

# Python活用基礎研修: 応用編2

## AI（機械学習）入門

# Python活用基礎研修について

## 研修の目標

- Pythonを通してプログラミングの基礎を学ぶ.
- 実際に手を動かして簡単なデータ解析およびAI（機械学習）を体験する.

## 入門編

- 入門編1: Pythonプログラミングの基本的なルールを学ぶ.
- 入門編2: NumPyを用いたプログラミングを学ぶ.

## 応用編: 各種モジュール（=便利なツール）の使い方を学ぶ

- 応用編1（データ解析）: pandas, Matplotlib
- 応用編2（AI, 機械学習）: **scikit-learn**

# 研修の進行速度および理解度把握のためのご協力をお願い

本日の研修では、研修の進行速度が適切かを把握するために、Zoomの**リアクション機能**を用いたレスポンスをお願いすることがございます。研修中にこちらから進行速度が早すぎないか質問した際に、**リアクション機能**の絵文字で**進行速度**や**理解度**に対する感想をお願いいたします。

## リアクションの例

- 👍, 👏 : ちょうど良い または 理解できている
- 😬, 🤔 : すこし早い または 少し難しい
- 😓, 🥲 : もうすこしゆっくりが良い または 前の説明範囲で躓いている

# 0. matplotlibで日本語のキャプションを使う準備

matplotlibには日本語フォントがないため、キャプションなどに日本語が使えない。  
それを解決する `japanize_matplotlib` モジュールを利用する。

対話型開発環境（JupyterLabやJupyter Notebook）では、  
コードセルで以下のように書くと開発環境にないモジュールを新たにインストールできる。

```
!pip3 install japanize_matplotlib
```

`import` した後からは、matplotlibのグラフ上で日本語が使用できる。

```
import japanize_matplotlib
```

# 前回の1.6. 練習問題の答え

DataFrame `titanic_df` について,

1. 全ての列を含む, 最初から10行だけ抽出せよ.
2. `Survived`, `Pclass`, `Sex` の列のみを含む, 最初から10行だけ抽出せよ.
3. `PassengerId` が `300` である1行のみを抽出せよ.

## 解答例

```
titanic_df.head(10)    # head関数
titanic_df.loc[0:9]     # loc
titanic_df.iloc[0:10]  # iloc
titanic_df[0:10]       # []
```

# 前回の1.6. 練習問題の答え

DataFrame `titanic_df` について,

1. 全ての列を含む, 最初から10行だけ抽出せよ.
2. `Survived`, `Pclass`, `Sex` の列のみを含む, 最初から10行だけ抽出せよ.
3. `PassengerId` が `300` である1行のみを抽出せよ.

## 解答例

```
titanic_df.head(10)[['Survived', 'Pclass', 'Sex']] # head関数 + []
titanic_df.loc[0:9, ['Survived', 'Pclass', 'Sex']] # loc
titanic_df.iloc[0:10, [1, 2, 4]]                  # iloc
titanic_df[0:10][['Survived', 'Pclass', 'Sex']]   # [] + []
```

# 前回の1.6. 練習問題の答え

DataFrame `titanic_df` について,

1. 全ての列を含む, 最初から10行だけ抽出せよ.
2. `Survived`, `Pclass`, `Sex` の列のみを含む, 最初から10行だけ抽出せよ.
3. `PassengerId` が `300` である1行のみを抽出せよ.

## 解答例

```
titanic_df.loc[[299]] # loc
titanic_df.iloc[[299]] # iloc
titanic_df[299:300] # []
```

# 前回の2.5. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )について,

1. DataFrameとして読み込め.

```
# 1. データの読み込み
```

```
drill_2020_csv_file_path = './data/toyonaka_temp_2020.csv'  
drill_2020_df = pd.read_csv(drill_2020_csv_file_path)
```



# 前回の2.5. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )について,

2. 読み込んだDataFrameについて, 降水量の折れ線グラフを描画せよ.

## 解答例

```
# pandasの場合
drill_2020_df.plot(x='月', y='降水量')
# matplotlibの場合
fig, axes = plt.subplots()
axes.plot(drill_2020_df['月'], drill_2020_df['降水量'])
```

### ヒント: 折れ線グラフの関数

Matplotlib: `Axes.plot(data_x, data_y)` # 引数に描画したい各軸の列を入れる.

pandas: `DataFrame.plot(x='列名', y='列名')` # 引数x,yに描画する各軸の列を入れる.

## 前回の3.8. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )

および2021年の豊中市の気温と降水量のデータ( `toyonaka_temp_2021.xlsx` )について,

1. それぞれをDataFrameとして読み込め.

### 解答例

```
# csvファイルをDataFrameとして読み込む
drill_2020_csv_file_path = './data/toyonaka_temp_2020.csv'
drill_2020_df = pd.read_csv(drill_2020_csv_file_path)

# excelファイルをDataFrameとして読み込む
drill_2021_excel_file_path = './data/toyonaka_temp_2021.xlsx'
drill_2021_df = pd.read_excel(drill_2021_excel_file_path)
```

# 前回の3.8. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )

および2021年の豊中市の気温と降水量のデータ( `toyonaka_temp_2021.xlsx` )について,

2. 降雨量の折れ線グラフをそれぞれのDataFrameで描画せよ.

**解答例: pandasの場合**

```
drill_2020_df.plot('月', '降水量')
```

**解答例: matplotlibの場合**

```
fig, axes = plt.subplots()  
axes.plot(drill_2020_df['月'], drill_2020_df['降水量'])
```

# 前回の3.8. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )

および2021年の豊中市の気温と降水量のデータ( `toyonaka_temp_2021.xlsx` )について,

3. 気温の棒グラフをそれぞれのDataFrameで描画せよ.

**解答例: pandasの場合**

```
drill_2020_df.plot.bar('月', '気温')
```

**解答例: matplotlibの場合**

```
fig, axes = plt.subplots()
axes.bar(drill_2020_df['月'], drill_2020_df['気温'])
```

## 前回の3.8. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )

および2021年の豊中市の気温と降水量のデータ( `toyonaka_temp_2021.xlsx` )について,

4. 気温と降雨量の相関をそれぞれのDataFrameで求めよ.

```
drill_2020_df.corr()
```

## 前回の3.8. 練習問題の答え

2020年の豊中市の気温と降水量のデータ( `toyonaka_temp_2020.csv` )

および2021年の豊中市の気温と降水量のデータ( `toyonaka_temp_2021.xlsx` )について,

5. 各DataFrameの気温と降雨量の散布図を1枚の画像で描画せよ.

### 解答例

```
# matplotlibのみの場合
fig, axes = plt.subplots(ncols=2, figsize=(16, 8))
axes[0].scatter(drill_2020_df['気温'], drill_2020_df['降水量'])
axes[1].scatter(drill_2021_df['気温'], drill_2021_df['降水量'])

# matplotlibとpandasを組み合わせた場合
fig, axes = plt.subplots(ncols=2, figsize=(16, 8))
drill_2020_df.plot.scatter(x='気温', y='降水量', ax=axes[0])
drill_2021_df.plot.scatter(x='気温', y='降水量', ax=axes[1])
```

## 1. AIと機械学習

1.1 機械学習とは

1.2 教師なし学習と教師あり学習

1.3 機械学習の主なタスク

1.4 機械学習の手順

1.5 scikit-learn

2. 教師なし学習：クラスタリング

3. 教師あり学習：分類

4. 教師あり学習：回帰

# 1.1 AIと機械学習

## 人工知能（Artificial Intelligence; AI）の定義はあいまい

**AI**は, 人間の知的能力をコンピュータ上で実現する様々な技術またはアルゴリズムのこと. 例えば, 下記のような視点によってAIは分類される.

- AIの知性の持つという視点: **強いAI**と**弱いAI**
- AIの解決の能力という視点: **汎化型AI**と**特化型AI**

**機械学習**は, 弱いAIかつ特化型AIと分類でき, 機械学習の一分野に深層学習がある.

- **弱いAI**: 人間のような心や意識を持たないAI
- **特化型AI**: 特定の問題のみを解決できる能力を持ったAI



# 1.2 教師なし学習と教師あり学習

## 教師なし学習 (Unsupervised Learning)

教師なし学習は, 与えられたデータの構造や法則をモデル化する手法.

## 教師あり学習 (Supervised Learning)

教師あり学習は, **説明変数**から**目的変数**を推測するモデルを学習する手法.

**目的変数**: 推測したい変数. 学習を目的にデータに付与された変数を**ラベル**と呼ぶ.

**説明変数**: 目的変数に影響を与えている変数.

例えばBMI値は, 肥満度を目的変数, 身長と体重を説明変数とした線形回帰モデル.

# 1.3 機械学習の主なタスク

## 2. 教師なし学習：クラスタリング

データ間の類似性を定義または学習することで、与えられたデータをいくつかのクラスター（＝データのグループ）に分割するタスク。

## 3. 教師あり学習：分類

説明変数に基づいて推測したい目的変数が**質的変数**のタスク。

質的変数：種類を区別する変数（性別, 血液型, チームの順位など）

## 4. 教師あり学習：回帰

説明変数に基づいて推測したい目的変数が**量的変数**のタスク。

量的変数：数値や量で測ることができる変数（年齢, 身長, テストの点数など）

# 1.4 機械学習の手順

多くの場合、機械学習は下記の手順でモデルの構築を目指す。

ただし、**状況に応じて手順は前後するため本手順は参考程度にとどめること。**

(例えば, 3. 課題の検討をしてから1. データを収集するなど.)

1. データの収集
2. データの確認・可視化
3. 課題の検討
4. 機械学習の手法の選択
5. データの前処理
6. モデルの学習
7. 学習結果の確認
8. パラメータや前処理の再検討

# 1.5 scikit-learn

**scikit-learn**は、機械学習における一連の手順をサポートするモジュールである。  
モデル学習だけでなく、データの前処理やモデル評価など関連する数多くの機能を実装。

## 利用する準備

scikit-learnは、機能毎に `import` することが多い。

下記のように、モデル、前処理、評価など、それぞれ利用したい機能毎に `import` する。

```
from sklearn.linear_model import LinearRegression # モデルに関するimport
from sklearn.preprocessing import StandardScaler # 前処理に関するimport
from sklearn.metrics import confusion_matrix # 評価に関するimport
```

1. AIと機械学習
2. 教師なし学習：クラスタリング
  - 2.1 データの収集
  - 2.2 データの確認・可視化
  - 2.3 課題の検討, 機械学習の手法の選択
  - 2.4 データの前処理, モデルの学習
  - 2.5 学習結果の確認
  - 2.6 パラメータや前処理の再検討（練習問題）
3. 教師あり学習：分類
4. 教師あり学習：回帰

## 2.1 データの収集

### Fisher's Iris Dataset

機械学習や統計学の世界において最も有名なデータセットのひとつ.

Iris (=あやめ) の花について, その花びらやがく片を測定した数値が記録されている.

元となったデータセットの配布元は以下のとおり

URL: <https://archive.ics.uci.edu/dataset/53/iris>

```
import pandas as pd
iris_df = pd.read_csv('./data/iris_ja.csv')
iris_df.head()
```

## 2.2 データの確認・可視化

`info` 関数を用いてデータの概要を取得し、データの種類や欠損値の有無を確認.

```
iris_df.info()
```

欠損値がなく、全ての数値が連続値だったので、`corr` 関数を用いて各列間の相関を確認.

```
iris_df.corr()
```

最も相関係数が高かった花びらの長さ と 花びらの幅 について散布図を作成.

```
iris_df.plot.scatter(x='花びらの長さ', y='花びらの幅')
```

## 2.3 課題の検討, 機械学習の手法の選択

### 課題の検討

相関係数と散布図の分析から, 「花びらの長さと花びらの幅を用いることで, データセットを2つのクラスター (=グループ) に分割できる」と仮説を立てられる.

### 機械学習の手法の選択

そこで, **N** 個のデータを指定した **k** 個のクラスターに分ける教師なし学習として, 最も一般的な手法の一つである**k-means** (=k-平均法) を用いる.

#### 機械学習手法の選択

適切な機械学習手法は, 課題の内容やデータセットの状況によって変換する.  
scikit-learnでは適切な手法選択のためのチャートを公開している.

(URL: [https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/](https://scikit-learn.org/stable/tutorial/machine_learning_map/))



## 2.3 課題の検討, 機械学習の手法の選択

**k-means**は, 類似データは集合する, という仮説に基づいてデータを **k** 個に分割する手法. 類似データの集合がクラスターであり, クラスターの代表的な値を重心と呼ぶ. 具体的には, 下記のアルゴリズム (=手順, 算法) によってクラスターを特定する.

### k-meansのアルゴリズム

1. 指定した **k** 個のクラスターの重心の初期値を決定する.
  - **k** 個のデータをランダムに選択
  - **k** 個のランダムな値を指定
2. 各データを, 最も近い重心のクラスターに割り当てる.
3. クラスターの重心を再計算する.
4. **手順3**または**手順2**において変化がなければ終了. 変化があれば**手順2**に戻る.

## 2.4 データの前処理, モデルの学習

Irisデータセットを, 花びらの長さ と 花びらの幅 に基づいて,  
2つのクラスターに分割したいので対応する2列のみを抽出する.

```
iris_train_df = iris_df[['花びらの長さ', '花びらの幅']]
```

`sklearn` モジュールの `KMeans` 関数は, 初期化されたk-meansモデルを構築する.  
引数 `n_clusters` を用いてクラスター数 `k` を指定する. (デフォルトは `k=8`)

```
from sklearn.cluster import KMeans  
kmeans_model = KMeans(n_clusters=2)
```

変数 `kmeans_model` の `fit` 関数を用いて学習.

以降は, 学習済みの変数 `kmeans_model` を操作することで学習結果を得る.

```
kmeans_model.fit(iris_train_df)
```

## 2.5 学習結果の確認

学習済みモデルの `predict` 関数は、引数で与えられたデータに対する処理結果を返す。  
k-meansモデルの場合は、学習した重心に基づいてデータのクラスターを予測する。

```
result_labels = kmeans_model.predict(iris_train_df)
```

k-meansモデルが学習した各クラスターの重心を確認。

```
kmeans_model.cluster_centers_
```

## 2.5 学習結果の確認

k-meansのクラスター分割結果を `seaborn` モジュールを用いて散布図で可視化する。

**seaborn**モジュールとその他の可視化モジュール

**seaborn**は, matplotlibをベースにした可視化モジュール.

特に統計分析に特化した高水準の機能が充実している.

その他の可視化モジュールにはBokeh, Plotly, Mayaviなどがある.

`seaborn` の各関数は, 引数 `data` で `DataFrame` の変数を受け取ることができる.

下記の場合は, `data` に `iris_train_df` を渡すことで, 列名でx軸とy軸を指定している.

また, 引数 `hue` にクラスターの分類結果を渡すことで, クラスターを色で表現している.

```
import seaborn as sns # seabornモジュールをsnsとしてimport
sns.scatterplot(x='花びらの長さ', y='花びらの幅', hue=result_labels, data=iris_train_df)
```

## 2.6 パラメータや前処理の再検討（練習問題）

Irisデータセット `iris_df` について,

1. 入力変数を `がく片の長さ` および `がく片の幅` とし, `k=2` で `k-means` を実行せよ.
2. 上記1. の分割結果について散布図で可視化せよ.
3. **発展問題:** 上記1. のモデルを**混合正規分布モデル**に変更し学習および可視化せよ.

```
# GaussianMixture (=混合正規分布モデル) をimport
from sklearn.mixture import GaussianMixture
# モデルを構築. n_componentsは, k-meansにおける分割数kと同じイメージ
gm_model = GaussianMixture(n_components=2)
```

# 概要

1. AIと機械学習
2. 教師なし学習：クラスタリング
3. **教師あり学習：分類**
  - 3.1 データの収集
  - 3.2 データの確認・可視化
  - 3.3 課題の検討, 機械学習の手法の選択
  - 3.4 データの前処理
  - 3.5 モデルの学習
  - 3.6 学習結果の確認
  - 3.7 パラメータや前処理の再検討（練習問題）
4. 教師あり学習：回帰

## 3.1 データの収集

### (真) Fisher's Iris Dataset

機械学習や統計学の世界において最も有名なデータセットのひとつ.

Iris (=あやめ) の花について, その花びらやがく片を測定した数値が記録されている.  
また, **人手により分類されたあやめの種類**も記録されている.

```
iris_labeled_df = pd.read_csv('./data/iris_labeled_ja.csv')  
iris_labeled_df.head()
```

## 3.2 データの確認・可視化

`info` 関数を用いてデータの概要を取得し, データの種類や欠損値の有無を確認.  
あやめの種類が `setosa`, `versicolor` および `virginica` の3種類であることがわかる.

```
iris_labeled_df.info()
```

`seaborn` モジュールの `pairplot` 関数を用いてデータセットを可視化する.

`pairplot` 関数は, 数値列について散布図行列 (列間の散布図と各列の分布) を生成する.

```
sns.pairplot(data=iris_labeled_df, hue='あやめの種類')
```



## 3.3 課題の検討, 機械学習の手法の選択

### 課題の検討

散布図行列の分析から以下の仮説を立てられる.

「花びらの長さと花びらの幅を用いることで, データからあやめの種類を分類できる。」

### 機械学習の手法の選択

そこで, 木構造を用いて分類・回帰を行う教師あり学習として,  
最も一般的な手法の一つである**Decision Tree** (=決定木) を用いる.

**Decision Tree**は, 分類問題を解く**分類木**および回帰問題を解く**回帰木**の総称.

木構造を用いた分類方法の最も単純なイメージは二値分類.

花びらの長さは10cm以下?

-Yes-> 花びらの幅は5cm未満? -Yes/No-> ... -> あやめの種類は**versicolor**

-No -> 花びらの長さは3cm以上? -Yes/No-> ... -> あやめの種類は**setosa**

## 3.4 データの前処理

Irisデータセットについて, 花びらの長さや花びらの幅に基づいてあやめの種類を分類したいので, まず**説明変数として対応する2列**のみを抽出する. また分類の対象である**目的としてあやめの種類**を抽出する.

```
iris_labeled_X_df = iris_labeled_df[['花びらの長さ', '花びらの幅']]
iris_labeled_y_df = iris_labeled_df[['あやめの種類']]
```

`train_test_split` 関数を用いて, 学習データ:評価データ = 4:1に分割する. 引数 `test_size` に評価データに使用したい割合を渡す (1.0 = 100%とする). 引数 `shuffle=True` とすることで, 分割時に行の順番をシャッフルしてくれる.

```
from sklearn.model_selection import train_test_split
iris_labeled_X_train_df, iris_labeled_X_test_df, iris_labeled_y_train_df, iris_labeled_y_test_df \
    = train_test_split(iris_labeled_X_df, iris_labeled_y_df, test_size=0.2, shuffle=True)
```

# データセットを分割する理由

教師あり学習において、データセットは以下の3種類に分割されることが多い。  
これは、機械学習の目的が「**既知のデータ**から構築したモデルを用いて、**未知のデータ**に対して正しく分類や回帰を予想すること」であるため。

- 学習データ：モデルを学習するための**既知のデータ**。
- 検証データ：モデルを検証・調整するための**擬似的な未知のデータ**。
- 評価データ：モデルの評価するための**完全な未知のデータ**。

モデルの調整を誤ると**学習データ**の正答率が100%、未知データは正解率が0%のような機械学習の目的に合わないモデルが構築される可能性がある。

上記の事態を避けるために、**擬似的な未知データの検証データ**でモデルを検証・調整し、最終的に、モデルが正しく学習できたかを**未知のデータである評価データ**で確認する。

## 3.5 モデルの学習

`sklearn` モジュールの `DecisionTreeClassifier` 関数は, 初期化された分類木を構築する.

```
from sklearn.tree import DecisionTreeClassifier
dtree_model = DecisionTreeClassifier()
```

分類木の変数 `dtree_model` について, 説明変数の学習データ `iris_labeled_X_train_df` および目的変数の学習データ `iris_labeled_y_train_df` を用いて `fit` 関数で学習. 以降は, 学習済みの変数 `dtree_model` を操作することで学習結果を得る.

```
dtree_model.fit(iris_labeled_X_train_df, iris_labeled_y_train_df)
```

## 3.6 学習結果の確認

学習済みモデルの `predict` 関数は、引数で与えられたデータに対する処理結果を返す。分類木の場合は、学習した木構造に基づいて説明変数に対応する目的変数を入力する。学習に用いていない、モデルにとって未知のデータ `iris_labeled_X_test_df` を入力し、学習済みの変数 `dtree_model` の分類結果 `iris_labeled_y_pred_np` を取得。

```
iris_labeled_y_pred_np = dtree_model.predict(iris_labeled_X_test_df)
```

`plot_tree` 関数を用いて学習した分類木を描画。

```
import sklearn.tree
unique_labels = iris_labeled_df['あやめの種類'].unique()
plt.figure(figsize=(8, 8))
sklearn.tree.plot_tree(dtree_model, class_names=unique_labels, filled=True)
```

## 3.6 学習結果の確認

評価データを用いることで構築したモデルの性能を評価できる.  
scikit-learnには様々な評価関数が用意されている.

```
from sklearn.metrics import accuracy_score, classification_report
# 分類結果の正解率を算出
accuracy_score(iris_labeled_y_test_df, iris_labeled_y_pred_np)
# 分類結果の適合率, 再現率, F1スコア, 正解率, マクロ平均, マイクロ平均を算出
print(classification_report(iris_labeled_y_test_df, iris_labeled_y_pred_np))
```

**混同行列** (Confusion Matrix) は評価データとモデルの分類結果について,  
正しく分類したデータの個数および誤って他に分類されたデータの個数をまとめた表.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
dtree_matrix = confusion_matrix(iris_labeled_y_test_df, iris_labeled_y_pred_np)
ConfusionMatrixDisplay(dtree_matrix, display_labels=unique_labels).plot()
```

## 3.7 パラメータや前処理の再検討（練習問題）

(真) Irisデータセット `iris_labeled_df` について,

1. 説明変数を `がく片の長さ` および `がく片の幅` とし, 分類木を新たに学習せよ.
2. 上記1. の分類結果について散布図で可視化せよ.
3. **発展問題:** 様々な評価関数を用いてモデルの性能を自由に検証せよ.

1. AIと機械学習
2. 教師なし学習：クラスタリング
3. 教師あり学習：分類
4. **教師あり学習：回帰**
  - 4.1 データの収集
  - 4.2 データの確認・可視化
  - 4.3 課題の検討, 機械学習の手法の選択
  - 4.4 データの前処理
  - 4.5 モデルの学習
  - 4.6 学習結果の確認
  - 4.7 パラメータや前処理の再検討（練習問題）



# 4.1 データの取得

## California Housing

機械学習や統計学の世界において有名なデータセットの一つ.

1990年の米国国勢調査から得られたカリフォルニア州の住宅情報が記録されている.

元となったデータセットの配布元は以下のとおり

URL: <https://lib.stat.cmu.edu/datasets/houses.zip>

```
housing_df = pd.read_csv('./data/california_housing.csv')
```

## 4.2 データの確認・可視化

`info` 関数を用いてデータの概要を取得し, データの種類や欠損値の有無を確認.  
MedInc (所得の中央値) や AveRooms (部屋数の平均値) など列を9つ含むことがわかる.

```
housing_df.info()
```

`corr` 関数を用いて各列間の相関を算出し,  
`seaborn` モジュールの `heatmap` 関数を用いてデータセットを可視化する.

```
sns.heatmap(housing_df.corr(), annot=True, fmt='.2f', cmap='RdBu')
```

特に, MedInc (所得の中央値) と HousingPrices (住宅価格の中央値) の相関が高い.

### 注意点

HousingPrices と MedInc の単位が異なる点に注意.

HousingPrices の単位は10万ドル, MedInc の単位は1万ドル.

## 4.3 課題の検討, 機械学習の手法の選択

### 課題の検討

相関係数のヒートマップを用いた分析から以下の仮説を立てられる.

「**MedInc**を用いることで, **HousingPrices**を予測できる.

### 機械学習の手法の選択

そこで, 説明変数と目的変数の関係をモデル化する教師あり学習として, 最も単純な手法の一つであるLinear Regression (線形回帰) を用いる.

## 4.4 データの前処理

California Housingデータセットについて, MedInc（所得の中央値）に基づいて HousingPrices（住宅価格の中央値）を予測したいので, まず**説明変数としてMedInc**のみを抽出する.  
また予測の対象である**目的変数としてHousingPrices**を抽出する.

```
housing_X_df = housing_df[['MedInc']]  
housing_y_df = housing_df[['HousingPrices']]
```

`train_test_split` 関数を用いて, 学習データ:評価データ = 4:1に分割する.  
引数 `test_size` に評価データに使用したい割合を渡す (1.0 = 100%とする) .  
引数 `shuffle=True` とすることで, 分割時に行の順番をシャッフルしてくれる.

```
housing_X_train_df, housing_X_test_df, housing_y_train_df, housing_y_test_df \  
= train_test_split(housing_X_df, housing_y_df, test_size=0.2, shuffle=True)
```

## 4.5 モデルの学習

`LinearRegression` 関数は, 初期化された線形回帰モデルを構築する.

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
```

線形回帰モデルの変数 `lr_model` について, 説明変数の学習データ `housing_X_train_df` および目的変数の学習データ `housing_y_train_df` を用いて `fit` 関数で学習.  
以降は, 学習済みの変数 `lr_model` を操作することで学習結果を得る.

```
lr_model.fit(housing_X_train_df, housing_y_train_df)
```

## 4.6 学習結果の確認

学習済みモデルの `predict` 関数は、引数で与えられたデータに対する処理結果を返す。線形回帰では、学習した重みと切片に基づいて説明変数に対応する目的変数を入力する。学習に用いていない、モデルにとって未知のデータ `housing_X_test_df` を入力し、学習済みの変数 `lr_model` の予測結果 `housing_y_pred_np` を取得。

```
housing_y_pred_np = lr_model.predict(housing_X_test_df)
```

評価データを用いることで構築したモデルの性能を評価できる。  
scikit-learnには様々な評価関数が用意されている。

```
from sklearn.metrics import mean_squared_error, r2_score
# 正解値と予測値の平均二乗誤差を算出
mean_squared_error(housing_y_test_df, housing_y_pred_np)
# 決定係数（相関係数の二乗）を算出
r2_score(housing_y_test_df, housing_y_pred_np)
```

## 4.6 学習結果の確認

線形回帰モデル `lr_model` の回帰直線（赤）と評価データの散布図（青）を描画。  
説明変数 `MedInc`（所得の中央値）のみでは、適切なモデルの構築は難しいことがわかる。

```
fig, axes = plt.subplots()
axes.scatter(housing_X_test_df, housing_y_test_df, color='blue', marker='x')
axes.plot(housing_X_train_df, lr_model.predict(housing_X_train_df), color='red')
axes.set_title('Regression Line')
axes.set_xlabel('MedInc')
axes.set_ylabel('HousingPrices')
axes.grid()
plt.show()
```

## 4.7 パラメータや前処理の再検討（練習問題）

Housingデータセット `housing_df` について, 目的変数を `HousingPrices` とするとき,

1. 目的変数と相関の高い2列を説明変数として, 新たな線形回帰モデルを学習せよ.
2. 上記1. 精度を求め, 説明変数が `MedInc` 1つ場合のモデルと性能を比較せよ.
3. 上記1. の説明変数を標準化し, 新たな線形回帰モデルを学習・評価せよ.

### データセットの標準化とは

データセットの平均が0, 分散が1となるように変換する操作のこと.

この操作により, スケールの異なるデータ同士を比較できるようになる.

scikit-learnの `StandardScaler` によって簡単に実現できる.

```
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
standard_scaler.fit(X_train) # 学習データの平均と分散を算出. 未知のデータであるX_testは計算に含めない.
scaled_X_train = standard_scaler.transform(X_train) # 学習データの平均・分散を用いて学習データを標準化
scaled_X_test = standard_scaler.transform(X_test) # "学習データ"の平均・分散を用いて評価データの標準化
```