実務者のための
Python機械学習超入門

愛知県

本日の内容

1.データ分析の重要性
2.Pythonの基礎知識
3.顧客満足度分析演習
4.需要予測分析演習

本講座のゴール

1.Pythonによるデータ分析を身につける
2.機械学習の概要を理解する
3.データ分析を実務に活用するヒントを得る

データ分析の重要性

データ分析の重要性

- データ分析とは、**仮説を検証するための道具**である
- データ分析とは、**意思決定するための道具**である

- 仮説を立てるのは人間
- データ分析結果を解釈するのは人間
- データ分析によりわかった課題を対処するのも人間

- **データ分析は「夢の箱」ではない。**

データ分析で「できること」

- 仮説が正しいかどうかを検証する
- 施策の実施前後の状態を検証する
- 現在および過去の事実を明らかにする

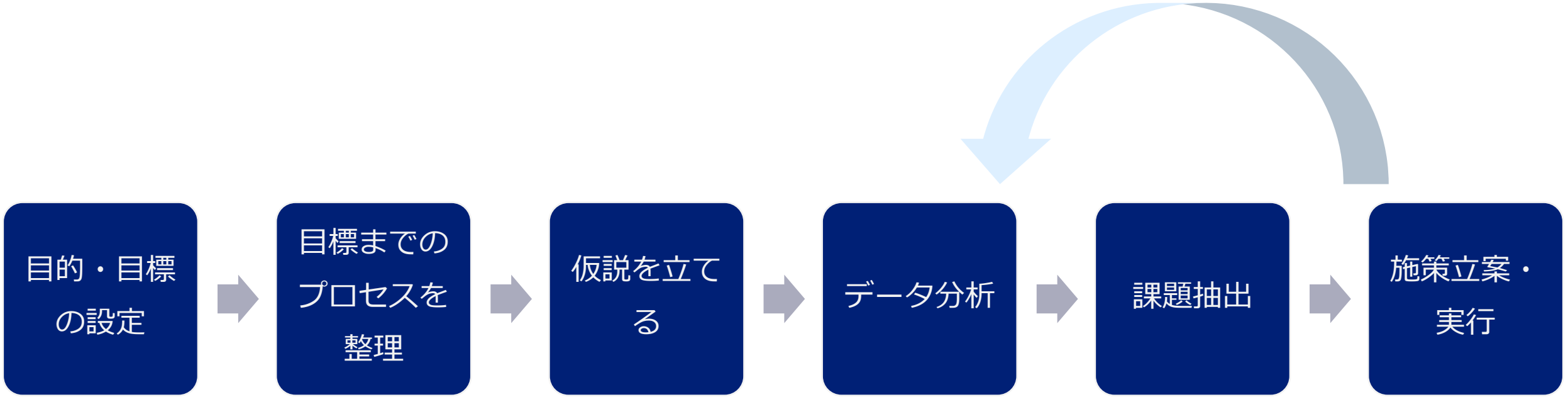
データ分析で「できないこと」

- 未来を完璧に予想する
- 課題を見つけて、課題に対する施策を提示する
- 施策を実行する

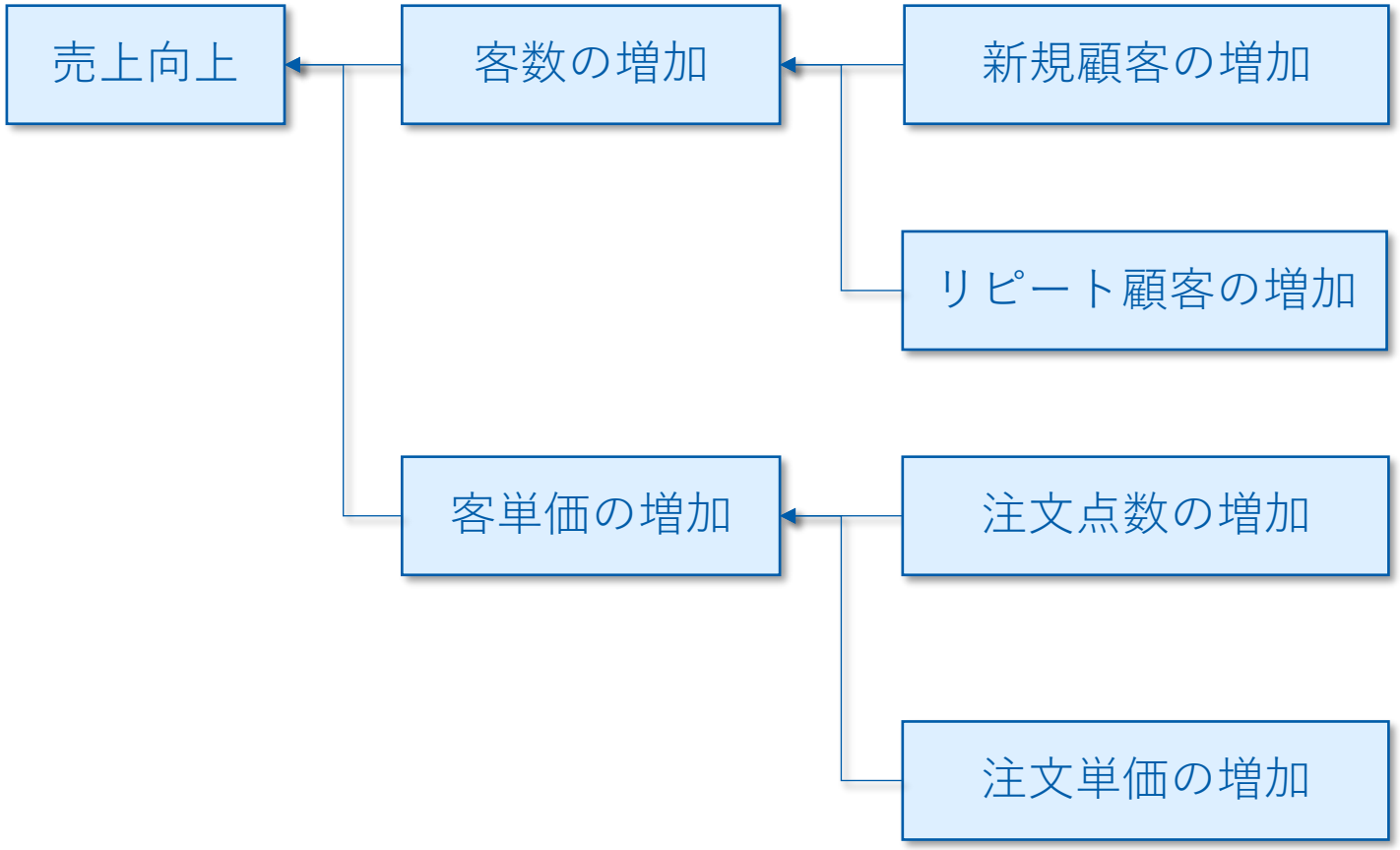
よくある困った相談

- 会社にデータがたくさん溜まっているので、分析してみてほしい
- わが社が今後どうすべきかデータ分析によって決めてほしい
- データ分析で未来を予測してほしい
- AIを導入したい

データ分析の流れ



目標までのプロセスを整理する例



データ分析の重要性

- データを分析することで
  - 仮説を検証することができる
  - 新たな気づきを得られる
  - 改善すべき課題を抽出できる
  - 施策の実施結果を検証できる

- 勘や経験ではなく「データ」に基づいた意思決定が可能となる

Pythonの基礎知識

Pythonとは

- オープンソースの汎用スクリプト言語
- とてもシンプルな言語仕様
- 機械学習のライブラリが豊富
- 世界で最も人気の高い言語
- 名前の由来は、イギリスのテレビ局 BBC が製作したコメディ番組『空飛ぶモンティ・パイソン』

Pythonの実行環境

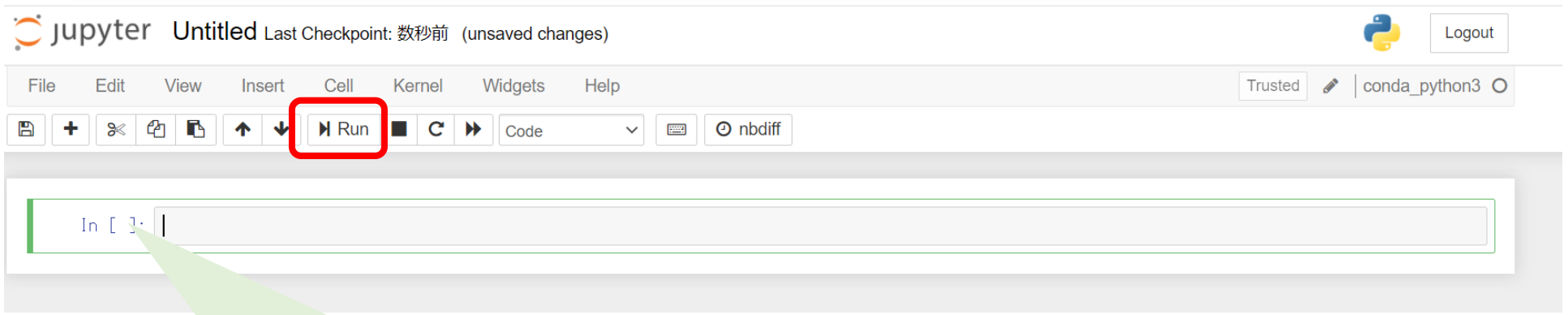
- Anaconda（アナコンダ）
  - ✓ Pythonとデータ分析用ライブラリがセットになったディストリビューション
- PyCharm（パイチャーム）
  - ✓ 総合開発環境（IDE）
  - ✓ 他にはVisual Studio CodeやAtomなど
- Jupyter Notebook（ジュピター）
  - ✓ ブラウザベースの実行環境
  - ✓ ソースコードと結果をセットで配布できる

Pythonの実行環境

- Amazon SageMaker
  - ✓ 機械学習の統合開発環境（SageMaker Studio）
  - ✓ Jupyter Notebookインスタンスを簡単に構築できる

Jupyter Notebookの使い方

- コードを書いてRun（Ctrl + Enter）
- 編集は自動で保存される

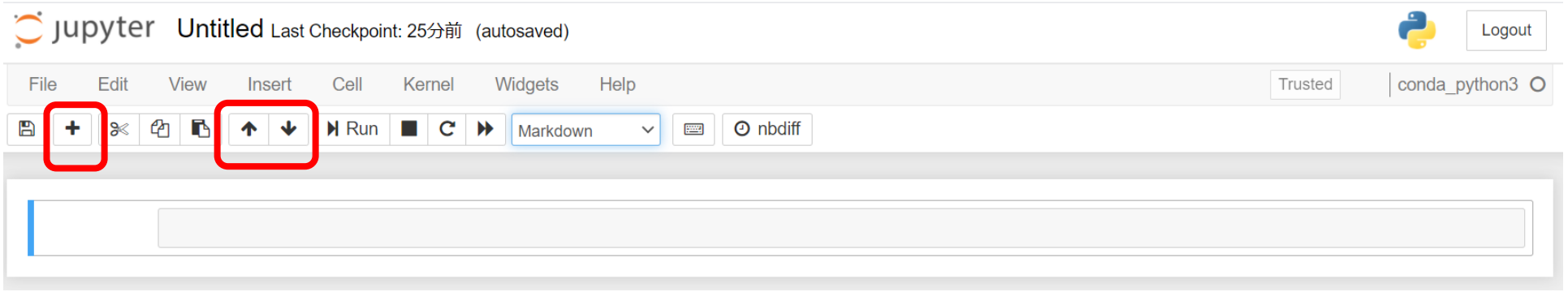


実行中は[*]になる
→ 実行が完了すると数字になる

愛知県

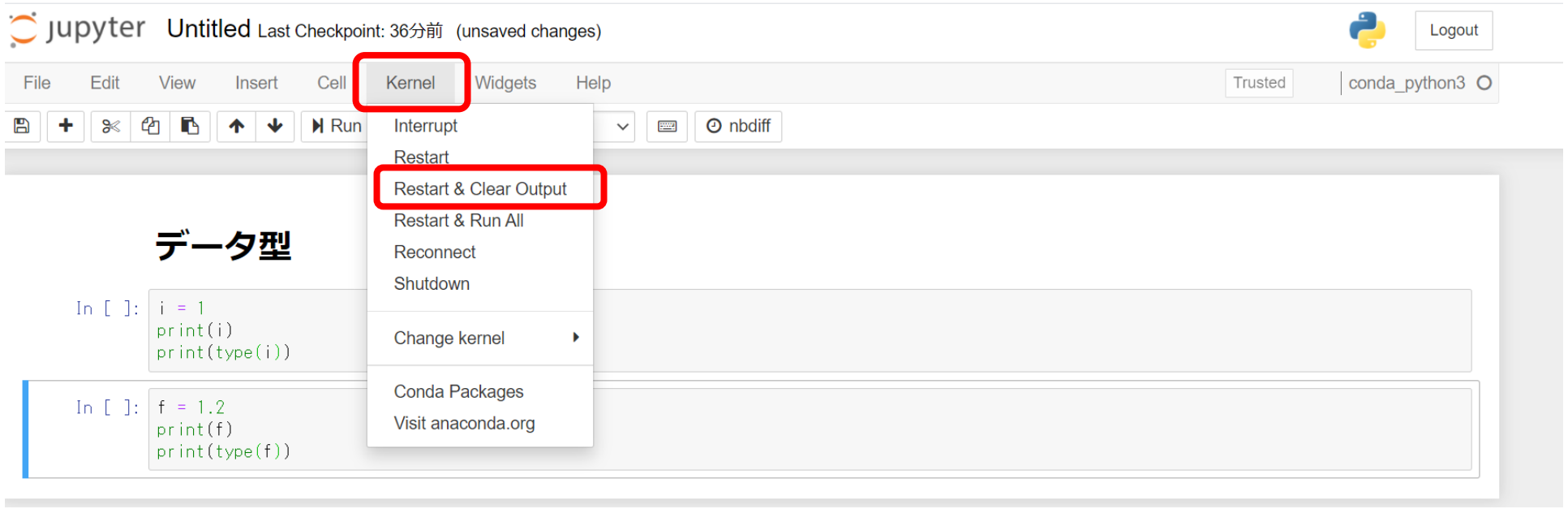
Jupyter Notebookの使い方

- 「＋」でブロックを追加できる
- 「↑」「↓」でブロックを移動できる



Jupyter Notebookの使い方

- Kernel →　Restart & Clear Outputで出力結果をクリアできる



便利なショートカット

ショートカット	説明
[Shift] + [Enter]	セルを実行して下のセルに移動
[Ctrl] + [Enter]	セルの実行
[Alt] + [Enter]	セルを実行して下にセルを追加
[Tab]	入力補完もしくはインデント
[Shift] + [Tab]	ツールチップの表示
[Ctrl] + [/]	コメントの切り替え
[Ctrl] + [Z]	操作の取り消し
[Ctrl] + [Y]	取り消した操作のやり直し
[Ctrl] + [D]	1行削除
[Esc]	コマンドモードに変更

Pythonの基礎知識

データ型

- 整数型（int）
  - ✓ 小数点を含まない数字を表す
- 小数型（float）
  - ✓ 小数点を含む数字を表す（浮動小数点）
- 文字列型（str）
  - ✓ 文字列を表す（「”」or「’」で囲む）
- 真偽値型（bool）
  - ✓ True/Falseの2値を表す

データ構造

- リスト（list）
  - ✓ 複数の要素を格納する（型はなんでもよい）

```
l = [1, 2, 3, “4”, 5]
```

- タプル（tuple）
  - ✓ 要素の変更ができないリスト

```
t = (1, 2, 3, “4”, 5)
```

- 辞書（dict）
  - ✓ キーと値のペア

```
d = {“key1“ : 1, “key2” : “value2”, “key3” : 3 }
```

条件分岐

- if文

```
if 条件１：
        条件１に一致した場合の処理
elif 条件２：
    条件２に一致した場合の処理
else：
        条件１にも条件２にも一致しなかった場合の処理
```

※Pythonはインデントでブロックを表す

繰返し

- for文

for 変数 in 繰り返し対象

✓ 繰り返し対象は繰り返し可能なもの（イテレーター）であれば何でもOK

ライブラリの読み込み

- ライブラリ ＝ モジュール ＝ パッケージ
  - ✓ **import ライブラリ as 別名**

    → ライブラリ全体を読み込んで別名を付ける

    例：import pandas as pd

  - ✓ **from ライブラリ import オブジェクト**

    → ライブラリ内の指定したオブジェクトのみを読み込む

    例：from sklearn.datasets import load_iris

よく使うライブラリ

- pandas（パンダス） 別名：pd
  - ✓ 表形式のデータ（Panel data）を扱う
- numpy（ナムパイ、ナンパイ） 別名：np
  - ✓ 科学技術計算ライブラリ
- matplotlib（マットプロットリブ）別名：plt
  - ✓ グラフ描画ライブラリ
  - ✓ import matplotlib.pyplot as plt
- seaborn（シーボーン） 別名：sns
  - ✓ 高機能なグラフ描画ライブラリ
- scikit-learn（サイキットラーン）
  - ✓ 機械学習ライブラリ

グラフによる可視化

グラフの種類

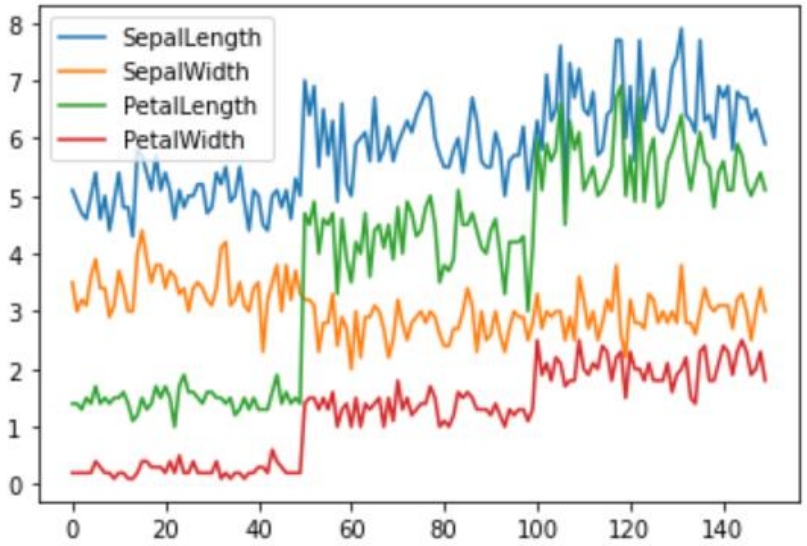
1. 折れ線グラフ
2. 棒グラフ
3. 円グラフ
4. ヒストグラム
5. 散布図

グラフの描画

```
%matplotlib inline
import matplotlib.pyplot as plt
```

```
import pandas as pd

df = pd.read_csv("data/iris.csv")

df.plot()
```

```
<AxesSubplot:>
```

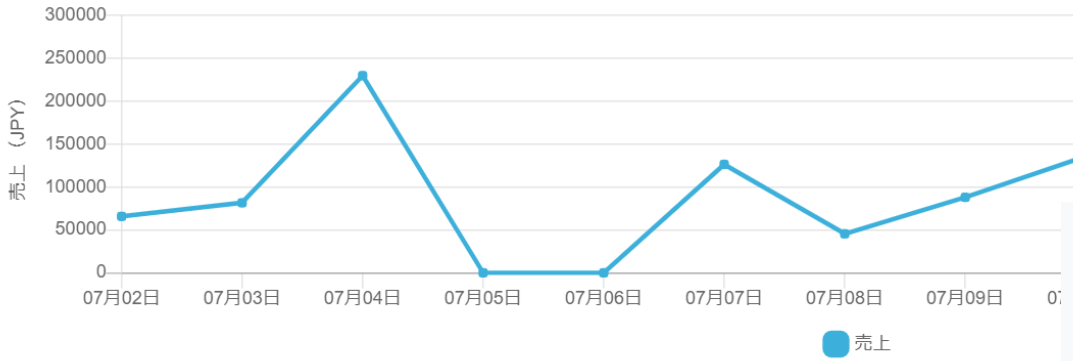


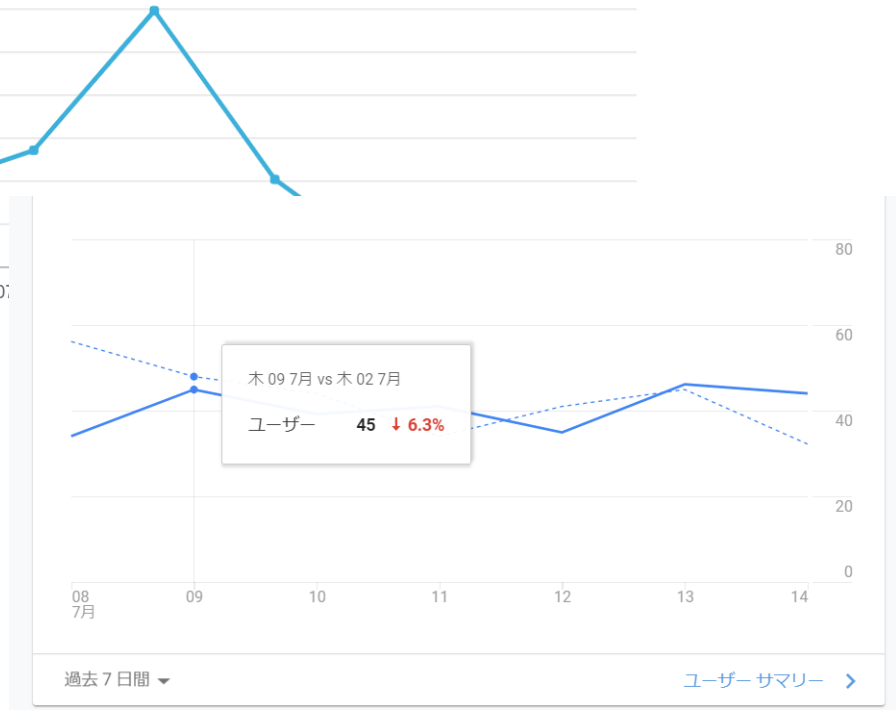
JupyterNotebook内にグラフを描画する指定

愛知県

折れ線グラフ

- 時系列などで値の変化の傾向を把握する





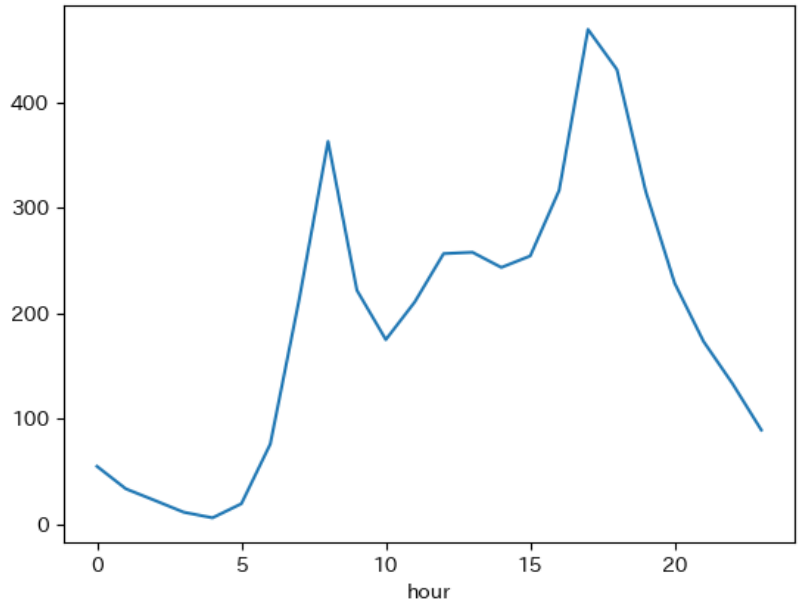
折れ線グラフ

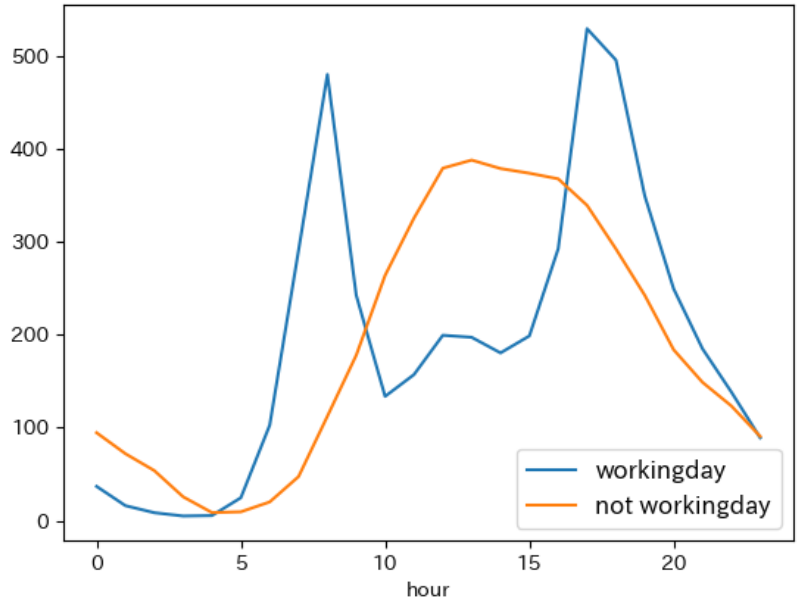
- わかること
  - ✓ 減少傾向にあるか、増加傾向にあるか
  - ✓ 前年同時期と比較してどうか
- わからないこと
  - ✓ 厳密な変化
  - ✓ 厳密な差異

  →厳密な差異を比較するのは棒グラフが向いている

折れ線グラフ

- データを分解することで差異が見える



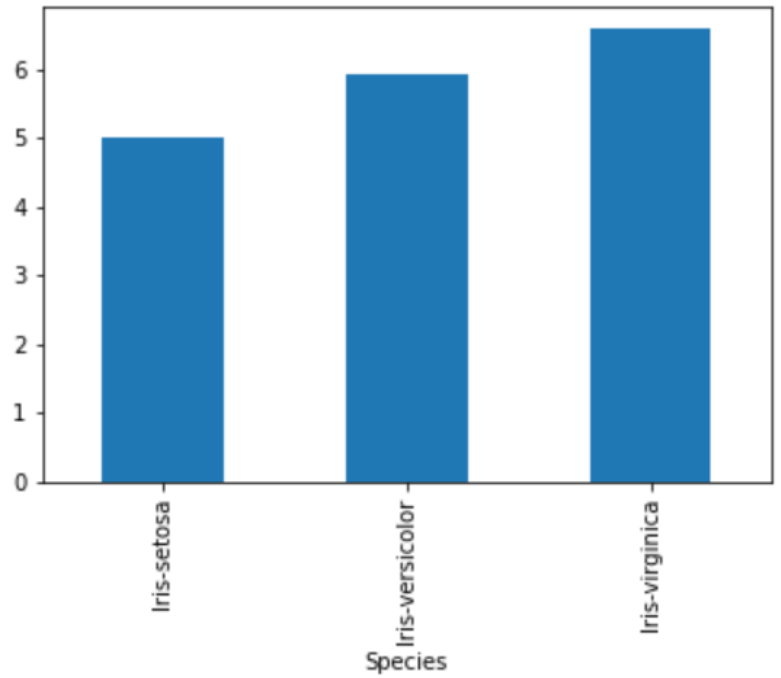


棒グラフ

- 値の大小を比較する

```
df.groupby("Species")["SepalLength"].mean().plot.bar()
```

```
<AxesSubplot:xlabel='Species'>
```

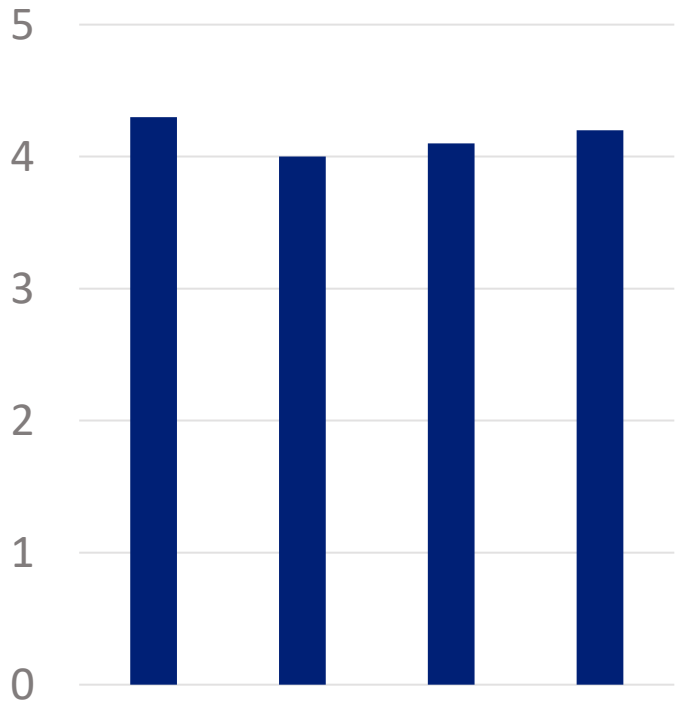


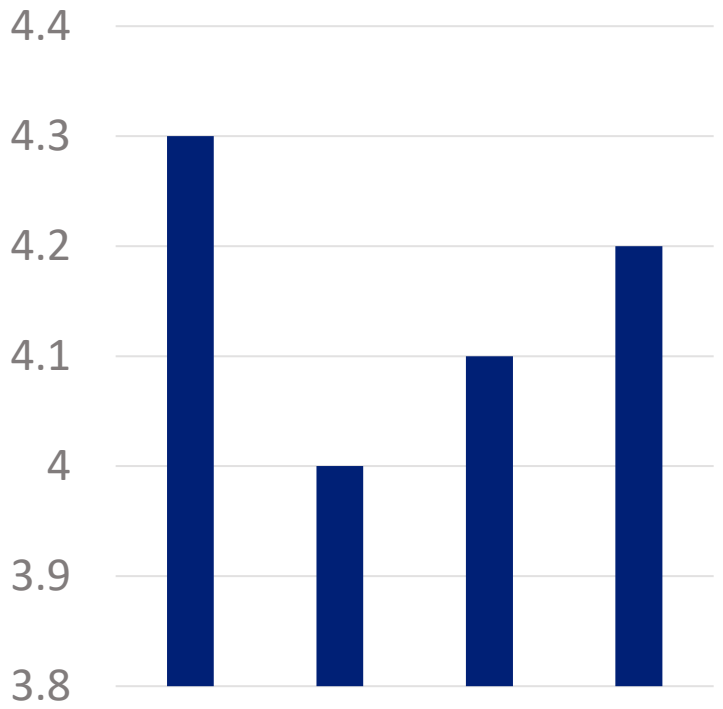
棒グラフ

- わかること
  - ✓ どちらの値が大きいか
- わからないこと
  - ✓ 値の推移の傾向（→折れ線グラフ）
  - ✓ 最大値、最小値（→ソートする）

棒グラフ

- 軸のスケールによって差異を明確にできる
  - ✓ 恣意的にならないように注意





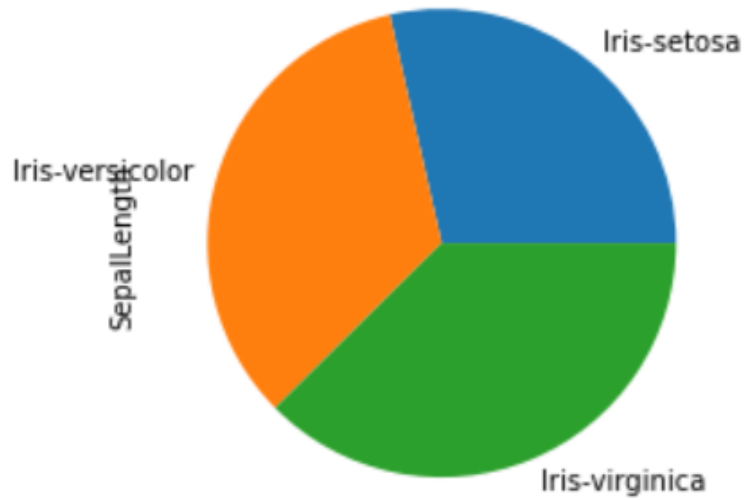
愛知県

円グラフ

- ざっくりとした割合を把握する

```
df.groupby("Species")["SepalLength"].mean().plot.pie()
```

```
<AxesSubplot:ylabel='SepalLength'>
```

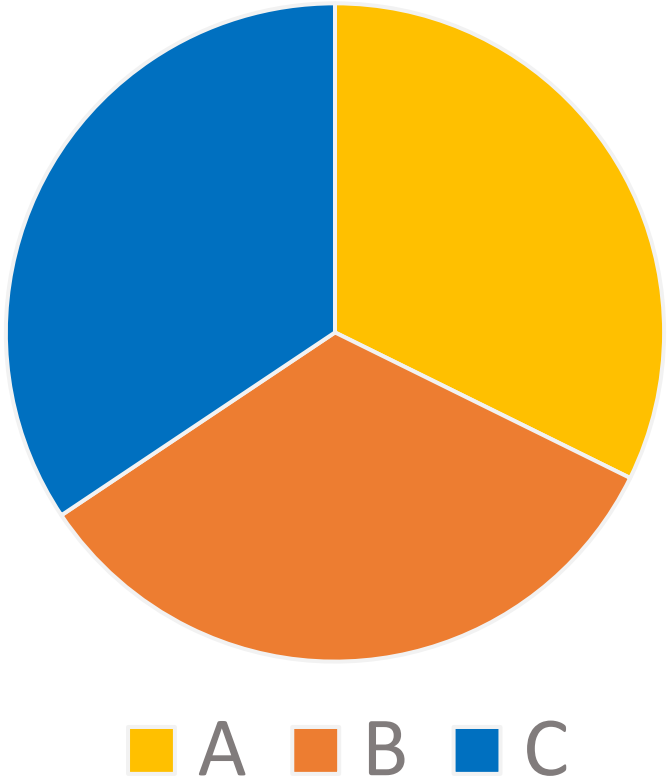


円グラフ

- わかること
  - ✓ ざっくりとした割合のイメージ
- わからないこと
  - ✓ 厳密な差異
    - → 厳密な差異を把握するには棒グラフが向いている

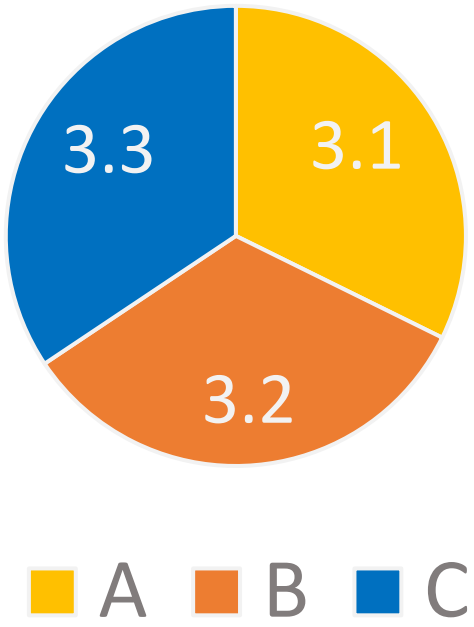
円グラフ

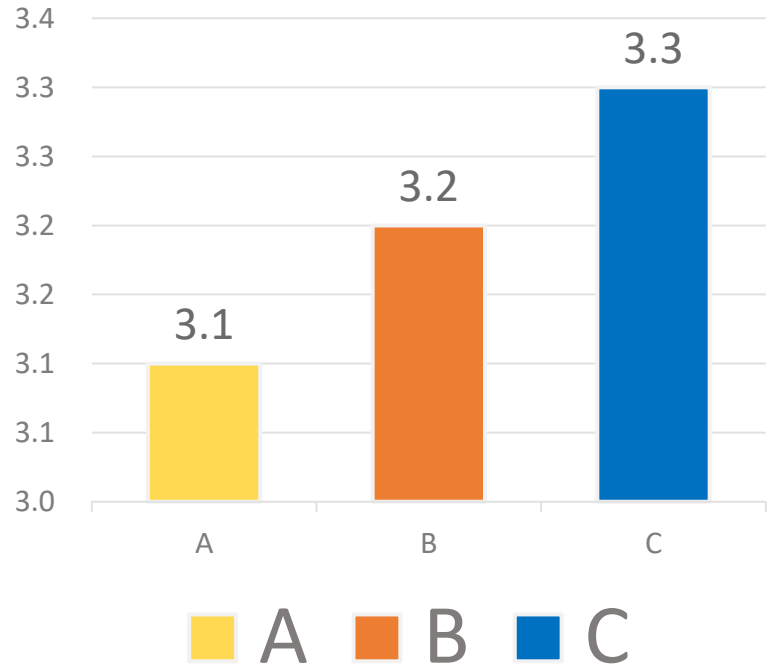
- 円グラフは微妙な違いがわからない
  - ✓ 色によっても人が感じる大きさは異なる



円グラフ

- データラベルを付けると差異が明確になる
- 棒グラフにすると差異が明確になる





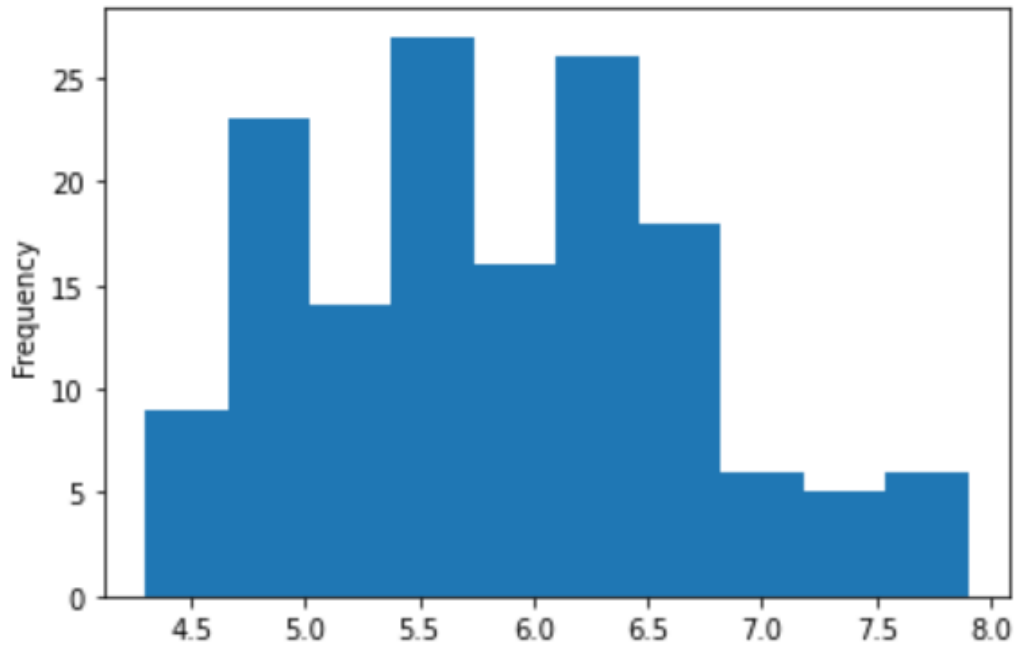
- 円グラフによる印象操作

ヒストグラム

- 値の分布を確認する（正規分布）

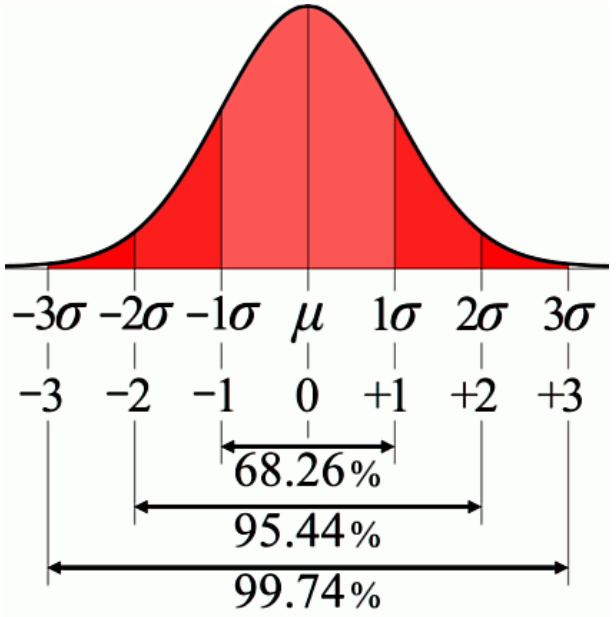
```
: df["SepalLength"].plot.hist()
```

```
: <AxesSubplot:ylabel='Frequency'>
```



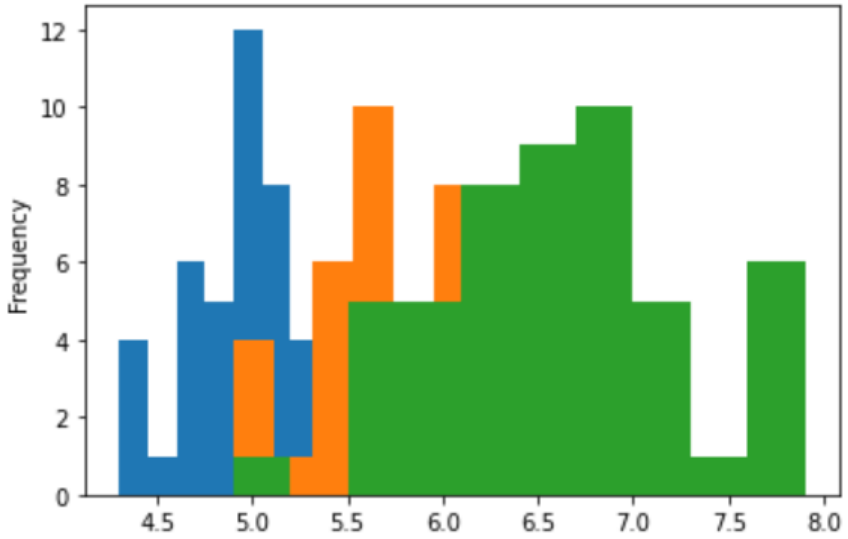
正規分布①

- 正規分布＝中央が盛り上がった左右対称型
  - ✓ 世の中の事象は基本的に正規分布となる
  - ✓ さまざまな分析手法は分析対象が正規分布であることを前提としている
- 正規分布の幅 ＝ 標準偏差（σ）
  - ✓ ±3σに99.74%の値が存在
  - ✓ 平均値を50、標準偏差を10として正規化したものが偏差値



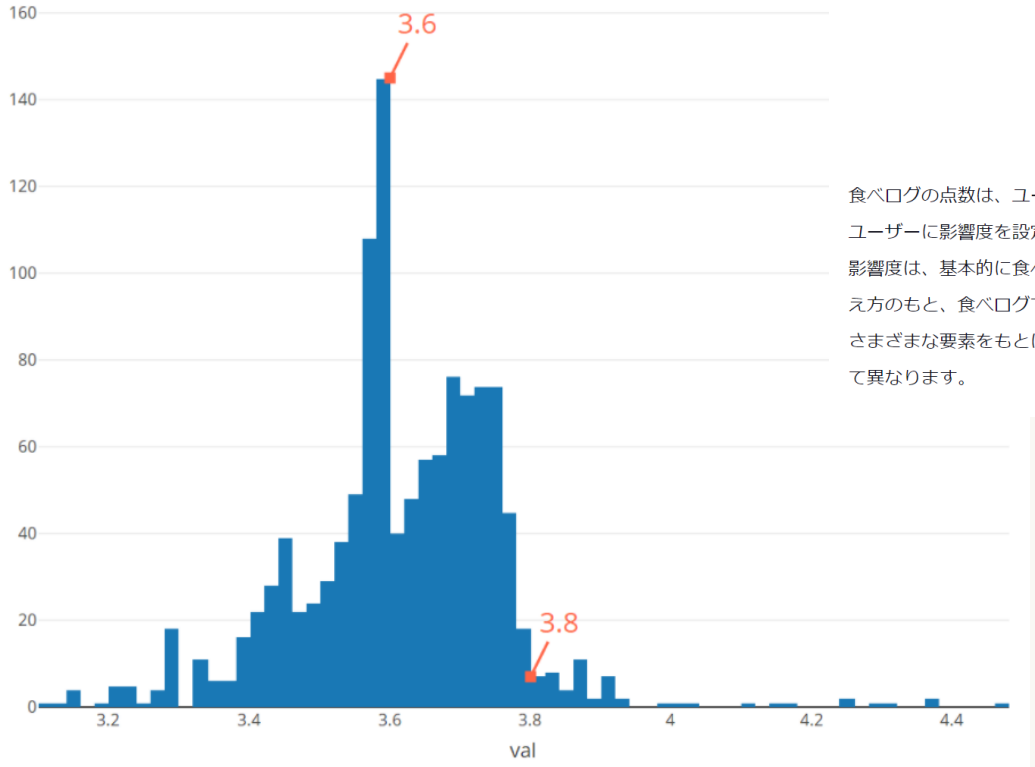
正規分布②

- 山が複数になっている場合（正規分布になっていない場合）は、以下を疑う
  - ✓ 複数のデータが混在している → データを分けて分析する
  - ✓ サンプルが少ないため偏っている → データを増やす



ヒストグラム

- 食べログ3.8問題



評価点のヒストグラム

1

点数の決まり方について

ユーザー影響度

食べログの点数は、ユーザーの各評価を単純平均したものではありません。
ユーザーに影響度を設定して、点数の算出要素の一つにしています。
影響度は、基本的に食べ歩きの経験が豊富な方の影響を大きくするという考え方のもと、食べログでの投稿をある程度繰り返しているユーザーについてさまざまな要素をもとに設定しています。そのため影響度はユーザーによって異なります。

飲食店向け「集客サービス（有料）」を利用しているかどうかが点数に影響することは一切ありません。

食べログでは飲食店向けに有料の集客サービスを提供していますが、
その内容は、検索の並び順の１つ「標準リスト」において優先的に表示されることで集客を増やすサービスです。
よって、飲食店向け「集客サービス（有料）」を利用しているかどうかが点数に影響することは一切ありません。
点数はあくまでもユーザー評価をもとに算出した数値であり、ユーザーのお店選びをさらに便利にする、という食べログのポリシーに沿って継続的なアップデートを行っています。

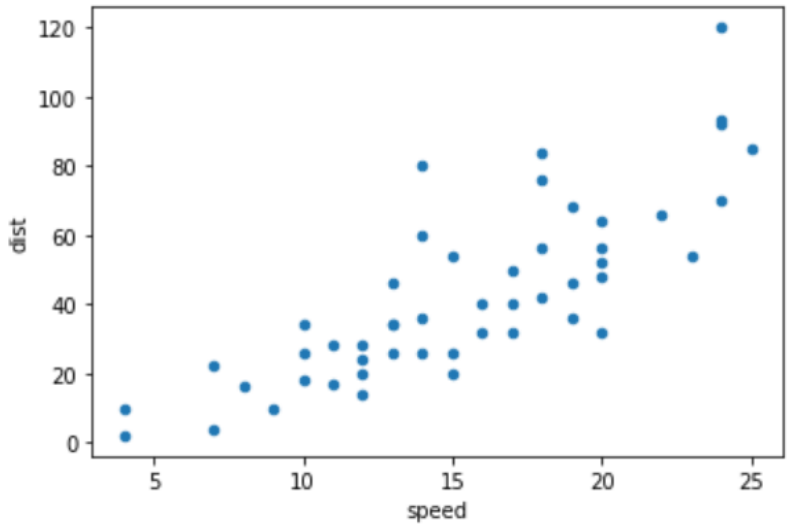
https://clean-copy-of-onenote.hatenablog.com/entry/tabelog38_problem

散布図

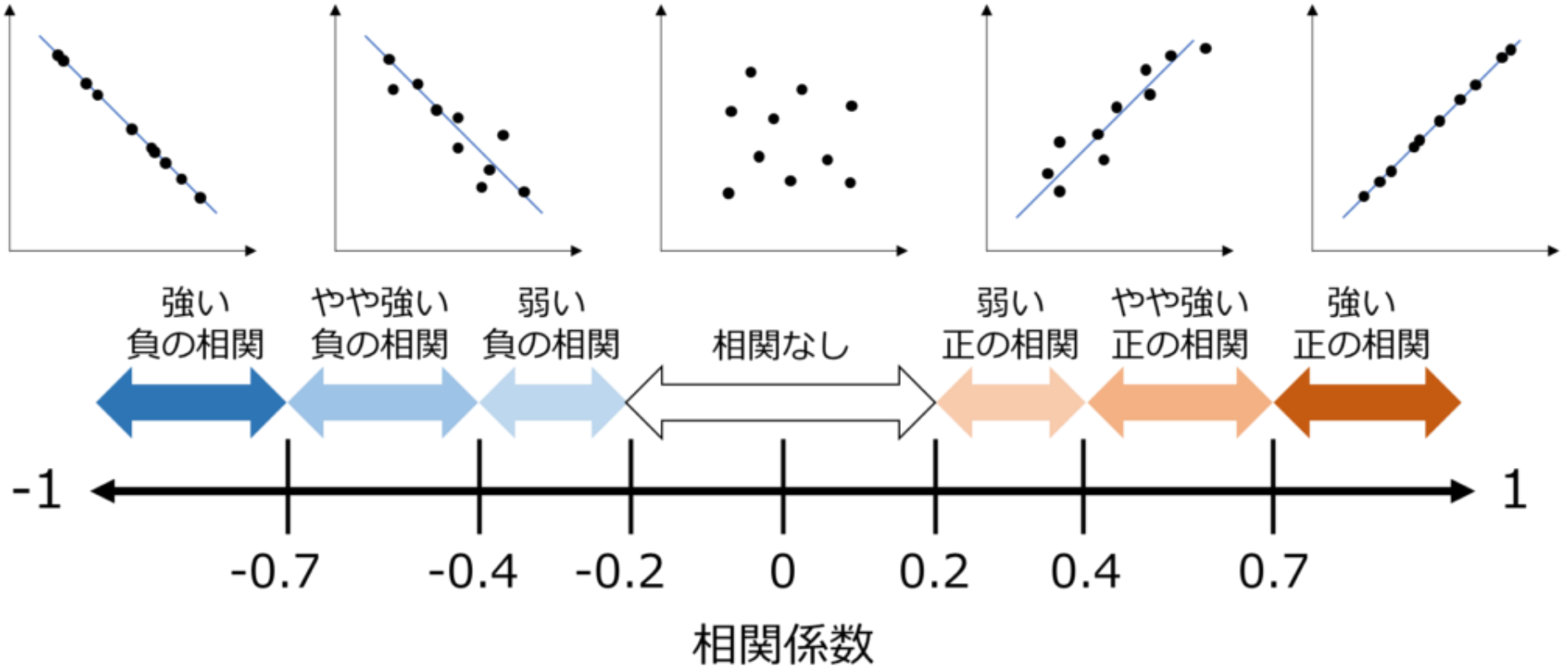
- ２変数の間の関係性（相関関係）を把握する
  - ✓ スピード（説明変数）と停止距離（目的変数）の関係

```
: import pandas as pd
  cars = pd.read_csv("data/cars.csv")
  cars.plot.scatter(y = "dist", x = "speed")
```

```
: <AxesSubplot:xlabel='speed', ylabel='dist'>
```

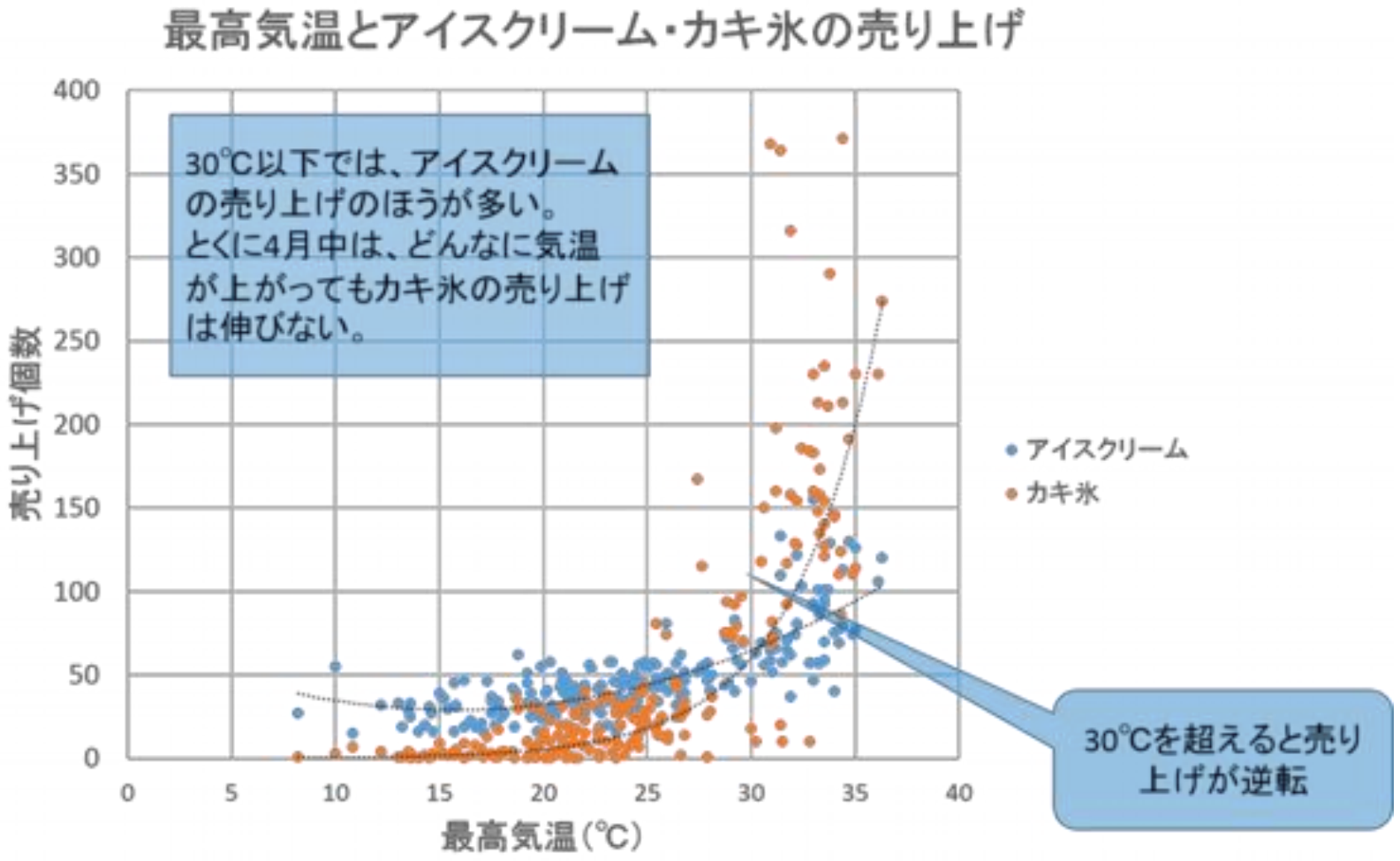


相関関係と因果関係



出典：**【メモ】心理統計：相関関係**
https://note.mu/peoplepersons/n/nfc6a94aed61d

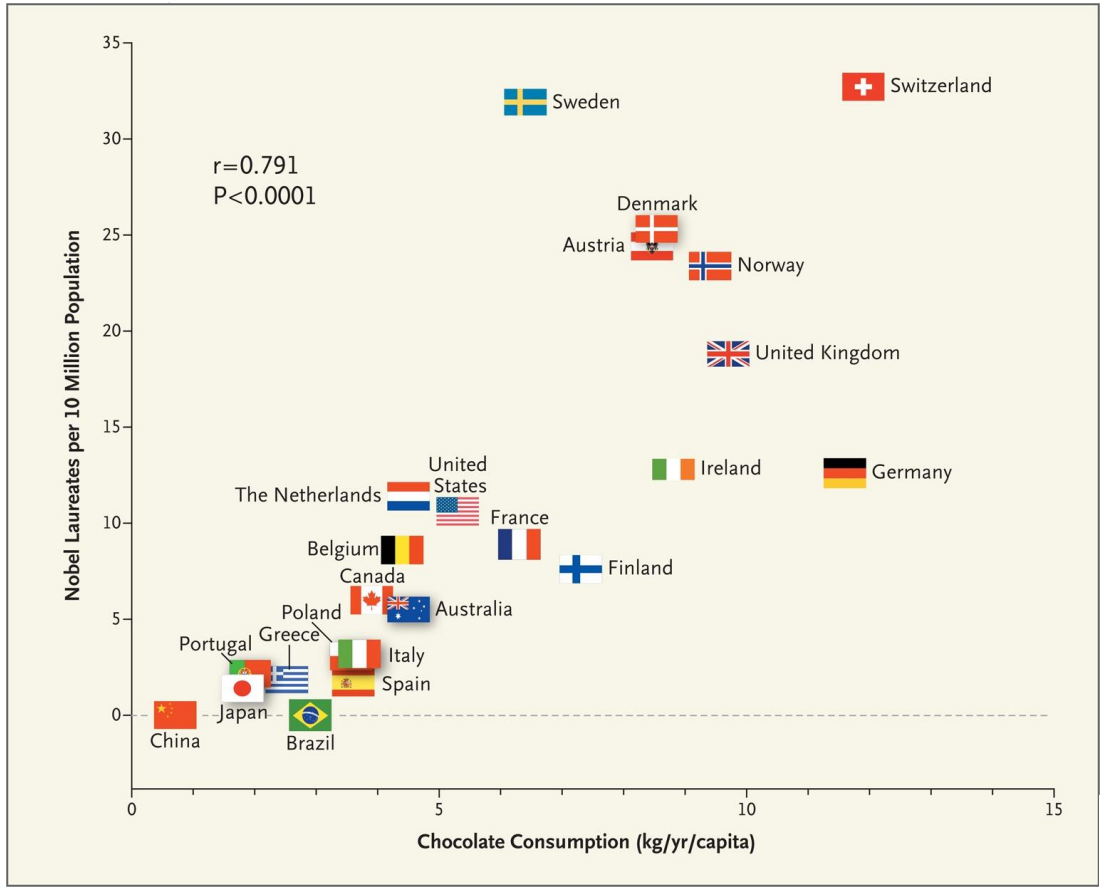
相関関係と因果関係



出典：生活気象ラボ http://lab.lbw.jp/c-ok-ice.html

相関関係と因果関係

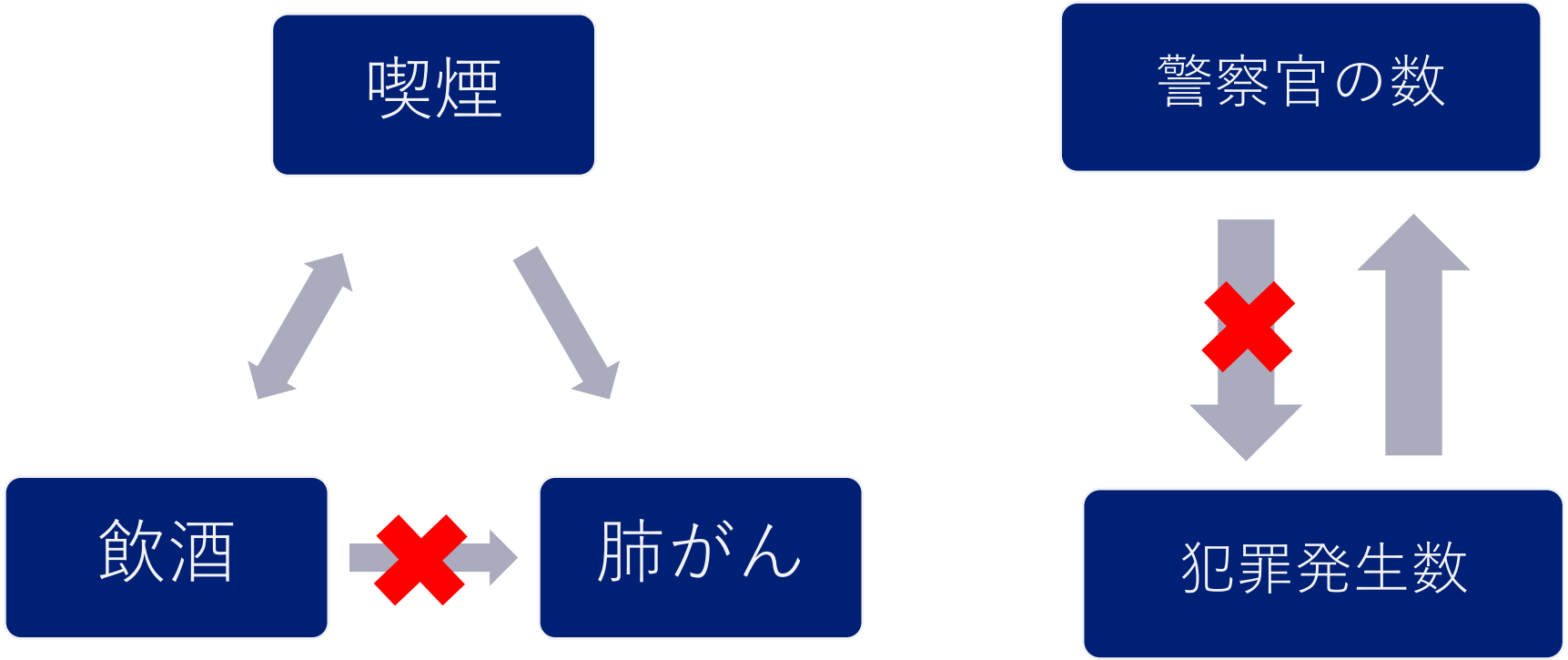
- チョコレートの消費量とノーベル賞の受賞数の



出典：

The NEW ENGLAND JOURNAL of MEDICINE

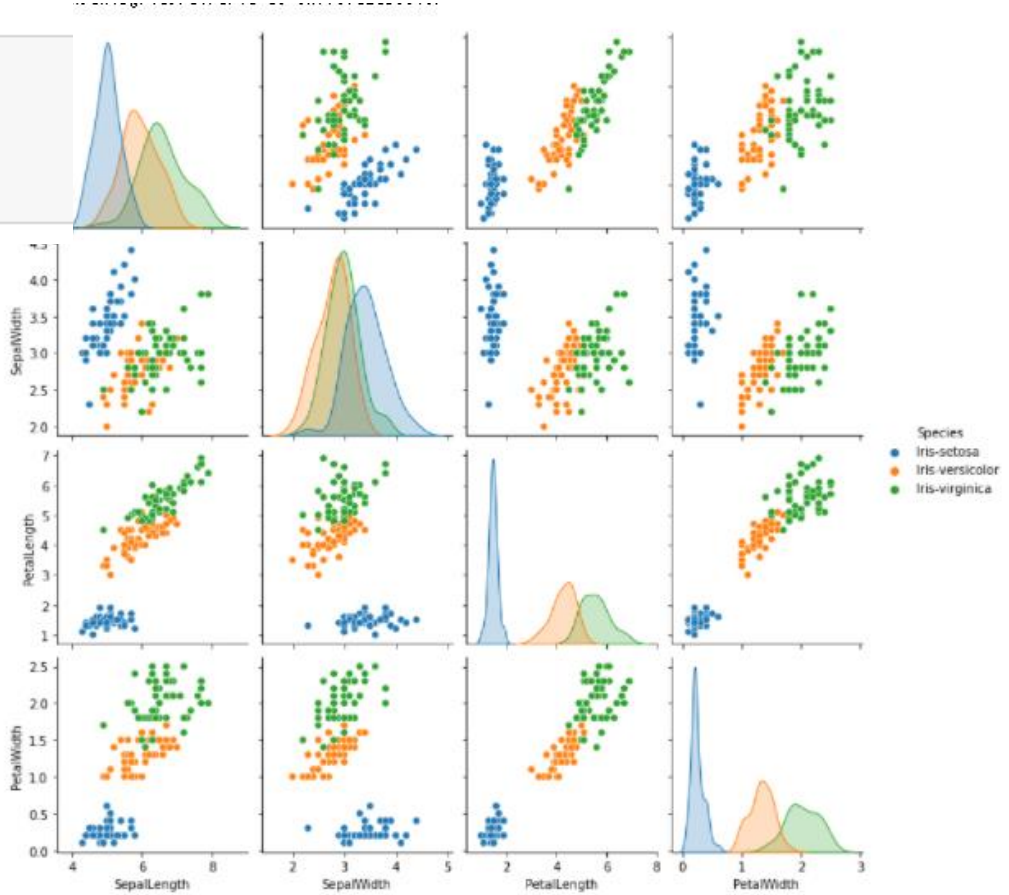
相関関係≠因果関係



ペアプロット

- すべての列の組み合わせで散布図を表示

```
import seaborn as sns
iris = pd.read_csv("data/iris.csv")
sns.pairplot( data =iris , hue ="Species")
```

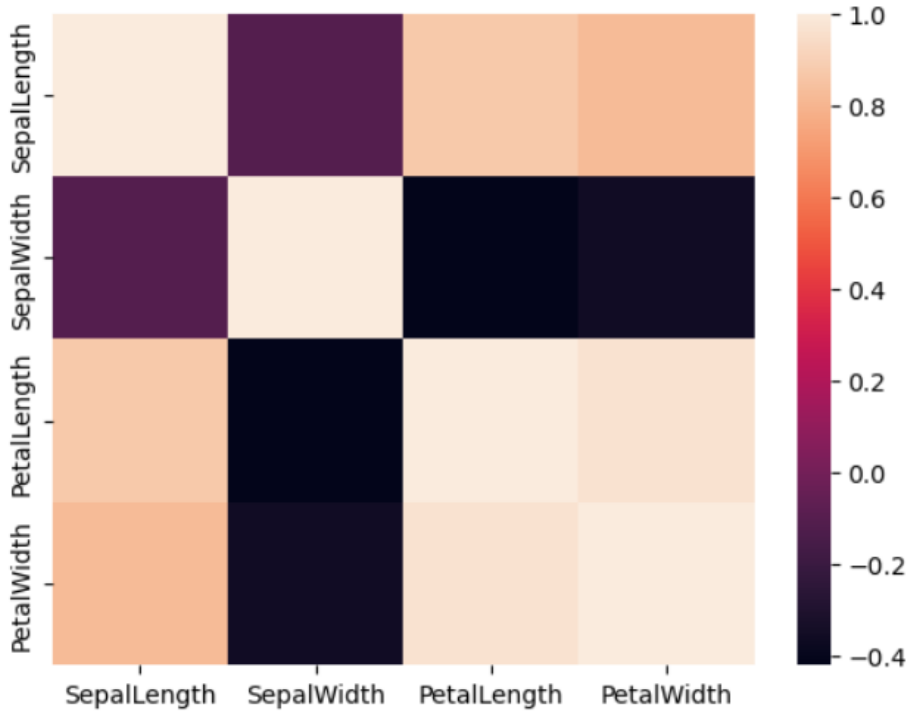


ヒートマップ

- 相関係数を色分けして表示

```
iris = iris.drop('Species', axis=1)
sns.heatmap(iris.corr())
```

```
<Axes: >
```



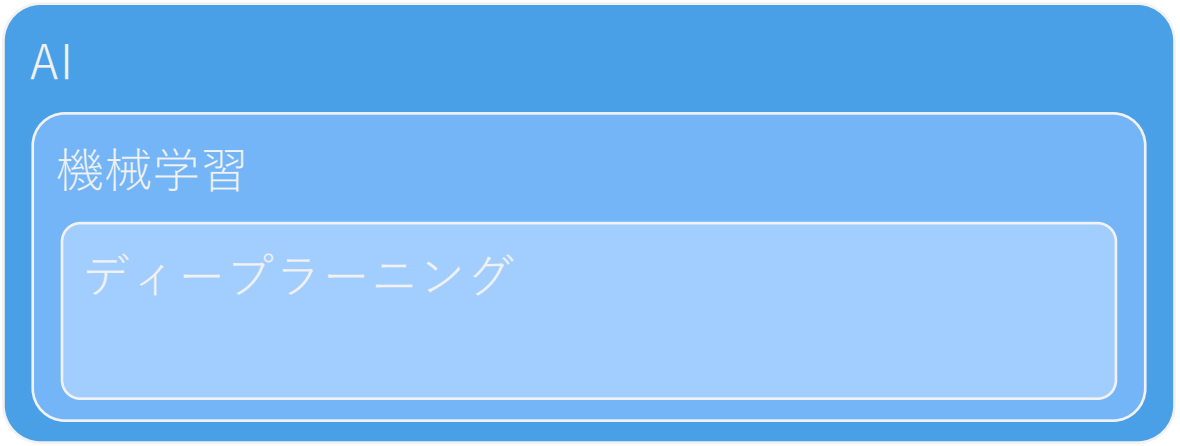
グラフによる可視化のまとめ

- グラフにより可視化することでデータを理解できる
  - ✓ まずはグラフを書いてみる
- 目的に合わせてグラフを使い分ける
- 分解することで埋もれた差異を把握できる

機械学習基礎

機械学習とは

- AIを実現する技術の一部
- 大量のデータからルールやパターンを導き出す
  - ✓ 人ではルール化できない複雑な問題も処理できる
  - ✓ 予測や識別に活用できる
- ディープラーニングは機械学習の一種



機械学習の種類

教師あり学習

入力データと「答え」を与える
（正解を知っている）

■主な分析手法

- 最近傍法
- 線形モデル
- ナイーブベイズ
- 決定木
- ランダムフォレスト
- 勾配ブースティング
- サポートベクターマシン
- ニューラルネットワーク

教師なし学習

入力データ「のみ」を与える
（正解を知らない）

■主な分析手法

- 主成分分析（PCA）
- 非負値行列因子分解（NMF）
- 多様体学習（t-SNE）
- クラスタリング（k-means、凝集型、階層型、DBSCAN）

強化学習

条件のみを与えて、自身で繰り返し勝手に学習する

■主な分析手法

・Q学習
・DQN

教師あり学習

- 入力データと答えを学習させて、答えを予測するモデルを構築する
  - ✓ 手書き文字認識
  - ✓ 顔認識
  - ✓ 迷惑メールフィルタ
- 人間が特徴を把握できない複雑な問題を対処できる
- あらかじめ答えを用意できないと学習できない
- 予測値の的中度合いでモデルの精度を評価する

教師あり学習の種類

分類（Classfication）
クラス（カテゴリ）を予測する

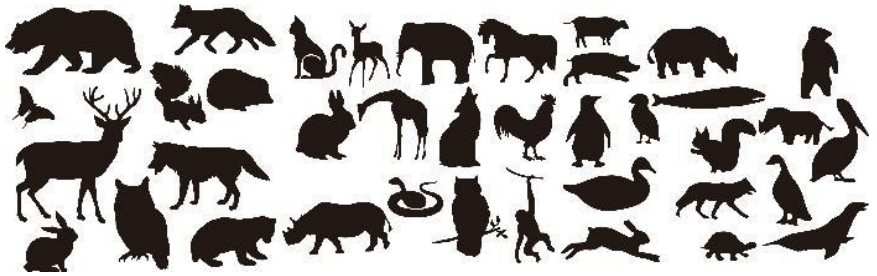
■主な分析手法
- K-最近傍法
- ロジスティック回帰
- サポートベクターマシン
- 決定木（分類木）
- ランダムフォレスト
- 勾配ブースティング
- ニューラルネットワーク

回帰（Regression）
数値（連続値）を予測する

■主な分析手法
- 線形回帰
- リッジ回帰
- Lasso回帰
- 決定木（回帰木）
- ランダムフォレスト
- 勾配ブースティング
- ニューラルネットワーク

教師なし学習

- 入力データのみを与えて、自動で分類を作成する
  - ✓ 顧客セグメンテーション
  - ✓ 画像の自動分類
  - ✓ 信号分析



- 分類値を持っていないデータに分類を付与する
  - ✓ 教師あり学習の前処理として使用する
- 答えを教えなくても勝手に学習するものではない
- 結果が妥当かどうかを判断するのは人間

強化学習

- 現在の状態から価値を最大化する行動を学習する
  - ✓ 将棋、囲碁、ゲーム
  - ✓ 自動運転
  - ✓ 産業ロボット
- 価値（どのような状態となるのが理想か）を定義する必要がある
  - ✓ 価値を定義できないと学習できない
  - ✓ 価値を定義できれば、勝手に学習する
  - ✓ 繰り返し学習することで、どんどん賢くなる

機械学習で出来ること

- ある目的に対して結果を予測する
  - ✓ 意思決定の道具として利用できる
- 繰り返し結果を出すもの
- パターン化できるものを認識する
  - ✓ 画像認識、テキスト解析、音声認識
- 決められたルールに基づいて最適な結果を出す

機械学習で出来ないこと

- 目的を持たないモノに対して結果を出す
  - ✓ あくまでも目的は人間が設定する
- 入力値をデータとして与えられないものを認識する
  - ✓ 人の感情など
- 学習したことのない事象に対応する
- 課題を見つけて解決策を提案する

機械学習にできないことの例

- 馬車を認識できないテスラ

https://youtu.be/tpOg87AQvbo

顧客満足度分析演習

概要

- Airline Passenger Satisfaction

https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction

✓ 飛行機の乗客の顧客満足度調査結果

✓ 総合満足度を向上したい

データ項目

- Gender ：乗客の性別（ Female, Male ）
- Customer Type: 顧客種別 (Loyal customer:忠実な顧客, disloyal customer:不忠な顧客)
- Age: The actual age of the passengers
- Type of Travel: 旅行対応 (Personal Travel:個人旅行, Business Travel:出張)
- Class: 飛行機のクラス (Business, Eco, Eco Plus)
- Flight distance: 飛行距離
- Inflight wifi service:機内wifiサービスの満足度(0:該当なし Applicable;1-5)
- Departure/Arrival time convenient:出発/到着時間の満足度
- Ease of Online booking: オンライン予約の満足度
- Gate location: ゲート位置の満足度
- Food and drink: 飲食物の満足度
- Online boarding: オンライン搭乗の満足度
- Seat comfort: 座席の快適さの満足度
- Inflight entertainment: 機内エンターテインメントの満足度
- On-board service: 船内サービスの満足度
- Leg room service: レッグルームサービスの満足度
- Baggage handling: 手荷物取り扱いの満足度
- Check-in service: チェックインサービスの満足度
- Inflight service: 機内サービスの満足度
- Cleanliness: 清潔度の満足度
- Departure Delay in Minutes: 出発遅延（分）
- Arrival Delay in Minutes: 到着遅延（分）
- Satisfaction: 航空会社の満足度 (satisfied ：満足、 neutral or dissatisfaction：中立か不満)

満足度の調査指標

- NPS（ネットプロモータースコア）

仮説の立案

- 各項目と顧客満足度にどのような関係があるか考えてみましょう

データの読み込み

```
DAT_PATH = "../../master/python_data_analysis_train_data/airline-passenger-satisfaction/"
df_train = pd.read_csv(DAT_PATH + "train.csv")
# カラム名の半角スペースを置換
df_train.columns = df_train.columns.str.replace(' ', '_')
df_train.head()
```

	Unnamed:_0	id	Gender	Customer_Type	Age	Type_of_Travel	Class	Flight_Distance	Inflight_wifi_service	Departure/Arrival_time_convenient	...	I
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	

データの要約統計量を表示

```
df_train.describe()
```

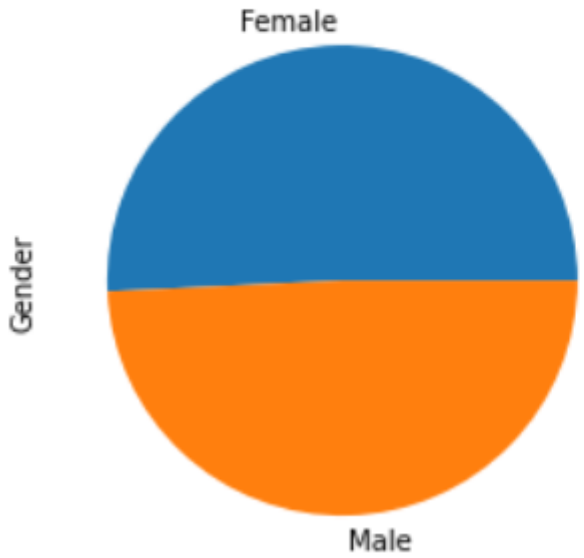
	Unnamed:_0	id	Age	Flight_Distance	Inflight_wifi_service	Departure/Arrival_time_convenient	Ease_of_Online_booking	Gate_loca
count	103904.000000	103904.000000	103904.000000	103904.000000	103904.000000	103904.000000	103904.000000	103904.00(
mean	51951.500000	64924.210502	39.379706	1189.448375	2.729683	3.060296	2.756901	2.97(
std	29994.645522	37463.812252	15.114964	997.147281	1.327829	1.525075	1.398929	1.27
min	0.000000	1.000000	7.000000	31.000000	0.000000	0.000000	0.000000	0.00(
25%	25975.750000	32533.750000	27.000000	414.000000	2.000000	2.000000	2.000000	2.00(
50%	51951.500000	64856.500000	40.000000	843.000000	3.000000	3.000000	3.000000	3.00(
75%	77927.250000	97368.250000	51.000000	1743.000000	4.000000	4.000000	4.000000	4.00(
max	103903.000000	129880.000000	85.000000	4983.000000	5.000000	5.000000	5.000000	5.00(

男女比の把握

- 大きく偏りがないことを確認

```
df_train.groupby('Gender')['Gender'].count().plot.pie()
```

```
<AxesSubplot:ylabel='Gender'>
```

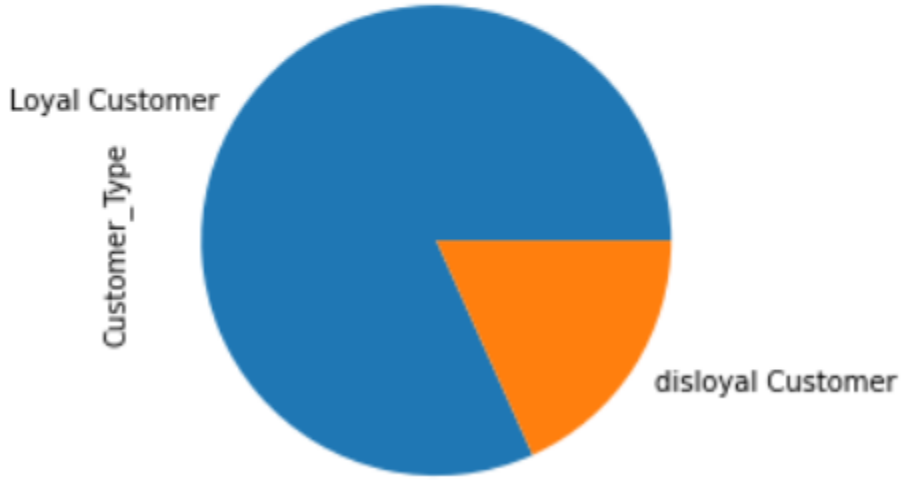


顧客種別の把握

- LoyalCustomerが8割近い

```
: df_train.groupby('Customer_Type')['Customer_Type'].count().plot.pie()
```

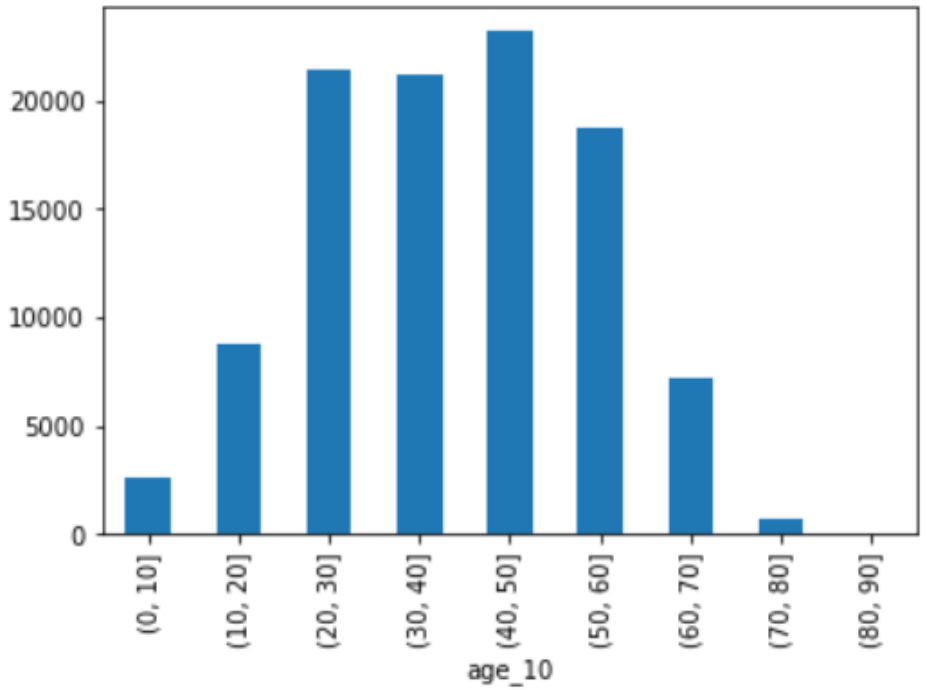
```
: <AxesSubplot:ylabel='Customer_Type'>
```



年齢の把握

```
# 年齢は連続値のため、わかりやすいように10歳ごとに分割
df_train['age_10'] = pd.cut(df_train['Age'], [0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
df_train.groupby('age_10')['age_10'].count().plot.bar()
```

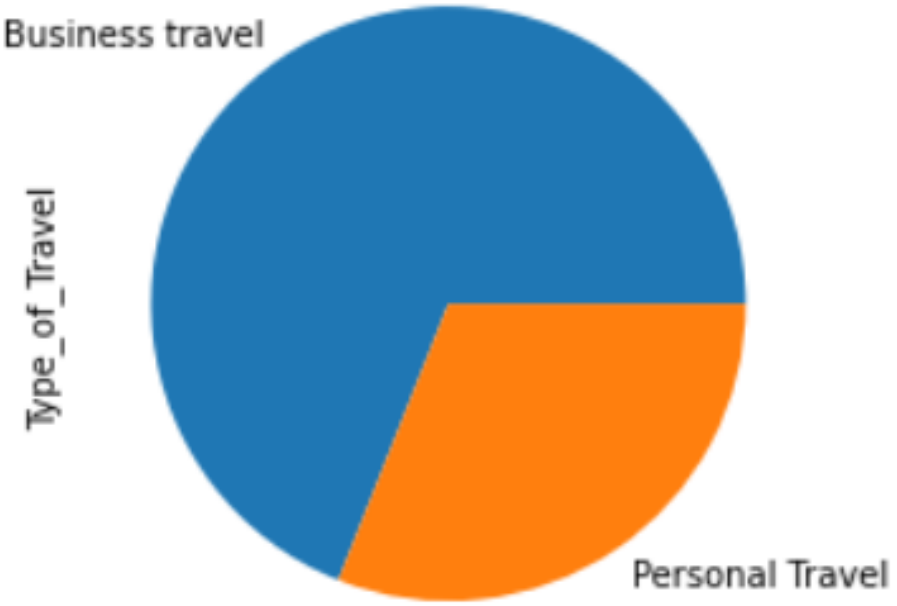
```
<AxesSubplot:xlabel='age_10'>
```



旅行種別の把握

```
df_train.groupby('Type_of_Travel')['Type_of_Travel'].count().plot.pie()
```

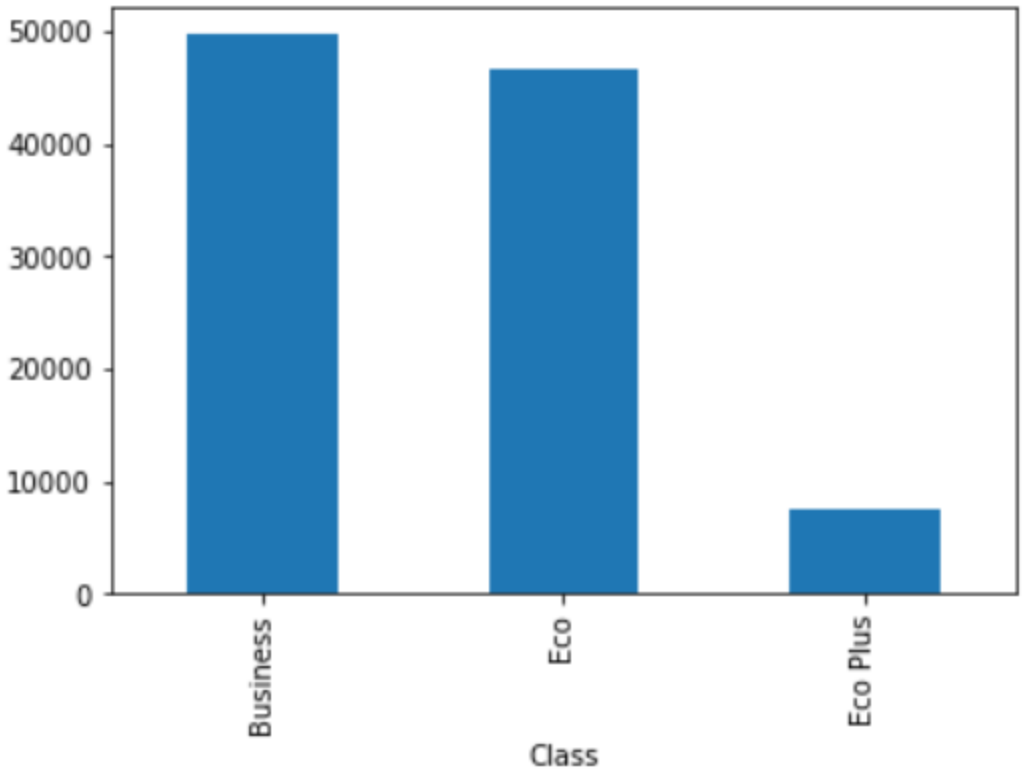
```
<AxesSubplot:ylabel='Type_of_Travel'>
```



クラスの把握

```
df_train.groupby('Class')['Class'].count().plot.bar()
```

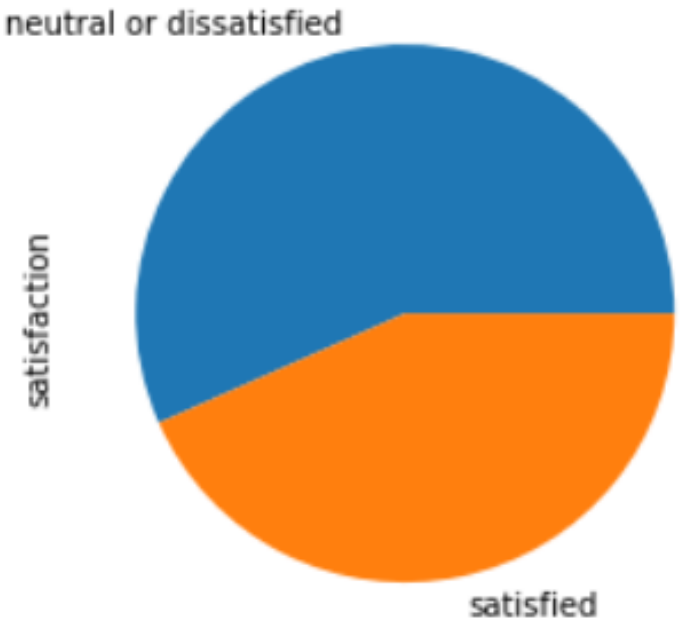
```
<AxesSubplot:xlabel='Class'>
```



満足度の把握

```
df_train.groupby('satisfaction')['satisfaction'].count().plot.pie()
```

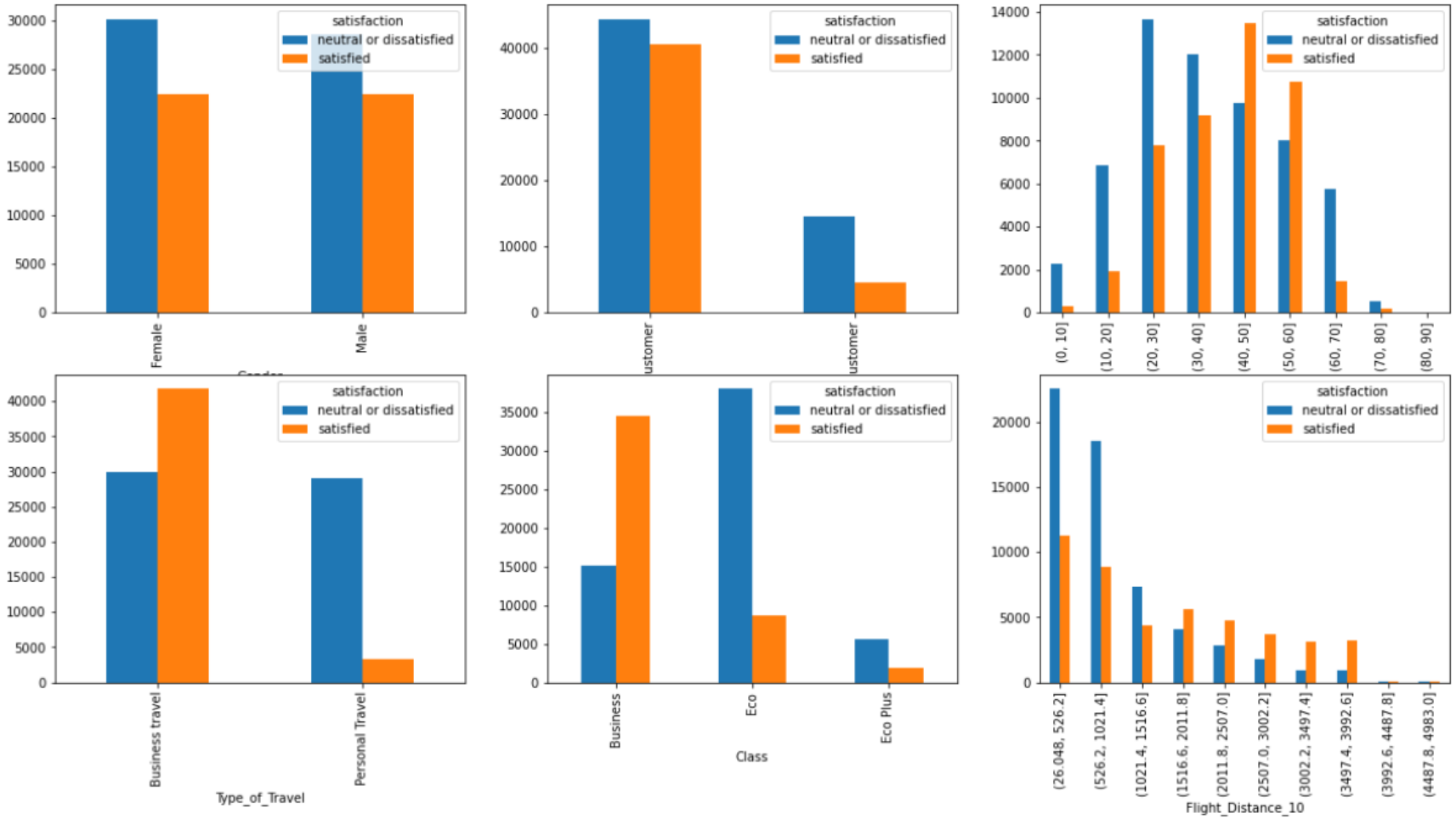
```
<AxesSubplot:ylabel='satisfaction'>
```



属性ごと満足度クロス集計

```
fig = plt.figure(figsize=(20, 10))
ax1 = fig.add_subplot(2, 3, 1)
ax2 = fig.add_subplot(2, 3, 2)
ax3 = fig.add_subplot(2, 3, 3)
ax4 = fig.add_subplot(2, 3, 4)
ax5 = fig.add_subplot(2, 3, 5)
ax6 = fig.add_subplot(2, 3, 6)
# 性別ごと満足度
pd.crosstab(df_train['Gender'], df_train['satisfaction']).plot.bar(ax = ax1)
# 顧客種別ごと満足度
pd.crosstab(df_train['Customer_Type'], df_train['satisfaction']).plot.bar(ax = ax2)
# 年代ごと満足度
pd.crosstab(df_train['age_10'], df_train['satisfaction']).plot.bar(ax = ax3)
# 旅行種別ごと満足度
pd.crosstab(df_train['Type_of_Travel'], df_train['satisfaction']).plot.bar(ax = ax4)
# クラスごと満足度
pd.crosstab(df_train['Class'], df_train['satisfaction']).plot.bar(ax = ax5)
# 距離ごと満足度
df_train['Flight_Distance_10'] = pd.cut(df_train['Flight_Distance'], 10)
pd.crosstab(df_train['Flight_Distance_10'], df_train['satisfaction']).plot.bar(ax = ax6)
```

属性ごと満足度クロス集計



カテゴリカル変数を変換

- 性別・クラスなど分類を表す項目（カテゴリカル変数）

→ 文字列のままでは分析できないため変換（エンコード）する

1. Label encoding（ラベルエンコーディング）

✓ カテゴリカル変数を数値（スカラ値）に変換する

例：male→1、female→0

2. One hot Encoding（ワンホットエンコーディング）

✓ カテゴリカル変数の値ごとに列が作られて、0 or 1の値になる

✓ すべてが0になる列を除いたものがダミーエンコーディング

例

male	female
1	0
0	1

愛知県

カテゴリカル変数を変換

```
df_gender = pd.DataFrame(df_train['Gender'])

# Label encoding（ラベルエンコーディング）
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_gender['Gender_le'] = le.fit_transform(df_gender['Gender'])
df_gender.head()
```

	Gender	Gender_le
0	Male	1
1	Male	1
2	Female	0
3	Female	0
4	Male	1

```
# One hot Encoding（ワンホットエンコーディング）
df_gender = pd.get_dummies(df_train['Gender'])
df_gender.head()
```

	Female	Male
0	0	1
1	0	1
2	1	0
3	1	0
4	0	1

項目間の相関を把握

満足度を表す項目だけ抜き出し

```
df_train.columns
```

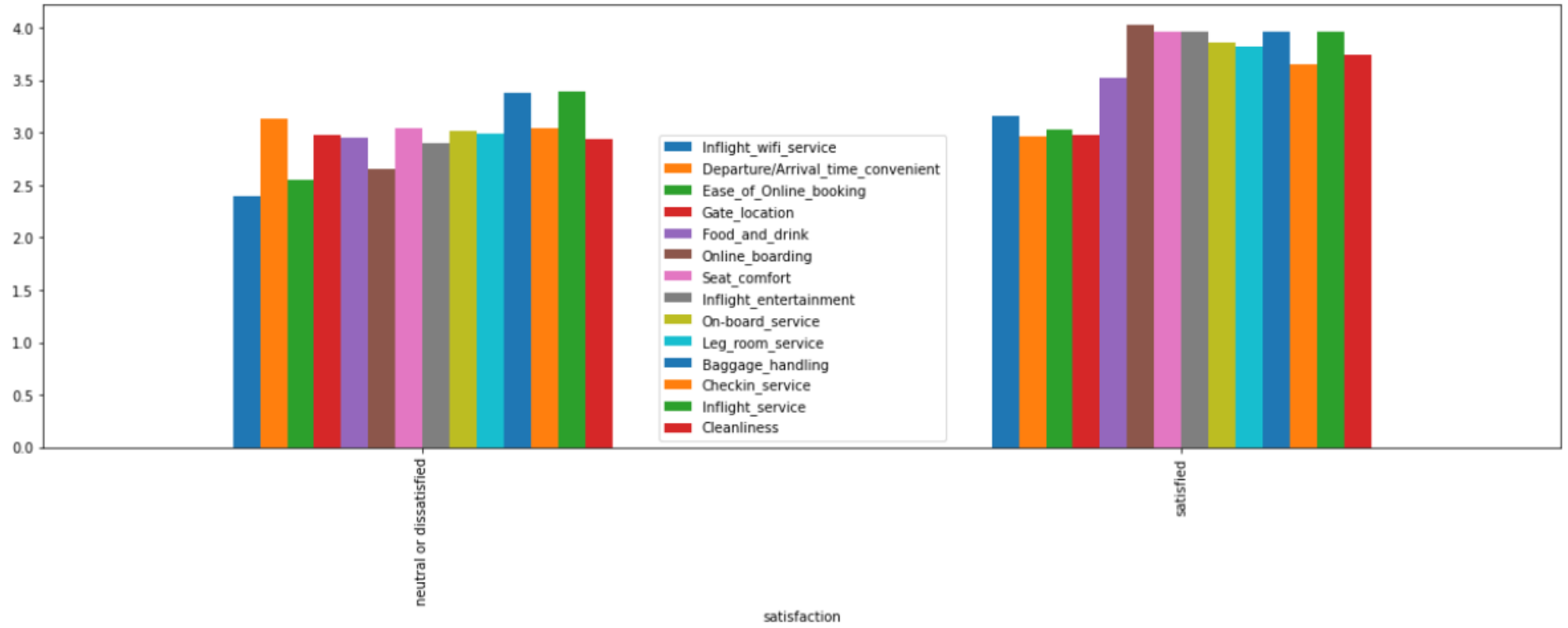
```
Index(['Unnamed:_0', 'id', 'Gender', 'Customer_Type', 'Age', 'Type_of_Travel',
       'Class', 'Flight_Distance', 'Inflight_wifi_service',
       'Departure/Arrival_time_convenient', 'Ease_of_Online_booking',
       'Gate_location', 'Food_and_drink', 'Online_boarding', 'Seat_comfort',
       'Inflight_entertainment', 'On-board_service', 'Leg_room_service',
       'Baggage_handling', 'Checkin_service', 'Inflight_service',
       'Cleanliness', 'Departure_Delay_in_Minutes', 'Arrival_Delay_in_Minutes',
       'satisfaction', 'age_10', 'Flight_Distance_10'],
      dtype='object')
```

```
df_sub = df_train[['Inflight_wifi_service',
                   'Departure/Arrival_time_convenient',
                   'Ease_of_Online_booking',
                   'Gate_location',
                   'Food_and_drink',
                   'Online_boarding',
                   'Seat_comfort',
                   'Inflight_entertainment',
                   'On-board_service',
                   'Leg_room_service',
                   'Baggage_handling',
                   'Checkin_service',
                   'Inflight_service',
                   'Cleanliness',
                   'satisfaction']]
```

項目間の相関を把握

```
: df_sub.groupby('satisfaction').mean().plot.bar(figsize=(20, 6))
```

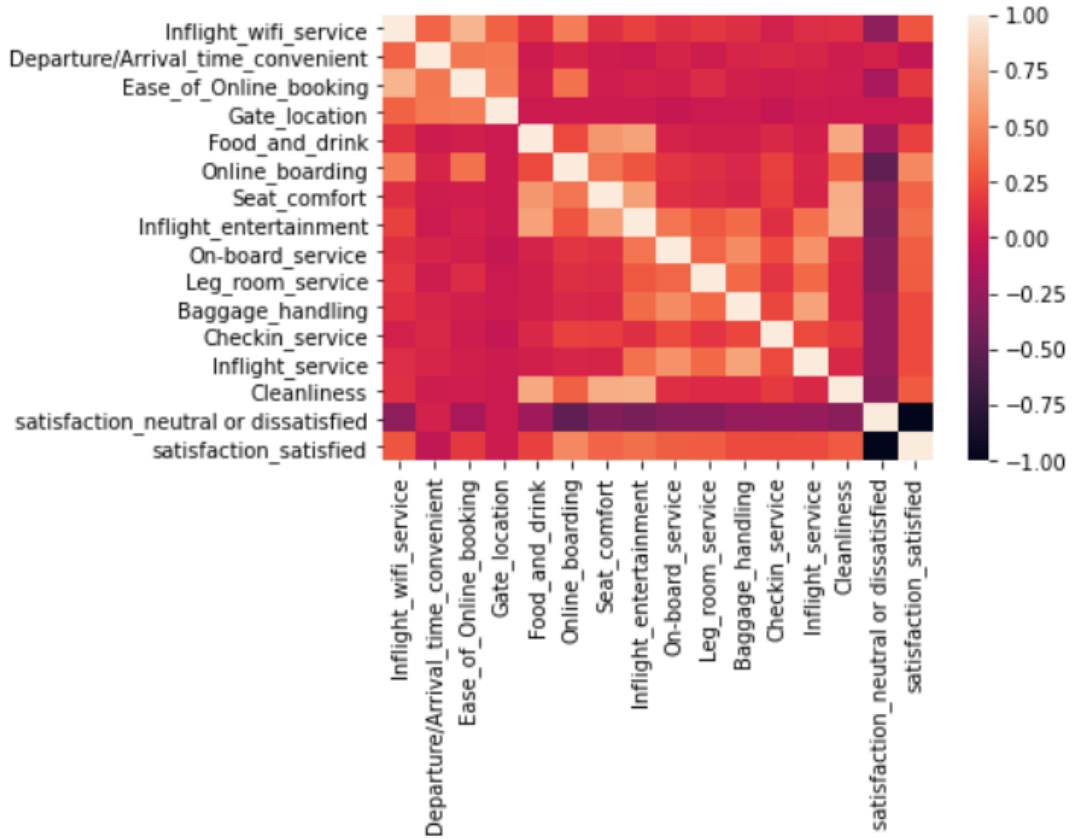
```
: <AxesSubplot:xlabel='satisfaction'>
```



項目間の相関を把握

```
df_sub = pd.get_dummies(df_sub)
sns.heatmap(df_sub.corr())
```

<AxesSubplot:>



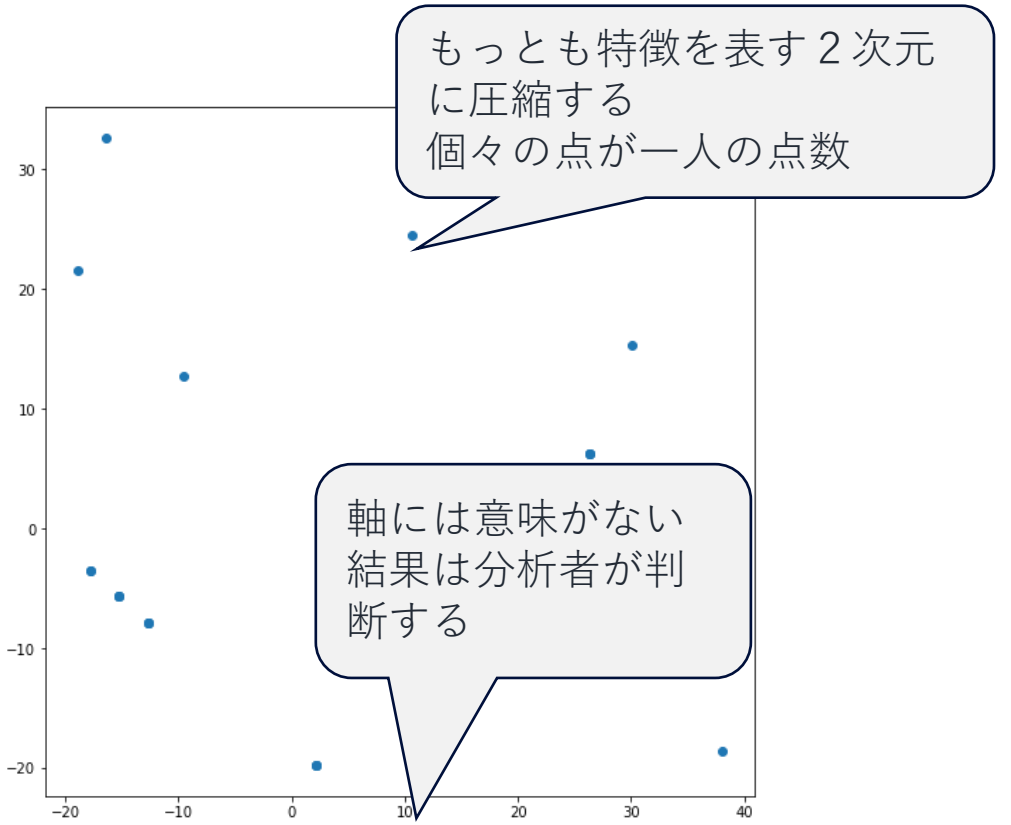
愛知県

主成分分析（PCA）

- もっとも特徴をよく表す（バラツキが大きくなる）軸を生成して、次元を圧縮する

No	Language	English	Math	Science	Society
1	40	40	80	60	50
2	45	50	70	60	46
3	70	60	50	60	80
4	80	90	70	80	90
5	55	60	70	60	50

5科目のテストの点数
（5次元）

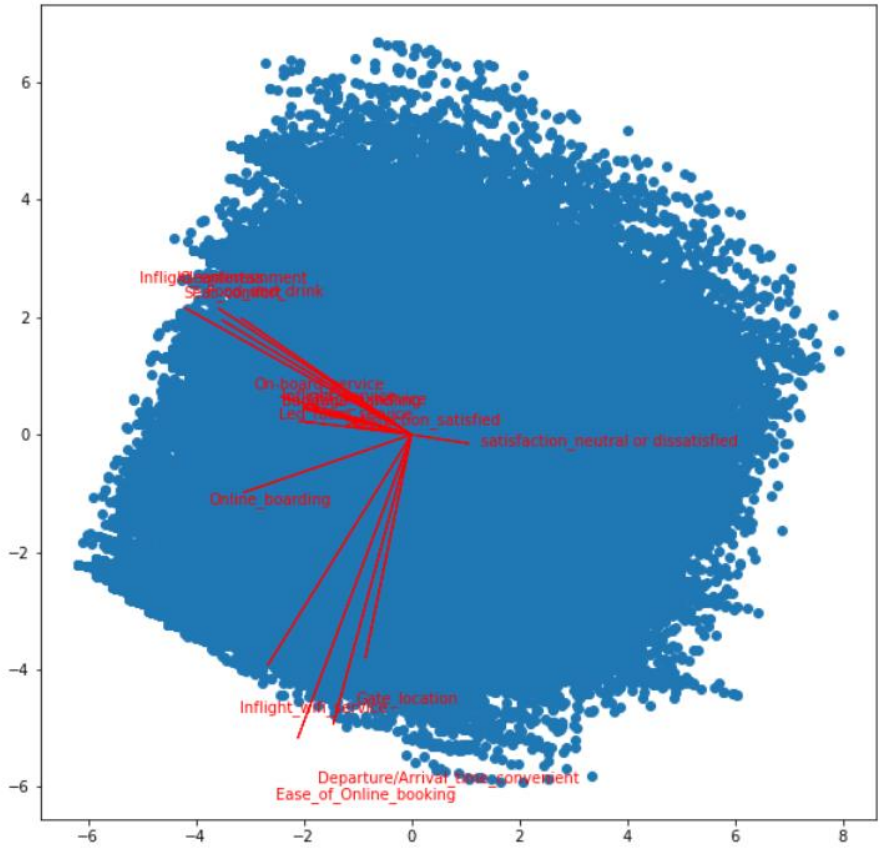


主成分分析

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(df_sub)

# 結果の表示
pca_point = pca.transform(df_sub)
x_data = pca_point[:, 0]
y_data = pca_point[:, 1]
fa0 = pca.components_[0]
fa1 = pca.components_[1]
plt.figure(figsize=(10,10))
plt.scatter(x_data, y_data)
for i in range(fa0.shape[0]):
    plt.arrow(0, 0, fa0[i]*10, fa1[i]*10, color='r')
    plt.text(fa0[i]*12, fa1[i]*12, df_sub.columns.values[i], color='r')
```

主成分分析

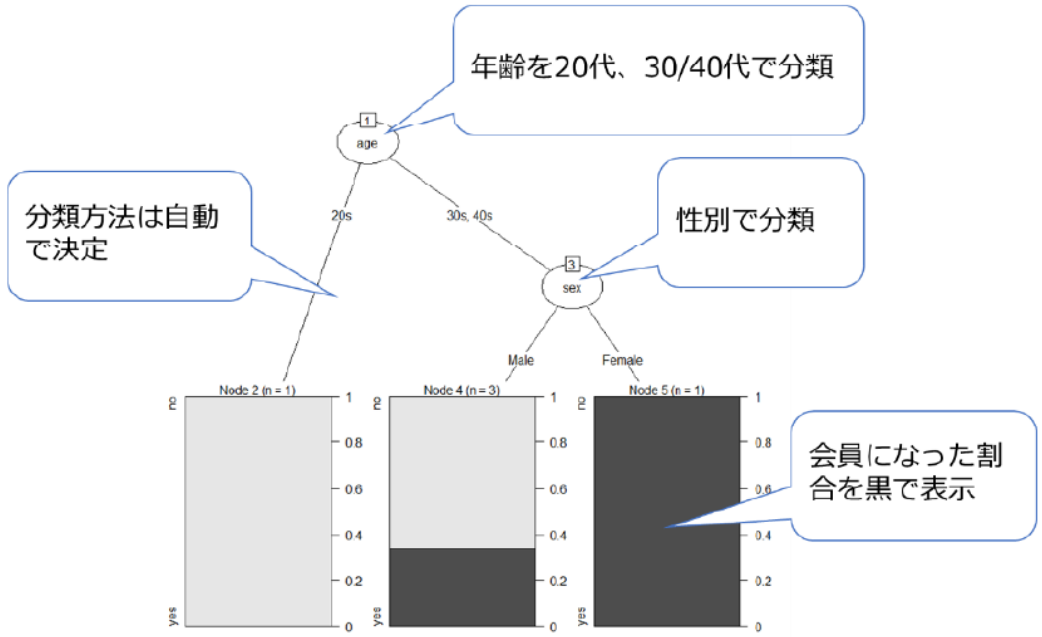
- 上側項目と下側項目でわかれる



決定木

- もっとも偏りが大きくなるように自動で分割する
  - ✓ 目的変数が分類値：分類木
  - ✓ 目的変数が連続値：回帰木

■どのような属性の人が会員になりやすいか



決定木の特徴

- 結果を理解しやすい
  - ✓ 木構造に可視化することでわかりやすい
- 特徴量を選別する必要がない
  - ✓ もっとも影響の大きい特徴量を自動的に採用する
- 過剰適合しやすく汎化性能が低い
  - ✓ 木の深さや葉の数で、複雑さを制限する（枝刈り）
- 決定木ベースのランダムフォレストや勾配ブースティングが非常に強力（アンサンブル法）

決定木

- カテゴリカル変数を変換（処理前）

```
from dtreeviz.trees import dtreeviz
from sklearn.tree import DecisionTreeClassifier

# 説明変数
df_X = df_train.drop(['Unnamed: 0', 'id', 'satisfaction', 'age_10', 'Flight_Distance_10'], axis=1)
# 目的変数
df_y = df_train['satisfaction']
df_X.head()
```

	Gender	Customer_Type	Age	Type_of_Travel	Class	Flight_Distance	Inflight_wifi_service	Departure/Arrival_time_convenient	Ease_of_Online_booking	C
0	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	
1	Male	disloyal Customer	25	Business travel	Business	235	3	2	3	
2	Female	Loyal Customer	26	Business travel	Business	1142	2	2	2	
3	Female	Loyal Customer	25	Business travel	Business	562	2	5	5	
4	Male	Loyal Customer	61	Business travel	Business	214	3	3	3	

決定木

- カテゴリカル変数を変換（処理後）

```
# カテゴリカル変数を変換(ラベルエンコーディング)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df_X['Gender'] = le.fit_transform(df_X['Gender'].values)
df_X['Customer_Type'] = le.fit_transform(df_X['Customer_Type'].values)
df_X['Type_of_Travel'] = le.fit_transform(df_X['Type_of_Travel'].values)
df_X['Class'] = le.fit_transform(df_X['Class'].values)
df_y = le.fit_transform(df_y.values)
df_X.head()
```

	Gender	Customer_Type	Age	Type_of_Travel	Class	Flight_Distance	Inflight_wifi_service	Departure/Arrival_time_convenient	Ease_of_Online_booking	Gat
0	1	0	13	1	2	460	3	4	3	
1	1	1	25	0	0	235	3	2	3	
2	0	0	26	0	0	1142	2	2	2	
3	0	0	25	0	0	562	2	5	5	
4	1	0	61	0	0	214	3	3	3	

決定木

- 欠損値を処理

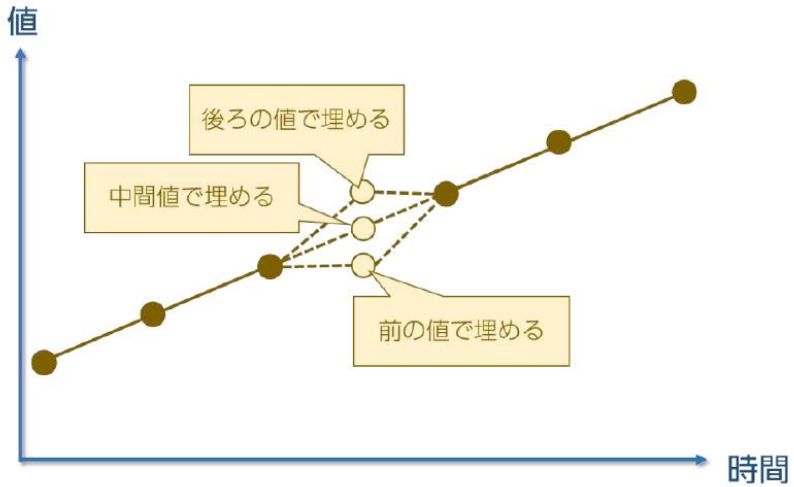
```
# 欠損値があるかチェック
print(df_X.isnull().any())
# 欠損値を0で埋める
df_X['Arrival_Delay_in_Minutes'] = df_X['Arrival_Delay_in_Minutes'].fillna(0)
```

```
Gender                             False
Customer_Type                      False
Age                                False
Type_of_Travel                     False
Class                              False
Flight_Distance                    False
Inflight_wifi_service              False
Departure/Arrival_time_convenient  False
Ease_of_Online_booking             False
Gate_location                      False
Food_and_drink                     False
Online_boarding                    False
Seat_comfort                       False
Inflight_entertainment             False
On-board_service                   False
Leg_room_service                   False
Baggage_handling                   False
Checkin_service                    False
Inflight_service                   False
Cleanliness                        False
Departure_Delay_in_Minutes         False
Arrival_Delay_in_Minutes            True
dtype: bool
```

愛知県

欠損値の処理方法

- 欠損値のある行を削除する
  - ✓ すべての列の値が揃わないと正しくない場合
- 欠損値を埋める
  - ✓ 中央値、平均値、最頻値で埋める
  - ✓ 前の値、中間値、後ろの値で埋める（時系列）

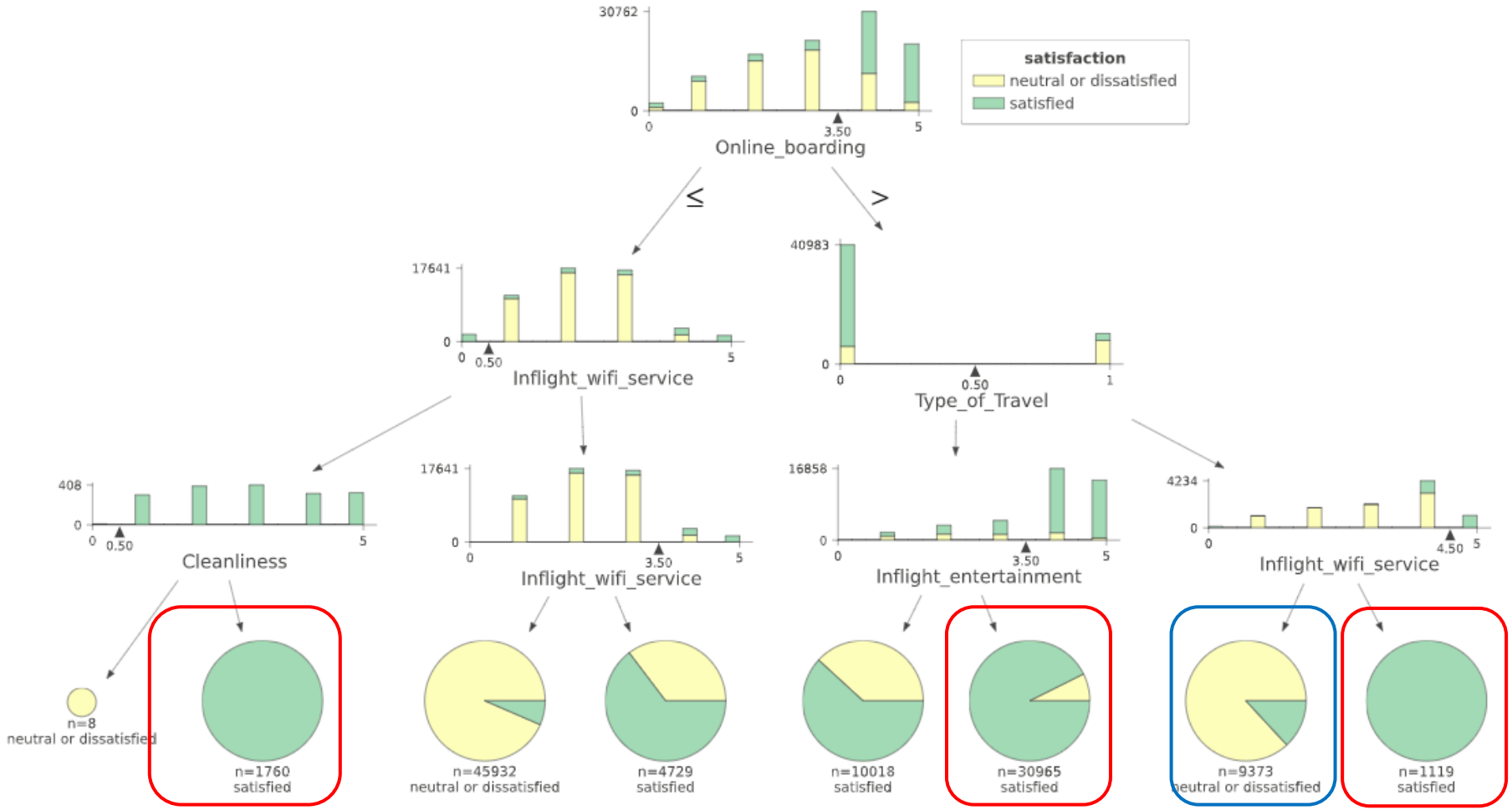


愛知県

決定木分析

```
# 決定木分析実行
regr = DecisionTreeClassifier(max_depth=3)
regr.fit(df_X, df_y)
viz = dtreeviz(regr,
               df_X,
               df_y,
               target_name='satisfaction',
               class_names=['neutral or dissatisfied', 'satisfied'],
               feature_names=df_X.columns)
display(viz)
```

分析結果


satisfaction
neutral or dissatisfied
satisfied
30762
0
0
3.50
5
Online_boarding
≤
>
17641
0
0
0.50
5
Inflight_wifi_service
40983
0
0
0.50
1
Type_of_Travel
408
0
0
0.50
5
Cleanliness
17641
0
0
3.50
5
Inflight_wifi_service
16858
0
0
3.50
5
Inflight_entertainment
4234
0
0
4.50
5
Inflight_wifi_service
n=8
neutral or dissatisfied
n=1760
satisfied
n=45932
neutral or dissatisfied
n=4729
satisfied
n=10018
satisfied
n=30965
satisfied
n=9373
neutral or dissatisfied
n=1119
satisfied

結果の考察

- 満足の割合が大きいのは以下のパターン

→満足度への影響が大きい

1. Online_bordingの満足度が4以下、Inflight_wifi_Serviceが該当なし、 Inflight entertainmentの満足度が1以上（あればよい）

2. Online_bordingの満足度が4以上、 Type of Travel が 0 （ビジネス）、 Inflight entertainmentの満足度が4以上
   → ビジネスは機内エンターテイメントがまあまあ重要

3. Online_bordingの満足度が4以上、 Type of Travel が1（個人旅行）、 Inflight_wifi_Serviceの満足度が 5 以上
   → Inflight_wifi_Serviceが4以下の場合の満足が13%
   → 個人旅行は機内WiFiが超重要

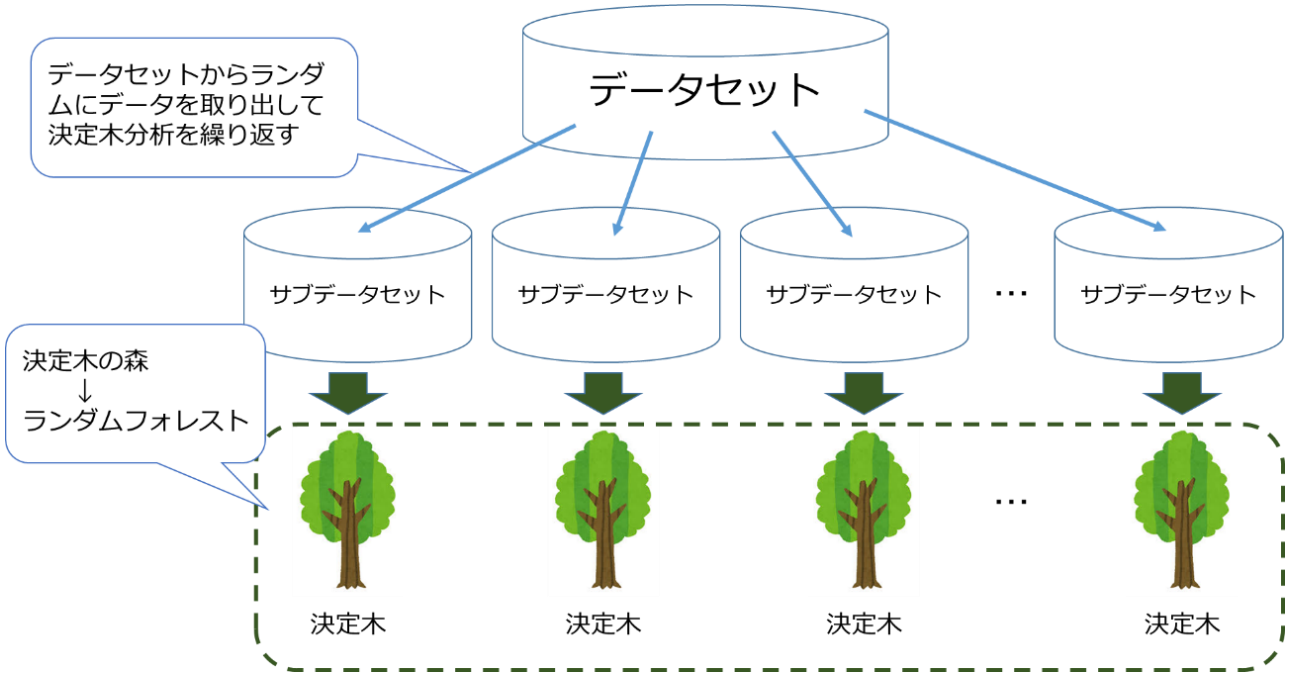
ランダムフォレスト

- 母集団からランダムにサンプリングして決定木を作る

ランダムの決定木の森 → 学習済みのデータセット

✓ 決定木の多数決で答えを出す

✓ 調整をしなくても大抵の場合でよい結果が出る



ランダムフォレスト

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# トレーニングデータとテストデータに分ける
(train_X, test_X, train_y, test_y) = train_test_split(df_X, df_y, random_state=0)

# ランダムフォレストモデルを生成
# n_estimators 決定木をいくつ生成するか（デフォルトは10）
clf = RandomForestClassifier(random_state=0, n_estimators=20)
clf = clf.fit(train_X, train_y)

# 結果検証(的中率)
print(clf.score(test_X, test_y))
print(clf.score(train_X, train_y))
```

```
0.9595780720665229
0.9991273996509599
```

需要予測分析演習

概要

- レンタル自転車のデータセットを使ってレンタル数の予測モデルを構築する
- 「Bike Sharing Demand」

  Washington, D.C.に展開するレンタル自転車サービスの、過去２年間(2011-2012)における１時間ごとのレンタルデータと気象条件等のデータ

  https://www.kaggle.com/c/bike-sharing-demand

データセットの概要

変数名	定義
datetime	時間
seazon	季節（1 = 春, 2 = 夏, 3 = 秋, 4 = 冬）
holiday	祝日かどうか
workingday	働く日かどうか（土日祝日以外）
weather	天気（1 = 晴れ、2 = 曇り、3 = 雨、4 = 大雨）
temp	気温
atemp	体感温度
humidity	湿度
windspeed	風速
casual	非会員のレンタル数
registered	会員のレンタル数
count	総レンタル数

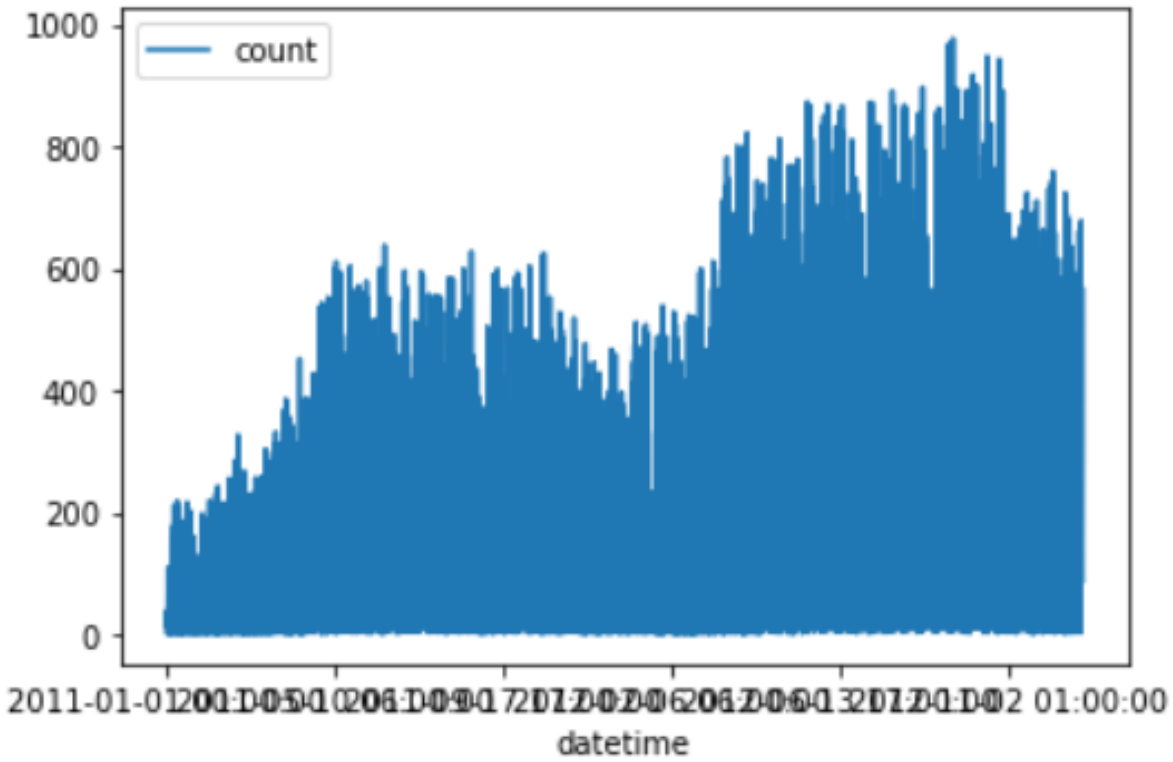
仮説の立案

- 各変数がレンタル数にどのように影響するのか
  - ✓ レンタル数に影響しないだろう、という仮説も重要

貸出数の推移

```
]: df.plot(x='datetime', y='count')
```

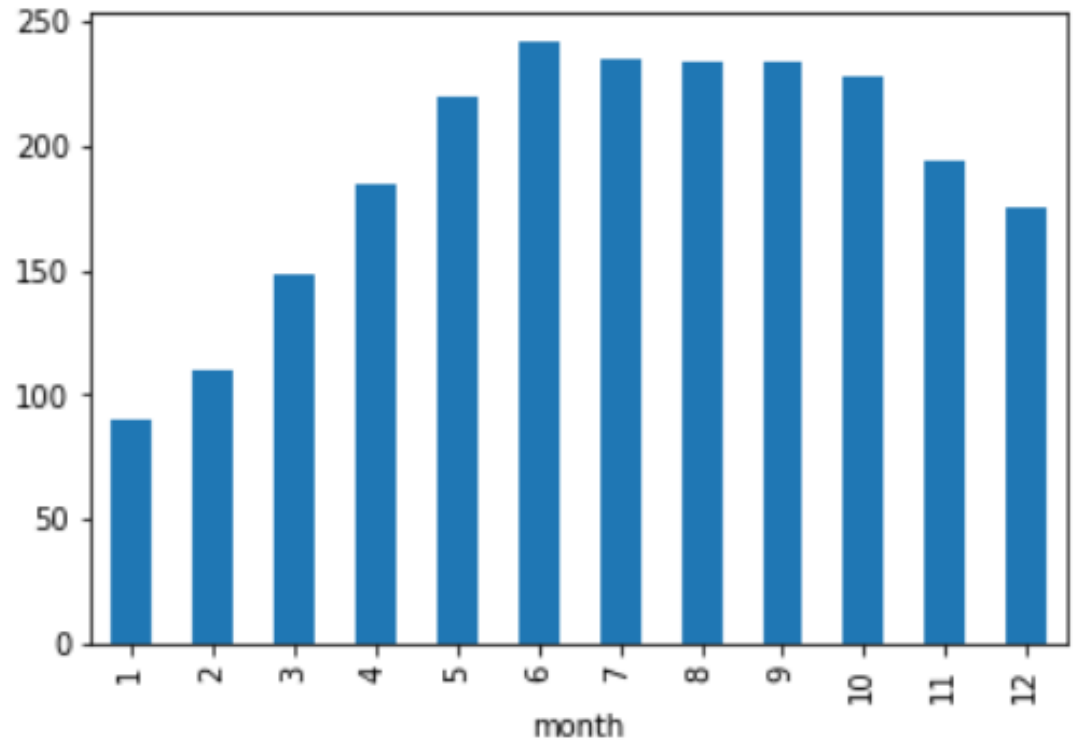
```
]: <AxesSubplot:xlabel='datetime'>
```



月ごとの貸し出し数（平均）

```
df.groupby('month')['count'].mean().plot.bar()
```

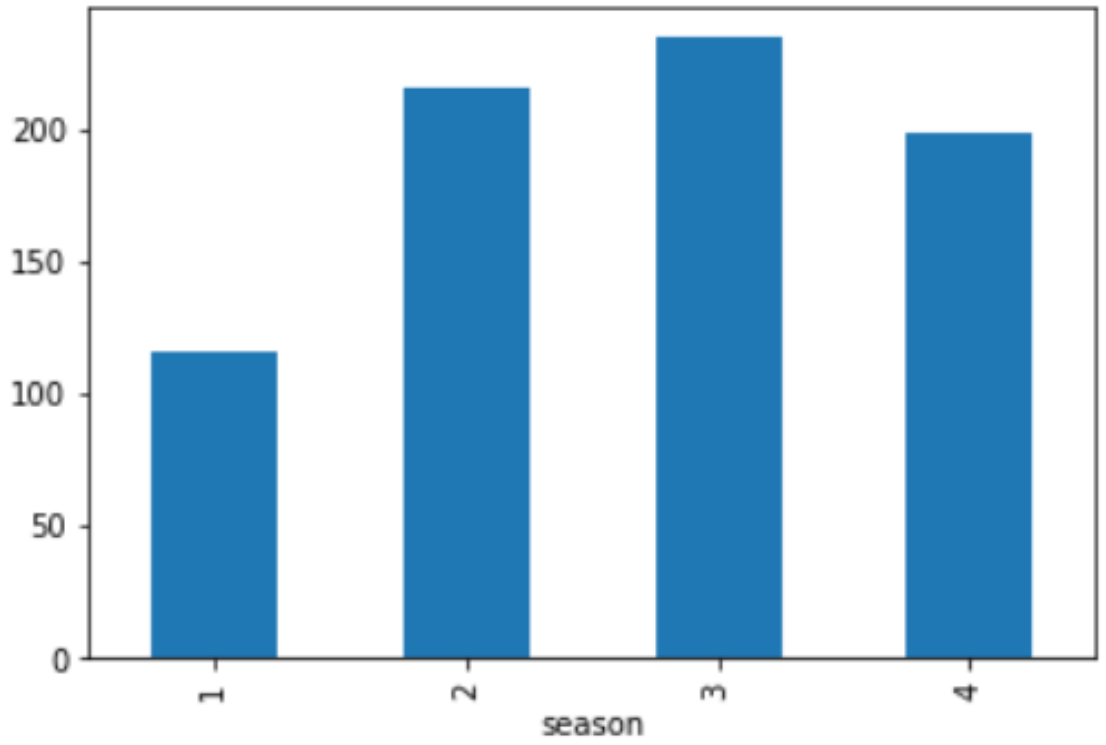
```
<AxesSubplot:xlabel='month'>
```



季節ごとの貸し出し数（平均）

```
df.groupby('season')['count'].mean().plot.bar()
```

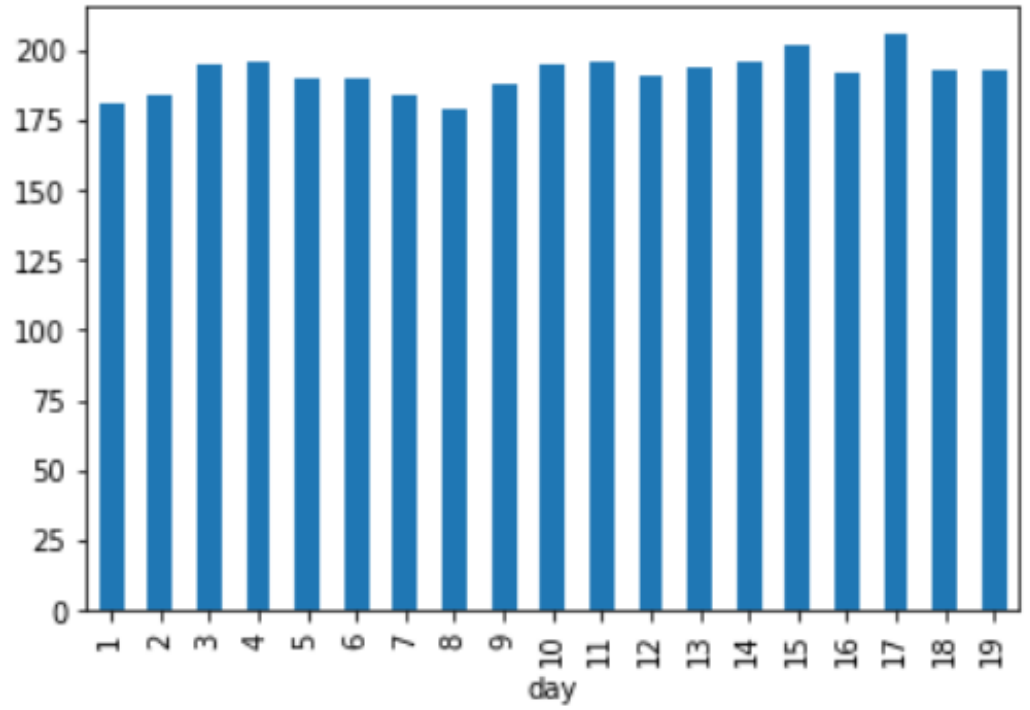
```
<AxesSubplot:xlabel='season'>
```



日ごとの貸し出し数（平均）

```
df.groupby('day')['count'].mean().plot.bar()
```

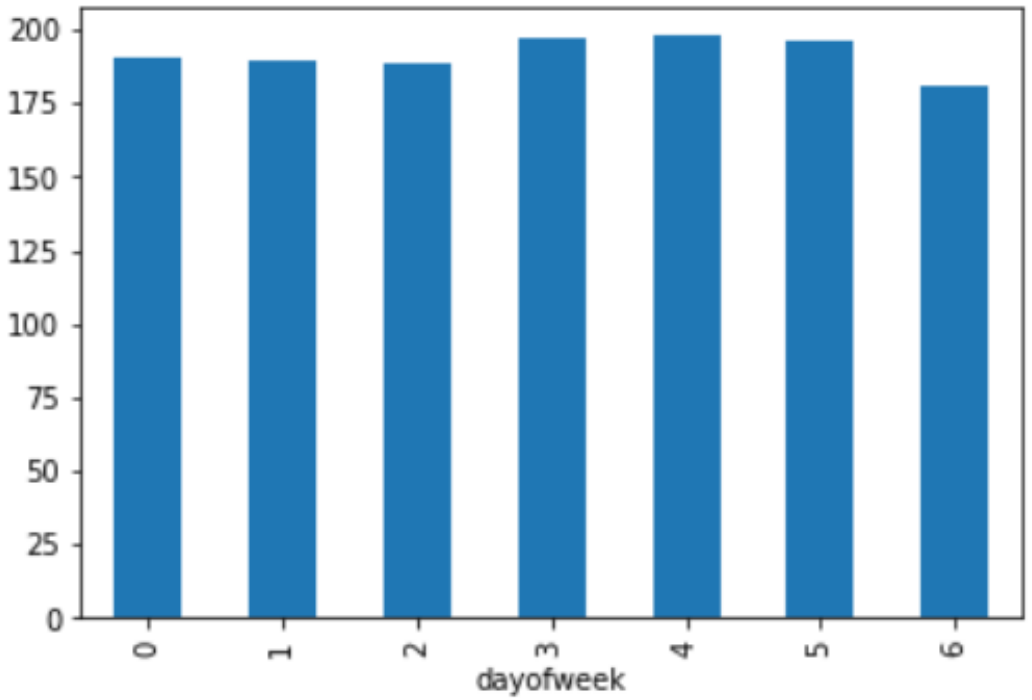
```
<AxesSubplot:xlabel='day'>
```



曜日ごとの貸し出し数（平均）

```
df.groupby('dayofweek')['count'].mean().plot.bar()
```

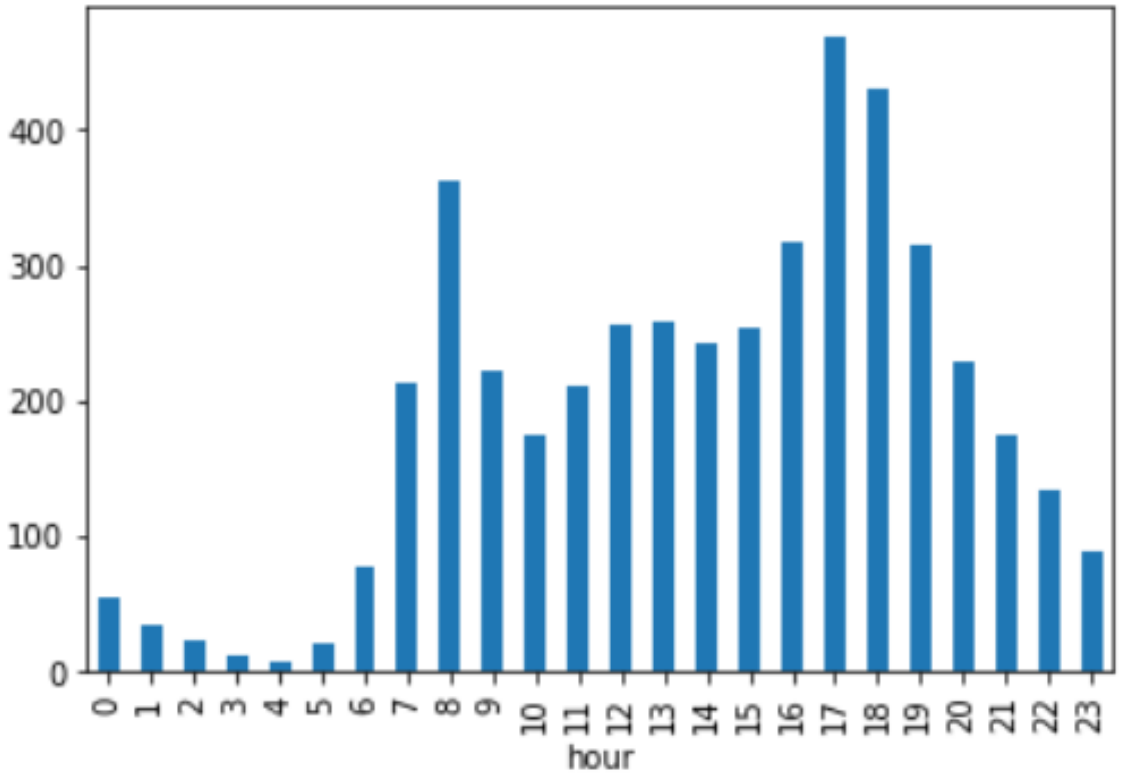
```
<AxesSubplot:xlabel='dayofweek'>
```



時間ごとの貸し出し数（平均）

```
df.groupby('hour')['count'].mean().plot.bar()
```

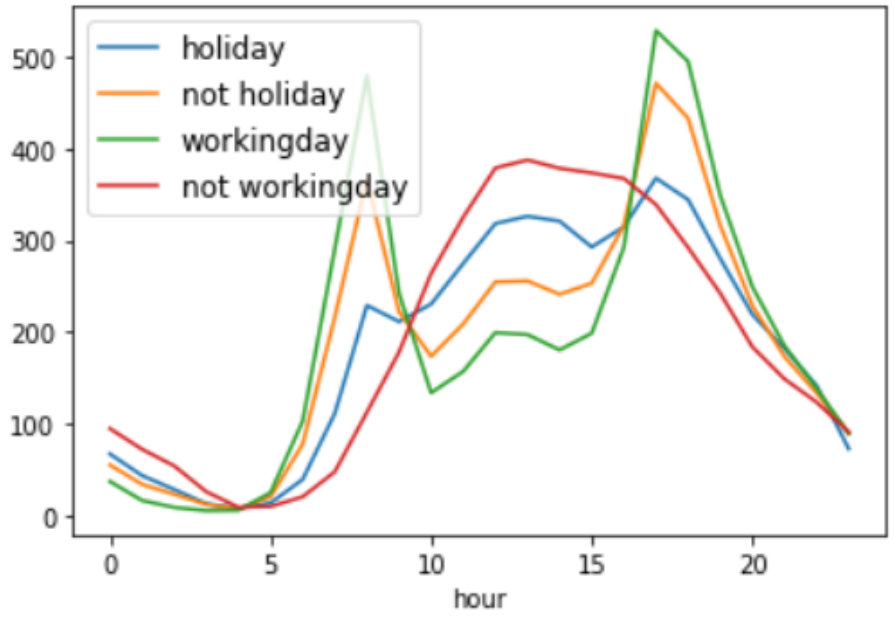
```
<AxesSubplot:xlabel='hour'>
```



holiday,workingday 時間ごと平均

```
df.query('holiday == 1').groupby('hour')['count'].mean().plot(label='holiday')
df.query('holiday == 0').groupby('hour')['count'].mean().plot(label='not holiday')
df.query('workingday == 1').groupby('hour')['count'].mean().plot(label='workingday')
df.query('workingday == 0').groupby('hour')['count'].mean().plot(label='not workingday')
plt.legend(fontsize=12)
```

```
<matplotlib.legend.Legend at 0x7fe7103f8a58>
```

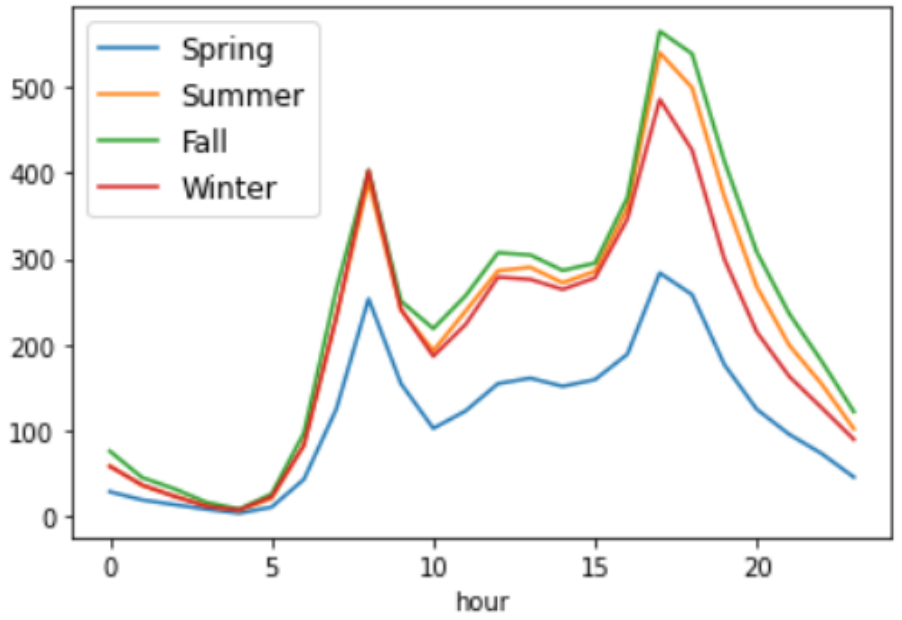


愛知県

季節による変化

```
df.query('season == 1').groupby('hour')['count'].mean().plot(label='Spring')
df.query('season == 2').groupby('hour')['count'].mean().plot(label='Summer')
df.query('season == 3').groupby('hour')['count'].mean().plot(label='Fall')
df.query('season == 4').groupby('hour')['count'].mean().plot(label='Winter')
plt.legend(fontsize=12)
```

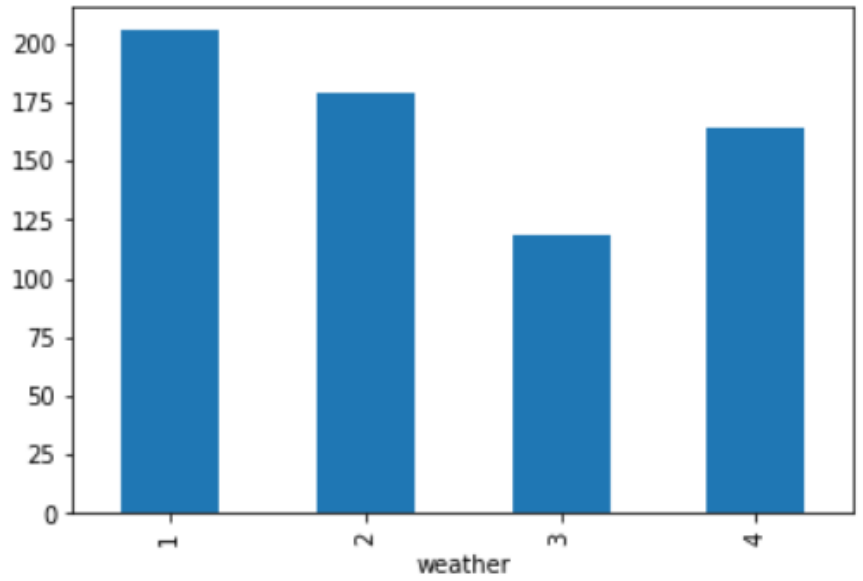
```
<matplotlib.legend.Legend at 0x7fe70e3189e8>
```



天候ごと貸出数（平均）

```
df.groupby('weather')['count'].mean().plot(kind='bar')
```

```
<AxesSubplot:xlabel='weather'>
```



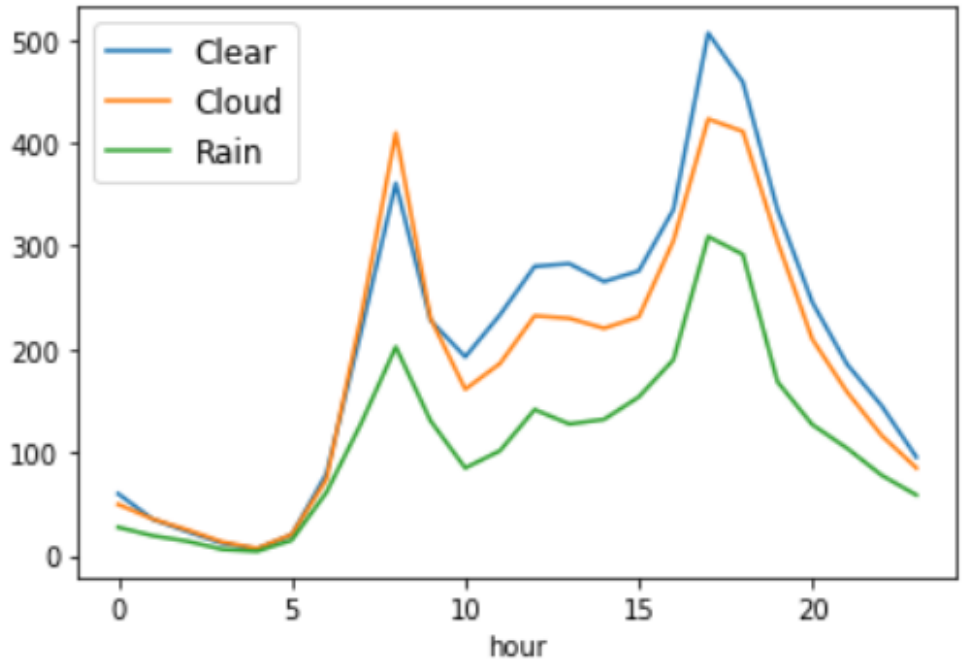

```
# 大雨のデータ件数
print(len(df.query('weather == 4')))
```

```
1
```

天気による変化

```
df.query('weather == 1').groupby('hour')['count'].mean().plot(label='Clear')
df.query('weather == 2').groupby('hour')['count'].mean().plot(label='Cloud')
df.query('weather == 3').groupby('hour')['count'].mean().plot(label='Rain')
plt.legend(fontsize=12)
```

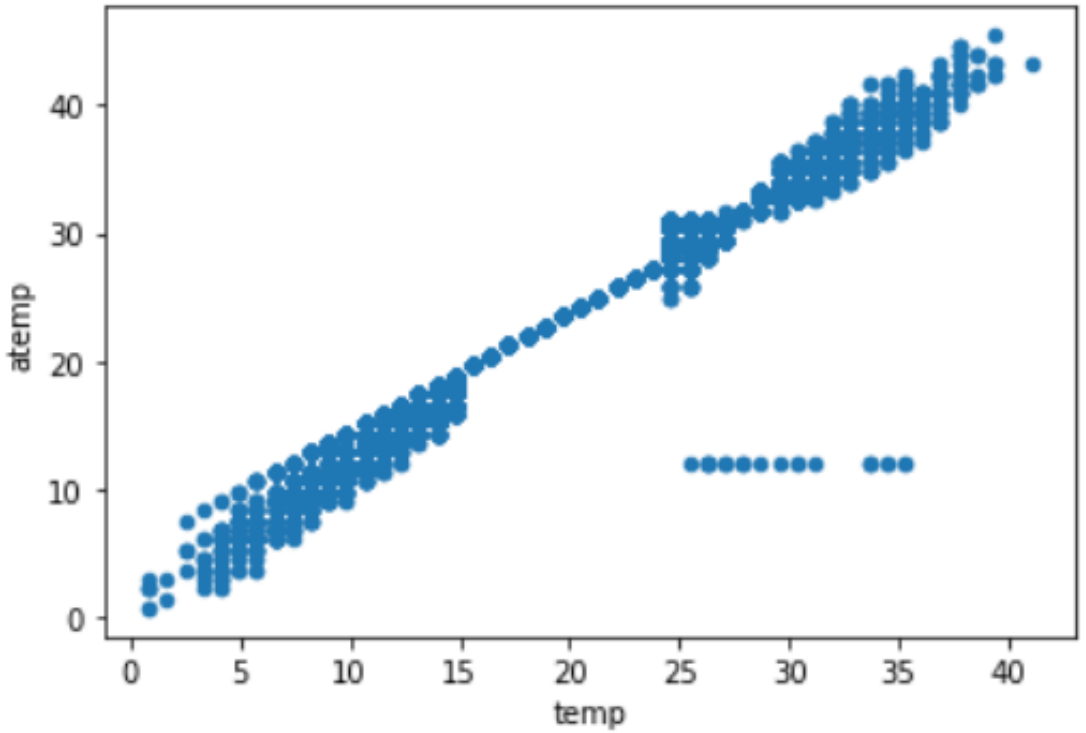
```
<matplotlib.legend.Legend at 0x7fe70e0f6e80>
```



気温と体感温度の関係

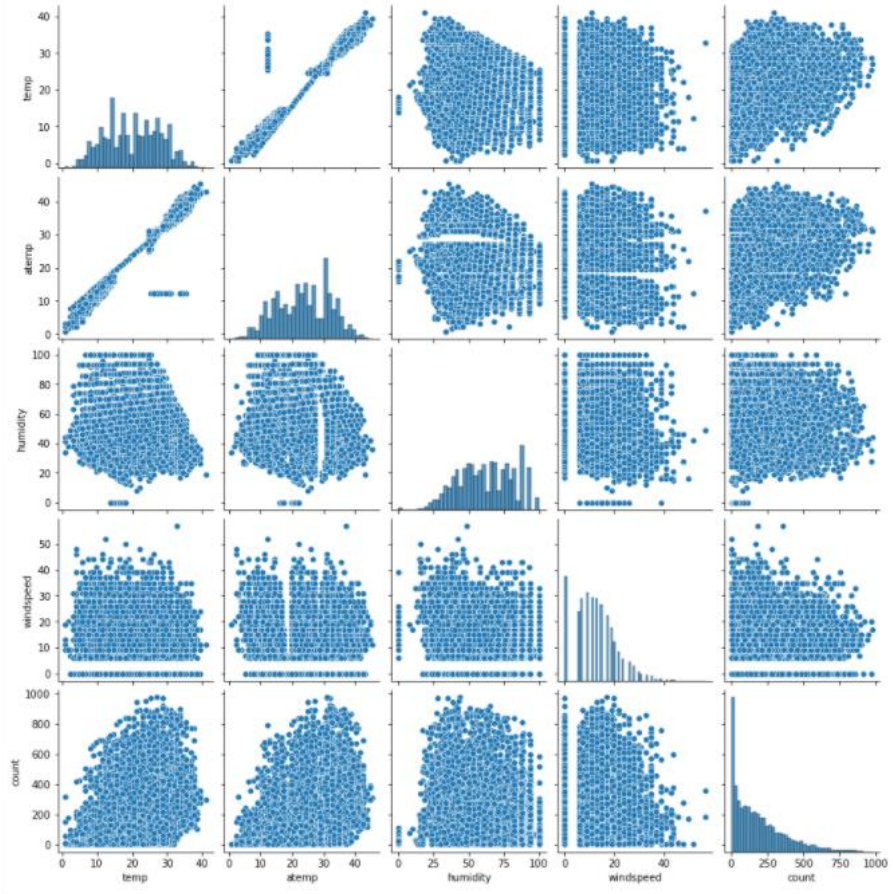
```
df.plot.scatter(x='temp', y='atemp')
```

```
<AxesSubplot:xlabel='temp', ylabel='atemp'>
```



温度、体感温度、湿度、風、貸出数

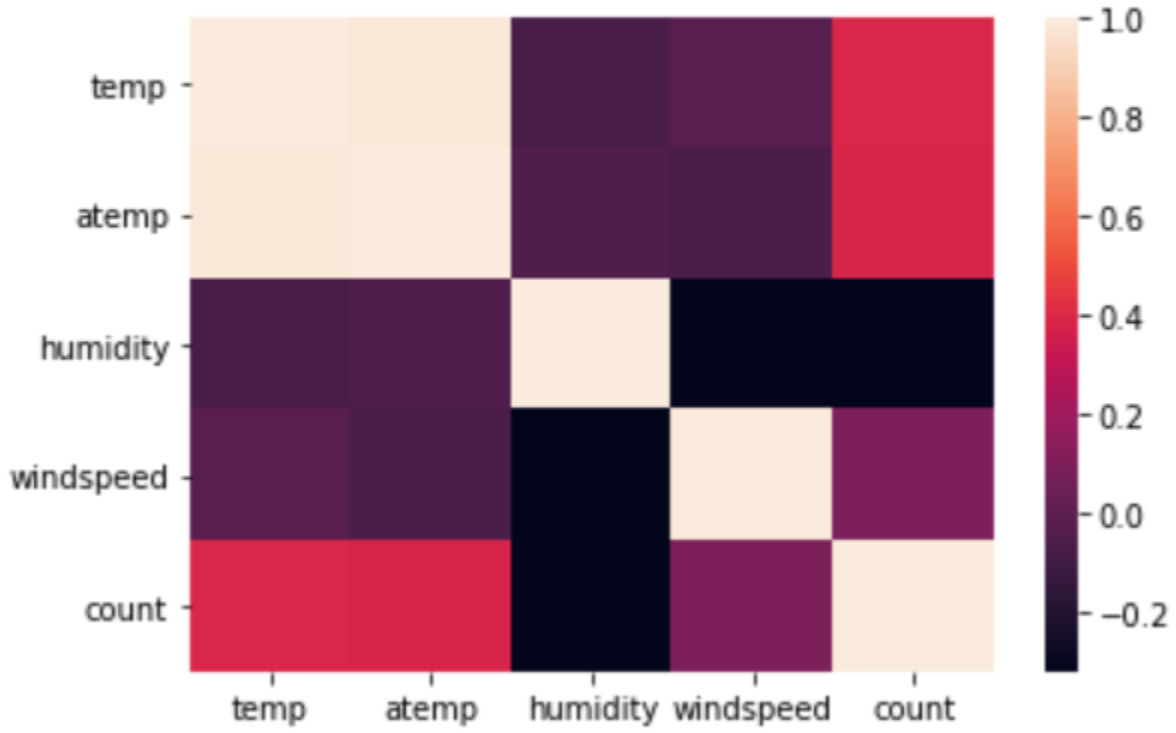
```
sns.pairplot(data=df[['temp', 'atemp', 'humidity', 'windspeed', 'count']])
```



相関係数をヒートマップで表示

```
: sns.heatmap(df[['temp', 'atemp', 'humidity', 'windspeed', 'count']].corr())
```

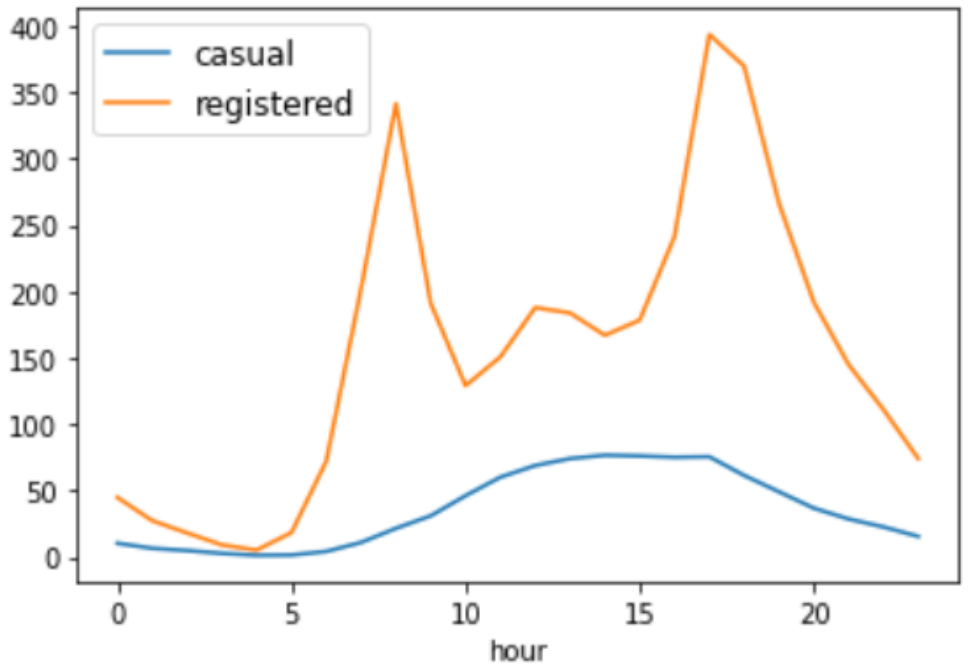
```
: <AxesSubplot:>
```



会員・非会員の違い

```
df.groupby('hour')['casual'].mean().plot(label='casual')
df.groupby('hour')['registered'].mean().plot(label='registered')
plt.legend(fontsize=12)
```

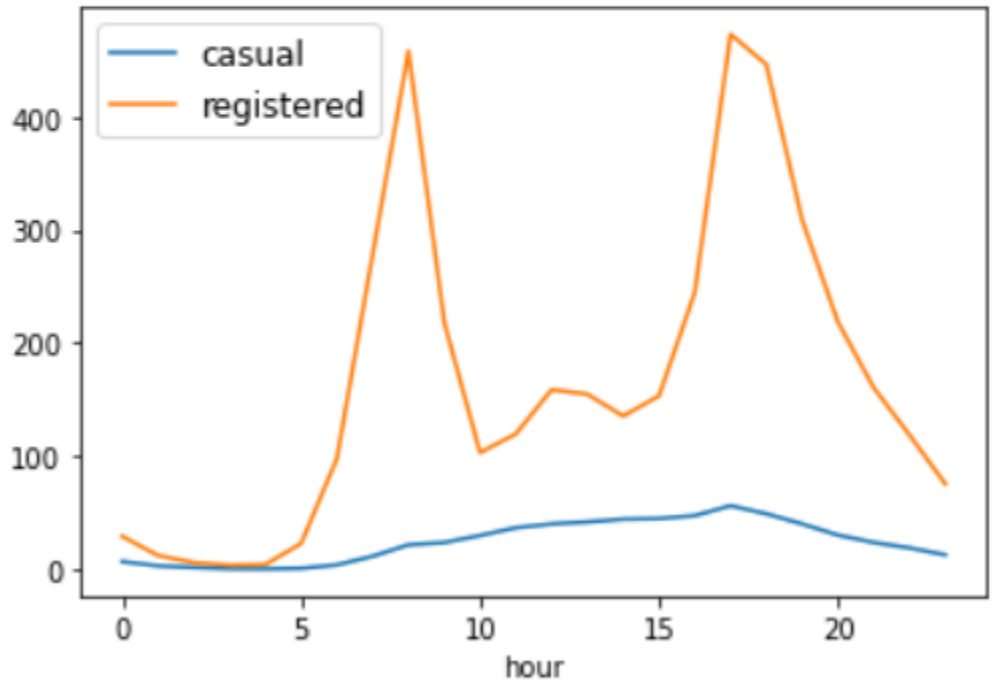
```
<matplotlib.legend.Legend at 0x7fe70c5994a8>
```



会員・非会員の違い

```
# 会員、非会員(働く日)
df.query('workingday == 1').groupby('hour')['casual'].mean().plot(label='casual')
df.query('workingday == 1').groupby('hour')['registered'].mean().plot(label='registered')
plt.legend(fontsize=12)
```

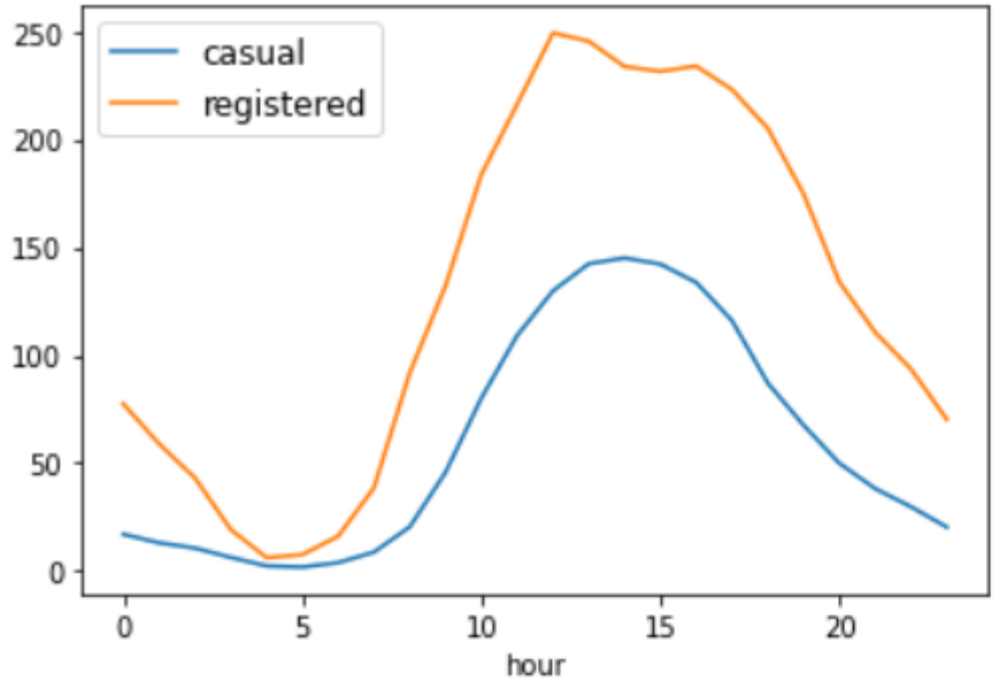
```
<matplotlib.legend.Legend at 0x7fe70d7c94e0>
```



会員・非会員の違い

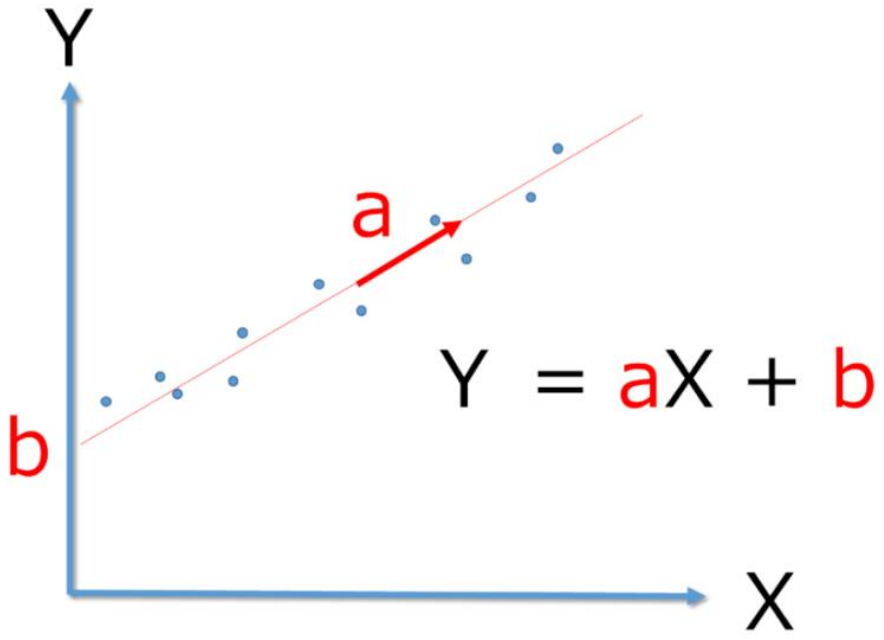
```
# 会員、非会員(土日祝日)
df.query('workingday == 0').groupby('hour')['casual'].mean().plot(label='casual')
df.query('workingday == 0').groupby('hour')['registered'].mean().plot(label='registered')
plt.legend(fontsize=12)
```

```
<matplotlib.legend.Legend at 0x7fe70e0f6908>
```



線形回帰

- 特徴量（説明変数）と目的変数の関係式を求める
  - ✓ 最小二乗法により誤差が最小となる関係式を求める
  - ✓ 関係式により特徴量から目的変数の値を予測できる
  - ✓ 特徴量が一つ：単回帰分析
  - ✓ 特徴量が複数：重回帰分析



予測モデル構築：重回帰分析

```
# 質的変数をカテゴリ変数に変換
for v in ["season","holiday","workingday","weather","month","year","hour"]:
    df[v] = df[v].astype("category")
df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count	datetimeobj	year	month	day	dayofwee
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16	2011-01-01 00:00:00	2011	1	1	
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40	2011-01-01 01:00:00	2011	1	1	
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32	2011-01-01 02:00:00	2011	1	1	
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13	2011-01-01 03:00:00	2011	1	1	
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1	2011-01-01 04:00:00	2011	1	1	

予測モデル構築：重回帰分析

```
import statsmodels.formula.api as sm
model = sm.ols(formula="count ~ year + month + hour + temp + weather + workingday", data=df)
result = model.fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	count	**R-squared:**	0.689
Model:	OLS	**Adj. R-squared:**	0.688
Method:	Least Squares	**F-statistic:**	602.0
Date:	Sat, 16 Oct 2021	**Prob (F-statistic):**	0.00
Time:	18:25:30	**Log-Likelihood:**	-65680.
No. Observations:	10886	**AIC:**	1.314e+05
Df Residuals:	10845	**BIC:**	1.317e+05
Df Model:	40		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-129.2953	6.456	-20.027	0.000	-141.950	-116.640
year[T.2012]	89.9983	1.963	45.836	0.000	86.150	93.847
month[T.2]	14.8029	4.830	3.065	0.002	5.336	24.270

予測モデル構築：ランダムフォレスト

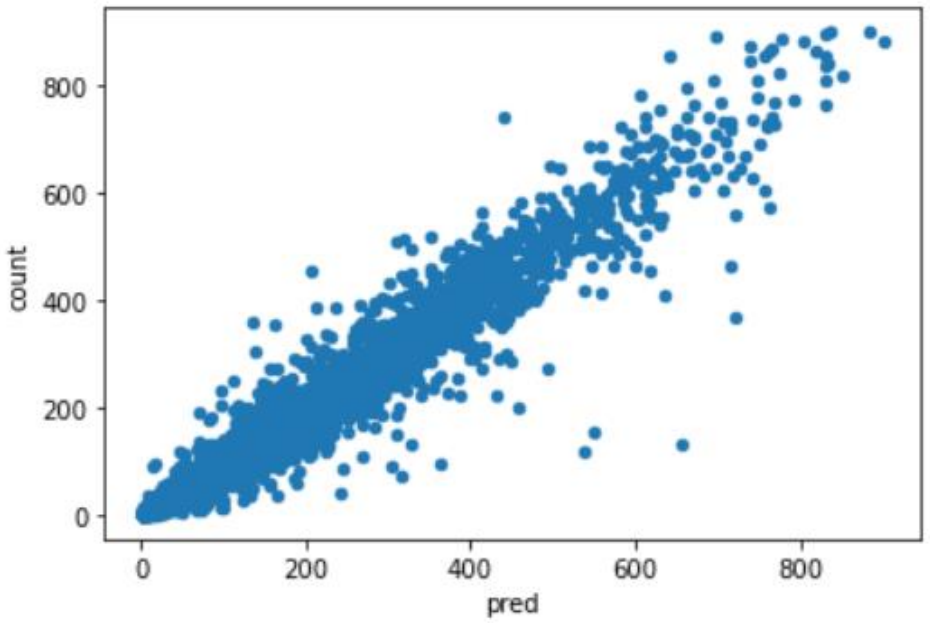
```
df_x = df.drop(['count','casual','registered', 'datetime', 'datetimeobj'], axis=1) # すべて放り込む
df_y = df['count']

# トレーニングデータとテストデータに分ける
(train_x, test_x, train_y, test_y) = train_test_split(
    df_x, df_y, test_size=0.3, random_state=0)

# ランダムフォレストモデルを生成
# n_estimators 決定木をいくつ生成するか（デフォルトは10）
clf = RandomForestRegressor(random_state=0, n_estimators=20)
clf = clf.fit(train_x, train_y)

# 結果検証(R2決定係数)
print(clf.score(test_x, test_y))
print(clf.score(train_x, train_y))

# 連続値の予測の場合は、正答率やAUCで評価できない。
# RMSLE(Root Mean Squared Logarithmic Error)
# 予測値と結果の対数の差の二乗の平均の平方根
pred = clf.predict(test_x)
rmsle = np.sqrt(mean_squared_log_error(test_y, pred))
print('RMSLE:' + str(rmsle))
```

```
0.9404809267400105
0.9912824824908756
RMSLE:0.34685178191990634
```

予測モデル構築：ランダムフォレスト

```
# 予測値と結果をプロット
result = pd.DataFrame(test_y)
result["pred"] = pred
result.plot.scatter(x="pred", y="count")
```

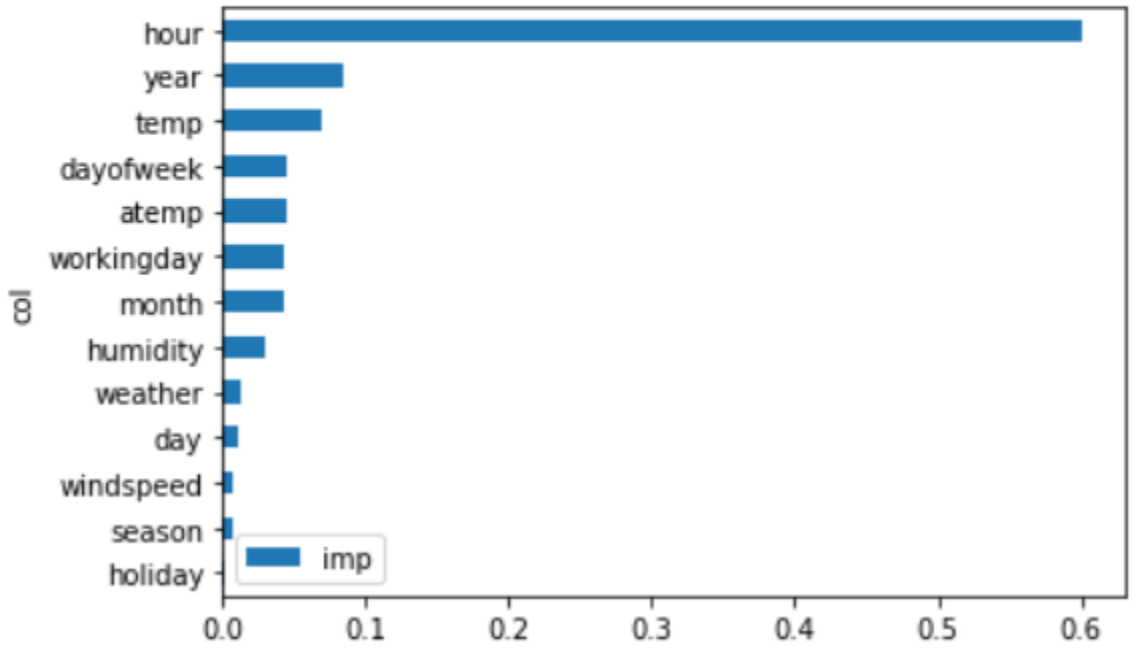
```
<AxesSubplot:xlabel='pred', ylabel='count'>
```



予測モデル構築：ランダムフォレスト

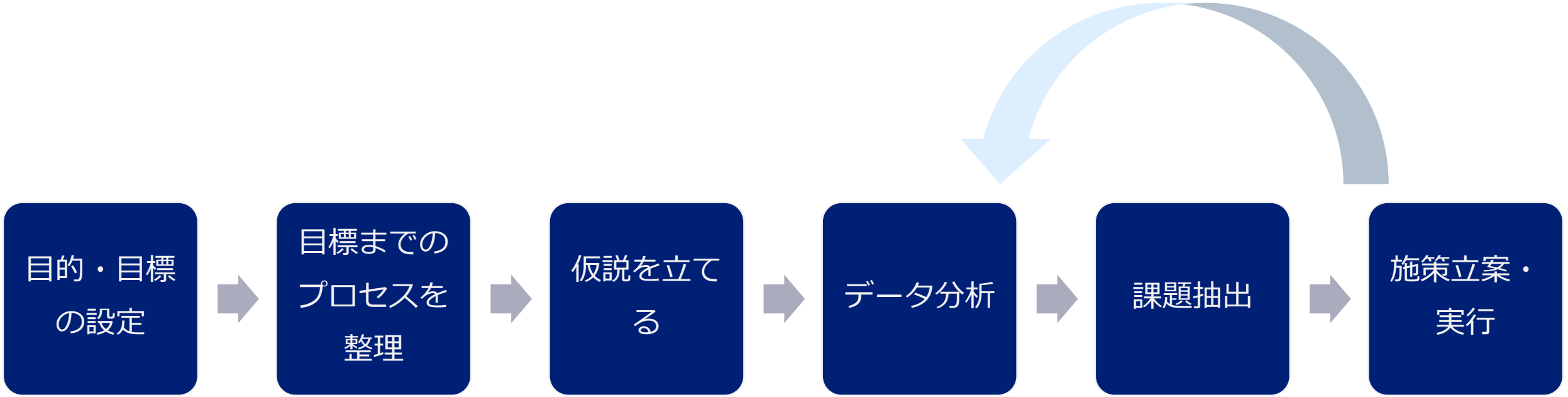
```
# ランダムフォレストの説明変数の重要度を可視化
fea_rf_imp = pd.DataFrame({'imp': clf.feature_importances_, 'col': df_x.columns})
fea_rf_imp.sort_values(by='imp', ascending=True).plot.barh(x='col', y='imp')
```

```
<AxesSubplot:ylabel='col'>
```



まとめ

データ分析の流れ



まとめ

- データ分析は仮説を検証するための道具である
- データ分析は意思決定のための道具である
- データを分析することで
  - ✓ 仮説を検証することができる
  - ✓ 新たな気づきを得られる
  - ✓ 改善すべき課題を抽出できる
  - ✓ 施策の実施結果を検証できる
  - ✓ データに基づいた意思決定が可能となる
- ビジネスの現場にデータ分析を活用しましょう！

愛知県 デジタル活用人材育成事業

Powered by