JupyterLite チュートリアル

実践ガイドブック

Python データサイエンス入門

河合勝彦

名古屋市立大学大学院経済学研究科

kkawai@econ.nagoya-cu.ac.jp

2026 年 1 月 9 日

はじめに

本書は、JupyterLite 上で Python データサイエンスを学ぶための実践的なガイドブックです。ブラウザだけで動作する JupyterLite 環境を活用し、NumPy、pandas、matplotlib、scipy、scikit-learn といった主要ライブラリの基礎を習得することを目的としています。

本書の特徴

- **実践的**：すべてのコードは JupyterLite 上で実際に動作します
- **段階的**：初級から順を追って学習できる構成
- **演習付き**：各章末に練習問題と模範解答を用意
- **インストール不要**：ブラウザさえあればすぐに始められます

対象読者

- Python の基本文法（変数、リスト、ループ、関数）を理解している方
- データ分析・機械学習を学び始めたい方
- 大学の授業や自習でデータサイエンスを学ぶ学生

本書の使い方

本書を読みながら、対応する Jupyter Notebook を開いて実際にコードを実行してください。各章の「重要ポイント」は特に理解を深めるべき概念を示しています。練習問題は自分で解いてから模範解答を確認することをお勧めします。

Contributors

- katzkawai@gmail.com

目次

第 I 部

JupyterLite 入門

第 1 章

JupyterLite の基本

1.1 JupyterLite とは

JupyterLite は、**ブラウザだけで完結する Jupyter Notebook 環境**です。通常の Jupyter Notebook とは異なり、サーバーへのインストールが不要で、URL にアクセスするだけで利用できます。

重要ポイント

JupyterLite の主な特徴：

- インストール不要（ブラウザだけで動作）
- 軽量・高速な起動
- OS を問わない（Windows / Mac / Linux / ChromeOS）
- 完全ローカルでも動作可能

1.1.1 狙い

本チュートリアルでは、JupyterLite の基本操作を習得し、Python・NumPy・pandas・matplotlib を使った簡単なデータ分析ができるようになることを目指します。

1.2 Notebook の基本操作

Notebook は「**セル**」という単位でコードや文章を管理します。

種類	用途	見分け方
Code セル	Python コードの実行	左側に `[ ]:` が表示される
Markdown セル	説明文・見出し	実行すると整形されたテキスト

表 1.1　セルの種類

操作	ショートカット
セルを実行	`Shift + Enter`
セルを実行（移動なし）	`Ctrl + Enter`
上にセル追加	`A`
下にセル追加	`B`
セル削除	`D D`（D を 2 回）
Markdown に変更	`M`
Code に変更	`Y`

表 1.2　よく使うショートカット

1.2.1　セルの種類

1.2.2　重要なショートカット

1.3　Python の基本

JupyterLite では Python のコードを対話的に実行できます。

```python
print("Hello JupyterLite!")

# 1 から 10 までの合計
total = sum(range(1, 11))
print(f"1から10の合計: {total}")
```

Listing 1.1　最初の Python コード

1.3.1　練習問題と解答

練習問題

1. JupyterLite と通常の Jupyter Notebook（ローカル版）の違いを 2 点挙げて説明しなさい。
2. 自分の名前を含むメッセージを表示するコードを書きなさい。
3. 1 から 100 までの整数の合計を計算して表示するコードを書きなさい。

模範解答

1. 違いの例

- 通常の Jupyter Notebook は、自分の PC に Python や必要なライブラリをインストールする必要があるが、JupyterLite はブラウザだけで動作し、インストールがいらない。
- 通常版はローカルの Python 実行環境を使うのに対し、JupyterLite は Pyodide（ブラウザ上の Python）を利用している。

2. 名前を含むメッセージ

```
print("こんにちは、私は田中です。")
```

3. 1 から 100 までの合計

```
total = 0
for i in range(1, 101):
    total += i
print("1から100までの合計:", total)
# 出力: 1から100までの合計: 5050
```

第 II 部

NumPy による数値計算

第 2 章

NumPy 初級

2.1 NumPy とは

NumPy（Numerical Python）は、Python で高速な数値計算を行うためのライブラリです。

2.1.1 狙い

本チュートリアルでは、NumPy の中心的なデータ構造である `ndarray` の作成・操作・統計計算を習得します。

重要ポイント

NumPy の主な特徴：

- **ndarray**: 高速な多次元配列オブジェクト
- **ベクトル化演算**: ループを使わずに配列全体に演算を適用
- **ブロードキャスト**: 異なる形状の配列間での演算
- Python リストより約 10〜100 倍高速

2.2 配列の作成

2.2.1 リストから配列を作成

```
import numpy as np

# 1次元配列
arr1d = np.array([1, 2, 3, 4, 5])
print(arr1d)  # [1 2 3 4 5]

# 2次元配列
arr2d = np.array([[1, 2, 3],
                  [4, 5, 6]])
```

```
print(arr2d)
```

Listing 2.1 NumPy 配列の作成

2.2.2 特殊な配列の作成

```
# ゼロ配列
zeros = np.zeros((3, 4))

# 1で満たされた配列
ones = np.ones((2, 3))

# 等差数列
arr_range = np.arange(0, 10, 2)  # [0 2 4 6 8]

# 等間隔の数列
arr_lin = np.linspace(0, 1, 5)  # [0. 0.25 0.5 0.75 1.]
```

Listing 2.2 特殊な配列

2.3 インデックスとスライス

```
arr = np.array([10, 20, 30, 40, 50])

# インデックスでアクセス
print(arr[0])   # 10（最初の要素）
print(arr[-1])  # 50（最後の要素）

# スライス
print(arr[1:4])  # [20 30 40]

# ブールインデックス
print(arr[arr > 30])  # [40 50]
```

Listing 2.3 インデックスとスライス

重要ポイント

ブールインデックスは条件に合う要素を抽出する強力な機能です。データのフィルタリングに頻繁に使用します。

2.4 統計関数

```
data = np.array([78, 85, 92, 67, 88])
```

```

print(f"平均: {np.mean(data):.2f}")     # 82.00
print(f"中央値: {np.median(data):.2f}")  # 85.00
print(f"標準偏差: {np.std(data):.2f}")   # 8.60
print(f"最大値: {np.max(data)}")         # 92
print(f"最小値: {np.min(data)}")         # 67
```

Listing 2.4　基本的な統計関数

2.4.1　練習問題と解答

練習問題

1. 5x5 のゼロ行列を作成し、対角成分を 1, 2, 3, 4, 5 に設定してください。
2. 0 から 100 まで 11 個の等間隔の数列を作成してください。
3. 以下の配列から偶数の要素のみを抽出してください。

```
arr = np.arange(1, 26).reshape(5, 5)
```

4. 以下のデータの平均点、最高点、最低点、レンジを求めてください。

```
scores = np.array([78, 85, 92, 67, 88, 95, 72, 81, 89, 76])
```

模範解答

1. 対角成分を設定した行列

```
import numpy as np

mat = np.zeros((5, 5))
np.fill_diagonal(mat, [1, 2, 3, 4, 5])
print(mat)

# 別解: np.diag を使う
mat2 = np.diag([1, 2, 3, 4, 5])
print(mat2)
```

2. 等間隔の数列

```
print(np.linspace(0, 100, 11))
# [  0.  10.  20.  30.  40.  50.  60.  70.  80.  90. 100.]
```

3. 偶数の要素を抽出

```
arr = np.arange(1, 26).reshape(5, 5)
print(arr[arr % 2 == 0])
# [ 2  4  6  8 10 12 14 16 18 20 22 24]
```

4. 統計量の計算

```
scores = np.array([78, 85, 92, 67, 88, 95, 72, 81, 89, 76])
print(f"平均点: {np.mean(scores):.2f}")    # 82.30
print(f"最高点: {np.max(scores)}")          # 95
print(f"最低点: {np.min(scores)}")          # 67
print(f"レンジ: {np.ptp(scores)}")          # 28
```

第 III 部

pandas によるデータ操作

第 3 章

pandas 初級

3.1 pandas とは

pandas は、表形式のデータを簡単に扱うためのライブラリです。Excel のスプレッドシートのようなデータ構造を Python で操作できます。

3.1.1 狙い

本チュートリアルでは、Series と DataFrame の基本操作、データの読み込み・フィルタリング・集計を習得します。

重要ポイント

pandas の 2 つの主要なデータ構造：

- **Series**: 1 次元のラベル付き配列（Excel の 1 列分）
- **DataFrame**: 2 次元のラベル付き表（Excel シート全体）

3.2 Series の基本

```python
import pandas as pd

# インデックス付きで作成
s = pd.Series([85, 92, 78, 95],
              index=["数学", "英語", "国語", "理科"])
print(s)
print(f"平均: {s.mean()}")
```

Listing 3.1 Series の作成

3.3 DataFrame の作成

```python
import pandas as pd

# 辞書から DataFrame を作成
data = {
    "名前": ["田中", "佐藤", "鈴木"],
    "年齢": [22, 25, 30],
    "部署": ["営業", "開発", "営業"]
}

df = pd.DataFrame(data)
print(df)
```

Listing 3.2 DataFrame の作成

出力：

	名前	年齢	部署
0	田中	22	営業
1	佐藤	25	開発
2	鈴木	30	営業

3.4 データの選択

```python
# 列の選択
df["名前"]            # 1列を Series として取得
df[["名前", "年齢"]]  # 複数列を DataFrame として取得

# loc: ラベルで選択
df.loc[0, "名前"]     # "田中"

# iloc: 位置（番号）で選択
df.iloc[0, 0]         # "田中"
```

Listing 3.3 データの選択方法

注意

loc と iloc の違いに注意：

- `loc[0:2]` → インデックス 0, 1, 2 の行（**終端を含む**）
- `iloc[0:2]` → 位置 0, 1 の行（**終端を含まない**）

3.5 条件によるフィルタリング

```python
# 年齢が 25 以上
df[df["年齢"] >= 25]

# 複数条件（AND）
df[(df["部署"] == "営業") & (df["年齢"] >= 25)]

# 複数条件（OR）
df[(df["部署"] == "営業") | (df["部署"] == "開発")]

# query メソッド
df.query("年齢 >= 25 and 部署 == '営業'")
```

Listing 3.4　条件フィルタリング

3.5.1　練習問題と解答

練習問題

1. 自分の好きな食べ物を 5 つ選び、その値段（円）を Series として作成しなさい。平均値、最大値、最小値を求めなさい。
2. 以下の列を持つ DataFrame を作成しなさい：「商品名」「カテゴリ」「価格」「在庫数」（5 行以上）
3. 作成した DataFrame に「在庫金額」（価格 × 在庫数）という新しい列を追加しなさい。
4. 給与が 320000 円以上の人を抽出するコードを書きなさい。

模範解答

1. Series の作成と統計量

```python
prices = pd.Series({
    "りんご": 150,
    "バナナ": 100,
    "みかん": 80,
    "ぶどう": 300,
    "いちご": 400
})
print(f"平均: {prices.mean()}")  # 206.0
print(f"最大: {prices.max()}")   # 400
print(f"最小: {prices.min()}")   # 80
```

2. DataFrame の作成

```python
df = pd.DataFrame({
    "商品名": ["ノート", "ペン", "消しゴム", "定規", "ファイル"],
    "カテゴリ": ["文具", "文具", "文具", "文具", "収納"],
    "価格": [200, 150, 100, 300, 500],
```

```
    "在庫数": [50, 100, 80, 30, 20]
})
print(df)
```

3. 在庫金額列の追加

```
df["在庫金額"] = df["価格"] * df["在庫数"]
print(df)
```

4. フィルタリング

```
# 給与 DataFrame があると仮定
df_salary = pd.DataFrame({
    "名前": ["田中", "佐藤", "鈴木"],
    "給与": [300000, 350000, 320000]
})
print(df_salary[df_salary["給与"] >= 320000])
```

第 IV 部

matplotlib による可視化

第 4 章

matplotlib 初級

4.1 matplotlib とは

matplotlib は、Python で最も広く使われているグラフ描画ライブラリです。

4.1.1 狙い

本チュートリアルでは、折れ線グラフ、散布図、棒グラフ、ヒストグラム、円グラフの作成方法を習得します。

4.2 基本のグラフ

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title("サイン波")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)
plt.show()
```

Listing 4.1　基本の折れ線グラフ

4.3 グラフの種類

4.4 散布図

```
np.random.seed(42)
```

グラフ	関数	用途
折れ線グラフ	plt.plot()	時系列データ、連続的な変化
散布図	plt.scatter()	2 変数の関係、相関
棒グラフ	plt.bar()	カテゴリ別の比較
ヒストグラム	plt.hist()	データの分布
円グラフ	plt.pie()	全体に対する割合

表 4.1　主なグラフの種類

```
x = np.random.randn(50)
y = x + np.random.randn(50) * 0.5

plt.scatter(x, y, alpha=0.7)
plt.title("散布図")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

Listing 4.2　散布図

4.5　棒グラフ

```
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 32, 48]

plt.bar(categories, values)
plt.title("カテゴリ別の値")
plt.xlabel("カテゴリ")
plt.ylabel("値")
plt.show()
```

Listing 4.3　棒グラフ

4.6　ヒストグラム

```
data = np.random.randn(1000)

plt.hist(data, bins=30, density=True, alpha=0.7)
plt.title("正規分布のヒストグラム")
plt.xlabel("値")
plt.ylabel("密度")
plt.show()
```

Listing 4.4　ヒストグラム

4.6.1 練習問題と解答

練習問題

1. $y = x^2$ のグラフを $x = -5$ から 5 の範囲で描きなさい。
2. 5 つの都市の人口を棒グラフで表示しなさい。
3. 正の相関、負の相関、無相関のデータをそれぞれ作成し、散布図で可視化しなさい。
4. 一様分布からサンプルを生成し、ヒストグラムで表示しなさい。

模範解答

1. $y = x^2$ のグラフ

```
x = np.linspace(-5, 5, 100)
y = x ** 2

plt.plot(x, y)
plt.title("y = x^2")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.show()
```

2. 都市の人口

```
cities = ['東京', '大阪', '名古屋', '札幌', '福岡']
population = [1400, 884, 233, 197, 162]

plt.bar(cities, population)
plt.title("都市別人口（万人）")
plt.ylabel("人口（万人）")
plt.show()
```

3. 相関の可視化

```
np.random.seed(42)
x = np.random.randn(50)

# 正の相関
y_pos = x + np.random.randn(50) * 0.3

# 負の相関
y_neg = -x + np.random.randn(50) * 0.3

# 無相関
y_none = np.random.randn(50)

fig, axes = plt.subplots(1, 3, figsize=(12, 4))
```

```
axes[0].scatter(x, y_pos)
axes[0].set_title("正の相関")
axes[1].scatter(x, y_neg)
axes[1].set_title("負の相関")
axes[2].scatter(x, y_none)
axes[2].set_title("無相関")
plt.tight_layout()
plt.show()
```

4. 一様分布のヒストグラム

```
data = np.random.uniform(0, 10, 1000)

plt.hist(data, bins=20, density=True, alpha=0.7)
plt.title("一様分布のヒストグラム")
plt.xlabel("値")
plt.ylabel("密度")
plt.show()
```

第 V 部

scipy.stats による統計分析

第 5 章

scipy.stats 初級

5.1 scipy.stats とは

scipy.stats は、Python で統計計算を行うための包括的なモジュールです。

5.1.1 狙い

本チュートリアルでは、基本統計量の計算、確率分布の扱い、乱数生成を習得します。

5.2 基本統計量

```python
from scipy import stats
import numpy as np

data = np.random.normal(loc=65, scale=15, size=100)

result = stats.describe(data)
print(f"データ数: {result.nobs}")
print(f"平均: {result.mean:.2f}")
print(f"分散: {result.variance:.2f}")
print(f"歪度: {result.skewness:.2f}")
print(f"尖度: {result.kurtosis:.2f}")
```

Listing 5.1　基本統計量

重要ポイント

歪度と尖度：

- **歪度**（Skewness）：分布の非対称性（正→右に裾が長い）
- **尖度**（Kurtosis）：分布の尖り具合（正→正規分布より尖っている）

5.3 正規分布

```python
from scipy import stats

# 標準正規分布
norm_dist = stats.norm(loc=0, scale=1)

# 確率密度関数 (PDF)
print(f"f(0) = {norm_dist.pdf(0):.4f}")

# 累積分布関数 (CDF)
print(f"P(X <= 1.96) = {norm_dist.cdf(1.96):.4f}")

# パーセント点関数 (逆CDF)
print(f"95%点 = {norm_dist.ppf(0.95):.4f}")

# 乱数生成
samples = norm_dist.rvs(size=100)
```

Listing 5.2 正規分布の操作

5.4 二項分布とポアソン分布

```python
# 二項分布: n=10, p=0.5
binom_dist = stats.binom(n=10, p=0.5)
print(f"P(X=5): {binom_dist.pmf(5):.4f}")

# ポアソン分布: lambda=5
poisson_dist = stats.poisson(mu=5)
print(f"P(X=3): {poisson_dist.pmf(3):.4f}")
```

Listing 5.3 離散分布

5.4.1 練習問題と解答

練習問題

1. 平均 170cm、標準偏差 6cm の身長分布で、180cm 以上の人の割合を計算しなさい。
2. 上記の分布で、上位 10% に入るための身長を計算しなさい。
3. サイコロを 10 回振って、6 が出る回数の分布を二項分布で表しなさい ($p = 1/6$)。6 が 2 回以上出る確率を求めなさい。
4. 1 時間に平均 3 件の注文が来るとき、5 件以上注文が来る確率をポアソン分布で計算しなさい。

模範解答

1. 180cm 以上の割合

```python
from scipy import stats

height_dist = stats.norm(loc=170, scale=6)
prob = 1 - height_dist.cdf(180)
print(f"180cm以上の割合: {prob*100:.2f}%")
# 出力: 180cm以上の割合: 4.78%
```

2. 上位 10% の身長

```python
height_90 = height_dist.ppf(0.90)
print(f"上位10%の身長: {height_90:.2f}cm")
# 出力: 上位10%の身長: 177.69cm
```

3. サイコロの二項分布

```python
dice_dist = stats.binom(n=10, p=1/6)

# 6が2回以上出る確率 = 1 - P(0回) - P(1回)
prob = 1 - dice_dist.cdf(1)
print(f"6が2回以上出る確率: {prob*100:.2f}%")
# 出力: 6が2回以上出る確率: 51.54%
```

4. ポアソン分布

```python
order_dist = stats.poisson(mu=3)

# 5件以上 = 1 - P(4件以下)
prob = 1 - order_dist.cdf(4)
print(f"5件以上の確率: {prob*100:.2f}%")
# 出力: 5件以上の確率: 18.47%
```

第 VI 部

scikit-learn による機械学習

第 6 章

scikit-learn 初級

6.1 scikit-learn とは

scikit-learn は、Python の機械学習ライブラリです。

6.1.1 狙い

本チュートリアルでは、データ前処理、線形回帰、ロジスティック回帰、決定木、モデル評価の基礎を習得します。

重要ポイント

scikit-learn の一貫した API：

- `fit()`: モデルの学習
- `predict()`: 予測
- `score()`: 評価

すべてのモデルが同じインターフェースを持つため、学習コストが低い。

6.2 基本的なワークフロー

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_iris

# 1. データの読み込みと分割
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target, test_size=0.3, random_state=42)

# 2. モデルの作成と学習
```

```
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# 3. 予測
y_pred = model.predict(X_test)

# 4. 評価
accuracy = accuracy_score(y_test, y_pred)
print(f"正解率: {accuracy:.4f}")
```

Listing 6.1　機械学習の基本ワークフロー

6.3　データの前処理

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)  # fit はしない!
```

Listing 6.2　特徴量の標準化

注意

テストデータには transform() のみを使用し、fit_transform() は使わないでください。訓練データの統計量でテストデータを変換することで、データリークを防ぎます。

6.4　モデルの評価

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, iris.data, iris.target, cv=5)
print(f"各フォールド: {scores}")
print(f"平均スコア: {scores.mean():.4f}")
print(f"標準偏差: {scores.std():.4f}")
```

Listing 6.3　交差検証

6.5　決定木

```
from sklearn.tree import DecisionTreeClassifier

tree_model = DecisionTreeClassifier(max_depth=3, random_state=42)
tree_model.fit(X_train, y_train)
```

```

# 特徴量の重要度
importance = tree_model.feature_importances_
for name, imp in zip(iris.feature_names, importance):
    print(f"{name}: {imp:.4f}")
```

Listing 6.4　決定木分類器

6.5.1　練習問題と解答

練習問題

1. iris データセットを訓練データ（70%）とテストデータ（30%）に分割しなさい。
2. 特徴量を標準化し、ロジスティック回帰で分類しなさい。
3. 5 分割交差検証でモデルを評価しなさい。
4. 決定木分類器を学習させ、特徴量の重要度を棒グラフで可視化しなさい。

模範解答

1. データ分割

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
    iris.data, iris.target,
    test_size=0.3, random_state=42
)
print(f"訓練データ: {X_train.shape}")
print(f"テストデータ: {X_test.shape}")
```

2. 標準化とロジスティック回帰

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LogisticRegression(max_iter=200)
model.fit(X_train_scaled, y_train)
print(f"正解率: {model.score(X_test_scaled, y_test):.4f}")
```

3. 交差検証

```python
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model, iris.data, iris.target, cv=5)
print(f"各フォールド: {scores}")
print(f"平均: {scores.mean():.4f} (+/- {scores.std():.4f})")
```

4. 決定木と特徴量重要度

```python
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

tree = DecisionTreeClassifier(max_depth=5, random_state=42)
tree.fit(X_train, y_train)

importance = tree.feature_importances_
plt.barh(iris.feature_names, importance)
plt.xlabel("重要度")
plt.title("特徴量の重要度")
plt.show()
```

付録 A

推奨学習パス

A.1　プログラミング初心者

1. JupyterLite 入門（基本操作に慣れる）
2. NumPy 初級（配列計算の基礎）
3. pandas 初級（データ操作の基礎）
4. matplotlib 初級（グラフ作成の基礎）

A.2　データ分析を学びたい方

1. pandas 初級 → pandas 中級
2. scipy.stats 初級（統計の基礎）
3. seaborn 初級（統計的可視化）
4. statsmodels チュートリアル（回帰分析）

A.3　機械学習を学びたい方

1. NumPy 初級 → NumPy 中級
2. pandas 初級 → pandas 中級
3. scikit-learn 初級 → scikit-learn 中級

付録 B

よく使う関数一覧

B.1 NumPy

関数	説明
`np.array()`	配列の作成
`np.zeros()`	ゼロ配列の作成
`np.ones()`	1 の配列の作成
`np.arange()`	等差数列の作成
`np.linspace()`	等間隔数列の作成
`np.mean()`	平均
`np.std()`	標準偏差
`np.sum()`	合計
`np.max()`, `np.min()`	最大値・最小値

B.2 pandas

メソッド	説明
`pd.DataFrame()`	DataFrame の作成
`pd.read_csv()`	CSV の読み込み
`df.head()`	先頭行の表示
`df.info()`	情報の表示
`df.describe()`	統計量の表示
`df.loc[]`	ラベルでの選択
`df.iloc[]`	位置での選択
`df.sort_values()`	並べ替え

メソッド	説明
`df.groupby()`	グループ化

B.3 matplotlib

関数	説明
`plt.plot()`	折れ線グラフ
`plt.scatter()`	散布図
`plt.bar()`	棒グラフ
`plt.hist()`	ヒストグラム
`plt.pie()`	円グラフ
`plt.title()`	タイトル
`plt.xlabel()`, `plt.ylabel()`	軸ラベル
`plt.legend()`	凡例
`plt.savefig()`	保存

B.4 scipy.stats

関数/メソッド	説明
`stats.describe()`	記述統計量
`stats.norm`	正規分布
`stats.binom`	二項分布
`stats.poisson`	ポアソン分布
`.pdf(x)`	確率密度関数
`.cdf(x)`	累積分布関数
`.ppf(q)`	パーセント点関数
`.rvs(size)`	乱数生成

B.5 scikit-learn

クラス/関数	説明
`train_test_split()`	データ分割
`StandardScaler`	標準化

クラス/関数	説明
`LogisticRegression`	ロジスティック回帰
`LinearRegression`	線形回帰
`DecisionTreeClassifier`	決定木分類器
`cross_val_score()`	交差検証
`accuracy_score()`	正解率
`confusion_matrix()`	混同行列