社内研修・データ分析

Version: 1.0

コンテンツ

Part 6

時系列・位置情報データと高度なデータ加工

Version: 1.0

本研修の目的

データ分析の現場で即戦力となる人材の育成

パート6の目的:

これまでのパートで学んだ基本的なデータ加工（クレンジング・数値化）に加え、実務で頻繁に遭遇する 「時系列データ（Time Series Data）」 や 「位置情報データ（Geospatial Data）」 といった、より専門的な知識が必要なドメインのデータ加工手法を習得します。

また、単純な平均値埋めでは対応できない複雑なケースに向けた、高度な欠損値補完（多重代入法の概念など） についても学び、ETLプロセスの「変換（Transform）」の質をさらに高め、分析精度を底上げする技術を身につけます。

目次

環境セットアップ

パート6では、Pandasに加え、以下のライブラリを使用します。

- `geopy`: 地球上の2点間の距離を正確に計算するための地理空間ライブラリです。
- `scikit-learn`: 機械学習ライブラリですが、ここでは高度な欠損値補完（`IterativeImputer`）のために使用します。

前パートと同じように、モジュールインポートやデータベースの接続をノートブックで行います。

目次

時系列データの加工

なぜ変換が必要か？

	A	B
1	日付	店舗売上
2	2025-03-01	54332
3	2025/03/02	64889
4	20250303	23512
5	2025/3/4 0:00	43654

pd.to_datetime

	日付	店舗売上
0	2025-03-01	54332
1	2025-03-02	64889
2	2025-03-03	23512
3	2025-03-04	43654

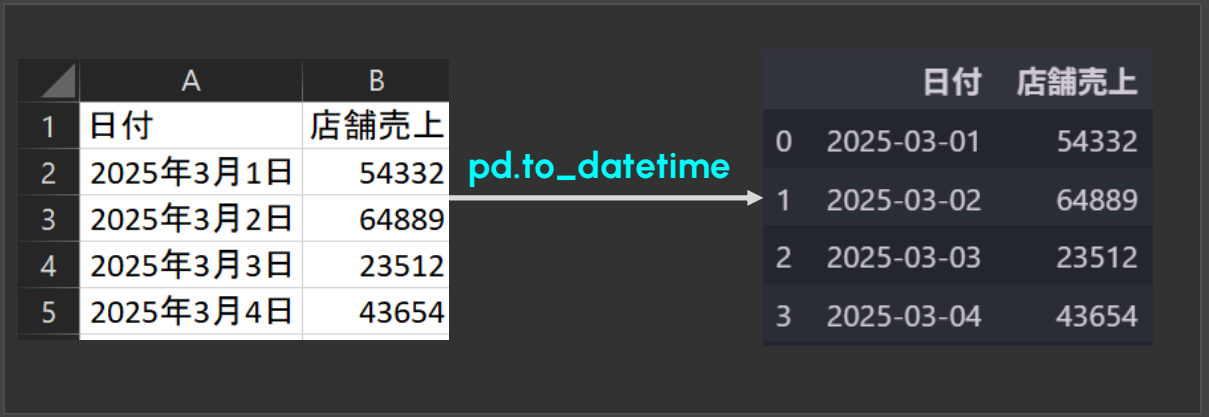
日時型 (Datetime) への変換

基本構文

日本語フォーマット例

"YYYY年M月D日"の日本語フォーマットを読むためには、次のように記述します:

```
df['日付'] = pd.to_datetime(df['日付'], format='%Y年%m月%d日')
```



日時型 (Datetime) への変換

使用例

```
--- 変換後（日時型：datetime64） ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       4 non-null      int64
 1   contact_date_str  4 non-null      object
 2   contact_date      4 non-null      datetime64[ns]
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 228.0+ bytes
None
None
display(df_time)
```

	customer_id	contact_date
0	101	2023-05-01 00:00:00
1	102	2023-05-15 00:00:00
2	103	2023-06-01 00:00:00
3	104	2023-06-10 14:30:00

日付要素の抽出と差分計算

Datetime型の `.dt` アクセサを使って様々な情報を簡単に抽出できます。

数学的背景：時間の差分

時間は連続的な数値として扱うことができます。

$$\Delta t = t_{end} - t_{start}$$

機械学習における重要性:

「2025年5月1日」という日付そのものよりも、**「それは日曜日だったか？（在宅率に関係）」** や **「前回の接触から何日空いたか？（記憶の定着に関係）」** といった情報のほうが、予測モデルにとっては価値ある**特徴量**になります。

日付要素の抽出と差分計算

基本構文

Datetime型のメソッド:　「datetime(2025-05-01 00:00:00)」

→ .dt.year ⇒ 2025

→ .dt.month ⇒ 5

→ .dt.day ⇒ 1

→ .dt.day_name(locale='jpn') ⇒ 木

→ .dt.dayofweek ⇒ 4

※locale=jpnはWindows適用、Linuxの場合locale=ja_JP.utf8

日付要素の抽出と差分計算

使用例: 1

「年」「月」「曜日」

を抽出し、さらに「キャンペーン開始日からの経過日数」を計算します。

```
# 1. 要素の抽出
df_time['年'] = df_time['contact_date'].dt.year
df_time['月'] = df_time['contact_date'].dt.month
# day_name()で曜日名を取得（設定されたロケールに従って出力されます）
```

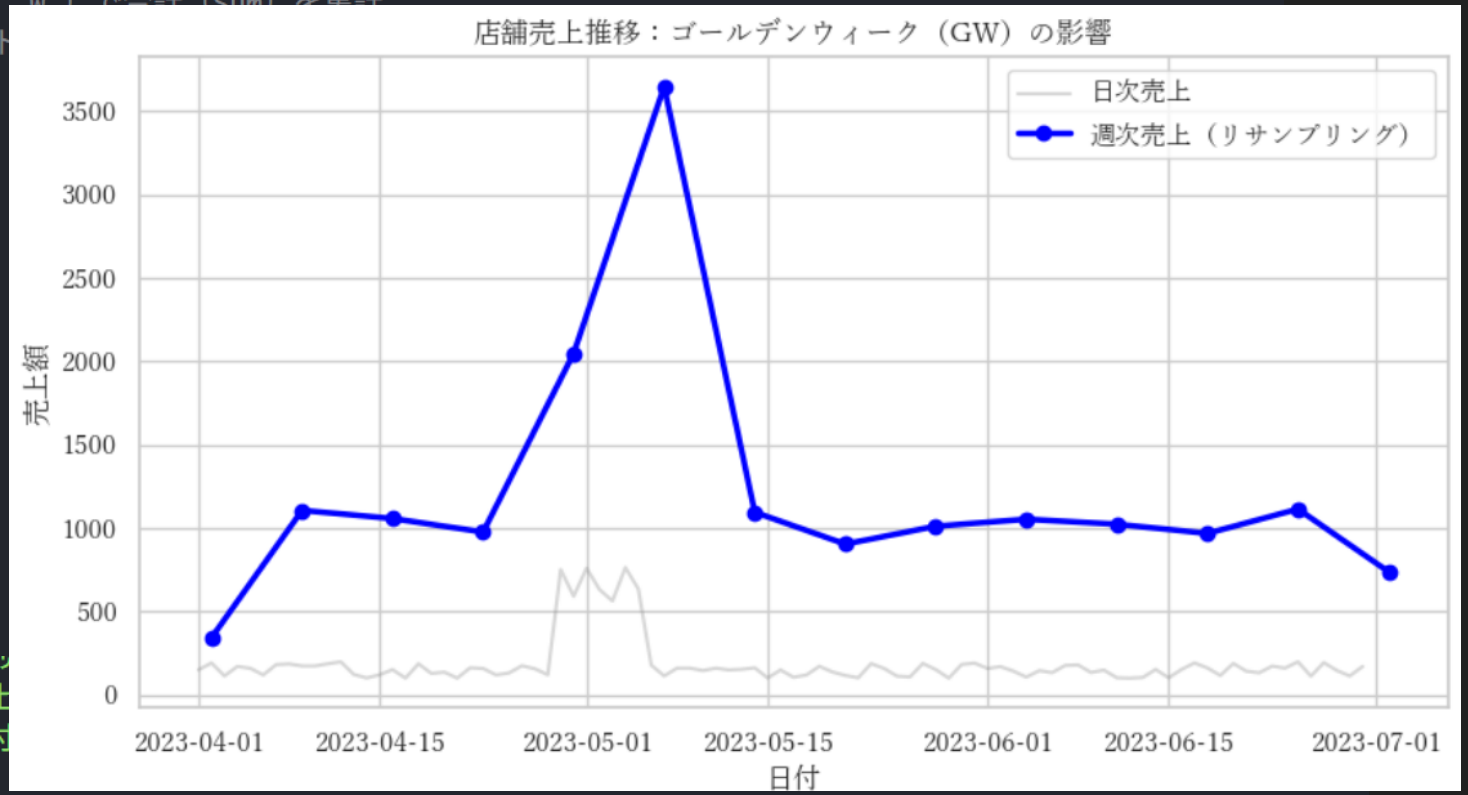
	contact_date	月	曜日	週末フラグ	経過日数
0	2023-05-01 00:00:00	5	月曜日	False	30
1	2023-05-15 00:00:00	5	月曜日	False	44
2	2023-06-01 00:00:00	6	木曜日	False	61
3	2023-06-10 14:30:00	6	土曜日	True	70

```
# 日数として数値化（dt.days属性で日数部分だけを取り出す）
df_time['経過日数'] = df_time['delta'].dt.days

print("--- 時系列特徴量の生成結果 ---")
display(df_time[['contact_date', '月', '曜日', '週末フラグ', '経過日数']])
```

```
# 3. リサンプリングによる集計
# 日付をインデックスに設定（resampleには必須）
df_sales.set_index('日付', inplace=True)

# 週次（Weekly: 'W'）で合計（Sum）を集計
# 'W'はデフォルト
```

週次	売上
2023-04-02	343
2023-04-09	1107
2023-04-16	1058
2023-04-23	977
2023-04-30	2049

```
plt.ylabel('売上
plt.xlabel('日付
plt.legend()
plt.grid(True)
plt.show()
```



目次

時系列データの加工

ひっかけ罠：なぜ単純な引き算ではだめなのか？

地球は球体（正確には楕円体）であるため、

平面上の距離：

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

を適用すると、誤差が生じます。

解決策：測地線距離

距離計算には、一般的にHaversineの公式などが用いられます。

Pythonでは geopy ライブラリを使用して実装できます

$$d = 2r \arcsin(\sqrt{\sin^2(\frac{\phi_2 - \phi_1}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\frac{\lambda_2 - \lambda_1}{2})})$$

緯度・経度と距離計算

基本構文

```
from geopy.distance import geodesic

# 座標は（緯度, 経度）のタプルで指定
point_A = (35.6895, 139.6917) # 東京
point_B = (34.6937, 135.5023) # 大阪

# 距離を計算（キロメートル単位）
distance = geodesic(point_A, point_B).km
```

日付要素の抽出と差分計算

使用例

顧客の座標と支店の座標から、その間の距離（km）を計算します。

計算の関数をデータフレームに適用するために、

`df.apply(関数)`

を使用します

```
# 距離計算関数の定義
def calculate_distance(row):
    # （緯度，経度）のタプルを作成
print("--- 距離計算結果 ---")
display(df_geo)
```

✓ 0.0s

```
--- 距離計算結果 ---
```

	customer_id	cust_lat	cust_lon	branch_lat	branch_lon	distance_km
0	1	35.6895	139.6917	35.6812	139.7671	6.887093
1	2	35.6586	139.7454	35.6812	139.7671	3.185542

```
])
```

目次

高度な欠損値補完

パート４の「**平均値埋め**」や単純な「**回帰補完**」に加え、

多重代入法（Multiple Imputation）の手法を紹介します。

単純な代入の問題点

1. 欠損値を値で埋めると、データが本来持っている「ばらつき（分散）」が失われる
2. 統計的な信頼性が過大評価される（実際よりも確信度が高すぎる結果が出る）恐れがあります。
3. 他の特徴量との関係性が無視される（予測モデルに誤算を生じさせる）

多変量連鎖方程式による代入法

他のすべての変数を説明変数とした回帰モデルによって予測して埋めます。

- 本来の関係性を保持できる

数学的イメージ

変数 X_1（年収）が欠損している場合、他の変数 X_2（年齢）、X_3（職種）などを使って予測します。

$$X_1 = f(X_2, X_3, \ldots) + \epsilon$$

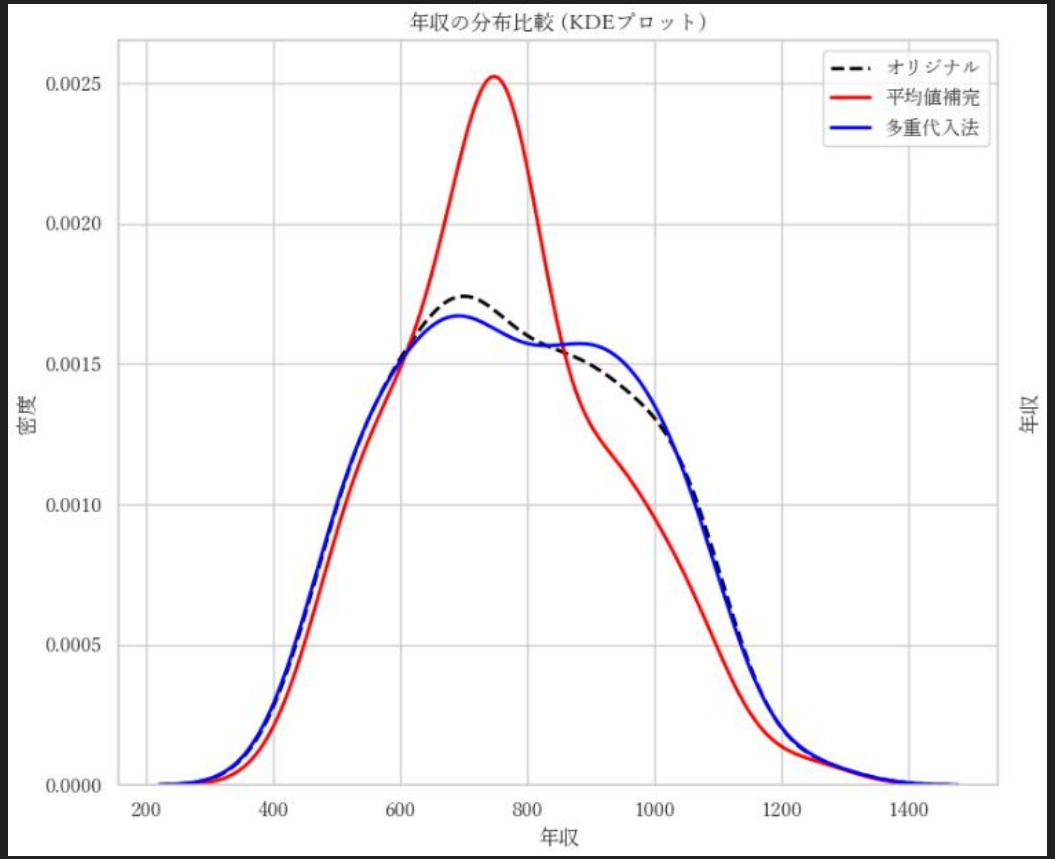
scikit-learnの `IterativeImputer` はこのアプローチを実装しており、精度の高い補完が期待できます。

単純欠損値補完とのパフォーマンス比較

二つの手法の分布への影響を比較

平均値補完: ピークが強調され、元のデータの分散が大分減った

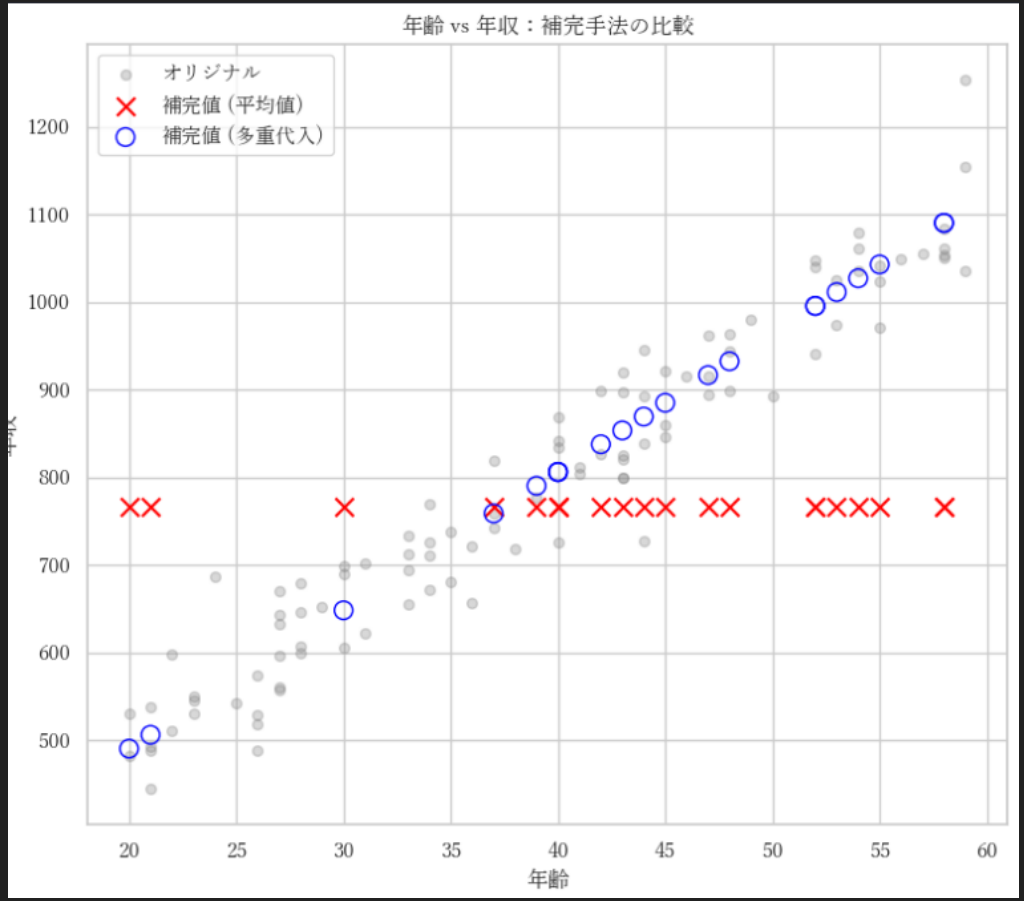
多重代入法: 元のデータとよく似た分布が保持され、分散にも影響が少ない



単純欠損値補完とのパフォーマンス比較

注意点

本来、個々のデータにはモデルでは説明しきれない「**誤差（ノイズ）**」が含まれて散らばっているはずですが、代入値は線上に並びすぎる傾向があります。



多変量連鎖方程式による代入法

基本構文

```
from sklearn.impute import IterativeImputer

# インスタンス化（max_iterはベイズモデル計算の繰り返しの回数）
imputer = IterativeImputer(max_iter=10, random_state=0)

# データの学習と変換（欠損値が埋まった配列が返る）
imputed_data = imputer.fit_transform(欠損を含むデータフレーム)
```

多変量連鎖方程式による代入法

使用例

年齢や経験年数といった他の情報を利用して、欠損している年収情報を推定して埋めます。

```
# IterativeImputerの適用
# max_iter: 推定の繰り返しの最大回数
# random_state: 再現性のための乱数シード
imputer = IterativeImputer(max_iter=10, random_state=42)

# 学習と変換を一括実行
# 返り値はnumpy配列になるため、DataFrameに戻す必要があります
imputed_array = imputer.fit

# 結果をDataFrameに戻す
df_imputed = pd.DataFrame(i
                          columns
print("\n--- 補完後（Iterati
print("年齢や経験年数との相関
display(df_imputed)
```

```
す")
```

	age	experience	income
0	25.0	1.0	300.000000
1	30.0	5.0	450.000000
2	45.0	20.0	810.495804
3	35.0	10.0	550.000000
4	50.0	25.0	936.985119
5	23.0	1.0	280.000000

総合ハンズオン: 訪問ルート最適化

欠損値補完

データベース操作

機械学習

シナリオ

あなたは銀行の営業推進部のデータアナリストです。
営業担当者が顧客を訪問する計画を立てるため、以下の情報を含む「営業支援データマート」を作成するよう依頼されました。

タスク

ミッションの流れ (ETL):

1. **データの抽出 (Extract):** 顧客データと、今回新たに提供される「支店位置情報マスタ」を読み込む。
2. **時系列加工 (Transform - Time):** 日付データの変換と経過日数の計算。
3. **位置情報加工:** 顧客と支店の位置情報から距離を計算。
4. **欠損値補完:** 年収の欠損を高度な手法で補完。
5. **データの格納:** 結果を `sales_support_mart` テーブルに格納。