Sheikh-Sedat Touray

DSP 562

Spring 2024

Problem Set 4 (7.5 points)

Data Visualization in Python, using matplotlib, seaborn and bokeh

Week 4 References: McKinney, Chapter 9

https://docs.bokeh.org (https://docs.bokeh.org)

In this Problem Set you simply need to execute the code (in order), retain your output, and submit the output-containing notebook.

Plotting and Visualization

Matplotlib

```
In [64]: # Importing libraries and setting up display options below
```

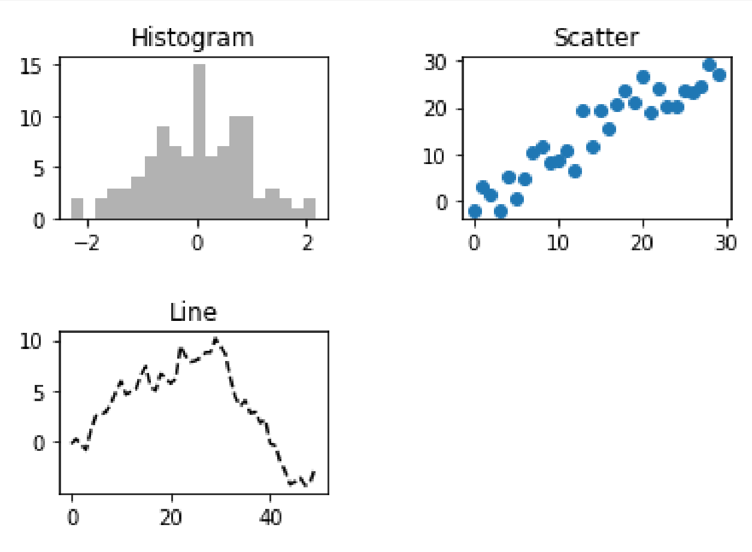
```
In [2]: #! ipython suppress id=61c3e93e7da045348bdcf837eeeb1ed6
        #%pushd book-materials
        import numpy as np
        import pandas as pd
        PREVIOUS_MAX_ROWS = pd.options.display.max_rows
        pd.options.display.max_rows = 20
        pd.options.display.max_colwidth = 80
        pd.options.display.max_columns = 20
        np.random.seed(12345)
        import matplotlib.pyplot as plt
        import matplotlib
        plt.rc("figure", figsize=(10, 6))
        np.set_printoptions(precision=4, suppress=True)
```

Figures and Subplots

"One nuance of using Jupyter notebooks is that plots are reset after each cell is evaluated, so for more complex plots you must put all of the plotting commands in a single notebook cell." - McKinney

1

```
In [3]: #Create a new figure
        fig = plt.figure()
        #now we add our subplots with add_subplot() function
        # we want the figure to be 2x2
        # Meaning a maximum of 2 plots horizontally and vertically
        ax1 = fig.add_subplot(2, 2, 1)
        ax2 = fig.add_subplot(2, 2, 2)
        ax3 = fig.add_subplot(2, 2, 3)

        # Add titles to subplots
        ax1.set_title('Histogram')
        ax2.set_title('Scatter')
        ax3.set_title('Line')

        # Adjust spacing around subplots
        plt.subplots_adjust(wspace=0.5, hspace=0.7)

        #using the plot axis method we are going to create a line plot
        # We want the style of the line to be dashes
        ax3.plot(np.random.standard_normal(50).cumsum(), color="black",
                 linestyle="dashed")
        # We create a histogram using the plot axis method
        # setting the transparency of the plot with the style alpha=0.3
        ax1.hist(np.random.standard_normal(100), bins=20, color="black", alpha=0.3);

        #! figure,id=mpl_subplots_two,width=4in,title="Data visualization after additional plots"
        #We create a Scatterplot using the plot axis method
        ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.standard_normal(30));
```

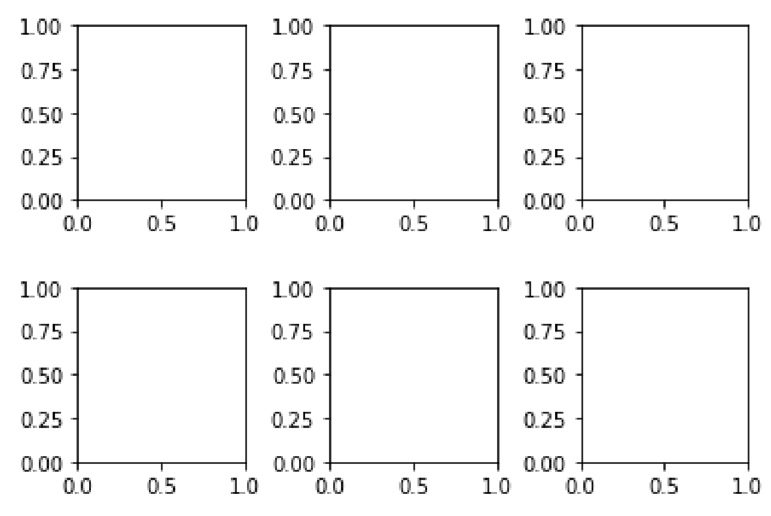


```
In [4]: #! ipython suppress id=9ca014254f154d0b8b9f9fdae36a04e4
        #Resets the windows
        plt.close("all")
```

Adjusting the spacing around subplots

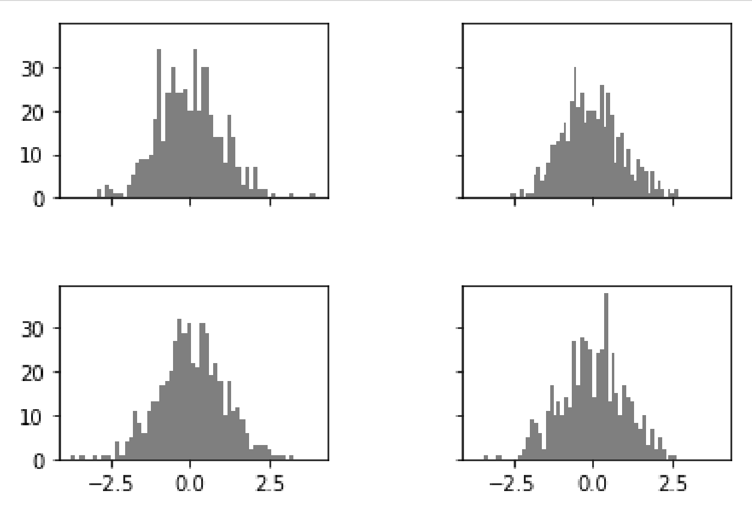
2

```
In [5]: #! ipython id=a38b9f8424f348ebb559a7945b609121
        fig, axes = plt.subplots(2, 3)

        # Adjust spacing around subplots
        plt.subplots_adjust(wspace=0.5, hspace=0.5)

        # Show the plot
        axes;
```



3

```
In [6]: #! ipython suppress id=56bf61e88f414933be21cfca7bb7a89e
        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=50,
                                color="black", alpha=0.5)
        #! figure,id=mpl_subplots_adjust,width=4in,title="Data visualization with no inter-subplot spacing"
        # Adding this line of code provided the seperation
        # To see each of these plots as individual plots
        fig.subplots_adjust(wspace=0.5, hspace=0.5)
```

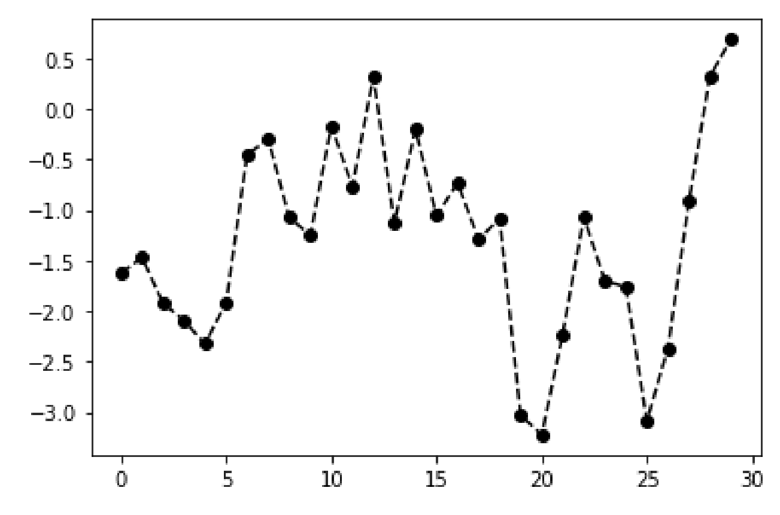


In a case like I was able to seperate the plots by adjusting the wspace and hspace. However, it may also be solved specifying the explicit tick locations and labels which we shall discuss further in this chapter.

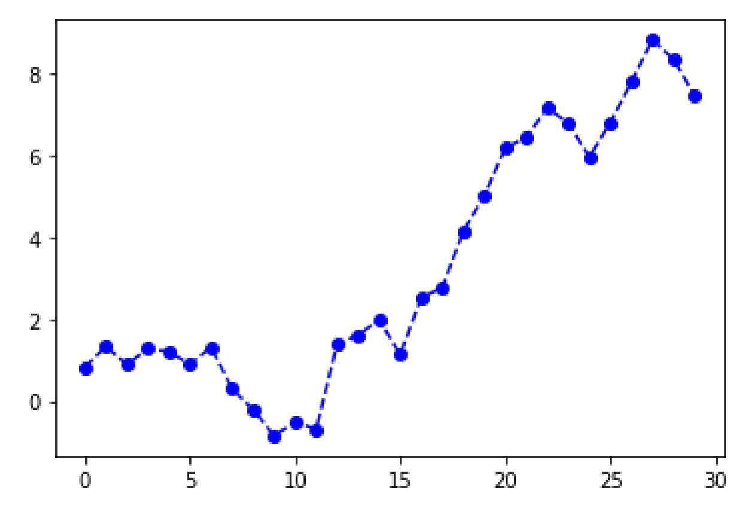
Colors, Markers, and Line Styles

4

```
In [7]: #! ipython id=7c2ae213e9214fe0a8780d91a64ed273
        fig = plt.figure()
        ax = fig.add_subplot()
        #! figure,id=mpl_marker_ex,width=4in,title="Line plot with markers"
        #This below plots a line with markers and colors
        ax.plot(np.random.standard_normal(30).cumsum(), color="black",
                linestyle="dashed", marker="o");
```



```
In [8]: # to better view this problem
        # I may update the color in the code
        fig = plt.figure()
        ax = fig.add_subplot()
        #! figure,id=mpl_marker_ex,width=4in,title="Line plot with markers"
        #This below plots a line with markers and colors
        ax.plot(np.random.standard_normal(30).cumsum(), color="b",
                linestyle="dashed", marker="o");
```



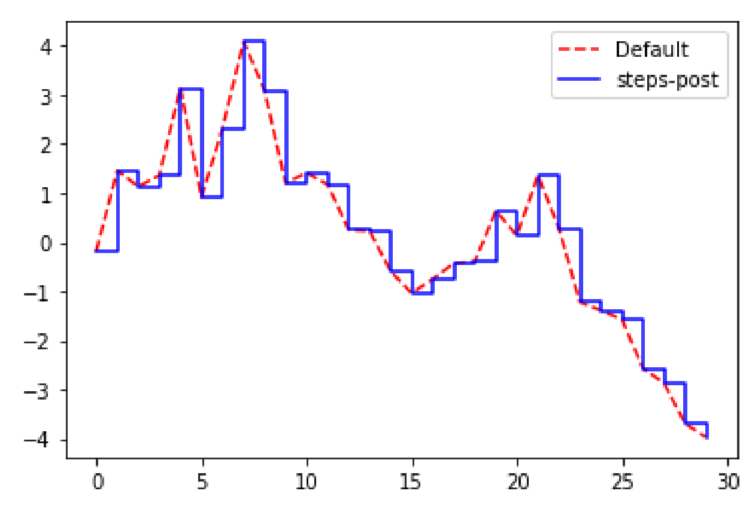
```
In [9]: #! ipython suppress id=5e0afabc7ebd43d383a0c42d2188d9af
        plt.close("all")
```

Ticks, Labels, and Legends

Setting the title, axis labels, ticks, and ticklabels

5

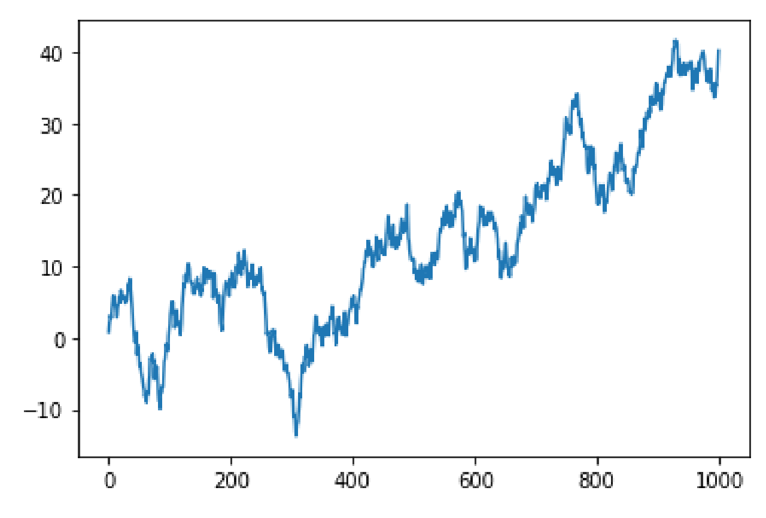
```
In [11]: #! ipython id=600cec64d3e842749a84b16b8a549145
         fig = plt.figure()
         ax = fig.add_subplot()
         data = np.random.standard_normal(30).cumsum()
         ax.plot(data, color="red", linestyle="dashed", label="Default");
         ax.plot(data, color="b", linestyle="-",
                 drawstyle="steps-post", label="steps-post");
         #! figure,id=mpl_drawstyle_ex,width=4in,title="Line plot with different drawstyle options"
         #ax.lengend method
         ax.legend();
```



Intially it was too difficult to distinguish between the steps and default lines even with the legend printed because they were both dashed. So I changed the style of the steps with a solid line and I changd both colors to show their difference.

6

```
In [12]: #! ipython id=bac0ed2c52a04e62aad3cab21801a44c
         fig, ax = plt.subplots()
         #! figure,id=vis_ticks_one,width=3.5in,title="Simple plot for illustrating xticks (with default labels)"
         ax.plot(np.random.standard_normal(1000).cumsum());
```



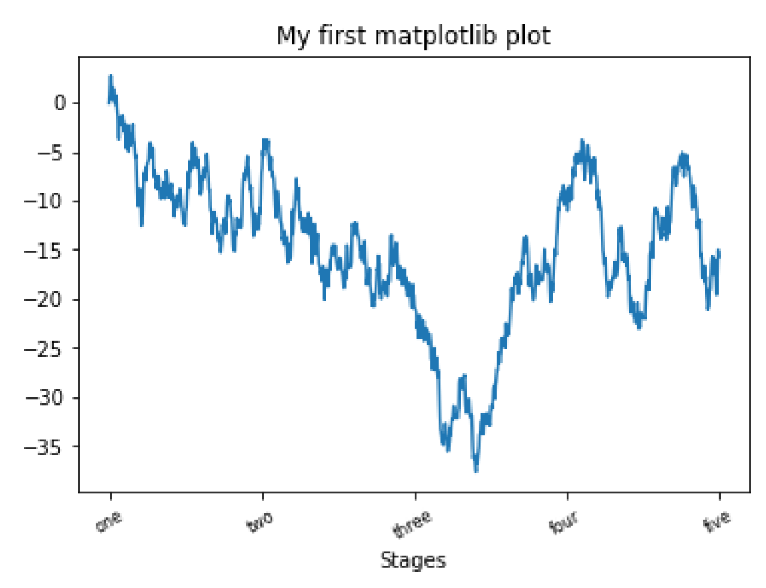
Inorder for us the control the plot range, tivk locations, and labels, we can add methods like xlim, xticks and xtick labels. And they can be used in two ways:

Called with no arguments returns the current parameter value (e.g., ax.xlim() returns the current x-axis plotting range)

Called with parameters sets the parameter value (e.g., ax.xlim([0,100]) sets the x-axis range from 0 to 100)

7

```
In [13]: #! ipython id=08aa54cdf68c46fa8137f4c30189fb7d
         fig, ax = plt.subplots()
         #! figure,id=vis_ticks_one,width=3.5in,title="Simple plot for illustrating xticks (with default labels)"
         ax.plot(np.random.standard_normal(1000).cumsum());
         ticks = ax.set_xticks([0, 250, 500, 750, 1000])
         labels = ax.set_xticklabels(["one", "two", "three", "four", "five"],
                                     rotation=30, fontsize=8)
         #! ipython id=933a74fa372e40349697f0c32e50f259
         # Setting the labels as stages
         ax.set_xlabel("Stages")
         #! figure,id=vis_ticks_two,width=3.5in,title="Simple plot for illustrating custom xticks"
         #Defining the title with the set_title function
         ax.set_title("My first matplotlib plot");
```



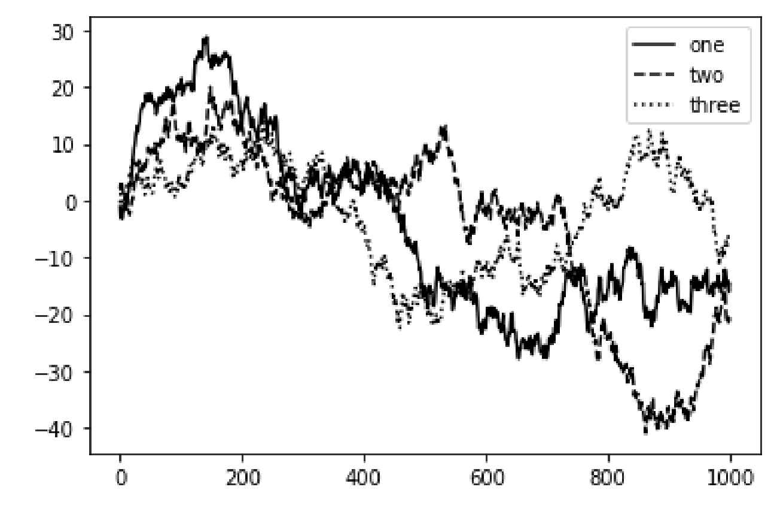
I added a semicolon at the end of the code to get rid of the warning.

Adding legends

Legends are another critical element for identifying plot elements. There are a couple of ways to add one. The easiest is to pass the label argument when adding each piece of the plot:

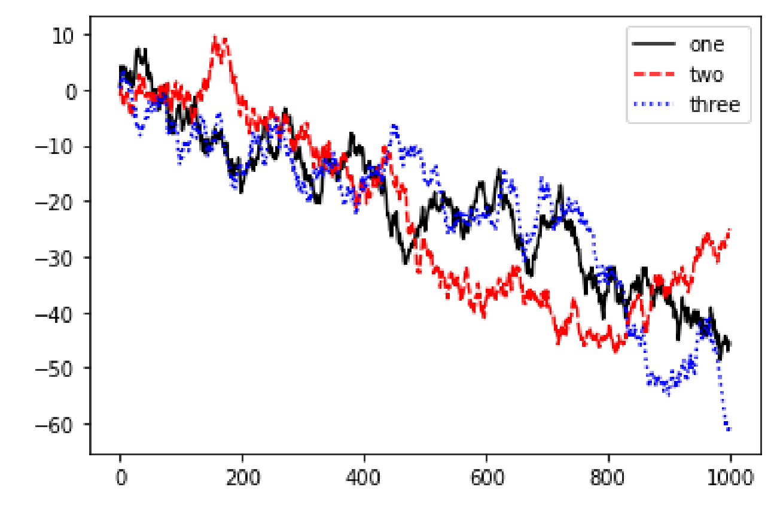
8

In [14]:
```python
#! ipython id=b89655b6b64e49adb709a8dcaa4b1fec
fig, ax = plt.subplots()
ax.plot(np.random.randn(1000).cumsum(), color="black", label="one");
ax.plot(np.random.randn(1000).cumsum(), color="black", linestyle="dashed",
        label="two");
ax.plot(np.random.randn(1000).cumsum(), color="black", linestyle="dotted",
        label="three");
#adds a legend at the best location
ax.legend(loc='best');
```



In the plot above has different line styles and a legend has already been added but it does not show much differentiation in terms of visualization. It would be very difficult to tell them apart.

In [15]:
```python
#updating the line colors
fig, ax = plt.subplots()
ax.plot(np.random.randn(1000).cumsum(), color="black", label="one");
ax.plot(np.random.randn(1000).cumsum(), color="r", linestyle="dashed",
        label="two");
ax.plot(np.random.randn(1000).cumsum(), color="b", linestyle="dotted",
        label="three");
ax.legend(loc='best');
```

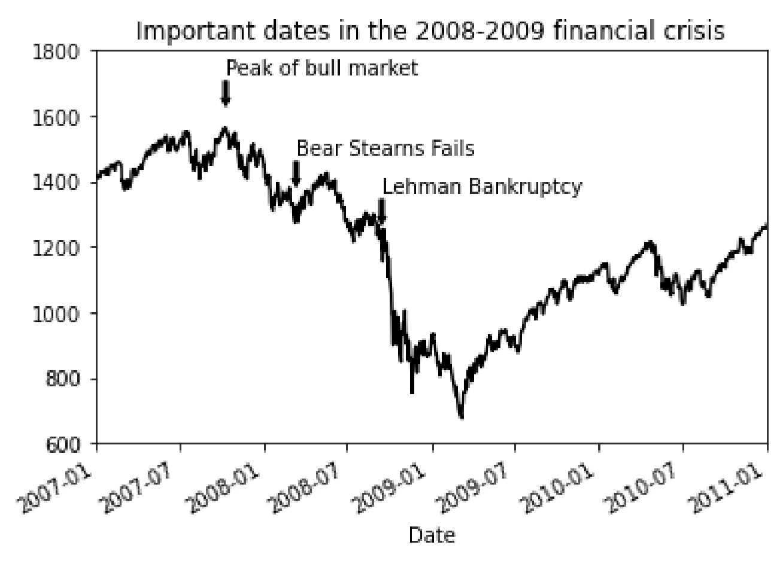


It is easier to visualize and read these three different labels

Annotations and Drawing on a Subplot

9

In [16]:
```python
#! ipython verbatim id=00b59126f3364fdf900df06d486ed23d
#importing the date and time library
from datetime import datetime

#Creating a new figure and plot
fig, ax = plt.subplots()

#Read in the data as a pandas dataframe
data = pd.read_csv("spx.csv", index_col=0, parse_dates=True)
#storing the spx column to the spx variable
spx = data["SPX"]

#plot the SPX
spx.plot(ax=ax, color="black")

#creating a list to be annotated on the plot
crisis_data = [
    (datetime(2007, 10, 11), "Peak of bull market"),
    (datetime(2008, 3, 12), "Bear Stearns Fails"),
    (datetime(2008, 9, 15), "Lehman Bankruptcy")
]

#Use a for loop to specify the location of the annotations
for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
                xytext=(date, spx.asof(date) + 225),
                arrowprops=dict(facecolor="black", headwidth=4, width=2,
                                headlength=4),
                horizontalalignment="left", verticalalignment="top")

# Zoom in on 2007-2010
ax.set_xlim(["1/1/2007", "1/1/2011"])

#zoom in on 600-1800 on the y axis
ax.set_ylim([600, 1800])

# Adds title
ax.set_title("Important dates in the 2008-2009 financial crisis");
```

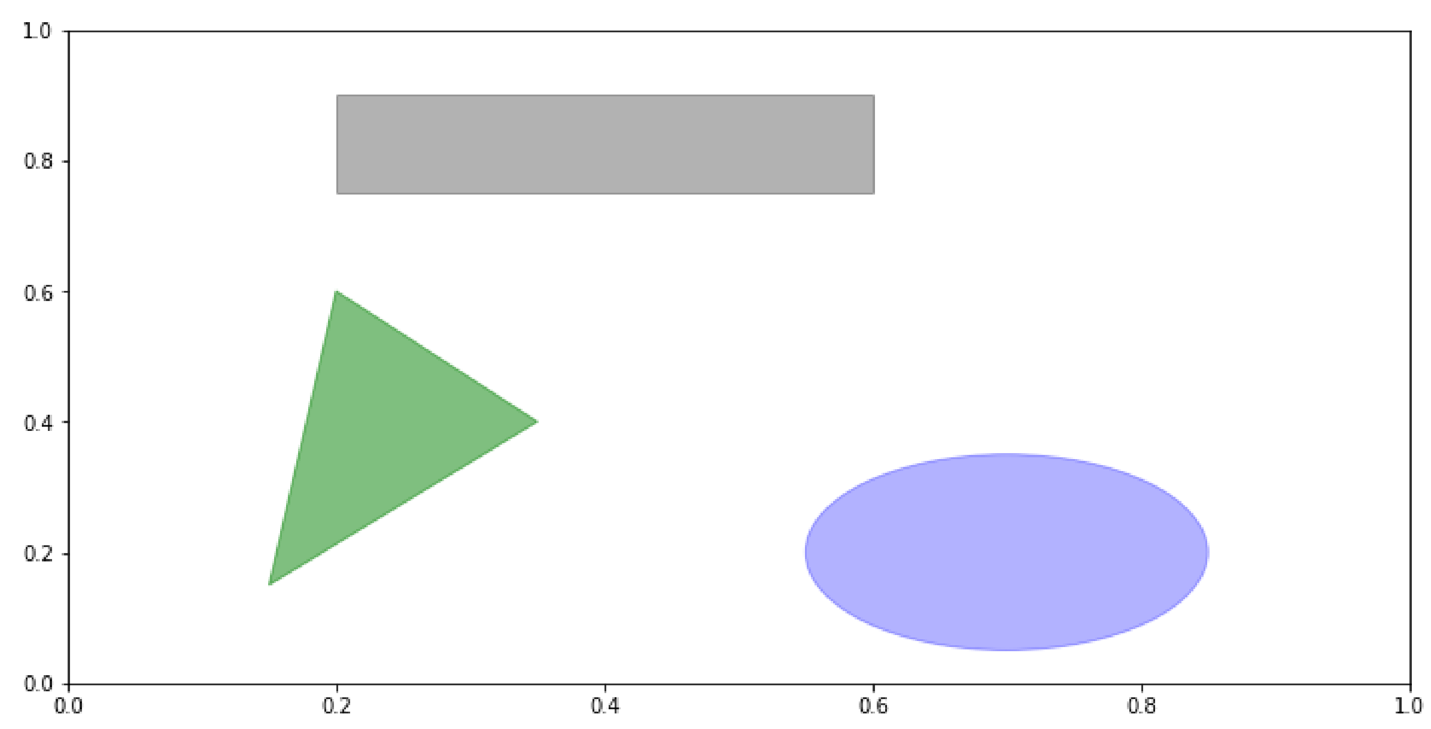


There are a couple of important points to highlight in this plot. The ax.annotate method can draw labels at the indicated x and y coordinates. We use the set_xlim and set_ylim methods to manually set the start and end boundaries for the plot rather than using matplotlib's default. Lastly, ax.set_title and a semi-colon at the end of the code adds a main title to the plot and ignores the warning respectively.

10

Drawing shapes requires some more care. matplotlib has objects that represent many common shapes, referred to as patches. Some of these, like Rectangle and Circle, are found in matplotlib.pyplot, but the full set is located in matplotlib.patches.

In [17]:
```python
#! ipython suppress id=038b73befd9a4dbab9a08fee140e53a3
fig, ax = plt.subplots(figsize=(12, 6))
rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color="black", alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color="blue", alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                   color="green", alpha=0.5)
ax.add_patch(rect)
ax.add_patch(circ)
#! figure,id=vis_patch_ex,width=4in,title="Data visualization composed from three different patches"
ax.add_patch(pgon)
plt.savefig('figpath.svg') #saving the active figure to a file
plt.savefig('figpath.png', dpi=400, bbox_inches='tight') #saving the active figure to a file
```



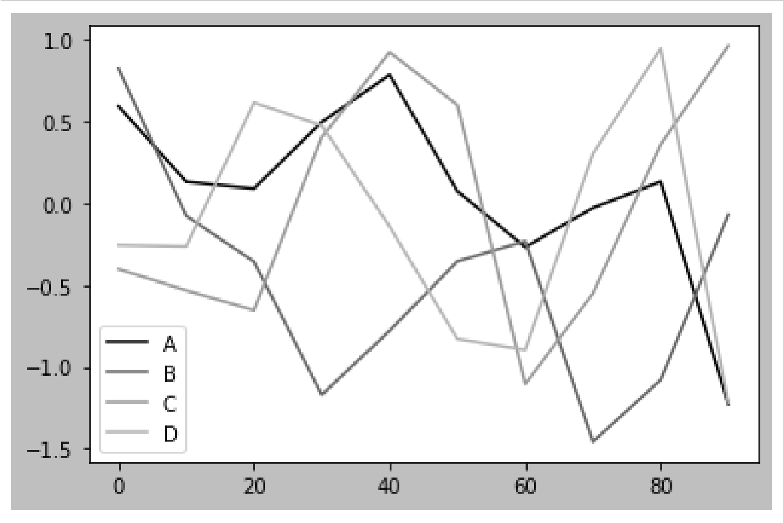
The extention after the filepathdetermines how the file isgoing to be saved. Whether it is going to be a png, pdf, word doc etc.

Plotting with pandas and seaborn

Line Plots

11

```
In [18]: plt.close("all")
         #! ipython id=1ef68f01e6f248259804975945e44951
         df = pd.DataFrame(np.random.standard_normal((10, 4)).cumsum(0),
                           columns=["A", "B", "C", "D"],
                           index=np.arange(0, 100, 10))
         plt.style.use('grayscale')
         #! figure,id=vis_frame_plot_1,width=4in,title="Simple DataFrame plot"

         df.plot();
```

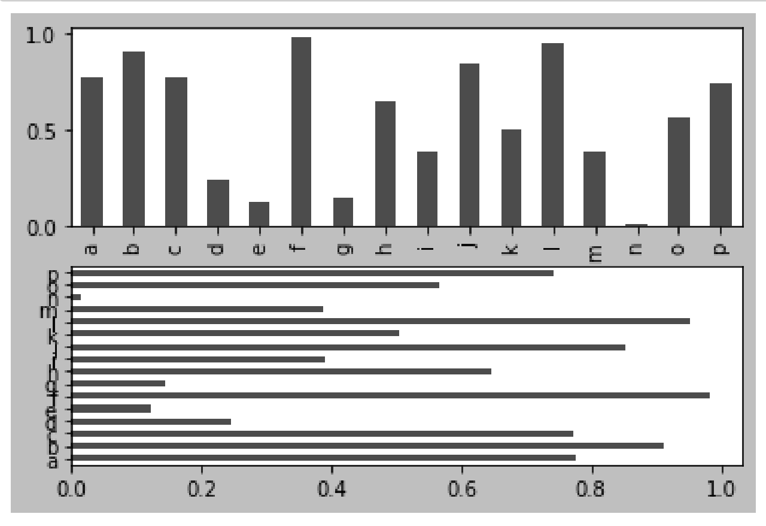


DataFrame's plot method plots each of its columns as a different line on the same subplot, creating a legend automatically, as seen above. The 'grayscale' style is used because it is more suited for black and white publication. Also the df.plot line contains a "family" of methods for different plot types and arguments can be passed through the plots to further customize the plots.

Bar Plots

12

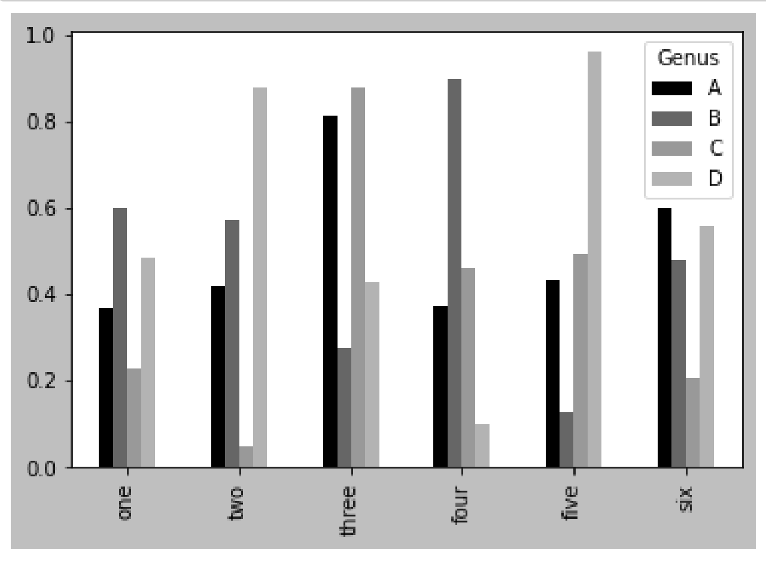
```
In [19]: #! ipython id=55da9f184455476f8ce79e4eaf79742c
         fig, axes = plt.subplots(2, 1)
         data = pd.Series(np.random.uniform(size=16), index=list("abcdefghijklmnop"))
         data.plot.bar(ax=axes[0], color="black", alpha=0.7);
         #! figure,id=vis_bar_plot_ex,width=4.5in,title="Horizonal and vertical bar plot"
         data.plot.barh(ax=axes[1], color="black", alpha=0.7);
```



The plot.bar() and plot.barh() make vertical and horizontal bar plots, respectively. In this case, the Series or DataFrame index will be used as the x (bar) or y (barh) ticks as seen above.

13

```
In [20]: np.random.seed(12348)
         #! ipython id=da2de3b4af64451ca5b47b6687e40c44
         df = pd.DataFrame(np.random.uniform(size=(6, 4)),
                           index=["one", "two", "three", "four", "five", "six"],
                           columns=pd.Index(["A", "B", "C", "D"], name="Genus"))
         df
         #! figure,id=vis_frame_barplot,width=4in,title="DataFrame bar plot"
         df.plot.bar();
```



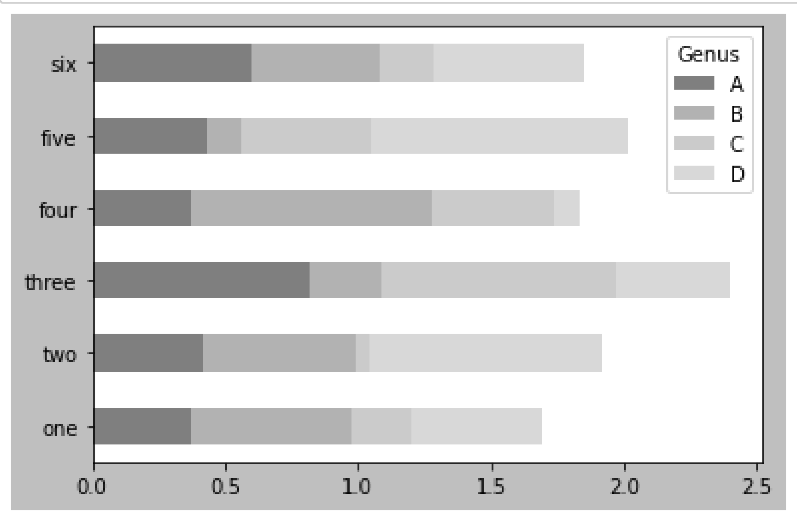
The name Genus of the dataframe columns is used as the title of the legend.

```
In [21]: #! ipython suppress id=583fcb73d46a44ee9c25f4b652d8767a
         plt.figure()

Out[21]: <Figure size 432x288 with 0 Axes>

         <Figure size 432x288 with 0 Axes>
```

We call this function to initialize the creation of a new figure. Before adding data to make new plots.

14

```
In [22]: #! ipython id=ead442e8fee747aea00117cb77de970c
         #! figure,id=vis_frame_barplot_stacked,width=4in,title="DataFrame stacked bar plot"
         df.plot.barh(stacked=True, alpha=0.5);
```



Now we use the plot to put some data on the figure and in this case is a stacked horizontal barplot with style of alpha=0.5.

```
In [23]: #! ipython suppress id=70abd941e8d843678a1c1cf5010eaa27
         plt.close("all")
```

15

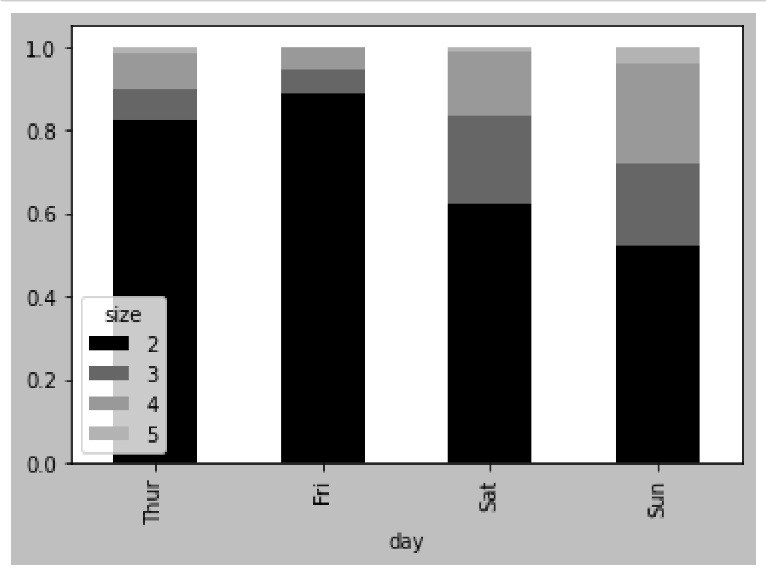
```
In [24]: #! ipython id=1bb875213a92468788f61c4e8012932c
         #store the tips csv as a dataframe to the tips variable
         tips = pd.read_csv("tips.csv")
         #Tips head prints the first 5 observations
         tips.head()
         #cross tabbing creates different indexing
         party_counts = pd.crosstab(tips["day"], tips["size"])
         #The index are set to be the appropriate days of the week
         party_counts = party_counts.reindex(index=["Thur", "Fri", "Sat", "Sun"])
         party_counts
```

Out[24]:

size	1	2	3	4	5	6
day						
Thur	1	48	4	5	1	3
Fri	1	16	1	1	0	0
Sat	2	53	18	13	1	0
Sun	0	39	15	18	3	1

```
In [25]: #! ipython id=4966d9560d0740db829531eb2a8f6c7a
         # Storing the party counts of all the rows but
         # Only from columns 2 to 5 both included.
         party_counts = party_counts.loc[:, 2:5]
```

```
In [26]: #! ipython id=b88726f319554e9cb32498dce98e66ca
         # Normalize to sum to 1
         party_pcts = party_counts.div(party_counts.sum(axis="columns"),
                                       axis="index")
         party_pcts
         #! figure,id=vis_tips_barplot,width=4in,title="Fraction of parties by size within each day"
         party_pcts.plot.bar(stacked=True);
```

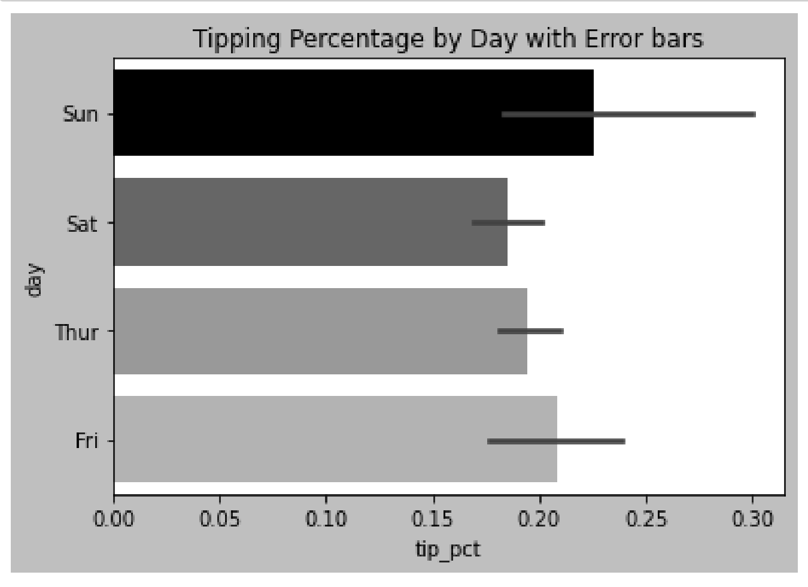


From the stacked barplots we may observe that the party sizes tend to increase more during Saturday and Sunday. And this can be seen through the guidance of the legend and the representation of the shades of gray on each plot.

```
In [27]: #! ipython suppress id=2e6c41575fc34b8ca7963b2c24167be2
         plt.close("all")
```

16

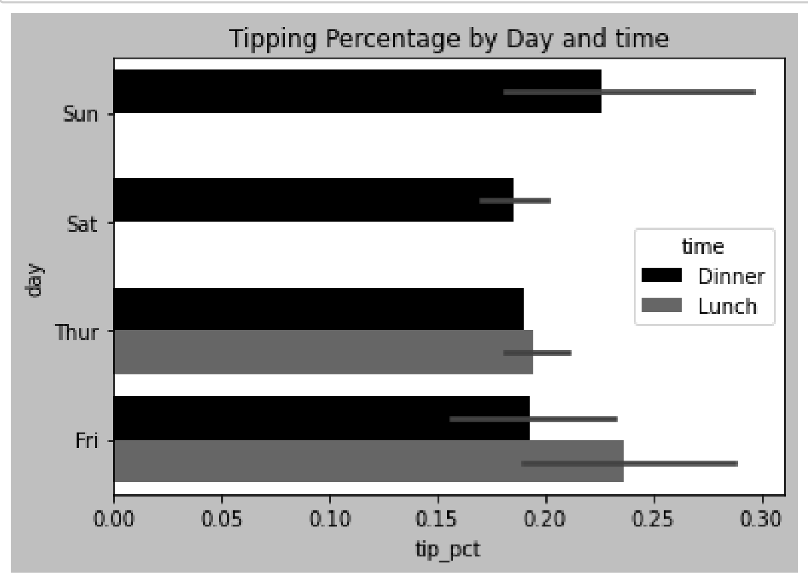
```
In [28]: #! ipython id=97f4026c26cd42faa7d03d3e0dcc75fa
         import seaborn as sns

         tips["tip_pct"] = tips["tip"] / (tips["total_bill"] - tips["tip"])
         tips.head()
         #! figure,id=vis_tip_pct_seaborn,width=4in,title="Tipping percentage by day with error bars"
         sns.barplot(x="tip_pct", y="day", data=tips, orient="h",);
         # Adding a title to the plot
         plt.title("Tipping Percentage by Day with Error bars");
```



```
In [29]: #! ipython suppress id=05ae3cd4fb7048c5ba0d9ed6390c1f03
         plt.close("all")
```

17

```
In [30]: #! ipython id=fd8ada60aab448848ded49ef8a81081c
         #! figure,id=vis_tip_pct_sns_grouped,width=4in,title="Tipping percentage by day and time"
         sns.barplot(x="tip_pct", y="day", hue="time", data=tips, orient="h")
         # Adding a title to the plot
         plt.title("Tipping Percentage by Day and time");
```



seaborn.barplot has a hue option that enables us to split by an additional categorical value as seen in the plot above.

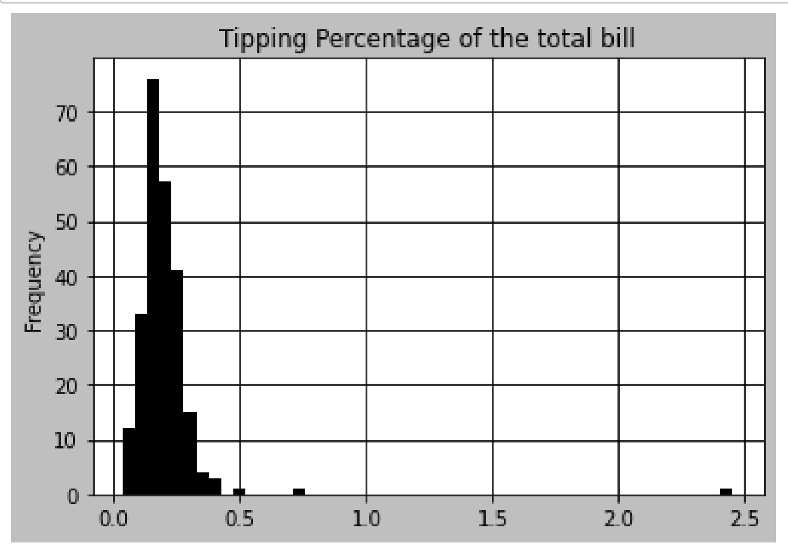
```
In [31]: #! ipython suppress id=d849ee158e424d2ca9e9741fdef08a7a
         plt.close("all")
```

Histograms and Density Plots

18

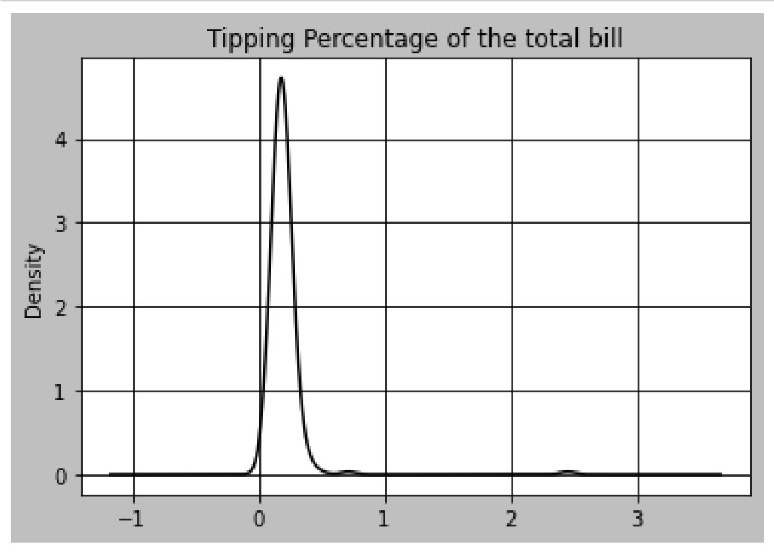
```
In [32]: #! ipython suppress id=ce4d75faf525411bbb4e8edbe016ff27
         plt.figure()
Out[32]: <Figure size 432x288 with 0 Axes>

         <Figure size 432x288 with 0 Axes>
```

```
In [33]: #! ipython id=1589674e6dda4960800de8803aa5c3ab
         #! figure,id=vis_hist_ex,width=4in,title="Histogram of tip percentages"
         tips["tip_pct"].plot.hist(bins=50)
         # Adding a title to the plot
         plt.title("Tipping Percentage of the total bill");
         # adding gridlines
         plt.grid()
```



19

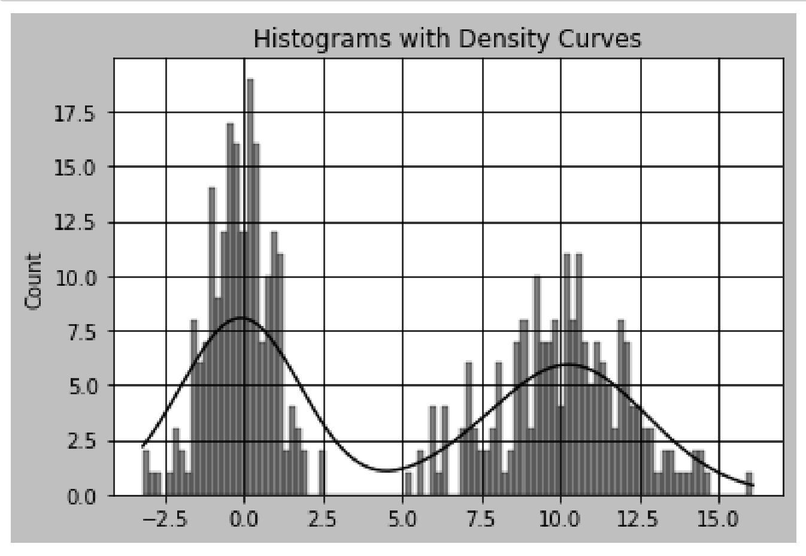
```
In [34]: #! ipython id=0173a110796b44ec85d62f05b9d703be
         #! figure,id=vis_kde_ex,width=4in,title="Density plot of tip percentages"
         tips["tip_pct"].plot.density()
         plt.title("Tipping Percentage of the total bill");
         # adding gridlines
         plt.grid()
```



20

```
In [35]: #! ipython suppress id=0bee68ae5d834d4f8ad68ff964bd39f6
         plt.figure()
Out[35]: <Figure size 432x288 with 0 Axes>

         <Figure size 432x288 with 0 Axes>
```

In [36]:
```
#! ipython id=95802c949eee44d7a9f7e5760e85ddde
comp1 = np.random.standard_normal(200)
comp2 = 10 + 2 * np.random.standard_normal(200)
values = pd.Series(np.concatenate([comp1, comp2]))
sns.histplot(values, bins=100, color="black", kde = True)
plt.title("Histograms with Density Curves");
# adding gridlines
plt.grid()
```



seaborn makes histograms and density plots even easier through its histplot method, which can plot both a histogram and a continuous density estimate simultaneously as seen above.

Scatter or Point Plots

21

In [37]:
```
#! ipython id=b99bf04e3b824743a391b9590e4b201b
macro = pd.read_csv("macrodata.csv")
data = macro[["cpi", "m1", "tbilrate", "unemp"]]
#transforming the data by taking the log values
trans_data = np.log(data).diff().dropna()
# the tail function views the last 5 rows
trans_data.tail()
```

Out[37]:

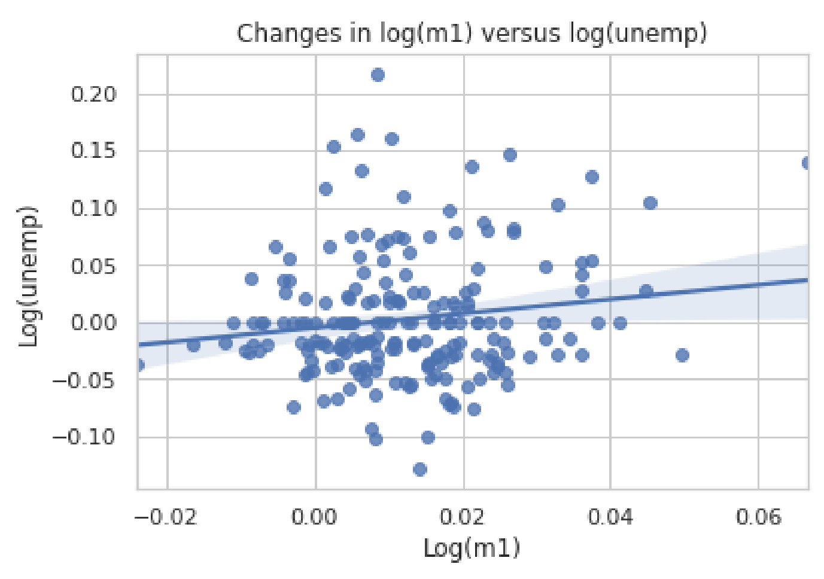
	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

In [38]:
```
#! ipython suppress id=0b0300f35b1a49d7935c15315a863735
plt.figure()
```

Out[38]:
```
<Figure size 432x288 with 0 Axes>
```

```
<Figure size 432x288 with 0 Axes>
```

In [39]:
```
#! ipython id=715be198942a441688a1bd8765f88cb3
# Set the style to whitegrid for a cleaner look
sns.set(style="whitegrid")

ax = sns.regplot(x="m1", y="unemp", data=trans_data)
#! figure,id=scatter_plot_ex,width=3in,title="A seaborn regression/scatter plot"
plt.title("Changes in log(m1) versus log(unemp)")
# Set axis labels
ax.set_xlabel("Log(m1)", fontsize=12)
ax.set_ylabel("Log(unemp)", fontsize=12);
```

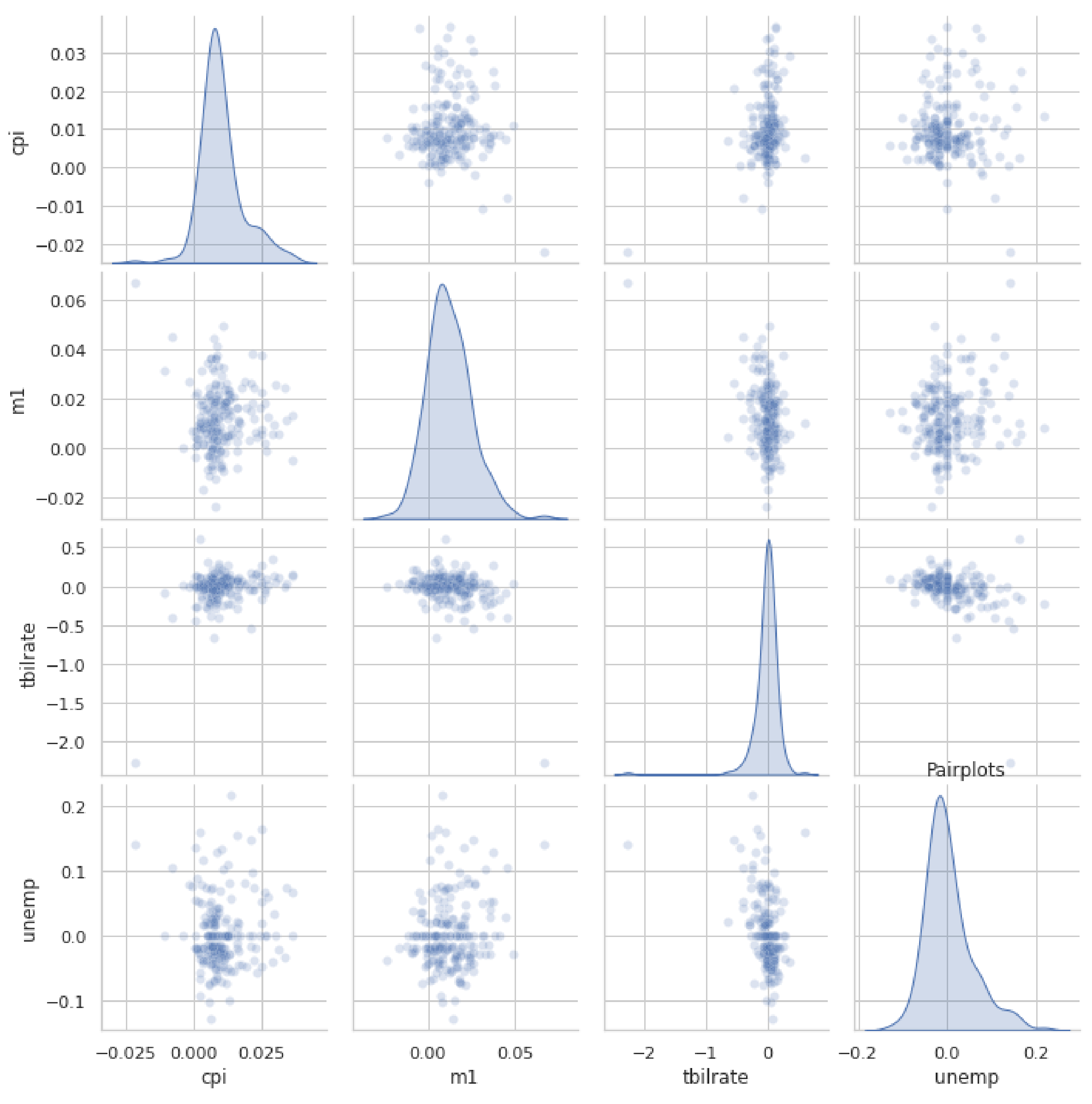


From this scatter plot and the regression line running through it, we may observe that there is little to no correlation between unemployment and m1. With he gridline we can see that correlation is less than 0.05.

22

In [67]:
```
#! ipython id=c847f9d3a28f4c0f98518f94eee846ce
#! figure,id=scatter_matrix_ex,width=4in,title="Pair plot matrix of statsmodels macro data"
sns.pairplot(trans_data, diag_kind="kde", plot_kws={"alpha": 0.2});
#adding title
plt.title("Pairplots")
```

Out[67]:
```
Text(0.5, 1.0, 'Pairplots')
```

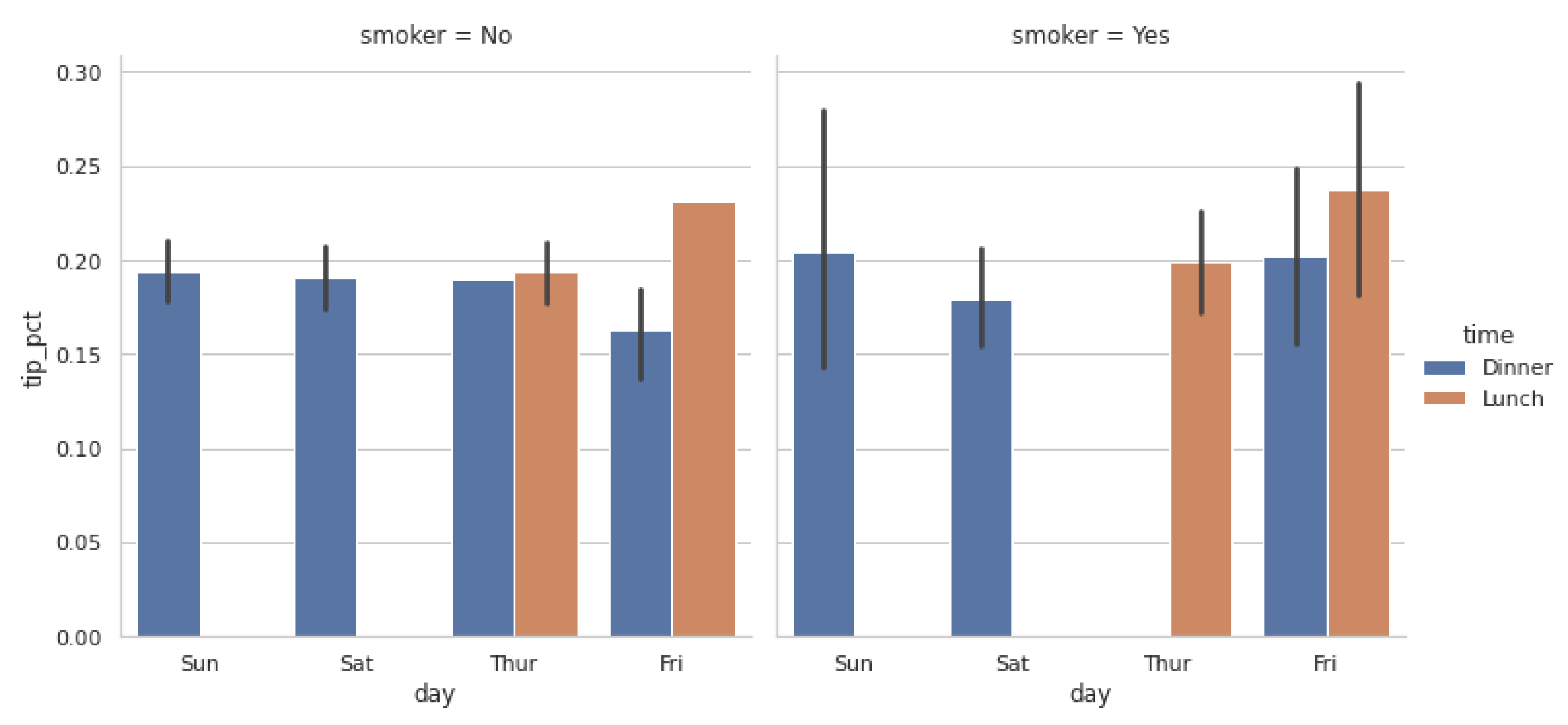


The pairplots also show little to no correlation among all the variables. CPI, m1 and unemp seem to be skewed to right whereas tbilrate seems to be skewed to the left. So there is a possibility of some outliers but we cannot conclude on that now as we need to do some further investigations.

Facet Grids and Categorical Data

23

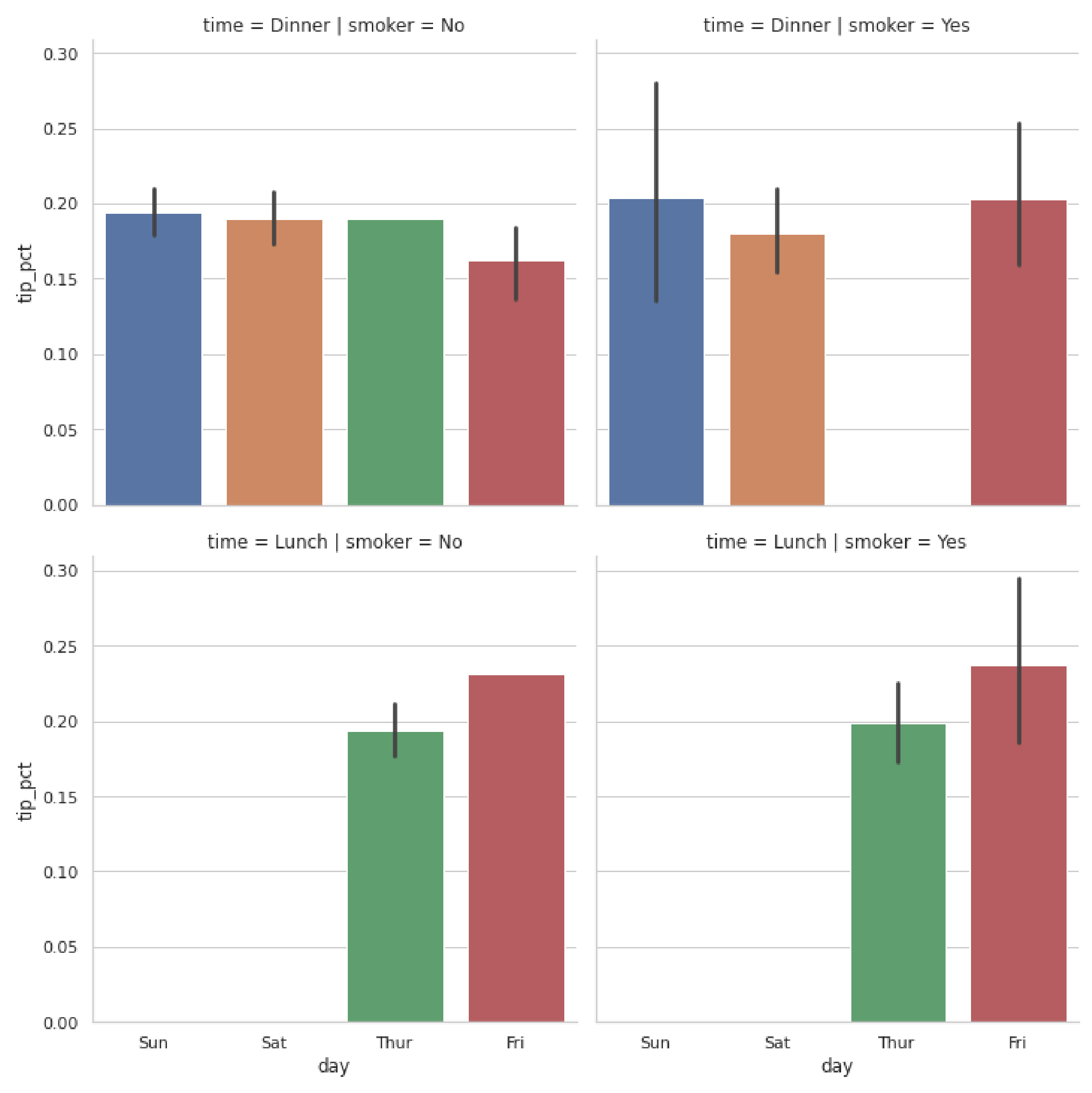
In [43]:
```
#! ipython id=e7ac5cdddb9a40afbc2d3d1fcc07f0e9
#! figure,id=vis_tip_pct_sns_factorplot,width=4in,title="Tipping percentage by day/time/smoker"
sns.catplot(x="day", y="tip_pct", hue="time", col="smoker",
            kind="bar", data=tips[tips.tip_pct < 1]);
#plt.title("Barchart of tip percentage during Lunch and Dinner");
```



One way to visualize data with many categorical variables is to use a facet grid, which is a two- dimensional layout of plots where the data is split across the plots on each axis based on the distinct values of a certain variable. seaborn has a useful built-in function cat plot that simplifies making many kinds of faceted plots split by categorical variables just like we have seen above. Where tipping percentage is split by day, time and smoker status.

24

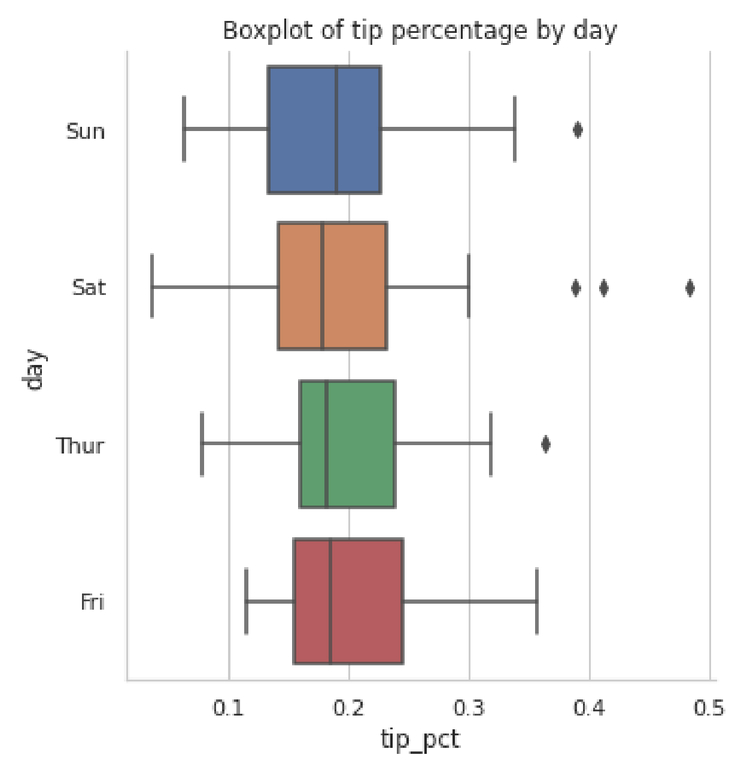
In [44]:
```
#! ipython id=1039f6945d4242ccb653e1fa645e2c81
#! figure,id=vis_tip_pct_sns_factorplot2,width=4in,title="Tipping percentage by day split by time/smoker"
sns.catplot(x="day", y="tip_pct", row="time",
            col="smoker",
            kind="bar", data=tips[tips.tip_pct < 1]);
```



There is even more splits with this cat.plot because tipping percentage by day is split by time and smoker status.

25

In [46]:
```
#! ipython id=511751cc57d844a1b39052cca90803f8
#! figure,id=vis_tip_pct_sns_factor_box,width=4in,title="Box plot of tipping percentage by day"
sns.catplot(x="tip_pct", y="day", kind="box",
            data=tips[tips.tip_pct < 0.5])
plt.title("Boxplot of tip percentage by day");
```



Using bokeh

Lorenz attractor example

26

In [48]:
```
# Importing the libraries and setting up the viewing options below
```

In [49]:
```
%matplotlib inline
import numpy as np
import pandas as pd
PREVIOUS_MAX_ROWS = pd.options.display.max_rows
pd.options.display.max_rows = 20
pd.options.display.max_colwidth = 80
pd.options.display.max_columns = 20
np.random.seed(12345)
import matplotlib.pyplot as plt
import matplotlib

plt.rc("figure", figsize=(10, 6))
np.set_printoptions(precision=4, suppress=False)
```

In [50]:
```
import numpy as np
from scipy.integrate import odeint
```

In [51]:
```
from bokeh.plotting import figure, show
```

In [52]:
```
from bokeh.io import output_notebook, show #needed to display in a Jupyter notebook
```

In [53]:
```
#initial steps to create function to be plotted
sigma = 10
rho = 28
beta = 8.0/3
theta = 3 * np.pi / 4
```

In [54]:
```
#defining the function to be plotted
def lorenz(xyz, t):
    x, y, z = xyz
    x_dot = sigma * (y - x)
    y_dot = x * rho - x * z - y
    z_dot = x * y - beta* z
    return [x_dot, y_dot, z_dot]
```

In [55]:
```
initial = (-10, -7, 35)
t = np.arange(0, 100, 0.006)
```

In [56]:
```
solution = odeint(lorenz, initial, t)
```

```
In [57]: x = solution[:, 0]
         y = solution[:, 1]
         z = solution[:, 2]
```

```
In [58]: xprime = np.cos(theta) * x - np.sin(theta) * y

         colors = ["#C6DBEF", "#9ECAE1", "#6BAED6", "#4292C6", "#2171B5", "#08519C", "#08306B",]
```

```
In [59]: output_notebook() #needed to display in a Jupyter notebook
```

(https://bokeh.org) Loading BokehJS ...

Bokeh can display content in classic Jupyter notebooks as well as in JupyterLab

```
In [60]: p = figure(title="Lorenz attractor example", background_fill_color="#fafafa")

         p.multi_line(np.array_split(xprime, 7), np.array_split(z, 7),
                      line_color=colors, line_alpha=0.8, line_width=1.5)
```

```
Out[60]: GlyphRenderer(id = '1039', …)
```

```
In [62]: #use the show() function to generate your graph and open a web browser to display the generated HTML file.
         show(p)
```

It has interactive features. Use the tools on the right of the plot to explore:

The first icon below the bokeh logo repesents pan tool Use the pan tool to move the graph within your plot.

Second, represents box zoom Use the box zoom tool to zoom into an area of your plot.

Third, represents the wheel zoom Use the wheel zoom tool to zoom in and out with a mouse wheel.

Fourth, represents the save tool Use the save tool to export the current view of your plot as a PNG file.

Fifth, represents the reset tool Use the reset tool to return your view to the plot's default settings.

Sixth, represents help.

With just a few lines of Python code, Bokeh enables you to create interactive, JavaScript-powered visualizations displayable in a web browser.

The basic idea of Bokeh is a two-step process: First, you select from Bokeh's building blocks to create your visualization. Second, you customize these building blocks to fit your needs.

To do that, Bokeh combines two elements:

A Python library for defining the content and interactive functionalities of your visualization.

A JavaScript library called BokehJS that is working in the background to display your interactive visualizations in a web browser.

Based on your Python code, Bokeh automatically generates all the necessary JavaScript and HTML code for you. In its default setting, Bokeh automatically loads any additional JavaScript code from Bokeh's CDN (content delivery network).

```
In [ ]:
```