Matplotlib

Class name	Complete Machine Learning and Data Science Bootcamp
Dates	@Jun 16, 2021
Language	
Type	
lesson number	8
重要內容	

Remember🔥

- Extra Regular Expressions excerise: https://regexone.com/
- When plotting something quickly, it's okay to use the pyplot method
- When plotting something more advance, use the Object Oriented method

What is Matplotlib?

- Turn data into some pretty visualizations

Why Matplotlib?

- Built on NumPy arrays(and Pandas)
- Integrates directly with pandas
- Can create basic or advanced plots
- Simple to use interface (once you get the foundations)

A Matplotlib workflow

1. Create data

2. Create plot (figure, canvas)
3. Plot data (axes on figure)
4. Customise plot
5. Save/ share plot

Importing and using Matplotlib

- `%matplotlib inline` → **for plot to show in the notebook**
- `import matplotlib.pyplot as plt`
- plt.plot() → blank figure (canvas)
- plt.show() → You could use plt.show() if you want
- add some data → plt.plot([1, 2, 3, 4])
- create some data →
  - x = [1, 2, 3, 4]
  - y = [11, 22, 33, 44]
  - plt.plot(x, y);

2 ways of plotting: pyplot API & object-oriented API

- The pyplot API is generally less-flexible than the object-oriented API
- **In general, try to use the object-oriented interface over the pyplot interface.**

1st way (not recommend)

- fig = plt.figure()
- ax = fig.add_subplot()
- plt.show()

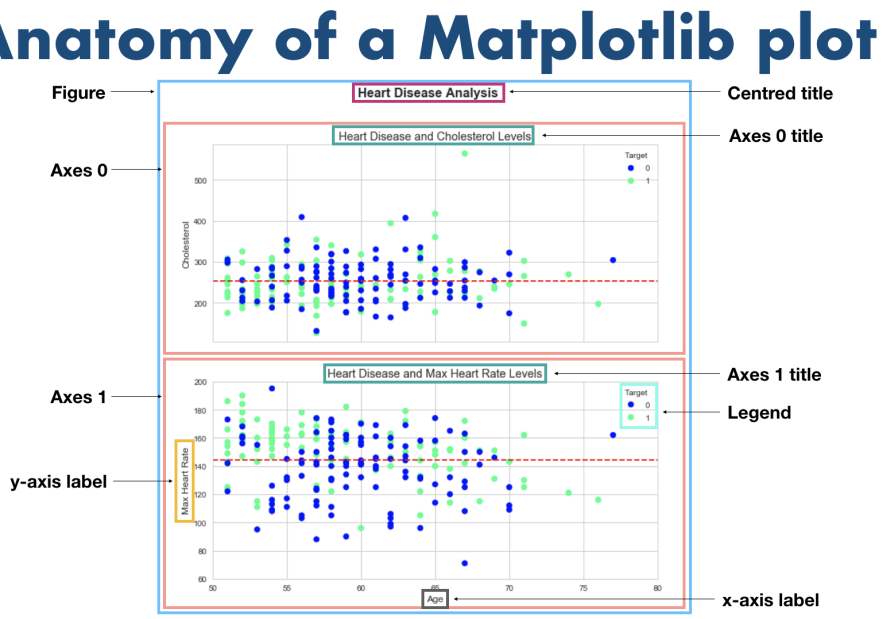
2nd way (not recommend)

- fig = plt.figure()

- ax = fig.add_axes([1, 1, 1, 1])
- ax.plot(x, y)
- plt.show()

3rd way (recommend)

- fig, ax = plt.subplots()
- ax.plot(x, y) #add some data
- Easier and more robust going forward (what we're going to use)



A matplotlib workflow Example

```
# 0. Import and get matplotlib ready
%matplotlib inline
importmatplotlib.pyplotasplt

# 1. Prepare data
x = [1, 2, 3, 4]
y = [11, 22, 33, 44]

# 2. Setup plot
fig, ax = plt.subplots(figsize=(10,10))
# (width, height)

# 3. Plot data
ax.plot(x, y)

# 4. Customize plot
```

```
ax.set(title="Sample Simple Plot", xlabel="x-axis", ylabel="y-axis")

# 5. Save & show
fig.savefig("../images/simple-plot.png")
```

Making plots using NumPy arrays

▼ Build some of the most common types of plots using NumPy arrays.

- line
- scatter
- bar
- hist
- subplots()

Line Chart

- Line is the default type of visualization in Matplotlib. Usually, unless specified otherwise, your plots will start out as lines.
- # Create an array

```
x = np.linspace(0, 10, 100)
# (start, stop, num)
x[:10]
```

- # Plot the data, **the default plot is line**

```
fig, ax = plt.subplots()
ax.plot(x, x**2);
```

Scatter Plot

- # **Need to recreate our figure and axis instances when we want a new figure**

```
fig, ax = plt.subplots()
ax.scatter(x, np.exp(x));
```

```
fig, ax = plt.subplots()
ax.scatter(x, np.sin(x));
```

Bar Chart

- # You can make plots from a dictionary

```
nut_butter_prices = {"Almond butter": 10,
                     "Peanut butter": 8,
                     "Cashew butter": 12}
```

- **Vertical Bar Chart**

```
fig, ax = plt.subplots()
ax.bar(nut_butter_prices.keys(), nut_butter_prices.values())
ax.set(title="Dan's Nut Butter Store", ylabel="Price ($)");
```

- **Horizontal Bar Chart - need to turn into list**

```
fig, ax = plt.subplots()
ax.barh(list(nut_butter_prices.keys()), list(nut_butter_prices.values()));
```

Histogram (hist)

- # Make some data from a normal distribution

```
x = np.random.randn(1000)
# pulls data from a normal distribution
```

- Could show image of normal distribution here

```
fig, ax = plt.subplots()
ax.hist(x);
```

```
x = np.random.random(1000)
# random data from random distribution

fig, ax = plt.subplots()
ax.hist(x);
```

2 options for Subplots - Multiple plots on one figure

- docs: https://matplotlib.org/3.1.1/gallery/recipes/create_subplots.html
- **Subplots option 1 (we will use this)**

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2,
                                             ncols=2,
                                             figsize=(10, 5))

# Plot data to each axis
ax1.plot(x, x/2);
ax2.scatter(np.random.random(10), np.random.random(10));
ax3.bar(nut_butter_prices.keys(), nut_butter_prices.values());
ax4.hist(np.random.randn(1000));
```

- **Subplots option 2**

```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(10, 5))

# Index to plot data
ax[0, 0].plot(x, x/2);
ax[0, 1].scatter(np.random.random(10), np.random.random(10));
ax[1, 0].bar(nut_butter_prices.keys(), nut_butter_prices.values());
ax[1, 1].hist(np.random.randn(1000));
```

Plotting data directly with pandas

▼ This section uses the pandas pd.plot() method on a DataFrame to plot columns directly.

- https://datatofish.com/plot-dataframe-pandas/
- https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- line
- scatter
- bar
- hist
- df.plot(subplots=True, figsize=(6, 6))

- Now we need some data to check out. Let's import the car_sales dataset

```
car_sales = pd.read_csv("../data/car-sales.csv")
car_sales
```

Line Chart

- Let's start with trying to replicate the pandas visualization documents.

```
# Start with some dummy data
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('1/1/2020', periods=1000))
ts
```

```
# What does cumsum() do?
ts.cumsum()
```

```
ts.cumsum().plot();
```

Working with actual data

- Let's do a little data manipulation on our car_sales DataFrame.
- # Remove price column symbols

```
car_sales["Price"] = car_sales["Price"].str.replace('[\$\,\.]', '')
car_sales
```

- # Remove last two zeros

```
car_sales["Price"] = car_sales["Price"].str[:-2]
car_sales
```

- # Add a date column

```
car_sales["Sale Date"] = pd.date_range("1/1/2020", periods=len(car_sales))
car_sales
```

- # Make total sales column

```
#(doesn't work, adds as string)
car_sales["Total Sales"] = car_sales["Price"].cumsum()
```

```
# Oops... want them as ints not string
car_sales["Total Sales"] = car_sales["Price"].astype(int).cumsum()
car_sales
```

- # Plot the total sales column

```
car_sales.plot(x='Sale Date', y='Total Sales');
```

Scatter Plot

- # Doesn't work - scatter requires y column to be numeric

```
car_sales.plot(x="Odometer (KM)", y="Price", kind="scatter")
```

- # Convert Price to int

```
car_sales["Price"] = car_sales["Price"].astype(int)
# Plot it
car_sales.plot(x="Odometer (KM)", y="Price", kind='scatter');
```

Bar Chart

- **Aggregate** : Operate on many records to produce 1 value- Aggregate data, produce a summary
- # Create some data first and turn it into dataframe

```
x = np.random.rand(10, 4)
df = pd.DataFrame(x, columns=['a', 'b', 'c', 'd'])
```

- # Bar chart 1

```
df.plot.bar();
```

- # Bar chart 2, with 'kind' keyword

```
df.plot(kind='bar');
```

- # Bar chart with car_sales

```
car_sales.plot(x='Make', y='Odometer (KM)', kind='bar');
```

Histograms

- # Histograms 1

```
car_sales["Odometer (KM)"].plot.hist();
```

- # Histograms 2, with 'kind' keyword

```
ar_sales["Odometer (KM)"].plot(kind="hist");
```

- # Changing bin → Default number of bins is 10

```
car_sales["Odometer (KM)"].plot.hist(bins=20);
```

Subplots

- # Let's try with another dataset

```
heart_disease = pd.read_csv("../data/heart-disease.csv")
heart_disease.head()
```

- # every single column's histogram in one plot

```
heart_disease.plot.hist(figsize=(10, 30), subplots=True);
```

Plotting with pandas using the OO (Object Oriented) method

- # Perform data analysis on patients over 50.

```
over_50 = heart_disease[heart_disease["age"] > 50]
```

- # **Python method**

```
over_50.plot(kind='scatter',
             x='age',
             y='chol',
             c='target',
             figsize=(10, 6));
# c is color
```

- # **OO method mix with of pyplot method**

```
fig, ax = plt.subplots(figsize=(10, 6))
over_50.plot(kind='scatter',
             x="age",
             y="chol",
             c='target',
             ax=ax);

# set the x axis
ax.set_xlim([45, 100]);
```

- # **Pure OO method** - Make a bit more complicated plot
- add a horizontal docs→ https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.axes.Axes.axhline.html

```
# Create the plot
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the data
scatter = ax.scatter(x=over_50["age"],
                     y=over_50["chol"],
                     c=over_50["target"])

# Customize the plot
ax.set(title="Heart Disease and Cholesterol Levels",
       xlabel="Age",
       ylabel="Cholesterol");
```

```
# Add a legend - the c parameter
ax.legend(*scatter.legend_elements(), title="Target");

# Add a horizontal line (meanline)
ax.axhline(over_50["chol"].mean(), linestyle="--");
```

Adding another plot to existing styled one

- **# Subplot of chol, age, thalach**

```
# Setup blank figures (2 rows, 1 column)
fig, (ax0, ax1) = plt.subplots(nrows=2, # 2 rows
                               ncols=1,
                               sharex=True,
                               figsize=(10, 8))
# --------------------ax0-------------------
# Add data for ax0
scatter = ax0.scatter(x=over_50["age"],
                      y=over_50["chol"],
                      c=over_50["target"])
# Customize ax0
ax0.set(title="Heart Disease and Cholesterol Levels",
        ylabel="Cholesterol")
ax0.legend(*scatter.legend_elements(), title="Target")

# Setup a mean line
ax0.axhline(y=over_50["chol"].mean(),
            color='b',
            linestyle='--',
            label="Average")

# --------------------ax1-------------------
# Add data for ax1
scatter = ax1.scatter(over_50["age"],
                      over_50["thalach"],
                      c=over_50["target"])

# Customize ax1
ax1.set(title="Heart Disease and Max Heart Rate Levels",
        xlabel="Age",
        ylabel="Max Heart Rate")
ax1.legend(*scatter.legend_elements(), title="Target")

# Setup a mean line
ax1.axhline(y=over_50["thalach"].mean(),
            color='b',
            linestyle='--',
            label="Average")

# Add a title to the figure
fig.suptitle('Heart Disease Analysis', fontsize=16, fontweight='bold');
```

Customize your plots

Style

- Try out some different styles

```
# See the available styles
plt.style.available
```

- /** all kinds of style
  ['seaborn-dark',
  'seaborn-darkgrid',
  'seaborn-ticks',
  'fivethirtyeight',
  'seaborn-whitegrid',
  'classic',
  '_classic_test',
  'fast',
  'seaborn-talk',
  'seaborn-dark-palette',
  'seaborn-bright',
  'seaborn-pastel',
  'grayscale',
  'seaborn-notebook',
  'ggplot',
  'seaborn-colorblind',
  'seaborn-muted',
  'seaborn',
  'Solarize_Light2',
  'seaborn-paper',
  'bmh',
  'tableau-colorblind10',
  'seaborn-white',
  'dark_background',
  'seaborn-poster',
  'seaborn-deep']
  **/

```
# Plot before changing style
car_sales["Price"].plot();

# Change the style...
plt.style.use('seaborn-whitegrid')
```

```
# Change to another style...
plt.style.use('seaborn')
```

```
# Change to another style...
plt.style.use('ggplot')
```

- WHERE "<column>" IN (value1, value2, ...)
- limits (xlim, ylim), colors, styles, legends
- WHERE "<column>" IN (value1, value2, ...)

Changing the title, legend, axes

```
# First create some data
x = np.random.randn(10, 4)

df = pd.DataFrame(x, columns=['a', 'b', 'c', 'd'])
```

- # **set()** method to Customize our plot with title and labels

```
ax = df.plot(kind='bar')

# Add labels and title
ax.set(title="Random Number Bar Graph from DataFrame",
       xlabel="Row number",
       ylabel="Random number")

# Make the legend visible
ax.legend().set_visible(True)
```

Changing the cmap (color scheme)

- Choosing Colormaps in Matplotlib docs: https://matplotlib.org/stable/tutorials/colors/colormaps.html

- Customize our own style within the SET style
- # use the SET style

```
# SET the style
plt.style.use('seaborn-whitegrid')

# No cmap change
fig, ax = plt.subplots(figsize=(10, 6))
scatter = ax.scatter(over_50["age"],
                     over_50["chol"],
                     c=over_50["target"])
ax.set(title="Heart Disease and Cholesterol Levels",
       xlabel="Age",
       ylabel="Cholesterol");
ax.axhline(y=over_50["chol"].mean(),
           c='b',
           linestyle='--',
           label="Average");
ax.legend(*scatter.legend_elements(), title="Target");
```

- # Customize our own style within the SET style

```
# SET the style
plt.style.use('seaborn-whitegrid')

# Change cmap and horizontal line to be a different colour
fig, ax = plt.subplots(figsize=(10, 6))
scatter = ax.scatter(over_50["age"],
                     over_50["chol"],
                     c=over_50["target"],
                     cmap="winter")
ax.set(title="Heart Disease and Cholesterol Levels",
       xlabel="Age",
       ylabel="Cholesterol")
ax.axhline(y=over_50["chol"].mean(),
           color='r',
           linestyle='--',
           label="Average");
ax.legend(*scatter.legend_elements(), title="Target");
```

Changing the xlim & ylim

- # adding in different x & y limitations - with set_xlim() & set_ylim()

```
## After adding in different x & y limitations

fig, (ax0, ax1) = plt.subplots(nrows=2, ncols=1, sharex=True, figsize=(10, 10))
scatter = ax0.scatter(over_50["age"],
                      over_50["chol"],
```

```
                      c=over_50["target"],
                      cmap='winter')
ax0.set(title="Heart Disease and Cholesterol Levels",
        ylabel="Cholesterol")

# Set the x axis
ax0.set_xlim([50, 80])

# Setup a mean line
ax0.axhline(y=over_50["chol"].mean(),
            color='r',
            linestyle='--',
            label="Average");
ax0.legend(*scatter.legend_elements(), title="Target")

# Axis 1, 1 (row 1, column 1)
scatter = ax1.scatter(over_50["age"],
                      over_50["thalach"],
                      c=over_50["target"],
                      cmap='winter')
ax1.set(title="Heart Disease and Max Heart Rate Levels",
        xlabel="Age",
        ylabel="Max Heart Rate")

# Set the y axis
ax1.set_ylim([60, 200])

# Setup a mean line
ax1.axhline(y=over_50["thalach"].mean(),
            color='r',
            linestyle='--',
            label="Average");
ax1.legend(*scatter.legend_elements(), title="Target")

# Title the figure
fig.suptitle('Heart Disease Analysis', fontsize=16, fontweight='bold');
```

Saving plots

- way1. Just click "save image as"...
- way2. Saving plots to images using fig.savefig("filename.png")

fig.savefig("filename.png")

```
# Save the file
fig.savefig("../images/heart-disease-analysis.png")
```

```
# Check the supported filetypes
fig.canvas.get_supported_filetypes()
```

- → {'ps': 'Postscript',
  'eps': 'Encapsulated Postscript',
  'pdf': 'Portable Document Format',
  'pgf': 'PGF code for LaTeX',
  'png': 'Portable Network Graphics',
  'raw': 'Raw RGBA bitmap',
  'rgba': 'Raw RGBA bitmap',
  'svg': 'Scalable Vector Graphics',
  'svgz': 'Scalable Vector Graphics'}

```
# Check our fig
fig
```

```
# Resets figure
fig, ax = plt.subplots()
```

A function which follows the Matplotlib workflow.

- If you're doing something like this often, to save writing excess code, you might put it into a function.

```
# Potential function

def plotting_workflow(data):
    # 1. Manipulate data

    # 2. Create plot

    # 3. Plot data

    # 4. Customize plot

    # 5. Save plot

    # 6. Return plot

    return plot
```