

🐼 完全初心者のための pandas データ分析マスター講座

対象者: Python 未経験の非エンジニア・ビジネスパーソン

期間: 5 日間で実務レベルまで

コンセプト: 「今日から使える」実践的データ分析スキル

📋 学習ロードマップ

📝 事前準備: 環境構築 (1 日)

💻 Day 1: Python 基礎と pandas 入門

🔧 Day 2: DataFrame の基本操作をマスター

✍ Day 3: データの読み込み・加工・クリーニング

⌚ Day 4: 高度なデータ操作と分析

📈 Day 5: グラフ作成とデータ可視化

📝 事前準備: 環境構築

必要なツールのインストール

1. Anaconda (推奨)

- **何これ?**: Python とデータ分析に必要なツールがすべて入ったパッケージ
- **なぜ必要?**: 面倒な設定を自動でやってくれる
- **ダウンロード**: <https://www.anaconda.com/products/distribution>

2. インストール手順

Windows の場合

1. Anaconda のサイトからインストーラーをダウンロード
2. ダウンロードした .exe ファイルを実行
3. 「Install for: Just Me」を選択
4. インストール先はデフォルトのまま
5. 「Add Anaconda to my PATH environment variable」にチェック
6. インストール完了まで待つ (10-15 分)

Mac の場合

1. Anaconda のサイトからインストーラーをダウンロード
2. ダウンロードした .pkg ファイルを実行

3. 指示に従ってインストール

3. 動作確認

Jupyter Notebook の起動

1. スタートメニュー（Windows）または Launchpad（Mac）で「Anaconda Navigator」を検索
2. Anaconda Navigator を起動
3. 「Jupyter Notebook」の「Launch」ボタンをクリック
4. ブラウザが開いて Jupyter Notebook が起動すれば OK

簡単な動作テスト 新しいノートブックを作成して、以下のコードを実行してみましょう：

```
# これは最初のPythonコード！  
print("Hello, Data Analysis!")  
  
# pandas（データ分析ライブラリ）が使えるかチェック  
import pandas as pd  
print("pandas バージョン:", pd.__version__)  
  
# 簡単なデータを作ってみる  
data = pd.DataFrame({  
    '名前': ['田中', '佐藤', '鈴木'],  
    '年齢': [25, 30, 35],  
    '部署': ['営業', '開発', '人事']  
})  
print(data)
```

このコードが問題なく実行できれば、環境構築完了です！

📊 Day 1: Python 基礎と pandas 入門

⌚ 今日のゴール

- Python の基本的な書き方を覚える
- pandas とは何かを理解する
- 最初のデータ分析を体験する

1. Python の基礎

```
# 🎉 Day 1: Pythonの基礎  
print("🎉 Welcome to Python & Data Analysis! 🎉")  
  
# 1. print()関数 - 画面に文字を表示  
print("これが最初のPythonコードです！")  
  
# 2. 変数 - データを入れる箱  
name = "田中太郎"
```

```
age = 28
salary = 500000

print(f"私の名前は{name}です")
print(f"年齢 : {age}歳")
print(f"月給 : {salary:,}円") # :, で3桁区切り

# 3. リスト - 複数のデータをまとめて保存
sales_data = [120, 150, 180, 200, 165]
products = ["ノートPC", "デスクトップPC", "タブレット", "スマートフォン"]

print("今月の日別売上(万円) : ", sales_data)
print("取り扱い商品 : ", products)

# リストから特定の要素を取得(0から数える!)
print(f"最初の商品 : {products[0]}")
print(f"最後の商品 : {products[-1]}") # -1で最後の要素

# 4. 辞書 - 名前付きでデータを保存
employee = {
    "名前": "佐藤花子",
    "部署": "営業部",
    "年齢": 32,
    "月給": 550000
}

print("従業員情報:")
print(f"名前: {employee['名前']}")
print(f"部署: {employee['部署']}")
print(f"年収: {employee['月給'] * 12:,}円")

# 5. 基本的な計算
monthly_sales = [480, 520, 450, 600, 580]
total_sales = sum(monthly_sales)
average_sales = total_sales / len(monthly_sales)

print(f"合計売上: {total_sales:,}円")
print(f"平均売上: {average_sales:.1f}円")
print(f"最高売上: {max(monthly_sales)}円")
print(f"最低売上: {min(monthly_sales)}円")

# 6. 条件分岐
target = 500 # 目標売上
actual = 520 # 実際の売上

if actual >= target:
    print(f"🎉 目標達成! (実績: {actual}円)")
else:
    print(f"😢 目標未達成 (実績: {actual}円)")

# 7. 繰り返し処理
products_sales = {
    "ノートPC": 250,
    "デスクトップPC": 180,
```

```
"タブレット": 320,  
"スマートフォン": 420  
}  
  
print("商品別売上評価:")  
for product, sales in products_sales.items():  
    if sales >= 300:  
        evaluation = "好調 🚀"  
    elif sales >= 200:  
        evaluation = "普通 📈"  
    else:  
        evaluation = "改善必要 🚧"  
  
    print(f"{product}: {sales}万円 - {evaluation}")
```

2. pandas の基礎

```
# ② pandasの基礎  
import pandas as pd  
  
print("☑ pandasが読み込まれました！")  
  
# 最初のDataFrame作成  
sales_team_data = {  
    '営業担当': ['田中', '佐藤', '鈴木', '高橋', '山田'],  
    '年齢': [28, 32, 26, 35, 30],  
    '今月売上': [520, 680, 450, 720, 600],  
    '担当地域': ['東京', '大阪', '名古屋', '福岡', '仙台']  
}  
  
df = pd.DataFrame(sales_team_data)  
print("営業チームのデータ:")  
print(df)  
  
# データの基本情報  
print(f"データの形: {df.shape}")  
print(f"行数: {df.shape[0]}人")  
print(f"列数: {df.shape[1]}項目")  
  
# 基本統計量  
print("\n基本統計量:")  
print(df.describe())  
  
# データを見る方法  
print("\n最初の3行:")  
print(df.head(3))  
  
print("\n最後の2行:")  
print(df.tail(2))  
  
# 簡単な分析
```

```
total_sales = df['今月売上'].sum()
average_sales = df['今月売上'].mean()

print(f"\nチーム全体の売上: {total_sales:,}万円")
print(f"平均売上: {average_sales:.1f}万円")

# 最高売上の人
top_performer = df.loc[df['今月売上'].idxmax()]
print(f"最高売上: {top_performer['営業担当']}さん ({top_performer['今月売上']}万円)")

# データの並び替え
df_sorted = df.sort_values('今月売上', ascending=False)
print("\n売上の高い順:")
print(df_sorted)

# 条件でデータを絞り込む
high_performers = df[df['今月売上'] >= 600]
print("\n売上600万円以上の営業担当:")
print(high_performers[['営業担当', '今月売上']])

# 新しい列を追加
def get_rank(sales):
    if sales >= 700:
        return 'S'
    elif sales >= 600:
        return 'A'
    elif sales >= 500:
        return 'B'
    else:
        return 'C'

df['売上ランク'] = df['今月売上'].apply(get_rank)
print("\n売上ランクを追加:")
print(df[['営業担当', '今月売上', '売上ランク']])
```

📅 Day 1まとめ

今日学んだこと: Python の基本的な書き方

- 変数、リスト、辞書の使い方
- 条件分岐と繰り返し処理
- pandas と DataFrame の基本概念
- 基本統計量の計算
- データの並び替えと絞り込み

⌚ Day 2: DataFrame の基本操作をマスター

⌚ 今日のゴール

- loc と iloc の使い分けをマスターする
- 複雑な条件でのデータ絞り込みができる

- インデックスを活用できる

1. loc と iloc の基礎

```
# ⚡ loc と iloc の基礎
import pandas as pd
import numpy as np

# サンプルデータ準備
sales_data = {
    '営業ID': ['S001', 'S002', 'S003', 'S004', 'S005', 'S006'],
    '営業担当': ['田中太郎', '佐藤花子', '鈴木一郎', '高橋美咲', '山田健太', '中村由美'],
    '年齢': [28, 32, 26, 35, 30, 29],
    '部署': ['東京営業', '大阪営業', '東京営業', '名古屋営業', '東京営業', '大阪営業'],
    '2023年売上': [5200, 6800, 4500, 7200, 6000, 5800],
    '2024年売上': [5800, 7200, 5100, 7800, 6500, 6200],
    '顧客数': [25, 35, 20, 40, 30, 28]
}

df = pd.DataFrame(sales_data)
print("営業チームデータ:")
print(df)

# インデックスを営業IDに設定
df_indexed = df.set_index('営業ID')
print("\n営業IDをインデックスに設定:")
print(df_indexed)

# iloc - 位置による選択
print("\n❷ iloc - 位置による選択")
print("1行目1列目:", df.iloc[0, 0])
print("最初の行:")
print(df.iloc[0])

print("\n最初の3行、最初の4列:")
print(df.iloc[0:3, 0:4])

# loc - ラベルによる選択
print("\n⌚ loc - ラベルによる選択")
print("S001の情報:")
print(df_indexed.loc['S001'])

print("\nS001とS003の情報:")
print(df_indexed.loc[['S001', 'S003']])

# 条件による選択
print("\n⌚ 条件による選択")
print("2024年売上が6000以上:")
high_performers = df_indexed.loc[df_indexed['2024年売上'] >= 6000]
print(high_performers[['営業担当', '2024年売上']])
```

```

print("\n年齢30歳以上で売上6000以上:")
experienced_high = df_indexed.loc[
    (df_indexed['年齢'] >= 30) & (df_indexed['2024年売上'] >= 6000)
]
print(experienced_high[['営業担当', '年齢', '2024年売上']])

print("\n東京営業または売上7000以上:")
tokyo_or_high = df_indexed.loc[
    (df_indexed['部署'] == '東京営業') | (df_indexed['2024年売上'] >= 7000)
]
print(tokyo_or_high[['営業担当', '部署', '2024年売上']])

```

2. 複雑な条件とインデックス操作

```

# 🔍 複雑な条件での絞り込み
business_data = {
    '従業員ID': ['E001', 'E002', 'E003', 'E004', 'E005', 'E006', 'E007', 'E008'],
    '氏名': ['田中太郎', '佐藤花子', '鈴木一郎', '高橋美咲', '山田健太', '中村由美',
    '小林大輔', '加藤真紀'],
    '部署': ['営業', '営業', '開発', '人事', '営業', '開発', '経理', '人事'],
    '年齢': [28, 32, 26, 45, 30, 29, 48, 27],
    '基本給': [350000, 320000, 400000, 550000, 340000, 380000, 520000, 310000],
    '勤続年数': [4, 6, 2, 15, 5, 3, 18, 1],
    '評価': ['A', 'B', 'A', 'S', 'C', 'A', 'S', 'B']
}

df = pd.DataFrame(business_data)

# 複雑な昇進候補者の特定
print("昇進候補者（年齢30歳以上、評価A以上、勤続年数5年以上）:")
promotion_candidates = df[
    (df['年齢'] >= 30) &
    (df['評価'].isin(['A', 'S'])) &
    (df['勤続年数'] >= 5)
]
print(promotion_candidates[['氏名', '年齢', '評価', '勤続年数']])

# 文字列操作での条件
print("\n文字列条件:")
print("名前に'田'が含まれる従業員:")
tanaka_group = df[df['氏名'].str.contains('田')]
print(tanaka_group[['氏名']])

# 範囲条件
print("\n年齢が30-40歳の従業員:")
age_range = df[df['年齢'].between(30, 40)]
print(age_range[['氏名', '年齢']])

# インデックス操作
df_indexed = df.set_index('従業員ID')

```

```
print("\n従業員IDをインデックスに設定後、E003の情報:")
print(df_indexed.loc['E003'])
```

📅 Day 2まとめ

今日学んだこと: loc と iloc の違いと使い分け

- 複雑な条件でのデータ絞り込み
- 文字列操作での条件指定
- インデックスの設定と活用
- 範囲条件 (between) の使用

✍ Day 3: データの読み込み・加工・クリーニング

⌚ 今日のゴール

- CSV ファイルの読み込み・保存をマスターする
- 汚れたデータをきれいにする方法を学ぶ
- データ結合の基礎を理解する

1. CSV ファイルの操作

```
# CSVファイルの操作
import pandas as pd
import numpy as np

# サンプルデータ作成
employee_data = {
    '従業員ID': ['EMP001', 'EMP002', 'EMP003', 'EMP004', 'EMP005'],
    '氏名': ['田中太郎', '佐藤花子', '鈴木一郎', '高橋美咲', '山田健太'],
    '部署': ['営業', '営業', '開発', '人事', '営業'],
    '年齢': [28, 32, 26, 35, 30],
    '基本給': [320000, 380000, 350000, 450000, 340000]
}

df_employees = pd.DataFrame(employee_data)
print("従業員データ:")
print(df_employees)

# CSVファイルとして保存
df_employees.to_csv('employees.csv', index=False, encoding='utf-8-sig')
print("☑ CSVファイル保存完了")

# CSVファイルを読み込む
df_loaded = pd.read_csv('employees.csv', encoding='utf-8-sig')
print("\nCSVファイル読み込み完了:")
print(df_loaded)

# データの詳細確認
print(f"\nデータ型:\n{df_loaded.dtypes}")
```

```
print(f"\n欠損値の確認:\n{df_loaded.isnull().sum()}")
print(f"\n基本統計量:")
print(df_loaded.describe())
```

2. データクリーニング

```
# データクリーニング
# 意図的に汚れたデータを作成
dirty_data = {
    '従業員ID': ['EMP001', 'EMP002', 'emp003', 'EMP004', 'EMP002', 'EMP005', ''],
    '氏名': ['田中太郎', '佐藤 花子', '鈴木一郎', '', '佐藤花子', '山田健太', '高橋美咲'],
    '年齢': [28, 32, -5, 150, 32, 30, '三十五'],
    '部署': ['営業', '営業部', '開発', '人事', '営業部', '開発', 'HR'],
    '給与': [320000, 380000, '', 450000, 380000, 340000, 420000]
}

df_dirty = pd.DataFrame(dirty_data)
print("汚れたデータ:")
print(df_dirty)

# 欠損値の処理
df_clean = df_dirty.replace('', np.nan)
print("\n欠損値の数:")
print(df_clean.isnull().sum())

# 重複データの処理
duplicates = df_clean.duplicated()
print(f"\n重複行の数: {duplicates.sum()}")

if duplicates.sum() > 0:
    df_no_duplicates = df_clean.drop_duplicates()
    print(f"重複削除後: {len(df_clean)}行 → {len(df_no_duplicates)}行")

# 異常値の修正
def fix_age(age):
    try:
        age_num = int(age)
        if age_num < 0 or age_num > 100:
            return np.nan
        return age_num
    except:
        return np.nan

df_clean['年齢'] = df_clean['年齢'].apply(fix_age)
print("\n年齢修正後:")
print(df_clean['年齢'].describe())

# 文字列データの統一
dept_mapping = {
    '営業': '営業',
```

```
'営業部': '営業',
'開発': '開発',
'人事': '人事',
'HR': '人事'

}

df_clean['部署'] = df_clean['部署'].map(dept_mapping)
print("\n部署名統一後:")
print(df_clean['部署'].value_counts())
```

3. データ結合の基礎

```
# 🔍 データ結合の基礎
# 顧客マスター
customers = pd.DataFrame({
    '顧客ID': ['C001', 'C002', 'C003', 'C004', 'C005'],
    '顧客名': ['田中太郎', '佐藤花子', '鈴木一郎', '高橋美咲', '山田健太'],
    '年齢': [28, 32, 26, 35, 30],
    '住所': ['東京', '大阪', '名古屋', '福岡', '札幌']
})

# 注文データ
orders = pd.DataFrame({
    '注文ID': ['0001', '0002', '0003', '0004', '0005'],
    '顧客ID': ['C001', 'C002', 'C001', 'C003', 'C002'],
    '商品': ['ノートPC', 'タブレット', 'スマートフォン', 'ノートPC', 'タブレット'],
    '金額': [120000, 80000, 150000, 120000, 80000]
})

print("顧客マスター:")
print(customers)
print("\n注文データ:")
print(orders)

# 基本的な結合
orders_with_customer = pd.merge(orders, customers, on='顧客ID')
print("\n注文データに顧客情報を追加:")
print(orders_with_customer)

# 顧客別売上集計
customer_sales = orders_with_customer.groupby('顧客名')['金額'].sum().sort_values(ascending=False)
print("\n顧客別売上:")
print(customer_sales)
```

⌚ Day 3まとめ

- 今日学んだこと:**
- CSV ファイルの読み込み・保存
 - データクリーニング（欠損値、重複、異常値の処理）
 - 文字列データの統一

データ結合の基礎（merge）

結合後のデータ分析

⌚ Day 4: 高度なデータ操作と分析

⌚ 今日のゴール

- 時系列データを扱えるようになる
- 高度な集計・分析ができる
- 実践的なデータ分析を体験する

1. 時系列データの操作

```
# 📈 時系列データの操作
import pandas as pd
import numpy as np

# 日別売上データの作成
np.random.seed(42)
daily_sales = pd.DataFrame({
    '日付': pd.date_range('2024-01-01', periods=30, freq='D'),
    '売上': np.random.randint(50000, 150000, 30),
    '来客数': np.random.randint(100, 300, 30)
})

print("日別売上データ:")
print(daily_sales.head(10))

# 日付をインデックスに設定
daily_sales_indexed = daily_sales.set_index('日付')
print("\n日付をインデックスに設定:")
print(daily_sales_indexed.head())

# 日付情報の抽出
daily_sales['年'] = daily_sales['日付'].dt.year
daily_sales['月'] = daily_sales['日付'].dt.month
daily_sales['日'] = daily_sales['日付'].dt.day
daily_sales['曜日'] = daily_sales['日付'].dt.day_name()
daily_sales['平日フラグ'] = daily_sales['日付'].dt.weekday < 5

print("\n日付情報を抽出:")
print(daily_sales[['日付', '年', '月', '日', '曜日', '平日フラグ']].head())

# 曜日別分析
weekday_analysis = daily_sales.groupby('曜日')[['売上', '来客数']].mean()
print("\n曜日別平均:")
print(weekday_analysis)

# 移動平均(トレンド分析)
daily_sales_indexed['売上_移動平均'] = daily_sales_indexed['売上'].rolling(window=7).mean()
```

```
print("\n7日移動平均:")
print(daily_sales_indexed[['売上', '売上_移動平均']].head(10))
```

2. 高度な集計と分析

```
# 📈 高度な集計と分析
# 複雑なビジネスデータ
business_data = {
    '従業員ID': ['E001', 'E002', 'E003', 'E004', 'E005', 'E006', 'E007', 'E008'],
    '氏名': ['田中太郎', '佐藤花子', '鈴木一郎', '高橋美咲', '山田健太', '中村由美',
    '小林大輔', '加藤真紀'],
    '部署': ['営業', '営業', '開発', '人事', '営業', '開発', '経理', '人事'],
    '役職': ['主任', '一般', 'リーダー', '部長', '一般', '主任', '部長', '一般'],
    '年齢': [28, 32, 26, 45, 30, 29, 48, 27],
    '基本給': [350000, 320000, 400000, 550000, 340000, 380000, 520000, 310000],
    '勤続年数': [4, 6, 2, 15, 5, 3, 18, 1],
    '評価': ['A', 'B', 'A', 'S', 'C', 'A', 'S', 'B']
}

df = pd.DataFrame(business_data)

# 部署別分析
print("部署別分析:")
dept_analysis = df.groupby('部署').agg({
    '氏名': 'count', # 人数
    '年齢': 'mean', # 平均年齢
    '基本給': ['mean', 'max', 'min'], # 給与統計
    '勤続年数': 'mean' # 平均勤続年数
}).round(1)

dept_analysis.columns = ['人数', '平均年齢', '平均給与', '最高給与', '最低給与', '平均勤続年数']
print(dept_analysis)

# 年齢層別分析
def age_group(age):
    if age < 30:
        return '20代'
    elif age < 40:
        return '30代'
    else:
        return '40代以上'

df['年齢層'] = df['年齢'].apply(age_group)
age_analysis = df.groupby('年齢層').agg({
    '氏名': 'count',
    '基本給': 'mean'
}).round(0)

age_analysis.columns = ['人数', '平均給与']
print("\n年齢層別分析:")
print(age_analysis)
```

```
# クロス集計 (ピボットテーブル)
pivot_table = pd.crosstab(df['部署'], df['評価'])
print("\n部署別・評価別人数:")
print(pivot_table)

# 相関分析
correlation = df['勤続年数'].corr(df['基本給'])
print(f"\n勤続年数と給与の相関係数: {correlation:.3f}")
```

3. 実践的なデータ分析

```
# 実践的なデータ分析
# 売上分析のケーススタディ
sales_data = {
    '日付': pd.date_range('2024-01-01', periods=100, freq='D'),
    '店舗': np.random.choice(['新宿店', '渋谷店', '池袋店'], 100),
    '商品カテゴリ': np.random.choice(['食品', '衣料', '雑貨', '電子機器'], 100),
    '売上': np.random.randint(10000, 100000, 100),
    '来客数': np.random.randint(50, 200, 100)
}

sales_df = pd.DataFrame(sales_data)

# 店舗別パフォーマンス
store_performance = sales_df.groupby('店舗').agg({
    '売上': ['sum', 'mean', 'count'],
    '来客数': 'mean'
}).round(0)

store_performance.columns = ['売上合計', '平均売上', '取引数', '平均来客数']
print("店舗別パフォーマンス:")
print(store_performance)

# 商品カテゴリ別分析
category_analysis = sales_df.groupby('商品カテゴリ')['売上'].agg(['sum', 'count',
    'mean']).round(0)
category_analysis.columns = ['売上合計', '取引数', '平均売上']
category_analysis = category_analysis.sort_values('売上合計', ascending=False)
print("\n商品カテゴリ別分析:")
print(category_analysis)

# 日別トレンド分析
daily_trend = sales_df.groupby('日付')['売上'].sum()
print("\n日別売上トレンド (最初の10日):")
print(daily_trend.head(10))

# 移動平均でトレンド分析
trend_analysis = pd.DataFrame({
    '日付': daily_trend.index,
    '売上': daily_trend.values
```

```

    })
trend_analysis['移動平均_7日'] = trend_analysis['売上'].rolling(window=7).mean()
trend_analysis['移動平均_14日'] = trend_analysis['売上'].rolling(window=14).mean()

print("\n売上トレンド分析:")
print(trend_analysis.head(15))

```

▣ Day 4まとめ

今日学んだこと: 時系列データの操作と分析

- 日付情報の抽出
- 高度な集計 (groupby、agg)
- クロス集計 (crosstab)
- 相関分析
- 移動平均によるトレンド分析

▣ Day 5: グラフ作成とデータ可視化

⌚ 今日のゴール

- pandas でグラフを作成できる
- 様々な種類のグラフを使い分けられる
- 実践的なダッシュボードを作成する

1. 基本的なグラフ作成

```

# ▣ 基本的なグラフ作成
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 日本語フォントの設定
plt.rcParams['font.family'] = 'DejaVu Sans' # 環境に応じて調整

# サンプルデータ
np.random.seed(42)
sales_data = pd.DataFrame({
    '月': ['1月', '2月', '3月', '4月', '5月', '6月'],
    '売上': [1200, 1350, 1180, 1420, 1580, 1650],
    '来客数': [450, 480, 420, 510, 540, 580]
})

print("売上データ:")
print(sales_data)

# 1. 線グラフ
plt.figure(figsize=(10, 6))
plt.plot(sales_data['月'], sales_data['売上'], marker='o', linewidth=2)
plt.title('月別売上推移')

```

```

plt.xlabel('月')
plt.ylabel('売上(万円)')
plt.grid(True, alpha=0.3)
plt.show()

# 2. 棒グラフ
plt.figure(figsize=(10, 6))
plt.bar(sales_data['月'], sales_data['売上'], color='skyblue', alpha=0.7)
plt.title('月別売上')
plt.xlabel('月')
plt.ylabel('売上(万円)')
plt.show()

# 3. 複数系列のグラフ
fig, ax1 = plt.subplots(figsize=(10, 6))

# 売上(左軸)
ax1.plot(sales_data['月'], sales_data['売上'], 'b-o', label='売上')
ax1.set_xlabel('月')
ax1.set_ylabel('売上(万円)', color='b')
ax1.tick_params(axis='y', labelcolor='b')

# 来客数(右軸)
ax2 = ax1.twinx()
ax2.plot(sales_data['月'], sales_data['来客数'], 'r-s', label='来客数')
ax2.set_ylabel('来客数(人)', color='r')
ax2.tick_params(axis='y', labelcolor='r')

plt.title('売上と来客数の推移')
plt.show()

```

2. 様々なグラフの種類

```

# 📈 様々なグラフの種類
# より詳細な分析データ
detailed_data = pd.DataFrame({
    '店舗': ['新宿店', '渋谷店', '池袋店', '銀座店', '品川店'],
    '売上': [2500, 2200, 1800, 3200, 2100],
    '来客数': [800, 750, 600, 900, 650],
    '従業員数': [12, 10, 8, 15, 9]
})

# 1. 散布図
plt.figure(figsize=(10, 6))
plt.scatter(detailed_data['来客数'], detailed_data['売上'],
            s=detailed_data['従業員数']*20, alpha=0.7, c='red')
plt.xlabel('来客数(人)')
plt.ylabel('売上(万円)')
plt.title('来客数と売上の関係(円の大きさ=従業員数)')

# 各点にラベル追加

```

```

for i, store in enumerate(detailed_data['店舗']):
    plt.annotate(store, (detailed_data['来客数'][i], detailed_data['売上'][i]))

plt.show()

# 2. 円グラフ
plt.figure(figsize=(8, 8))
plt.pie(detailed_data['売上'], labels=detailed_data['店舗'], autopct='%1.1f%%')
plt.title('店舗別売上構成比')
plt.show()

# 3. ヒストグラム
np.random.seed(42)
customer_ages = np.random.normal(35, 10, 1000)
customer_ages = customer_ages[customer_ages > 0] # 負の年齢を除去

plt.figure(figsize=(10, 6))
plt.hist(customer_ages, bins=30, alpha=0.7, color='green')
plt.xlabel('年齢')
plt.ylabel('人数')
plt.title('顧客年齢分布')
plt.axvline(customer_ages.mean(), color='red', linestyle='--',
            label=f'平均年齢: {customer_ages.mean():.1f}歳')
plt.legend()
plt.show()

# 4. 箱ひげ図
category_sales = pd.DataFrame({
    'カテゴリ': ['電子機器']*100 + ['衣料']*100 + ['食品']*100 + ['雑貨']*100,
    '売上': list(np.random.normal(150, 30, 100)) +
              list(np.random.normal(80, 20, 100)) +
              list(np.random.normal(50, 15, 100)) +
              list(np.random.normal(30, 10, 100))
})
plt.figure(figsize=(10, 6))
category_sales.boxplot(column='売上', by='カテゴリ', ax=plt.gca())
plt.title('カテゴリ別売上分布')
plt.suptitle('') # デフォルトのタイトルを非表示
plt.show()

```

3. pandas の組み込みグラフ機能

```

# 🐍 pandasの組み込みグラフ機能
# 時系列データでのグラフ作成
dates = pd.date_range('2024-01-01', periods=30, freq='D')
time_series_data = pd.DataFrame({
    '日付': dates,
    '売上': np.random.randint(80, 150, 30) + 50*np.sin(np.arange(30)/5),
    '来客数': np.random.randint(200, 400, 30)
})

```

```

time_series_data = time_series_data.set_index('日付')

# pandas の plot() メソッド
time_series_data['売上'].plot(kind='line', figsize=(12, 6), title='日別売上推移')
plt.ylabel('売上(万円)')
plt.show()

# 複数列の同時プロット
time_series_data.plot(kind='line', figsize=(12, 6), title='売上と来客数の推移')
plt.show()

# 棒グラフ
monthly_summary = pd.DataFrame({
    '月': ['1月', '2月', '3月', '4月', '5月'],
    '売上': [1200, 1350, 1180, 1420, 1580],
    '予算': [1100, 1300, 1200, 1400, 1500]
})

monthly_summary.set_index('月').plot(kind='bar', figsize=(10, 6), title='月別売上 vs 予算')
plt.ylabel('金額(万円)')
plt.xticks(rotation=0)
plt.show()

```

4. 実践的なダッシュボード作成

```

# 📈 実践的なダッシュボード作成
# 包括的な分析とグラフ
np.random.seed(42)

# 売上データ生成
dashboard_data = pd.DataFrame({
    '日付': pd.date_range('2024-01-01', periods=90, freq='D'),
    '店舗': np.random.choice(['新宿店', '渋谷店', '池袋店'], 90),
    '商品カテゴリ': np.random.choice(['電子機器', '衣料', '食品', '雑貨'], 90),
    '売上': np.random.randint(20000, 200000, 90),
    '来客数': np.random.randint(50, 300, 90)
})

# 複数のグラフを一つの画面に表示
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# 1. 日別売上推移
daily_sales = dashboard_data.groupby('日付')['売上'].sum()
daily_sales.plot(ax=axes[0, 0], title='日別売上推移', color='blue')
axes[0, 0].set_ylabel('売上(円)')

# 2. 店舗別売上
store_sales = dashboard_data.groupby('店舗')['売上'].sum()
store_sales.plot(kind='bar', ax=axes[0, 1], title='店舗別売上')

```

```

axes[0,1].set_ylabel('売上(円)')
axes[0,1].tick_params(axis='x', rotation=45)

# 3. 商品カテゴリ別売上(円グラフ)
category_sales = dashboard_data.groupby('商品カテゴリ')['売上'].sum()
category_sales.plot(kind='pie', ax=axes[1,0], title='商品カテゴリ別売上構成比',
autopct='%1.1f%%')
axes[1,0].set_ylabel('')

# 4. 売上と来客数の相関
axes[1,1].scatter(dashboard_data['来客数'], dashboard_data['売上'], alpha=0.6)
axes[1,1].set_xlabel('来客数(人)')
axes[1,1].set_ylabel('売上(円)')
axes[1,1].set_title('来客数と売上の関係')

plt.tight_layout()
plt.show()

# 基本統計の表示
print("📊 基本統計サマリー:")
print(f"総売上: {dashboard_data['売上'].sum():,}円")
print(f"平均日別売上: {dashboard_data.groupby('日付')['売上'].sum().mean():,.0f}円")
print(f"最高売上日: {dashboard_data.groupby('日付')['売上'].sum().max():,}円")
print(f"総来客数: {dashboard_data['来客数'].sum():,}人")

print("\n店舗別パフォーマンス:")
store_performance = dashboard_data.groupby('店舗').agg({
    '売上': ['sum', 'mean', 'count'],
    '来客数': 'mean'
}).round(0)
store_performance.columns = ['売上合計', '平均売上', '取引数', '平均来客数']
print(store_performance)

```

5. 総合演習

```

# 🎯 総合演習: 完全なデータ分析プロジェクト
# これまで学んだすべてのスキルを統合

print("📊 総合演習: ECサイト分析プロジェクト")
print("*"*50)

# 1. データ準備
np.random.seed(42)
ec_data = pd.DataFrame({
    '日付': pd.date_range('2024-01-01', periods=180, freq='D'),
    '商品カテゴリ': np.random.choice(['Electronics', 'Fashion', 'Books', 'Sports'], 180),
    '価格帯': np.random.choice(['低価格', '中価格', '高価格'], 180),
    '売上': np.random.randint(50000, 500000, 180),
    '注文数': np.random.randint(10, 100, 180),
    '新規顧客数': np.random.randint(5, 50, 180)
})

```

```
})

# 2. データ加工
ec_data['月'] = ec_data['日付'].dt.month
ec_data['曜日'] = ec_data['日付'].dt.day_name()
ec_data['平均注文金額'] = ec_data['売上'] / ec_data['注文数']

# 3. 基本分析
print("基本統計:")
print(f"総売上: {ec_data['売上'].sum():,}円")
print(f"総注文数: {ec_data['注文数'].sum():,}件")
print(f"平均注文金額: {ec_data['平均注文金額'].mean():,.0f}円")

# 4. カテゴリ別分析
category_analysis = ec_data.groupby('商品カテゴリ').agg({
    '売上': 'sum',
    '注文数': 'sum',
    '平均注文金額': 'mean'
}).round(0)
print("\nカテゴリ別分析:")
print(category_analysis)

# 5. 月別トレンド
monthly_trend = ec_data.groupby('月').agg({
    '売上': 'sum',
    '注文数': 'sum',
    '新規顧客数': 'sum'
})
print("\n月別トレンド:")
print(monthly_trend)

# 6. 可視化
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

# 月別売上推移
monthly_trend['売上'].plot(kind='line', ax=axes[0,0], marker='o', title='月別売上推移')
axes[0,0].set_ylabel('売上(円)')

# カテゴリ別売上
category_analysis['売上'].plot(kind='bar', ax=axes[0,1], title='カテゴリ別売上')
axes[0,1].set_ylabel('売上(円)')
axes[0,1].tick_params(axis='x', rotation=45)

# 価格帯別構成比
price_composition = ec_data.groupby('価格帯')['売上'].sum()
price_composition.plot(kind='pie', ax=axes[0,2], title='価格帯別売上構成比', autopct='%1.1f%%')
axes[0,2].set_ylabel('')

# 日別売上推移
daily_sales = ec_data.groupby('日付')['売上'].sum()
daily_sales.plot(ax=axes[1,0], title='日別売上推移', alpha=0.7)
axes[1,0].set_ylabel('売上(円)')
```

```

# 注文数と売上の相関
axes[1,1].scatter(ec_data['注文数'], ec_data['売上'], alpha=0.6)
axes[1,1].set_xlabel('注文数')
axes[1,1].set_ylabel('売上(円)')
axes[1,1].set_title('注文数と売上の関係')

# 曜日別パフォーマンス
weekday_performance = ec_data.groupby('曜日')['売上'].mean()
weekday_performance.plot(kind='bar', ax=axes[1,2], title='曜日別平均売上')
axes[1,2].set_ylabel('平均売上(円)')
axes[1,2].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

# 7. インサイトの抽出
print("\n⌚ 分析結果のインサイト:")
print(f"• 最も売上の高いカテゴリ: {category_analysis['売上'].idxmax()}")
print(f"• 最も売上の高い月: {monthly_trend['売上'].idxmax()}月")
print(f"• 最も売上の高い曜日: {weekday_performance.idxmax()}")
print(f"• 注文数と売上の相関係数: {ec_data['注文数'].corr(ec_data['売上']):.3f}")

# 8. 改善提案
print("\n💡 改善提案:")
low_performing_category = category_analysis['売上'].idxmin()
print(f"• {low_performing_category}カテゴリの売上向上施策が必要")
print(f"• 平日の売上向上策を検討")
print(f"• 高価格帯商品の販促強化")

```

📅 Day 5 まとめ

今日学んだこと: matplotlib を使った基本的なグラフ作成

- 線グラフ、棒グラフ、散布図、円グラフの作成
- pandas の組み込みグラフ機能
- 複数のグラフを組み合わせたダッシュボード
- 実践的なデータ分析プロジェクト

🎓 5 日間の学習完了！

🎉 おめでとうございます！

あなたは 5 日間で以下のスキルを習得しました：

- 【】 習得したスキル:** **Python 基礎** - 変数、リスト、辞書、条件分岐、繰り返し
- pandas 基礎** - DataFrame 作成、基本操作、統計量計算
- データ選択** - loc/iloc、複雑な条件での絞り込み
- データ処理** - CSV ファイル操作、データクリーニング
- データ結合** - merge を使った複数データの統合
- 時系列分析** - 日付データの操作、トレンド分析

- 高度な分析** - groupby、集計、相関分析
- データ可視化** - matplotlib、グラフ作成、ダッシュボード

⌚ 次のステップ

すぐに始められること:

1. 自分の業務データでこれらのテクニックを試す
2. CSV ファイルを読み込んで基本的な分析を行う
3. 簡単なグラフを作成してみる

さらに学習を進めたい場合:

1. **統計分析** - scipy を使った高度な統計処理
2. **機械学習** - scikit-learn を使った予測モデル
3. **Web ダッシュボード** - Streamlit や Dash を使った対話的なアプリ
4. **データベース連携** - SQL と pandas の連携

⌚ 継続のコツ

1. **毎日少しずつ** - 15 分でも良いので触り続ける
2. **実際のデータで練習** - 業務や興味のあるデータを使う
3. **エラーを恐れない** - エラーは学習の機会
4. **コミュニティ参加** - 質問したり、知識を共有する

📖 便利なリファレンス

よく使う関数チートシート:

```

# データ読み込み・保存
pd.read_csv('file.csv')                      # CSV読み込み
df.to_csv('file.csv', index=False)             # CSV保存

# データ確認
df.head()                                     # 最初の5行
df.shape                                       # 行数×列数
df.info()                                      # データ型と欠損値
df.describe()                                   # 基本統計量

# データ選択
df.loc[条件, 列名]                            # ラベルで選択
df.iloc[行番号, 列番号]                       # 位置で選択
df[df['列名'] > 値]                           # 条件絞り込み

# データ操作
df.groupby('列名').sum()                      # グループ別集計
df.sort_values('列名')                         # 並び替え
pd.merge(df1, df2, on='キー')                  # データ結合
df.fillna(値)                                 # 欠損値補完

# 可視化

```

```
df.plot()                      # 基本グラフ  
df.plot(kind='bar')            # 棒グラフ  
df.plot(kind='hist')           # ヒストグラム
```

あなたは今や実務で使えるデータ分析スキルを持っています！

自信を持って、データ分析の世界を楽しんでください！ ☺ ♦