令和７年度「地方やデジタル分野における専修学校理系転換等推進事業」

ビッグデータ技術応用教材ワークブック

本ビッグデータ技術応用教材ワークブックは、文部科学省の教育政策推進事業委託費による委託事業として、一般社団法人全国専門学校情報教育協会が実施した令和７年度「地方やデジタル分野における専修学校理系転換等推進事業」の成果物です。

情報成長分野の教育プログラム整備と教員育成による学科転換・新設推進事業

目次

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
導入ガイド

🎯 実習の概要

ようこそ、実習 1 へ！ この実習では、ビッグデータ基礎教材（特に第 1〜3 章）で学んだ知識を活かし、Python を使って実際の顧客購買データを分析する**「一連の流れ」**を体験します。
「データを見て、問題を特定し、綺麗にして、集計・可視化し、発見をまとめて報告する」という、データ分析の基本プロセスを、皆さんの手で実践できるようになることが目標です。

💡 この実習で身につくスキル

📋 分析計画

課題を分析可能な問いに分解する力

🔧 データ処理

pandas でのデータお掃除スキル

📈 可視化技術

matplotlib/seaborn でのグラフ作成力

📝 報告力

分析結果を分かりやすく伝える力

🎓 学習目標

この実習（Part 1〜4）を完了することで、あなたは...

1. ビジネス課題「優良顧客は誰か？」にデータで答えを出せるようになる
2. Python（pandas）を使って、CSV データを読み込み、欠損値・異常値を処理できるようになる
3. 基本的な集計（groupby）や、グラフ作成（棒、折れ線）ができるようになる
4. 顧客分析の定番手法「RFM 分析」を理解し、実行できるようになる
5. 分析結果から「何が言えるか」を考え、簡単なレポートにまとめられるようになる

🌟 ゴール：基本的なデータ分析プロジェクトを一人で完遂できる自信をつける！

💻 必要な環境とツール

📦 分析環境（どちらかを選ぼう）

🅰️ Google Colaboratory（推奨 ✨）

- ✅ **メリット:** インストール不要！ ブラウザだけで OK！ 無料！
- 必要なもの: Google アカウント、インターネット接続
- 🔗 アクセス: https://colab.research.google.com/
- 👉 **初めての方、PC 設定に自信がない方はこちらを選びましょう！** Part 1 で使い方を説明します。

🅱️ ローカル環境（Jupyter Notebook）

- ✅ **メリット:** オフラインでも使える、実務環境に近い
- 必要なもの: Python, Jupyter Notebook, pandas, matplotlib, seaborn, openpyxl (Anaconda がお勧め)
- 👉 既に環境がある方はこちらでも OK ですが、環境構築のサポートは限定的になります。

📂 使用するデータセット

「E-Commerce Data」(Kaggle)

イギリスのオンライン小売店の約 1 年分、約 54 万件の購買データです。実務で出会うようなリアルなデータ（欠損値なども含む）です。

- **主な列:** InvoiceNo(請求書番号), StockCode(商品コード), Description(商品説明), Quantity(数量), InvoiceDate(購入日時), UnitPrice(単価), CustomerID(顧客 ID), Country(国名)

📥 データの入手方法

1. **Kaggle にアクセス:** https://www.kaggle.com/ (アカウント作成が必要・無料)
2. **データセット検索:**「E-Commerce Data」で検索 (提供者: carrie1)
3. **ダウンロード:** data.csv (約 50MB) をダウンロードします。

👉 ダウンロードしたファイルは、Part 1 のワークブックを実行する際に Google Colaboratory にアップロードします。

📖 Part 別の学習内容

実習は 4 つの Part に分かれています。

Part 1　環境構築とデータ理解

分析準備、データ読み込み、データの全体像と問題点を発見！

⏱️ **約 1.5 時間**

Part 2　データクレンジングと前処理

欠損値・異常値のお掃除！分析しやすい形にデータを整える。

⏱️ **約 2 時間**

Part 3　探索的データ分析と可視化

集計とグラフ作成で、売上トレンドや人気商品を発見！

⏱️ **約 2 時間**

Part 4　統合分析（RFM）とレポート作成

優良顧客を見つけ出し、分析結果をレポートにまとめる！

⏱️ **約 2 時間**

📅 推奨学習スケジュール（例）

合計約 8 時間の内容です。自分のペースで進めましょう。

タイミング	学習内容
Week 1	Part 1 完了
Week 2	Part 2 完了
Week 3	Part 3 完了
Week 4	Part 4 完了 ＆ レポート提出

💡 **学習のコツ：** 各 Part の終わりにワークブックの「考察問題」にしっかり取り組みましょう。コードを書くだけでなく、「考える」ことが重要です！

✨ 次のステップ

🎯 準備はいいですか？

以下の準備をして、Part 1 の学習を始めましょう！

- ✅ Google アカウントの準備（Colab を使う場合）
- ✅ data.csv ファイルのダウンロード
- ✅ Part 1 のワークブックとオンライン講義（動画）の準備

🚀 データ分析の世界へ、Let's Dive In!

エラーを恐れず、楽しみながら進めていきましょう！

📚 ビッグデータ応用教材 - 実習 1 導入ガイド (学生用)

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
Part 1 （環境構築とデータ理解）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 1 の学習目標

この Part では、データ分析の第一歩を踏み出します：

- Python 分析環境（Google Colaboratory）を準備し、使えるようになる
- pandas を使って CSV（または Excel）データを読み込む方法を学ぶ
- データの全体像（行数、列数、データ型）を把握する方法を習得する
- 基本統計量（平均、最小値、最大値など）を算出し、その意味を解釈する
- データに含まれる「問題点」（欠損値、異常値）を発見する

📖 Part 1 について

データ分析は、まず**「データを正しく読み込み、その正体を知る」**ことから始まります。
この Part では、実際の E コマースデータを使い、分析の「土台作り」を行います。ここでしっかりデータと向き合うことが、後の分析の質を大きく左右します。

🛠️ タスク 1：Python 環境のセットアップ

📌 Task 1-1: 環境の選択と準備

Python を実行する環境を準備します。「導入ガイド」で説明したように、2 つの選択肢があります。

オプション A：Google Colaboratory（推奨）

インストール不要で、ブラウザと Google アカウントがあればすぐに始められます。**初心者の方、環境構築でつまずきたくない方は、こちらを強く推奨します。**

1. Web ブラウザで https://colab.research.google.com/ にアクセスします。
2. Google アカウントでログインします。
3. 「ファイル」メニューから「ノートブックを新規作成」をクリックします。
4. ノートブックの名前（例: bigdata_ex1_part1.ipynb）を変更します。

オプション B：ローカル環境（Jupyter Notebook）

自分の PC で環境を構築済みの方は、こちらを使用しても構いません。

1. Anaconda Navigator を起動し、Jupyter Notebook を「Launch」します。
2. または、ターミナル（コマンドプロンプト）で jupyter notebook と入力して実行します。
3. ブラウザが開き、Jupyter が起動したら、「New」→「Python 3」で新しいノートブックを作成します。

📌 Task 1-2: 必要なライブラリのインポート

データ分析に必要な「道具」（ライブラリ）をインポートします。新しいセルに以下のコードを入力してください。

```
# データ操作・分析のための pandas
import pandas as pd

# 数値計算のための numpy
import numpy as np

# グラフ描画のための matplotlib
import matplotlib.pyplot as plt

# より美しいグラフを描画するための seaborn
import seaborn as sns

# バージョン確認（任意）
print(f"pandas version: {pd.__version__}")
print("環境構築完了！")
```

💡 ヒント：コードの実行方法

- セルを選択した状態で **Shift + Enter** キーを同時に押します。
- または、セルの左側にある「▶」（再生）ボタンをクリックします。

ローカル環境でエラーが出たら：

ModuleNotFoundError が出た場合は、ライブラリが未インストールです。ターミナルで pip install pandas numpy matplotlib seaborn を実行してください。

📝 実行結果の記録

「環境構築完了！」のメッセージが表示されたことを確認してください。

📂 タスク 2：データの読み込み

Kaggle で公開されている「E-Commerce Data」を使用します。「導入ガイド」に従って、事前に data.csv ファイルをダウンロードしておいてください。

📌 Task 2-1: データセットの準備

data.csv ファイルを読み込みます。環境によって方法が異なります。

Google Colaboratory の場合（推奨）

以下のコードを実行し、表示される「ファイルを選択」ボタンから data.csv をアップロードします。

```
from google.colab import files
import io

# ファイルアップロードのダイアログを表示
uploaded = files.upload()

# アップロードしたファイル名を自動で取得
file_name = list(uploaded.keys())[0]

# CSV ファイルを pandas で読み込む
# encoding='ISO-8859-1' は文字化け対策です
df = pd.read_csv(io.BytesIO(uploaded[file_name]), encoding='ISO-8859-1')

print(f"ファイル '{file_name}' の読み込み完了！")
```

ローカル環境（Jupyter）の場合

data.csv を、この Jupyter Notebook ファイルと同じフォルダに置いてから、以下を実行します。

```
# ファイルパスを指定して読み込む
file_name = 'data.csv'
df = pd.read_csv(file_name, encoding='ISO-8859-1')

print(f"ファイル '{file_name}' の読み込み完了！")
```

📌 encoding='ISO-8859-1' とは？

このデータセットには、英語以外の文字（例：ヨーロッパのアクセント記号）が含まれています。通常の読み込み方法(UTF-8)ではエラーになるため、ISO-8859-1 という文字コードを指定しています。これは CSV ファイルや海外のデータを扱う際によく遭遇する問題です。

📌 Task 2-2: データの最初の確認

データが正しく読み込めたか、.head() を使って最初の 5 行を表示し、確認します。

```
# データの最初の 5 行を表示
df.head()
```

💡 なぜ .head() を使うの？

データは 54 万行もあります。すべてを表示すると大変なことになります。.head() を使うと、データがどんな列（カラム）で構成されているか、どんな値が入っているかを「チラ見」することができます。データ分析の第一歩として必須のコマンドです。

📝 観察の記録

表示された表を見て、気づいたことを記録してください：

🔍 タスク 3：データ構造の把握

.head() での「チラ見」の次は、データ全体の「健康診断」を行います。

📌 Task 3-1: データの形状確認（.shape）

データの「行数」と「列数」を確認します。

```
# データの形状（行数、列数）を確認
print(f"データの形状: {df.shape}")

# 行数と列数を個別に表示
print(f"行数: {df.shape[0]:,}")
print(f"列数: {df.shape[1]}")
```

📝 結果の記録

行数（レコード数）	
列数（変数の数）	

📌 Task 3-2: データの基本情報（.info()）

データの「戸籍謄本」のようなものです。各列のデータ型や、欠損値（空白）の有無を一覧で確認できます。

```
# データ型と欠損値の情報を表示
df.info()
```

💡 .info() の見方

- **Non-Null Count:** 欠損していない（値が入っている）データの数。
- 全体の行数（例：541,909）よりこの数が少ない列は、**「欠損値」がある**ことを意味します。
- **Dtype:** データ型。object は文字列、float64 は小数、int64 は整数です。

📝 .info() からの発見

.info() の結果を見て、以下の 2 点について記録してください：

1．　欠損値がある列はどれですか？

2．　気になるデータ型はありますか？

📌 Task 3-3: 欠損値の数（.isnull().sum()）

.info() で欠損値があることは分かりましたが、具体的に「何件」あるのかを数えます。

```
# 各列の欠損値の数を集計
missing_values = df.isnull().sum()

print("=== 各列の欠損値の数 ===")
print(missing_values)

# 欠損値がある列だけを表示
print("¥n=== 欠損値がある列（抜粋） ===")
print(missing_values[missing_values > 0])
```

📝 欠損値の数の記録

欠損値があった列について、その数を記録してください。

⚠️ 発見した問題点 ①

CustomerID が **13 万件以上も欠損**しています。これは「顧客分析」において大きな問題です。Part 2 でこのデータをどう扱うか考えます。

📊 タスク 4：基本統計量の算出

データの「健康診断」の仕上げとして、数値データの基本的な統計量（平均、最小、最大など）を確認します。

📌 Task 4-1: 数値データの統計量（.describe()）

.describe() を使うと、数値列（int64, float64）の統計量を一度に計算できます。

```
# 数値列の基本統計量を表示
df.describe()
```

💡 .describe() の見方

- **count:** データ数（欠損値を除く）
- **mean:** 平均値
- **std:** 標準偏差（データのばらつき具合）
- **min:** 最小値
- **25% / 50% / 75%:** 四分位数（50%は中央値と同じ）
- **max:** 最大値

📝 統計量からの発見

.describe() の結果を見て、Quantity と UnitPrice の min（最小値）はいくつですか？ そこから何がわかりますか？

⚠️ 発見した問題点 ②③

Quantity に**マイナスの値（返品）**が、UnitPrice に **0.0（無料）**が含まれています。これらも分析の邪魔になる「異常値」です。Part 2 で処理します。

📌 Task 4-2: 文字列データの確認（.describe(include='object')）

.describe() は通常、数値データのみを表示しますが、オプションで文字列（object）の統計量も確認できます。

```
# 文字列列（object 型）の基本統計量を表示
df.describe(include='object')
```

💡 文字列データの統計量

- **count:** データ数
- **unique:** ユニークな値の数（何種類あるか）
- **top:** 最も頻繁に出現する値（最頻値）
- **freq:** top の値が出現した回数

📝 文字列データからの発見

Country（国）の unique（種類数）と top（最頻国）はいくつですか？

🧐 Part 1 考察問題

ここまでの分析を通じて、以下の問いについて考え、記録してください。
問 1： このデータを使って「優良顧客を見つける」分析をしたい場合、Part 1 で見つかった「問題点」のうち、どれが最も深刻な問題だと思いますか？ その理由も説明してください。

問 2： Quantity（数量）の mean（平均値）と 50%（中央値）を比べてみましょう。大きな差がありますか？もし差がある場合、なぜそのような差が生まれていると考えられますか？（ヒント：max の値を見てみましょう）

✅ Part 1 のまとめ

🎉 お疲れ様でした！

Part 1 では、データ分析の第一歩である「環境構築とデータ理解」を完了しました。

📚 今日学んだこと

- Google Colaboratory での環境構築
- pd.read_csv() でのデータ読み込み
- .head(), .shape, .info(), .describe(), .isnull().sum() を使ったデータの「健康診断」
- データに潜む 3 つの主な問題点（①欠損値、②マイナス値、③ゼロ円）の発見

🔜 次のステップ：Part 2

Part 2 では、今日発見した「問題点」をすべて解決する「データクレンジング（お掃除）」を行います！

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習 1 Part 1）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
Part 2 （データクレンジングと前処理）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 2 の学習目標

この Part では、分析のためにデータを「きれい」にする技術を習得します：

- 欠損値（空白のデータ）の適切な処理方法（削除、補完）を学ぶ
- 異常値や外れ値（マイナスの数量など）の検出と処理方法を学ぶ
- データ型を適切な形式に変換する（例：float → int）
- 分析に役立つ新しい列（特徴量）を作成する（例：合計金額）
- 処理前と処理後でデータがどう変化したかを比較検証する

✨ 到達目標レベル

Part 2 を完了すると、「生の」データから分析可能な「きれいな」データを作成する一連のプロセスを一人で実施できるようになります。

📖 Part 2 について

Part 1 では、データに**「欠損値」**や**「マイナスの数量」**など、多くの問題点があることを発見しました。
「ゴミが入ったまま料理しても美味しくならない」のと同じで、汚れたデータのまま分析しても、正しい結果は得られません。
この Part では、データ分析プロセスの中で最も重要とも言われる**「データクレンジング（お掃除）」**と**「前処理」**を行います。

🔄 タスク 1：データの再読み込みと準備

📌 Task 1-1: ライブラリのインポートとデータの読み込み

まず、Part 1 と同様に、必要なライブラリをインポートし、データを読み込みます。

```
# 必要なライブラリのインポート
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# URL からデータを読み込む（Kaggle 版 CSV）
# この URL は Kaggle のデータセットを直接参照できるものです（要変更の可能性あり）
# 実際にはダウンロードした 'data.csv' を Colab にアップロードして使います
# url = "https://www.kaggle.com/datasets/carrie1/ecommerce-
data/download?datasetVersionNumber=1"
# df = pd.read_csv(url, encoding='ISO-8859-1')

# --- Google Colab でファイルをアップロードして読み込む場合 ---
from google.colab import files
uploaded = files.upload()

# アップロードしたファイル名（例: 'data.csv'）を指定
file_name = list(uploaded.keys())[0]
df = pd.read_csv(file_name, encoding='ISO-8859-1')

print(f"'{file_name}' を読み込みました。")
df.head()
```

💡 Google Colab でのファイルアップロード

1. 上記の files.upload() コードを実行します。
2. 「ファイルを選択」ボタンが表示されるので、Kaggle からダウンロードした data.csv を選択します。
3. アップロードが完了するまで待ちます。

encoding='ISO-8859-1' について：
このデータには特殊な文字が含まれているため、この「エンコーディング（文字コード）」を指定しないとエラーが出ます。
CSV ファイルを扱う際によくある問題です。

✓ チェックポイント 1-1

□ データが正常に読み込まれ、df.head() で先頭 5 行が表示された。

📌 Task 1-2: 作業用コピーの作成

元のデータを直接変更してしまうと、失敗したときに最初からやり直す必要があり大変です。
元のデータを df_raw として保持し、処理用のコピー df を作成しましょう。

```
# 元のデータをコピーして作業用 DataFrame を作成
df_raw = df.copy()
print("元のデータを df_raw にバックアップしました。")
```

📌 なぜ .copy() が必要なのか？

df_raw = df のように単純に代入すると、データそのものではなく「データの場所」だけがコピーされます。その結果、df を変更すると df_raw も変わってしまいます。
.copy() を使うと、データを丸ごと複製するため、お互いに影響しなくなります。これは pandas の重要なルールです。

🖌️ タスク 2：欠損値の処理

Part 1 で、CustomerID と Description に多くの欠損値（空白）があることがわかりました。これらを処理します。

📌 Task 2-1: 欠損値の再確認

まずは、Part 1 のおさらいです。欠損値の数を確認しましょう。

```
# 欠損値の数を確認
print("=== 欠損値の数（処理前） ===")
print(df.isnull().sum())

print(f"¥n 処理前の総行数: {len(df):,} 行")
```

📝 欠損値の記録（処理前）

CustomerID と Description の欠損値の数を記録してください。

📌 Task 2-2: CustomerID の欠損値処理

CustomerID は「顧客分析」の核となる情報です。これが無いデータは、今回の分析目的（優良顧客を見つける）には使えません。
そこで、CustomerID が欠損している行は、分析対象から除外（削除）します。

```
# CustomerID が欠損している行を削除する
df_cleaned = df.dropna(subset=['CustomerID'])

print("CustomerID の欠損行を削除しました。")
```

💡 dropna() の使い方

.dropna() は欠損値（NA）を削除（drop）する関数です。

- subset=['列名'] を指定すると、その列が欠損値の場合のみ行を削除します。
- （参考）指定しないと、どれか 1 つでも欠損値があれば行を削除します。

📌 Task 2-3: 処理結果の確認

処理によってデータがどう変わったか、必ず確認しましょう。

```
# 処理前後の行数を比較
print(f"処理前の総行数: {len(df):,} 行")
print(f"処理後の総行数: {len(df_cleaned):,} 行")
print(f"削除された行数: {len(df) - len(df_cleaned):,} 行")

# 処理後の欠損値を再確認
print("¥n=== 欠損値の数（処理後） ===")
print(df_cleaned.isnull().sum())
```

📝 処理結果の記録

処理後の総行数と、CustomerID の欠損値が 0 になったことを確認して記録してください。

✓ チェックポイント 2-3

☐ CustomerID の欠損値が 0 になった。
☐ 総行数が、削除した行数ぶん減っている。

🚫 タスク 3：異常値の処理

Part 1 で、Quantity（数量）や UnitPrice（単価）にマイナスの値や 0 があることを発見しました。これらを処理します。

📌 Task 3-1: 数量（Quantity）の異常値処理

Quantity がマイナスのデータは「返品」を表している可能性が高いです。今回の分析では「購入」データのみを対象とするため、Quantity が 0 以下の行は除外します。

```
# 処理前の統計量を確認
print("--- Quantity（処理前） ---")
print(df_cleaned['Quantity'].describe())

# Quantity が 0 より大きいデータのみを抽出
df_cleaned = df_cleaned[df_cleaned['Quantity'] > 0]

# 処理後の統計量を確認
print("¥n--- Quantity（処理後） ---")
print(df_cleaned['Quantity'].describe())
```

💡 条件でのデータ抽出

df[df['列名'] > 0] という書き方は、pandas で非常によく使うテクニックです。
これは、「df['列名'] > 0 という条件に当てはまる（True になる）行だけを選んでください」という意味です。

📝 記録

処理後の Quantity の min（最小値）が 1 になっていることを確認してください。

📌 Task 3-2: 単価（UnitPrice）の異常値処理

同様に、UnitPrice が 0 のデータは「無料サンプル」や「調整」などの可能性があり、売上分析にはノイズとなります。
UnitPrice が 0 より大きいデータのみを抽出します。

```
# 処理前の統計量を確認
print("--- UnitPrice（処理前） ---")
print(df_cleaned['UnitPrice'].describe())

# UnitPrice が 0 より大きいデータのみを抽出
df_cleaned = df_cleaned[df_cleaned['UnitPrice'] > 0]

# 処理後の統計量を確認
print("¥n--- UnitPrice（処理後） ---")
print(df_cleaned['UnitPrice'].describe())
```

📝 記録

処理後の UnitPrice の min（最小値）が 0 より大きい値になっていることを確認してください。

📌 Task 3-3: 最終的なデータ量の確認

すべてのクレンジングが終わりました。元のデータと比べて、どれくらい「きれいな」データが残ったか確認しましょう。

```
# 最終的なデータ量を確認
print(f"元のデータ行数（df_raw): {len(df_raw):,} 行")
print(f"クレンジング後の行数（df_cleaned): {len(df_cleaned):,} 行")
print(f"残ったデータの割合: {len(df_cleaned) / len(df_raw) * 100 :.2f} %")

# 最終的な欠損値チェック
print("¥n=== 最終的な欠損値の数 ===")
print(df_cleaned.isnull().sum())
```

✨ これが分析用データ！

これで、欠損値や異常値のない、分析に使用できる「きれいな」データセットが完成しました！

📝 最終確認

最終的な行数と、すべての列の欠損値が 0 になっていることを記録してください。

🔧 タスク 4：データ型の変換と特徴量作成

データをより分析しやすくするために、データ型を整えたり、新しい列（特徴量）を作成したりします。

📌 Task 4-1: データ型の変換

Part 1 で、CustomerID が float64（小数）になっていました。欠損値を削除したので、int64（整数）に変換しましょう。

```
# 現在のデータ型を確認
print("--- 変換前のデータ型 ---")
print(df_cleaned.dtypes)

# CustomerID を int64（整数）型に変換
df_cleaned['CustomerID'] = df_cleaned['CustomerID'].astype('int64')

# 変換後のデータ型を確認
print("¥n--- 変換後のデータ型 ---")
print(df_cleaned['CustomerID'].dtype)
```

💡 astype() の使い方

.astype('変換したい型名') を使うと、データ型を明示的に変換できます。
SettingWithCopyWarning という警告が出る場合がありますが、今回は無視しても問題ありません。これは「コピーに対して操作していますよ」というお知らせです。

✓ チェックポイント 4-1

☐ CustomerID のデータ型が int64 に変換された。

📌 Task 4-2: 特徴量の作成（TotalPrice）

データには「数量」と「単価」はありますが、「合計金額」がありません。分析に便利なので、TotalPrice 列を作成しましょう。

合計金額（TotalPrice） = 数量（Quantity） × 単価（UnitPrice）

```
# TotalPrice 列を作成
df_cleaned['TotalPrice'] = df_cleaned['Quantity'] * df_cleaned['UnitPrice']

# 作成した列を確認
print("TotalPrice 列を作成しました。")
df_cleaned.head()
```

📝 記録

df_cleaned.head() の結果を見て、一番右に TotalPrice 列が追加され、正しく計算されていることを確認してください。

📌 Task 4-3: 特徴量の作成（日付データ）

InvoiceDate（購入日時）は、そのままでは集計しづらいです。この列から「月」「曜日」「時間」といった情報を抽出して、新しい列を作成しましょう。

```
# InvoiceDate が object 型（文字列）なので、datetime 型（日時）に変換
df_cleaned['InvoiceDate'] = pd.to_datetime(df_cleaned['InvoiceDate'])

# 日時データから新しい列を抽出
df_cleaned['YearMonth'] = df_cleaned['InvoiceDate'].dt.to_period('M') # 年月
df_cleaned['Month'] = df_cleaned['InvoiceDate'].dt.month              # 月
df_cleaned['DayOfWeek'] = df_cleaned['InvoiceDate'].dt.day_name() # 曜日
df_cleaned['Hour'] = df_cleaned['InvoiceDate'].dt.hour                # 時間

# 作成した列を確認
print("日付関連の列を作成しました。")
df_cleaned[['InvoiceDate', 'YearMonth', 'Month', 'DayOfWeek', 'Hour']].head()
```

💡 pd.to_datetime() と .dt アクセサ

pd.to_datetime() は、文字列を日付型に変換する強力な関数です。
日付型に変換すると、.dt というアクセサ（道具箱のようなもの）が使えるようになり、.dt.month（月）や .dt.hour（時間）などを簡単に取り出せます。

✓ チェックポイント 4-3

☐ YearMonth, Month, DayOfWeek, Hour の 4 列が追加された。

🤔 Part 2 考察問題

ここまでの分析を通じて、以下の問いについて考え、記録してください。

問 1： なぜ CustomerID の欠損値は「削除」し、Quantity のマイナス値も「削除」するという判断をしたのですか？今回の分析目的（優良顧客を見つける）と関連付けて説明してください。

問 2： もし分析目的が「返品の多い商品を発見する」だったら、Quantity がマイナスのデータを削除する判断は正しいですか？理由とともに答えてください。

問 3： 今回作成した TotalPrice, Month, DayOfWeek, Hour といった新しい列（特徴量）は、Part 3 以降の分析でどのように役立つと思いますか？（例：〇〇列を使えば、〇〇な分析ができそう）

✅ Part 2 のまとめ

🎉 お疲れ様でした！

Part 2 では、データ分析で最も時間がかかるとも言われる「データクレンジングと前処理」を学びました。

📚 今日学んだこと

- 欠損値の削除（.dropna()）
- 条件に基づく行の抽出（例：df[df['Quantity'] > 0]）
- データ型の変換（.astype()）
- 新しい特徴量の作成（TotalPrice の計算）
- 日付データの処理（pd.to_datetime(), .dt）

🔜 次のステップ：Part 3

ついに、きれいにしたデータを使って分析と可視化を行います！

Part 3 のテーマ： 探索的データ分析と可視化

- groupby() を使ったデータ集計
- matplotlib や seaborn を使ったグラフ作成
- 売上トレンドや人気商品の発見

✓ Part 2 最終チェックリスト

以下の項目が完了していることを確認してください：

- ☐ df_cleaned という「きれいな」DataFrame が作成されている
- ☐ df_cleaned には欠損値が残っていない
- ☐ Quantity と UnitPrice は 0 より大きい値のみになっている
- ☐ CustomerID が int64 型になっている
- ☐ TotalPrice, YearMonth, Month, DayOfWeek, Hour 列が追加されている
- ☐ 考察問題に取り組んだ

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習 1 Part 2）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
Part 3 （探索的データ分析と可視化）ワークブック

📝 実習記録

氏名	
実習日	年　　月　　日

🎯 Part 3 の学習目標

この Part では、きれいになったデータから「発見」する技術を学びます：

- .groupby() を使って、目的に応じた集計ができるようになる
- matplotlib や seaborn を使って、基本的なグラフを作成できる
- 集計結果やグラフから、データに潜むパターンや傾向を読み解く（考察する）
- 「売上トレンド」「人気商品」「主要国」など、ビジネス上の問いにデータで答える

✨ 到達目標レベル

Part 3 を完了すると、データから問いに対する答えを導き出し、それをグラフで「見える化」して他者に伝える基礎スキルが身につきます。

📖 Part 3 について

Part 2 で、分析の「下ごしらえ」が完了し、df_cleaned という「きれいな」データが手に入りました。
Part 3 では、いよいよこのデータを使って「分析」と「可視化」を行います。このプロセスは**「探索的データ分析（EDA: Exploratory Data Analysis）」**と呼ばれ、データと「対話」しながら面白い発見（インサイト）を探っていく、データ分析で最もクリエイティブな作業です。
推奨所要時間：約 2 時間

🔄 タスク 1：準備（Part 2 のデータ引き継ぎ）

📌 Task 1-1: ライブラリのインポート

まずは、必要なライブラリをすべてインポートします。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# グラフをノートブック内に表示するためのおまじない
%matplotlib inline

# グラフのスタイルを設定
sns.set(style='darkgrid')
```

📌 Task 1-2: Part 2 のデータの復元

Part 2 で作成した df_cleaned を使います。もしノートブックを閉じてしまった場合は、Part 1, 2 のコードをすべて再実行して df_cleaned を作成してください。

ここでは、処理済みのきれいなデータ（df_cleaned）が既にあるものとして進めます。

```
# Part 2 で作成した df_cleaned が存在するか確認
# もしエラーが出る場合は、Part 1, 2 のコードを再実行してください。
print(f"クレンジング済みデータの行数: {len(df_cleaned):,} 行")
df_cleaned.head()
```

✓ チェックポイント 1-2

☐ df_cleaned が表示され、TotalPrice や Month 列が存在することを確認した。

📈 タスク 2：全体の売上トレンド分析

最初の問い：「このお店の売上は、時間とともにどう変化しているか？」

📌 Task 2-1: 月別の売上集計

Part 2 で作成した YearMonth（年月）列を使って、月ごとの TotalPrice（合計金額）を計算します。ここで、データ分析の最重要テクニック **.groupby()** が登場します。

```
# 1. YearMonth ごとにグループ分け
# 2. TotalPrice 列を指定
# 3. .sum() で合計を計算
```

```
monthly_sales = df_cleaned.groupby('YearMonth')['TotalPrice'].sum()

# 結果の表示
print("=== 月別の売上高 ===")
print(monthly_sales)
```

💡 .groupby() の考え方

df.groupby('A')['B'].sum() は、以下のような操作です。

1. 「A 列」の値が同じ行（例: '2011-01'）をすべて集める
2. 集めた行の「B 列」だけを取り出す
3. それらを .sum()（合計）する

「月ごと」「顧客ごと」「国ごと」など、集計の基本となる必須テクニックです。

📌 Task 2-2: 売上トレンドの可視化（折れ線グラフ）

集計したデータ（数字の羅列）を、matplotlib を使って折れ線グラフにし、「傾向」を視覚化します。

```
# グラフのサイズを指定
plt.figure(figsize=(12, 6))

# 折れ線グラフを作成
# monthly_sales.index が X 軸（年月）、monthly_sales.values が Y 軸（売上高）
plt.plot(monthly_sales.index.to_timestamp(), monthly_sales.values, marker='o')

# グラフのタイトルとラベルを設定
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Total Sales')

# グリッド線を表示
plt.grid(True)

# グラフを表示
plt.show()
```

💡 .to_timestamp()

YearMonth は '2011-01' のような「期間」の型なので、.to_timestamp() で '2011-01-01' のような「時点」の型に変換してからグラフに描画しています。

📝 グラフからの考察

表示された折れ線グラフを見て、気づいた「傾向」を記録してください。

🛍️ タスク 3：売れ筋商品の分析

次の問い：「どの商品が一番売れているのか？」

📌 Task 3-1: 商品別の販売数量を集計

Description（商品説明）でグループ分けし、Quantity（数量）を合計します。

```
# 商品(Description) ごとに Quantity を合計
product_sales = df_cleaned.groupby('Description')['Quantity'].sum()

# .sort_values() で販売数量が多い順に並び替え
top_10_products = product_sales.sort_values(ascending=False).head(10)

print("=== 売れ筋商品 Top 10 ===")
print(top_10_products)
```

📌 Task 3-2: 売れ筋商品の可視化（棒グラフ）

ランキングは「棒グラフ」で可視化するのが効果的です。今回は seaborn を使ってみましょう。

```
# グラフのサイズを指定
plt.figure(figsize=(12, 7))

# seaborn で棒グラフを作成
# X 軸に販売数量、Y 軸に商品名を設定
sns.barplot(x=top_10_products.values, y=top_10_products.index, palette='viridis')

# グラフのタイトルとラベルを設定
plt.title('Top 10 Best-Selling Products (by Quantity)')
plt.xlabel('Total Quantity Sold')
plt.ylabel('Product Description')

# グラフを表示
plt.show()
```

💡 matplotlib vs seaborn

matplotlib はグラフの「基礎」で、細かいカスタマイズが得意です。
seaborn は matplotlib をベースにしており、より少ないコードで統計的に美しく、複雑なグラフ（棒グラフ、ヒストグラム、散布図など）を描画できます。

📝 グラフからの考察

グラフを見て、最も売れている商品や、ランキングの傾向について記録してください。

🌍 タスク 4：顧客・国別の分析

最後の問い：「どの国からの注文が多いのか？」

📌 Task 4-1: 国別の売上を集計

Country（国）でグループ分けし、TotalPrice（合計金額）を合計します。

```
# 国(Country) ごとに TotalPrice を合計し、上位 10 カ国を抽出
country_sales =
df_cleaned.groupby('Country')['TotalPrice'].sum().sort_values(ascending=False)

top_10_countries = country_sales.head(10)

print("=== 売上高 Top 10 カ国 ===")
print(top_10_countries)

# 全売上高に占めるイギリスの割合
uk_ratio = country_sales['United Kingdom'] / country_sales.sum() * 100
print(f"¥n 全売上に占めるイギリス（UK）の割合: {uk_ratio:.2f} %")
```

📌 Task 4-2: 国別売上の可視化（棒グラフ）

これも棒グラフで可視化しますが、今回はイギリス（UK）を除外して、他の国の比較を見やすくしてみましょう。

```
# イギリス(UK)を除外したトップ 10（UK が圧倒的すぎるため）
top_10_countries_excluding_uk = country_sales.drop('United Kingdom').head(10)

plt.figure(figsize=(12, 7))
sns.barplot(x=top_10_countries_excluding_uk.values,
y=top_10_countries_excluding_uk.index, palette='mako')

plt.title('Top 10 Countries by Sales (Excluding UK)')
```

```
plt.xlabel('Total Sales')
plt.ylabel('Country')
plt.show()
```

📝 グラフからの考察

集計結果とグラフを見て、このオンラインショップの地理的な特徴を記録してください。

🤔 Part 3 考察問題

ここまでの分析を通じて、以下の問いについて考え、記録してください。
問 1： 月別売上のグラフ（Task 2-2）では、11 月が突出していました。なぜ 11 月の売上がこれほど高いのか、考えられる「仮説」を 3 つ挙げてください。

問 2： 売れ筋商品のグラフ（Task 3-2）を見て、このお店は「どのような種類の商品」を主に扱っていると推測できますか？

問 3： 国別売上の分析（Task 4-1）から、もしあなたがこの会社のマーケティング担当者なら、次はどのような分析を行いますか？（例：売上が 2 位の〇〇国について、XX を調べる）

✅ Part 3 のまとめ

🎉 お疲れ様でした！

Part 3 では、データを集計し、可視化することで「問いに答える」プロセスを学びました。

📚 今日学んだこと

- .groupby() を使った集計（グループ化 → 集計関数）
- .sort_values() での並び替えと .head() での Top N 抽出
- matplotlib での折れ線グラフ作成（plt.plot()）
- seaborn での棒グラフ作成（sns.barplot()）
- 集計結果やグラフからビジネス的な「考察」を行うことの重要性

🔜 次のステップ：Part 4

いよいよ最終章です。これまでの分析の集大成として、「顧客」に焦点を当てた RFM 分析を行い、分析レポートを作成します！

Part 4 のテーマ： 統合分析（RFM）とレポート作成

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習 1 Part3）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
Part４（統合分析（RFM）とレポート作成）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part４ の学習目標

実習 1 の集大成です。以下のスキルを習得します：

- 顧客分析の代表的手法である **RFM 分析**の概念を理解し、実装できる
- RFM 指標（Recency, Frequency, Monetary）を計算し、顧客をスコアリングできる
- スコアに基づいて顧客をセグメント（グループ）分けできる
- これまでの分析結果（Part 1～4）を統合し、ビジネス上の「示唆」を導き出す
- 分析結果を他者に伝えるための簡単なレポートを作成できる

✨ 到達目標レベル

Part 4 を完了すると、データ分析の一連のプロセスを完遂し、その結果を基に簡単なビジネス提案ができるようになります。

📖 Part 4 について

Part 3 では、全体の売上トレンドや人気商品について分析しました。
Part 4 では、分析の焦点を**「顧客」**に絞り、マーケティングで広く使われる**「RFM 分析」**という手法を用いて、「どのような顧客がこのお店にとって重要（優良）なのか？」を明らかにします。
そして、これまでの分析結果をすべて統合し、最終的な**「分析レポート」**を作成します。分析スキルだけでなく、「伝える力」もここで試されます。
推奨所要時間：約 2 時間

🔄 タスク 1：準備（Part 3 のデータ引き継ぎ）

📌 Task 1-1: ライブラリのインポートとデータの復元

Part 3 までと同様に、ライブラリをインポートし、Part 2 で作成した df_cleaned を準備します。
import pandas as pd

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import datetime as dt # 日付計算のために datetime をインポート

%matplotlib inline
sns.set(style='darkgrid')

# --- ここに Part 1, 2 のコードを実行して df_cleaned を作成する ---
# (もしノートブックが継続していれば不要)
# 例: df = pd.read_csv(...)
#       df_cleaned = df.dropna(...)
#       df_cleaned = df_cleaned[df_cleaned['Quantity'] > 0] ... など

# df_cleaned が存在するか確認
if 'df_cleaned' in locals():
    print(f"クレンジング済みデータの行数: {len(df_cleaned):,} 行")
    df_cleaned.head()
else:
    print("エラー: df_cleaned が存在しません。Part 1, 2 のコードを再実行してください。")
```

📊 タスク 2：RFM 分析の実践

💡 RFM 分析とは？

顧客を以下の 3 つの指標で評価し、優良顧客を見つけるための分析手法です。

- **Recency (R)**: 最新購買日（最近いつ買ったか？） → 最近買う人ほど良い
- **Frequency (F)**: 購買頻度（何回買ったか？） → 頻繁に買う人ほど良い
- **Monetary (M)**: 購買金額（いくら使ったか？） → たくさん使う人ほど良い

📌 Task 2-1: RFM 指標の計算に必要なデータの準備

RFM を計算するために、基準となる「今日」の日付を設定します。データ期間の最終日の翌日としましょう。

```
# 基準日を設定（最終取引日の翌日）
# InvoiceDate が datetime 型であることを確認
if df_cleaned['InvoiceDate'].dtype != '
```

次に、顧客ごとに集計（groupby）して、R, F, M の元となるデータを計算します。

```
# 顧客 ID(CustomerID)ごとに集計
rfm_data = df_cleaned.groupby('CustomerID').agg({
    'InvoiceDate': lambda date: (snapshot_date - date.max()).days, # Recency: 基準日 - 最
```

```
新購買日（日数）
    'InvoiceNo': 'nunique',                                    # Frequency：請求書番号の
ユニーク数（購買回数）
    'TotalPrice': 'sum'                                        # Monetary：合計金額
})

# 列名を分かりやすく変更
rfm_data.rename(columns={'InvoiceDate': 'Recency',
                         'InvoiceNo': 'Frequency',
                         'TotalPrice': 'Monetary'}, inplace=True)

# 結果を確認
print("¥n=== RFM 指標の元データ ===")
rfm_data.head()
```

💡 .agg() の使い方

.agg() は、groupby した後に、列ごとに異なる集計（合計、ユニーク数、最大値など）を一度に行うための強力な関数です。
lambda date: ... は、「各顧客の購買日リスト(date)に対して、(基準日 - 最新購買日).days を計算してください」という少し高度な指示です。

📌 Task 2-2: RFM スコアの計算

R, F, M の値をそのまま比較するのは難しいので、それぞれをランク付け（例：5 段階評価）して「スコア」にします。
ここでは、pd.qcut() を使って、各指標を値の大きさに基づいて 5 つのグループ（1〜5）に分けます。

- Recency：日数が**短い**ほどスコアが高い（5 点）
- Frequency：回数が**多い**ほどスコアが高い（5 点）
- Monetary：金額が**高い**ほどスコアが高い（5 点）

```
# R, F, M それぞれを 5 段階でスコアリング（qcut で 5 等分）
# Recency は値が小さいほど良いので、ラベルを逆順 [5, 4, 3, 2, 1] にする
r_labels = range(5, 0, -1)
rfm_data['R_Score'] = pd.qcut(rfm_data['Recency'], q=5, labels=r_labels, duplicates='drop')

# Frequency, Monetary は値が大きいほど良いので、ラベルは [1, 2, 3, 4, 5]
fm_labels = range(1, 6)
rfm_data['F_Score'] = pd.qcut(rfm_data['Frequency'].rank(method='first'), q=5,
labels=fm_labels, duplicates='drop') # rank() は同値対策
rfm_data['M_Score'] = pd.qcut(rfm_data['Monetary'], q=5, labels=fm_labels,
duplicates='drop')
```

```
# スコアが数値型でない場合があるので、数値型に変換
rfm_data['R_Score'] = rfm_data['R_Score'].astype(int)
rfm_data['F_Score'] = rfm_data['F_Score'].astype(int)
rfm_data['M_Score'] = rfm_data['M_Score'].astype(int)


# 結果を確認
print("=== RFM スコア ===")
rfm_data[['R_Score', 'F_Score', 'M_Score']].head()
```

💡 pd.qcut() とは？

データを値の大きさ順に並べて、指定した数（q=5 なら 5 つ）のグループに均等に分割し、それぞれにラベル（labels=[...]）を付ける関数です。スコアリングによく使われます。
duplicates='drop' や .rank(method='first') は、同じ値がたくさんある場合に qcut がエラーになるのを防ぐための工夫です。

📌 Task 2-3: RFM セグメントの作成

R, F, M のスコアを組み合わせて、顧客をいくつかのセグメント（グループ）に分類します。ここでは簡単な例として、「R スコア」と「F スコア」を文字列として結合したものをセグメント名とします。

```
# R スコアと F スコアを文字列として結合してセグメント名を作成
rfm_data['Segment'] = rfm_data['R_Score'].astype(str) + rfm_data['F_Score'].astype(str)

# 結果を確認
print("=== RFM セグメント ===")
rfm_data[['R_Score', 'F_Score', 'Segment']].head()
```

さらに、代表的なセグメント名を定義して、より分かりやすく分類してみましょう。

```
# セグメント名を定義する関数
def assign_segment_name(row):
    if row['R_Score'] >= 4 and row['F_Score'] >= 4:
        return 'Champions' # R も F も高い -> 最優良顧客
    elif row['R_Score'] >= 3 and row['F_Score'] >= 3:
        return 'Loyal Customers' # R も F もそこそこ高い -> 優良顧客
    elif row['R_Score'] >= 4 and row['F_Score'] < 3:
        return 'Potential Loyalists' # 最近来ているが頻度が低い -> 育成候補
    elif row['R_Score'] < 3 and row['F_Score'] >= 4:
        return 'At Risk Customers' # 頻度は高いが最近来ていない -> 離反危険
    elif row['R_Score'] < 3 and row['F_Score'] < 3:
```

```
        return 'Lost Customers' # 最近来ておらず頻度も低い -> 離反顧客
    else:
        return 'Others' # その他

# 各行に関数を適用してセグメント名を割り当て
rfm_data['Segment_Name'] = rfm_data.apply(assign_segment_name, axis=1)

# 結果を確認
print("¥n=== セグメント名の割り当て ===")
rfm_data[['R_Score', 'F_Score', 'Segment_Name']].head()
```

📌 セグメント定義は一例

このセグメント分けのルールはあくまで一例です。ビジネスの状況に合わせて、「どの顧客層に注目したいか」によって自由に定義できます。

📌 Task 2-4: セグメント分析と可視化

作成したセグメントごとに、顧客数や平均購入金額などを集計し、比較してみましょう。

```
# セグメントごとの顧客数、平均 Recency、平均 Frequency、平均 Monetary を集計
segment_summary = rfm_data.groupby('Segment_Name').agg(
    Count=('Segment_Name', 'count'),
    Avg_Recency=('Recency', 'mean'),
    Avg_Frequency=('Frequency', 'mean'),
    Avg_Monetary=('Monetary', 'mean')
).round(1) # 小数点第一位で丸める

# 結果を表示 (顧客数が多い順)
print("=== セグメント別サマリー ===")
print(segment_summary.sort_values(by='Count', ascending=False))

# セグメントごとの顧客数を棒グラフで可視化
plt.figure(figsize=(10, 6))
sns.countplot(y='Segment_Name', data=rfm_data,
order=rfm_data['Segment_Name'].value_counts().index, palette='magma')
plt.title('Number of Customers by Segment')
plt.xlabel('Number of Customers')
plt.ylabel('Segment Name')
plt.show()
```

📝 分析結果の記録

集計結果とグラフを見て、特に注目すべきセグメント（例：Champions, Lost Customers）とその特徴を記録してください。

📝 タスク 3：分析レポートの作成

実習 1 の最終成果物として、ここまでの分析結果をまとめた簡単なレポートを作成します。
以下の構成要素を含めて、読み手に「何が分かり」「どうすべきか」が伝わるように記述してください。

📊 分析レポート

1. 分析の目的
この分析で何を明らかにしようとしたのか？（例：優良顧客の特徴を把握し、売上向上の施策につなげる）
2. 使用データと前処理
どのようなデータを使用し、どのような前処理（クレンジング）を行ったか？
3. 主な発見事項（インサイト）
Part 3 の分析（トレンド、売れ筋商品、国別）や、Part 4 の RFM 分析からわかった重要なポイントを 3〜5 つ記述してください。グラフも添えると良いでしょう。
4. 結論と提案（ネクストステップ）
分析結果から、どのような結論が導き出せますか？ それに基づき、具体的なアクション（施策）を提案してください。

💡 レポート作成のヒント

- **読み手を意識する：**専門用語を避け、分かりやすい言葉で書く。
- **結論から書く：**最初に最も伝えたいことを書く。
- **根拠を示す：**主張には必ず分析結果（グラフや数値）を添える。
- **簡潔に：**長文ではなく、要点を絞って書く。
- **Jupyter Notebook の Markdown：** このワークブック自体をレポートとして提出することも可能です。Markdown 記法を使って、コードの間に説明や考察を書き込みましょう。

✅ Part 4 のまとめ

🎉 実習 1 完了！お疲れ様でした！

Part 4 では、顧客分析の代表的手法である RFM 分析を実践し、分析結果をレポートにまとめるスキルを習得しました。

📚 実習 1 全体で学んだこと

- Python によるデータ分析の基本プロセス（読み込み→前処理→分析→可視化→レポート）
- Pandas を使ったデータ操作・集計
- Matplotlib/Seaborn を使った基本的なグラフ作成
- データからインサイトを導き出し、考察する力
- 分析結果を他者に伝えるためのレポート作成

🚀 次のステップ

実習 1 で身につけた分析スキルを土台に、実習 2 では「データの流れ（ETL）」と「ダッシュボード作成（BI）」に挑戦します！
ぜひ、発展課題にもチャレンジして、さらにスキルを磨いてください。

☑ 実習振り返りシート （ビッグデータ技術応用 - 実習 1 Part 4）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 1 Python による顧客データ分析とレポーティング
発展課題

🌟 発展課題について

実習 1 の基本内容をマスターした皆さん、おめでとうございます！
ここでは、学んだ知識をさらに応用するためのチャレンジ課題を用意しました。難易度別に**初級・中級・上級**の 3 レベルがあります。
自分の興味やレベルに合わせて、ぜひ挑戦してみてください。完璧にできなくても構いません。試行錯誤するプロセスが重要です！
初級 曜日・時間帯別分析 推奨：30-60 分

🎯 目的

実習で作成した日付関連の特徴量（DayOfWeek, Hour）を活用し、曜日や時間帯による売上の傾向を分析・可視化する。

📝 実装内容

1. DayOfWeek ごとに TotalPrice を合計し、棒グラフで曜日別の売上を比較する。
2. Hour ごとに TotalPrice を合計し、折れ線グラフで時間帯別の売上トレンドを可視化する。
3. （オプション）曜日と時間帯を組み合わせて、どの曜日の何時頃が最も売上が高いか分析する（例：ヒートマップ）。

💡 ヒント

- Part 3 の Task 2 (月別売上) や Task 3 (商品別売上) のコードが参考になります。
- groupby() の対象列を DayOfWeek や Hour に変更してみましょう。
- 曜日を正しい順序（月曜→日曜）で表示するには、pandas.Categorical を使うと便利です（調べてみましょう！）。
- ヒートマップは seaborn.heatmap() で作成できます。データをピボットテーブル形式にする必要があります。

✅ 評価ポイント

- groupby() を正しく使って集計できているか。
- 結果を適切なグラフ（棒グラフ、折れ線グラフ）で可視化できているか。
- グラフから読み取れる曜日・時間帯の傾向について考察できているか。

中級 簡単なバスケット分析 推奨：60-90 分

🎯 目的

顧客が一緒に購入しやすい商品の組み合わせ（バスケット）の傾向を分析する基礎を学ぶ。「この商品を買った人は、これも買いやすい」といった関連性を探る。

📝 実装内容

1. 最も売れている商品（例：Part 3 で見つけた Top 1 商品）を特定する。
2. その Top 1 商品が含まれる InvoiceNo（請求書＝買い物カゴ）をすべてリストアップする。
3. リストアップした InvoiceNo の取引データのみを抽出し、その中で Top 1 商品以外にどの商品が多く含まれているかを集計・ランキング化する。
4. 結果を棒グラフで可視化する。

💡 ヒント

- 特定の商品の Description を変数に格納します（例: target_product = '...'）。
- df_cleaned[df_cleaned['Description'] == target_product]['InvoiceNo'].unique() で、対象商品を含む請求書番号リストが取得できます。
- .isin() メソッドを使うと、リストに含まれる請求書番号のデータを効率的に抽出できます（例: df_cleaned[df_cleaned['InvoiceNo'].isin(invoice_list)]）。
- 抽出したデータから、target_product 自身を除外して集計する必要があります。
- value_counts() が頻度集計に便利です。

✅ 評価ポイント

- pandas のフィルタリングや .isin() を駆使して、関連データを正しく抽出できているか。
- 一緒に買われやすい商品のランキングを正しく集計・可視化できているか。
- 結果から、どのような商品プロモーション（セット販売など）が考えられるか考察できているか。

上級 コホート分析による顧客定着率 推奨：90-120 分

🎯 目的

顧客が初めて購入した月（コホート）ごとに、その後のリピート購入率（定着率）がどのように推移するかを分析する。顧客維持の状況を把握する。

📝 実装内容

1. 顧客ごとに「初回購入年月（CohortMonth）」を特定する。
2. 各取引データに、その顧客の CohortMonth を紐付ける。
3. 各取引データについて、「初回購入から何ヶ月後（CohortIndex）」の購入かを計算する。
4. CohortMonth と CohortIndex でグループ化し、各月にアクティブだった（購入した）顧客数を集計する。

5. 各コホートの初回顧客数を基準として、CohortIndex ごとのリテンション率（顧客維持率 = 当該月の顧客数 / 初回顧客数）を計算する。
6. 結果をヒートマップで可視化する。

💡 ヒント

- 初回購入月
  は df_cleaned.groupby('CustomerID')['InvoiceDate'].transform('min').dt.to_period('M') で計算できます（transform がポイント）。
- CohortIndex は、購入年月と初回購入年月の差で計算できます。
- 集計には groupby(['CohortMonth', 'CohortIndex'])['CustomerID'].nunique() が使えます。
- リテンション率の計算には、pivot_table が便利です。
- ヒートマップは seaborn.heatmap() で作成します。パーセンテージ表示（fmt='.0%'）やカラーマップ（cmap='viridis'）を設定すると見やすくなります。

✅ 評価ポイント

- pandas の時系列処理や transform, pivot_table を適切に利用できているか。
- リテンション率を正しく計算できているか。
- ヒートマップを生成し、そこから顧客定着に関する傾向（例：初期の離脱率が高い、特定のコホートの定着率が良い/悪いなど）を考察できているか。

🚀 ビッグデータ応用教材 - 実習 1 発展課題

ビッグデータ技術応用
実習２　アクセスログ ETL 処理と BI ツールによる可視化
導入ガイド

🎯 実習の概要

この実習では、Web サイトの**アクセスログ**という、コンピュータが自動生成する「生の」データを題材にします。
実習 1 で学んだ Python スキルを応用して、このログデータを**分析しやすい形に加工（ETL 処理）**し、最終的に **Tableau** や **Power BI** といった専門的な **BI（ビジネスインテリジェンス）ツール**を使って、Web サイトの状況を把握するための**インタラクティブなダッシュボード**を作成します。

💡 この実習で身につくスキル

🔧 ETL 処理

非構造化データ（ログ）を Python で加工・整形するスキル

🧱 データマート設計

分析目的に合わせたデータマート（中間データ）を設計する考え方

📈 BI ツール活用

Tableau/Power BI の基本操作とダッシュボード作成スキル

👀 定常モニタリング

分析結果を継続的に監視する仕組み作りの基礎

🎓 学習目標

この実習を完了することで、以下の能力を習得できます：

1. アクセスログの構造を理解し、必要な情報を抽出できる
2. Python (pandas) を使ってログデータを集計・加工できる
3. 分析目的に適したデータマート（CSV 形式）を作成できる
4. Tableau Public の基本的な使い方（データ接続、グラフ作成、ダッシュボード構築）
5. Power BI Desktop の基本的な使い方（データ接続、グラフ作成、レポート構築）
6. 作成したダッシュボードから Web サイトの利用状況に関するインサイトを得る

🌟 **到達目標レベル：** 生データから BI ツールで見られる形までデータを加工し、簡単なダッシュボードを作成できるようになる。

💻 必要な環境とツール

この実習では、以下のツールを使用します。すべて無料で利用できます。

🛠️ ツール一覧

ツール名	用途	入手方法
Python (+ pandas)	アクセスログの ETL 処理	実習 1 と同じ環境（Google Colab 推奨）
Tableau Public	データ可視化、ダッシュボード作成	公式サイトから無料ダウンロード・インストール https://public.tableau.com/ja-jp/s/ (※作成物は Web 公開されます)
Power BI Desktop	データ可視化、レポート作成	公式サイトから無料ダウンロード・インストール（Wi https://powerbi.microsoft.com/ja-jp/de

💡 BI ツールについて：

- この実習では、**Tableau Public** と **Power BI Desktop** の両方の手順を説明します。どちらか一方、または両方に取り組むことができます。
- Tableau Public は Mac/Windows 両対応ですが、作成したダッシュボードは Web に公開されます（学習には最適）。
- Power BI Desktop は Windows 専用ですが、ローカル PC で作業を完結できます。

⚠️ **事前準備のお願い：**

Tableau Public / Power BI Desktop は、事前にダウンロード・インストールしておいてください。インストールには少し時間がかかる場合があります。

📂 使用するデータセット

この実習では、Web サーバーのアクセスログを使用します。アクセスログは通常、以下のような形式です。

192.168.1.1 - - [10/Oct/2025:13:55:36 +0900] "GET /index.html HTTP/1.1" 200 1024

この一行から、「いつ」「誰が」「どのページにアクセスし」「結果はどうだったか」といった情報を読み取ります。

📥 データの準備方法

方法 1：教材付属のサンプルデータを使用

実習をスムーズに進めるため、加工しやすい形式のサンプルアクセスログファイル（例: sample_access.log）を教材に同梱します。まずはこのサンプルデータを使用してください。

方法 2（オプション）：データ生成スクリプトを使用

より大量の、あるいは特定パターンのログデータで試したい場合のために、擬似的なアクセスログを生成する Python スクリプトも提供します。これを使えば、自分でデータ量や期間を調整したログファイルを生成できます。

方法 3（参考）：公開データセットの利用

学習目的で公開されているアクセスログデータセットも存在します（例：NASA httpd logs）。ただし、形式が複雑な場合があるため、まずはサンプルデータでの学習を推奨します。

📖 Part 別の学習内容

この実習は **4 つの Part** に分かれています。

Part 1　アクセスログの理解とパース

ログ形式の理解、Python での文字列処理・正規表現を用いた情報抽出（パース）。

⏱️ **推奨時間：1.5 時間**

Part 2　データ加工と集計（ETL）

抽出した情報を pandas DataFrame に格納、不要データの除去、日時データの変換、必要な指標（PV 数、UU 数など）の集計。

⏱️ **推奨時間：1.5 時間**

Part 3　データマート作成と BI ツール①（Tableau）

集計結果を CSV（データマート）に出力。Tableau Public で読み込み、基本グラフ作成、ダッシュボード構築。

⏱️ **推奨時間：2 時間**

Part 4　BI ツール②（Power BI）と考察

データマートを Power BI Desktop で読み込み、同様のダッシュボード（レポート）を作成。両ツールの違いを比較し、結果を考察。

⏱️ **推奨時間：1.5 時間**

📌 データマート設計について

Part 3 では、「なぜこの形式でデータを出力するのか？」という**データマート設計**の考え方についても詳しく解説します。

✨ 次のステップ

🎯 準備ができたら、Part 1 から始めましょう！

以下の準備が整っていることを確認してください：

- ✅ Python 実行環境（Google Colaboratory 推奨）

- ✅ Tableau Public / Power BI Desktop のインストール
- ✅ サンプルアクセスログデータ（または生成スクリプト）の準備
- ✅ ワークブックとナレーション資料の準備

📥 教材ダウンロード

(ここに各 Part のワークブック等へのリンクを配置)

📊 ビッグデータ応用教材 - 実習 2 導入ガイド

ビッグデータ技術応用
実習２ アクセスログ ETL 処理と BI ツールによる可視化
Part１（アクセスログの理解とパース）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part１ の学習目標

この Part では、アクセスログという「生データ」を読み解くスキルを学びます：

- Web サーバーアクセスログの基本的な形式（Apache Combined Log Format）を理解する
- Python を使ってテキストファイルを一行ずつ読み込む方法を学ぶ
- Python の文字列処理（.split()など）を使ってログから情報を切り出す
- 正規表現（regex）を使って、より複雑なパターンから情報を抽出する基礎を学ぶ
- 抽出した情報を pandas DataFrame に格納する方法を学ぶ

📖 Part 1 について

実習 2 の最初のステップは、コンピュータが自動生成した**アクセスログ**というテキストデータを「解読」することです。
このログには、Web サイトの利用状況を知るための貴重な情報（誰が、いつ、何を見たか）が詰まっていますが、そのままでは分析できません。
この Part では、Python の文字列処理や**正規表現**といったテクニックを使い、ログから必要な情報を抽出（**パース**）し、pandas DataFrame という扱いやすい形に変換します。これが ETL 処理の「E (Extract)」と「T (Transform)」の第一歩です。
推奨所要時間：約 1.5 時間

🛠️ タスク１：準備

📌 Task 1-1: ライブラリのインポートとデータ準備

必要なライブラリをインポートし、サンプルアクセスログファイル（sample_access.log）を準備します。

```
import pandas as pd
import re # 正規表現ライブラリをインポート
from google.colab import files # Colab 用ファイルアップロード
import io
```

```
# 1. ライブラリインポート確認
print("ライブラリの準備 OK")

# 2. サンプルログファイルを Colab にアップロード
print("¥n サンプルログファイル（sample_access.log）をアップロードしてください:")
uploaded = files.upload()

# 3. アップロードしたファイル名を取得
log_file_name = list(uploaded.keys())[0]
print(f"¥n'{log_file_name}' をアップロードしました。")

# 4. ファイルの中身を読み込んで変数に格納（最初の数行を表示）
log_content = io.BytesIO(uploaded[log_file_name]).read().decode('utf-8')
print("¥n=== ログファイルの内容（最初の 5 行）===")
print('¥n'.join(log_content.splitlines()[:5]))
```

💡 ファイルの読み込み

ここでは、ログファイルを Python で扱えるように、まずファイル全体を log_content という変数（文字列）に読み込んでいます。.decode('utf-8') は文字コードの指定です。
.splitlines()[:5] で、ファイルを行ごとに分割し、最初の 5 行だけを取り出しています。

📜 タスク 2：ログ形式の理解

📌 Task 2-1: Apache Combined Log Format の解析

今回使用するサンプルログは、一般的な **Apache Combined Log Format** に従っています。各行が以下の要素で構成されています。
IP アドレス - ユーザー名 [日時 +タイムゾーン] "リクエストメソッド URL プロトコル" ステータスコード データ転送量 "リファラ" "ユーザーエージェント"
例：
192.168.1.1 - - [10/Oct/2025:13:55:36 +0900] "GET /index.html HTTP/1.1" 200 1024 "http://example.com/" "Mozilla/5.0 ..."
この中から、今回は以下の情報を抽出することを目指します。

- **IP アドレス** (例: 192.168.1.1)
- **日時** (例: 10/Oct/2025:13:55:36 +0900)
- **リクエスト URL** (例: /index.html)
- **ステータスコード** (例: 200)

💡 各要素の意味

要素	意味
IP アドレス	アクセスしてきたコンピュータの識別番号
ユーザー名	認証ユーザー名（通常は '-' ）
日時	アクセスがあった日時
リクエスト	どのような要求か（GET=ページ表示要求）
ステータスコード	サーバーの応答（200=成功, 404=Not Found）
データ転送量	送受信したデータサイズ（byte）
リファラ	どのページから来たか
ユーザーエージェント	使用しているブラウザや OS の情報

✂️ タスク 3：文字列処理によるパース

まずは、簡単な文字列処理（.split()）を使って、ログから情報を切り出してみましょう。

📌 Task 3-1: ログ 1 行を処理してみる

ログファイルの最初の 1 行を取り出して、スペースで区切ってみます。

```
# ログの最初の 1 行を取得
first_line = log_content.splitlines()[0]
print("--- 元のログ行 ---")
print(first_line)

# スペースで分割
parts = first_line.split(' ')
print("¥n--- スペースで分割した結果（リスト） ---")
print(parts)

# IP アドレスを取得（リストの最初の要素）
ip_address = parts[0]
print(f"¥n 抽出した IP アドレス: {ip_address}")
```

💡 .split('区切り文字')

文字列を指定した「区切り文字」（ここではスペース ' '）で分割し、結果をリスト（[]で囲まれた複数の要素）として返します。
リストの要素には [0]（最初の要素）、[1]（2 番目の要素）... のようにアクセスできます。

📝 記録

分割結果のリストを見て、目的の情報（日時、URL、ステータスコード）がリストの何番目の要素に対応するか、おおよそ見当をつけてください。

📌 Task 3-2: 文字列処理で情報を抽出する関数

Task 3-1 の考え方をもとに、ログ 1 行から必要な情報（IP, 日時, URL, ステータス）を抽出する関数 parse_log_simple() を作成します。

```
def parse_log_simple(line):
    """ 文字列処理 (split) でログをパースする関数 """
    try:
        parts = line.split(' ')
        ip = parts[0]
        # 日時は '[' と '+' の間の部分を抽出
        timestamp_str = line[line.find('[')+1:line.find('+')-1]
        # リクエスト部分は " " で囲まれた中身の、スペース区切り 2 番目
        request_part = line[line.find('"')+1:line.rfind('"')]
        url = request_part.split(' ')[1]
        status_code = parts[parts.index('"') + 2] # " の 2 つ後がステータスコード

        return ip, timestamp_str, url, status_code
    except Exception as e:
        # エラーが起きた場合は None を返す (形式が違う行などへの対策)
        # print(f"Error parsing line: {line} - {e}") # デバッグ用
        return None, None, None, None

# 最初の 1 行で試してみる
parsed_data = parse_log_simple(first_line)
print("--- 文字列処理でのパース結果 ---")
print(f"IP: {parsed_data[0]}")
print(f"Timestamp: {parsed_data[1]}")
print(f"URL: {parsed_data[2]}")
print(f"Status Code: {parsed_data[3]}")
```

📌 ちょっと複雑？

line.find('[') や line.rfind('"')、parts.index('"') など、少し複雑な文字列操作を使っています。これは、スペースの数が一定でないログ形式に対応するためです。
このように、単純な split() だけでは限界があることがわかります。

🧩 タスク 4：正規表現によるパース

文字列処理の限界を超えるために、より強力な武器**「正規表現（Regular Expression, regex)」**を使います。

💡 正規表現とは？

正規表現は、文字列の中から特定の**「パターン」**に一致する部分を見つけ出すための特殊な記述方法です。
例えば、「IP アドレスっぽいパターン（数字.数字.数字.数字）」や「日時っぽいパターン（[DD/Mon/YYYY:...]）」などを定義して、ログから正確に情報を抽出できます。
最初は少し難しく感じますが、ログ解析やテキスト処理では必須のスキルです。

📌 Task 4-1: 正規表現パターンの定義

Apache Combined Log Format に合わせた正規表現パターンを定義します。（このパターンは Web 検索などで見つけられます）

```
# Apache Combined Log Format 用の正規表現パターン
# 各部分が()で囲まれており、これが抽出したい情報に対応します
log_pattern = re.compile(
    r'(¥d{1,3}¥.¥d{1,3}¥.¥d{1,3}¥.¥d{1,3})'  # 1. IP アドレス
    r'¥s-¥s-¥s'                                # 固定文字列 ' - - '
    r'¥[(.+?)¥]'                               # 2. 日時 ( []の中身 )
    r'¥s"(.+?)"'                               # 3. リクエスト全体 ( ""の中身 )
    r'¥s(¥d{3})'                               # 4. ステータスコード (3 桁の数字)
    r'¥s(¥d+|-)'                               # 5. データ転送量 (数字または '-')
    r'¥s"(.+?)"'                               # 6. リファラ ( ""の中身 )
    r'¥s"(.+?)"'                               # 7. ユーザーエージェント ( ""の中身 )
)
```

💡 正規表現の簡単な見方

- ¥d: 数字
- ¥.: ピリオド文字
- {1,3}: 1 回から 3 回繰り返し
- ¥s: スペース
- ¥[¥]: 角括弧文字

- .+?: 任意の文字列（最短一致）
- (): グループ化（抽出したい部分）

すべて覚える必要はありません。「こういうパターンで情報を抜き出せるんだな」と理解できれば OK です。

📌 Task 4-2: 正規表現で情報を抽出する関数

定義したパターンを使って、ログ 1 行から情報を抽出する関数 parse_log_regex() を作成します。

```
def parse_log_regex(line):
    """ 正規表現でログをパースする関数 """
    match = log_pattern.match(line)
    if match:
        groups = match.groups()
        # リクエスト部分をさらに分割して URL を取得
        request_parts = groups[2].split(' ')
        if len(request_parts) > 1:
            url = request_parts[1]
        else:
            url = '-' # URL が取得できない場合

        # 抽出したい情報だけを返す (IP, 日時, URL, ステータス)
        return groups[0], groups[1], url, groups[3]
    else:
        # パターンに一致しない行は None を返す
        return None, None, None, None

# 最初の 1 行で試してみる
parsed_data_regex = parse_log_regex(first_line)
print("--- 正規表現でのパース結果 ---")
print(f"IP: {parsed_data_regex[0]}")
print(f"Timestamp: {parsed_data_regex[1]}")
print(f"URL: {parsed_data_regex[2]}")
print(f"Status Code: {parsed_data_regex[3]}")
```

✓ チェックポイント 4-2

□ 文字列処理 (Task 3-2) と同じ結果が、より簡潔なコードで得られたことを確認した。

📌 Task 4-3: 全てのログ行をパースして DataFrame に格納

作成した parse_log_regex() 関数を使って、ログファイル全体を処理し、結果を pandas DataFrame に格納します。

```
# ログファイル全体を一行ずつ処理
parsed_lines = []
for line in log_content.splitlines():
    parsed = parse_log_regex(line)
    # パースに成功した行だけを追加
    if all(parsed): # 全ての要素が None でないことを確認
        parsed_lines.append(parsed)

# パース結果を DataFrame に変換
columns = ['IPAddress', 'Timestamp', 'URL', 'StatusCode']
df_parsed = pd.DataFrame(parsed_lines, columns=columns)

# 結果の確認
print(f"パースに成功した行数: {len(df_parsed):,} 行")
print("¥n=== パース結果の DataFrame (最初の 5 行) ===")
df_parsed.head()
```

✨ DataFrame 化 成功！

これで、ただのテキストだったアクセスログが、構造化された扱いやすい DataFrame になりました！ Part 2 では、この DataFrame をさらに加工していきます。

🧐 Part 1 考察問題

問 1： 文字列処理（Task 3）と正規表現（Task 4）によるパースを比べて、正規表現を使うメリットは何だと思いますか？

問 2： 今回抽出した 4 つの情報（IP, 日時, URL, ステータスコード）を使って、Part 2 以降でどのような分析ができそうですか？（自由に発想してみてください）

✅ Part 1 のまとめ

🎉 お疲れ様でした！

Part 1 では、アクセスログという非構造化データを読み解き、分析可能な形にする第一歩を踏み出しました。

📚 今日学んだこと

- アクセスログの基本的な形式
- Python でのテキストファイルの読み込みと基本的な文字列処理
- 正規表現を使ったパターンマッチングによる情報抽出（パース）
- パース結果を pandas DataFrame に格納する方法

🔜 次のステップ：Part 2

Part 2 では、作成した DataFrame をさらに加工・集計し、ETL 処理を完成させます！

- 日時データの変換
- 不要なアクセスの除外（例：画像ファイル）
- PV 数、UU 数の集計

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習 2 Part 1）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習２　アクセスログ ETL 処理と BI ツールによる可視化
Part２（データ型の整備とデータマート基礎）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part２　の学習目標

この Part では、Part 1 で抽出したデータをさらに洗練させます：

- アクセスログのタイムスタンプ（文字列）を適切な日時型（datetime）に変換する
- 日時データから、分析に役立つ特徴量（年、月、曜日、時間など）を抽出する
- URL データから不要な部分（例：パラメータ）を削除する（オプション）
- データマートの概念を理解し、BI ツールで使うためのデータセットを設計する
- 処理したデータを CSV ファイルとして保存する（ETL の Load ステップ）

📖 Part 2 について

Part 1 では、生のログテキストから情報を抽出し、DataFrame という表形式に変換しました（ETL の E と T の一部）。
Part 2 では、**Transform（変換・加工）**をさらに進めます。特に重要なのが**「データ型」**の整備です。例えば、日時の情報がただの文字列のままでは、「月ごとの集計」などができません。
データ型を整え、分析に必要な情報を追加したら、最後に BI ツールで読み込むための**「データマート」**を作成し、ファイルとして保存（**Load**）します。
推奨所要時間：約 1.5 時間

🔄 タスク 1：準備（Part 1 のデータ引き継ぎ）

📌 Task 1-1: ライブラリのインポートとデータの復元

Part 1 と同様にライブラリをインポートし、Part 1 の最後に作成した log_df DataFrame を準備します。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import datetime as dt # 日付計算のため

%matplotlib inline
sns.set(style='darkgrid')

# --- ここに Part 1 のコードを実行して log_df を作成する ---
# (もしノートブックが継続していれば不要)
# 例: parsed_data = [] ... (ログ解析ループ)
#       log_df = pd.DataFrame(parsed_data)

# log_df が存在するか確認
if 'log_df' in locals():
    print(f"Part 1 で作成したデータの行数: {len(log_df):,} 行")
    log_df.info() # データ型を再確認
else:
    print("エラー: log_df が存在しません。Part 1 のコードを再実行してください。")
```

✓ チェックポイント 1-1

☐ log_df が存在し、.info() で 'time' 列が object 型（文字列）であることを確認した。

⏱️ タスク 2：日時データの処理

アクセスログの時間情報（time 列）は現在文字列です。これを分析で使える日時型（datetime）に変換し、さらに役立つ情報を抽出します。

📌 Task 2-1: 文字列から日時型への変換

time 列の文字列（例: '10/Oct/2000:13:55:36 -0700'）を、pandas が理解できる日時型に変換します。

```
# time 列の最初の 5 件を表示して形式を確認
print("--- 変換前の time 列（文字列）---")
print(log_df['time'].head())

# 文字列を datetime 型に変換
# format='%d/%b/%Y:%H:%M:%S %z' で元の文字列の形式を指定
log_df['timestamp'] = pd.to_datetime(log_df['time'], format='%d/%b/%Y:%H:%M:%S %z', errors='coerce')

# errors='coerce' は、もし形式が違うデータがあってもエラーにせず NaT (Not a Time) にするオプション

# 変換後のデータ型と、変換に失敗したデータがないか確認
```

```
print("¥n--- 変換後の timestamp 列（datetime 型）---")
print(log_df['timestamp'].dtype)
print(f"変換失敗（NaT）の数: {log_df['timestamp'].isnull().sum()}")

# 変換結果の確認
log_df[['time', 'timestamp']].head()
```

💡 pd.to_datetime() と format

pd.to_datetime() は文字列を日時に変換する強力な関数です。
format=... は、元の文字列がどのような形式（日/月/年:時:分:秒 タイムゾーン）であるかを Python に教えるための「書式指定子」です。

- %d: 日（01-31）
- %b: 月の短縮名（Jan, Feb, ...）
- %Y: 4 桁の年（2000）
- %H: 時（00-23）
- %M: 分（00-59）
- %S: 秒（00-59）
- %z: UTC からのオフセット（+HHMM or -HHMM）

この書式指定は、様々なデータを扱う上で非常に重要です。

📝 記録

timestamp 列のデータ型が datetime64[ns, ...] のような日時型になっていること、変換失敗（NaT）の数が 0 であることを確認して記録してください。

📌 Task 2-2: 日時からの特徴量抽出

日時型に変換できたので、実習 1 と同様に .dt アクセサを使って、分析に役立つ「年」「月」「曜日」「時間」などを抽出します。

```
# 日時データから年月日時などを抽出
log_df['year'] = log_df['timestamp'].dt.year
log_df['month'] = log_df['timestamp'].dt.month
log_df['day'] = log_df['timestamp'].dt.day
log_df['hour'] = log_df['timestamp'].dt.hour
log_df['dayofweek'] = log_df['timestamp'].dt.dayofweek # 月曜日=0, 日曜日=6
log_df['dayname'] = log_df['timestamp'].dt.day_name()  # 曜日の名前

# 結果を確認（関連する列だけ表示）
print("日時関連の特徴量を抽出しました。")
log_df[['timestamp', 'year', 'month', 'day', 'hour', 'dayname']].head()
```

✓ チェックポイント 2-2

☐ year, month, day, hour, dayofweek, dayname 列が追加され、正しく抽出されていることを確認した。

🖌️ タスク 3：URL データのクリーニング（オプション）

アクセスログの url 列には、/page?query=1 のように、ページの本体（/page）の後ろにパラメータ（?query=1）が付いていることがあります。分析によっては、これらを除去してページ本体だけで集計したい場合があります。

📌 Task 3-1: URL からパラメータを除去（オプション）

url 列の ? 以降を取り除く処理を行います。（この処理は今回の分析では必須ではありません）

```
# url 列のサンプルを表示
print("--- 処理前の url 列（一部）---")
print(log_df['url'].dropna().unique()[:10]) # dropna() は欠損値があるとエラーになるため

# '?' が含まれる場合に、それ以降を除去する
# .str.split('?') で '?' で分割し、[0] で最初の部分を取得
log_df['url_cleaned'] = log_df['url'].str.split('?').str[0]

# 結果を確認
print("¥n--- 処理後の url_cleaned 列（一部）---")
log_df[['url', 'url_cleaned']].head(10)
```

💡 .str アクセサ

pandas の列が文字列の場合、.str を付けることで、文字列操作の便利な機能（split, contains, replace など）が使えます。

📝 記録（任意）

url 列と url_cleaned 列を比較し、? 以降が除去されていることを確認してください。

🧱 タスク 4：データマートの設計と作成

💡 データマートとは？

分析目的（今回は BI ダッシュボード作成）に特化して、必要な情報だけを抜き出し、使いやすい形に整えた「小さなデータセット」のことです。
元の大きなデータ（データウェアハウス）から、特定のテーマ（例：アクセス解析）に必要な部分だけを取り出した「売店（マート）」のようなイメージです。

メリット：

- データ量が小さくなり、分析ツール（BI ツール）での処理が高速になる
- 不要な情報がなくなり、分析に集中しやすくなる
- 分析者ごとに最適化されたデータを提供できる

📌 Task 4-1: データマートの設計

Part 3 & 4 で作成する BI ダッシュボードでは、「いつ」「どのページが」「どれくらい」見られたかを可視化したいと考えます。
そのために必要な列を、log_df から選びましょう。
必要な列（例）：

- timestamp: 正確な日時
- year, month, day, hour, dayname: 集計用の日時要素
- ip: アクセス元 IP（ユニークユーザー数カウント用）
- url (または url_cleaned): アクセス先ページ
- status: 結果コード（エラー分析用）

（今回は、元の time 文字列は不要と判断します）

📌 Task 4-2: データマート用 DataFrame の作成

設計に基づき、必要な列だけを選択して、新しい DataFrame access_mart_df を作成します。

```
# データマートに必要な列名をリストで定義
mart_columns = [
    'timestamp',
    'year', 'month', 'day', 'hour', 'dayname', 'dayofweek', # 日時関連
    'ip',                                                   # アクセス元
    'url', # または 'url_cleaned' を使う場合もある
    'status'                                                # 結果
]

# timestamp 列に欠損値(NaT)があれば削除（to_datetime でエラーになった行）
log_df_final = log_df.dropna(subset=['timestamp'])



# 必要な列だけを選択してデータマート用 DataFrame を作成
access_mart_df = log_df_final[mart_columns].copy() # .copy() を付けておくのが安全

# 結果を確認
print("データマート用 DataFrame を作成しました。")
access_mart_df.info()
```

access_mart_df.head()

✓ チェックポイント 4-2

□ access_mart_df が作成され、.info() で選択した列だけが含まれていることを確認した。

💾 タスク 5：データのエクスポート (Load)

ETL プロセスの最後、Load ステップです。作成したデータマートを、BI ツール（Tableau, Power BI）が読み込める CSV ファイルとして保存します。

📌 Task 5-1: CSV ファイルへの保存

.to_csv() を使って、DataFrame を CSV ファイルに書き出します。

```
# 保存するファイル名を定義
output_file_name = 'access_log_mart.csv'

# DataFrame を CSV ファイルに保存
# index=False は、DataFrame のインデックス（行番号）をファイルに含めないための指定
access_mart_df.to_csv(output_file_name, index=False)

print(f"データマートを '{output_file_name}' として保存しました。")

# Google Colab の場合、ファイルをダウンロードする
try:
    from google.colab import files
    files.download(output_file_name)
    print(f"'{output_file_name}' のダウンロードを開始します。")
except ModuleNotFoundError:
    print("ローカル環境のため、ファイルはカレントディレクトリに保存されました。")
```

✨ ETL プロセス完了！

おめでとうございます！これで、生のアクセスログから、BI ツールで分析可能なデータマートを作成する ETL プロセスが完了しました。

access_log_mart.csv ファイルが、Part 3 & 4 での可視化の「材料」となります。

🤔 Part 2 考察問題

以下の問いについて考え、記録してください。

問 1： なぜログのタイムスタンプ（文字列）を日時型(datetime)に変換する必要があるのでしょうか？ 文字列のままではできない分析は具体的に何ですか？

問 2： 今回作成したデータマートには、元のログにあった「ユーザーエージェント（ブラウザ情報）」などの列は含めませんでした。もし「どのブラウザからのアクセスが多いか？」を分析したくなった場合、どうすればよいでしょうか？

✅ Part 2 のまとめ

🎉 お疲れ様でした！

Part 2 では、ETL の「Transform」と「Load」を完了させました。

📚 今日学んだこと

- pd.to_datetime() を使った文字列から日時型への変換と書式指定
- 日時型データからの特徴量抽出（.dt アクセサ）
- 文字列データのクリーニング（.str アクセサ、オプション）
- データマートの概念と設計の考え方
- .to_csv() を使った DataFrame のファイル保存

🔜 次のステップ：Part 3 & 4

いよいよ、Part 2 で作成した access_log_mart.csv を使って、BI ツール（Tableau / Power BI）での可視化に挑戦します！
Part 3 : Tableau Public でのダッシュボード作成
Part 4 : Power BI Desktop でのダッシュボード作成

☑ 実習振り返りシート （ビッグデータ技術応用 - 実習２ Part２）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習２　アクセスログ ETL 処理と BI ツールによる可視化
Part３（Tableau Public によるダッシュボード作成）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part３　の学習目標

この Part では、BI ツール　Tableau Public　の基本操作を学びます：

- Tableau Public　を起動し、CSV データを接続（インポート）できる
- ディメンションとメジャーの違いを理解し、データ型を適切に設定できる
- 基本的なグラフ（棒グラフ、折れ線グラフ、地図）を作成できる
- 計算フィールドを使って簡単な指標（例：ユニークユーザー数）を作成できる
- 複数のグラフを組み合わせてインタラクティブなダッシュボードを作成できる
- 作成したダッシュボードを　Tableau Public　サーバーに保存・公開できる

📖 Part３　について

Part 2 で作成した CSV データマート access_log_mart.csv　を使って、いよいよ**データの「見える化」**に挑戦します。

Part 3 では、無料で利用できる高機能 BI ツール**「Tableau Public」**を使います。ドラッグ＆ドロップを中心とした直感的な操作で、Python コードを書かずに、インタラクティブ（操作可能）なグラフやダッシュボードを作成するスキルを身につけます。

推奨所要時間：約 2 時間

🛠️ タスク１：準備（Tableau Public とデータの用意）

📌 Task 1-1: Tableau Public　の準備

Tableau Public　を使用するには、アカウント作成とソフトウェアのインストールが必要です。

1. **アカウント作成:** Tableau Public　公式サイト にアクセスし、無料でアカウントを作成します。
2. **ソフトウェアインストール:** 公式サイトから「Tableau Public Desktop Edition」をダウンロードし、お使いの PC（Windows または Mac）にインストールします。
3. **起動確認:** Tableau Public Desktop を起動し、ログインできることを確認します。

⚠️ Tableau Public の注意点

Tableau Public は無料ですが、作成したワークブック（分析ファイル）は**必ず Tableau Public サーバーに公開**する必要があります。ローカルに非公開で保存することはできません。学習目的での利用に適していますが、機密データは扱わないようにしましょう。

📌 Task 1-2: データファイルの準備

Part 2 で作成・保存したデータマートファイル access_log_mart.csv を用意します。
もしファイルが見当たらない場合は、Part 2 の最後のコードを再実行して CSV ファイルを作成・ダウンロードしてください。

🔗 タスク 2：データ接続とデータ型の設定

📌 Task 2-1: Tableau Public へのデータ接続

Tableau Public Desktop を起動し、access_log_mart.csv に接続します。

1. Tableau Public を起動します。
2. 左側の「接続」メニューから「テキストファイル」を選択します。
3. ファイル選択ダイアログで、Part 2 で保存した access_log_mart.csv を選択し、「開く」をクリックします。
4. データソース画面が表示され、CSV ファイルの内容がプレビューされます。

📝 確認

データソース画面に、timestamp, year, month, ip, url などの列が表示されていることを確認してください。

📌 Task 2-2: データ型の確認と変更

Tableau が自動で認識したデータ型を確認し、必要に応じて修正します。

1. データソース画面の各列名の上にあるアイコン（例: Abc, #, カレンダー）を確認します。これが Tableau が認識したデータ型です。
2. 特に以下の列が正しいデータ型になっているか確認し、違っていればアイコンをクリックして修正します:
    - timestamp:「日付と時刻」
    - year, month, day, hour, dayofweek:「数値（整数）」
    - dayname, ip, url, status:「文字列」
3. **重要:** ip 列（IP アドレス）は、地理的役割を割り当てて地図上に表示できるように設定します。
    - ip 列の「Abc」アイコンをクリックします。
    - 「地理的役割」→「IP アドレス」を選択します。（Tableau が自動で IP アドレスから地域情報を付与してくれます）

💡 ディメンションとメジャー

Tableau では、データ項目を「ディメンション」と「メジャー」に分類します。

- **ディメンション（青色）:** 分析の切り口となる項目（例：日付、商品名、地域）。通常は不連続なカテゴリデータ。
- **メジャー（緑色）:** 集計対象となる数値（例：売上、数量、訪問者数）。通常は連続的な数値データ。

データ型の設定は、この分類にも影響します。例えば、「年」を数値ではなくカテゴリとして扱いたい場合は、「文字列」に変換することもあります。

✓ チェックポイント 2-2

☐ 各列のデータ型が適切に設定されている。
☐ ip 列に地球儀アイコン（地理的役割）が付いている。

📊 タスク 3：ワークシートでのグラフ作成

Tableau では、「ワークシート」という単位で個々のグラフを作成していきます。

📌 Task 3-1: シートへの移動と基本操作

1. データソース画面左下の「シート 1」タブをクリックして、ワークシート画面に移動します。
2. 画面構成を確認します:
    - **左側:** データペイン（ディメンションとメジャーの一覧）
    - **上部:** 列シェルフ、行シェルフ（ここに項目をドラッグ＆ドロップしてグラフを作る）
    - **中央:** ビュー（グラフが表示される領域）
    - **右側:** マークカード（色、サイズ、ラベルなどを設定）、表示形式（グラフの種類を選択）

📌 Task 3-2: 時間帯別アクセス数の折れ線グラフ

「どの時間帯にアクセスが多いか？」を可視化します。

1. データペインの「メジャー」にある「レコード数」（Tableau が自動生成した行数）を「行」シェルフにドラッグ＆ドロップします。
2. データペインの「ディメンション」にある「Hour」を「列」シェルフにドラッグ＆ドロップします。
3. 自動的に折れ線グラフが作成されます。
4. シート名を「時間帯別アクセス」などに変更します（シートタブをダブルクリック）。

📝 考察

作成したグラフを見て、アクセスが多い時間帯、少ない時間帯の傾向を記録してください。

📌 Task 3-3: 曜日別アクセス数の棒グラフ

「どの曜日にアクセスが多いか？」を可視化します。

1. 新しいワークシートを作成します（画面下部の新しいシートアイコンをクリック）。
2. 「レコード数」を「行」シェルフにドラッグ＆ドロップします。
3. 「Dayname」を「列」シェルフにドラッグ＆ドロップします。
4. 自動的に棒グラフが作成されます。
5. （オプション）曜日の並び順がアルファベット順になっている場合、「Dayname」ピルを右クリック → 並べ替え → 手動で月曜から日曜の順に並び替えます。
6. シート名を「曜日別アクセス」などに変更します。

📌 Task 3-4: アクセス元 IP アドレスの地図表示

「どの地域からのアクセスが多いか？」を地図で可視化します。

1. 新しいワークシートを作成します。
2. データペインの「ディメンション」にある「Ip」（地球儀アイコン付き）をダブルクリックします。
3. 自動的に地図が表示され、IP アドレスの位置に点がプロットされます。
4. データペインの「メジャー」にある「レコード数」を、マークカードの「サイズ」にドラッグ＆ドロップします。→ アクセス数に応じて点の大きさが変わります。
5. （オプション）マークカードの「色」で、アクセス数に応じて色を変えることもできます。
6. シート名を「アクセス元マップ」などに変更します。

⚠️ 地図表示について

Tableau Public は IP アドレスから国レベルの地理情報を付与しますが、精度は完全ではありません。また、無料版では詳細な市区町村レベルでの表示は難しい場合があります。

🖥️ タスク 4：ダッシュボードの作成

作成した複数のグラフ（ワークシート）を組み合わせて、1 つの画面（ダッシュボード）にまとめます。

📌 Task 4-1: 新規ダッシュボードの作成

1. 画面下部の新しいダッシュボードアイコン（田マーク）をクリックします。
2. ダッシュボード画面が表示されます。左側に作成したシートの一覧が表示されます。

📌 Task 4-2: シートの配置

1. 左側のシート一覧から、「時間帯別アクセス」シートを右側のダッシュボード領域にドラッグ＆ドロップします。
2. 同様に、「曜日別アクセス」シート、「アクセス元マップ」シートを、レイアウトを考えながら配置します。（例：時間帯別を上に、曜日別とマップを下に並べる）
3. 各グラフのサイズや位置を調整して、見やすいレイアウトにします。

💡 レイアウトのコツ

Tableau のダッシュボードでは、オブジェクト（シートやテキストなど）を「タイル」（自動調整）または「浮動」（自由配置）で配置できます。最初は「タイル」で配置し、必要に応じて「浮動」に切り替えると調整しやすいです。
ダッシュボードの「サイズ」を「自動」にしておくと、画面サイズに合わせて調整されます。

📌 Task 4-3: インタラクションの設定（フィルター）

ダッシュボードのグラフを連動させます。例えば、地図上のある国をクリックしたら、他のグラフもその国のデータだけを表示するようにします。

1. 「アクセス元マップ」シートを選択した状態で、シート右上の「フィルターとして使用」アイコン（漏斗マーク）をクリックします。
2. これで、地図上の点をクリックすると、他のグラフ（時間帯別、曜日別）がクリックした場所のデータのみにフィルタリングされるようになります。試してみましょう。
3. 同様に、曜日別グラフなどもフィルターとして設定できます。

✨ インタラクティブダッシュボード完成！

これで、見る人が操作しながら様々な角度からデータを探索できる、インタラクティブなダッシュボードが完成しました！

☁️ タスク 5：Tableau Public への保存と公開

作成したダッシュボードを Tableau Public サーバーに保存・公開します。

📌 Task 5-1: 保存と公開の手順

1. 「ファイル」メニュー → 「Tableau Public に保存」を選択します。
2. Tableau Public アカウントでログインします。
3. ワークブックの名前（例: Access Log Dashboard）を入力します。
4. 「保存」をクリックすると、ダッシュボードがサーバーにアップロードされ、Web ブラウザで公開されたダッシュボードが表示されます。

⚠️ 公開設定

保存されたダッシュボードは、デフォルトで **Web 上に一般公開**されます。URL を知っていれば誰でも見ることができます。個人情報や機密情報を含まないデータを使用していることを再確認してください。
（設定で「他のユーザーにワークブックのダウンロードと探索を許可する」のチェックを外すこともできます）

📝 公開 URL の記録

公開されたダッシュボードの URL を記録してください。これが提出物の一部となります。

🤔 Part 3 考察問題

作成したダッシュボードを操作しながら、以下の問いについて考え、記録してください。

問 1： 作成したダッシュボードを使って、どのような新しい「発見」や「気づき」がありましたか？（例：特定の曜日の特定の時間帯にアクセスが集中している、など）

問 2： もし、このダッシュボードに「アクセス数の多い URL ランキング」のグラフを追加するとしたら、どのように作成しますか？

✅ Part３のまとめ

🎉 お疲れ様でした！

Part 3 では、Tableau Public を使って、Python で加工したデータをインタラクティブなダッシュボードとして可視化するスキルを習得しました。

📚 今日学んだこと

- Tableau Public へのデータ接続とデータ型設定
- ディメンションとメジャーを使ったグラフ作成（折れ線、棒、地図）
- 複数のグラフを組み合わせたダッシュボードの構築
- フィルターを使ったインタラクションの設定
- Tableau Public への保存と公開

🔜 次のステップ：Part 4

Part 4 では、もう一つの代表的な BI ツールである **Power BI Desktop** を使って、同様のダッシュボード作成に挑戦します！ツールによる操作の違いなどを比較してみましょう。

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習２ Part３）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習２　アクセスログ ETL 処理と BI ツールによる可視化
Part４（Power BI Desktop によるダッシュボード作成）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part４ の学習目標

この Part では、BI ツール Power BI Desktop の基本操作を学びます：

- Power BI Desktop を起動し、CSV データを接続（取得）できる
- Power Query エディターでデータ型を確認・変更できる
- 基本的な視覚化（グラフ）を選択し、フィールドを割り当てて作成できる
- レポートビューで複数のグラフを配置し、レポート（ダッシュボード）を作成できる
- グラフ間のインタラクション（連動）を確認する
- 作成したレポートを Power BI ファイル（.pbix）として保存できる

📖 Part 4 について

Part 3 では Tableau Public を使ってダッシュボードを作成しました。
Part 4 では、もう一つの主要な BI ツールである**「Microsoft Power BI Desktop」**を使って、Part 2 で作成した access_log_mart.csv を基に、同様のアクセスログダッシュボードを作成します。
Tableau と Power BI の操作感や考え方の違いを体験することで、BI ツール全般への理解を深めることを目指します。
推奨所要時間：約 2 時間

🛠️ タスク 1：準備（Power BI Desktop とデータの用意）

📌 Task 1-1: Power BI Desktop の準備

Power BI Desktop は無料のソフトウェアですが、**Windows PC** が必要です。

1. **ソフトウェアインストール:** Microsoft Store から「Power BI Desktop」を検索してインストールするか、Power BI 公式サイト からダウンロードしてインストールします。
2. **起動確認:** Power BI Desktop を起動できることを確認します。（サインインは必須ではありませんが、Web 発行機能を使う場合は Microsoft アカウントが必要です）

⚠️ Mac ユーザーの方へ

Power BI Desktop は現在 Windows 専用です。Mac ユーザーの方は、仮想環境（Parallels Desktop など）に Windows をインストールするか、Web 版の Power BI Service（一部機能制限あり）を利用する、あるいは Part 3 の Tableau Public の学習に集中するなど、代替策をご検討ください。

📌 Task 1-2: データファイルの準備

Part 2 で作成・保存したデータマートファイル access_log_mart.csv を用意します。
もしファイルが見当たらない場合は、Part 2 の最後のコードを再実行して CSV ファイルを作成してください。

🔗 タスク 2：データの取得とデータ型の設定

📌 Task 2-1: Power BI Desktop へのデータ取得

Power BI Desktop を起動し、access_log_mart.csv を読み込みます。

1. Power BI Desktop を起動します。
2. 「ホーム」タブにある「データを取得」をクリックし、「テキスト/CSV」を選択します。（または、起動時の画面から「データを取得」）
3. ファイル選択ダイアログで、Part 2 で保存した access_log_mart.csv を選択し、「開く」をクリックします。
4. プレビューが表示されるので、内容を確認し、「読み込み」をクリックします。（文字化けなどなければ通常は「読み込み」で OK）
5. データが Power BI に読み込まれ、右側の「フィールド」ペインに列名が表示されます。

📝 確認

右側の「フィールド」ペインに、timestamp, year, month, ip, url などの列名が表示されていることを確認してください。

📌 Task 2-2: データ型の確認と変更（Power Query エディター）

Power BI では、データ型の確認・変更や簡単なデータ加工は「Power Query エディター」で行います。

1. 「ホーム」タブにある「データの変換」をクリックします。Power Query エディターが別ウィンドウで開きます。
2. 各列名にあるアイコン（例: 1.2, ABC, カレンダー）で、Power BI が認識したデータ型を確認します。
3. 特に以下の列が正しいデータ型になっているか確認し、違っていればアイコンをクリックして修正します:
    - timestamp:「日付/時刻」
    - year, month, day, hour, dayofweek:「整数」
    - dayname, ip, url, status:「テキスト」
4. **重要:** ip 列を地図で使えるように設定します。
    - ip 列を選択します。
    - 上部メニューの「変換」タブまたは「ホーム」タブにある「データ型」のドロップダウンをクリックします。

- 「テキスト」が選択されていることを確認します。（IP アドレスはテキストとして扱います）
- 次に、上部メニューの「列の追加」タブ… ではなく、Power BI Desktop 本体の「データビュー」（左側の表アイコン）に移動し、ip 列を選択後、上部「列ツール」タブの「データ カテゴリ」で「IP アドレス」を選択します。（Tableau とは異なり、Power Query ではなく本体側で設定します） ※この手順は後でグラフ作成時に再確認します。

5. データ型の修正が終わったら、Power Query エディター左上の「閉じて適用」をクリックします。

💡 Power Query エディター

Power BI のデータ加工担当です。Excel の Power Query と同じ機能で、データ型の変更、列の分割・結合、不要な行の削除など、様々なデータ整形作業を GUI で行えます。右側の「適用したステップ」で操作履歴を確認・修正できるのが特徴です。

✓ チェックポイント 2-2

☐ Power Query エディターで各列のデータ型を確認・修正し、「閉じて適用」した。

📊 タスク 3：レポートビューでの視覚化作成

Power BI では、「レポートビュー」（左側のグラフアイコン）でグラフ（視覚化）を作成し、配置していきます。

📌 Task 3-1: 画面構成と基本操作

1. 左側のアイコンで「レポートビュー」が選択されていることを確認します。
2. 画面構成を確認します:
   - **中央:** レポートキャンバス（ここにグラフを配置）
   - **右側:** 「視覚化」ペイン（グラフの種類を選択）、「フィールド」ペイン（データ項目を選択）
3. グラフ作成の基本操作：
   - ①「視覚化」ペインでグラフの種類をクリック
   - ② キャンバスに表示されたグラフの枠を選択
   - ③「フィールド」ペインから、グラフの「軸」「値」「凡例」などにデータ項目をドラッグ＆ドロップ

📌 Task 3-2: 時間帯別アクセス数の折れ線グラフ

「どの時間帯にアクセスが多いか？」を可視化します。

1. 「視覚化」ペインで「折れ線グラフ」を選択します。
2. キャンバスに表示された折れ線グラフの枠を選択します。
3. 「フィールド」ペインから「hour」を、「視覚化」ペインの「X 軸」にドラッグ＆ドロップします。
4. 「フィールド」ペインから「timestamp」（または任意の列）を、「視覚化」ペインの「Y 軸」にドラッグ＆ドロップします。
5. Y 軸にドロップした項目の集計方法が「カウント」になっていることを確認します（もし「合計」などになっていたら、項目名の横の▼をクリックして「カウント」に変更）。

6. グラフのサイズやタイトルを調整します（「視覚化」ペインの書式設定（刷毛アイコン）で設定）。

📌 Task 3-3: 曜日別アクセス数の棒グラフ

「どの曜日にアクセスが多いか？」を可視化します。

1. キャンバスの空いている場所をクリックし、「視覚化」ペインで「集合縦棒グラフ」を選択します。
2. グラフ枠を選択し、「フィールド」ペインから「dayname」を「X 軸」にドラッグします。
3. 「フィールド」ペインから「timestamp」（または任意の列）を「Y 軸」にドラッグし、集計方法を「カウント」にします。
4. （重要）X 軸の曜日がアルファベット順になっている場合、正しい曜日の順序（月曜始まりなど）に並べ替えます。
    - 左側の「データビュー」に移動します。
    - dayname 列を選択します。
    - 上部メニュー「列ツール」タブの「並べ替えの基準列」をクリックし、「dayofweek」を選択します。
    - 「レポートビュー」に戻ると、曜日が正しく並び替えられています。
5. グラフの書式を調整します。

📌 Task 3-4: アクセス元 IP アドレスの地図表示

「どの地域からのアクセスが多いか？」を地図で可視化します。

1. キャンバスの空いている場所をクリックし、「視覚化」ペインで「マップ」（地球儀アイコン）を選択します。
2. マップの枠を選択します。
3. 「フィールド」ペインから「ip」を「場所」にドラッグします。（ここで、Task 2-2 のデータカテゴリ設定が正しくできていれば、地図が表示されます。）
4. 「フィールド」ペインから「timestamp」（または任意の列）を「バブル サイズ」にドラッグし、集計方法を「カウント」にします。→ アクセス数に応じて円の大きさが変わります。
5. マップの書式（テーマなど）を調整します。

⚠️ 地図表示について

Power BI も IP アドレスから地理情報を推定しますが、精度は完全ではありません。国レベルでの表示が主になります。

💻 タスク 4：レポートの作成とインタラクション

作成した複数の視覚化（グラフ）を 1 つのページ（レポート）に配置し、連携させます。

📌 Task 4-1: レポートのレイアウト調整

1. 3 つのグラフ（折れ線、棒、地図）がキャンバス上に配置されている状態にします。
2. 各グラフの枠をドラッグして、サイズや位置を調整し、見やすいレイアウトにします。（例：時間帯別を上に、曜日別とマップを下に）

3. 必要であれば、テキストボックスを追加してタイトルなどを記述します。

📌 Task 4-2: インタラクションの確認

Power BI では、デフォルトで同じページ上のグラフが連動（クロスフィルター）するようになっています。

1. 曜日別グラフの「Monday」の棒をクリックしてみてください。
2. 他のグラフ（時間帯別、地図）が、月曜日のデータだけに絞り込まれて表示が変わることを確認します。
3. 同様に、地図上の円をクリックしても、他のグラフが連動することを確認します。
4. 再度同じ場所をクリックするか、グラフ外をクリックするとフィルターが解除されます。

✨ インタラクティブレポート完成！

特別な設定なしに、グラフ同士が連動するレポートが完成しました！

💡 インタラクションの編集

グラフを選択した状態で、上部メニュー「書式」タブの「相互作用を編集」をクリックすると、グラフ間の連動方法（フィルター、強調表示、なし）を個別に設定できます。

💾 タスク 5：Power BI ファイルの保存

作成したレポートを Power BI ファイル（.pbix）として保存します。

📌 Task 5-1: .pbix ファイルとして保存

1. 「ファイル」メニュー → 「名前を付けて保存」を選択します。
2. ファイル名（例: AccessLogReport.pbix）を入力し、保存場所を選択して「保存」をクリックします。

📌 .pbix ファイルとは？

Power BI Desktop の作業ファイルです。データ、クエリ（加工手順）、レポート（グラフやレイアウト）のすべてが含まれています。このファイルがあれば、他の Power BI Desktop 環境でも同じレポートを開くことができます。
提出物： この .pbix ファイルが、Part 4 の主要な提出物となります。

⚠️ Power BI Service への発行について

Tableau Public と異なり、Power BI Desktop で作成したレポートはローカルに保存されます。Web で共有したい場合は、Power BI Service（有償または組織アカウントが必要な場合あり）に「発行」する必要があります。今回はローカル保存までとします。

🧐 Part 4 考察問題

作成したレポートを操作しながら、以下の問いについて考え、記録してください。

問 1： Tableau Public (Part 3) と Power BI Desktop (Part 4) を使ってみて、操作感や機能で「良いな」と思った点、「難しいな」と思った点をそれぞれ挙げてください。

問 2： Power BI のレポートに、「アクセスの多い URL トップ 5」を表示するグラフを追加するとしたら、どの「視覚化」タイプを選び、どの「フィールド」を使いますか？

✅ Part４のまとめ

🎉 実習２ 完了！おめでとうございます！

Part 4 では、Power BI Desktop を使って、ETL 処理したデータをインタラクティブなレポートとして可視化するスキルを習得しました。

📚 実習２全体で学んだこと

- ETL のプロセス（Extract, Transform, Load）の実践
- Python によるテキストログデータの解析（正規表現）
- 日時データの処理と特徴量エンジニアリング
- データマートの設計と作成
- BI ツール（Tableau, Power BI）の基本操作
- インタラクティブなダッシュボード（レポート）の構築

🚀 次のステップ：実習３

実習 2 では扱いきれなかった、さらに「大規模」なデータを扱う技術に挑戦します！
実習３のテーマ： Spark による大規模データ処理

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習２ Part４）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 2　アクセスログ ETL 処理と BI ツールによる可視化
🚀 発展課題

🌟 発展課題へようこそ！

実習 2 の基本内容をマスターした皆さん、素晴らしいです！
ここでは、Python による ETL 処理の自動化・効率化や、BI ツールでのより高度な分析・表現に挑戦する課題を用意しました。難易度別に**初級・中級・上級**があります。

初級 ETL 処理の関数化とエラーハンドリング 推奨：60 分

🎯 目的

Part 1 で行ったログ解析処理を Python の関数にまとめ、再利用しやすくする。また、予期せぬログ形式に対応するための簡単なエラーハンドリングを追加する。

📝 実装内容

1. Part 1 のログ解析ロジック（正規表現マッチと辞書作成）を parse_log_line(line)という関数にまとめる。
2. 正規表現にマッチしなかった場合にエラーで止まるのではなく、None を返すように関数を修正する。
3. ログファイル読み込みループの中で、この関数を呼び出し、None が返ってきた場合はスキップ（またはエラーログとして記録）するように処理を変更する。
4. 関数化したコードを使って、再度 DataFrame を作成し、Part 1 と同じ結果になることを確認する。

💡 ヒント

- 関数の定義は def function_name(argument): で行います。
- try-except ブロックを使うと、エラーが発生してもプログラムを止めずに処理を続けることができます（例：match.group() でエラーになる場合）。
- 関数に Docstring（説明文）を付ける習慣をつけましょう。

✅ 評価ポイント

- ログ解析ロジックが正しく関数化されているか。
- 不正な形式のログ行があっても、エラーで停止せずに処理を継続できるか。
- 関数化によってコードの可読性や再利用性が向上しているか。

中級 ユーザーエージェント解析と BI での活用 推奨：90 分

🎯 目的

アクセスログに含まれるユーザーエージェント文字列を解析し、ブラウザ、OS、デバイスの種類といった情報を抽出する。

抽出した情報をデータマートに追加し、BI ツールで可視化する。

📝 実装内容

1. Python ライブラリ user-agents をインストールする（pip install pyyaml ua-parser user-agents）。
2. Part 1 の正規表現パターンを修正し、ユーザーエージェント文字列も抽出できるようにする。（ヒント: Common Log Format の最後に追加されることが多い）
3. ETL 処理の中で、抽出したユーザーエージェント文字列を user_agents.parse() 関数で解析し、ブラウザファミリー(.browser.family)、OS ファミリー(.os.family)、デバイスタイプ(.is_mobile, .is_tablet, .is_pc) などの情報を取得する。
4. 取得した情報を新しい列として DataFrame に追加する。
5. これらの列を含んだ新しいデータマート（access_log_mart_ua.csv）を作成・保存する。
6. Tableau または Power BI で新しいデータマートを読み込み、「デバイスタイプ別アクセス数」や「ブラウザ別アクセス数」などを可視化する。

💡 ヒント

- Common Log Format にユーザーエージェントが含まれる場合、正規表現の末尾に ¥"(.*)¥" ¥"(.*)¥" のようなパターンを追加する必要があるかもしれません（Referer と User-Agent）。
- user_agents.parse() は非常に便利ですが、処理に少し時間がかかる場合があります。
- .is_mobile, .is_tablet, .is_pc は True/False を返すので、「PC」「Mobile」「Tablet」のようなカテゴリ名に変換すると BI ツールで扱いやすいです。np.select() などが使えます。

✅ 評価ポイント

- user-agents ライブラリを使って正しく情報を抽出できているか。
- 新しい特徴量をデータマートに追加できているか。
- BI ツールでデバイスタイプやブラウザに関するインサイト（例：モバイルからのアクセスが多い）を可視化・考察できているか。

上級 ETL 処理のパイプライン化検討 推奨：120 分

🎯 目的

実習 2 で行った Python による ETL 処理を、定期的に自動実行するための「パイプライン」として構成する方法を調査・検討し、簡単なスクリプトを作成する。Apache Airflow [cite: 355] のようなツールについても調査する。

📝 実装内容

1. Part 1, 2 の ETL 処理コード（読み込み、解析、変換、保存）を、単一の Python スクリプトファイル（.py）にまとめる。
2. 入力ファイル名と出力ファイル名を、スクリプト実行時の引数で指定できるように修正する（argparse ライブラリなどを使用）。

3. （オプション）処理の開始・終了時刻や処理件数などをログとして出力する機能を追加する（logging ライブラリなどを使用）。
4. OS のタスクスケジューラ（Windows）や cron（Mac/Linux）を使って、このスクリプトを定期実行する方法を調査し、手順をまとめる。
5. （調査）Apache Airflow [cite: 355] のようなワークフロー管理ツールが、このような ETL 処理の自動化にどのように役立つかを調査し、利点と欠点をまとめる。

💡 ヒント

- Jupyter Notebook のコードを .py ファイルにするには、単純にセルの中身をテキストファイルにコピー＆ペーストします。import 文はファイルの先頭にまとめます。
- argparse を使うと、コマンドラインから python etl_script.py --input access.log --output mart.csv のように引数を渡せるようになります。
- Airflow については、公式ドキュメントや入門記事を読み、「DAG」「Task」「Operator」といった基本概念を理解することが目標です。

✅ 評価ポイント

- ETL 処理が単一の実行可能な Python スクリプトとしてまとめられているか。
- コマンドライン引数やログ出力など、自動化を意識した機能が実装されているか。
- タスクスケジューラや cron での定期実行方法について、具体的な手順がまとめられているか。
- Airflow のようなツールの役割と利点・欠点について、調査結果がまとめられているか。

🚀 ビッグデータ応用教材 - 実習 2 発展課題

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
導入ガイド

🎯 実習の概要

実習 1, 2 では扱いきれなかった、**PC のメモリを超えるような「大規模データ」**の処理に挑戦します。
この実習では、ビッグデータ処理の標準技術である **Apache Spark** を使い、単一マシンでの処理（pandas）と比較しながら、分散処理の基本概念と効果を体験的に学びます。

💡 この実習で身につくスキル

🔥 分散処理の必要性理解

なぜ大規模データには分散処理が必要なのかを体感する

⚡ Spark の基本操作

Spark (PySpark) を使ったデータ読み込み、変換、集計の基本

⚖️ パフォーマンス比較

pandas と Spark の処理速度の違いを定量的に比較・考察する力

⚙️ 実行環境の理解

Google Colab や Docker を使った Spark 実行環境の基礎知識

🎓 学習目標

この実習を完了することで、以下の能力を習得できます：

1. 大規模データ処理における課題（メモリ不足、処理時間）を説明できる
2. Apache Spark の基本的な仕組み（分散処理、RDD/DataFrame）を説明できる
3. Google Colaboratory (または Docker) 上で PySpark 環境をセットアップできる
4. Spark DataFrame API を使って、大規模なデータ（数 GB レベル）の読み込み、基本的な集計（カウント、平均など）を実行できる
5. 同じ処理を pandas で行った場合とのパフォーマンス差を測定し、その理由を考察できる
6. 列指向ストレージフォーマット（Parquet）の利点を理解し、利用できる

💻 必要な環境とツール

📦 実行環境（いずれかを選択）

🅰️ オプション 1：Google Colaboratory（推奨・簡単）

メリット： インストール不要、無料で Spark 環境を利用可能。

必要なもの： Google アカウント、Web ブラウザ。

セットアップ： ノートブック内で数行のコマンドを実行するだけで Spark が利用可能になります（Part 1 で解説）。

🅱️ オプション 2：ローカル環境（Docker）

メリット： より実務に近い環境、オフラインでも利用可能。

必要なもの： Docker Desktop (Windows/Mac/Linux)。

セットアップ： Docker イメージ（例：`jupyter/pyspark-notebook`）をダウンロードして実行する必要があります。手順はやや複雑になります（発展課題で詳細解説）。

👉 **初めての方は、Google Colaboratory での実施を強く推奨します。**

📚 必要な知識（復習）

- Python および pandas の基本操作（実習 1 レベル）
- ビッグデータ基礎 第 5 章：列指向ストレージ（Parquet）の概要 [cite: 372-405]
- ビッグデータ基礎 第 7 章：大規模分散処理、Hadoop、Spark の概要 [cite: 440-473]

📂 使用するデータセット

実習では、PC のメモリを超える規模（例：数 GB）の**擬似センサーデータ**を使用します。

🌡️ 擬似センサーデータ

データの概要：

- 複数のセンサーが、一定時間ごとに測定値（温度、湿度など）を記録した架空のデータ。
- データ量：数 GB（ファイルサイズは調整可能）。
- 形式：CSV 形式、および効率的な Parquet 形式。

📥 データの生成・入手方法

データ生成スクリプトを提供します。

Part 1 のワークブック内で、指定した行数（例：数千万行～1 億行）の CSV データを生成する Python コードを提供します。これにより、各々の環境で「メモリに乗らない」状況を再現しやすくなります。

```
# (Part 1 ワークブックで提供されるコードのイメージ)
import pandas as pd
import numpy as np
```

```
def generate_sensor_data(num_rows, file_path):
    # ... (データ生成ロジック) ...
    df.to_csv(file_path, index=False)
    print(f"{num_rows:,}行のデータを{file_path}に生成しました。")

# 例: 1 億行のデータを生成
# generate_sensor_data(100_000_000, 'large_sensor_data.csv')
```

生成した CSV データを、さらに効率的な Parquet 形式に変換する手順も実習に含まれます。

📖 Part 別の学習内容

この実習は **4 つの Part** に分かれています。

Part 1　分散処理の必要性と Spark 環境

なぜ分散処理が必要か？pandas の限界を体感。Colab (or Docker) で Spark 環境を準備。

⏱️ **推奨時間：1.5 時間**

Part 2　Spark DataFrame の基本操作

大規模データの読み込み (CSV, Parquet)。基本的な集計 (count, mean) とフィルタリング。

⏱️ **推奨時間：1.5 時間**

Part 3　パフォーマンス比較と考察

同じ集計処理を pandas と Spark で実行し、処理時間を比較。なぜ Spark が速いのか？

⏱️ **推奨時間：1.5 時間**

Part 4　データ集計と可視化（応用）

センサーごとの統計量算出など、より実践的な集計。結果の可視化とまとめ。

⏱️ **推奨時間：1.5 時間**

🚀 実習の進め方

各 Part は以下の流れで進めます：

1. **概念理解：** その Part で学ぶ分散処理や Spark の考え方を理解する。
2. **環境準備：** (Part 1) Spark が動作する環境を整える。
3. **データ準備：** (Part 1) 大規模な擬似データを生成する。
4. **実践演習：** ワークブックに従って PySpark コードを実行する。
5. **比較・考察：** (Part 3) pandas との性能差を確認し、理由を考える。
6. **記録：** 実行結果や考察をワークブックに記録する。

⚠️ **注意：** 大規模データを扱うため、コードの実行に時間がかかる場合があります（数分〜十分程度）。焦らず待ちましょう。

✨ 次のステップ

🎯 準備ができたら、Part 1 から始めましょう！

以下の準備が整っているか確認してください：

- ✅ Google Colaboratory（または Docker Desktop）が利用可能
- ✅ ワークブックとナレーション資料の準備
- ✅ 学習時間の確保（最初は 1.5 時間程度）

📥 教材ダウンロード

(ここに各 Part のワークブック等へのリンクを配置)

📘 **Part 1 ワークブック** …

📚 ビッグデータ応用教材 - 実習 3 導入ガイド

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
Part 1 （分散処理の必要性と Spark 環境）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 1 の学習目標

この Part では、大規模データ処理の「入口」に立ちます：

- pandas で大規模データを扱おうとすると何が起きるか（限界）を体験する
- 分散処理の基本的な考え方（分割・並列処理）を理解する
- Google Colaboratory 上で Apache Spark (PySpark) を使うための環境をセットアップできる
- Spark の基本的な構成要素である SparkSession を作成できる
- 分析対象となる大規模な擬似センサーデータを生成する

📖 Part 1 について

実習 1, 2 で皆さんが使いこなしてきた pandas は、小〜中規模データ（数万〜数百万行程度）の分析には非常に強力なツールです。
しかし、データが PC のメモリに収まらないほど巨大（数 GB〜TB）になると、pandas だけでは太刀打ちできません。
Part 1 では、まずその「限界」を体感し、「なぜ分散処理が必要なのか」を理解することから始めます。そして、そのための強力なツールである Apache Spark を使う準備を整えます。
推奨所要時間：約 1.5 時間

💣 タスク 1：pandas の限界を知る（メモリ不足体験）

最初に、pandas で「メモリに乗り切らない」状況を意図的に作り出し、何が起きるかを見てみましょう。

📌 Task 1-1: 準備とメモリ確認

必要なライブラリをインポートし、現在の環境（Colab）で利用可能なメモリ量を確認します。

```
import pandas as pd
import numpy as np
import time # 処理時間計測用
```

```
# Colab 環境のメモリ情報を表示
!free -h
# または
# from psutil import virtual_memory
# ram_gb = virtual_memory().total / 1e9
# print(f'Available RAM: {ram_gb:.2f} GB')
```

📝 メモリ容量の記録

あなたの Colab 環境で利用可能なメモリ容量（Total または Available）を記録してください。

📌 Task 1-2: 巨大な DataFrame を生成してみる

利用可能なメモリを超える可能性のある、非常に大きな DataFrame を pandas で生成してみます。
⚠️ **注意：** このコードは意図的にメモリ不足エラーを引き起こす可能性があります。環境によっては Colab ランタイムがクラッシュすることがあります。

```
# 生成する行数を設定（環境に合わせて調整）
# Colab 無料枠（約 12GB メモリ）の場合、1 億行（100_000_000）程度でメモリ不足になる可能性
num_rows_pandas = 100_000_000
num_cols = 5

print(f"{num_rows_pandas:,} 行 x {num_cols} 列 の DataFrame 生成を試みます...")
print(f"推定メモリ使用量: 約 {(num_rows_pandas * num_cols * 8) / 1e9:.2f} GB") # 1 要素 8 バイトと仮定

try:
    start_time = time.time()
    # ランダムな数値で巨大な DataFrame を作成
    huge_df = pd.DataFrame(np.random.randn(num_rows_pandas, num_cols),
                           columns=[f'col_{i}' for i in range(num_cols)])
    end_time = time.time()
    print(f"生成成功！ ({end_time - start_time:.2f} 秒)")
    # print(huge_df.info(memory_usage='deep')) # メモリ使用量を表示（これもエラーになる可能性）
    del huge_df # メモリ解放のため削除
except MemoryError as e:
    print("¥n💥 メモリ不足エラーが発生しました！ 💥")
    print(e)
except Exception as e:
    print(f"¥n⚠️ その他のエラーが発生しました: {e}")
```

print("¥n(メモリ不足が発生した場合) これが pandas の限界です。")

💡 行数の調整

もしエラーが発生しない場合は、num_rows_pandas の値を増やして再実行してみてください（例: 2 億行）。逆に Colab のメモリが少ない場合は、値を減らして試してください。
目的はエラーを出すことです！

📝 結果の記録

実行結果はどうなりましたか？メモリ不足エラーが発生しましたか？もし生成できた場合、何秒かかりましたか？

💡 タスク 2：分散処理と Spark の概要

🔥 なぜメモリ不足になるのか？

Pandas は基本的に、データを**すべてメモリ上に展開**して処理しようとします。そのため、データサイズが物理メモリを超えると処理できなくなります。

✨ 解決策：分散処理

データを小さな塊に**分割（Partition）**し、それらを複数のコンピュータ（または CPU コア）で**手分けして並列処理**すれば、メモリに乗り切らないデータも扱えます。

🚀 Apache Spark

この「分散処理」を簡単かつ高速に行うためのフレームワーク（ソフトウェア基盤）が **Apache Spark** です。特に大規模データの処理・分析において、世界標準の技術となっています。

📌 Task 2-1: Spark の構成要素（簡単に）

Spark を使う上で知っておきたい、いくつかの用語があります。

- **SparkSession:** Spark アプリケーションのエントリーポイント（入口）。まずこれを作成します。
- **Driver:** Spark の処理全体を管理するプログラム（通常、皆さんがコードを実行している場所）。
- **Executor:** 実際に分割されたデータ（タスク）を処理するワーカー。複数存在し、並列処理を行います。
- **RDD (Resilient Distributed Datasets):** Spark の基本的なデータ構造。分割可能で、障害に強い（Resilient）。
- **DataFrame / Dataset:** RDD をより扱いやすくした、表形式のデータ構造（pandas DataFrame に似ています）。実習 3 では主にこれを使います。

📌 詳細

これらの詳細は基礎教材 第 7 章 [cite: 440-473] を参照してください。今は「Spark はデータを分割して並列処理するもの」「DataFrame で操作する」というイメージを持てれば OK です。

⚙️ タスク 3：Spark 環境のセットアップ（Google Colab)

Google Colaboratory 上で Spark を使うための準備を行います。

📌 Task 3-1: PySpark のインストール

Colab にはデフォルトで Spark が入っていないため、pip を使って PySpark（Python 用 Spark ライブラリ）をインストールします。

```
# PySpark をインストール
!pip install pyspark -q # -q はログ出力を抑制するオプション
```

✓ チェックポイント 3-1

□ エラーなくインストールが完了した。（Successfully installed ... のようなメッセージが表示される）

📌 Task 3-2: SparkSession の作成

Spark を使うための「入口」である SparkSession を作成します。

```
from pyspark.sql import SparkSession

# SparkSession を作成（または既存のものを取得)
spark = SparkSession.builder.appName("BigDataEx3_Part1").getOrCreate()

print("SparkSession の準備ができました！")
spark # spark オブジェクトの内容を表示してみる
```

💡 SparkSession

.appName(...) で、この Spark アプリケーションに名前を付けています。
.getOrCreate() は、もし既存の Session があればそれを使い、なければ新しく作成するという便利なメソッドです。
通常、Spark を使うノートブックでは、最初にこのコードを実行します。

✨ Spark 利用準備完了！

これで、このノートブック上で Spark の機能が使えるようになりました！

💾 タスク 4：大規模データの生成

実習で使うための、メモリに乗り切らない可能性のある大規模な擬似センサーデータを CSV ファイルとして生成します。

📌 Task 4-1: データ生成関数の定義

センサーID、タイムスタンプ、測定値（複数）を持つデータフレームを生成する関数を定義します。

```
import pandas as pd
import numpy as np
import random
from datetime import datetime, timedelta

def generate_sensor_data(num_rows, file_path):
    """ 指定された行数の擬似センサーデータを生成し、CSV ファイルに保存する """
    print(f"{num_rows:,} 行のデータ生成を開始します...")
    start_gen_time = time.time()

    num_sensors = 100 # センサーの数
    start_date = datetime(2023, 1, 1)

    data = {
        'sensor_id': [f'sensor_{random.randint(1, num_sensors)}' for _ in
range(num_rows)],
        'timestamp': [start_date + timedelta(seconds=i) for i in range(num_rows)],
        'temperature': np.random.normal(25, 5, num_rows), # 平均 25 度, 標準偏差 5
        'humidity': np.random.normal(60, 10, num_rows),    # 平均 60%, 標準偏差 10
        'pressure': np.random.normal(1013, 2, num_rows)   # 平均 1013hPa, 標準偏差 2
    }
    df = pd.DataFrame(data)

    # タイムスタンプを文字列に変換（CSV 保存のため）
    df['timestamp'] = df['timestamp'].dt.strftime('%Y-%m-%d %H:%M:%S')

    print("DataFrame 生成完了。CSV ファイルへの書き込みを開始します...")
    df.to_csv(file_path, index=False)

    end_gen_time = time.time()
    print(f"完了！ {file_path} にデータを保存しました。({end_gen_time - start_gen_time:.2f}
秒)")
```

```
# メモリ解放
del df
import gc
gc.collect()
```

📌 Task 4-2: データ生成の実行

生成する行数を指定して、関数を実行します。

⚠️ **注意：** 行数が多いと**実行に非常に時間がかかり（数分〜十分以上）、Colab のディスク容量を圧迫します。**まずは少ない行数（例：100 万行）で試してから、徐々に増やしてください。

推奨行数（Colab 無料枠向け）：**5000 万行（50_000_000）** → 約 2GB 程度の CSV ファイルになります。

```
# --- 生成する行数を設定 ---
NUM_ROWS_TO_GENERATE = 50_000_000 # ← 環境に応じて調整してください
OUTPUT_CSV_PATH = 'large_sensor_data.csv'

# ディスク空き容量を確認（念のため）
!df -h .

# データ生成を実行
generate_sensor_data(NUM_ROWS_TO_GENERATE, OUTPUT_CSV_PATH)

# 生成されたファイルのサイズを確認
!ls -lh {OUTPUT_CSV_PATH}
```

⚠️ 時間と容量に関する注意

- **時間：** 5000 万行の生成には Colab でも数分かかります。その間、他の作業はできません。
- **容量：** Colab 無料枠のディスク容量（数十 GB）は有限です。生成するファイルサイズが容量を超えないように NUM_ROWS_TO_GENERATE を調整してください。
- もし容量不足になった場合は、「ランタイム」→「ランタイムの接続解除と削除」を行い、ノートブックを開き直して行数を減らして再試行してください。

📝 生成結果の記録

生成した行数、かかった時間、生成された CSV ファイルのサイズを記録してください。

🤔 Part 1 考察問題

問 1： Task 1-2 で pandas で巨大な DataFrame を作成しようとした際、なぜメモリ不足エラーが起きた（または非常に時間がかかった）のか、その理由を「メモリ」と「分散処理」という言葉を使って説明してください。

問 2： Task 4 で大規模な CSV データを生成しました。このデータを Spark で処理する場合、Spark は内部でどのような工夫をしてメモリ不足を防いでいると予想されますか？（ヒント：分散処理の考え方）

✅ Part 1 のまとめ

🎉 お疲れ様でした！

Part 1 では、大規模データ処理の「必要性」を体感し、そのための武器である Spark を使う準備が整いました。

📚 今日学んだこと

- Pandas のメモリ限界と分散処理の必要性
- Apache Spark の基本的な役割
- Google Colab での PySpark 環境セットアップ
- SparkSession の作成
- 大規模な擬似データの生成方法

➡️ 次のステップ：Part 2

いよいよ Spark を使って、今日生成した巨大な CSV データ（large_sensor_data.csv）を読み込み、DataFrame として基本的な操作を行う方法を学びます！

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習３ Part１）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
Part 2 （Spark DataFrame の基本操作）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 2 の学習目標

この Part では、Spark を使って大規模データを操作する基本を学びます：

- Spark を使って大規模 CSV ファイルを DataFrame として読み込むことができる
- DataFrame のスキーマ（列名とデータ型）を確認・理解できる
- .show(), .count() など、基本的な Action を使いこなせる
- .select(), .filter() (.where()) など、基本的な Transformation を使える
- .groupBy().agg() を使って簡単な集計（カウント、平均など）ができる
- DataFrame を効率的な Parquet 形式で保存・読み込みできる
- Spark の「遅延評価」の概念を理解する

📖 Part 2 について

Part 1 では Spark 環境を準備し、分析対象となる巨大な CSV データ（large_sensor_data.csv）を生成しました。
Part 2 では、いよいよ Spark を使ってこの**巨大 CSV ファイルを読み込み**、pandas DataFrame によく似た **Spark DataFrame** として操作する方法を学びます。
pandas との違い（特に「遅延評価」）も意識しながら、Spark でのデータ操作の基本をマスターしましょう。また、ビッグデータ処理で重要な **Parquet 形式**への変換も行います。
推奨所要時間：約 1.5 時間

🔄 タスク 1：準備（SparkSession の確認）

📌 Task 1-1: SparkSession の起動確認

まず、Part 1 で作成した SparkSession が有効か確認します。（ノートブックを開き直した場合などは、再度作成が必要です）

```
from pyspark.sql import SparkSession
```

```
# SparkSession を取得（なければ作成）
try:
    spark # spark 変数が存在するか確認
    print("既存の SparkSession が見つかりました。")
except NameError:
    print("SparkSession を作成します...")
    spark = SparkSession.builder.appName("BigDataEx3_Part2").getOrCreate()
    print("SparkSession の準備ができました！")

spark # 確認のため表示
```

📂 タスク 2：Spark による大規模 CSV データの読み込み

Part 1 で生成した巨大 CSV ファイル（large_sensor_data.csv）を Spark DataFrame として読み込みます。

📌 Task 2-1: CSV ファイルの読み込み

spark.read.csv() を使います。いくつかの重要なオプションを指定します。

```
import time

# 読み込む CSV ファイルのパス
csv_file_path = 'large_sensor_data.csv'

print(f"'{csv_file_path}' の読み込みを開始します...")
start_time = time.time()

# Spark DataFrame として CSV を読み込む
sensor_df = spark.read.csv(csv_file_path,
                           header=True,          # 1 行目をヘッダーとして使用
                           inferSchema=True      # データ型を自動推論（※大規模データでは注意）
                          )

# ↑ この時点ではまだ実際の読み込みは実行されない（遅延評価）

# 最初の数行を表示する Action を実行して、初めて読み込みが開始される
sensor_df.show(5) # .show() は pandas の .head() に相当

end_time = time.time()
print(f"¥nCSV ファイルの読み込みと最初の 5 行表示 完了（{end_time - start_time:.2f} 秒）")
```

💡 spark.read.csv() のオプション

- **header=True:** CSV ファイルの 1 行目が列名であることを Spark に伝えます。
- **inferSchema=True:** Spark がデータの中身を見て、自動でデータ型（数値、文字列など）を推測します。便利ですが、**非常に大きなファイルの場合、この推測処理自体に時間がかかる**ことがあります。実務では、スキーマ（データ型定義）を明示的に指定することが推奨されます（発展課題）。

⚠️ 実行時間について

数 GB の CSV ファイルを読み込むため、.show(5) の実行には**数分**かかる場合があります。気長に待ちましょう。これが Spark の処理です。

📌 Task 2-2: スキーマ（データ型）の確認

Spark が推測したデータ型（スキーマ）を確認します。.printSchema() を使います。

```
# DataFrame のスキーマ（列名とデータ型）を表示
print("=== DataFrame スキーマ ===")
sensor_df.printSchema()
```

📝 スキーマの記録

各列がどのようなデータ型として認識されているか記録してください。特に timestamp, temperature などが期待通りか確認しましょう。

📌 inferSchema の限界

timestamp 列が string（文字列）と推論されていませんか？ inferSchema=True は完璧ではありません。特に日付/時刻形式は正しく認識されないことが多いです。これも後で修正します。
数値列（temperature など）が double（倍精度浮動小数点数）になっているのは適切です。

📌 Task 2-3: 行数のカウント

DataFrame に何行のデータが含まれているか確認します。.count() を使います。

```
print("行数のカウントを開始します...")
start_time = time.time()

# DataFrame の総行数をカウント（これも Action）
total_rows = sensor_df.count()

end_time = time.time()
print(f"総行数: {total_rows:,} 行 ({end_time - start_time:.2f} 秒)")
```

⚠️ 実行時間について

.count() は全データをスキャンするため、**実行に時間がかかります（数分）**。これが大規模データ処理の現実です。

📝 行数の記録

カウントされた総行数と、かかった時間を記録してください。Part 1 で生成した行数と一致していますか？

⚙️ タスク 3：基本的な DataFrame 操作（Transformation & Action）

💡 Transformation と Action

Spark DataFrame の操作は、大きく 2 種類に分けられます。

- **Transformation (変換):** 元の DataFrame から新しい DataFrame を作成する操作（例: select, filter, groupBy）。実行を指示しても、**すぐには計算されません（遅延評価）**。
- **Action (実行):** 実際に計算を実行し、結果を返す操作（例: show, count, collect, save）。Action が実行された時に、それまでの Transformation がまとめて処理されます。

この「遅延評価」が Spark の重要な特徴です。

📌 Task 3-1: 列の選択（.select() - Transformation）

必要な列だけを選択して、新しい DataFrame を作成します。

```
# sensor_id と temperature 列だけを選択
selected_df = sensor_df.select("sensor_id", "temperature")

# この時点ではまだ計算は実行されていない

# .show() Action を実行して初めて select が処理される
print("=== 選択した列の表示 ===")
selected_df.show(5)
```

📌 Task 3-2: 条件によるフィルタリング（.filter() / .where() - Transformation）

特定の条件に合う行だけを抽出します。pandas の df[(条件)] に似ています。

```
# temperature が 30 より大きいデータだけを抽出
high_temp_df = sensor_df.filter(sensor_df.temperature > 30)
# または .where("temperature > 30") でも可

# この時点ではまだ計算は実行されていない

# .count() Action を実行して初めて filter が処理される
```

```
print("=== 30 度より大きいデータ数をカウント ===")
start_time = time.time()
count_high_temp = high_temp_df.count()
end_time = time.time()
print(f"該当件数: {count_high_temp:,} 件 ({end_time - start_time:.2f} 秒)")
```

📝 フィルタリング結果の記録

30 度より大きいデータは何件ありましたか？カウントにかかった時間は？

📌 Task 3-3: 簡単な集計 (.groupBy().agg() - Transformation)

センサーID ごとに、測定回数と平均温度を計算してみましょう。pandas の groupby と似ています。

```
from pyspark.sql.functions import count, avg # 集計関数をインポート

# sensor_id ごとにグループ化し、集計 (agg) する
agg_df = sensor_df.groupBy("sensor_id").agg(
    count("*").alias("record_count"),        # 各センサーのレコード数
    avg("temperature").alias("avg_temp")     # 各センサーの平均温度
)

# この時点ではまだ計算は実行されていない

# .show() Action を実行して初めて groupBy と agg が処理される
print("=== センサーごとの集計結果 ===")
agg_df.show(10)
```

💡 集計関数とエイリアス

pyspark.sql.functions から count, avg, sum, min, max などの集計関数をインポートして使います。
.alias("名前") を使うと、集計結果の列に分かりやすい名前を付けることができます。

✨ タスク 4：Parquet 形式への変換と読み込み

💡 Parquet とは？

ビッグデータ処理で標準的に使われる**「列指向」**のファイルフォーマットです。

CSV との違いとメリット：

- **効率的な圧縮:** 同じ列のデータは似ていることが多いため、CSV よりはるかに圧縮効率が高い → ファイルサイズが小さい！
- **列単位での読み込み:** 分析に必要な列だけを効率的に読み込める → 読み込みが速い！

- **スキーマ情報保持:** データ型情報がファイル内に埋め込まれている → inferSchema が不要で高速。

👉 大規模データを扱う場合、CSV より Parquet を使うのが常識です。

📌 Task 4-1: DataFrame を Parquet 形式で保存

Spark DataFrame を Parquet 形式で保存します。.write.parquet() を使います。
⚠️ **注意：** 書き込みにも時間がかかり（数分）、ディスク容量を使います。

```
# 保存先ディレクトリパス
parquet_output_path = 'sensor_data.parquet'

print(f"DataFrame を Parquet 形式で '{parquet_output_path}' に保存します...")
start_time = time.time()

# DataFrame を Parquet 形式で書き出す（Action）
# mode("overwrite") は、もし既に存在したら上書きするオプション
sensor_df.write.mode("overwrite").parquet(parquet_output_path)

end_time = time.time()
print(f"Parquet ファイルの保存完了（{end_time - start_time:.2f} 秒）")

# 保存されたファイル（ディレクトリ）のサイズを確認
!ls -lh {parquet_output_path}
```

📌 Parquet はディレクトリ？

Spark で Parquet を書き出すと、指定したパスはファイルではなく「ディレクトリ」として作成され、その中にデータファイル（分割されている場合あり）やメタデータが格納されます。これは分散処理のための仕組みです。

📝 Parquet 保存結果の記録

Parquet ファイルの保存にかかった時間と、生成されたディレクトリのサイズを記録してください。元の CSV ファイルサイズと比較してみましょう。

📌 Task 4-2: Parquet ファイルの読み込み

保存した Parquet ファイルを Spark DataFrame として読み込みます。spark.read.parquet() を使います。

```
print("Parquet ファイルの読み込みを開始します...")
start_time = time.time()

# Spark DataFrame として Parquet を読み込む
```

```
sensor_df_parquet = spark.read.parquet(parquet_output_path)

# ↑ この時点ではまだ読み込まれない（遅延評価）

# スキーマを表示（Parquet はスキーマ情報を持つので inferSchema 不要）
print("¥n=== Parquet DataFrame スキーマ ===")
sensor_df_parquet.printSchema()

# 最初の数行を表示（ここで初めて読み込み実行）
sensor_df_parquet.show(5)

end_time = time.time()
print(f"¥nParquet ファイルの読み込みと最初の 5 行表示 完了（{end_time - start_time:.2f} 秒)")
```

💡 Parquet 読み込みの速さ

CSV 読み込み(Task 2-1)と比べて、.show(5) がはるかに高速に完了しませんでしたか？
Parquet はスキーマ情報を持っているため inferSchema が不要なこと、列指向で効率的に読み込めることが理由です。
また、printSchema() で timestamp が string ではなく、正しく（あるいはより適切な型に）認識されている可能性もあります。

📝 Parquet 読み込み結果の記録

Parquet ファイルの読み込みと表示にかかった時間を記録してください。CSV の場合(Task 2-1)と比較してどうでしたか？ スキーマは正しく認識されていますか？

🤔 Part 2 考察問題

問 1： Spark の「遅延評価」とはどのような仕組みですか？ なぜ Spark はそのような仕組みを採用しているのでしょうか？（メリットを考えてみましょう）
問 2： CSV と Parquet で、ファイルサイズと読み込み速度にどのような違いがありましたか？ なぜ Parquet の方が効率的なのでしょうか？（「列指向」「圧縮」「スキーマ」という言葉を使って説明してみましょう）

✅ Part 2 のまとめ

🎉 お疲れ様でした！

Part 2 では、Spark DataFrame の基本的な操作と、効率的なデータフォーマットである Parquet について学びました。

📚 今日学んだこと

- spark.read.csv() / spark.read.parquet() によるデータ読み込み
- スキーマ確認（.printSchema()）と基本 Action（.show(), .count()）
- 基本 Transformation（.select(), .filter(), .groupBy().agg()）
- Spark の遅延評価の概念
- Parquet 形式での保存（.write.parquet()）とその利点

🔜 次のステップ：Part 3

いよいよ本番！同じ処理を pandas と Spark で実行し、その**処理速度**を比較します。Spark がなぜ大規模データ処理に適しているのかを、タイムで実感しましょう！

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習３ Part２）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
Part 3 （パフォーマンス比較と考察）ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part 3 の学習目標

この Part では、Spark の「速さ」を体感し、その理由を探ります：

- 同じデータ集計処理を pandas と Spark (PySpark) の両方で実行できる
- %%time マジックコマンド等を使って、各処理の実行時間を計測できる
- 計測結果を比較し、Spark が大規模データ処理において高速である理由を考察できる
- Spark の分散処理がパフォーマンスに与える影響を理解する

✨ 到達目標レベル

Part 3 を完了すると、「なぜ大規模データ処理に Spark が使われるのか」を、自身の体験と言葉で説明できるようになります。

📖 Part 3 について

Part 1 で pandas の限界を目の当たりにし、Part 2 で Spark DataFrame の基本操作と Parquet 形式を学びました。
Part 3 では、いよいよ**「pandas vs Spark」の直接対決**です！
Part 2 で準備した大規模な Parquet データを使って、同じ集計処理（センサーごとの統計量算出）を pandas と Spark の両方で実行し、その**処理時間を計測・比較**します。
この比較を通して、Spark（分散処理）がなぜ大規模データ処理に不可欠なのか、その**圧倒的なパフォーマンス**を体感しましょう。
推奨所要時間：約 1.5 時間

🔄 タスク 1：準備（環境とデータの確認）

📌 Task 1-1: ライブラリのインポートと SparkSession の確認

必要なライブラリをインポートし、SparkSession が有効か確認します。

```
import pandas as pd
import numpy as np
import time
from pyspark.sql import SparkSession
import pyspark.sql.functions as F # Spark の関数群を F としてインポート

# SparkSession を取得（なければ作成）
try:
    spark
    print("既存の SparkSession が見つかりました。")
except NameError:
    print("SparkSession を作成します...")
    spark = SparkSession.builder.appName("BigDataEx3_Part3").getOrCreate()
    print("SparkSession の準備ができました！")

spark
```

📌 Task 1-2: Parquet データのパス確認

Part 2 で保存した Parquet データのパス（ディレクトリ名）を確認します。

```
# Part 2 で保存した Parquet ディレクトリのパス
parquet_input_path = 'sensor_data.parquet'

# 存在するか確認（エラーが出なければ OK）
!ls -d {parquet_input_path}
```

✓ チェックポイント 1-2

☐ SparkSession が利用可能である。
☐ Parquet データのディレクトリが存在する。

🐢 タスク 2：Pandas での集計処理と時間計測

まず、比較対象として、pandas で大規模データを読み込み、集計処理を実行してみます。

📌 Task 2-1: Pandas で Parquet データを読み込む

pandas にも Parquet ファイルを読み込む機能があります。ただし、データ全体をメモリに読み込もうとします。
⚠️ **注意：** Part 1 と同様、データサイズが大きい場合は**メモリ不足エラー**が発生する可能性があります！

```
%%time
# ↑ Jupyter/Colab のマジックコマンド。セル全体の実行時間を計測
```

```
print(f"Pandas で '{parquet_input_path}' の読み込みを開始します...")
try:
    # Parquet ファイルを Pandas DataFrame として読み込む
    # PyArrow エンジンを使うのが一般的 (なければ pip install pyarrow)
    pandas_df = pd.read_parquet(parquet_input_path, engine='pyarrow')

    print(f"読み込み成功！ DataFrame の形状: {pandas_df.shape}")
    # メモリ使用量を確認 (推定)
    print(f"推定メモリ使用量: {pandas_df.memory_usage(deep=True).sum() / 1e9:.2f} GB")

except MemoryError as e:
    print("¥n💥 メモリ不足エラーが発生しました！ 💥")
    print("    Pandas ではこのサイズのデータをメモリに読み込めませんでした。")
    pandas_df = None # エラーの場合は変数を None にする
except Exception as e:
    print(f"¥n⚠️ その他のエラーが発生しました: {e}")
    pandas_df = None
```

💡 %%time マジックコマンド

Jupyter Notebook や Google Colab で、セルの先頭に %%time と書くと、そのセルの実行にかかった時間（Wall time など）を表示してくれます。パフォーマンス比較に非常に便利です。
engine='pyarrow' は Parquet ファイルを効率的に読み込むためのライブラリ指定です。Colab には通常入っていますが、ローカル環境では pip install pyarrow が必要な場合があります。

📝 Pandas 読み込み結果の記録

Parquet ファイルの読み込みは成功しましたか？ それともメモリ不足になりましたか？ 成功した場合、かかった時間と推定メモリ使用量を記録してください。

📌 Task 2-2: Pandas での集計処理

もし Task 2-1 で DataFrame が読み込めた場合のみ、センサーID ごとの統計量（レコード数、温度の平均/最大/最小）を集計し、その時間を計測します。

```
%%time

# pandas_df が正常に読み込めた場合のみ実行
if pandas_df is not None:
    print("Pandas での集計処理を開始します...")
```

```
    # sensor_id ごとにグループ化し、統計量を計算
    pandas_agg_result = pandas_df.groupby('sensor_id').agg(
        record_count=('sensor_id', 'count'), # レコード数
        avg_temp=('temperature', 'mean'),    # 平均温度
        max_temp=('temperature', 'max'),     # 最大温度
        min_temp=('temperature', 'min')      # 最小温度
    )

    print("集計完了！ 結果の最初の 5 行:")
    print(pandas_agg_result.head())

else:
    print("Pandas DataFrame が読み込めていないため、集計処理をスキップします。")
```

⚠️ メモリ不足の場合

Task 2-1 でメモリ不足になった場合、このセルは実行されません。「Pandas ではこのサイズのデータで集計処理を行うこと自体が困難である」という結論になります。

📝 Pandas 集計結果の記録

集計処理は実行できましたか？ 実行できた場合、かかった時間を記録してください。

🚀 タスク 3：Spark での集計処理と時間計測

次に、全く同じ処理を Spark (PySpark) で実行し、時間を計測します。

📌 Task 3-1: Spark で Parquet データを読み込む

Part 2 のおさらいです。Spark で Parquet ファイルを読み込みます。今回は処理時間も計測します。

```
%%time

print(f"Spark で '{parquet_input_path}' の読み込み (定義) を開始します...")

# Spark DataFrame として Parquet を読み込む (Transformation)
sensor_df_spark = spark.read.parquet(parquet_input_path)

# スキーマ表示 (これはすぐ返ってくる)
sensor_df_spark.printSchema()

print("¥nDataFrame の定義完了。")
```

```
# 注意：この時点ではまだ実データはほとんど読み込まれていない（遅延評価のため）
```

💡 読み込み定義の時間

このセルの実行時間は非常に短いはずです。なぜなら、Spark は遅延評価のため、まだ実際のデータ読み込みを行っていないからです。「こういうデータを読み込むぞ」という計画を立てただけです。

📌 Task 3-2: Spark での集計処理

pandas と全く同じ集計（センサーID ごとの統計量）を Spark DataFrame API で記述します。

```
%%time
# ↑ セル全体の時間を計測

print("Spark での集計処理を開始します...")

# sensor_id ごとにグループ化し、統計量を計算（Transformation）
spark_agg_result_df = sensor_df_spark.groupBy("sensor_id").agg(
    F.count("*").alias("record_count"),          # レコード数
    F.avg("temperature").alias("avg_temp"),     # 平均温度
    F.max("temperature").alias("max_temp"),      # 最大温度
    F.min("temperature").alias("min_temp")       # 最小温度
)

# .show() Action を実行して初めて、読み込みから集計までが一気に実行される
spark_agg_result_df.show(5)

print("¥n 集計処理（と表示）完了！")
```

⚠️ 実行時間について

このセルは、Parquet ファイルの読み込みから GroupBy、集計、そして結果の表示まで、すべての計算を実行するため、**時間がかかります（数秒〜数分）**。しかし、pandas でメモリ不足になったデータでも、Spark なら処理できるはずです。

📝 Spark 集計結果の記録

集計処理（%%time で計測された Wall time）にかかった時間を記録してください。

⚖️ タスク 4：パフォーマンス比較と考察

計測結果を比較し、なぜ Spark の方が大規模データ処理において有利なのかを考察します。

📌 Task 4-1: 処理時間の比較

Task 2-2（pandas 集計時間）と Task 3-2（Spark 集計時間）で記録した時間を比較してみましょう。

📝 時間比較の記録

処理	Pandas 実行時間	Spark 実行時間
センサー別統計量集計		

気づいたこと：

📌 Task 4-2: なぜ Spark は速いのか？（考察）

Spark が大規模データ処理において pandas より有利な理由を、これまでの学習内容（分散処理、メモリ管理、遅延評価など）を踏まえて考察してみましょう。

🤔 考察のヒント

- **分散処理：** データや処理を複数の Executor に「分散」できることの利点は？
- **メモリ管理：** 全データをメモリに読み込む pandas と、分散して処理する Spark の違いは？
- **遅延評価：** Action が呼ばれるまで処理を遅らせることで、どのような最適化が可能になるか？
- **データ形式：** Parquet 形式を使うことの利点は？

📝 あなたの考察

✅ Part３のまとめ

🎉 お疲れ様でした！

Part 3 では、pandas と Spark のパフォーマンスを比較することで、Spark が大規模データ処理にもたらす価値を体感しました。

📚 今日学んだこと

- %%time を使った処理時間の計測方法
- 大規模データに対する pandas の限界（メモリ不足 or 長時間）
- 同じ処理を Spark で実行した場合の速度と完遂能力
- Spark が高速である理由（分散処理、遅延評価など）の考察

🔜 次のステップ：Part 4

Part 4 では、Spark での集計処理をもう少し応用し、集計結果を pandas DataFrame に戻して可視化する方法などを学びます。

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習３ Part３）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
Part４（データ集計と可視化（応用））ワークブック

📝 **実習記録**

氏名	
実習日	年　　月　　日

🎯 Part４ の学習目標

実習 3 の総仕上げです。以下のスキルを習得します：

- Spark DataFrame で応用的な集計（例：ウィンドウ関数を使った移動平均）ができる
- Spark SQL を使って SQL ライクなデータ操作ができる（オプション）
- Spark での集計結果を pandas DataFrame に変換する（.toPandas()）方法と注意点を理解する
- 変換した pandas DataFrame を使って、matplotlib や seaborn で可視化できる
- Spark と pandas/matplotlib を連携させるワークフローを理解する

📖 Part 4 について

Part 3 では、Spark の「速さ」を体感しました。
Part 4 では、Spark の**集計能力**をもう少し掘り下げ、センサーデータに対してより実践的な分析（例：時間帯ごとの平均値、異常値の検出など）を行います。
そして、大規模データの「処理」は Spark で行い、最終的な「結果の可視化」は使い慣れた pandas と matplotlib/seaborn で行う、という**実務でもよく使われる連携パターン**を学びます。
推奨所要時間：約 1.5 時間

🔄 タスク１：準備（環境とデータの確認）

📌 Task 1-1: ライブラリのインポートと SparkSession の確認

必要なライブラリをインポートし、SparkSession を準備します。

```
import pandas as pd
import numpy as np
import time
from pyspark.sql import SparkSession
import pyspark.sql.functions as F # functions を F としてインポート
```

```
from pyspark.sql.window import Window # ウィンドウ関数を使うためにインポート

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set(style='darkgrid')

# SparkSession を取得 (なければ作成)
try:
    spark
    print("既存の SparkSession が見つかりました。")
except NameError:
    print("SparkSession を作成します...")
    spark = SparkSession.builder.appName("BigDataEx3_Part4").getOrCreate()
    print("SparkSession の準備ができました！")

spark
```

📌 Task 1-2: Parquet データの読み込み

```
Part 2, 3 で使った Parquet データを Spark DataFrame として読み込みます。
parquet_input_path = 'sensor_data.parquet'

# Parquet を読み込む
sensor_df = spark.read.parquet(parquet_input_path)

# 念のためスキーマと件数を確認
sensor_df.printSchema()
print(f"データ件数: {sensor_df.count():,} 件")
```

✓ チェックポイント 1-2

□ SparkSession が利用可能である。
□ sensor_df が Parquet から正しく読み込まれている。

📈 タスク 2：Spark による応用的な集計

Spark の集計機能をもう少し活用してみましょう。

📌 Task 2-1: 時間帯ごとの平均センサー値

timestamp 列が文字列（string）のままなので、まず日時型（Timestamp）に変換します。その後、時間帯（hour）ごとの各センサー値（temperature, humidity, pressure）の平均値を計算します。

```
# timestamp 列を Timestamp 型に変換
# F.to_timestamp() を使い、元の文字列フォーマットを指定
df_with_ts = sensor_df.withColumn(
    "ts",
    F.to_timestamp(F.col("timestamp"), 'yyyy-MM-dd HH:mm:ss')
)

# 新しい列 ts から hour を抽出
df_with_hour = df_with_ts.withColumn("hour", F.hour(F.col("ts")))

# hour ごとにグループ化し、各センサー値の平均を計算
hourly_avg_df = df_with_hour.groupBy("hour").agg(
    F.avg("temperature").alias("avg_temperature"),
    F.avg("humidity").alias("avg_humidity"),
    F.avg("pressure").alias("avg_pressure")
).orderBy("hour") # 時間順に並び替え

# 結果を表示（Action）
print("=== 時間帯ごとの平均センサー値 ===")
hourly_avg_df.show(24)
```

💡 .withColumn()

既存の列を基に新しい列を追加したり、既存の列を上書きしたりする Transformation です。
F.to_timestamp() や F.hour() は、pyspark.sql.functions に含まれる便利な関数です。

📌 Task 2-2: ウィンドウ関数を使った移動平均（オプション）

（少し高度な内容です）センサーごとに、時間順に並べた上で、前後 N 個のデータを使った「移動平均」を計算してみます。ノイズを除去したり、トレンドを見やすくしたりするのに使われます。

```
# ウィンドウの定義: sensor_id で分割し、timestamp で並び替える
windowSpec = Window.partitionBy("sensor_id").orderBy("ts")

# 過去 3 つのレコードと現在のレコードの平均（移動平均）を計算
moving_avg_df = df_with_ts.withColumn(
    "temp_moving_avg_3",
```

```
    F.avg("temperature").over(windowSpec.rowsBetween(-2, 0)) #  現在行とその前の 2 行
)

#  結果の一部を表示  (Action)
print("===  温度の移動平均  (過去 3 期間) ===")
moving_avg_df.select("sensor_id", "ts", "temperature",
"temp_moving_avg_3").orderBy("sensor_id", "ts").show(10)
```

💡 ウィンドウ関数

partitionBy でグループ化し、orderBy で並び替えた「窓（Window）」の中で、集計（avg, sum, rank など）を行う高度な機能です。rowsBetween(-2, 0) で「現在の行とその前の 2 行」という範囲を指定しています。

📌 Task 2-3: Spark SQL を使った集計（オプション）

Spark DataFrame は、SQL を使って操作することも可能です。SQL に慣れている人には便利です。

```
# DataFrame を一時的なテーブルとして登録
sensor_df.createOrReplaceTempView("sensor_data")

# SQL クエリを実行して集計 (Task 3-3 と同じ集計)
sql_agg_result = spark.sql("""
    SELECT
        sensor_id,
        COUNT(*) as record_count,
        AVG(temperature) as avg_temp,
        MAX(temperature) as max_temp,
        MIN(temperature) as min_temp
    FROM
        sensor_data
    GROUP BY
        sensor_id
""")

#  結果を表示  (Action)
print("=== Spark SQL での集計結果  ===")
sql_agg_result.show(10)
```

💡 Spark SQL

spark.sql("SQL クエリ") で、DataFrame に対して直接 SQL を実行できます。結果は Spark DataFrame として返されます。

📊 タスク 3：Pandas への変換と可視化

Spark で集計した結果（通常は元のデータより大幅に小さい）を、使い慣れた pandas DataFrame に変換し、matplotlib/seaborn で可視化します。

📌 Task 3-1: Spark DataFrame → Pandas DataFrame (.toPandas())

Spark DataFrame の .toPandas() メソッドを使います。
⚠️ **注意：** .toPandas() は、Spark クラスタ（Executor）に分散していたデータを、**すべて Driver（Colab 環境）のメモリに集める** Action です。変換対象のデータが巨大すぎると、ここでメモリ不足エラーが発生する可能性があります！

```
# Task 2-1 で計算した時間帯別平均データを Pandas に変換
print("Spark DataFrame を Pandas DataFrame に変換します...")
start_time = time.time()

# .toPandas() を実行（Action）
hourly_avg_pd_df = hourly_avg_df.toPandas()
# ↑ ここでメモリに乗るかどうかが重要！（hourly_avg_df は 24 行なので問題ない）

end_time = time.time()
print(f"変換完了（{end_time - start_time:.2f} 秒）")

# 変換後の Pandas DataFrame を確認
print("¥n=== 変換後の Pandas DataFrame ===")
print(hourly_avg_pd_df.info())
print(hourly_avg_pd_df.head())
```

⚠️ .toPandas() の注意点

必ず、groupby().agg() などで十分に集計され、**メモリに乗るサイズになった DataFrame に対してのみ**使用してください。元の巨大な sensor_df に対して .toPandas() を実行すると、ほぼ確実にメモリ不足になります。

📌 Task 3-2: Matplotlib/Seaborn での可視化

Pandas DataFrame に変換できれば、あとは実習 1, 2 で学んだのと同じ方法で可視化できます。

```
# 時間帯別の平均温度を折れ線グラフで可視化
plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_avg_pd_df, x='hour', y='avg_temperature', marker='o')
plt.title('Average Temperature by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Temperature')
```

```
plt.xticks(range(0, 24)) # X 軸のラベルを 0 時から 23 時に設定
plt.grid(True)
plt.show()

# 時間帯別の平均湿度を棒グラフで可視化
plt.figure(figsize=(12, 6))
sns.barplot(data=hourly_avg_pd_df, x='hour', y='avg_humidity', palette='coolwarm')
plt.title('Average Humidity by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Humidity (%)')
plt.show()
```

✨ Spark + Pandas/Matplotlib 連携

このように、「大規模データの**処理・集計**は Spark で行い」、「最終的な**結果の可視化**は pandas/matplotlib/seaborn で行う」というのが、実務でもよく使われる王道のパターンです。

📝 可視化結果からの考察

作成したグラフを見て、時間帯によるセンサー値の変動パターンについて気づいたことを記録してください。

🤔 Part 4 考察問題

問 1： なぜ Spark の集計結果を .toPandas() で変換してから可視化する、という手順を踏むことが多いのでしょうか？ Spark DataFrame のままでは可視化は難しいのでしょうか？（調べてみましょう）

問 2： .toPandas() を使う際に、最も注意すべきことは何ですか？ それはなぜですか？

✅ Part４ / 実習３のまとめ

🎉 実習３ 完了！ビッグデータ処理の第一歩！

Part 4 では、Spark での応用的な集計と、結果を pandas/matplotlib で可視化する連携を学びました。これで実習 3 全体が完了です！

📚 実習３全体で学んだこと

- 大規模データにおける pandas の限界と Spark（分散処理）の必要性
- Spark 環境の準備と SparkSession の開始
- Spark DataFrame の基本操作（読込、Action、Transformation）
- CSV vs Parquet のパフォーマンスと特性の違い
- Spark と pandas の処理速度比較と、その理由の考察
- Spark での集計結果を pandas に変換して可視化する連携フロー

🚀 これからの学習に向けて

実習 3 で学んだ Spark は、ビッグデータの世界への入口です。さらに深く学ぶなら...

- Spark のより高度な機能（RDD API、Streaming、MLlib）
- Hadoop エコシステム（HDFS, YARN, Hive など）との連携
- クラウド環境（AWS EMR, Google Dataproc, Azure Databricks など）での Spark 利用

などに挑戦していくと良いでしょう。今回の基礎が必ず役に立ちます！

🌟 全応用教材の完了

実習 1, 2, 3 を通して、データ分析の実践から ETL、BI、そして大規模データ処理まで、ビッグデータ活用の重要な側面を幅広く体験しました。ここで学んだ知識とスキルは、皆さんの大きな力となるはずです。お疲れ様でした！

✅ 実習振り返りシート （ビッグデータ技術応用 - 実習３ Part４）

振り返り日:＿＿＿＿年＿＿＿月＿＿＿日

氏名：＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

できたこと／わかったこと

今日学んで理解できたことを、自分の言葉で書いてください。

難しかったこと

うまくいかなかったこと、難しかったことを書いてください。

自己評価

1（できなかった）～ 5（よくできた）で◯をつけてください。

項目	評価	コメント
理解度	1　2　3　4　5	
完成度	1　2　3　4　5	
積極性	1　2　3　4　5	
楽しさ	1　2　3　4　5	

次回への目標

次の Part で頑張りたいことや意気込みを書いてください。

自由記入欄（先生へのメッセージ・質問／感想等）

ビッグデータ技術応用
実習 3 Spark による大規模データ処理
🚀 実習 3 発展課題

🌟 発展課題へようこそ！

実習 3 の基本内容をマスターした皆さん、Spark の世界への第一歩、おめでとうございます！
ここでは、Spark の性能をさらに引き出すためのテクニックや、DataFrame API 以外の側面、より実践的な環境構築に挑戦する課題を用意しました。
初級 Spark UI でのジョブ観察 & RDD API 入門 推奨：60 分

🎯 目的

Spark が内部でどのように処理を実行しているかを Spark UI で観察する。また、Spark の基本データ構造である RDD の簡単な操作を体験する。

📝 実装内容

1. Part 3 で実行した Spark の集計処理（Task 3-2）を再度実行し、実行中に Colab 上に表示される Spark UI のリンクを開く。
2. Spark UI の「Jobs」タブや「Stages」タブを見て、どのような処理（ステージ、タスク）が実行されたか、どれくらい時間がかかったかを観察・記録する。
3. Part 2 で読み込んだ Spark DataFrame (sensor_df) を RDD に変換する (.rdd)。
4. RDD API の map() を使って各行の温度 (temperature) だけを抽出し、filter() を使って 30 度以上のものだけをフィルタリングする。
5. take(10) Action を使って、結果の最初の 10 件を表示する。

💡 ヒント

- Colab での Spark UI へのアクセス方法は、SparkSession を作成した際の出力や、実行中のセルの下部に表示されるリンクを探してください。
- Spark UI では、特に処理が分割された「Stages」と、各 Executor で実行される「Tasks」の数や実行時間を見てみましょう。
- DataFrame を RDD に変換すると、各行は Row オブジェクトになります。map(lambda row: row.temperature) または map(lambda row: row['temperature']) のように列にアクセスします。
- RDD の操作は関数型プログラミングのスタイルに近いです。

✅ 評価ポイント

- Spark UI を開き、ジョブの実行状況（ステージ数、タスク数、時間など）を観察・記録できているか。

- DataFrame を RDD に変換し、map, filter, take を使って簡単な処理を実行できているか。

中級 Parquet パーティショニングとキャッシュの効果 推奨：90 分

🎯 目的

Parquet ファイルをパーティション分割して保存・読み込みすることによるパフォーマンス向上（Predicate Pushdown）を確認する。また、Spark のキャッシュ機能（.cache()）の効果を測定する。

📝 実装内容

1. Part 4 の Task 2-1 と同様に、sensor_df に year と month 列を追加する。
2. .write.partitionBy("year", "month").parquet(...) を使って、年と月でパーティション分割された Parquet ファイルを保存する。
3. 保存されたディレクトリ構造を確認する（!ls sensor_data_partitioned.parquet など）。
4. パーティション分割された Parquet 全体を読み込み、特定の月（例: month == 3）でフィルタリングして .count() する処理の時間を計測する。
5. 次に、同じ処理を、パーティション列をフィルタ条件に含めて（spark.read.parquet(...).where("month == 3").count()）実行し、時間を計測する。時間が短縮されるか確認する（Predicate Pushdown）。
6. フィルタリングした結果の DataFrame (filtered_df) を .cache() する。
7. filtered_df.count() を 2 回実行し、1 回目と 2 回目の実行時間を比較する（2 回目の方が速くなるはず）。

💡 ヒント

- partitionBy で保存すると、year=2023/month=1/ のようなサブディレクトリが自動で作成されます。
- Predicate Pushdown は、Spark がフィルタ条件を見て、不要なパーティション（ディレクトリ）のファイルを読み込まないようにする最適化です。where("month == 3") を spark.read.parquet() に繋げるのがポイントです。
- .cache() は Transformation ですが、初めて Action（例: count()）が実行された時に、その DataFrame がメモリ（やディスク）にキャッシュされます。2 回目以降の Action では、キャッシュされたデータが使われるため高速になります。
- キャッシュを解除するには .unpersist() を使います。

✅ 評価ポイント

- partitionBy を使って正しく Parquet を保存できているか。
- Predicate Pushdown の効果（フィルタリング速度の向上）を計測・確認できているか。
- .cache() の使い方を理解し、その効果（2 回目以降の高速化）を計測・確認できているか。

上級 Docker での PySpark 環境構築 推奨：120 分+α

🎯 目的

自分の PC 上に Docker を使って、Jupyter Notebook から PySpark を実行できる独立した環境を構築する。Colab 以外の実行環境を体験する。

📝 実装内容

1. Docker Desktop を自分の PC（Windows, Mac, Linux）にインストールする。
2. 公式の PySpark 入り Jupyter イメージ（jupyter/pyspark-notebook）を Docker Hub から Pull する（docker pull jupyter/pyspark-notebook）。
3. Docker コンテナを起動する。その際、以下の設定を行う：
    - Jupyter Notebook にアクセスするためのポートフォワーディング（例: -p 8888:8888）
    - ローカル PC のデータ（例: Part 1 で生成した CSV/Parquet）をコンテナ内から参照するためのボリュームマウント（例: -v /path/to/local/data:/home/jovyan/work/data）
    - （任意）コンテナに割り当てるメモリ量の指定（--memory オプション）
4. 起動時に表示される URL（トークン付き）を使って、ブラウザからコンテナ内の Jupyter Notebook にアクセスする。
5. 新しい Notebook を作成し、import pyspark と SparkSession の作成が成功することを確認する。
6. マウントしたボリューム内のデータ（/home/jovyan/work/data/...）を Spark で読み込み、簡単な処理（例: .count()）が実行できることを確認する。

💡 ヒント

- Docker の基本的なコマンド（docker pull, docker run, docker ps, docker stop）を調べてみましょう。
- docker run コマンドのオプションは非常に多いです。公式ドキュメントや解説記事を参考に、ポート（-p）とボリューム（-v）の設定を正しく行うことが重要です。
- コンテナ起動時のログに表示される http://127.0.0.1:8888/?token=... のような URL をコピーしてブラウザで開きます。
- コンテナ内のホームディレクトリは通常 /home/jovyan/work/ です。

✅ 評価ポイント

- Docker を使って jupyter/pyspark-notebook コンテナを起動できているか。
- ポートフォワーディングとボリュームマウントを正しく設定できているか。
- コンテナ内の Jupyter から PySpark を実行し、ローカルのデータを読み込めているか。
- 構築した手順を（コマンド含めて）記録・説明できているか。

🚀 ビッグデータ応用教材 - 実習 3 発展課題

令和7年度「地方やデジタル分野における専修学校理系転換等推進事業」
情報成長分野の教育プログラム整備と教員育成による学科転換・新設推進事業

ビッグデータ技術応用教材ワークブック

令和 8 年 2 月
一般社団法人全国専門学校情報教育協会
〒164-0003 東京都中野区東中野 1-57-8 辻沢ビル 3 F
電話：03-5332-5081 FAX.03-5332-5083