Intro. Comp. for Data Science (FMI08)

Dr. Nono Saha

June 7, 2023

Max Planck Institute for Mathematics in the Sciences
University of Leipzig/ScaDS.AI

Spring 2023

Course plan

1. Plotting with `pandas`
2. Introduction to `seaborn`

Plotting with pandas

pandas: plotting methods

Both Series and DataFrame objects have a plot method which can be used to create visualizations - `dtypes` determine the type of plot produced. Note these are just `pyplot` plots and can be formatted as such.

```
s = pd.Series(np.cumsum( np.random.normal(size=100) ),
index = pd.date_range("2022-01-01", periods=100, freq="D"))
plt.figure(figsize=(3,3), layout="constrained")
s.plot()
plt.show()
```

DataFrame plotting

```
df = pd.DataFrame( np.cumsum( np.random.normal(size=(100,4)),
    axis=0),
index = pd.date_range("2022-01-01", periods=100, freq="D"),
columns = list("ABCD"))

plt.figure(layout="constrained")
df.plot(figsize=(5,3))
plt.show()
```

pandas and subplots

```
plt.figure(figsize=(5,3))
plt.subplot(211)

df[["x1"]].plot.hist(bins=15, figsize=(5,3))
plt.subplot(212)

df[["x2"]].plot.hist(bins=15, figsize=(5,3))
plt.show()
```

```
plt.subplot(212)
df[["x2"]].plot.hist(bins=15, figsize=(5,3))
plt.show()

fig, (ax1, ax2) = plt.subplots(2,1, figsize=(5,5))
df[["x1"]].plot.hist(ax = ax1, bins=15)
df[["x2"]].plot.hist(ax = ax2, bins=15)

plt.show()
```

pandas: more plotting

```
# Using by
df_wide.plot.hist(bins=15, by="variable", legend=False, figsize
    =(5,5))
plt.show()
```

Note the **by** argument is not common to most of the other plotting functions - only **box** also has it.

- higher level plots - pair plot

The **pandas** library also provides the **plotting** submodule with several useful higher-level plots.

```
cov = np.identity(5)
cov[1,2] = cov[2,1] = 0.5
cov[3,0] = cov[0,3] = -0.8

df = pd.DataFrame( np.random.multivariate_normal(mean=[0]*5, cov
    =cov, size=1000), columns = ["x1","x2","x3","x4","x5"] )

pd.plotting.scatter_matrix(df, alpha=0.2, diagonal="kde")
```

seaborn

What is seaborn?

> **seaborn** is a library for making statistical graphics in Python. It builds on top of **matplotlib** and integrates closely with pandas data structures.

> **seaborn** helps you explore and understand your data. Its plotting functions operate on Dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean rather than on the details of how to draw them.

```
1 import matplotlib.pyplot as plt
2 import seaborn as sns
3
4 penguins = sns.load_dataset("penguins")
5 penguins
```

seaborn: basic plots

Exampe 1

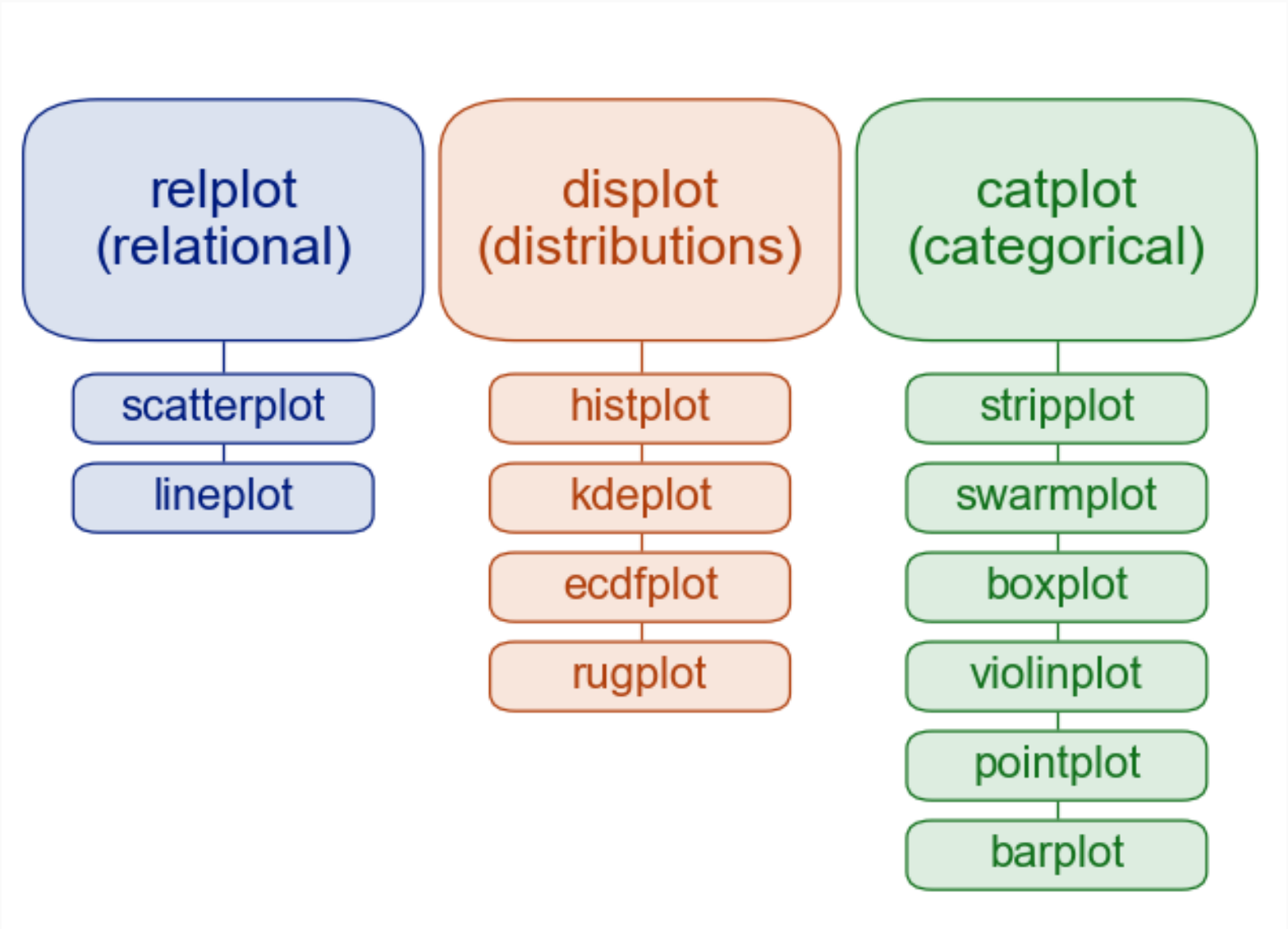
```
sns.relplot(
data=penguins,
x="bill_length_mm",
y="bill_depth_mm"
)

```

```
sns.relplot(
data=penguins,
x="bill_length_mm",
y="bill_depth_mm",
hue="species")

```

Example 2

```
sns.relplot(
data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species",
col="island", row="species")

```

seaborn: figure-level vs. axes-level functions



These are not the only axes-level functions. Please, do check the doc.

seaborn: figure-level plot examples

displots

```
sns.displot(
data = penguins,
x = "bill_length_mm", hue = "
    species",
alpha = 0.5,
aspect = 1.5)
```

```
sns.displot(
data = penguins,
x = "bill_length_mm", hue = "
    species",
kind = "kde", fill=True,
alpha = 0.5, aspect = 1)
```

catplots

```
sns.catplot(
data = penguins,
x = "species",
y = "bill_length_mm",
hue = "sex"
)
```

```
sns.catplot(
data = penguins,
x = "species",
y = "bill_length_mm",
hue = "sex",
kind = "box")
```

seaborn: figure-level plot size

To adjust the size of plots generated via a figure-level plotting function, update the aspect and height arguments, figure width is `aspect * height`.

Examples

```
sns.relplot(
data=penguins,
x="bill_length_mm", y="
    bill_depth_mm",
hue="species",
aspect = 1, height = 3
)

```

```
sns.relplot(
data=penguins,
x="bill_length_mm", y="
    bill_depth_mm",
hue="species",
aspect = 1, height = 5
)
```

Note this is the size of a facet (Axes) not the figure

seaborn: figure-level plot details

Examples

```
g = sns.relplot(
data=penguins,
x="bill_length_mm", y="
    bill_depth_mm",
hue="species",
aspect = 1)

print(g)
## <seaborn.axisgrid.FacetGrid
    object at 0x294d757f0>

```

```
h = sns.relplot(
data=penguins,
x="bill_length_mm", y="
    bill_depth_mm",
hue="species", col="island",
aspect = 1/2)

print(h)
## <seaborn.axisgrid.FacetGrid
    object at 0x28d4474f0

```

Then, check the official documentation to see all the possible methods implemented by the class `FaceGrid`.

seaborn: adjusting the labels

```
sns.relplot(data=penguins,
x="bill_length_mm",
y="bill_depth_mm",
hue="species",
aspect = 1)
.set_axis_labels(
"Bill Length (mm)", "Bill Depth
    (mm)"
)
```

```
sns.relplot(data=penguins,
x="bill_length_mm",
y="bill_depth_mm",
hue="species", col="island",
aspect = 1/2).set_axis_labels(
"Bill Length (mm)", "Bill Depth
    (mm)"
).set_titles("{col_var} - {
    col_name}" )
```

Using axes to modify plots

```
g = sns.relplot(data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species",aspect = 1)

g.ax.axvline(
x = penguins.bill_length_mm.mean(), c="k")
```

`seaborn`: why figure-level functions?

Advantages:

- Easy faceting by data variables
- Legend outside of plot by default
- Easy figure-level customization
- Different figure size parameterization

Disadvantage:

- Many parameters are not in the function signature
- Cannot be part of a larger `matplotlib` figure
- Different API from `matplotlib`
- Different figure size parameterization

Details based on `seaborn` doc

seaborn: lmplots and axes-leve functions

- lmplots

It is a convenient interface for fitting and plotting regression models across subsets of data,

```
sns.lmplot(
data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species", col="island",
aspect = 1, truncate=False)
```

- axes-level functions

They return a `matplotlib.pyplot.Axes` object instead of a `FacetGrid`.

```
plt.figure()
sns.scatterplot(data=penguins, x="bill_length_mm", y="
    bill_depth_mm", hue="species")
plt.xlabel("Bill Length (mm)")
plt.ylabel("Bill Depth (mm)")
plt.title("Length vs. Depth")
```

seaborn: subplots - pyplot style

```
plt.figure(figsize=(4,6), layout="constrained")

plt.subplot(211)

sns.scatterplot(data=penguins, x="bill_length_mm", y="
    bill_depth_mm", hue="species")

plt.subplot(212)

sns.countplot( data=penguins, x="species")

plt.show()

```

seaborn: subplots - OO style

```
fig, axs = plt.subplots( 2, 1, figsize=(4,6),  layout="
    constrained", sharex=True )

sns.scatterplot( data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species", ax = axs[0] )

sns.kdeplot( data=penguins,
x="bill_length_mm", hue="species",
fill=True, alpha=0.5,
ax = axs[1] )

plt.show()

```

seaborn: layering plots

```
plt.figure(layout="constrained")

sns.kdeplot( data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species" )

sns.scatterplot( data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species", alpha=0.5)

sns.rugplot( data=penguins,
x="bill_length_mm", y="bill_depth_mm",
hue="species")

plt.legend()

plt.show()

```

seaborn: themes

seaborn comes with a number of themes (darkgrid, whitegrid, dark, white, and ticks) which can be enabled by sns.set_theme() at the figure level or sns.axes_style() at the axes level.

Examples

```
def sinplot():
  x = np.linspace(0, 14, 100)
  for i in range(1, 7):
    plt.plot(x, np.sin(x + i
    * .5) * (7 - i))

sinplot()
plt.show()

```

```
with sns.axes_style("darkgrid"):
  sinplot()
  plt.show()

with sns.axes_style("whitegrid"):
  sinplot()
  plt.show()

with sns.axes_style("dark"):
  sinplot()
  plt.show()
```

seaborn: context

With seaborn, we can also set a context for the figure we want to create.

Examples

```
def sinplot():
  x = np.linspace(0, 14, 100)
  for i in range(1, 7):
    plt.plot(x, np.sin(x + i
    * .5) * (7 - i))

sns.set_context("notebook")
sinplot()
plt.show()

```

```
sns.set_context("paper")
sinplot()
plt.show()

with sns.set_context("talk"):
  sinplot()
  plt.show()

with sns.set_context("poster"):
  sinplot()
  plt.show()
```

seaborn: pair plots

```
sns.pairplot(data = penguins, height=5)

sns.pairplot(data = penguins, hue="species", height=5, corner=
    True)

```

`pairplot()` is a special case of the more general `PairGrid` - once constructed, there are methods that allow for mapping plot functions of the different axes,

```
sns.PairGrid(penguins, hue="species", height=5)

## Mapping
g = sns.PairGrid( penguins, hue="species", height=3 )

g = g.map_diag( sns.histplot, alpha=0.5)

g = g.map_lower( sns.scatterplot )

g = g.map_upper(sns.kdeplot)

```

seaborn: Pair subsets

```
x_vars = ["body_mass_g", "bill_length_mm", "bill_depth_mm", "
    flipper_length_mm"]
y_vars = ["body_mass_g"]

g = sns.PairGrid(penguins, hue="species", x_vars=x_vars, y_vars=
    y_vars, height=3)

g = g.map_diag(sns.kdeplot, fill=True)
g = g.map_offdiag(sns.scatterplot, size=penguins["body_mass_g"])
g = g.add_legend()

```

- Custom FacetGrids

Just like `PairGrids` it is possible to construct `FacetGrids` from scratch,

```
sns.FacetGrid(penguins, col="island", row="species")
g = sns.FacetGrid(penguins, col="island", hue="species")
g = g.map(sns.scatterplot, "bill_length_mm", "bill_depth_mm")
g = g.add_legend()

```

seaborn: custom plots / functions

```
from scipy import stats
def quantile_plot(x, **kwargs):
quantiles, xr = stats.probplot(x, fit=False)
plt.scatter(xr, quantiles, **kwargs)

g = sns.FacetGrid(penguins, col="species", height=3, sharex=
    False)
g.map(quantile_plot, "body_mass_g", s=2, alpha=0.5)

```

Example from axis grid tutorial

- `jointplot`

One final figure-level plot, is a joint plot which includes marginal distributions along the x and y-axis.

```
g = sns.jointplot(data=penguins, x="bill_length_mm", y="
    bill_depth_mm", hue="species")

```