3. Python とデータ処理の基礎

https://www.kkaneko.jp/cc/enshu2/index.html

金子邦彦

①効率的なデータ解析

②様々な分野に通用するスキル

③スキルアップとキャリアの進展

アウトライン

1. PythonとGoogle Colaboratoryの基本と特徴
2. Python の Pandas データフレーム

3-1. PythonとGoogle Colaboratoryの基本と特徴

Python

- **Python** は多くの人々に利用されている**プログラミング言語**の１つ
- **読みやすさ，書きやすさ，幅広い応用範囲**が特徴

```
from keras.models import Sequentia
model = Sequential()
from keras.layers import Dense, Ac

model.add(Dense(units=64, input_di
model.add(Activation('relu'))
model.add(Dense(units=max(set(y_tr
model.add(Activation('softmax'))
model.compile(loss='sparse_categor
              optimizer='sgd',
              metrics=['accuracy']
model.fit(x_train, y_train, epochs
score=model.evaluate(x_test, y_tes
print(score)
model.predict(x_test)
model.summary()
```

```
poch 1/200
'3 [==============================] - 0s
 3200
 och 2/200
 3 [==============================] - 0s
 200
  ch 3/200
   [==============================] - 0s
  0
```

Python 言語が広く使用されている理由

文法のシンプルさ

- **Python** は，**直感的で読みやすい**文法

拡張性

- **豊富なライブラリとモジュール**を持ち、**多岐にわたる分野で利用**が可能

柔軟性

- シンプルなスクリプトも、高度なプログラムも作成可能

主な機能

- **基本的な機能**: 変数、データ型、条件分岐（if）、繰り返し（for、while）、関数
- **オブジェクト指向**: クラス、スーパークラス、サブクラス、継承
- **モジュールとパッケージ**: 再利用可能なコード
- **例外処理**: エラーを効果的に処理
- **ファイル操作**: ファイルの作成、読み出し、書き込み
- **表示**: テキストでの表示、ビジュアルな表示

Python 言語が広く使用されている理由

文法のシンプルさ

- **Python** は，**直感的で読みやすい**文法
- 例えば、**print** で簡単に**出力**できる、**if** や **else** で**条件分岐**、**for** や **while** で**繰り返し（ループ）**

拡張性

- **多岐にわたる分野で利用**が可能
- 例えば、**関数やクラスを定義**するための **def** や **class**、**継承**や**オブジェクトの属性名と値**を操作するための **super** や **vars** などがある。

柔軟性

- シンプルなスクリプトも、高度なプログラムも作成可能
- **オブジェクト指向**の機能を持ち、**__init__** や **self** のようなキーワードを使用して**クラス**を利用できる。

Python の主なキーワード

- **print**:　出力
- **type**:　型名（クラス名）の取得
- **if**, **else**:　条件分岐
- **for**, **while**:　繰り返し
- **def**:　関数定義
- **return**:　関数の評価値（戻り値）
- **class**:　クラス定義
- **__init__**:　オブジェクトの初期化（コンストラクタ）
- **self**:　クラス内での自オブジェクトへのアクセス
- **vars**:　オブジェクトの属性名と値の取得
- **super**:　親クラス（スーパークラス）へのアクセス

Python の特性

インタープリタ言語

- Pythonはインタープリタ型の言語で、コンパイル不要。
- JavaやC++のようなコンパイル型言語と異なり、プログラムのデバッグが容易。

動的型付け

- Python では、変数を宣言する際にデータ型の指定が不要
- Java や C++ は、変数の宣言時にデータ型の指定が必要である静的片付け言語

多様な標準ライブラリ

- データ処理、言語処理、Web開発、データベース接続など、多岐にわたる領域でのライブラリを持つ

クロスプラットフォーム

- Python は多くの OS で動作する

Python のコーディングスタイル

インデントによるブロックの区切り

- Python はコードブロックを波括弧（{}）ではなく、インデントで区切る
- 字下げは、通常「**タブ**」**または**「**４つの半角スペース**」の**いずれか**を使用。（字下げをするときは、タブとスペースを**混在させない**ように注意してください）

```
if (x > 20):
  print("big")
else:
  print("small")
```

if での字下げ

空白行

- **プログラムを読みやすく**するために、**空白行**を挿入できる

コメント

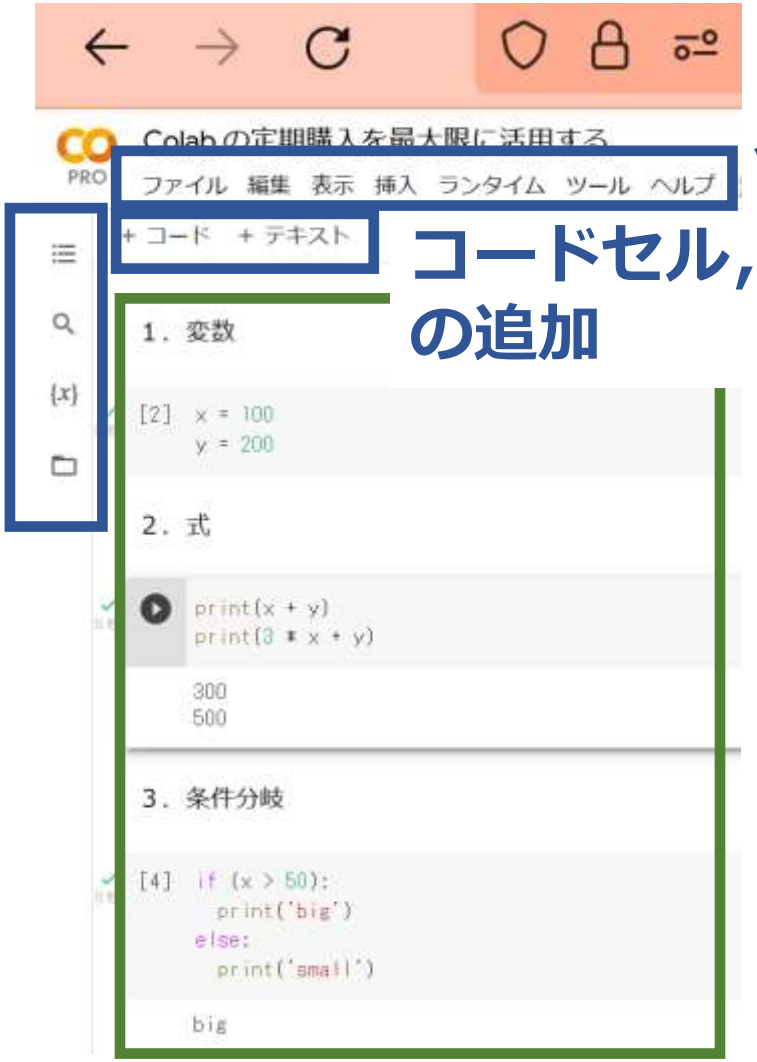
- コメントは 、説明を書いたもの
- コメントは、行の先頭に「#」を記述する。
- **コメント**の**記述内容は自由**である。

Google Colaboratory

URL: https://colab.research.google.com/

- **オンラインで動く**
- **Python のノートブック**の機能を持つ
- Python や種々の機能が**インストール済み**
- **本格的な利用**には，**Google アカウントが必要**

Google Colaboratory の全体画面



メニュー
（目次，検索と置換，変数，ファイル）

メニュー

コードセル，テキストセルの追加

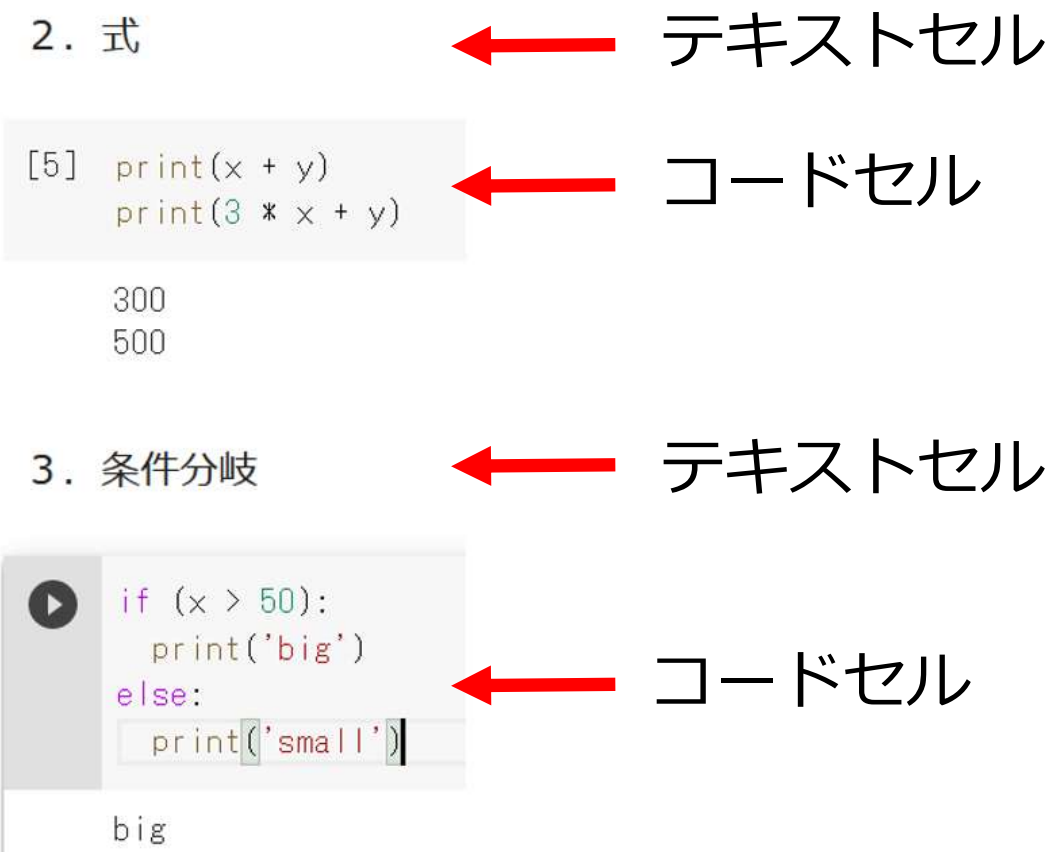
コードセル，テキストセルの並び

Web ブラウザの画面

Google Colaboratory のノートブック

コードセル，**テキストセル**の２種類

- **コードセル**： Python プログラム，コマンド，実行結果
- **テキストセル**：説明文，図

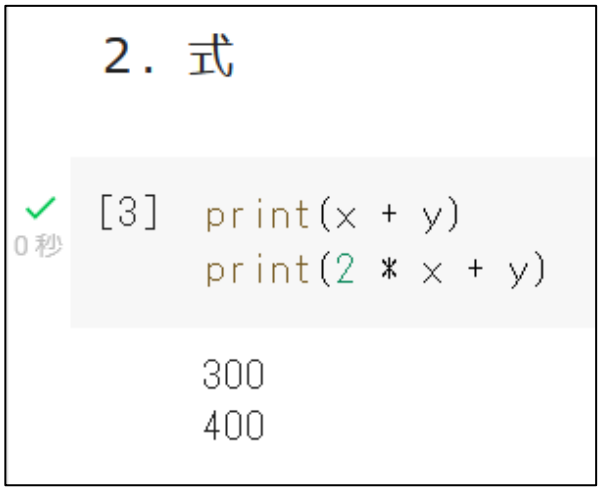


Google Colaboratory の本格的な機能（使用には Google アカウントが必要）

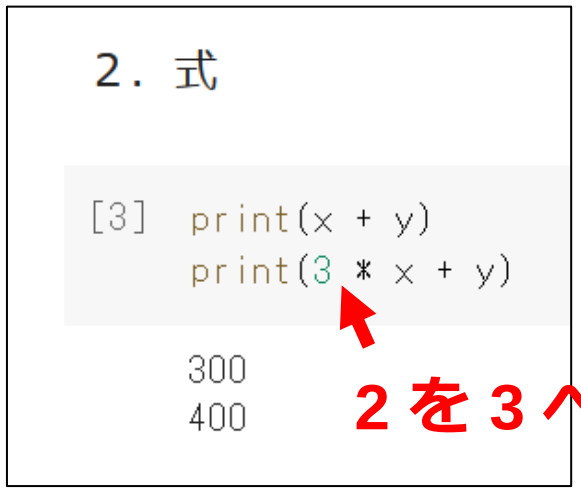
- **ノートブック**の**新規作成，編集，保存，公開**（Google Drive との連携による）
- 公開により，**第三者がノートブックをダウンロードし，編集や実行なども可能**
- **Python プログラム**（コードセル内）の**編集，実行**
- 「!pip」や「%cd」などのシステム操作のための**コマンド**（コードセル内）の**編集，実行**
- **ファイル**の**アップロード**，**ダウンロード**
- ドキュメントの編集（図，リンク，添付ファイルを含めることができる）

コードセルとプログラム実行

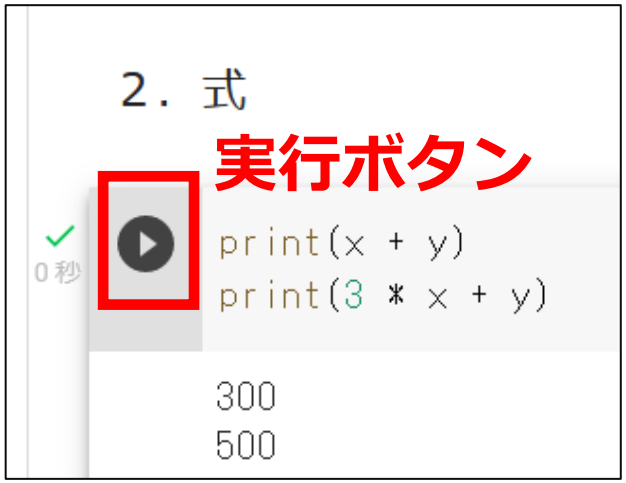
コードセルで，
Python プログラムやコマンドの編集，実行ができる．
（編集や実行には **Google アカウントが必要**）



編集前



編集後



実行ボタンと
実行結果

Google Colaboratory でうまく実行できない場合

混雑しているときなどは、**実行が止まり、再開しない場合**もある

【その対処】

次で、**アクティブなセッションの停止**を行い、その後最初から実行をやり直す

- **メニュー**で「**ランタイム**」，「**セッションの管理**」と操作する.
- **アクティブなセッションの一覧**が表示されるので，「**終了**」をクリックして，**すべてのアクティブなセッションを終了**する.

演習 1

トピックス】

- **Google Colaboratory**
- **変数とデータ型**
- **if、else による条件分岐**
- **for による繰り返し**
- **関数の定義と呼び出し**
- **モジュール**
- **クラスとインスタンス、メソッド**

① Google Colaboratory のページを開く
https://colab.research.google.com/drive/1l2pLl72PYG5TFKKu8ha0h_9fFCafxG4j?usp=sharing

② プログラムや説明や実行結果が表示されるので確認

③ 「１．変数とデータ型」から「４．関数の定義と呼び出し」までをよく読む

プログラムと説明をよく読み、掲載されている実行結果をよく確認。

④ 余裕があれば、次ページ以降の**自習**に取り組む

１．Python 言語が広く使用されている理由

【プログラムの説明】

- datetime モジュールをインポートしている。
- Animal というクラスが定義されており、動物の名前と種類を属性として持つ。
- Animal クラスには speak というメソッドがあり、動物の名前と種類を用いたメッセージを返す。
- display_date という関数は、現在の日時を返す。
- メイン処理では Animal クラスのインスタンスを作成し、そのインスタンスの speak メソッドを呼び出している。
- 現在の日付も表示される。

```
# モジュールのインポート
import datetime

# クラスの定義
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species
    def speak(self):
        return f"I'm {self.name}, a {self.species}!"

# 関数の定義
def display_date():
    return datetime.datetime.now()

# メイン処理
pet = Animal(name="Buddy", species="Dog")
print(pet.speak())
print(f"Today's date is: {display_date()}")
```

```
I'm Buddy, a Dog!
Today's date is: 2023-10-17 03:14:49.761529
```

自習１.

- 次のプログラムは、numberという変数の値を評価して、その値が10より大きいか、それ以外かを判断し、結果に応じてメッセージを表示する

numberの値を変更して、"数値は10より大きいです。"と表示させてください。

ヒント

- numberの値を変更するには、number = [新しい値]という形式でコードを書き換えてください。

```
number = 5


if number > 10:
print("数値は10より大きいです。")
else:
print("数値は10以下です。")
```

自習２.

- 次のプログラムは、半径を引数として受け取り、円の面積を計算して返す関数calculate_areaを定義している
- この関数は半径5の円の面積を計算し、結果を表示

calculate_area関数を使用して、半径10の円の面積を計算し、結果を表示してください。

ヒント

関数を呼び出すときは、関数名の後ろに括弧()をつけ、括弧の中に引数を指定する。例：calculate_area(10)

```
def calculate_area(radius):
pi = 3.14159
return pi * (radius**2)

area = calculate_area(5)
print(area)
```

自習３.

- 次は、いくつかの都市の名前が格納されているリストに関するソースコードです。
- このプログラムは、**リストの２番目**を表示しています。

リストの 3番目の都市を表示してください

ヒント

リストのインデックスは**0から開始**します。つまり、最初の要素はcities[0]でアクセスできます。

```
cities = ["Tokyo", "Osaka", "Kyoto", "Hokkaido", "Okinawa"]

print(cities[2])
```

自習４．

「５．Python 言語が広く使用されている理由」を読み、クラスとインスタンス、メソッドについて理解を深めてください

５．Python 言語が広く使用されている理由

【プログラムの説明】

- datetime モジュールをインポートしている。
- Animal というクラスが定義されており、動物の名前と種類を属性として持つ。
- Animal クラスには speak というメソッドがあり、動物の名前と種類を用いたメッセージを返す。
- display_date という関数は、現在の日時を返す。
- メイン処理では Animal クラスのインスタンスを作成し、そのインスタンスの speak メソッドを呼び出している。
- 現在の日付も表示される。

```
[2]  # モジュールのインポート
     import datetime

     # クラスの定義
     class Animal:
         def __init__(self, name, species):
             self.name = name
             self.species = species
         def speak(self):
             return f"I'm {self.name}, a {self.species}!"

     # 関数の定義
     def display_date():
         return datetime.datetime.now()

     # メイン処理
```

Google Colaboratoryの要点

- **アクセス**: Webブラウザからアクセス可能。
- **セルの種類**: コードセル（プログラム用）、テキストセル（説明用）。

基本操作

- **Googleアカウント**: 基本操作には**Googleアカウントが必要**。
- **操作の種類**: コードセルやテキストセルの**編集**、セルの**実行**、新規ノートブックの作成など。
- **保存**: 自動保存される。
- **セルの実行**: **基本、一番上のセルからすべてを実行してください**。このような、**複数のセルを一度**に実行することは、「**ランタイム**」メニューから「**すべてのセルを実行**」の操作でできます。

まとめ

文法のシンプルさ

- 直感的で読みやすい文法
- printで簡単に出力
- if, elseで条件分岐
- for, whileで繰り返し

拡張性

- 多岐にわたる分野での利用が可能
- 豊富なライブラリとモジュール

主なキーワード

print, type, if, else, for, while, def, return, class, __init__, self, vars, super

Google Colaboratoryの特徴

- オンラインで動作するPythonノートブック
- ノートブックは編集可能、実行可能（Google アカウントが必要）
- 混雑時の実行停止あり

3-2. Python の Pandas データフレーム

Python の Pandas データフレーム

表形式のデータ

	x	y
0	1	4
1	1	2
2	1	5
3	2	4
4	3	5
5	3	3

データ本体

Python の Pandas データフレームを組み立てるプログラム

	x	y
0	1	4
1	1	2
2	1	5
3	2	4
4	3	5
5	3	3

データフレーム

```
import pandas as pd

df = pd.DataFrame([[1, 4], [1, 2], [1, 5], [2, 4], [3, 5], [3, 3]], columns=['x', 'y'])
print(df)
```

```
   x  y
0  1  4
1  1  2
2  1  5
3  2  4
4  3  5
5  3  3
```

データフレームの作成と確認表示

データフレームの組み立て，結果は **df** に代入

```
df = pd.DataFrame([[1, 4], [1, 2], [1, 5], [2, 4], [3, 5], [3, 3]], columns=['x', 'y'])
```

Pandas データフレームの重要性

数多くの機能、柔軟性、高い性能を持つ

【主な機能】

- **データ分析**: 大量のデータを迅速に解析・操作する能力
- **データクリーニング**: 欠損データの処理や、データの変換・正規化を行う機能
- **時系列データ**: 時間データを扱うための機能
- **データの統合**: データの結合、マージ、グループ化が容易にできる。

【柔軟性】

様々なデータ形式（CSV, Excel, リレーショナルデータベース等）からの読み込みや書き出しをサポート。

【性能】

大規模なデータセットも効率的に処理できるよう最適化されている。

Pandas の基本的な使い方

- Google Colaboratoryにはpandasがインストールされている。
- **インポート**: pandasを使用する前に、import pandas as pdというコードでインポート。

主要なメソッド

- **CSVファイルの読み込み**: read_csvメソッド。

  例: pd.read_csv('file.csv')
- **Excelファイルの読み込み**: read_excelメソッド。

  例: pd.read_excel('file.xlsx')
- **データの先頭部分、末尾部分を表示**: head(), tail() メソッド。

  例: df.head() df.tail()
- **１列の選択**: [ ]を使用して列を選択。

  例: df['column_name']
- **複数列の選択:** [[ ]]を使用して列を選択。

  例: df[['a1', 'a2']]

データ分析の基本手順: 読み込みから散布図の作成まで

データの準備と確認

- **データの準備**: CSVファイルやExcelファイルを読み込むことも可能。

  例: df = pd.read_csv('file.csv') または df = pd.read_excel('file.xlsx')

- **データの先頭部分、末尾部分を表示（データが大きい場合）**: head(), tail() メソッド。

  例: df.head() df.tail()

列の選択（データに多数の列がある場合）

- [ ] （１列の選択）や [[ ]] （１列あるいは複数列の選択）を使用して列を選択。

  例: df[‘column_name’] df[['a1', 'a2']]

散布図の作成

- **Matplotlibのインポート**: 例 import matplotlib.pyplot as plt
- **２列から散布図を作成**: plt.scatter を使用
- **軸ラベルの追加:** 例 plt.xlabel('X-axis label') と plt.ylabel('Y-axis label')
- **タイトルの追加**: 例 plt.title('Scatter Plot Title')
- **最後にグラフを表示**: plt.show()

演習 2

トピックス】

- Pandas データフレーム
- データ分析
- データクリーニング（欠損データの置き換えや削除）
- 時系列データ
- データの統合
- 散布図

① Google Colaboratory のページを開く
https://colab.research.google.com/drive/1l2pLl72PYG5TFKKu8haOh_9fFCafxG4j?usp=sharing

② プログラムや説明や実行結果が表示されるので確認

③ **6，7，8**をよく読む

プログラムと説明をよく読み、掲載されている実行結果をよく確認。

④ 余裕があれば、次ページ以降の**自習**に取り組む

1．Python 言語が広く使用されている理由

【プログラムの説明】

- datetime モジュールをインポートしている。
- Animal というクラスが定義されており、動物の名前と種類を属性として持つ。
- Animal クラスには speak というメソッドがあり，動物の名前と種類を用いたメッセージを返す。
- display_date という関数は、現在の日時を返す。
- メイン処理では Animal クラスのインスタンスを作成し、そのインスタンスの speak メソッドを呼び出している。
- 現在の日付も表示される。

```
# モジュールのインポート
import datetime

# クラスの定義
class Animal:
    def __init__(self, name, species):
        self.name = name
        self.species = species
    def speak(self):
        return f"I'm {self.name}, a {self.species}!"

# 関数の定義
def display_date():
    return datetime.datetime.now()

# メイン処理
pet = Animal(name="Buddy", species="Dog")
print(pet.speak())
print(f"Today's date is: {display_date()}")
```

```
I'm Buddy, a Dog!
Today's date is: 2023-10-17 03:14:49.761529
```

自習．プログラムと説明を読み、次の基本を確認する

１．**データのダウンロード:**

「７」では、urllib.request.urlretrieve 関数を使用して、**指定したURLからデータをダウンロード**

2. **Pandasの基本:**

- pd.read_csv 関数を使って、**CSV ファイルのデータの読み込み**

- dropna を使用してデータクリーニング（欠損データの置き換えや削除）

- head を用いて、データの先頭部分を確認

3. 散布図

- グラフにタイトルやラベルを付ける

4. データの関係性の発見

- 散布図を通じて、間の関係性に気付くことができる

5. データの概観

- データの先頭を表示することで、どのようなデータが含まれているのか、その概観をつかむことができる

まとめ

Pandas データフレーム

- 表形式のデータを格納し、操作する機能を持つ

データフレームの組み立て: df = pd.DataFrame([[1, 4], [1, 2], ...], columns=['x', 'y'])

Pandas データフレームの重要性

- 主な機能:
  - **データ分析**: 大量のデータを迅速に解析・操作する能力。
  - **データクリーニング**: 欠損データの処理、データ変換、正規化など。
  - **時系列データ**: 時間データを扱うための専用の機能。
  - **データの統合**: データの結合、マージ、グループ化が簡単にできる。
- 柔軟性: CSV, Excel, リレーショナルデータベースなど、様々なデータ形式からの読み込みや書き出しをサポートしている。
- 性能: 大規模なデータセットも効率的に処理できるよう最適化されている。

①効率的なデータ解析

PandasはPythonの主要なデータ分析ライブラリであり、データフレームを中心とした強力な機能でデータ操作や解析を効率的に行います。これにより、データのパターンや関連性を迅速に把握することが可能です。

②様々な分野に通用するスキル

データサイエンス、AI、機械学習プロジェクト、研究データの管理など、多くの分野でPandasのスキルが求められています。前処理、データ解析、結果の出力まで一貫して行えます。

③スキルアップとキャリアの進展

現代は、「データ駆動型」の社会とも言えます。Pandas のスキルは、価値があり、自己PRにもプラスとなるスキルです。キャリアの進展や研究の質、日常作業の効率化に寄与します。