp.298-310 for examples. I tend to use matplotlib directly or plotly more than pandas, but some peple find it very convenient.

Pandas Plotting

Basic plot methods

- `df.plot()` – general plotting interface (line by default)
- `df.plot.line()` – line plots
- `df.plot.bar()` – vertical bar plots
- `df.plot.barh()` – horizontal bar plots
- `df.plot.hist()` – histograms
- `df.plot.box()` – box-and-whisker plots

Pandas Plotting

Basic plot methods

- `df.plot.area()` – stacked area plots
- `df.plot.scatter(x=..., y=...)` – scatter plots
- `df.plot.hexbin(x=..., y=...)` – hexagonal binning plot
- `df.plot.density()` / `df.plot.kde()` – kernel density estimate plots
- `df.plot.pie()` – pie charts (usually with a Series)

Seaborn

Here I will give a VERY quick overview of some ways that you might use seaborn. It has some really handy, and beautiful visualization packages that are more specific to statistical analysis.

Seaborn

Start by looking at some macroeconomic data.

	cpi	m1	tbilrate	unemp
0	28.980	139.7	2.82	5.8
1	29.150	141.7	3.08	5.1
2	29.350	140.5	3.82	5.3
3	29.370	140.0	4.33	5.6
4	29.540	139.6	3.50	5.2
...	...	...	...	...
198	216.889	1474.7	1.17	6.0
199	212.174	1576.5	0.12	6.9
200	212.671	1592.8	0.22	8.1
201	214.469	1653.6	0.18	9.2
202	216.385	1673.9	0.12	9.6

Seaborn - data

- cpi - Consumer Price Index
- m1 - Money Supply (M1)
- tbilrate - Treasury Bill Rate
- unemp - Unemployment Rate

Seaborn - data

Often with data we want to look at the log of the data

Taking the log can:

- make growth rates easier to interpret
- stabilize variance
- linearizes relationships
- make distributions closer to normal

Seaborn - data

	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

Seaborn - regplot()

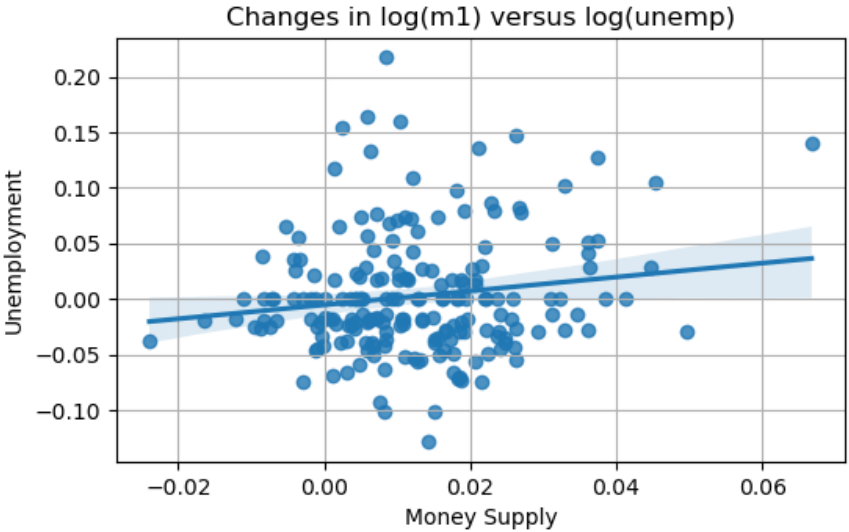
Now we can look at a scatter plot of the money supply vs the unemployment rate and add a linear regression line with 95% confidence interval around the fitted regression

Example regplot

```
ax = sns.regplot(x="m1", y="unemp", data=trans_data)
ax.set_title("Changes in log(m1) versus log(unemp)")

# You can add standard matplotlib style commands
ax.grid()
ax.set_xlabel('Money Supply')
ax.set_ylabel('Unemployment')
```

Seaborn - regplot()

Example regplot



Seaborn - pairplot

A seaborn pairplot gives a quick multivariate overview of the variables in your data set. You can see how each numerical variable varies against every other one and see the single variable distribution of individual variables (histograms or KDEs). This lets you very quickly look for correlations and interesting aspects of your data (like outliers).

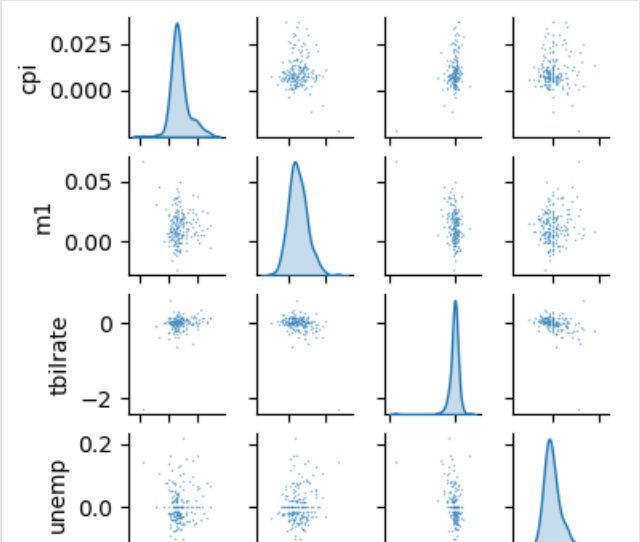
Seaborn - pairplot

Example

```
sns.pairplot(trans_data,
             diag_kind="kde",
             height=1,
             aspect=1,
             plot_kws={"alpha": 0.5,'s': 1})

plt.show()
```

Seaborn - pairplot

Example



Bokeh plots

The Bokeh packages allows you to create more interactive plots. While this is not necessary for exploratory data analysis, it can be a great way to allow your audience to interact with your data. I am not going to do a full tutorial here, but just show an example so you can see what Bokeh has to offer.

Bokeh plots

There are lots of tutorials online if you want to learn more!

On the side of the figure you can choose the tools that are available.

Lecture notes contain a list of the possible tools, markers, colors.

Bokeh plots

Imports

```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import HoverTool
```

Bokeh plots

Example

```
# This tells bokeh to output it's code to
# the jupyter notebook.
# the default is to create an html file
# and show the plot in a new browser window.
output_notebook()

# Create some data
x = np.arange(0,10,0.2)
y = np.sin(x)
```

Bokeh plots

Example - continued

```
# Create figure - this is calling the bokeh.plotting
# figure function
p = figure(
    title="Interactive Sine Wave",
    x_axis_label="X",
    y_axis_label="sin(X)",
    tools="pan,wheel_zoom,box_zoom,reset,save"
)
```

Bokeh plots

Example - continued

```
# Create a scatter plot of the data
# Notice that the "feel" is very matplotlib
# with some slight variations.
p.scatter(
    x, y,
    size=8,
    marker="circle",   # could be "square", "triangle",...
    color="navy",
    alpha=0.6,
    legend_label="sin(x)"
)
```

Bokeh plots

Example - continued

```
# Add line to connect the scatter plot points
p.line(x, y, line_width=2, color="orange", alpha=0.7)
# Add a mouse hover tool and tell it what data to show
hover = HoverTool(tooltips=[("x", "@x"), ("y", "@y")])
p.add_tools(hover)
# Show the plot you have created
show(p)
```

Bokeh plots

Example

Link to Bokeh Plot