Week 12: Plotting and Visualization using Matplotlib

please refer to the textbook: "Python for Data Analysis" by Wes McKinney for details of this topic (***Chapter 9***)

Load Matplotlib library

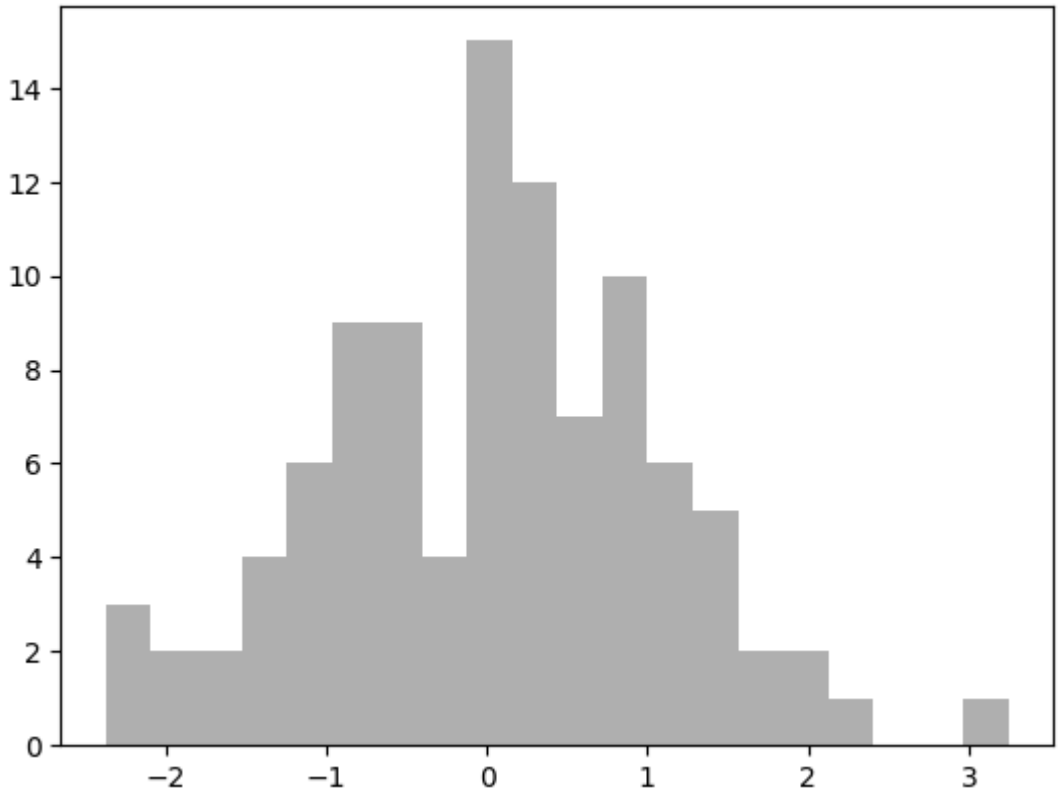
In [3]:
```
# Load library
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import numpy as np
import pandas as pd
import seaborn as sns
```

Individual plots

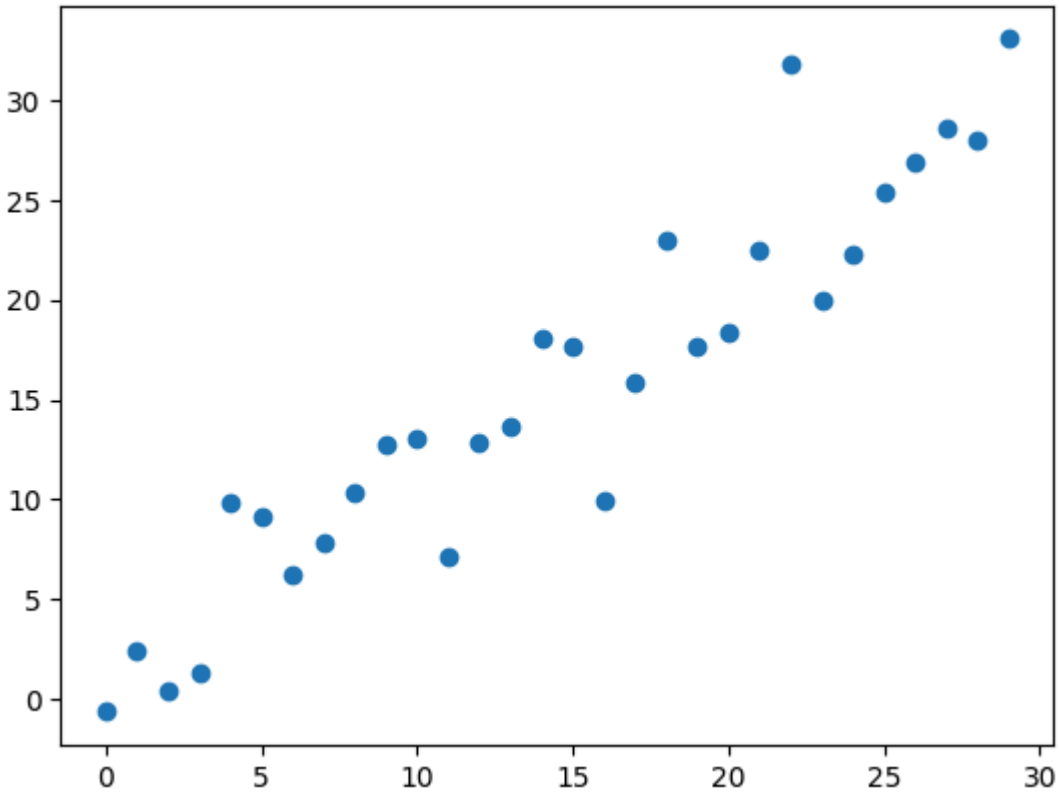
Histogram

- **`bins`** : Bins are the intervals into which the data is divided for counting and display in the histogram.
  - Having 20 bins means the data range will be split into 20 intervals
- **`alpha`** : controls the transparency of the bars.
  - A value of 1.0 would be completely opaque, while 0.0 would be completely transparent.

In [ ]:
```
# Histogram - plot 1
rng = np.random.RandomState(12345) #set.seed()
plt.hist(rng.standard_normal(100), bins=20, color='black',
         alpha=0.3); # ; will only show the plot
```



In [6]:
```
# Scatter plot - plot 2
rng = np.random.RandomState(12345)
plt.scatter(np.arange(30), np.arange(30) +
            3 * rng.standard_normal(30));
```



linestyle

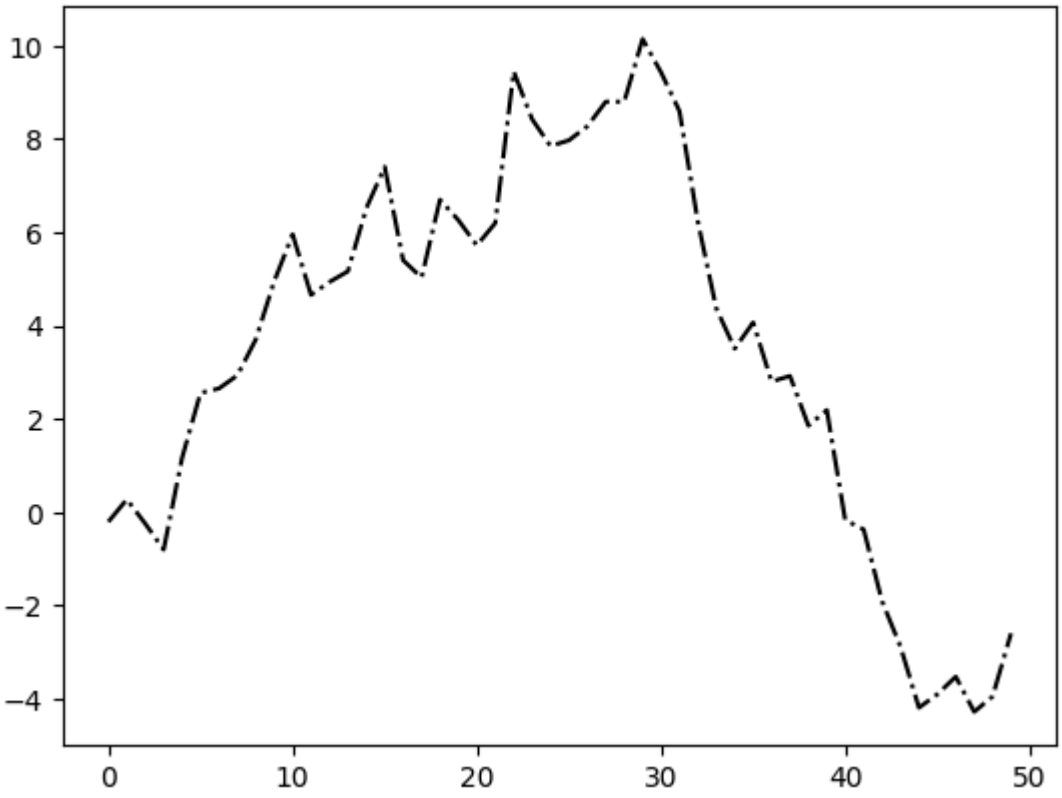
1. some other options for the linestyle argument:

- `'-'` (**solid**): This is the default linestyle and creates a solid line.
- `'--'` (**dashed**): Creates a dashed line.
- `'-.'` (**dash-dot**): Creates a line with alternating dashes and dots.
- `':'` (**dotted**): Creates a dotted line.
- `'**None**'` or `' '` or `''` (no line): This will plot only markers if specified, or no line at all.

2. we can also use the shorter codes for these linestyles:

- **`solid`** : `-`
- **`dashed`** : `--`
- **`dashdot`** : `-.`
- **`dotted`** : `:`

In [7]:
```
# Line plot : plot 3
rng = np.random.RandomState(12345)
plt.plot(rng.standard_normal(50).cumsum(),
         color='black', linestyle='-.');
```

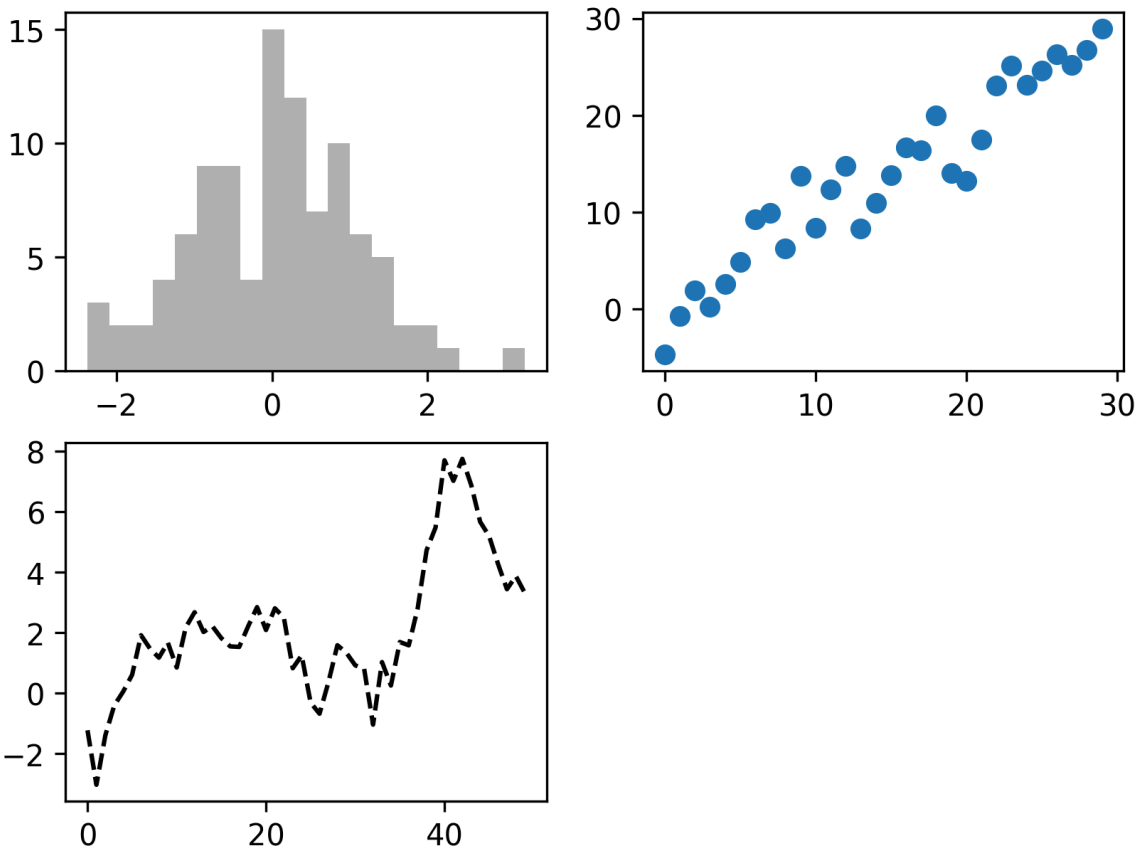


Place all the subplots into one panel

1. **plt.figure()** - create a new figure
2. **add_subplot** - to create one or more subplots
3. **Subplots** are numbered sequentially **starting from 1, going from left to right and top to bottom.** - a **(2, 2, 1)** argument means we're adding the first subplot (top-left corner) in the 2x2 grid.

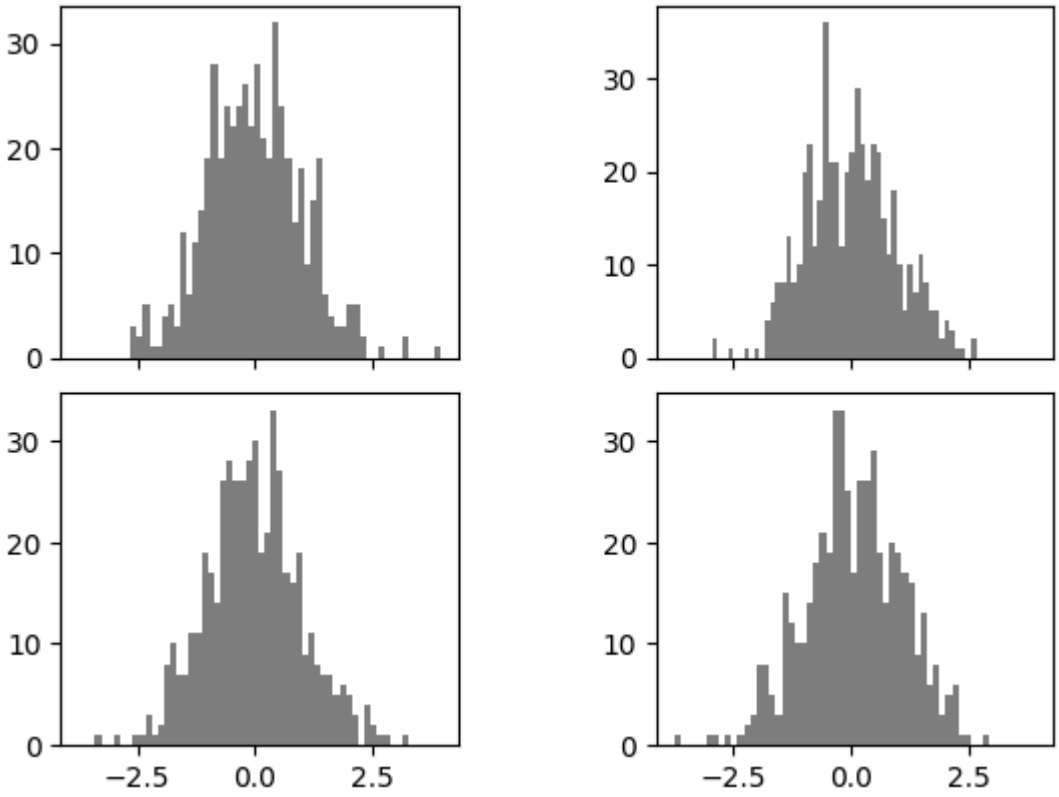
In [11]:
```
# Create one or more subplots
# 2 x 2; 4 plots in total
```

```
# Plot axis
rng = np.random.RandomState(12345)
fig = plt.figure(dpi=300) # dpi = resolution (dots per inch)
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax1.hist(rng.standard_normal(100), bins=20, color='black', alpha=0.3);
ax2.scatter(np.arange(30), np.arange(30) + 3 * rng.standard_normal(30));
ax3.plot(rng.standard_normal(50).cumsum(), color='black', linestyle='dashed');

# To save the figures
# plt.savefig('/content/drive/MyDrive/aliabu/combi3.pdf') - google colab
plt.savefig('./Output/combi3.pdf')
```



Adjusting the spacing around subplots

In [12]:
```
# An example that shrink the spacing of the subplots to zero
# Change the value of wspace and hspace
rng = np.random.RandomState(12345)
fig, axes = plt.subplots(2, 2, sharex=True, sharey=False)

# Using a nested loop to populate 2x2 grid
for i in range(2):
    for j in range(2):
        axes[i, j].hist(rng.standard_normal(500), bins=50,
                        color='black', alpha=0.5)

# Adjusting the spacing
fig.subplots_adjust(wspace=0.5, hspace=0.1)
```



Colors, Markers, and Line Styles

1. specify line styles using plt.plot documentation (***plt.plot?***)
2. specify color using hex code, e.g. ***"#CECECE"***
3. For more information, can visit this website: https://htmlcolorcodes.com/

Marker

specify a symbol or shape that will be placed at each data point on the plotted line

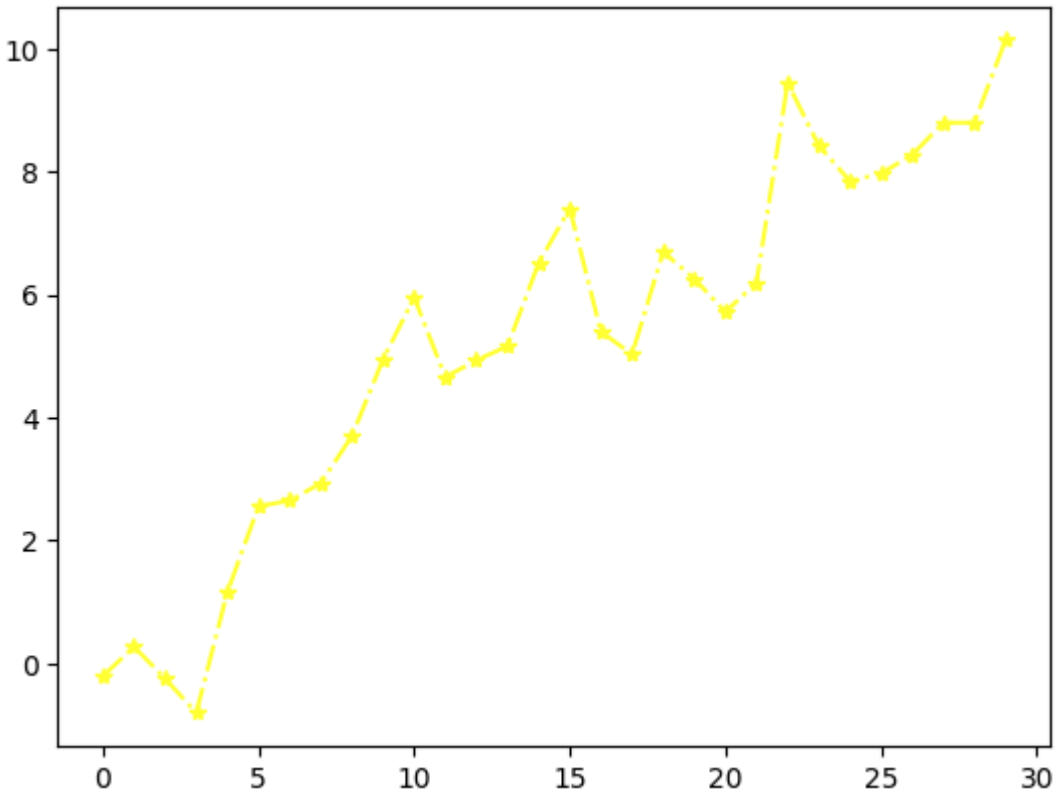
Other Marker Options

Matplotlib offers a wide variety of marker options. Here are a few examples:

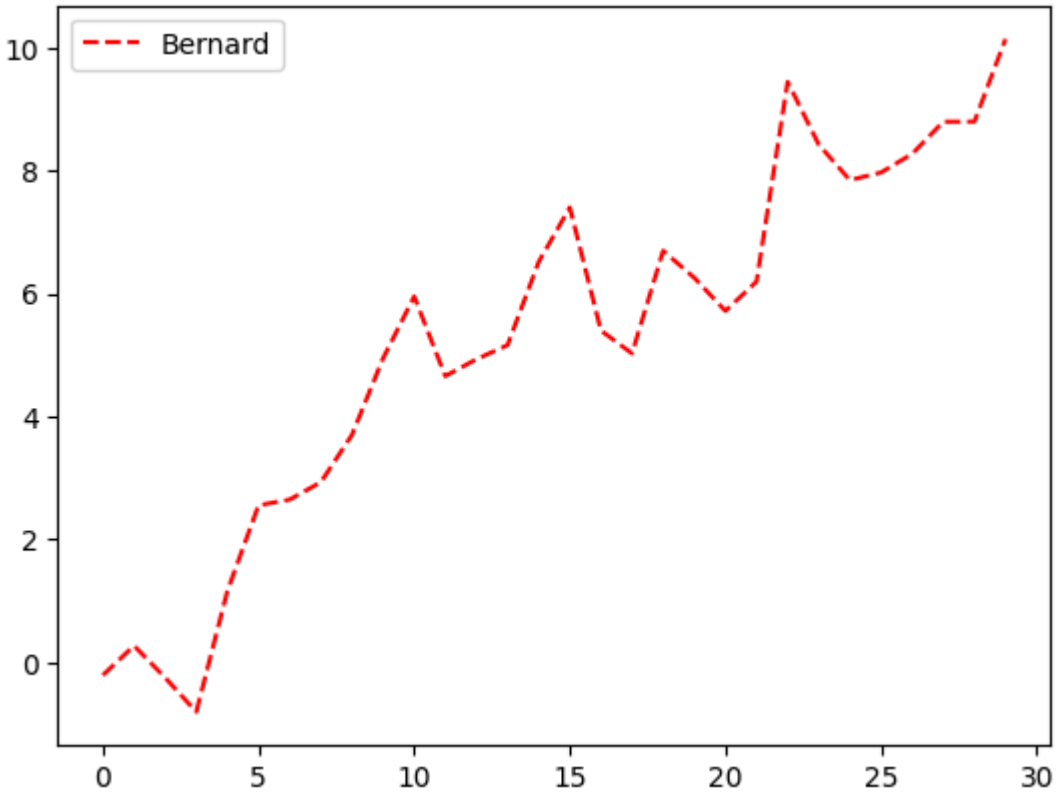
- `'.'` : Point
- `','` : Pixel
- `'o'` : Circle
- `'v'` : Triangle down
- `'^'` : Triangle up
- `'<'` : Triangle left
- `'>'` : Triangle right
- `'s'` : Square
- `'+'` : Plus
- `'x'` : X
- `'D'` : Diamond
- `'d'` : Thin diamond
- `'|'` : Vertical line

- `'_'`: Horizontal line

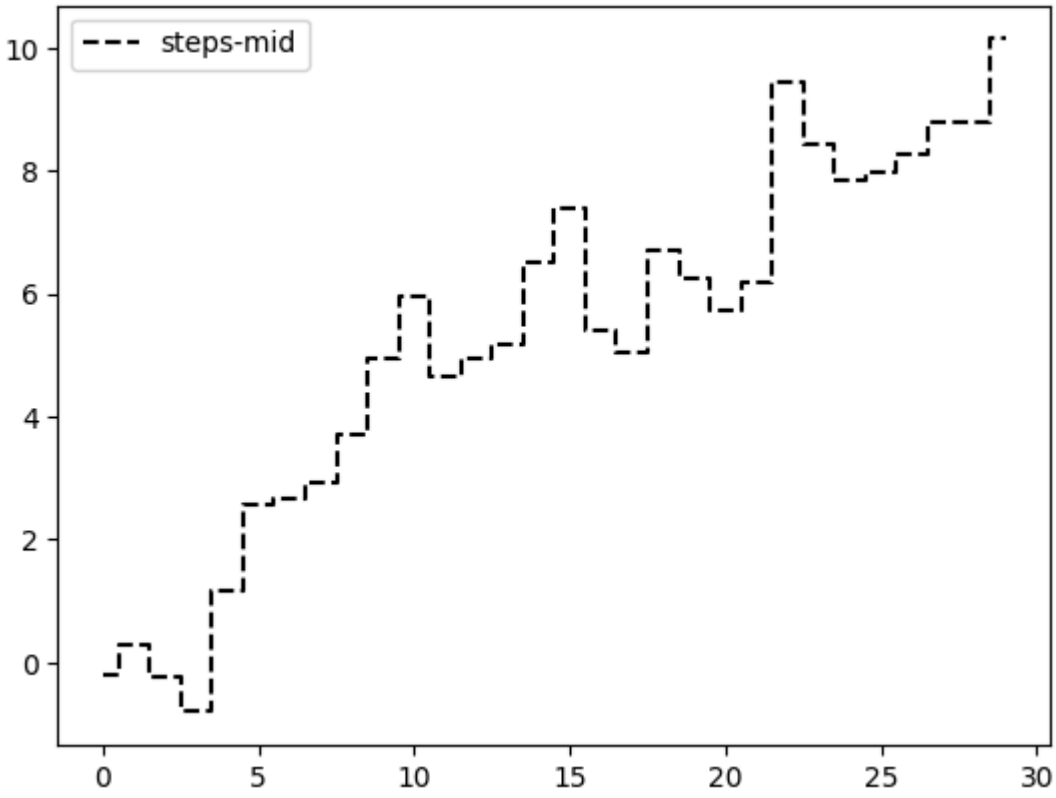
In [13]:
```
# Adding marker on line plot
rng = np.random.RandomState(12345)
plt.plot(rng.standard_normal(30).cumsum(), color='#ffff33',
        linestyle='-.', marker='*');
```



In [14]:
```
# Drawstyle default
rng = np.random.RandomState(12345)
data = rng.standard_normal(30).cumsum()
plt.plot(data, color='r', linestyle='dashed', label='Bernard')
plt.legend();
```

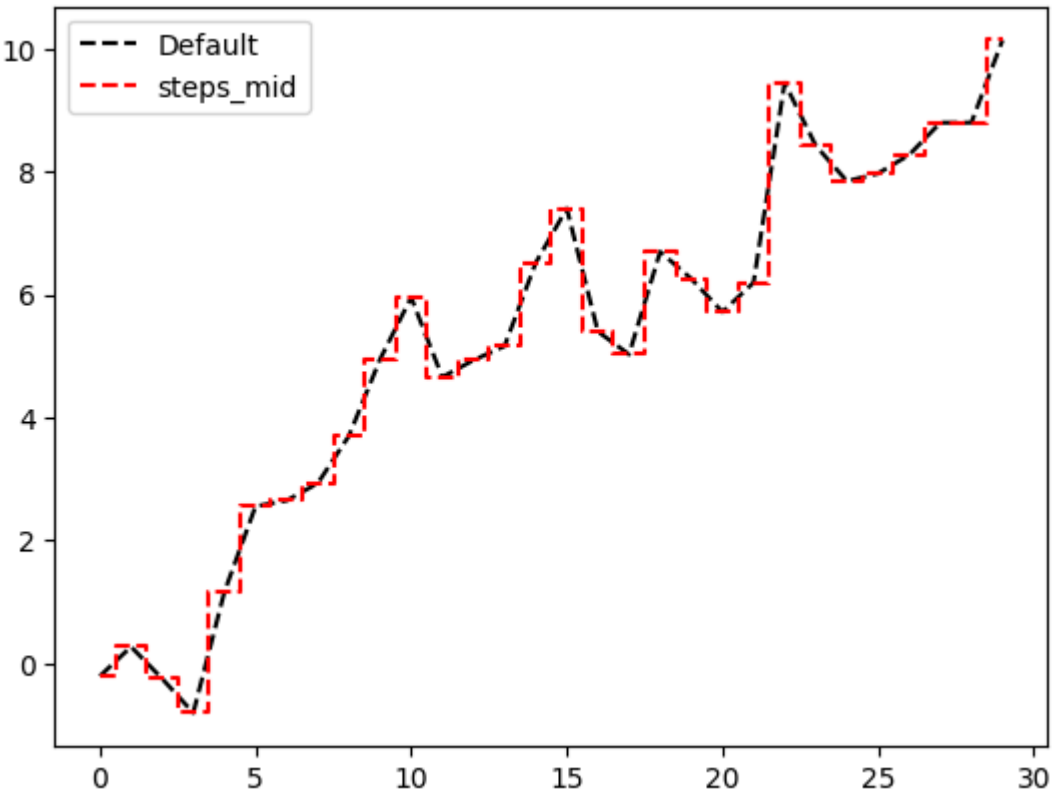


In [15]:
```
# Drawstyle steps-mid
plt.plot(data, color='black', linestyle='dashed',
         drawstyle='steps-mid', label='steps-mid')
plt.legend();
```



In [17]:
```
# Combine
plt.plot(data, color='black', linestyle='dashed', label='Default')
plt.plot(data, color='red', linestyle='dashed',
         drawstyle='steps-mid', label='steps_mid')
```

```
plt.legend();
#plt.savefig('/content/drive/MyDrive/lines.pdf', dpi=300)
plt.savefig('./Output/lines.pdf', dpi=300)
```

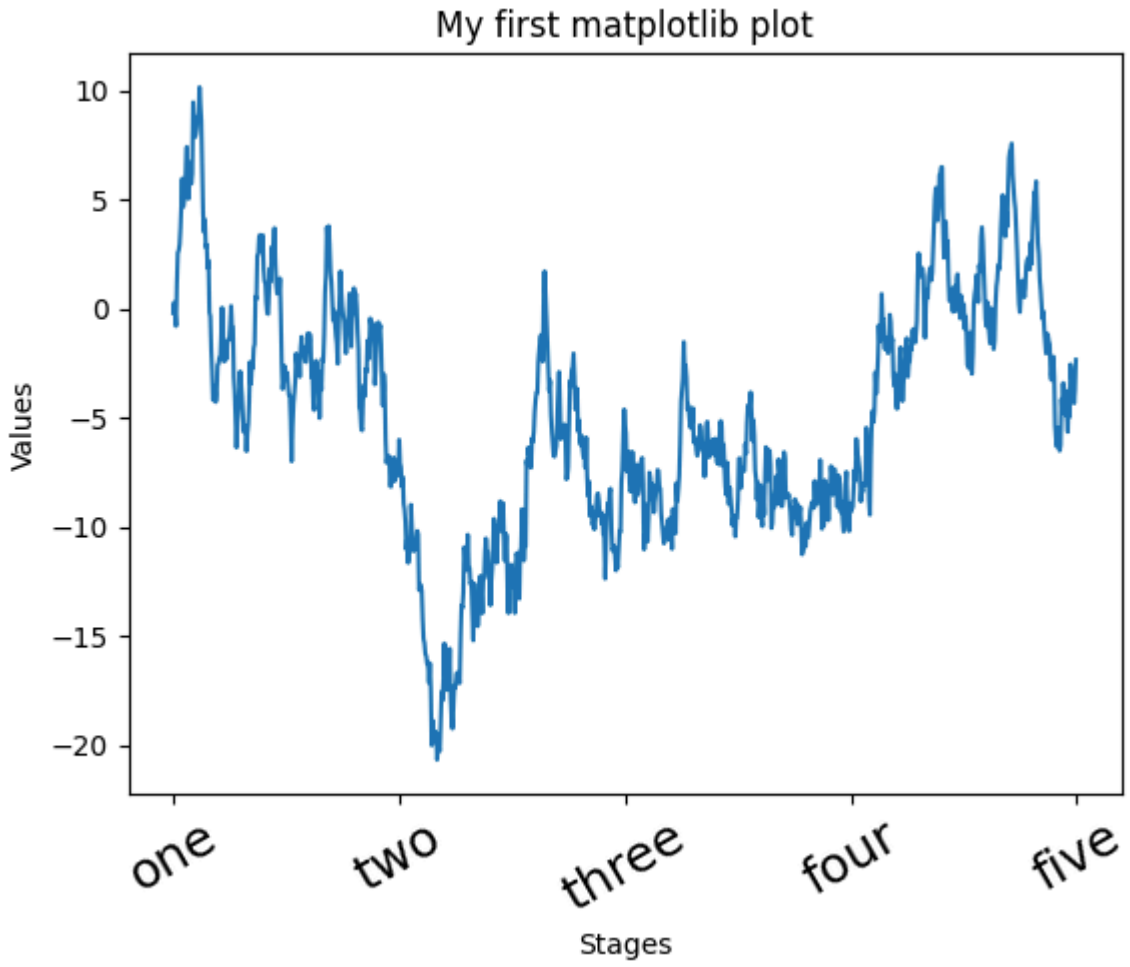


Ticks, Labels, and Legends

1. parameter such as ***xlim([0, 10])*** sets the x-xis range from ***0 to 10***

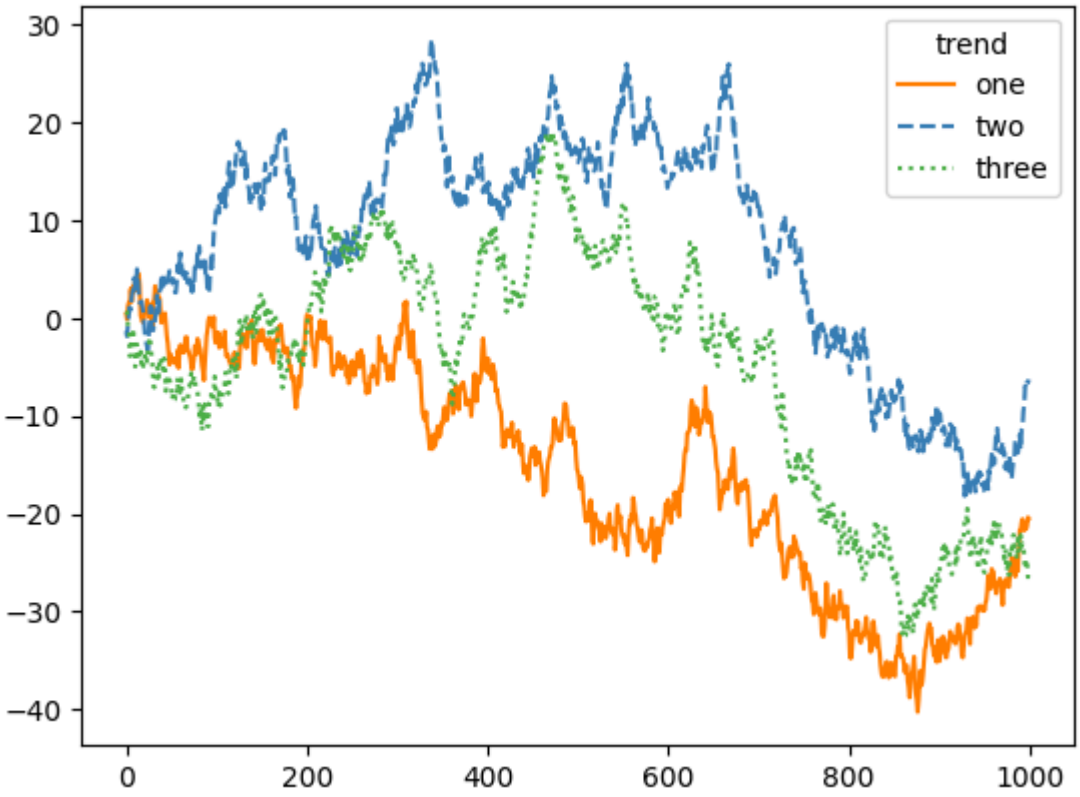
Setting the title, axis labels, ticks, and tick labels

In [18]:
```
# An example
rng = np.random.RandomState(12345)
fig, ax = plt.subplots()
ax.plot(rng.standard_normal(1000).cumsum())

# Changex x-axis ticks annotation
ax.set_xticks([0, 250, 500, 750, 1000])

# Cabel the x-axis ticks with names
ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],
                   rotation=30, fontsize=18)
ax.set_xlabel('Stages')
ax.set_ylabel('Values')
ax.set_title('My first matplotlib plot');
```



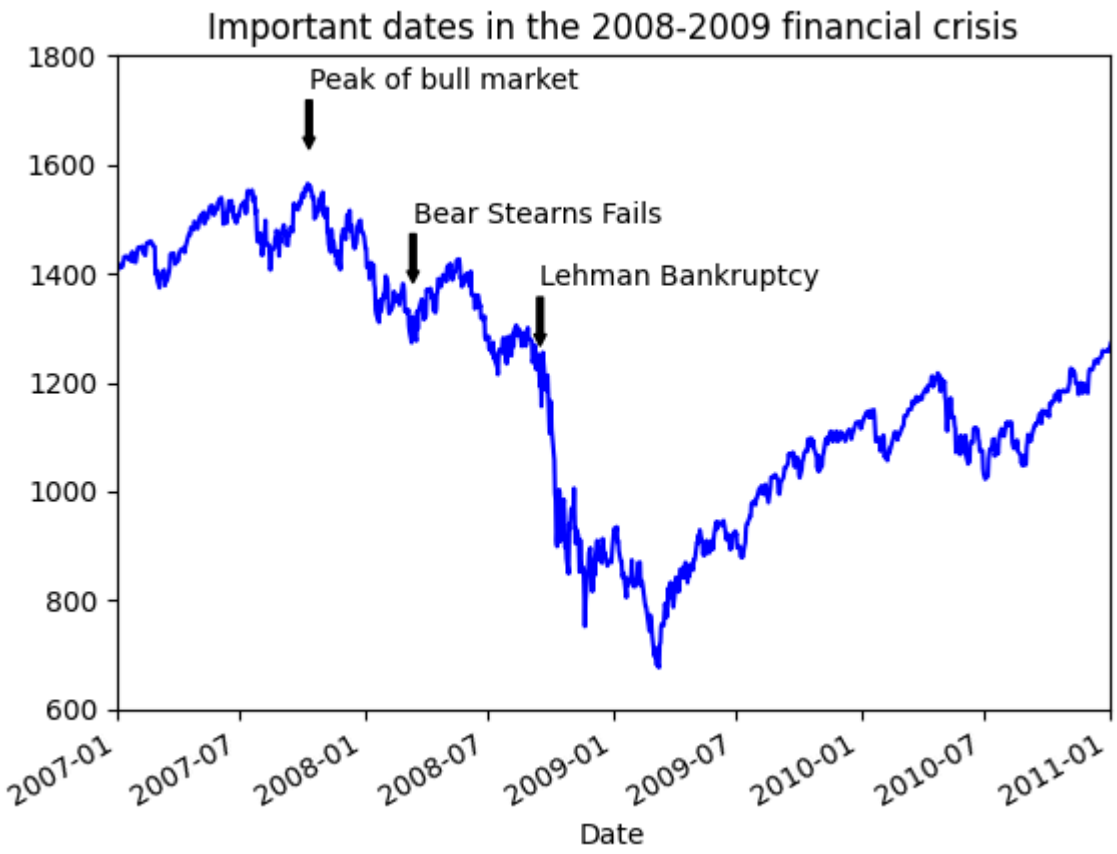
Adding Legends

In [19]:
```
# Adding legend
fig, ax = plt.subplots()
ax.plot(np.random.randn(1000).cumsum(), color='#FF7F00', label='one')
ax.plot(np.random.randn(1000).cumsum(), color='#377EB8', linestyle='dashed',
        label='two')
ax.plot(np.random.randn(1000).cumsum(), color='#4DAF4A', linestyle='dotted',
        label='three')
ax.legend(title='trend');
```

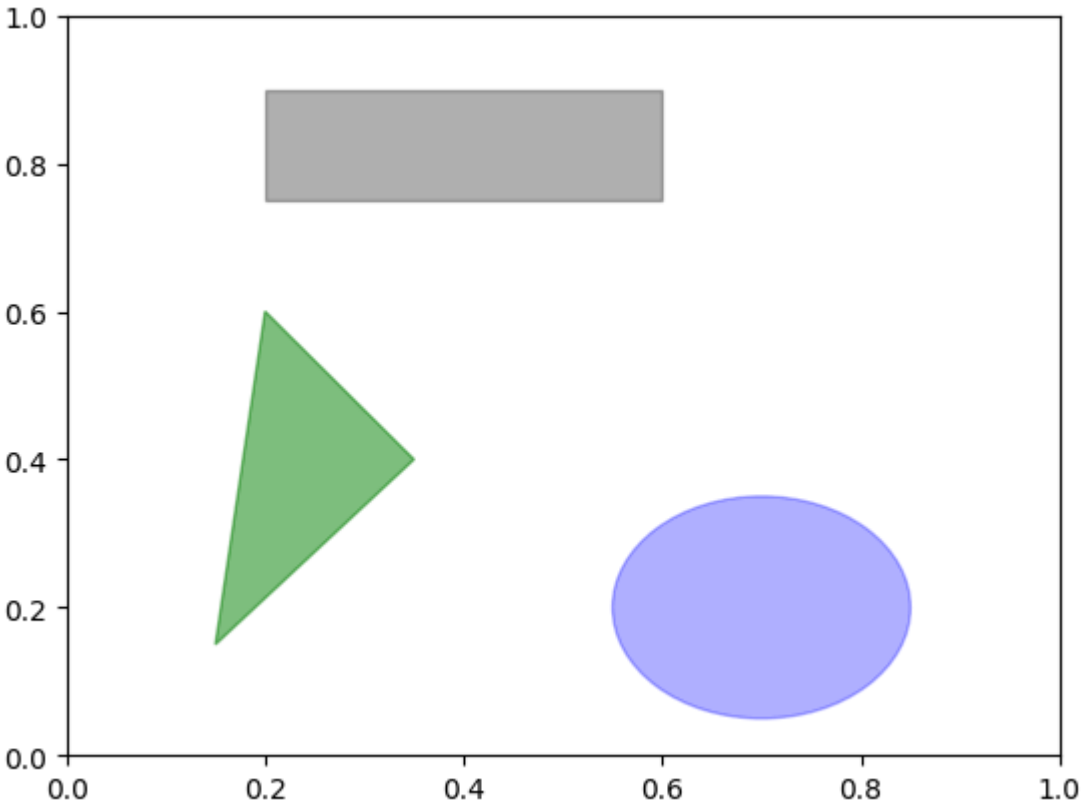


Annotations and Drawing on a Subplot

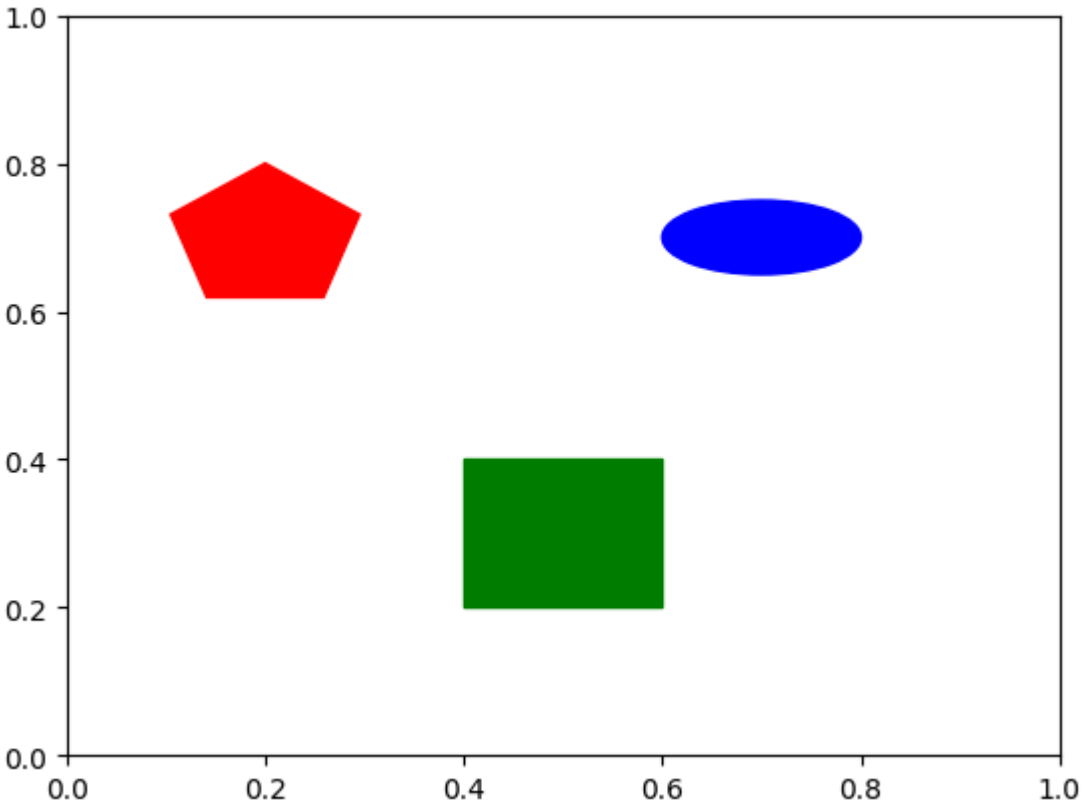
In [20]:
```
# The SPX data
from datetime import datetime

fig, ax = plt.subplots()

data = pd.read_csv('https://bit.ly/3iofIY8', index_col=0, parse_dates=True)
spx = data['SPX']

spx.plot(ax=ax, color='blue')

crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

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'])
ax.set_ylim([600, 1800])
ax.set_title('Important dates in the 2008-2009 financial crisis');
```



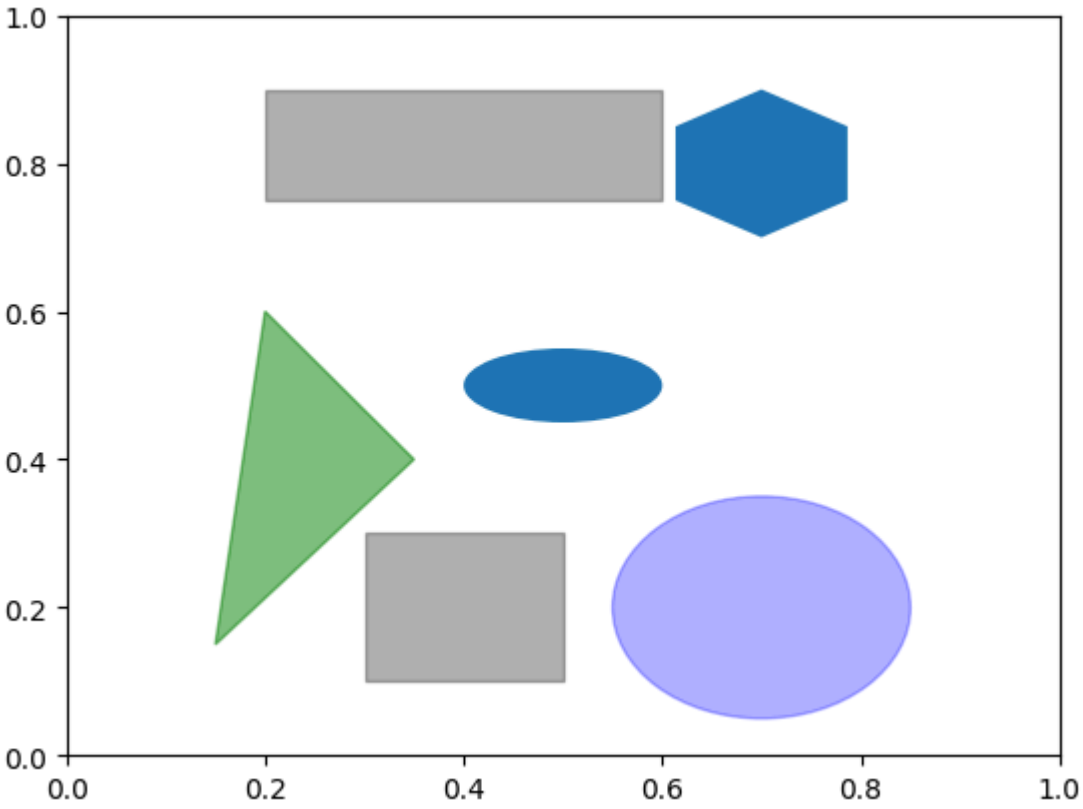
In [21]:
```
# Adding shapes to the same plot
fig, ax = plt.subplots()

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)
ax.add_patch(pgon);
```



In [24]:
```
# Challenge: plot a pentagon, ellipse, square figures in the exact same figure
fig, ax = plt.subplots()

# Pentagon
pentagon = patches.RegularPolygon((0.2, 0.7), 5, radius=0.1, color='red')  # Cen
ax.add_patch(pentagon)

# Ellipse
ellipse = patches.Ellipse((0.7, 0.7), 0.2, 0.1, color='blue')  # Center, width,
ax.add_patch(ellipse)

# Square
square = patches.Rectangle((0.4, 0.2), 0.2, 0.2, color='green')  # Lower-left, w
ax.add_patch(square)

plt.show()  # Display the figure
```



In [25]:
```
# Adding shapes to a plot
import matplotlib.patches as mpatches
fig, ax = plt.subplots()

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)
sqr = plt.Rectangle((0.3, 0.1), 0.2, 0.2, color='black', alpha=0.3)

# Add ellipse
ellipse = mpatches.Ellipse((0.5,0.5), 0.2, 0.1)

# Add regular hex
hex = mpatches.RegularPolygon((0.7,0.8), 6, radius=0.1)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
ax.add_patch(sqr)
ax.add_patch(ellipse)
ax.add_patch(hex);
```



In [26]:
```
# To find more information of the shapes
#plt.Rectangle?
#plt.Circle?
#plt.Polygon?
```

Saving Plots to File

1. File format: '***.png***','***.pdf***','***.svg***','***.ps***','***.eps***', .etc

In [27]:
```
# Saving the plot
#fig.savefig('/content/drive/MyDrive/STQD6014_Executive/_SEM1_20242025/Figures/s
fig.savefig('./Output/shapes.jpeg', dpi=400)
```

Matplotlib Configuration

using ***rc*** method

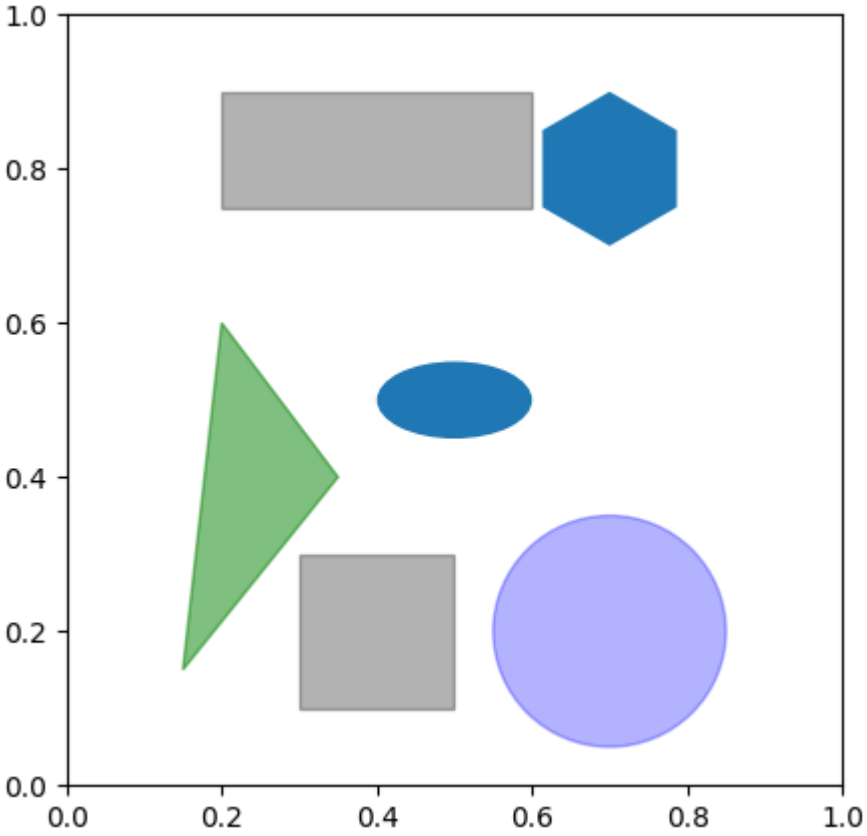
1. plt.rc('figure', figsize=(10, 10)) - to set the global default figure size to be 10 × 10

2. plt.rcdefaults() - restored to default values

In [28]:
```
# Challenge 1: To set the previous shapes figure to 5 x 5
plt.rc('figure', figsize=(5, 5))
```

In [29]:
```
# Adding shapes to a plot : Attention
import matplotlib.patches as mpatches
fig, ax = plt.subplots()

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)
sqr = plt.Rectangle((0.3, 0.1), 0.2, 0.2, color='black', alpha=0.3)

# Add ellipse
ellipse = mpatches.Ellipse((0.5,0.5),0.2,0.1)

# Add regular hex
hex = mpatches.RegularPolygon((0.7,0.8),6, radius=0.1)

ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
ax.add_patch(sqr)
ax.add_patch(ellipse)
ax.add_patch(hex);
```



In [30]:
```
# Reset to default value
plt.rcdefaults()
```

Plotting with pandas and seaborn

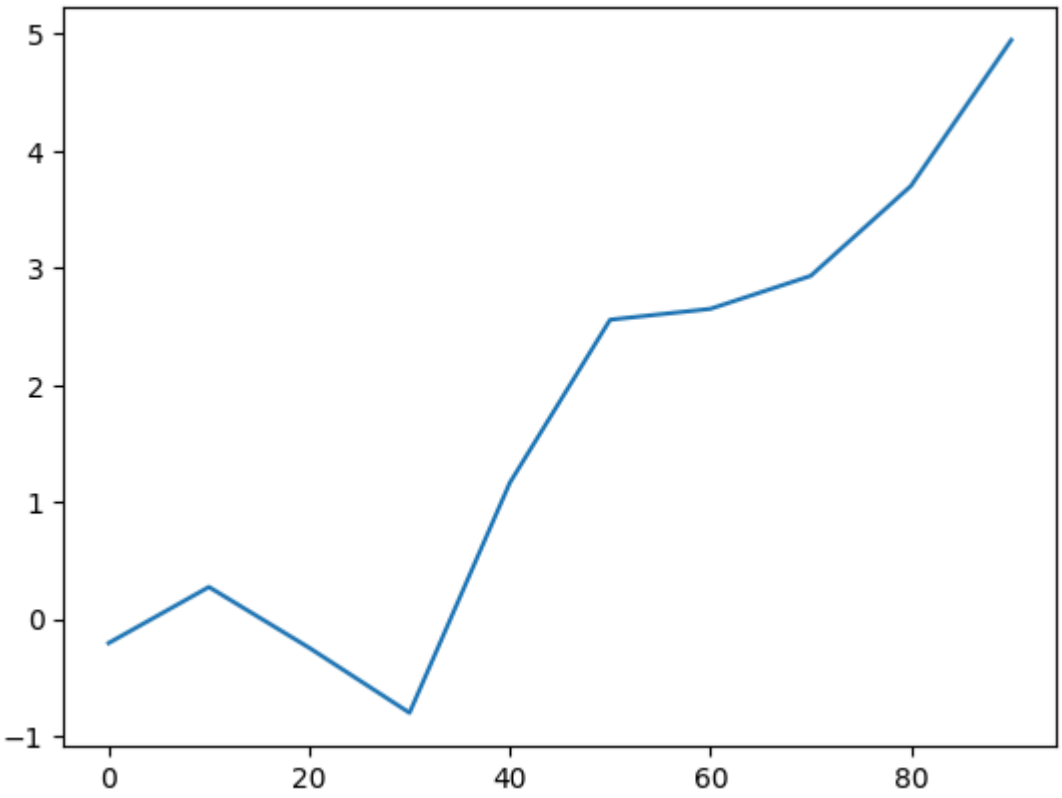
seaborn is a high-level ***statistical graphics library*** built on matplotlib.

seaborn simplifies creating many common visualization types.

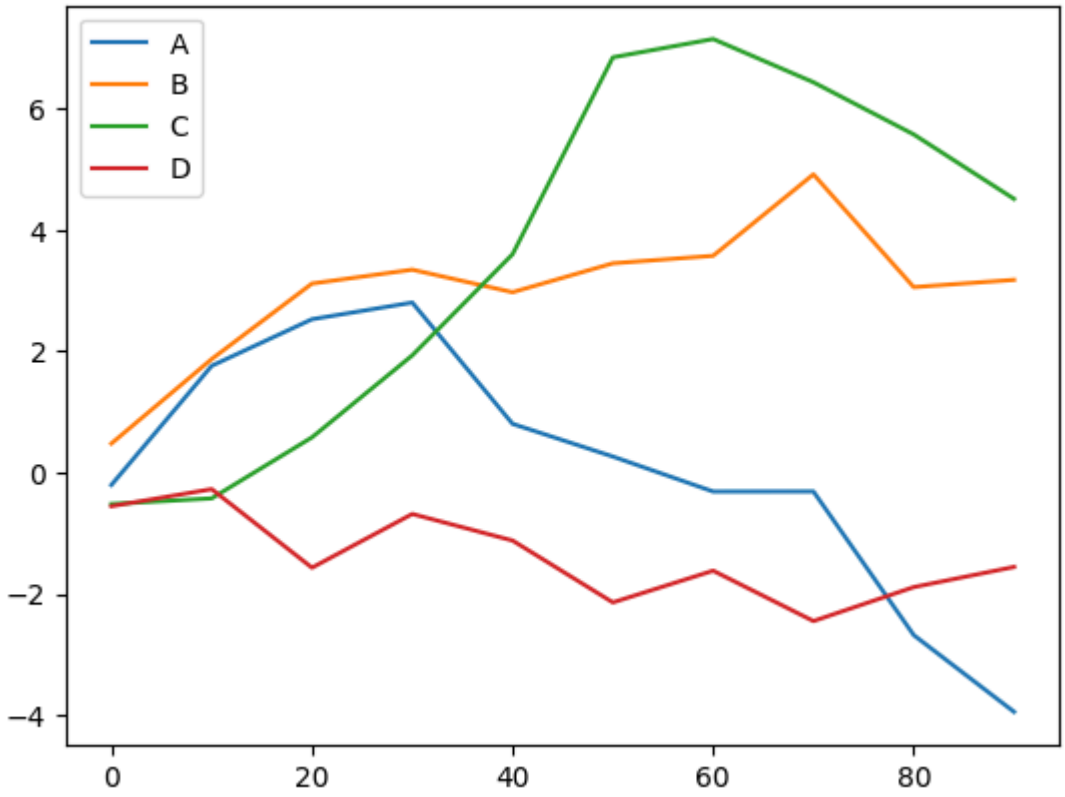
Line plots

In [31]:
```
# Using series data
rng = np.random.RandomState(12345)
```

```
s = pd.Series(rng.standard_normal(10).cumsum(),
              index=np.arange(0, 100, 10))
s.plot();
```



In [34]:
```
# Using Dataframe
rng = np.random.RandomState(12345)
df = pd.DataFrame(rng.standard_normal((10, 4)).cumsum(0),
                  columns=['A', 'B', 'C', 'D'],
                  index=np.arange(0, 100, 10))
df.plot();
```



In [35]:
```
# Show df content
df
```

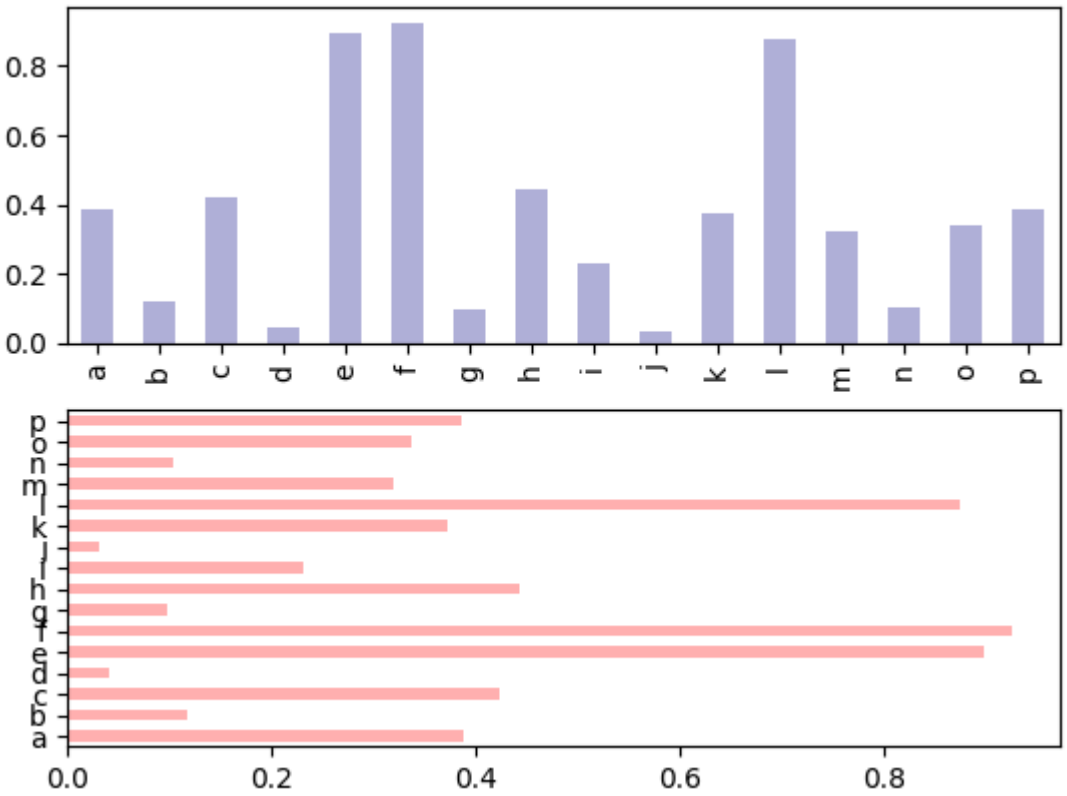
Out[35]:

	A	B	C	D
0	-0.204708	0.478943	-0.519439	-0.555730
10	1.761073	1.872349	-0.426531	-0.273984
20	2.530095	3.118784	0.580659	-1.570205
30	2.805087	3.347697	1.933575	-0.683776
40	0.803450	2.975854	3.602601	-1.122346
50	0.263708	3.452839	6.851545	-2.143573
60	-0.313379	3.576961	7.154158	-1.619801
70	-0.312439	4.920770	6.440614	-2.450955
80	-2.682670	3.060010	5.579857	-1.890809
90	-3.948605	3.179837	4.516344	-1.557927

Bar plots

In [40]:
```
# Series data
fig, axes = plt.subplots(2, 1)
data = pd.Series(np.random.uniform(size=16), index=list('abcdefghijklmnop'))

# Plot.bar is for plotting vertical bar
data.plot.bar(ax=axes[0], color='navy', alpha=0.3)
```

```
# Plot.barh is for plotting horizontal bar
data.plot.barh(ax=axes[1], color='red', alpha=0.3);
```

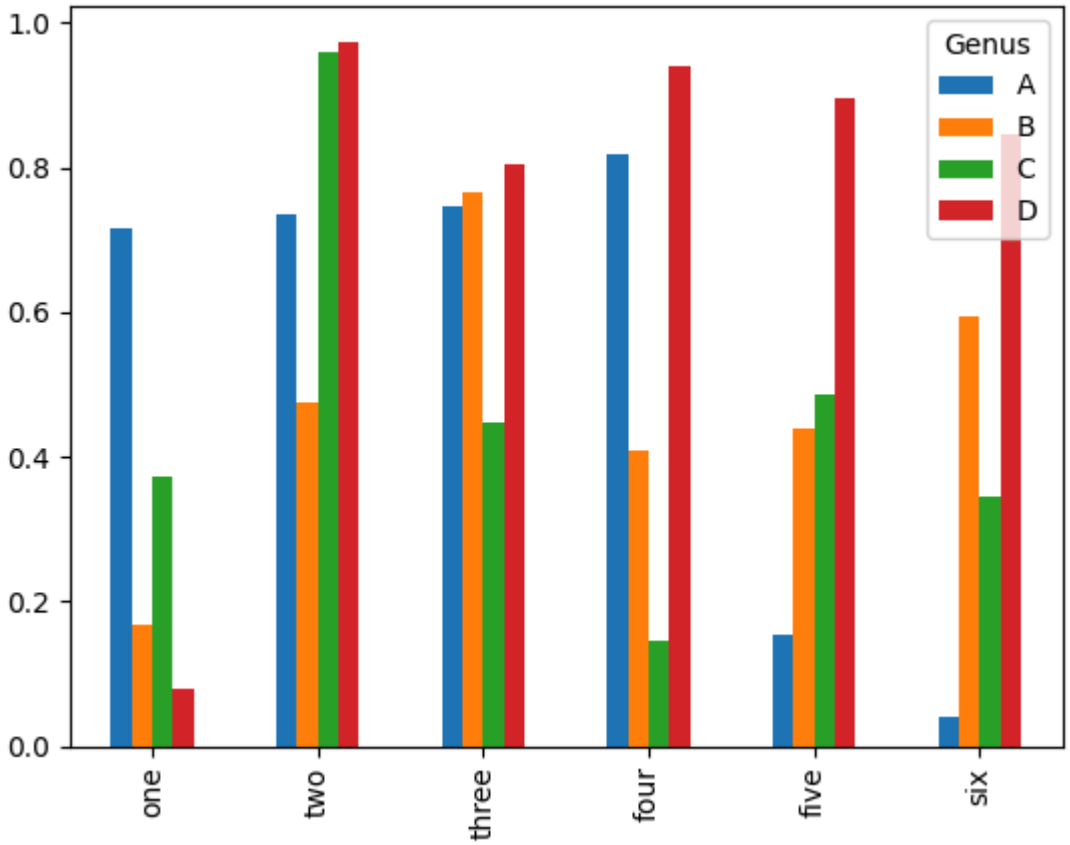


In [44]:
```
# Example DataFrame
df = pd.DataFrame(np.random.uniform(size=(6, 4)),
                  index=['one', 'two', 'three', 'four', 'five', 'six'],
                  columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
df
```

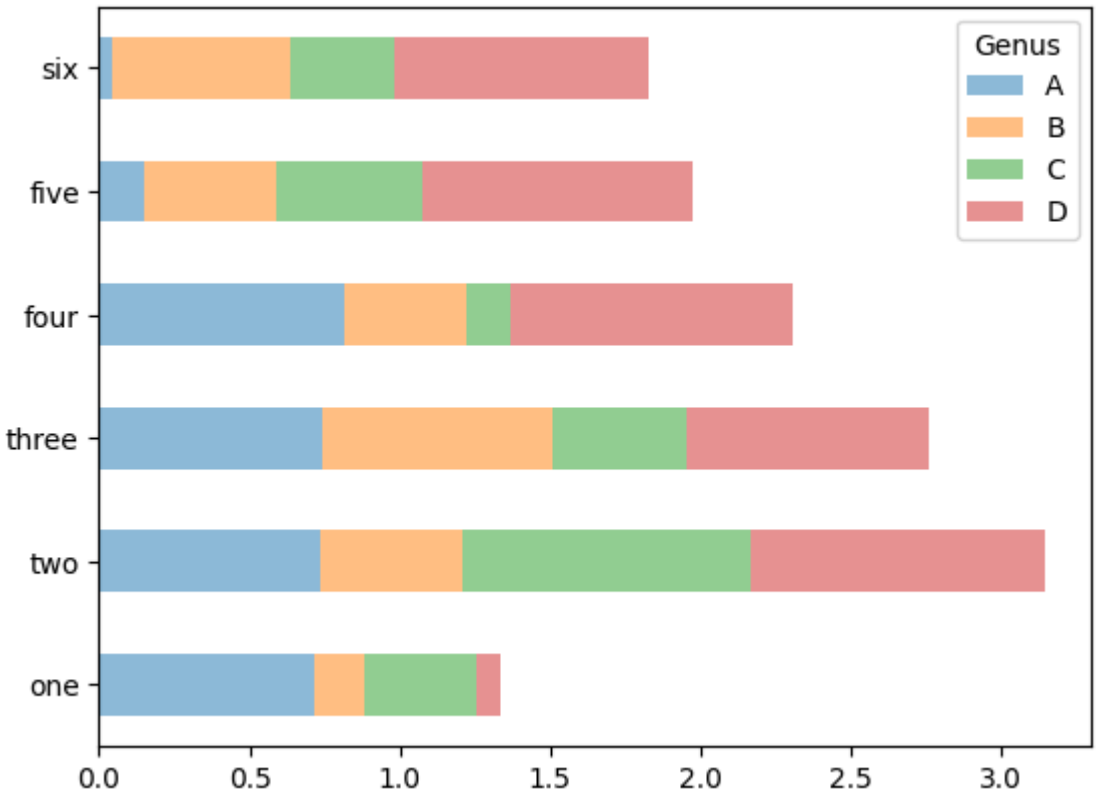
Out[44]:

Genus	A	B	C	D
one	0.715885	0.167505	0.371184	0.077939
two	0.734025	0.475724	0.960029	0.974574
three	0.745200	0.764236	0.446462	0.803392
four	0.816995	0.407603	0.146228	0.938993
five	0.152856	0.439007	0.485293	0.895518
six	0.041555	0.593589	0.344715	0.846730

In [45]:
```
# Bar plot
df.plot.bar();
```



In [46]:
```
# Stacked bar plots
df.plot.barh(stacked=True, alpha=0.5);
```



In [47]:
```
# A dataset about restaurant tipping
tips = pd.read_csv('https://bit.ly/3jCROII')
```

In [48]:
```
# Show tips content
```

```
tips
```

Out[48]:

	total_bill	tip	smoker	day	time	size
0	16.99	1.01	No	Sun	Dinner	2
1	10.34	1.66	No	Sun	Dinner	3
2	21.01	3.50	No	Sun	Dinner	3
3	23.68	3.31	No	Sun	Dinner	2
4	24.59	3.61	No	Sun	Dinner	4
...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3
240	27.18	2.00	Yes	Sat	Dinner	2
241	22.67	2.00	Yes	Sat	Dinner	2
242	17.82	1.75	No	Sat	Dinner	2
243	18.78	3.00	No	Thur	Dinner	2

244 rows × 6 columns

In [49]:
```
# Create a frequency table
party_counts = pd.crosstab(tips['day'], tips['size'], margins=True)
party_counts
```

Out[49]:

size	1	2	3	4	5	6	All
day							
Fri	1	16	1	1	0	0	19
Sat	2	53	18	13	1	0	87
Sun	0	39	15	18	3	1	76
Thur	1	48	4	5	1	3	62
All	4	156	38	37	5	4	244

In [50]:
```
# Reorder the index
party_counts = party_counts.reindex(index=['Thur', 'Fri', 'Sat', 'Sun'])
party_counts
```

Out[50]:

size	1	2	3	4	5	6	All
day							
Thur	1	48	4	5	1	3	62
Fri	1	16	1	1	0	0	19
Sat	2	53	18	13	1	0	87
Sun	0	39	15	18	3	1	76

In [51]:
```
# Extract party_count from col 2 to 5
party_counts = party_counts.loc[:, 2:5]
```

In [52]:
```
# Show content for party_counts
party_counts
```

Out[52]:

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

In [53]:
```
# Normalize each row to sum to 1
party_pcts = party_counts.div(party_counts.sum(axis='columns'),
                              axis='index')
party_pcts
```

Out[53]:

size	2	3	4	5
day				
Thur	0.827586	0.068966	0.086207	0.017241
Fri	0.888889	0.055556	0.055556	0.000000
Sat	0.623529	0.211765	0.152941	0.011765
Sun	0.520000	0.200000	0.240000	0.040000

In [54]:
```
# To check documentation for div() function
party_counts.div?
```

```
Signature:
party_counts.div(
    other,
    axis: 'Axis' = 'columns',
    level=None,
    fill_value=None,
) -> 'DataFrame'
Docstring:
Get Floating division of dataframe and other, element-wise (binary operator `true
div`).

Equivalent to ``dataframe / other``, but with support to substitute a fill_value
for missing data in one of the inputs. With reverse version, `rtruediv`.

Among flexible wrappers (`add`, `sub`, `mul`, `div`, `floordiv`, `mod`, `pow`) to
arithmetic operators: `+`, `-`, `*`, `/`, `//`, `%`, `**`.

Parameters
----------
other : scalar, sequence, Series, dict or DataFrame
    Any single or multiple element data structure, or list-like object.
axis : {0 or 'index', 1 or 'columns'}
    Whether to compare by the index (0 or 'index') or columns.
    (1 or 'columns'). For Series input, axis to match Series index on.
level : int or label
    Broadcast across a level, matching Index values on the
    passed MultiIndex level.
fill_value : float or None, default None
    Fill existing missing (NaN) values, and any new element needed for
    successful DataFrame alignment, with this value before computation.
    If data in both corresponding DataFrame locations is missing
    the result will be missing.

Returns
-------
DataFrame
    Result of the arithmetic operation.

See Also
--------
DataFrame.add : Add DataFrames.
DataFrame.sub : Subtract DataFrames.
DataFrame.mul : Multiply DataFrames.
DataFrame.div : Divide DataFrames (float division).
DataFrame.truediv : Divide DataFrames (float division).
DataFrame.floordiv : Divide DataFrames (integer division).
DataFrame.mod : Calculate modulo (remainder after division).
DataFrame.pow : Calculate exponential power.

Notes
-----
Mismatched indices will be unioned together.

Examples
--------
>>> df = pd.DataFrame({'angles': [0, 3, 4],
...                    'degrees': [360, 180, 360]},
...                   index=['circle', 'triangle', 'rectangle'])
>>> df
           angles  degrees
```

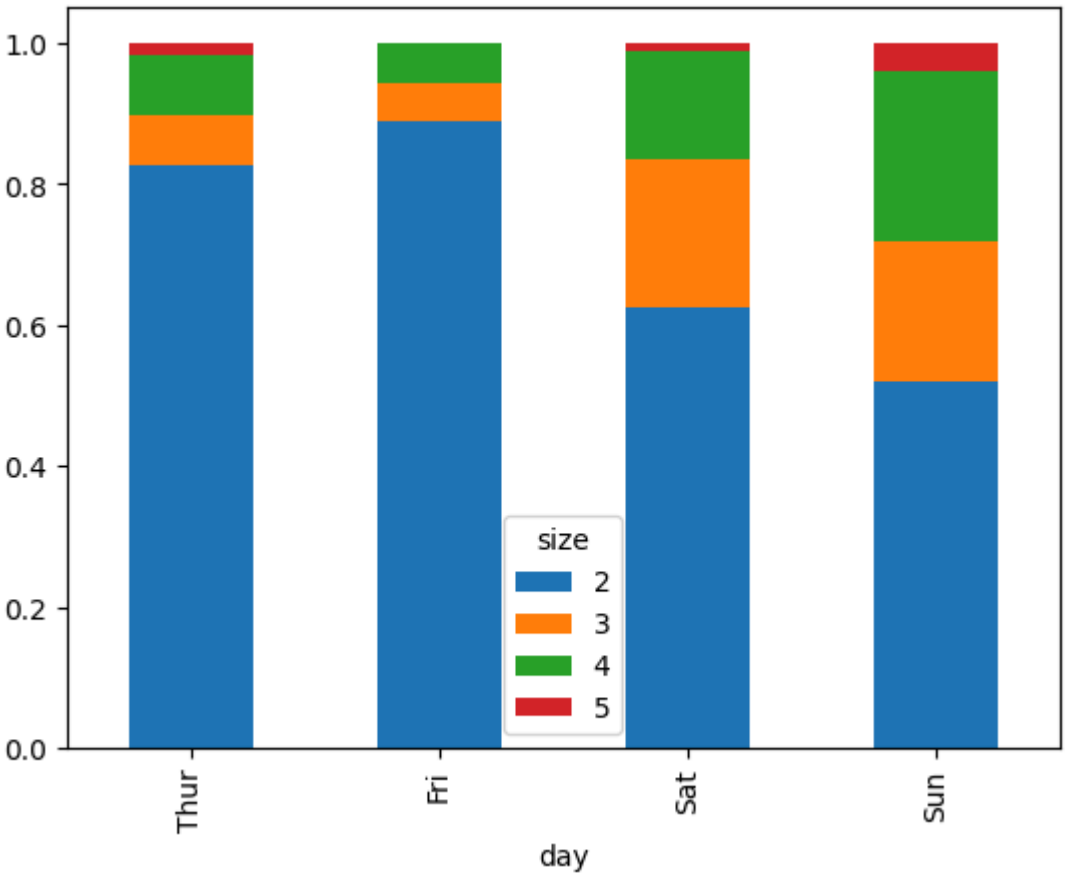
```
circle          0      360
triangle        3      180
rectangle       4      360

Add a scalar with operator version which return the same
results.

>>> df + 1
           angles  degrees
circle          1      361
triangle        4      181
rectangle       5      361

>>> df.add(1)
           angles  degrees
circle          1      361
triangle        4      181
rectangle       5      361

Divide by constant with reverse version.

>>> df.div(10)
           angles  degrees
circle        0.0     36.0
triangle      0.3     18.0
rectangle     0.4     36.0

>>> df.rdiv(10)
             angles   degrees
circle          inf  0.027778
triangle   3.333333  0.055556
rectangle  2.500000  0.027778

Subtract a list and Series by axis with operator version.

>>> df - [1, 2]
           angles  degrees
circle         -1      358
triangle        2      178
rectangle       3      358

>>> df.sub([1, 2], axis='columns')
           angles  degrees
circle         -1      358
triangle        2      178
rectangle       3      358

>>> df.sub(pd.Series([1, 1, 1], index=['circle', 'triangle', 'rectangle']),
...        axis='index')
           angles  degrees
circle         -1      359
triangle        2      179
rectangle       3      359

Multiply a dictionary by axis.

>>> df.mul({'angles': 0, 'degrees': 2})
            angles  degrees
circle           0      720
triangle         0      360
```

```
rectangle        0      720

>>> df.mul({'circle': 0, 'triangle': 2, 'rectangle': 3}, axis='index')
            angles  degrees
circle           0        0
triangle         6      360
rectangle       12     1080

Multiply a DataFrame of different shape with operator version.

>>> other = pd.DataFrame({'angles': [0, 3, 4]},
...                      index=['circle', 'triangle', 'rectangle'])
>>> other
           angles
circle          0
triangle        3
rectangle       4

>>> df * other
           angles  degrees
circle          0      NaN
triangle        9      NaN
rectangle      16      NaN

>>> df.mul(other, fill_value=0)
           angles  degrees
circle          0      0.0
triangle        9      0.0
rectangle      16      0.0

Divide by a MultiIndex by level.

>>> df_multindex = pd.DataFrame({'angles': [0, 3, 4, 4, 5, 6],
...                              'degrees': [360, 180, 360, 360, 540, 720]},
...                             index=[['A', 'A', 'A', 'B', 'B', 'B'],
...                                    ['circle', 'triangle', 'rectangle',
...                                     'square', 'pentagon', 'hexagon']])
>>> df_multindex
              angles  degrees
A circle          0      360
  triangle        3      180
  rectangle       4      360
B square          4      360
  pentagon        5      540
  hexagon         6      720

>>> df.div(df_multindex, level=1, fill_value=0)
              angles  degrees
A circle         NaN      1.0
  triangle       1.0      1.0
  rectangle      1.0      1.0
B square         0.0      0.0
  pentagon       0.0      0.0
  hexagon        0.0      0.0
File:      c:\users\hazim\appdata\local\programs\python\python312\lib\site-packag
es\pandas\core\frame.py
Type:      method
```

In [55]:
```
# Manually check the vaue
4/58
```

Out[55]: 0.06896551724137931

In [59]:
```
# Stacked bar plot
party_pcts.plot.bar(stacked=True);
```



Using seaborn package

In [62]:
```
# Import Seaborn package
import seaborn as sns
```

In [63]:
```
# Show content for tips
tips
```

Out[63]:

	total_bill	tip	smoker	day	time	size
0	16.99	1.01	No	Sun	Dinner	2
1	10.34	1.66	No	Sun	Dinner	3
2	21.01	3.50	No	Sun	Dinner	3
3	23.68	3.31	No	Sun	Dinner	2
4	24.59	3.61	No	Sun	Dinner	4
...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3
240	27.18	2.00	Yes	Sat	Dinner	2
241	22.67	2.00	Yes	Sat	Dinner	2
242	17.82	1.75	No	Sat	Dinner	2
243	18.78	3.00	No	Thur	Dinner	2

244 rows × 6 columns

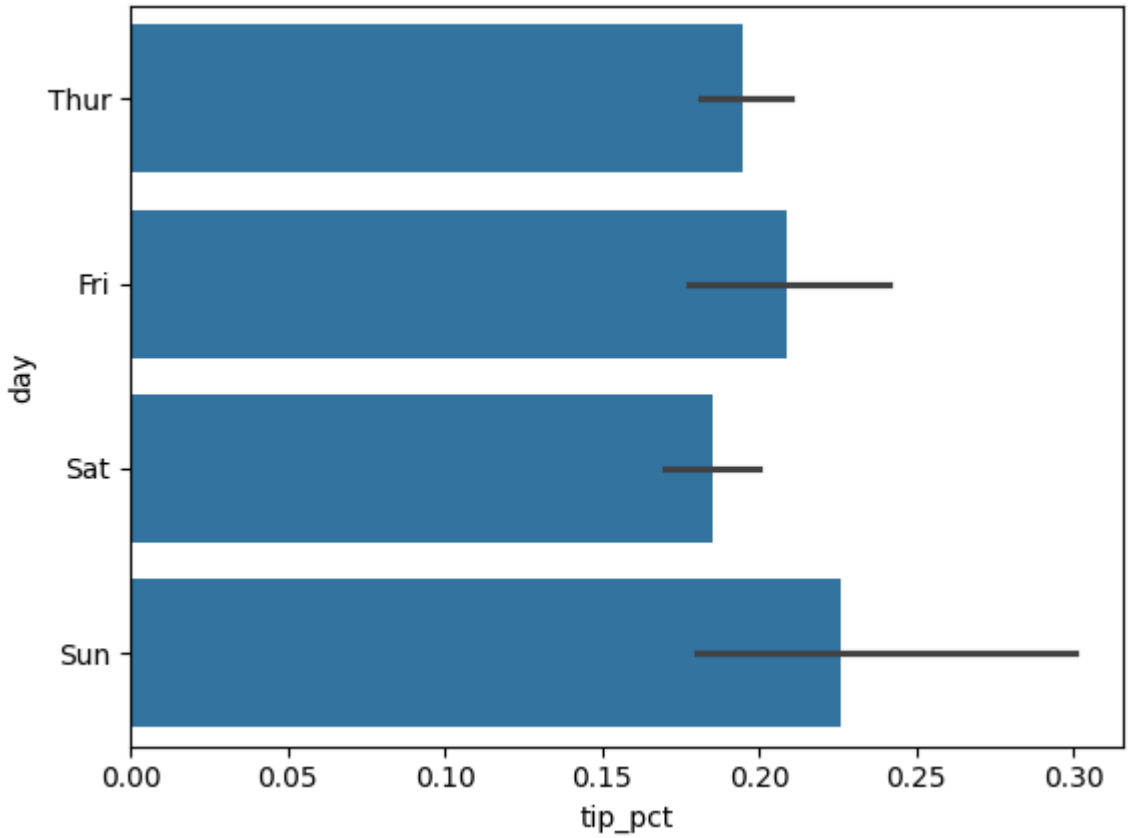
In [64]:
```
# Add a new column for 'tip_pct'
tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])
```

In [65]:
```
# Show top five rows
tips.head()
```

Out[65]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069

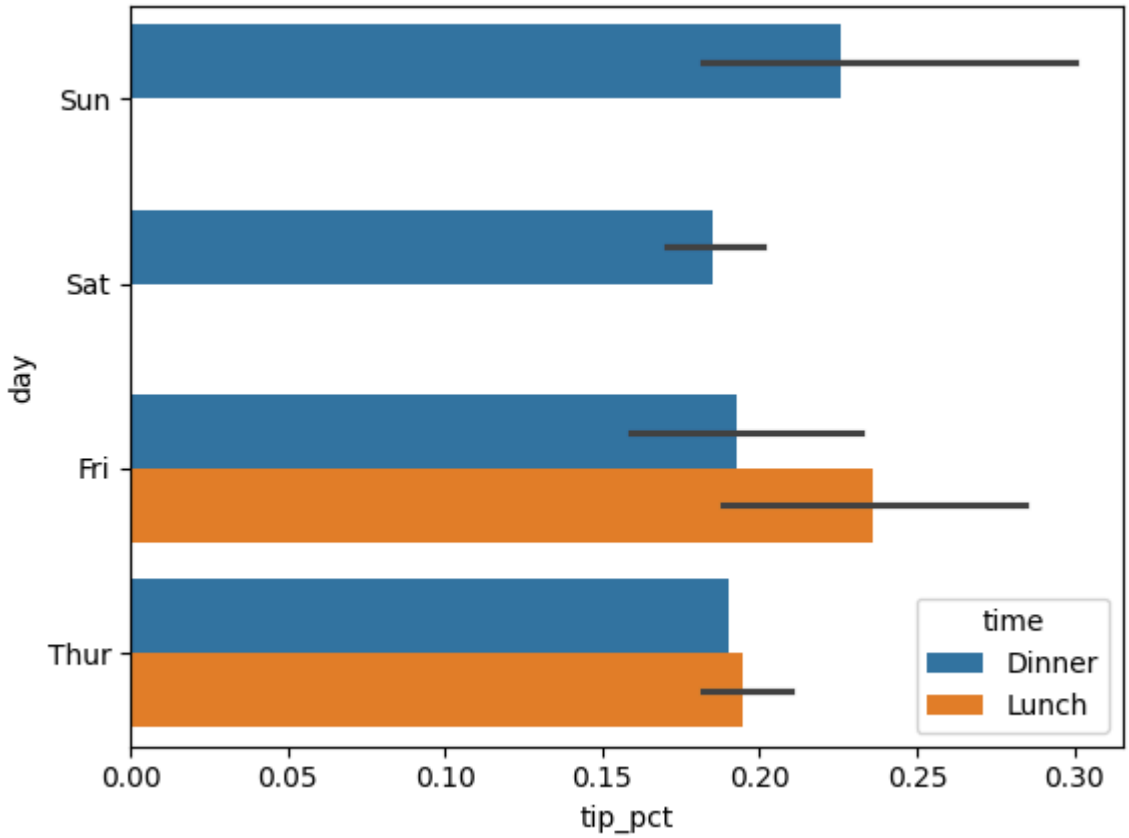
In [66]:
```
# Using black lines on the bars to represent 95% confidence interval
sns.barplot(x='tip_pct', y='day', data=tips, orient='h',
            order=['Thur', 'Fri', 'Sat', 'Sun']);
```



hue

- `hue` helps to visualize the impact of an **additional categorical variable** on our main comparison.

In [67]:
```
# Using hue to add a categorical distinction
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h',
            order=['Sun', 'Sat', 'Fri', 'Thur']);
```



In [68]:
```
# To check documentation for sns.barplot
sns.barplot?
```

```
Signature:
sns.barplot(
    data=None,
    *,
    x=None,
    y=None,
    hue=None,
    order=None,
    hue_order=None,
    estimator='mean',
    errorbar=('ci', 95),
    n_boot=1000,
    seed=None,
    units=None,
    weights=None,
    orient=None,
    color=None,
    palette=None,
    saturation=0.75,
    fill=True,
    hue_norm=None,
    width=0.8,
    dodge='auto',
    gap=0,
    log_scale=None,
    native_scale=False,
    formatter=None,
    legend='auto',
    capsize=0,
    err_kws=None,
    ci=<deprecated>,
    errcolor=<deprecated>,
    errwidth=<deprecated>,
    ax=None,
    **kwargs,
)
Docstring:
Show point estimates and errors as rectangular bars.

A bar plot represents an aggregate or statistical estimate for a numeric
variable with the height of each rectangle and indicates the uncertainty
around that estimate using an error bar. Bar plots include 0 in the
axis range, and they are a good choice when 0 is a meaningful value
for the variable to take.

See the :ref:`tutorial <categorical_tutorial>` for more information.

.. note::
    By default, this function treats one of the variables as categorical
    and draws data at ordinal positions (0, 1, ... n) on the relevant axis.
    As of version 0.13.0, this can be disabled by setting `native_scale=True`.


Parameters
----------
data : DataFrame, Series, dict, array, or list of arrays
    Dataset for plotting. If `x` and `y` are absent, this is
    interpreted as wide-form. Otherwise it is expected to be long-form.
x, y, hue : names of variables in `data` or vector data
    Inputs for plotting long-form data. See examples for interpretation.
```

```
order, hue_order : lists of strings
    Order to plot the categorical levels in; otherwise the levels are
    inferred from the data objects.
estimator : string or callable that maps vector -> scalar
    Statistical function to estimate within each categorical bin.
errorbar : string, (string, number) tuple, callable or None
    Name of errorbar method (either "ci", "pi", "se", or "sd"), or a tuple
    with a method name and a level parameter, or a function that maps from a
    vector to a (min, max) interval, or None to hide errorbar. See the
    :doc:`errorbar tutorial </tutorial/error_bars>` for more information.

    .. versionadded:: v0.12.0
n_boot : int
    Number of bootstrap samples used to compute confidence intervals.
seed : int, `numpy.random.Generator`, or `numpy.random.RandomState`
    Seed or random number generator for reproducible bootstrapping.
units : name of variable in `data` or vector data
    Identifier of sampling units; used by the errorbar function to
    perform a multilevel bootstrap and account for repeated measures
weights : name of variable in `data` or vector data
    Data values or column used to compute weighted statistics.
    Note that the use of weights may limit other statistical options.

    .. versionadded:: v0.13.1
orient : "v" | "h" | "x" | "y"
    Orientation of the plot (vertical or horizontal). This is usually
    inferred based on the type of the input variables, but it can be used
    to resolve ambiguity when both `x` and `y` are numeric or when
    plotting wide-form data.

    .. versionchanged:: v0.13.0
        Added 'x'/'y' as options, equivalent to 'v'/'h'.
color : matplotlib color
    Single color for the elements in the plot.
palette : palette name, list, or dict
    Colors to use for the different levels of the ``hue`` variable. Should
    be something that can be interpreted by :func:`color_palette`, or a
    dictionary mapping hue levels to matplotlib colors.
saturation : float
    Proportion of the original saturation to draw fill colors in. Large
    patches often look better with desaturated colors, but set this to
    `1` if you want the colors to perfectly match the input values.
fill : bool
    If True, use a solid patch. Otherwise, draw as line art.

    .. versionadded:: v0.13.0
hue_norm : tuple or :class:`matplotlib.colors.Normalize` object
    Normalization in data units for colormap applied to the `hue`
    variable when it is numeric. Not relevant if `hue` is categorical.

    .. versionadded:: v0.12.0
width : float
    Width allotted to each element on the orient axis. When `native_scale=True`,
    it is relative to the minimum distance between two values in the native scal
e.
dodge : "auto" or bool
    When hue mapping is used, whether elements should be narrowed and shifted alo
ng
    the orient axis to eliminate overlap. If `"auto"`, set to `True` when the
    orient variable is crossed with the categorical variable or `False` otherwis
```

```
e.

    .. versionchanged:: 0.13.0

        Added `"auto"` mode as a new default.
gap : float
    Shrink on the orient axis by this factor to add a gap between dodged element
s.

    .. versionadded:: 0.13.0
log_scale : bool or number, or pair of bools or numbers
    Set axis scale(s) to log. A single value sets the data axis for any numeric
    axes in the plot. A pair of values sets each axis independently.
    Numeric values are interpreted as the desired base (default 10).
    When `None` or `False`, seaborn defers to the existing Axes scale.

    .. versionadded:: v0.13.0
native_scale : bool
    When True, numeric or datetime values on the categorical axis will maintain
    their original scaling rather than being converted to fixed indices.

    .. versionadded:: v0.13.0
formatter : callable
    Function for converting categorical data into strings. Affects both grouping
    and tick labels.

    .. versionadded:: v0.13.0
legend : "auto", "brief", "full", or False
    How to draw the legend. If "brief", numeric `hue` and `size`
    variables will be represented with a sample of evenly spaced values.
    If "full", every group will get an entry in the legend. If "auto",
    choose between brief or full representation based on number of levels.
    If `False`, no legend data is added and no legend is drawn.

    .. versionadded:: v0.13.0
capsize : float
    Width of the "caps" on error bars, relative to bar spacing.
err_kws : dict
    Parameters of :class:`matplotlib.lines.Line2D`, for the error bar artists.

    .. versionadded:: v0.13.0
ci : float
    Level of the confidence interval to show, in [0, 100].

    .. deprecated:: v0.12.0
        Use `errorbar=("ci", ...)`.
errcolor : matplotlib color
    Color used for the error bar lines.

    .. deprecated:: 0.13.0
        Use `err_kws={'color': ...}`.
errwidth : float
    Thickness of error bar lines (and caps), in points.

    .. deprecated:: 0.13.0
        Use `err_kws={'linewidth': ...}`.
ax : matplotlib Axes
    Axes object to draw the plot onto, otherwise uses the current Axes.
kwargs : key, value mappings
    Other parameters are passed through to :class:`matplotlib.patches.Rectangle`.
```

```
Returns
-------
ax : matplotlib Axes
    Returns the Axes object with the plot drawn onto it.

See Also
--------
countplot : Show the counts of observations in each categorical bin.
pointplot : Show point estimates and confidence intervals using dots.
catplot : Combine a categorical plot with a :class:`FacetGrid`.

Notes
-----

For datasets where 0 is not a meaningful value, a :func:`pointplot` will
allow you to focus on differences between levels of one or more categorical
variables.

It is also important to keep in mind that a bar plot shows only the mean (or
other aggregate) value, but it is often more informative to show the
distribution of values at each level of the categorical variables. In those
cases, approaches such as a :func:`boxplot` or :func:`violinplot` may be
more appropriate.

Examples
--------
.. include:: ../docstrings/barplot.rst
File:      c:\users\hazim\appdata\local\programs\python\python312\lib\site-packag
es\seaborn\categorical.py
Type:      function
```

In [69]:
```
# Display contents for tips
tips
```

Out[69]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069
...	...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3	0.256166
240	27.18	2.00	Yes	Sat	Dinner	2	0.079428
241	22.67	2.00	Yes	Sat	Dinner	2	0.096759
242	17.82	1.75	No	Sat	Dinner	2	0.108899
243	18.78	3.00	No	Thur	Dinner	2	0.190114

244 rows × 7 columns

Histograms and Density Plots

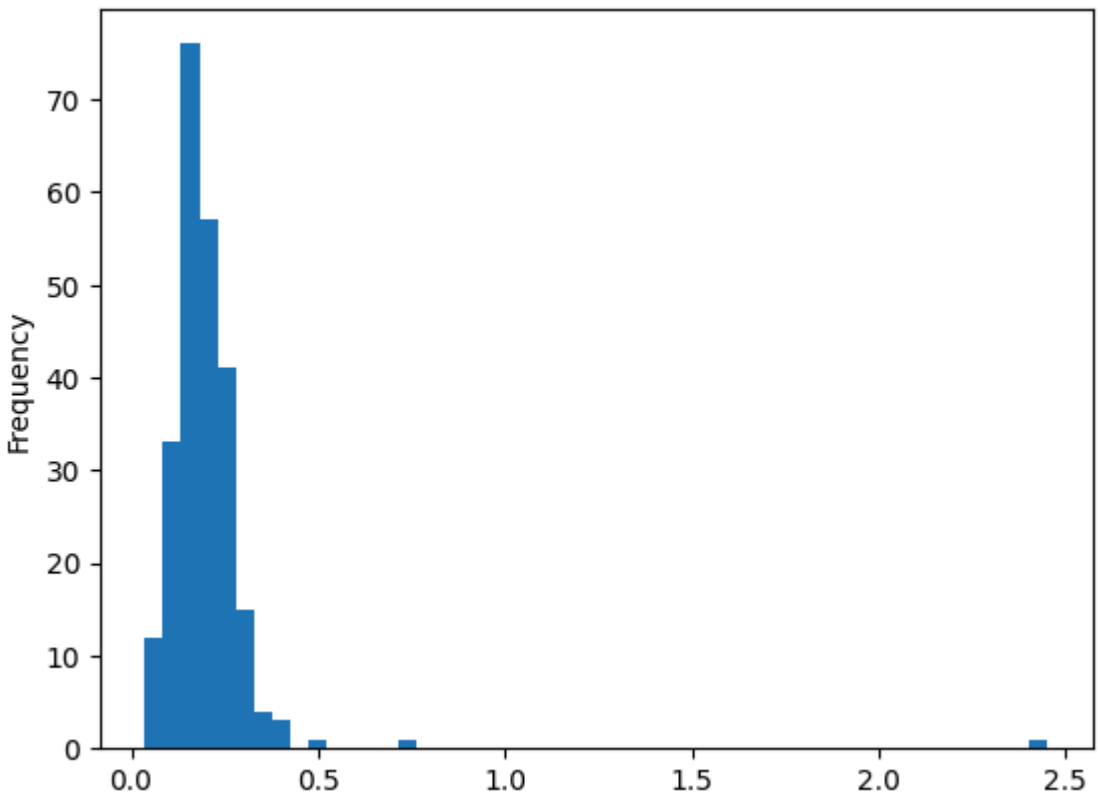
discretized display of **value frequency**

In [72]:
```
# Display top few rows
tips.head()
```

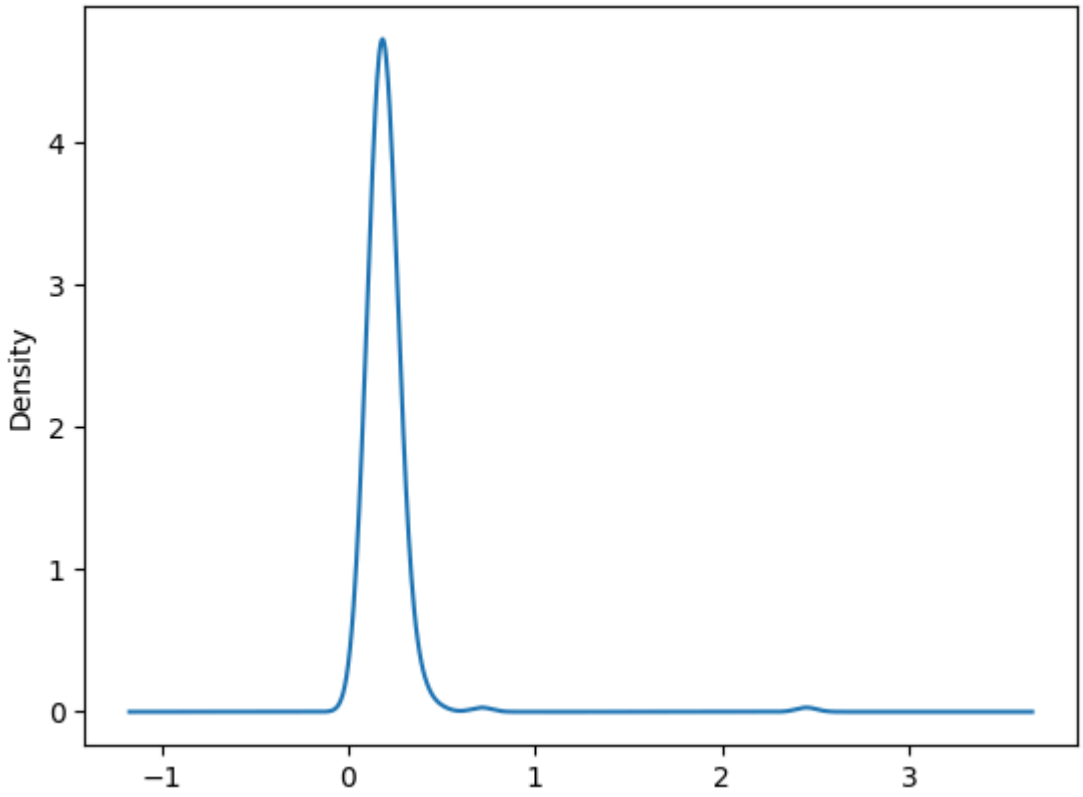
Out[72]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069

In [73]:
```
# Histogram
tips['tip_pct'].plot.hist(bins=50);
```



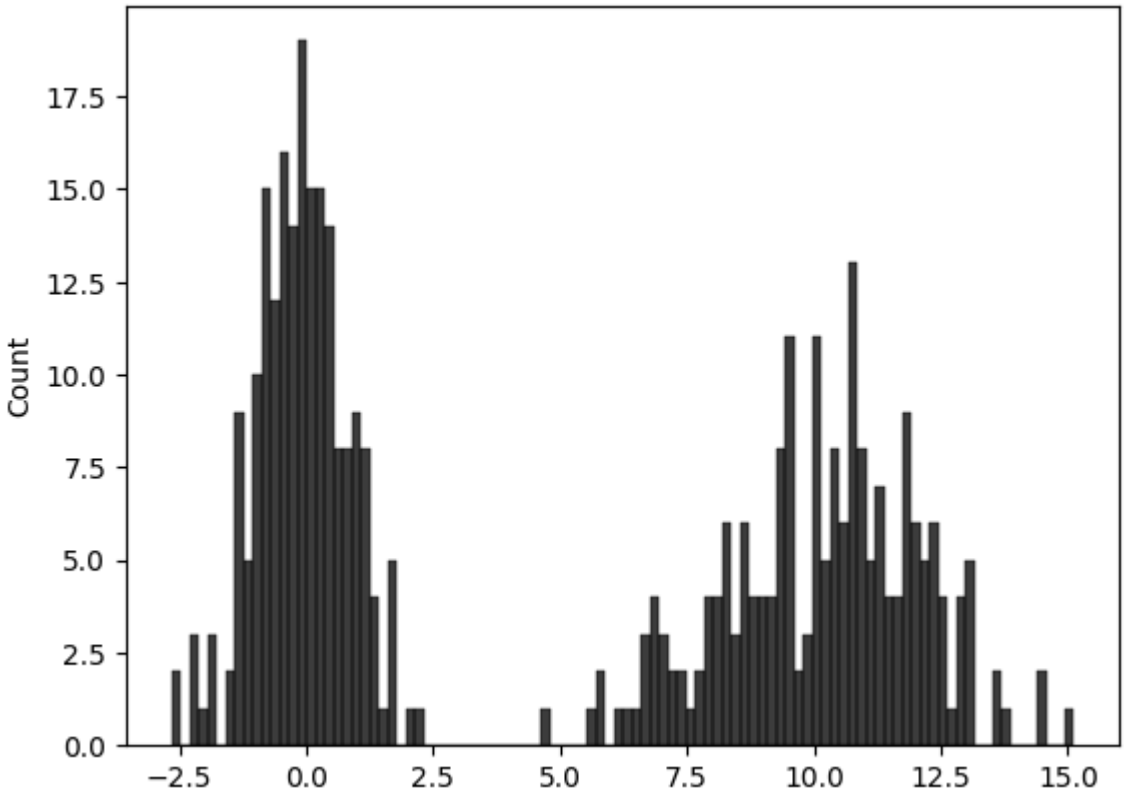
In [ ]:
```
# Density plot
tips['tip_pct'].plot.density();
```



In [74]:
```
# Example dataset
comp1 = np.random.standard_normal(200)
comp2 = 10 + 2 * np.random.standard_normal(200)
values = pd.Series(np.concatenate([comp1, comp2]))

# Display content for values
values
```

Out[74]:
```
0      -0.313559
1      -0.367684
2      -0.034959
3      -0.089517
4      -0.728365
         ...
395    10.421917
396     7.855848
397    12.561823
398     9.259135
399    10.066034
Length: 400, dtype: float64
```

In [75]:
```
# Plot histogram
sns.histplot(values, bins=100, color='black');
```



Scatter or Point Plots

examining the relationship between two one-dimensional data series

Macro dataset

- Collection of economic data points that reflect the overall performance and health of a country, region, or the global economy.

In [76]:
```
# Example macro dataset
macro = pd.read_csv('https://bit.ly/3FYxCYZ')
macro
```

Out[76]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	t
0	1959	1	2710.349	1707.4	286.898	470.045	1886.9	28.980	139.7	
1	1959	2	2778.801	1733.7	310.859	481.301	1919.7	29.150	141.7	
2	1959	3	2775.488	1751.8	289.226	491.260	1916.4	29.350	140.5	
3	1959	4	2785.204	1753.7	299.356	484.052	1931.3	29.370	140.0	
4	1960	1	2847.699	1770.5	331.722	462.199	1955.5	29.540	139.6	
...	...	...	...	...	...	...	...	...	...	
198	2008	3	13324.600	9267.7	1990.693	991.551	9838.3	216.889	1474.7	
199	2008	4	13141.920	9195.3	1857.661	1007.273	9920.4	212.174	1576.5	
200	2009	1	12925.410	9209.2	1558.494	996.287	9926.4	212.671	1592.8	
201	2009	2	12901.504	9189.0	1456.678	1023.528	10077.5	214.469	1653.6	
202	2009	3	12990.341	9256.0	1486.398	1044.088	10040.6	216.385	1673.9	

203 rows × 14 columns

In [77]:
```
# Only select a few variables
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
data
```

Out[77]:

	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

203 rows × 4 columns

In [78]:
```
# Applies a logarithmic transformation
# Calculates the difference between consecutive values
# Removes any rows with missing values (NaN)
trans_data = np.log(data).diff().dropna()
```

```
# Display content
trans_data
```

Out[78]:

	cpi	m1	tbilrate	unemp
1	0.005849	0.014215	0.088193	-0.128617
2	0.006838	-0.008505	0.215321	0.038466
3	0.000681	-0.003565	0.125317	0.055060
4	0.005772	-0.002861	-0.212805	-0.074108
5	0.000338	0.004289	-0.266946	0.000000
...	...	...	...	...
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

202 rows × 4 columns

In [79]:
```
# Display top few rows for data
data.head()
```

Out[79]:

	cpi	m1	tbilrate	unemp
0	28.98	139.7	2.82	5.8
1	29.15	141.7	3.08	5.1
2	29.35	140.5	3.82	5.3
3	29.37	140.0	4.33	5.6
4	29.54	139.6	3.50	5.2

In [80]:
```
# Check manually
np.log(29.150)-np.log(28.980)
```

Out[80]:
```
np.float64(0.005848975903965048)
```

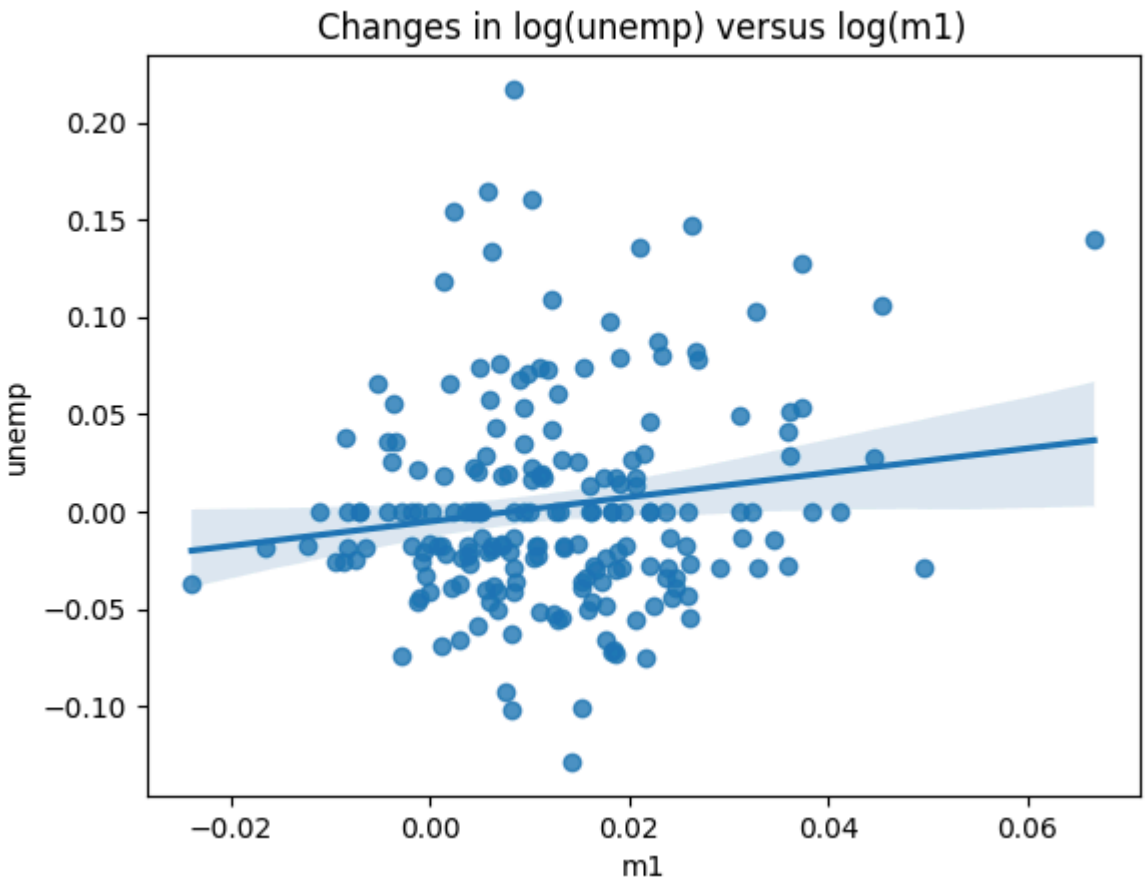
In [81]:
```
# Display contents for trans_data
trans_data
```

Out[81]:

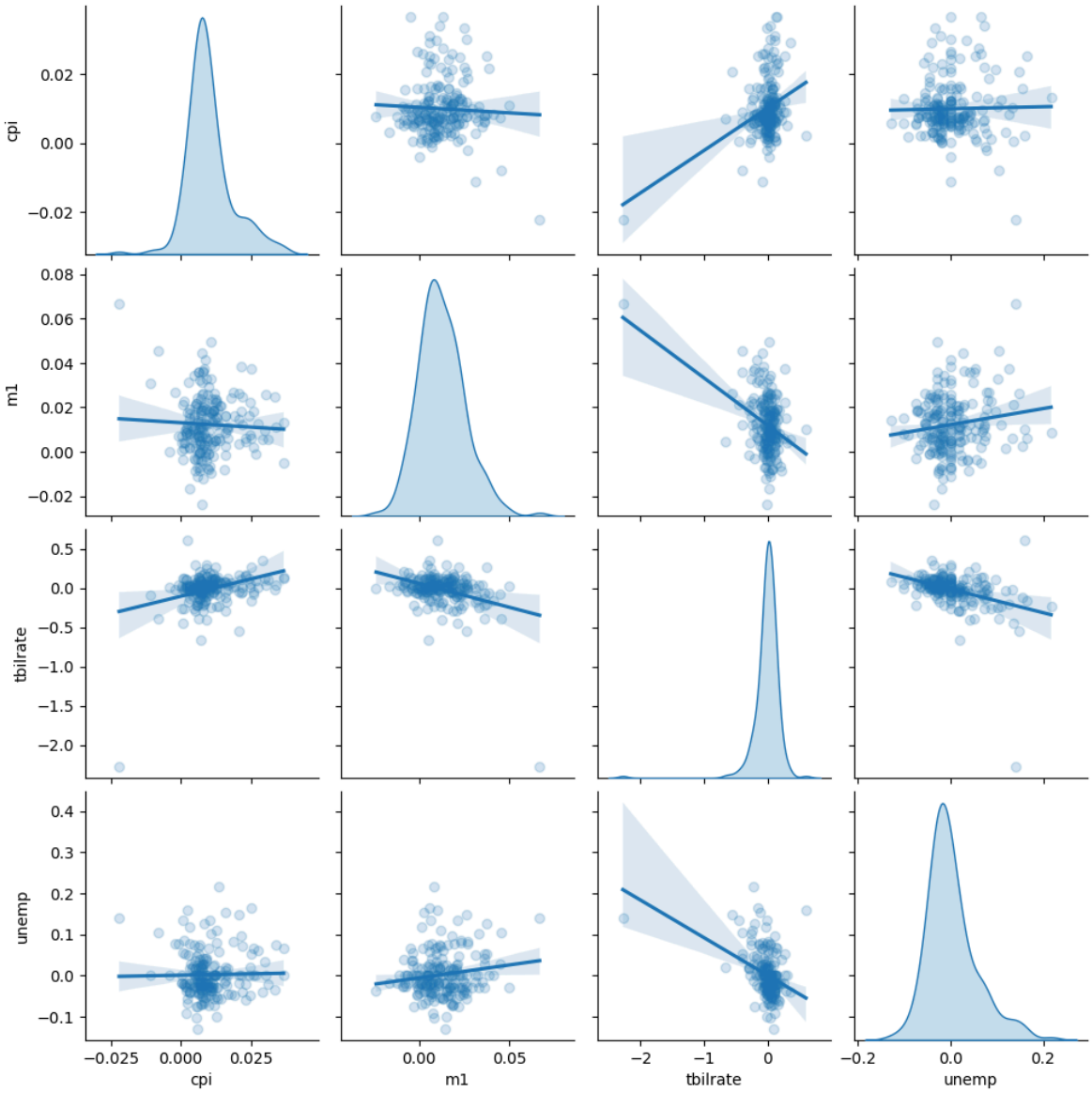
	cpi	m1	tbilrate	unemp
1	0.005849	0.014215	0.088193	-0.128617
2	0.006838	-0.008505	0.215321	0.038466
3	0.000681	-0.003565	0.125317	0.055060
4	0.005772	-0.002861	-0.212805	-0.074108
5	0.000338	0.004289	-0.266946	0.000000
...	...	...	...	...
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-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

202 rows × 4 columns

In [82]:
```
# Generate scatter plot with regression line
fig = sns.regplot(x='m1', y='unemp', data=trans_data)

# Add a descriptive title that summarizes the plot's content
fig.set(title='Changes in log(unemp) versus log(m1)');
```



In [83]:
```
# kde = kernel density estimate - another name for density plot
# Estimate of the continuous probability distribution
sns.pairplot(trans_data, diag_kind='kde', kind='reg',
             plot_kws={'scatter_kws': {'alpha': 0.2}});
```



Facet Grids and Categorical Data

visualize data with many ***categorical variables***

Seaborn **catplot** function simplifies making many kinds of faceted plots split by ***categorical variables***

faceted means having many sides

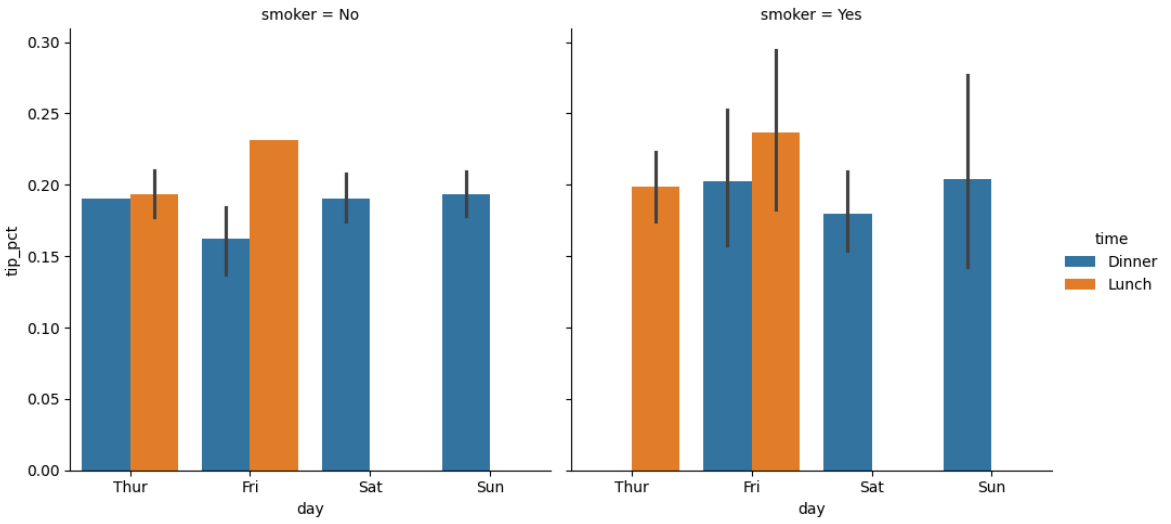
In [84]:
```
# Display contents for tips dataframe
tips
```

Out[84]:

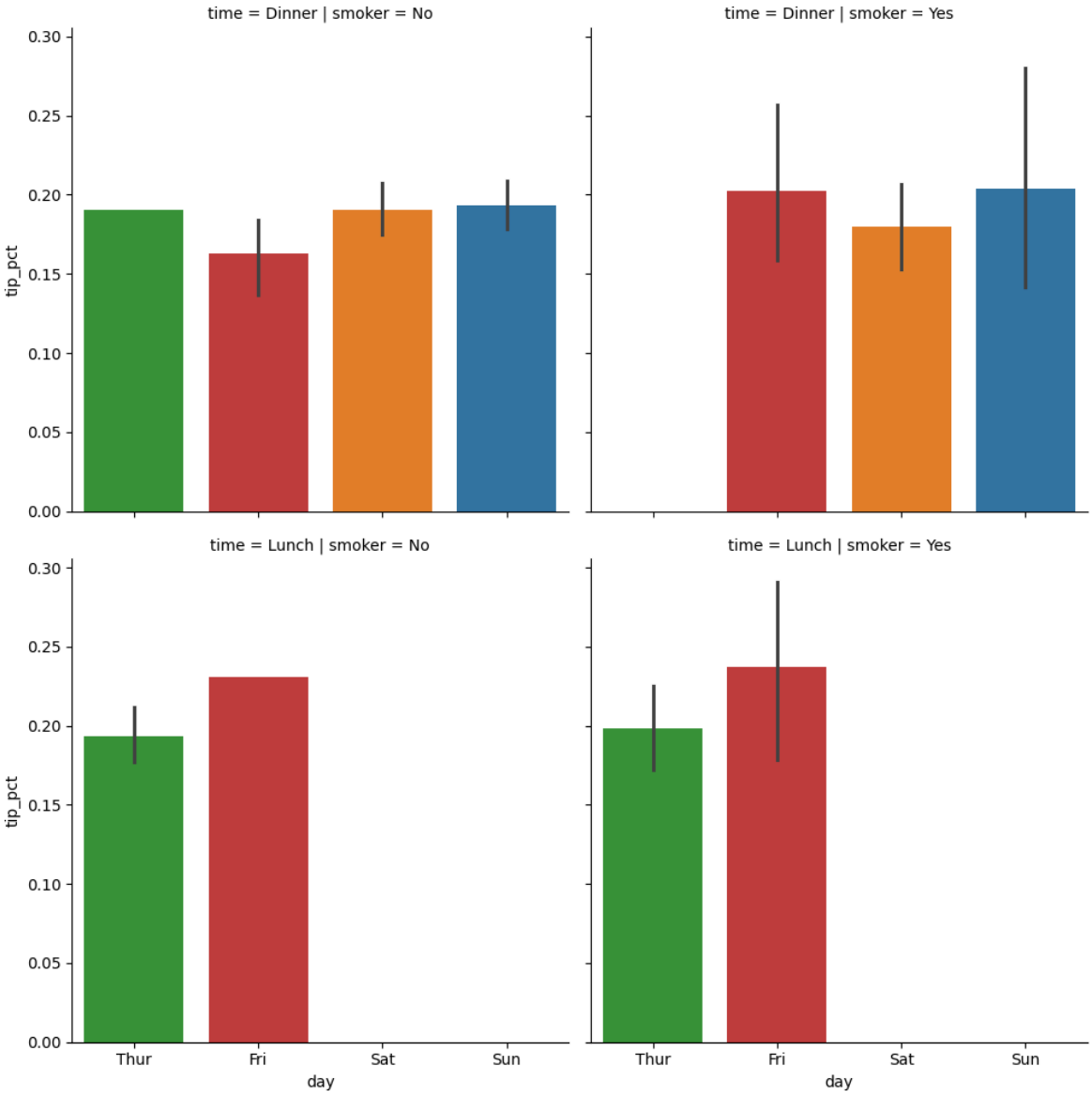
	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.063204
1	10.34	1.66	No	Sun	Dinner	3	0.191244
2	21.01	3.50	No	Sun	Dinner	3	0.199886
3	23.68	3.31	No	Sun	Dinner	2	0.162494
4	24.59	3.61	No	Sun	Dinner	4	0.172069
...	...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3	0.256166
240	27.18	2.00	Yes	Sat	Dinner	2	0.079428
241	22.67	2.00	Yes	Sat	Dinner	2	0.096759
242	17.82	1.75	No	Sat	Dinner	2	0.108899
243	18.78	3.00	No	Thur	Dinner	2	0.190114

244 rows × 7 columns

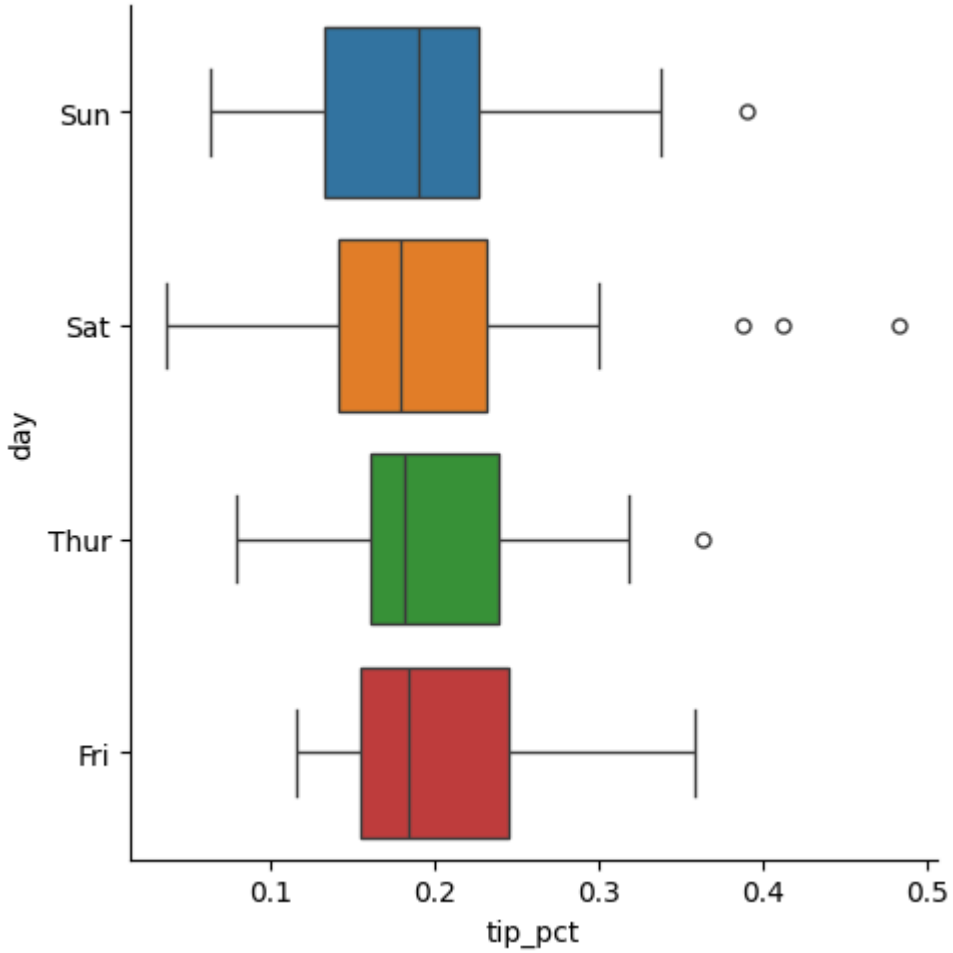
In [85]:
```
# Generate categorical plot
sns.catplot(x='day', y='tip_pct', hue='time', col='smoker',
            kind='bar', data=tips[tips.tip_pct < 1],
            order=['Thur', 'Fri', 'Sat', 'Sun']);
```



In [86]:
```
# Generate categorical plot
sns.catplot(x='day', y='tip_pct', row='time',
            col='smoker', hue='day', order=['Thur', 'Fri', 'Sat', 'Sun'],
            kind='bar', data=tips[tips.tip_pct < 1]);
```



In [87]:
```
# Generate categorical plot: boxplot
sns.catplot(x='tip_pct', y='day', hue='day', kind='box',
            data=tips[tips.tip_pct < 0.5]);
```



That's it for the day!!! :)

Please join this Whatsapp group for general communication and updates

STQD6014_S1_20242025_ArusPerdana

WhatsApp group