Python for Data Science

Kylie A. Bemis

Northeastern University
Khoury College of Computer Sciences

Northeastern University

Goals for today

- Data frames with **pandas**
- Visualization with **matplotlib**
- Modeling with **statsmodels** and **sklearn**
- Review (Q&A)

SERIES AND DATA FRAMES

Working with data in **pandas**

- Structures for representing *tabular data*
  - **Series** represent *homogenous* data attributes (vectors)
  - **DataFrame**s represent tabular data (rows and columns)
- Methods for manipulating data tables
  - Subsetting, grouping, and summarization
  - Joining and organizing multiple tables
- Built on top of **NumPy**

A **Series** represents a data vector

- *Homogenous* vector consisting of:
  - **Values** in the data series
  - **Index** of labels for each data element
  - Data type (**dtype**) of the values
- Built from a `numpy.ndarray`

A **DataFrame** represents tabular data

- Data table made of rows and columns
  - Each *column* is represented by a **Series**
  - Columns may have *different* data types (**dtype**)
  - Columns must *share* the *same* **index** labels
  - Each *row* is identified by an **index** label
- Similar to a `dict` of data columns

Creating a **DataFrame** from a `dict`

```
In : import pandas as pd

In : A = pd.DataFrame(
         {'x': [100, 200, 300, 400],
          'y': [1.11, 2.22, 3.33, 4.44],
          'z': ['foo', 'bar', 'baz', 'qux']},
         index=['a', 'b', 'c', 'd'])
In : A
Out:
     x     y    z
a  100  1.11  foo
b  200  2.22  bar
c  300  3.33  baz
d  400  4.44  qux
```

Advanced pandas

- Grouping and aggregation
  - Summary statistics over rows or columns
  - Aggregation over categories in a column
- Merging and other relational operations
  - Join two or more tables based on common rows
  - Similar to SQL operations on a database
- Input to other data science libraries

VISUALIZATION

Visualization with **matplotlib**

- Provides MATLAB-like plotting interface
  - Flexible object-oriented plotting
  - Figures organized into hierarchical structure
  - Support for different GUI backends
- Export to many formats (PDF, PNG, etc.)

Plotting a basic scatter plot

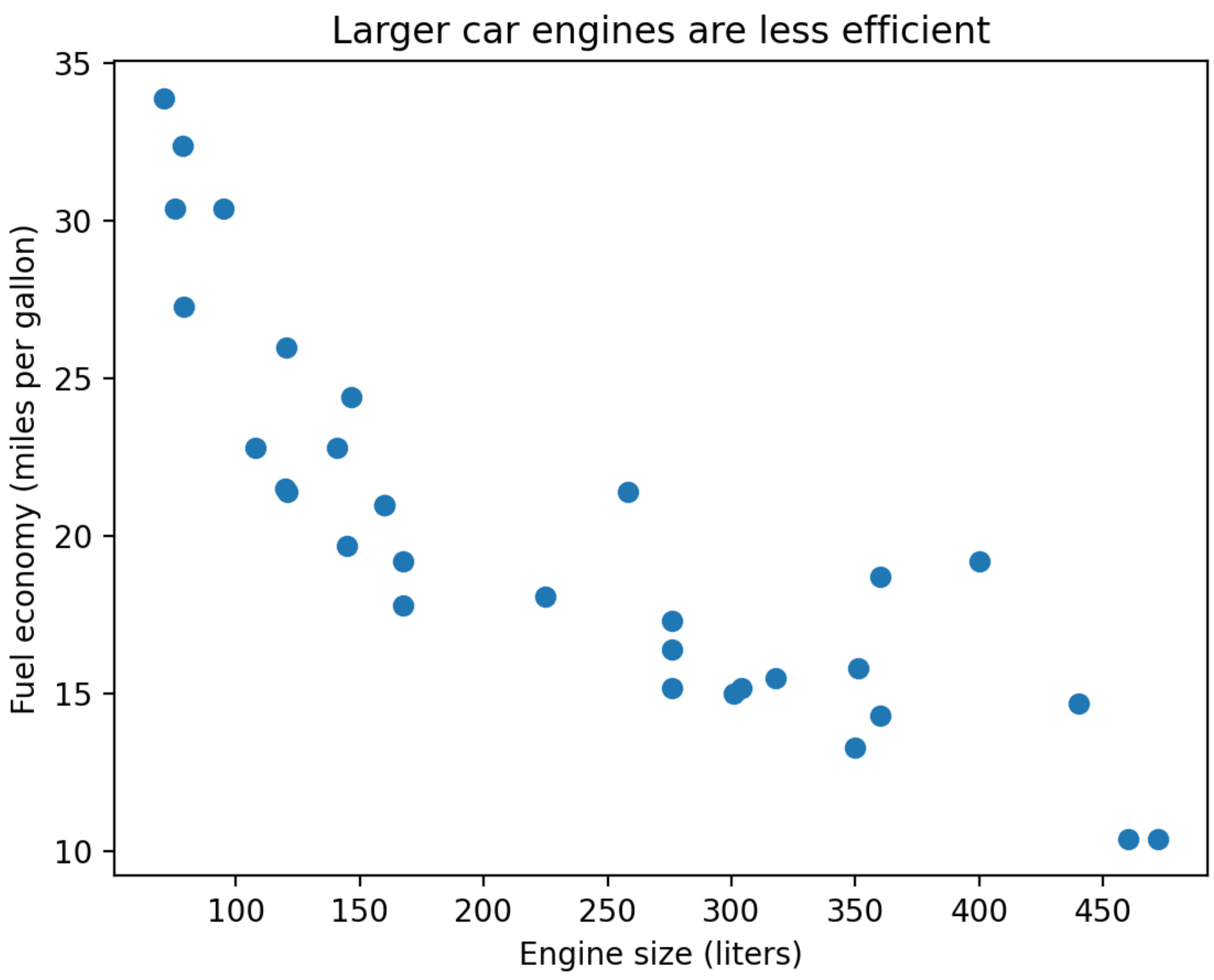
```
import matplotlib.pyplot as plt

cars = pd.read_csv("mtcars.csv", index_col=0)

xs = cars['disp']
ys = cars['mpg']

plt.scatter(xs, ys)
plt.title("Larger car engines are less efficient")
plt.xlabel("Engine size (liters)")
plt.ylabel("Fuel economy (miles per gallon)")
plt.show()
```

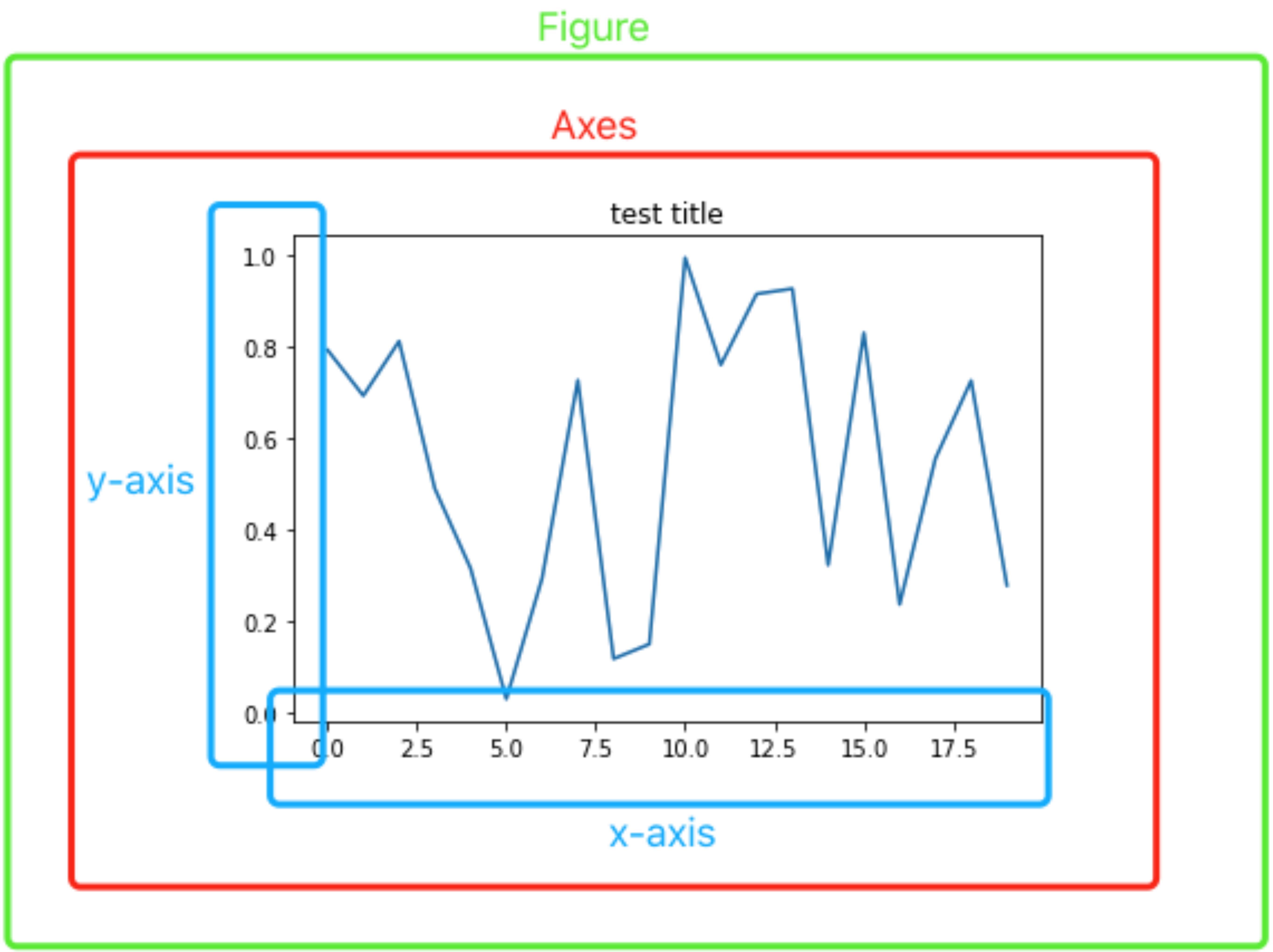
Plotting a basic scatter plot (2)



Hierarchy of a figure

- A **Figure** is *container* for other graphics
  - Outermost layer of a matplotlib graphic
  - Contains one or more Axes objects
- **Axes** objects contain *subplots*
  - Contains an individual plot or graphic
  - Does not refer to a traditional plot "axis"

Figures and Axes



https://towardsdatascience.com/what-are-the-plt-and-ax-in-matplotlib-exactly-d2cf4bf164a9

Plotting a figure with subplots

```
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

ax1.scatter('disp', 'mpg', data=cars, color='tab:blue')
ax1.set_title("Mileage vs. engine size")
ax1.set_xlabel("Engine size (liters)")
ax1.set_ylabel("Fuel economy (mpg)")

...

ax4.scatter('qsec', 'mpg', data=cars, color='tab:red')
ax4.set_title("Mileage vs. acceleration")
ax4.set_xlabel("Time to 1/4 mile (sec)")
ax4.set_ylabel("Fuel economy (mpg)")

fig.show()
```

Plotting a figure with subplots (2)

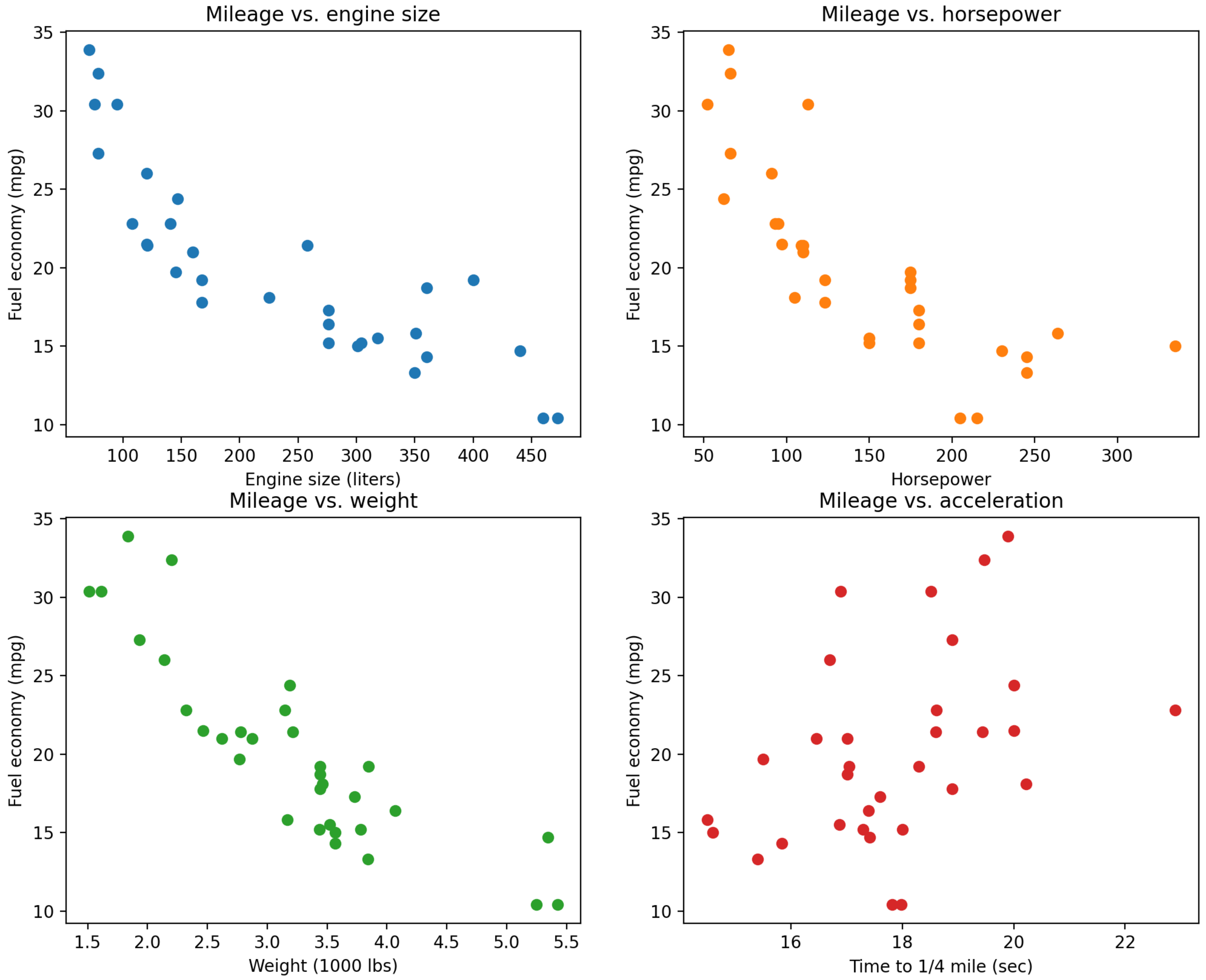
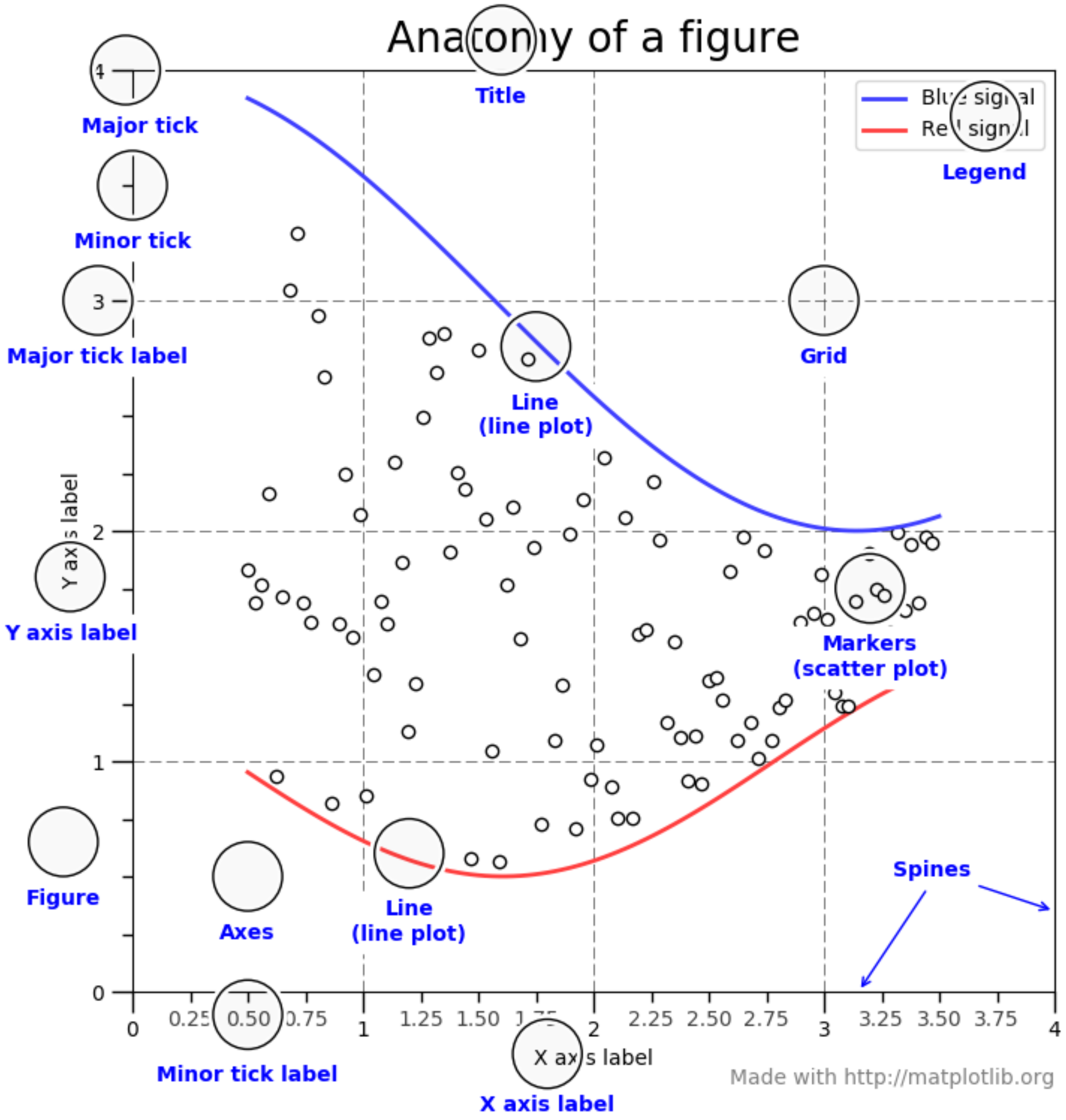


Figure components

- Title
- Labels
- Axes
- Ticks
- Legend
- Grid
- Spines
- Markers



More on visualization

- **Always label your figures!!!**
  - Label the axes (include units)
  - Provide useful and informative titles
  - Include a legend whenever necessary
- Consider higher-level libraries
  - **Seaborn** provides higher-level plotting interface
  - Most built on top of **matplotlib**

MODELING

Modeling in Python

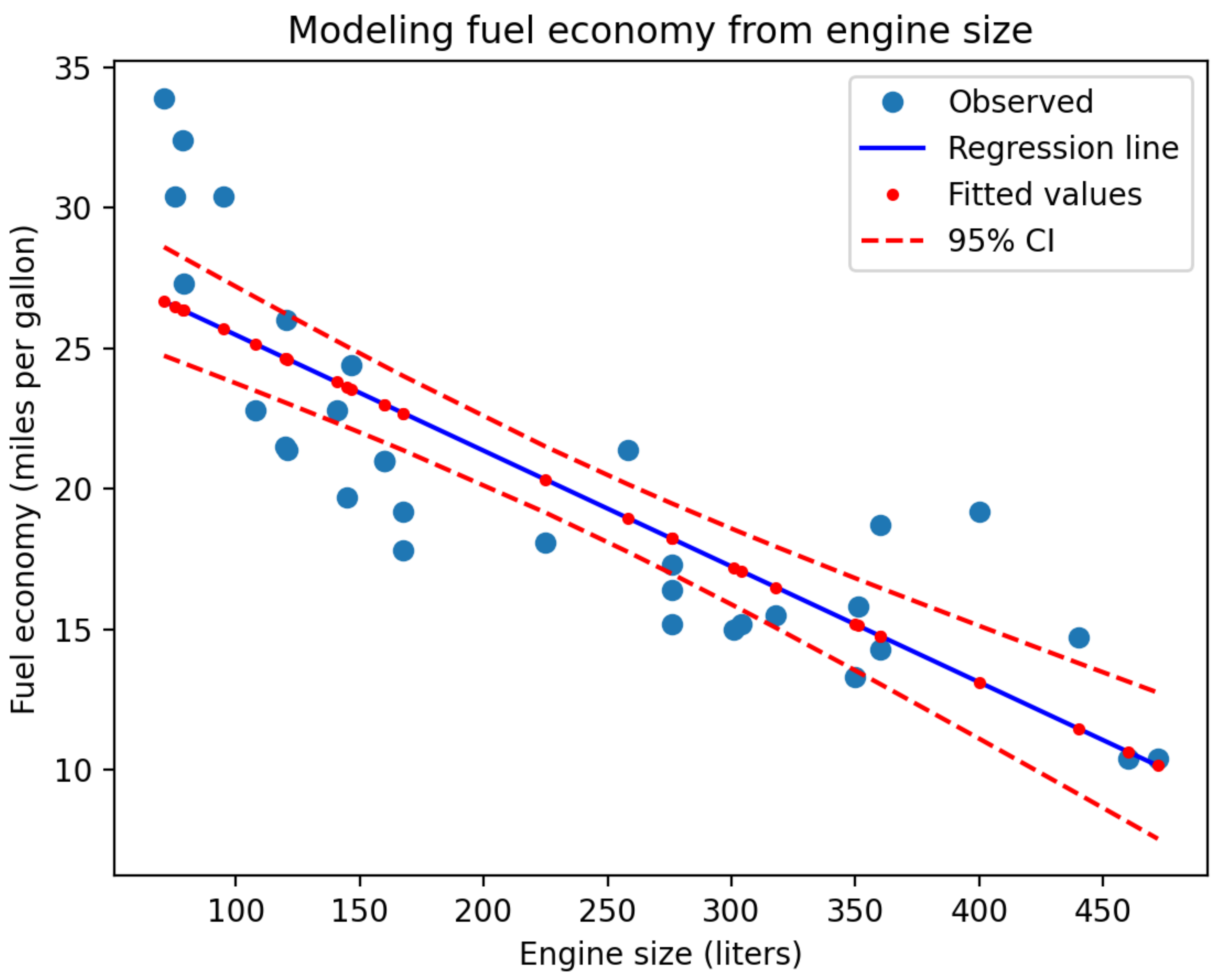
- **statsmodels** provides statistical models
  - More focus on *statistical inference*
  - More similar to other statistical software (R, Stata, etc.)
- **scikit-learn** performs machine learning
  - More focus on *predictive performance*
  - Defaults may differ from statistics-focused alternatives
  - Very mature platform for machine learning
- Consider primary goals of analysis

Regression with **statsmodels**

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.718
Model:                            OLS   Adj. R-squared:                  0.709
Method:                 Least Squares   F-statistic:                     76.51
Date:                Tue, 13 Apr 2021   Prob (F-statistic):           9.38e-10
Time:                        17:21:57   Log-Likelihood:                -82.105
No. Observations:                  32   AIC:                             168.2
Df Residuals:                      30   BIC:                             171.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     29.5999      1.230     24.070      0.000      27.088      32.111
disp          -0.0412      0.005     -8.747      0.000      -0.051      -0.032
==============================================================================
Omnibus:                        3.368   Durbin-Watson:                   0.986
Prob(Omnibus):                  0.186   Jarque-Bera (JB):                3.049
Skew:                           0.719   Prob(JB):                        0.218
Kurtosis:                       2.532   Cond. No.                         558.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Regression with **statsmodels** (2)



Classification with **sklearn**

```
In : from sklearn.model_selection import train_test_split

In : sonar = pd.read_csv("sonar.csv")
In : sonar_X = sonar.iloc[:,0:60]
In : sonar_y = sonar["Class"]

In : part = train_test_split(sonar_X, sonar_y, test_size=0.2)

In : X_train, X_test, y_train, y_test = part

In : model2.score(X_train, y_train)  # accuracy (training)
Out: 0.8975903614457831

In : model2.score(X_test, y_test)  # accuracy (testing)
Out: 0.8333333333333334
```

Classification with **sklearn** (2)

```
In : from sklearn.metrics import classification_report

In : classification_report(y_test, model2.predict(X_test))
Out:
              precision    recall  f1-score   support

        Mine       0.80      0.91      0.85        22
        Rock       0.88      0.75      0.81        20

    accuracy                           0.83        42
   macro avg       0.84      0.83      0.83        42
weighted avg       0.84      0.83      0.83        42
```

Modeling considerations

- “All models are wrong, but some are useful.”
  - Consider what is the goal of the analysis
  - Evaluate a model by how *useful* it is for the goal
- Don’t use a model you don’t understand
  - Understand modeling assumptions
  - What kinds of models are appropriate?
- Follow best practices for model evaluation

REVIEW