Intro. Comp. for Data Science (FMI08)

Dr. Nono Saha

May 31, 2023

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

Spring 2023

Lesson 09 - More pandas and matplotlib

1. Reshaping data: exercice
2. More pandas: split-apply-combine
3. Introduction to matplotlib
4. Plotting with pandas

more pandas: exercice

Create a DataFrame from the data available at `../data/rent.csv` using `pd.read_csv()`. These data come from the 2017 American Community Survey and reflect the following values:

- `name` - name of state
- `variable` - Variable name: income = median yearly income, rent = median monthly rent
- `estimate` - Estimated value
- `moe` - 90% margin of error

Using these data, find the state(s) with the lowest income-to-rent ratio.

pandas: split-apply-combine

Group by: split-apply-combine

By ”group by”, we are referring to a process involving one or more of the following steps:

- **Splitting** the data into groups based on some criteria.
- **Applying** a function to each group independently.
- **Combining** the results into a data structure.

Groups can be created within a DataFrame via `groupby()` - these groups are then used by the standard summary methods (e.g. `sum()`, `mean()`, `std()`, etc…).

```
cereal = pd.read_csv("../data/cereal.csv")

cereal.groupby("type")
## <pandas.core.groupby.generic.DataFrameGroupBy object at 0x143e2a460>

cereal.groupby("type").mean()
cereal.groupby("mfr").size()

```

Group by: selecting and iterating groups

Groups can be accessed via `get_group()` or the DataFrameGroupBy can be iterated over,

```
cereal.groupby("type").get_group("Hot")

cereal.groupby("mfr").get_group("Post")

for name, group in cereal.groupby("type"):
  print(name)
  print(group)
  print("")

```

Group by: named aggregation

It is also possible to use special syntax to aggregate specific columns into a named output column,

```
cereal.groupby("mfr", as_index=False).agg(
min_cal = ("calories", "min"),
max_cal = ("calories", "max"),
med_sugar = ("sugars", "median"),
avg_rating = ("rating", "mean"))

##               mfr  min_cal  max_cal  med_sugar  avg_rating
## 0   General Mills      100      140        8.5   34.485852
## 1 Kellogg's 50      160        7.0   44.038462
## 2          Maltex      100      100        3.0   54.850917
## 3         Nabisco       70      100        0.0   67.968567
## 4            Post       90      120       10.0   41.705744
## 5     Quaker Oats       50      120        6.0   42.915990
## 6  Ralston Purina       90      150        5.5   41.542997
```

Tuples can also be passed using `pd.NamedAgg()` but this offers no additional functionality.

Group by: transformation

The `transform()` method returns a DataFrame with the aggregated result matching the size (or length 1) of the input group(s),

```
cereal.groupby("mfr").transform(np.mean)
# For the new version of pandas, this may not work

cereal.groupby("type").transform("mean")
# For the new version of pandas, this may not work

<string>:1: FutureWarning: Dropping invalid columns in DataFrameG...

```

Note that we have lost the non-numeric columns, in case it works. And there will be a warning message.

Group by: practical transformation

`transform()` will generally be most useful via a user-defined function. The lambda argument is for each column of each group.

```
( cereal.groupby("mfr").transform(
lambda x: (x - np.mean(x))/np.std(x)) )

```

Above, we are standardizing each numerical column of each manufacturer

Group by: filtering groups

`filter()` also respects groups and allows for the inclusion/exclusion of groups based on user-specified criteria,

```
cereal.groupby("mfr").size()

## mfr
## General Mills     22
## Kellogg's 23
## Maltex             1
## Nabisco            6
## Post               9
## Quaker Oats        8
## Ralston Purina     8
## dtype: int64

cereal.groupby("mfr").filter(lambda x: len(x) > 10)

(cereal.groupby("mfr").filter(lambda x: len(x) > 10)
.groupby("mfr").size())

```

matplotlib

matplotlib vs. pyplot

> matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

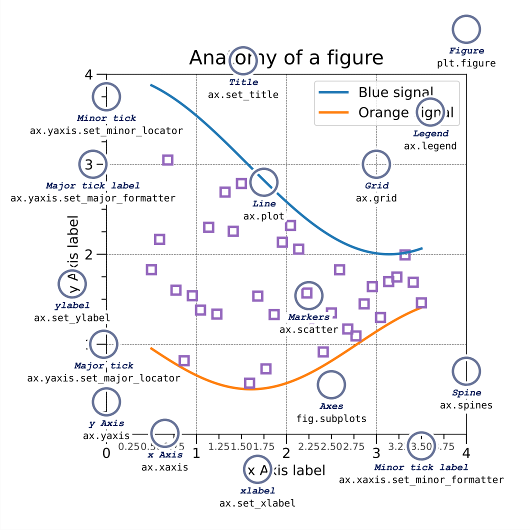
```python
import matplotlib as mpl
import matplotlib.pyplot as plt

```

Why do we usually import only pyplot then?

> matplotlib is the whole package; matplotlib.pyplot is a module in matplotlib; and pylab is a module that gets installed alongside matplotlib.
> pyplot provides the state-machine interface to the underlying object-oriented plotting library. The state-machine implicitly and automatically creates figures and axes to achieve the desired plot.

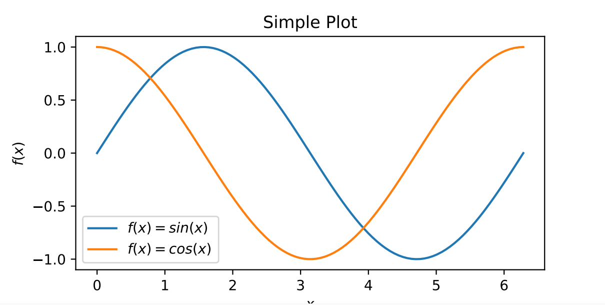
`matplotlib`: plot anatomy

- **Figure** - The entire plot (including subplots)
- **Axes** - Subplot attached to a figure, contains the region for plotting data and axis'
- **Axis** - Set the scale and limits, generate ticks and ticklabels
- **Artist** - Everything visible on a figure: text, lines, axis, axes, etc.



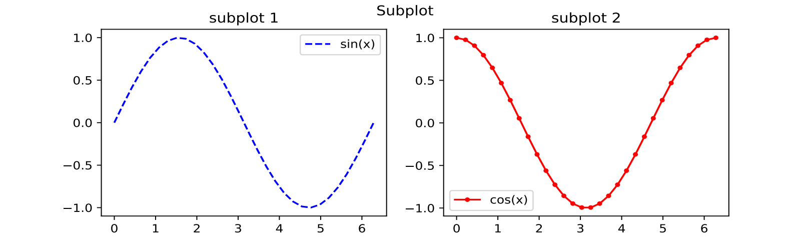
matplotlib: basic plot

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(x, y1, label="sin(x)")
ax.plot(x, y2, label="cos(x)")
ax.set_title("Simple Plot")
ax.legend()

```



matplotlib: subplot

```
x = np.linspace(0, 2*np.pi, 30)
y1 = np.sin(x)
y2 = np.cos(x)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 3))
ax1.plot(x, y1, "--b", label="sin(x)")
ax2.plot(x, y2, ".-r", label="cos(x)")
fig.suptitle("Subplot")
ax1.set_title("subplot 1")
ax2.set_title("subplot 2")
ax1.legend()
ax2.legend()

```



matplotlib: more subplots and fancy

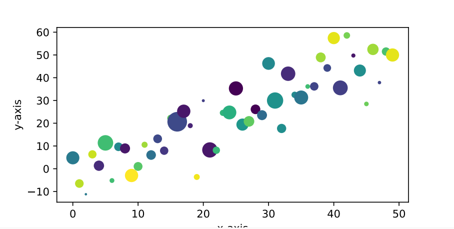
```python
x = np.linspace(-2, 2, 101)
fig, axs = plt.subplots(2, 2, figsize=(6, 4))
axs[0,0].plot(x, x, "b", label="linear")
axs[0,1].plot(x, x**2, "r", label="quadratic")
axs[1,0].plot(x, x**3, "g", label="cubic")
axs[1,1].plot(x, x**4, "c", label="quartic")
fig.legend(loc='upper right', bbox_to_anchor=(1.12, 0.9))
fig.suptitle("More subplots")

```

```python
x = np.linspace(0, 2*np.pi, 30)
y1 = np.sin(x)
y2 = np.cos(x)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 3))
ax1.plot(x, y1, "--b", label="sin(x)")
ax2.plot(x, y2, ".-r", label="cos(x)")
fig.suptitle("Subplot")
ax1.set_title("subplot 1")
ax2.set_title("subplot 2")
ax1.legend()
ax2.legend()

```

matplotlib: plotting data

Beyond creating plots for arrays (and lists), addressable objects like dicts and DataFrames can be used via data.

```
np.random.seed(19680801)  # seed the random number generator.
d = {'x': np.arange(50),
  'color': np.random.randint(0, 50, 50),
  'size': np.abs(np.random.randn(50)) * 100}
d['y'] = d['x'] + 10 * np.random.randn(50)
plt.figure(figsize=(6, 3))
plt.scatter('x', 'y', c='color', s='size', data=d)
plt.xlabel("x-axis")
plt.ylabel("y-axis")

```



matplotlib - pyplot w/ pandas data

Data can also come from `DataFrame` objects or series,

```python
df = pd.DataFrame({"x": np.random.normal(size=10000)
}).assign(y = lambda d: np.random.normal(0.75*d.x, np.sqrt(1-0.75**2), size=10000))
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter('x', 'y', c='k', data=df, alpha=0.1, s=0.5)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title("Bivariate normal ($\\rho=0.75$)")
```

Series objects can also be plotted directly. The index is used as the *x* axis values,

```python
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")
plt.plot(s)
plt.show()
```

matplolib: scales

Axis scales can be changed via `plt.xscale()`, `plt.yscale()`, `ax.set_xscale()`, or `ax.set_yscale()`, supported values are ”linear”, ”log”, ”symlog”, and ”logit”.

```
y = np.sort( np.random.sample(size=1000) )
x = np.arange(len(y))
plt.figure(layout="constrained")
scales = ['linear', 'log', 'symlog', 'logit']
for i, scale in zip(range(4), scales):
plt.subplot(221+i)
plt.plot(x, y)
plt.grid(True)
if scale == 'symlog':
plt.yscale(scale, linthresh=0.01)
else:
plt.yscale(scale)
plt.title(scale)
plt.show()

```

matplolib: categorical data

```python
df = pd.DataFrame({"cat": ["A", "B", "C", "D", "E"], "value": np.exp(range(5)) })

plt.figure(figsize=(4, 6), layout="constrained")
plt.subplot(321)
plt.scatter("cat", "value", data=df)
plt.subplot(322)
plt.scatter("value", "cat", data=df)
plt.subplot(323)
plt.plot("cat", "value", data=df)
plt.subplot(324)
plt.plot("value", "cat", data=df)
plt.subplot(325)
plt.bar("cat", "value", data=df)
plt.subplot(326)
plt.bar("value", "cat", data=df)
plt.show()

```

matplotlib: histograms

```python
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100)
})
plt.figure(figsize=(4, 6), layout="constrained")
plt.subplot(311)
plt.hist("x1", bins=10, data=df, alpha=0.5)
plt.hist("x2", bins=10, data=df, alpha=0.5)
plt.subplot(312)
plt.hist(df, alpha=0.5)
plt.subplot(313)
plt.hist(df, stacked=True, alpha=0.5)
plt.show()

```

matplotlib: boxplot

```python
df = pd.DataFrame({
  "x1": np.random.normal(size=100),
  "x2": np.random.normal(1,2, size=100),
  "x3": np.random.normal(-1,3, size=100)
}).melt()
plt.figure(figsize=(4, 4), layout="constrained")
plt.boxplot("value", positions="variable", data=df)

##ValueError: List of boxplot statistics and `positions` values must have same the length

```

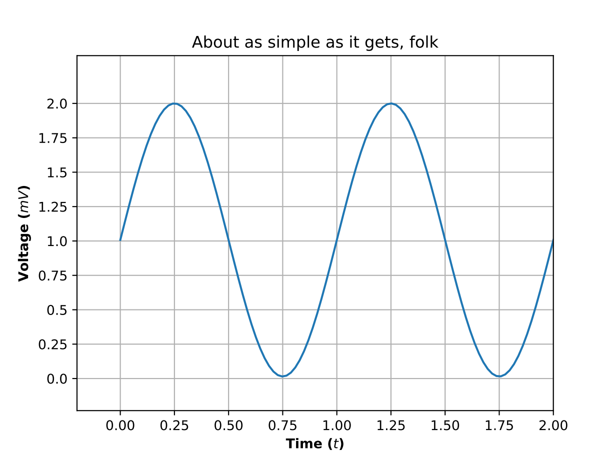
```python
plt.boxplot(df.value, positions=df.variable)
##ValueError: List of boxplot statistics and `positions` values must have same the length

```

matplotlib: other type of plots

https://matplotlib.org/stable/plot_types/index.html

matplotlib: Exercise 1

Using what we just learnt, recreate the following plot,



From matplotlib examples

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.

```python
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.

```python
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()
```