社内研修・データ分析

Version: 1.0

コンテンツ

Part 1

データのインポートとエクスポート

Version: 1.0

本研修の目的

データ分析の現場で即戦力となる人材の育成

基礎スキル:

- Pythonスキルの習得: データ分析に必須となるPythonの主要ライブラリを自在に使いこなす力
- 実践的プロセスの学習: 現場のデータに基づいた課題を通じ、データ抽出から分析までの一連の工程
- 応用力の育成: 学んだ知識を実務で応用するための土台を固めます

パート1の目的:

様々な形式のデータをPandas DataFrameに読み込み（抽出）、また書き出す（格納）ための基本的なスキルを習得します。

目次

ETLの概念

抽出 (Extraction):

- 基幹システム
- データベース
- 外部ファイルなど

様々なデータソースから必要なデータを抜き出す工程

変換 (Transformation):

- 抽出したデータを分析しやすい形式に整える工程です。
- データクレンジング、結合、新しい特徴量の作成など

格納 (Loading):

- 変換したデータを、分析用のデータベース（DWH）やデータマートに保存する工程です。

準備: ライブラリのダイレクトリーの確認

ライブラリの読み込み

```
import os
import pandas as pd
import sqlite3
from sqlalchemy import create_engine
```

ワーキングディレクトリの変更

```
if not os.getcwd().endswith('data'):
    os.chdir('data')

print(os.getcwd())
```

```
j:\Matchsul\kenshuu\data
```

本パートで取り扱うデータ

./data に保存されています

- car_braking.csv
- json1/20180328_A_0.json
- user_topic_follow_dummy_shiftjis.csv
- excel1/ClientSampleList.xlsx
- csv2/kc_house_data.csv
- bankdata.db (sqlite)
- bankdata (postgresql)
- newproducts.csv
- customers.csv

目次

データのインポート（抽出）

pandas

データインポート機能

Pandasは、多種多様なデータソースからデータを直接読み込むための強力な機能を提供しています。

ここでは、主要な4つの形式からのデータ読み込みを学びます。

データ形式	
	CSV/DSV　(.csv)
	Excel　(.xls/.xlsx)
	JSON (.json)
	SQL (.db/拡張子なし)

CSVファイルからのインポート（`read_csv`）

CSV/DSV　(.csv)

基本構文: `pandas.read_csv()`

注目引数:

- `filepath_or_buffer`: 読み込むファイルへのパス、もしくはURL。
- `sep`: 区切り文字。デフォルトは , (カンマ)。
- `header`: ヘッダーとして使用する行の番号（0から始まる）。ヘッダーがない場合は `None` を指定する。デフォルトは `0`。
- `index_col`: インデックス（行ラベル）として使用する列の番号や列名を指定する。
- `usecols`: 読み込む列を限定する場合に、列名または列番号のリストで指定する。
- `encoding`: ファイルのエンコーディングを指定する（例: `'utf-8'`, `'shift_jis'`）。
- `dtype`: 列ごとのデータ型を辞書形式で指定する（例: `{'col_A': str, 'col_B': int}`）

CSVファイルからのインポート（`read_csv`）

使用例

```
# 基本的なCSVファイルの読み込み
df_car = pd.read_csv('car_braking.csv')

# DataFrameの上部5行を表示
display(df_car.head(5))
```

	car_weight	car_velocity	tire_width	road_type	measured_braking_distance
0	1496.469186	50.761917	185	tarmac_wet	21.853128
1	1086.139335	47.162196	145	tarmac_dry	10.547325
2	1026.851454	4.404138	185	tarmac_wet	0.166108
3	1351.314769	106.082351	145	tarmac_wet	96.814662
4	1519.468970	81.177613	165	tarmac_dry	37.847769

CSVファイルからのインポート（`read_csv`）

encoding 引数

文字コードエラーへの対処

```
# encodingを指定して正しく読み込む
df_sjis = pd.read_csv("user_topic_follow_dummy_shiftjis.csv", encoding="shift-jis")
print("Shift-JIS形式のCSVファイルを正しく読み込めました。")
df_sjis.head()
```

```
Shift-JIS形式のCSVファイルを正しく読み込めました。
```

	Unnamed: 0	user_id	topic_name
0	0	25126	(株)アップル
1	1	26285	(株)電通
2	2	15409	.NET_Framework
3	3	30466	.NET_Framework
4	4	30878	.NET_Framework

コードの不一致が

ード（例: shift-jis

す。

CSVファイルからのインポート（read_csv）

dtype 引数

データ型の指定

```
# kc_house_data.csvのid列を文字列として指定して読み込む
df_house = pd.read_csv("csv2/kc_house_data.csv", dtype={'id': str})
print("読み込み後のid列のデータ型:", df_house['id'].dtype)
df_house.head()
```

```
読み込み後のid列のデータ型: object
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...

5 rows × 21 columns

Excelファイルからのインポート（read_excel）

Excel（.xls/.xlsx）

基本構文: `pandas.read_excel()`

注目引数:

- `io`: 読み込むExcelファイルへのパス（`.xls`, `.xlsx` など）。
- `sheet_name`: 読み込むシートの名前（文字列）または番号（0から始まる整数）を指定する。`None` を指定するとすべてのシートを辞書として読み込む。デフォルトは `0`。
- `header`: ヘッダーとして使用する行の番号（0から始まる）。デフォルトは `0`。
- `index_col`: インデックス（行ラベル）として使用する列の番号を指定する。
- `usecols`: 読み込む列の範囲（例: `'A:E'`）や、列名のリストで指定する。
- `engine`: 使用するパーサーエンジンを指定する（例: `'openpyxl'` は `.xlsx` ファイル用、`'xlrd'` は古い `.xls` ファイル用）。

Excelファイルからのインポート（read_excel）

使用例

```
# 選択コラムを指定する
cols = ['メールアドレス', '氏名', '氏名（カタカナ）', '性別', '生年月日', '郵便番号', '住所1', '住所2', '電話番号']
# Excelファイルの'基本情報'シートを読み込む
df_excel = pd.read_excel('excel1/ClientSampleList.xlsx', sheet_name='基本情報', usecols=cols)
df_excel.head()
```

	メールアドレス	氏名	氏名（カタカナ）	性別	電話番号	郵便番号	住所1	住所2	生年月日
0	kisaku0695@odkcstsmjw.cvm	菅原喜作	スガハラキサク	男	888213643	788-0001	高知県	宿毛市	1974/09/01
1	joakzuio-dosatoko552@yovqji.eiox.dr	森谷砂登子	モリヤサトコ	女	577625240	503-2212	岐阜県	大垣市	1997/08/12
2	kaori6942@gjvlsvuf.wnojw.vs	土谷香織	ツチヤカオリ	女	288670476	322-0077	栃木県	鹿沼市	1969/11/02
3	tadahiko56968@fvdfjxt.ili.zss	松下忠彦	マツシタタダヒコ	男	971536312	879-6113	大分県	竹田市	1963/08/10
4	isami44474@kkino.pg	神保勇	ジンボイサミ	男	554163015	406-0002	山梨県	笛吹市	1989/07/31

JSONファイルからのインポート（read_json）

JSON (.json)

基本構文: `pandas.read_json()`

注目引数:

- `path_or_buf`: 読み込むJSONファイルへのパス、URL、またはJSON文字列。
- `orient`: JSON文字列のフォーマットを指定する。エクスポート時（`to_json`）の `orient` と一致させる必要がある。
  - `'split'`: `{ 'index': [...], 'columns': [...], 'data': [...] }` 形式。
  - `'records'`: `[ {column: value, ...}, ... ]` 形式（行指向）。
  - `'index'`: `{ index: {column: value, ...}, ... }` 形式。
  - `'columns'`: `{ column: {index: value, ...}, ... }` 形式。**（デフォルト）**
  - `'values'`: `[ [value, ...], ... ]` 形式。
- `lines`: JSONオブジェクトが改行区切りで複数行にわたって記録されているか（`True`/`False`）。JSON Lines (jsonl) 形式の場合に `True` にする。
- `dtype`: 列ごとのデータ型を辞書形式で指定する。

JSONファイルからのインポート（read_json）

使用例

```
# SNS データを読み込む
df_json = pd.read_json('json1/20180328_A_0.json')
df_json.head(3)
```

	create_date	comment_count	like_count	reaction_count	tags	title	user_id
0	2018-03-28T10:11:03+09:00	0	0	0	[rust, vector, HashSet]	[Rust] Vectorをuniqueしたい	yagince
1	2018-03-28T10:06:53+09:00	0	0	0	[mysql5.7]	MySQL5.7をインストールしたときに初期パスワードを探してパスワードを変更するまで	mindlessdoll
2	2018-03-28T10:05:00+09:00	0	0	0	[C#, Azure, .NET, VisualStudio, Xamarin]	.NET プラットフォームで始めるクラウドネイティブアプリ開発	AzureAntenna

SQLデータベース (`read_sql_query()`,`read_sql_table()`)

SQL (.db/拡張子なし)

基本構文 (テーブル全体を読み込む場合): `pandas.read_sql_table()`

基本構文 (クエリ結果を読み込む場合): `pandas.read_sql_query()`

注目引数:

- `table_name` (`read_sql_table` のみ): 読み込むテーブルの名前。
- `sql` (`read_sql_query` のみ): 実行するSQLクエリ文。
- `con`: データベースへの接続オブジェクト (`SQLAlchemy` の `Engine` や `Connection` オブジェクトなど)。
- `index_col`: DataFrameのインデックスとして使用する列名を指定する。
- `columns` (`read_sql_table` のみ): 読み込む列をリストで指定する。指定しない場合はすべての列を読み込む。
- `params`: SQLクエリに渡すパラメータのリストや辞書（SQLインジェクション対策)。

SQLデータベース (`read_sql_query()`,`read_sql_table()`)

業務時のデータ処理上の注意事項

インポート

・アクセス許可の確認:

利用するデータ（ファイル、データベース）へのアクセスが、業務上正式に許可されていることを確認する。

エクスポート

・書き込み先の容量確認:

出力先に、ファイルを保存するための十分な空き容量があるか確認する。

・書き込み権限の確認:

指定した場所へのデータ書き込みが許可されているかを確認する。

データベースアクセス

・事前確認の徹底:

管理者（DBA）やチームリーダーへ事前に相談・確認し、操作の許可を得る。

・影響範囲の把握:

既存データへの影響を正確に把握し、意図しないデータ破壊が起きないよう慎重に操作する。

SQLデータベース（`read_sql_query()`,`read_sql_table()`）

```
#まずはSQLiteデータベースに接続します
conn = sqlite3.connect('BankData.db')

# SQLクエリを使ってLoanProductsテーブルから全コラムから全レコードを取得
```

	product_id	product_name	description	min_interest_rate	max_interest_rate	max_loan_term	start_date	end_date
0	1	住宅ローン（変動金利型）	新築・中古住宅購入資金。市場金利に応じて金利が変動します。	0.45	0.85	35	2024-04-01	2026-03-31
1	2	住宅ローン（全期間固定金利）	最長35年の長期固定金利で、返済額の安定を求める方に適しています。	1.35	1.85	35	2024-04-01	2026-03-31
2	3	住宅ローン（当初10年固定特約）	当初10年間は固定金利、その後は変動金利に移行します。	0.98	2.50	30	2024-04-01	2026-03-31

SQLデータベース（`read_sql_query()`,`read_sql_table()`）

```
# sqlalchemyを使ってSQLiteデータベースに接続
engine = create_engine('sqlite:///BankData.db')

table = "LoanProducts"
index = 'product_id'
columns = ['product_name', 'min_interest_rate', 'max_interest_rate']

# df_sql_tableにテーブル名とengineオブジェクトを渡し、データを取得
df_sql_table = pd.read_sql_table(table, engine, index_col=index, columns=columns)
df_sql_table.head(5)
```

product_id	product_name	min_interest_rate	max_interest_rate
1	住宅ローン（変動金利型）	0.45	0.85
2	住宅ローン（全期間固定金利）	1.35	1.85
3	住宅ローン（当初10年固定特約）	0.98	2.50
4	住宅ローン（借り換え専用）	0.55	1.20
5	教育ローン（固定金利）	1.80	3.50

と、索引コ

SQLデータベース（`read_sql_query()`,`read_sql_table()`）

```
# PostgreSQLデータベースの接続情報
db_user = 'postgres'
db_password = 'postgres'
db_host = 'localhost'
db_pc
db_na

# Pos
engir                                                                    ame}')

# テ-
#※Pc
table
index
colum

# df_
df_sc
df_sql_table_pg.head(5)
```

product_id	product_name	min_interest_rate	max_interest_rate
1	住宅ローン（変動金利型）	0.45	0.85
2	住宅ローン（全期間固定金利）	1.35	1.85
3	住宅ローン（当初10年固定特約）	0.98	2.50
4	住宅ローン（借り換え専用）	0.55	1.20
5	教育ローン（固定金利）	1.80	3.50

目次

インポートしたデータの基本確認

インポートしたデータの基本確認

データを読み込んだら、まずそのデータがどのようなものかを確認する基本のメソッドについて紹介します。

メソッド、属性

```
.head(), .tail()
```

```
.shape, .columns
```

```
.info(), .describe()
```

```
.index()
```

先頭・末尾の表示(.head(),.tail())

パートの最初から.head()を使ってデータの先頭を表示していますが、末尾の方はいかがでしょうか。

.tail(n)のメソッドを使用すると、下からn行を表示できます。

```
#display()関数を使ってJupyter Notebook内で明示的にDataFrameの内容を表示
print("\n--- df_car 末尾3行 ---")
display(df_car.tail(3))

print("\n--- df_sql_table_pg 末尾5行 ---")
display(df_sql_table_pg.tail(5))
```

```
--- df_sql_table_pg 末尾5行 ---
```

product_id	product_name	min_interest_rate	max_interest_rate
36	アートローン	2.50	5.50
37	セカンドハウスローン	1.80	3.20
38	引越しローン	4.20	8.80
39	住宅ローン（親子リレー返済）	1.45	2.05
40	災害復旧ローン	0.50	1.50

次元、特徴名（.shape, .columns）

・.shape: DataFrameの形状（行数, 列数）をタプルで返します。

・.columns: 列名の一覧を取得します。または、リストなどを格納することによって列名を設定する

```
print("データ形状（行数, 列数):", df_car.shape)

print("\n列名リスト:", df_car.columns.tolist())

df_car.columns = ['車重', '最高速度', 'タイヤ幅', '道路構造種', '計測停止距離']
display(df_car.head(3))
```

```
データ形状（行数, 列数): (2000, 5)

列名リスト: ['車重', '最高速度', 'タイヤ幅', '道路構造種', '計測停止距離']
```

	車重	最高速度	タイヤ幅	道路構造種	計測停止距離
0	1496.469186	50.761917	185	tarmac_wet	21.853128
1	1086.139335	47.162196	145	tarmac_dry	10.547325
2	1026.851454	4.404138	185	tarmac_wet	0.166108

データの情報（.info(), .describe()）

```
print("\nデータ情報の要約:")
df_car.info()
```

✓ 0.0s

```
データ情報の要約:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   車重      2000 non-null   float64
 1   最高速度    2000 non-null   float64
 2   タイヤ幅    2000 non-null   int64
 3   道路構造種   2000 non-null   object
 4   計測停止距離  2000 non-null   float64
dtypes: float64(3), int64(1), object(1)
memory usage: 78.3+ KB
```

```
print("\n基本統計量の表示:")
display(df_car.describe())
```

✓ 0.0s

基本統計量の表示:

	車重	最高速度	タイヤ幅	計測停止距離
count	2000.000000	2000.000000	2000.000000	2.000000e+03
mean	1299.745418	59.552688	160.230000	4.653077e+01
std	292.640693	34.542191	17.230657	5.040148e+01
min	800.081888	0.008141	135.000000	2.892887e-07
25%	1043.157413	29.893755	145.000000	6.818761e+00
50%	1296.989436	60.444921	165.000000	3.125727e+01
75%	1550.724632	89.123798	175.000000	6.793168e+01
max	1799.644251	119.923511	185.000000	2.716036e+02

データの情報（.index）

・.index:　データフレームのインデックスを返す属性。

・リスト、イテラブル型を格納することでインデックスを変更できます。

```
print("\nデータフレームのインデックス:")
print(df_car.index)

# インデックスを1から始まる連番に変更
df_car.index = range(1, len(df_car) + 1)
print("\n変更後のデータフレームのインデックス:")
print(df_car.index)
```

✓ 0.0s

```
データフレームのインデックス:
RangeIndex(start=0, stop=2000, step=1)

変更後のデータフレームのインデックス:
RangeIndex(start=1, stop=2001, step=1)
```

目次

データのインポート（抽出）

pandas

データのエクスポート（格納）

PandasのDataFrameは、`to_`で始まるメソッドを使って簡単に各種ファイル形式へ書き出すことができます。

	メソッド
	to_csv()
	to_excel()
	to_json()
	to_sql()

CSVファイルへのエクスポート(to_csv)

基本構文

基本構文:

`dataframe_object.to_csv()`

注目引数:

- `path_or_buf`: ファイルパス、もしくは`.write()`メソッドを実装しているオブジェクト。指定しない場合、CSVフォーマットの文字列を返す。
- `sep`: 区切り文字。デフォルトは`,`(カンマ)。
- `na_rep`: 欠損値 (NaN) を表現するための文字列。デフォルトは空文字。
- `columns`: 書き出す列を指定するリスト。
- `header`: ヘッダー（列名）を出力するかどうか。`False`にすると出力しない。デフォルトは`True`。
- `index`: インデックス（行名）を出力するかどうか。`False`にすると出力しない。デフォルトは`True`。
- `index_label`: インデックス列の列名を指定する。
- `encoding`: 出力ファイルのエンコーディングを指定する（例: `'utf-8'`, `'shift_jis'`)。

CSVへ書き出す際、`index=False`を指定しないと、DataFrameのインデックス（0, 1, 2...）が列として保存されてしまいます。インデックスが行番号　(0, 1, 2, ...)を表している際は`index=False`を指定するのが良いでしょう。

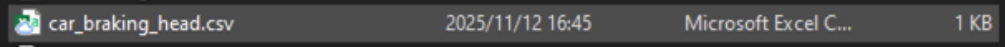
CSVファイルへのエクスポート（to_csv）

使用例

CSVへ書き出す際、`index=False`を指定しないと、DataFrameのインデックス（0, 1, 2...）が列として保存されてしまいます。インデックスが行番号　(0, 1, 2, ...)を表している際は`index=False`を指定するのが良いでしょう。

```
df_car.head(5).to_csv('car_braking_head.csv', index=False)
print("'car_braking_head.csv' が作成されました。")
```

```
'car_braking_head.csv' が作成されました。
```



Excelへのエクスポート（to_excel）

基本構文

基本構文: `dataframe_object.to_excel()`

注目引数:

- `excel_writer`: 書き込み先のファイルパス (例: `'output.xlsx'`) や `ExcelWriter` オブジェクトを指定する。
- `sheet_name`: 書き込むシート名。デフォルトは `'Sheet1'`。
- `na_rep`: 欠損値 (NaN) を表現するための文字列。
- `columns`: 書き出す列を指定するリスト。
- `header`: ヘッダー（列名）を出力するかどうか。デフォルトは `True`。
- `index`: インデックス（行名）を出力するかどうか。Excelでは不要な場合が多く、`False` にすることが多い。デフォルトは `True`。
- `index_label`: インデックス列の列名を指定する。
- `startrow`, `startcol`: 書き込みを開始する行と列の位置（0から始まるインデックス）を指定する。
- `engine`: 書き込みに使用するエンジンを指定する（例: `'openpyxl'`, `'xlsxwriter'`）。

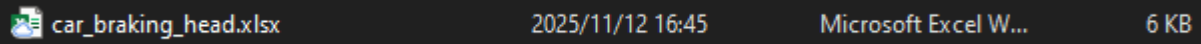
`openpyxl` のモジュールをインストールする必要があります。
`to_csv` と同様に `index=False` が便利です。`sheet_name` でシート名を指定することもできます。

Excelへのエクスポート（to_excel）

使用例

```
df_car.head(5).to_excel('car_braking_head.xlsx', sheet_name='先頭5行', index=False)
print("'car_braking_head.xlsx' が作成されました。")
```

```
'car_braking_head.xlsx' が作成されました。
```



JSONファイルへのエクスポート (to_json)

基本構文

基本構文: `dataframe_object.to_json()`

注目引数:

- `path_or_buf`: ファイルパス、もしくは書き込み可能なオブジェクト。指定しない場合、JSON文字列を返す。
- `orient`: JSONの出力形式を指定する。用途に応じて選ぶ。
  - `'split'`: 辞書形式でインデックス、列、データを持つ。
  - `'records'`: リスト形式で、各行を辞書として格納する。
  - `'index'`: インデックスをキーとした辞書の辞書。
  - `'columns'`: 列をキーとした辞書の辞書（デフォルト）。
  - `'values'`: 値のみをリストのリストとして格納する。
- `date_format`: 日付のフォーマットを指定する (`'epoch'` または `'iso'`)。
- `force_ascii`: 非ASCII文字をエスケープするかどうか。日本語を含む場合は `False` にすることが多い。デフォルトは `True`。
- `indent`: JSON文字列のインデント（空白の数）を指定し、出力を整形する。

JSONファイルへのエクスポート (to_json)

使用例

こちらの例では、orient=recordを指定したため、各行が辞書オブジェクトとして保存されています。

```
# JSONファイルにエクスポート
df_car.head(5).to_json('car_braking_head.json', orient='records', lines=True)
print("'car_braking_head.json' が作成されました。")
```

```
'car_braking_head.json' が作成されました。
```

car_braking_head.json	2025/11/12 16:45	JSON Source File	1 KB


```
{
    "car_weight": 1496.4691855979,
    "car_velocity": 50.7619168873,
    "tire_width": 185,
    "road_type": "tarmac_wet",
    "measured_braking_distance": 21.8531280201
}
{
    "car_weight": 1086.1393349504,
    "car_velocity": 47.1621955733,
    "tire_width": 145,
    "road_type": "tarmac_dry",
    "measured_braking_distance": 10.5473253891
}
{
    "car_weight": 1026.8514535642,
    "car_velocity": 4.4041379822,
    "tire_width": 185,
    "road_type": "tarmac_wet",
    "measured_braking_distance": 0.1661078798
}
{
    "car_weight": 1351.3147690829,
    "car_velocity": 106.0823513253,
    "tire_width": 145,
    "road_type": "tarmac_wet",
    "measured_braking_distance": 96.8146615315
}
{
    "car_weight": 1519.4689697856,
    "car_velocity": 81.1776131362,
    "tire_width": 165,
    "road_type": "tarmac_dry",
    "measured_braking_distance": 37.8477693667
}
```

データベースへのエクスポート (to_sql)

基本構文

基本構文: `dataframe_object.to_sql()`

注目引数:

- `name`: インポートと同様、書き込み先のSQLテーブル名を文字列で指定する。
- `con`: インポートと同様、データベースへの接続オブジェクト。`SQLAlchemy` の `Engine` や `Connection` オブジェクトを使用する。
- `schema`: 書き込み先のスキーマを指定する（データベースが対応している場合）。
- `if_exists`: テーブルが既に存在する場合の挙動を指定する。
  - `'fail'`: 何もせず `ValueError` を発生させる（デフォルト）。
  - `'replace'`: 既存のテーブルを削除してから新しいテーブルを作成し、データを挿入する。
  - `'append'`: 既存のテーブルにデータを追加する。
- `index`: DataFrameのインデックスをテーブルの列として書き出すかどうか。`True` の場合、インデックスが列になる。
- `index_label`: インデックスを列として書き出す場合の列名を指定する。
- `chunksize`: 一度に書き込む行数を整数で指定する。大きなDataFrameを扱う際にメモリ効率が向上する。
- `dtype`: 列のデータ型をSQLの型で指定する。`SQLAlchemy` の型を辞書形式でマッピングする（例: `{'col1': sqlalchemy.types.Integer}`）。

データベースへのエクスポート (to_sql)

SQLを安全に使いましょう

技術的な安全対策で、ヒューマンエラーを防ぐ

認証情報を安全に

【危険な例】 コード

```
create_engine
```

→ Gitなどでコードを

【推奨される方法】

.envファイルなどの
から除外する。

本番環境での操作は慎重に

- テスト環境で検証: まずは開発用やテスト用のデータベースで、コードが意図通りに動作することを十分に確認する。
- 許可なく変更しない: 本番（Live）データベースへの変更は、必ず事前に許可を得る。

ング）

```
ord@host/db')
```

ます。

渡す

oreでリポジトリ

データベースへのエクスポート (to_sql)

使用例

今回の使用例では、1.4で読み込んだPostgreSQLデータベースloanproductsテーブルにデータを追加します。

まずは、インポート時と同じエンジンを使用して、コラムのデータ型を確認します。

```
# PostgreSQLデータベースの既存のloanproductsテーブルを確認
table = "loanproducts"
index = 'product_id'
df_loan_existing = pd.read_sql_table(table, engine_pg, index_col=index)
df_loan_existing.dtypes
display(df_loan_existing.head(5))
```

```
product_name                 object
description                  object
min_interest_rate           float64
max_interest_rate           float64
max_loan_term                 int64
start_date           datetime64[ns]
end_date             datetime64[ns]
dtype: object
```

データベースへのエクスポート (to_sql)

使用例

同じように、テーブルに追加する予定のデータ newproducts.csvのデータ型を確認します。

```
# newproducts.csvをDataFrameに読み込み
df_new_products = pd.read_csv('newproducts.csv')
# データを確認
print(df_new_products.dtypes)
```

```
product_name          object
description           object
min_interest_rate    float64
max_interest_rate    float64
max_loan_term          int64
start_date            object
end_date              object
dtype: object
```

データベースへのエクスポート (to_sql)

使用例

新しいデータをデータベーステーブルのデータ型に合わせます。下記のコードでは、df_new_productsのコラムのデータ型を一括でdf_loan_existingのデータ型に変換します。場合によっては不要なステップですが、事前に合わせることでエラーを防げます。
実際にdf_new_productsのstart_dateとend_dateのデータ型がdf_loan_existingと一致しないため、データ型変換を行った方が無難です。

```
# df_loan_existingと同じデータ型に変換
df_new_products = df_new_products.astype(
    df_loan_existing.dtypes.to_dict())
display(df_new_products.dtypes)
```

```
product_name                 object
description                  object
min_interest_rate           float64
max_interest_rate           float64
max_loan_term                 int64
start_date           datetime64[ns]
end_date             datetime64[ns]
dtype: object
```

データベースへのエクスポート (to_sql)

使用例

最後に、追加するデータがデータベースのインデックス（プライベートキー）に違反しないよう、追加するテーブルのインデックスを既存のテーブルインデックスに合わせます。

```
# dataframeのインデックスを調整して一意にする
new_index = df_new_products.index + df_loan_existing.index.max() + 1 # 既存の最大インデックスに1を
df_new_products.index = new_index
# to_sqlを使ってPostgreSQLデータベースに新しいデータを追加
df_new_products.to_sql('loanproducts', engine_pg, if_exists='append', index='product_id')
print("新しいデータがPostgreSQLのloanproductsテーブルに追加されました。")
```

```
新しいデータがPostgreSQLのloanproductsテーブルに追加されました。
```

データベースへのエクスポート (to_sql)

使用例

Loanproductsをデータベースから再読み込み、データが追加されたことを確認します。

出力された.tail()の図表はデータが31から40のインデックスに追加されたことを表しています。

```
# 書き込まれたデータを確認
df_loan_updated = pd.read_sql_table(table, engine_pg, index_col=index)
display(df_loan_updated.tail(10)[['product_name']])
```

product_id	product_name
31	ペットローン
32	デンタルローン
33	自己投資ローン
34	トラベルローン
35	農業支援ローン
36	アートローン
37	セカンドハウスローン
38	引越しローン
39	住宅ローン（親子リレー返済）
40	災害復旧ローン

総合ハンズオン: Extraction & Loading

CSVからデータを抽出

データベースに格納

タスク

1. `A支店_customers.csv`をDataFrameに読み込みます。
2. 読み込んだDataFrameから、分析に必要となる以下の列のみを抽出した新しいDataFrame `df_extract` を作成してください。
   - 論理名: 顧客識別番号、名前、生年月日、性別、住所、電話番号、メールアドレス、職業、会社名、勤続年数、年収（万円）、申請目的
   - 物理名: `customer_id, name, date_of_birth, gender, address, phone_number, email, occupation, company_name, years_of_service, annual_income, objective`
3. PostgreSQLの`BankData`データベースに`customers`と言うテーブルとして書き込んでください。その際、以下の条件を守ってください。
   - もし同名のテーブルが既に存在していたら、新しい内容で**上書き**する(`if_exists='append'`)
   - DataFrameのインデックスはデータベースに**含めない**(`index=False`)。