oreilly-notes

Smarter Plotting

A Seminar by ‘Don’t Use This Code’

DO(N’T) USE THIS C<>DE

Presenter: James Powell james@dutc.io

Contents

`tmate`

Quick Link (open in new tab): `tmate` terminal screen-sharing

If your video conference quality ever becomes blurry, feel free to use the above `tmate` link for a crystal clear image of the presenter’s terminal!

Error: Not Found

Book a Class!

Book a class or training for your team!

Please reach out to us at learning@dutc.io if are interested in bringing this material, or any of our other material, to your team.

We have courses on topics such as:

- intro Python
- expert Python
- data engineering with Python
- data science and scientific computing with `numpy` , `pandas` , and `xarray`

If you reach out to us, we can also provide a printable copy of the notes, as well as a professionally edited video recording of this presentation.

About

Don’t Use This Code; Training & Consulting

Don’t Use This Code is a professional training, coaching, and consulting company. We are deeply invested in the open source scientific computing community, and are dedicated to bringing better processes, better tools, and better understanding to the world.

Don’t Use This Code is growing! We are currently seeking new partners, new clients, and new engagements for our expert consulting and training services.

Our ideal client is an organization, large or small, using open source technologies, centering around the PyData stack for scientififc and numeric computing. Organizations looking to better employ these tools would benefit from our wide range of training courses on offer, ranging from an intensive introduction to Python fundamentals to advanced applications of Python for building large-scale, production systems. Working with your team, we can craft targeted curricula to meet your training goals. We are also available for consulting services such as building scientific computing and numerical analysis systems using technologies like Python and React.

We pride ourselves on delivering top-notch training. We are committed to providing quality training that is uniquely valuable to each individual attendee, and we do so by investing in three key areas: our content, our processes, and our contributors.

James Powell; now, he uses his experience as a consultant for

those building data engineering and scientific computing platforms for a wide range of clients using cutting-edge open source tools like Python and React.

He also currently serves as a Board Director, Chair, and Vice President at NumFOCUS, the 501(c)3 non-profit that supports all the major tools in the Python data analysis ecosystem (i.e., pandas, numpy, jupyter, matplotlib). At NumFOCUS, he helps build global open source communities for data scientists, data engineers, and business analysts. He helps NumFOCUS run the PyData conference series and has sat on speaker selection and organizing committees for 18 conferences. James is also a prolific speaker: since 2013, he has given over seventy (70) conference talks at over fifty (50) Python events worldwide.

Cameron Riddell

Cameron Riddell is the newest instructor at DUTC. Prior to joining our team, he worked in academia studying various aspects of psychology, including the neural activity underlying social interaction, perception of taste, and human memory. Throughout his time in academia, he developed a passion for programming and has been using Python in his work and personal life for the past 8 years. Having answered hundreds of questions on StackOverflow, he brings a keen insight into common struggles with Python and is eager to share his knowledge while furthering his own Python expertise.

Notes: Smarter Plotting

why do we visualize?

- communicate message: highlight specific
- exploratory: understand our data

Better Matplotlib

```
print("Let's Get Started!")
```

```
from numpy import linspace, pi, sin, cos
from matplotlib.pyplot import plot, show

xs = linspace(0, 2 * pi, 200)

plot(xs, sin(xs), lw=4)
plot(xs, cos(xs), lw=4)

show()
```

You can draw anything

```
from matplotlib.pyplot import figure, show
from matplotlib.patches import Circle, Rectangle

fig = figure(figsize=(6,6))

# c = Circle((.5, .75), .1)
# fig.add_artist(c)

# body_rect = Rectangle((.47, .75), .05, -.5)
# fig.add_artist(body_rect)

# arms_rect = Rectangle((.35, .45), .3, .05)
# fig.add_artist(arms_rect)

# lleg_rect = Rectangle((.49, .3), .25, .05, angle=225)
# fig.add_artist(lleg_rect)

# rleg_rect = Rectangle((.46, .26), .25, .05, angle=-45)
# fig.add_artist(rleg_rect)

show()
```

what else can I do with a figure?

```
from matplotlib.pyplot import subplots, show, figure

fig = figure(dpi=200)
```

```
# Are these methods magic? How else do we make text?
# fig.suptitle('This is my Figure Title')
# fig.supylabel('This is my Figure ylabel')
# fig.supxlabel('This is my Figure xlabel')
# fig.text(.5, .9, 'This is my title', ha='center')

show()
```

but what's useful to draw?

```
from numpy import linspace, pi, sin, cos
from matplotlib.pyplot import figure, show, plot

xs = linspace(0, 2 * pi)

fig = figure()
ax = fig.add_axes([.3, .3, .5, .5])

ax.plot(xs, sin(xs))
ax.plot(xs, cos(xs))

ax.set_ylabel('this is my y label')
fig.supylabel('this my figure y label')

# print(fig, ax, sep='\n')
show()
```

Adding a layer of convenience

```
from numpy import linspace, pi, sin, cos
from matplotlib.pyplot import subplots, show

# fig, ax = subplots()

# fig, axes = subplots(nrows=1, ncols=2)
fig, axes = subplots(nrows=2, ncols=2)

xs = linspace(0, 2 * pi)

# print(axes)
# print(type(axes))

axes[1, 0].plot(xs, sin(xs))

# print(axes, type(axes), axes[0])
show()
```

```
from numpy import linspace, pi, sin, cos
from matplotlib.pyplot import subplot_mosaic, show, subplots

xs = linspace(0, 2 * pi)

fig, axes = subplots(2, 2)

mosaic = [
    ['upper left', 'upper right'],
    ['lower left', 'lower right']
]
fig, axd = subplot_mosaic(mosaic)

# show()

print(
    axes,
    # axes[0, 1],
    axd,
    # axd['upper left']
    sep='\n{}\n'.format('\N{box drawings light horizontal}' * 40),
)
```

simple plots

```
from matplotlib.pyplot import subplots, show
from scipy.stats import uniform, norm, linregress

population = norm(loc=100, scale=3)

xs = population.rvs(size=200, random_state=0)
ys = 3 * xs + 10 + norm.rvs(scale=15, size=xs.size)

fig, ax = subplots()
# ax = axes[0]

ax.scatter(xs, ys)

regression = linregress(xs, ys)
ax.plot(xs, regression.slope * xs + regression.intercept)
ax.vlines(xs, ys, regression.slope * xs + regression.intercept, color='gainsboro',

show()
```

layouts as supplemental information

```
from matplotlib.pyplot import subplot_mosaic, show
from scipy.stats import uniform, norm, linregress
from numpy.random import default_rng

rng = default_rng(0)

population = norm(loc=100, scale=3)

xs = population.rvs(size=200, random_state=rng)
ys = 3 * xs + 10 + norm.rvs(scale=15, size=xs.size, random_state=rng)

regression = linregress(xs, ys)
mosaic = [
    ['top',    '.'],
    ['main', 'right']
]

gridspec_kw = {}
gridspec_kw={'wspace': 0, 'hspace': 0, 'width_ratios': [2, 1], 'height_ratios': [1
fig, axd = subplot_mosaic(mosaic, gridspec_kw=gridspec_kw)

axd['main'].scatter(xs, ys)
axd['main'].plot(xs, regression.slope * xs + regression.intercept)

axd['top'].hist(xs, ec='white')
axd['right'].hist(ys, orientation='horizontal', ec='white')

for ax in (axd['top'], axd['right']):
    ax.set_axis_off()

show()
```

parts of an Axes: Axis

```
from matplotlib.pyplot import show, subplots
from matplotlib.patches import Ellipse
from pandas import Series, date_range
from numpy.random import default_rng

rng = default_rng(0)

data = Series(
    5 + (noise := rng.uniform(-1, 1, size=100)).cumsum(),
    index=date_range('2020-01-01', periods=noise.size, freq='7D')
)

fig, ax = subplots()
data.plot(ax=ax)
```

```
fig.add_artist(
    Ellipse(
        xy=(.5, .0), width=1, height=0.1, transform=ax.transAxes,
        ec='red', fill=False
    )
)

fig.add_artist(
    Ellipse(
        xy=(.0, .5), width=.1, height=1, transform=ax.transAxes,
        ec='red', fill=False
    )
)

ax.xaxis
ax.yaxis

show()
```

```
from matplotlib.pyplot import show, subplots
from matplotlib.dates import MonthLocator, DateFormatter, YearLocator
from matplotlib.patches import Ellipse
from pandas import Series, date_range
from numpy.random import default_rng

rng = default_rng(0)

data = Series(
    5 + (noise := rng.uniform(-1, 1, size=100)).cumsum(),
    index=date_range('2020-01-01', periods=noise.size, freq='7D')
)

fig, ax = subplots()
ax.plot(data.index, data)

# ax.xaxis.set_tick_params(which='both', width=10)
# ax.xaxis.set_tick_params(which='major', length=20)
# ax.xaxis.set_tick_params(which='minor', length=10, color='tab:green')

# ax.xaxis.set_major_locator(MonthLocator([1,7]))
# ax.xaxis.set_minor_locator(MonthLocator())
# ax.xaxis.set_major_formatter(DateFormatter('%b\n%Y'))

# ax.get_xticks()
# ax.set_xticklabels(ax.get_xticklabels(), ...)

ax.xaxis.set_major_locator(YearLocator())
ax.xaxis.set_major_formatter(DateFormatter('%Y'))
ax.xaxis.set_minor_locator(MonthLocator([3, 6, 9]))
ax.xaxis.set_minor_formatter(DateFormatter('%b'))
ax.xaxis.set_tick_params(which='major', length=30)
ax.xaxis.set_tick_params(which='minor', length=10)

show()
```

axis values are always backed by a number

```
from pandas import date_range, Series, cut
from numpy import arange
from numpy.random import default_rng
from matplotlib.pyplot import subplots, show, setp

rng = default_rng(0)

xs = arange(100)
# xs = date_range('2020-01-01', periods=100)
ys = rng.normal(0, 1, size=len(xs)).cumsum()

data = Series(ys, index=xs)
data = data.groupby(cut(data, 3, labels=['early', 'mid', 'late'])).mean()

fig, ax = subplots()

# ax.plot(data.index, data)
ax.bar(data.index, data)
# setp(ax.get_xticklabels(), rotation=25, ha='right')

# dates → numbers
# print(ax.xaxis.convert_units(['2020-03-01']))

# numbers → numbers
# print(ax.xaxis.convert_units([50]))

# strings → numbers
print(ax.xaxis.convert_units(['even later']))

show()
```

parts of an Axes: Spines

```
from pandas import date_range, Series, cut
from numpy import arange
from numpy.random import default_rng
from matplotlib.pyplot import subplots, show, setp

rng = default_rng(0)

xs = arange(100)
xs = date_range('2020-01-01', periods=100)
ys = rng.normal(0, 1, size=len(xs)).cumsum()
```

```
data = Series(ys, index=xs)
data = data.groupby(cut(data, 3, labels=['early', 'mid', 'late'])).mean()

fig, ax = subplots()

ax.bar(data.index, data)
setp(ax.get_xticklabels(), rotation=25, ha='right')

ax.spines[['left', 'top', 'right']].set_visible(False)
# ax.spines['bottom'].set_position('zero')
ax.spines['bottom'].set_position(('data', 2))

show()
```

confusing conveniences

implicit global state API

```
from matplotlib.pyplot import plot, show

from numpy import sin, cos, linspace, pi

xs = linspace(0, 4*pi)

plot(xs, sin(xs))
plot(xs, cos(xs))

show()
```

```
from matplotlib.pyplot import show, subplots
from numpy import sin, cos, linspace, pi

def overlay_mean(ax, xs, ys):
    ax.scatter(xs.mean(), ys.mean(), s=40)

xs = linspace(0, 4*pi, 200)

fig, ax = subplots()
ax.plot(xs, sin(xs))
overlay_mean(ax, xs, sin(xs))

show()
```

pandas

```
from matplotlib.pyplot import subplots, show, legend, title
from numpy import sin, cos, linspace, pi
from pandas import DataFrame

df = DataFrame(
    index= (xs := linspace(0, 4*pi, 200)),
    data={
        'sin': sin(xs),
        'cos': cos(xs),
    }
)

# df.plot()
fig, axes = subplots(2, 1)
print(axes)

for ax, col in zip(axes.flat, df.columns):
    df[col].plot(ax=ax)
    ax.set_title(col)

# df['sin'].plot(ax=ax1)
# ax1.set_title('sin function')

# df['cos'].plot(ax=ax2)
# ax2.set_title('cos function')

# df.plot(subplots=True, legend=False)
# legend()
# title('This is my sin wave')
# title('This is my cos wave')

# print(dir(axes[0]))

# show()
```

Exploratory Visualizations

```
print("Let's Get Started!")
```

pair grids

```
from pandas import DataFrame
from numpy.random import default_rng
from seaborn import pairplot

rng = default_rng(0)

df = DataFrame({
    'v1': rng.normal(70, 3, size=(sz := 100)),
    'v2': rng.uniform(40, 50, size=sz),
    'group': rng.choice(['A', 'B'], size=sz)
}).set_index('group').assign(v3=lambda d: d['v1'] * 3 + 10 + rng.normal(scale=10,

df.loc['B', 'v2'] += 5

from matplotlib.pyplot import rc, subplots, rcdefaults, show
from matplotlib.transforms import blended_transform_factory
from itertools import product
from numpy import histogram_bin_edges

rc('font', size=14)
rc('axes.spines', top=False, right=False)

fig, axes = subplots(
    df.columns.size, df.columns.size,
    sharex='col', sharey='row', dpi=144
)

for label, ax in zip(df.columns, axes[:, 0]):
    ax.set_ylabel(label)

for label, ax in zip(df.columns, axes[-1, :]):
    ax.set_xlabel(label)

colors = {'A': 'tab:blue', 'B': 'tab:orange'}
for group, gdf in df.groupby('group'):
    for i, j in product(range(df.columns.size), repeat=2):
        x, y = gdf.iloc[:, j], gdf.iloc[:, i]
        ax = axes[i, j]
        if i == j:
            ax = ax.twinx()
            ax.yaxis.set_visible(False)
            bins = histogram_bin_edges(df.iloc[:, j], bins='auto')
            ax.hist(x, bins=bins, ec='black', alpha=.6, density=True, label=group,
        else:
            ax.scatter(x, y, s=20, ec='white', lw=.2, label=group, fc=colors[group

        ax.margins(.2)

fig.align_ylabels()

show()
```

correlation matrix

```
from matplotlib.pyplot import subplots, show
from matplotlib.ticker import MultipleLocator
from numpy.random import default_rng
from pandas import DataFrame

rng = default_rng(0)
```

```
data = DataFrame(data=rng.uniform(-1, 1, size=(5, 6)), index=[*'ABCDE'])

fig, ax = subplots()
img = ax.pcolormesh(
    data.columns, data.index, data,
    shading='nearest', cmap='RdBu'
)
ax.set_title('Colormesh', fontsize='x-large')
fig.colorbar(img, ax=ax)

show()
```

correlelograms

```
from matplotlib.pyplot import subplots, show
from matplotlib.ticker import MultipleLocator
from matplotlib.colors import Normalize
from matplotlib.collections import PatchCollection
from matplotlib.patches import Circle
from numpy.random import default_rng
from pandas import DataFrame

rng = default_rng(0)
data = DataFrame(data=rng.uniform(-1, 1, size=(5, 6)), index=[*'ABCDE'])

fig, ax = subplots()

scatter_data = data.stack()
y, x = scatter_data.index.codes
norm = Normalize(-1, 1)
sizes = norm(abs(scatter_data))

collection = PatchCollection(
    [Circle((xi, yi), radius=si / 3) for xi, yi, si in zip(x, y, sizes)],
    cmap='RdBu', norm=norm, edgecolors='k', zorder=2
)
collection.set_array(scatter_data)
ax.add_collection(collection)

fig.colorbar(collection, ax=ax)

ax.grid(which='major')
ax.set_yticks([0, 1, 2, 3, 4], labels=data.index)

ax.set(
    title='Redundant Signal → Stronger Communication',
    xlim=(0-.5, 5+.5),
    ylim=(0-.5, 4+.5),
)
show()
```

small multiples

```
from pandas import read_csv, to_datetime
from matplotlib.pyplot import show, subplots
from matplotlib.pyplot import setp

trades = (
    read_csv('data/trades.Alice.NYC.csv')
    .assign(
        date=lambda df: to_datetime(df['date'], format='%Y-%m-%d %H:%M:%S%z', utc=
        revenue=lambda df: -df['volume'] * df['price'],
        cumul_revenue=lambda df: df.groupby('ticker')['revenue'].cumsum()
    )
    .sort_values(['ticker', 'date'])
)
# print(trades)

fig, ax = subplots()

for t, group in trades.groupby('ticker'):
    ax.plot(group.index, group['cumul_revenue'], label=t)
    ax.xaxis.set_tick_params(rotation=25)

ax.legend()

fig.suptitle('Cumulative Revenue By Ticker')

show()
```

```
from pandas import read_csv, to_datetime
from matplotlib.pyplot import show, subplot_mosaic
from matplotlib.pyplot import setp

trades = (
    read_csv('data/trades.Alice.NYC.csv')
    .assign(
        date=lambda df: to_datetime(df['date'], format='%Y-%m-%d %H:%M:%S%z', utc=
        revenue=lambda df: -df['volume'] * df['price'],
        cumul_revenue=lambda df: df.groupby('ticker')['revenue'].cumsum()
    )
    .sort_values(['ticker', 'date'])
)

tickers = [
    ['chmk', 'cwao'],
    ['evqx', 'hvra'],
    ['ibba', 'kwoa'],
    ['npzs', 'qooy'],
    ['tswe', 'wqnh'],
]
fig, axd = subplot_mosaic(
    tickers, sharex=True, sharey=True,
    gridspec_kw={'right': .9, 'hspace': .4}
)

for t, group in trades.groupby('ticker'):
    ax = axd[t]
    ax.plot(group.index, group['cumul_revenue'])

    rest_of_trades = (
        trades.loc[trades['ticker'] != t, 'cumul_revenue']
    )
    ax.plot(rest_of_trades.index, rest_of_trades, color='gainsboro', zorder=-1)

    setp(ax.get_xticklabels(), ha='right', rotation=25)
    ax.set_title(t)

fig.suptitle('Cumulative Revenue By Ticker')

show()
```

interactive visualizations

```
from matplotlib.pyplot import figure, show
from numpy.random import default_rng
from numpy import where, array
from matplotlib.widgets import LassoSelector
from matplotlib.path import Path
from scipy.stats import linregress
from string import ascii_lowercase

rng = default_rng(0)
xs = rng.random(size=100)
xs[[0, 1, 2]] += 1
ys = rng.random(size=xs.size)
names = rng.choice([*ascii_lowercase], size=(xs.size, 4)).view('<U4').ravel()

fig = figure()
gspec = fig.add_gridspec(3, 5, wspace=.5)
scatter_ax = fig.add_subplot(gspec[:, :3])
bar_ax = fig.add_subplot(gspec[0, 3])
table_ax = fig.add_subplot(gspec[1:, 3:])
table_ax.axis('off')

from pandas.plotting import table
from pandas import DataFrame
```

```
df = DataFrame({'name': names, 'x': xs, 'y': ys})
plot_df = df.assign(x=df['x'].round(2), y=df['y'].round(2))

t = table(table_ax, plot_df.head(10), loc='center')

scatter = scatter_ax.scatter(xs, ys)
reg_line, = scatter_ax.plot([], [])
barcollec = bar_ax.bar(['xs', 'ys'], [xs.mean(), ys.mean()])
bar_ax.set(ylim=(0, 1))

def onselect(verts):
    global reg_line
    global t

    path = Path(verts)
    indices = path.contains_points(scatter.get_offsets())
    if (indices == False).all():
        return

    for b, data in zip(barcollec, (xs, ys)):
        b.set_height(data[indices].mean())

    # scatter.set_alpha(where(indices, 1, .3))

    t.remove()
    t = table(table_ax, plot_df.iloc[indices].head(10), loc='center')

    lr = linregress(xs[indices], ys[indices])
    pred_xs = array([xs.min(), xs.max()])
    pred_ys = lr.slope * pred_xs + lr.intercept
    reg_line.set_data(pred_xs, pred_ys)

    fig.canvas.draw()

lasso = LassoSelector(scatter_ax, onselect=onselect)
show()
```

```
from matplotlib.pyplot import plot, show, rc, setp

from pandas import DataFrame
from numpy.random import default_rng
from scipy.stats import linregress

rng = default_rng(0)

df = DataFrame({
    'temperature': rng.normal(70, 3, size=(sz := 100)),
    'requests': rng.integers(40, 50, size=sz),
}).assign(
    power=lambda d: (d['temperature'] + rng.normal(scale=10, size=sz)),
    usage=lambda d: (d['power'] + rng.normal(scale=10, size=sz))
)

from matplotlib.pyplot import subplots, rc, rcdefaults
from numpy import unravel_index, stack, stack, linspace

rc('axes.spines', right=False, top=False)
rc('font', size=14)

fig, (heat_ax, rel_ax) = subplots(
    1, 2, figsize=(12, 5),
    gridspec_kw={'wspace': .3, 'top': .7, 'bottom': .2}
)

corr = df.corr()
im = heat_ax.pcolor(df.columns, df.columns, corr, vmax=1, vmin=-1, cmap='RdBu')
heat_ax.set_aspect('equal', anchor='SE')
heat_ax.invert_yaxis()
cbar = fig.colorbar(im, ax=heat_ax, orientation='horizontal', aspect=10)

x = df['power']
y = df['requests']
scatter = rel_ax.scatter(x, y)

reg = linregress(x, y)
grid = linspace(x.min(), x.max(), 200)
pred_y = grid * reg.slope + reg.intercept

line, = rel_ax.plot(grid, pred_y)

def onclick(event):
    in_bounds, poly_dict = im.contains(event)
    idx = poly_dict['ind']

    y_idx, x_idx = (unravel_index(idx, corr.shape))
    if len(y_idx) == 0 or len(x_idx) == 0:
        return

    offsets = df.iloc[:, [x_idx[0], y_idx[0]]].to_numpy()
    scatter.set_offsets(offsets)

    xlims = offsets[:, 0].min(), offsets[:, 0].max()
    ylims = offsets[:, 1].min(), offsets[:, 1].max()
    reg = linregress(offsets[:, 0], offsets[:, 1])
    grid = linspace(*xlims, 200)
    pred_y = grid * reg.slope + reg.intercept
    line.set_data(grid, pred_y)

    rel_ax.update_datalim(offsets)

    rel_ax.relim()
    rel_ax.autoscale_view()
    fig.canvas.draw_idle()
fig.canvas.mpl_connect('button_press_event', onclick)
setp(heat_ax.get_xticklabels(), rotation=15, ha='right', rotation_mode='anchor')
fig.suptitle('Click on a square in the heat map to update the scatter & regression

rel_ax.set(xlabel='Requests', ylabel='Power')

show()
```

Communicating More With Visualizations

```
print("Let's Get Started!")
```

what are the mediums for communication?

```
from matplotlib.pyplot import subplots, show
from matplotlib.cm import get_cmap, ScalarMappable
from matplotlib.colors import Normalize
from numpy.random import default_rng
from pandas import DataFrame

rng = default_rng(0)

df = DataFrame({
    'A': rng.normal(70, 3, size=(sz := 100)),
    'B': rng.normal(65, 3, size=sz),
    'C': rng.normal(73, 3, size=sz),
    'D': rng.normal(80, 3, size=sz),
    'E': rng.normal(70, 3, size=(sz := 100)),
    'F': rng.normal(65, 3, size=sz),
    'G': rng.normal(73, 3, size=sz),
    'H': rng.normal(80, 3, size=sz),
})

# print(df)
# df.plot()

plot_data = df.mean().sort_values(ascending=False)
# print(plot_data)
# print(plot_data.sort_values(ascending=False))

fig, axes = subplots(2, 2)

plot_data.plot.bar(ax=axes[0, 0])

cmap = get_cmap('Blues')
norm = Normalize()
mappable = ScalarMappable(norm, cmap)
```

```
axes[0, 1].bar(plot_data.index, [1] * len(plot_data), color=cmap(norm(plot_data)))
axes[0, 1].set_facecolor('gray')
fig.colorbar(mappable, ax=axes[0, 1])

axes[1, 0].pie(plot_data, labels=plot_data.index)

from pandas.plotting import table
table(ax=axes[1,1], data=plot_data.rename('avg').round(2), loc='center', colWidths
axes[1, 1].set_axis_off()

show()
```

```
from matplotlib.pyplot import subplots, rc, show
from pandas import DataFrame
from numpy.random import default_rng

from calendar import month_name

months = month_name[1:]

df = DataFrame({
    'Product A': [209, 192, 137, 108, 98, 96, 104, 120, 154, 191, 187, 215],
    'Product B': [184, 161, 123, 96, 88, 84, 97, 109, 124, 163, 156, 175]},
    index=months
)

rc('ytick', left=False)
rc('axes.spines', left=False, right=False, top=False)

fig, (ax1, ax2) = subplots(1, 2, sharey='row', gridspec_kw={'wspace': .01})

bc = ax1.barh(df.index, df['Product A'], color='tab:blue', label='Product A')
ax1.bar_label(bc, label_type='center')

bc = ax2.barh(df.index, df['Product B'], color='tab:red', label='Product B')
ax2.bar_label(bc, label_type='center')

xmax = max([*ax1.get_xlim(), *ax2.get_xlim()])
ax1.set_xlim(right=xmax)
ax2.set_xlim(right=xmax)
ax1.invert_xaxis()

def not_zero(val, pos):
    if val == 0:
        return ''
    return f'{val:.0f}'

for ax in (ax1, ax2):
    ax.xaxis.set_major_formatter(not_zero)

fig.legend(loc='lower center', frameon=False, ncols=2)
```

```
show()
```

```
from seaborn import barplot

from matplotlib.pyplot import subplots, show
from pandas import DataFrame
from numpy.random import default_rng

from calendar import month_name

months = month_name[1:]

df = DataFrame({
    'Product A': [209, 192, 137, 108, 98, 96, 104, 120, 154, 191, 187, 215],
    'Product B': [184, 161, 123, 96, 88, 84, 97, 109, 124, 163, 156, 175]},
    index=months
)

fig, ax = subplots()

barplot(
    df.rename_axis(columns='product', index='month').stack().rename('value').reset
    y='month', x='value', hue='product', orient='horizontal', ax=ax
)

show()
```

```
from matplotlib.pyplot import subplots, rc, show
from pandas import DataFrame
from numpy.random import default_rng

from calendar import month_name

months = month_name[1:]

df = DataFrame({
    'Product A': [209, 192, 137, 108, 98, 96, 104, 120, 154, 191, 187, 215],
    'Product B': [184, 161, 123, 96, 88, 84, 97, 109, 124, 163, 156, 175]},
    index=months
)

rc('ytick', left=False)
rc('axes.spines', left=False, right=False, top=False)

fig, ax = subplots()

ax.barh(df.index, df['Product B'] - df['Product A'])
```

```
max_abs_x = max(abs(v) for v in ax.get_xlim())
ax.set_xlim(-max_abs_x, max_abs_x)
ax.set_xlabel('Difference in Product B vs Product A\n(negative values indidicate h

show()
```

expressing uncertainty

```
from matplotlib.pyplot import subplots, show
from numpy.random import default_rng
from scipy.stats import norm, uniform, triang, cauchy
from pandas import DataFrame

rng = default_rng(0)
df = DataFrame({
    'unif': uniform.rvs(60, 100, size=(sz := 200), random_state=rng),
    'norm': norm.rvs(80, 5, size=sz, random_state=rng),
    'triang': triang.rvs(c=.6, loc=50, scale=30, size=sz, random_state=rng),
})

fig, ax = subplots()

# ax.bar(df.columns, df.mean(), alpha=.5)
# ax.bar(df.columns, df.median(), alpha=.5)

# ax.bar(df.columns, df.mean())
# ax.errorbar(df.columns, df.mean(), df.std(), color='k', zorder=9, ls='none', lw=

# ax.scatter(df.columns, df.mean(), color='k', s=100)
# ax.vlines(df.columns, df.mean() - df.std(), df.mean() + df.std(), color='k', lw=

# ax.violinplot(df)

show()
```

annotations that draw attention

highlighting areas

```
from pandas import read_csv, to_datetime
from matplotlib.pyplot import show, subplots
from matplotlib.dates import MonthLocator

trades = (
    read_csv('data/trades.Alice.NYC.csv')
    .assign(
        date=lambda df: to_datetime(df['date'], format='%Y-%m-%d %H:%M:%S%z', utc=
        revenue=lambda df: -df['volume'] * df['price'],
    )
    .sort_values(['ticker', 'date'])
    .groupby('ticker', as_index=False).rolling('3D', on='date').mean()
    .set_index(['ticker', 'date'])
)

# print(trades)

# fig, ax = subplots()
gridspec_kw={'height_ratios': [3, 1]}
fig, (ax1, ax2) = subplots(2, 1, gridspec_kw=gridspec_kw)
ax = ax2

revenue = trades.loc['chmk', 'revenue'].cumsum()
ax.plot(revenue.index, revenue)

ax.xaxis.set_major_locator(MonthLocator([1, 4, 7, 10]))

def as_quarter(value, pos):
    d = to_datetime(value, unit='D')
    return f'{d:%Y} Q{d.quarter}'
ax.xaxis.set_major_formatter(as_quarter)
ax.xaxis.grid()
ax.margins(x=0)

ax.axvspan('2020-04-01', '2020-07-01', 0, 1, alpha=.4, color='yellow')

ax1.plot(revenue.loc['2020-04-01':'2020-07-01'])
ax1.xaxis.set_major_locator(MonthLocator())
ax1.set_facecolor('lightyellow')
ax1.set_alpha(.4)

show()
```

highlighting outliers

```
from pandas import read_csv, to_datetime
from matplotlib.pyplot import show, subplots
from matplotlib.pyplot import setp

trades = (
    read_csv('data/trades.Alice.NYC.csv')
    .assign(
        date=lambda df: to_datetime(df['date'], format='%Y-%m-%d %H:%M:%S%z', utc=
        revenue=lambda df: -df['volume'] * df['price'],
    )
    .sort_values(['ticker', 'date'])
    .groupby('ticker', as_index=False).rolling('3D', on='date').mean()
    .set_index(['ticker', 'date'])
)
```

```
revenue = trades.loc['chmk', 'revenue'].cumsum()

fig, ax = subplots(gridspec_kw={'top': .9, 'right': .9})

# revenue.plot(ax=ax)

smooth_trades = revenue.rolling('3D').agg(['mean', 'std'])
ax.plot(smooth_trades.index, smooth_trades['mean'])
bottom, top = smooth_trades.eval('mean - (std * 2)'), smooth_trades.eval('mean + (
ax.fill_between(smooth_trades.index, bottom, top, alpha=.2)

outlier_mask = (revenue < bottom) | (revenue > top)
outliers = revenue.loc[outlier_mask]
ax.scatter(outliers.index, outliers, color='tab:red', s=6)
ax.set_title('Cumulative Revenue w/ Outlier Trades')

show()
```

annotations that clarify

```
from matplotlib.pyplot import subplots, show, rc
from numpy import linspace, pi, sin, cos
from pandas import DataFrame

df = DataFrame(
    index= (xs := linspace(0, 4*pi, 200)),
    data={
        'sin': sin(xs),
        'cos': cos(xs),
    }
)

rc('axes.spines', right=False, top=False)

fig, ax = subplots(gridspec_kw={'right': .8})
for col in df.columns:
    l, = ax.plot(df.index, df[col], label=col)
    max_loc = df.index.max()
    ax.text(
        max_loc, df.loc[max_loc, col], s=col,
        va='center', ha='left', color=l.get_color()
    )

# ax.legend()

show()
```

overplotting

```
from matplotlib.cm import get_cmap
from matplotlib.pyplot import subplots, show
from numpy.random import default_rng

rng = default_rng(0)
xs = rng.normal(80, 5, size=10_000)
ys =  5 * xs + rng.normal(0, 70, size=len(xs))

fig, ax = subplots()

# ax.scatter(xs, ys,s=10)
# ax.scatter(xs, ys, alpha=.2)

im = ax.hist2d(xs, ys, bins=50, cmap='Blues', cmin=1)
fig.colorbar(im[-1], ax=ax)
ax.set_facecolor('gainsboro')

show()
```