

[https://github.com/hilolani/noisy\\_diabetes/](https://github.com/hilolani/noisy_diabetes/)



データとコード

# 人工知能(機械学習)モデリングで やってはいけないこと 回帰モデルを例に

富山大学医学部

神戸大学医学部

赤間啓之

実習ファイル



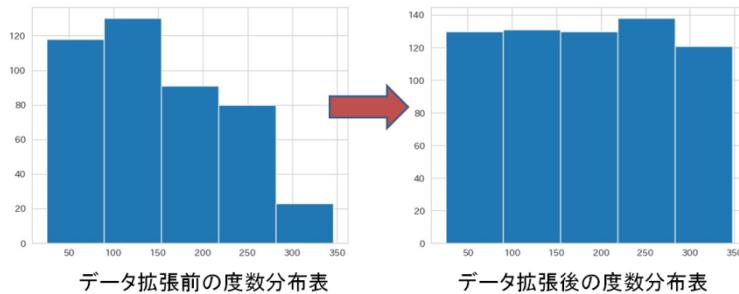
[How2avoid\\_doubledipping\\_keras\\_regression\\_noisy\\_diabetes.ipynb/py](#)

# ポイント

- データサイズを大きくしてモデルの精度を上げるデータ拡張(data augmentation)は、慎重に用いないと、魔術的に不自然な超高精度を叩き出すことがしばしばあります。
- これはデータ拡張が同一患者の同一診断画像を操作するため、反復測定のように、すべてのデータを患者の区別なくモデリングとその評価に投入してしまいがちだからです。
- その結果、同じ患者の情報が推論と検証に跨って使われるため、あらかじめ答えを見ながら問題を解くことになってしまいます。
- これを”double dipping(ソースの二度漬け)”と言います
- 回帰モデルで、実測値と推定値の間で異様に高い相関係数が導出される場合、これを魔術的”voodoo correlation(ブードゥー相関)”と言います。



# ポイント(続き)



- しかし、これだけがイカサマのすべてではありません。もっと悪質な方法があります。
- データの分散情報を考慮してそれを人為的に調節することです。
- つまり、度数分布表のすべてのbinsの高さを揃え、**稀少でクリティカルな関心領域のみデータ拡張を行う方法**です。
- この**部分調整、偏調整型ダブルディッピング**は、データセット全体に一様にデータ拡張を適用する従来のダブルディッピングよりも、**魔法的ブードゥー相関**を成功裏にイカサマ達成する上で**効果的**です。

# シミュレーションで実証

- `How2avoid_doubledipping_keras_regression_noisy_diabetes.ipynb/py`
- でシンプルなシミュレーションを行い、以上のポイントを実際に示す。
- 全データと関数、コードはGitHubの公開リポジトリから読み込む。
  - [https://github.com/hilolani/noisy\\_diabetes](https://github.com/hilolani/noisy_diabetes)
- このリポジトリは、Colab上では、
  - `!pip install git+https://github.com/hilolani/noisy_diabetes.git`
- によってインストールすることができる。
- ここでは、データ拡張は、同じ実験参加者に対する反復測定の結果として実現されると仮定。
- ノイズによるデータ拡張の例
  - `noisy_diabetes/noisyy_diabetes.py`に記述
- 実際の疑似データセット
  - `noisy_diabetes/noisyy_diabetes/data`に格納

github.com/hilolani

GPU コンピューティング... IE ブックマーク Used GE Healthcare... Oral Radiology Web... WiMAX レンタル | Wi... Magnetic Resonanc... 脳MRIの正常解剖図... M R I 基本

hilolani

Overview Repositories 8 Projects Packages Stars 3

Type / to search

[https://github.com/hilolani/noisy\\_diabetes](https://github.com/hilolani/noisy_diabetes)

Popular repositories

akamalablib Python basic codes inspired by Wolfram Mathematica Public Jupyter Notebook

noisy\_diabetes The toy datasets included with Scikit-learn tend to achieve overly high accuracy when used for machine learning, so noise is added to intentionally lower the accuracy. Public Jupyter Notebook

Customize your pins

noisy\_diabetes Public Jupyter Notebook

Pin Watch 0

Hiroyuki Akama hilolani

Edit profile

2 followers · 1 following

University of Toyama

<https://www.researchgate.net/profile/Hiroaki-Akama-2>

main 1 Branch 0 Tags

hiolani Add files via upload

noisy\_diabetes

How2avoid\_doubledipping\_keras\_regression\_n...

MANIFEST.in

README.md

ReadMe\_creating\_noisy\_datasets.ipynb

ReadMe\_installation\_Colab.ipynb

how2avoid\_doubledipping\_keras\_regression\_n...

requirements.txt

setup.py

io to file Local Codespaces

Clone

HTTPS SSH GitHub CLI

[https://github.com/hilolani/noisy\\_diabetes.git](https://github.com/hilolani/noisy_diabetes.git)

Clone using the web URL.

Open with GitHub Desktop

Download ZIP

Colab上で  
!pip install git+[https://github.com/hilolani/noisy\\_diabetes.git](https://github.com/hilolani/noisy_diabetes.git)

ダウソロードを希望する場合

- ScikitLearnの医療系toy dataset, diabetes(糖尿病)は回帰問題向き
  - [http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load\\_diabetes.html#sklearn.datasets.load\\_diabetes](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)
  - But 精度の異様な高さが特徴的、機械学習のトレーニングに向かない
- 糖尿病データのノイズを含んだ私家版で同型なもの
  - dataやtargetに正規ノイズや一様ノイズを加えたり、データ拡張(data augmentation)したりしたもの

- データに平均0,標準偏差0.1のガウシアンノイズを加え、精度を敢えて落とす。
- データ拡張(data augmentation)を想定した反復測定をシミュレートするため、行データに対し、5回繰り返して一定の([-0.001, 0.001]の範囲で)一様ノイズ
  - さらに生成した各行に対応するparticipantsとsessionsのナンバリングを追加
  - ターゲットは追加5セッションのノイジーターゲット値'y\_noisy'を作成
    - 一様分布[-2, 2]の範囲に従うランダムな整数を使用して、各オリジナルのy値に追加

```
diabetes = load_diabetes()
```

```
# show the input variables from the data
pds.DataFrame(diabetes.data, columns=("age", "sex", "bmi", "map", "tc", "ldl", "hdl", "tch", "ltg", "glu"))
```

	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930

```
[ ] from noisy_diabetes import *
```

```
noisy_diabetes = load_noisy_diabetes()
pds.DataFrame(noisy_diabetes.data, columns=("age", "sex", "bmi", "map", "tc", "ldl", "hdl", "tch", "ltg", "glu"))
```

	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	-0.107226	0.050680	0.073729	0.108586	0.033467	-0.115613	-0.036430	0.043853	-0.008717	-0.115178
1	0.053797	-0.044642	-0.090275	-0.081516	0.081129	0.103402	0.130280	-0.013074	-0.126097	-0.041420
2	0.110727	0.050680	0.167225	0.106081	-0.105451	-0.094694	0.087756	-0.052354	-0.102463	-0.137767

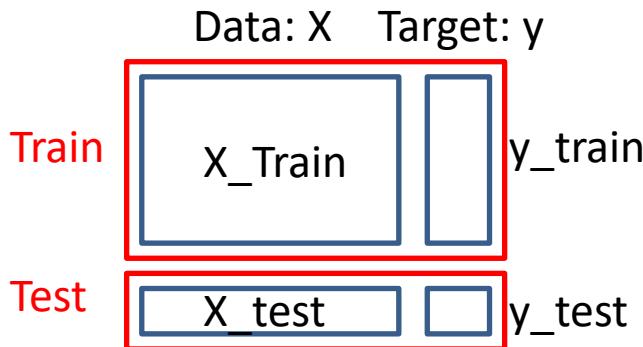
```
total_noisy_diabetes = load_total_noisy_diabetes()
df_total = pds.DataFrame(total_noisy_diabetes.data, columns=total_noisy_diabetes.columns)
df_total
```

	participants	sessions	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	0	0	-0.107226	0.050680	0.073729	0.108586	0.033467	-0.115613	-0.036430	0.043853	-0.008717	-0.115178
1	0	1	-0.107254	0.051164	0.073112	0.108686	0.033907	-0.115258	-0.036340	0.044152	-0.009285	-0.114733
2	0	2	-0.107879	0.051455	0.073902	0.108866	0.033473	-0.116342	-0.036398	0.043215	-0.008695	-0.115840
3	0	3	-0.107753	0.049759	0.073995	0.108848	0.033446	-0.115316	-0.035820	0.043690	-0.009423	-0.115210
4	0	4	-0.107363	0.050990	0.073846	0.109327	0.032722	-0.115243	-0.036483	0.043051	-0.009518	-0.114587
...	...	...	...	...	...	...	...	...	...	...	...	...
2647	441	1	-0.012037	0.051532	-0.090679	0.001281	0.122911	0.001370	0.182082	-0.075786	0.094903	0.096780
2648	441	2	-0.012543	0.050103	-0.090297	0.002462	0.121620	0.000514	0.180313	-0.076701	0.094544	0.096711
2649	441	3	-0.012821	0.050231	-0.089817	0.002347	0.122144	0.001999	0.180549	-0.076293	0.094155	0.096137
2650	441	4	-0.013084	0.051335	-0.089856	0.001280	0.121424	0.001841	0.181477	-0.076037	0.095077	0.095085
2651	441	5	-0.013301	0.049745	-0.090870	0.001433	0.122245	0.001220	0.181834	-0.076793	0.094371	0.096190

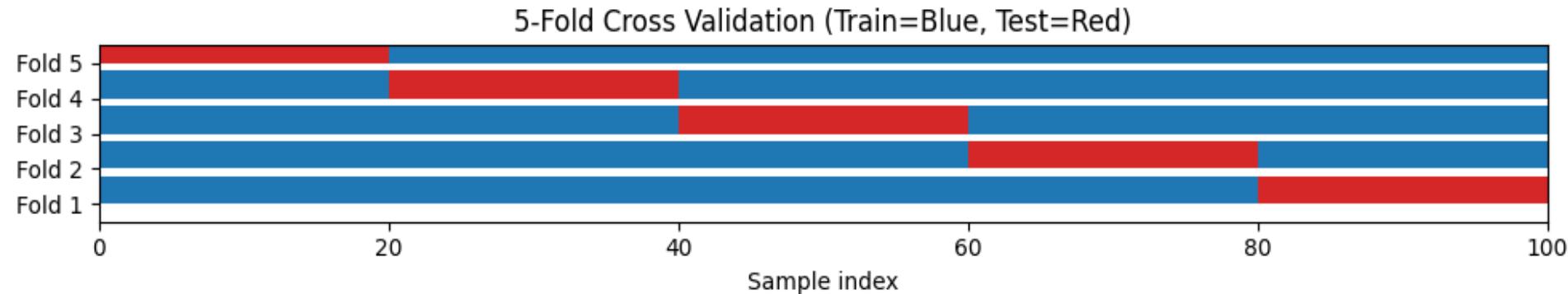
2652 rows × 12 columns

# 交差評価(Cross-validation)

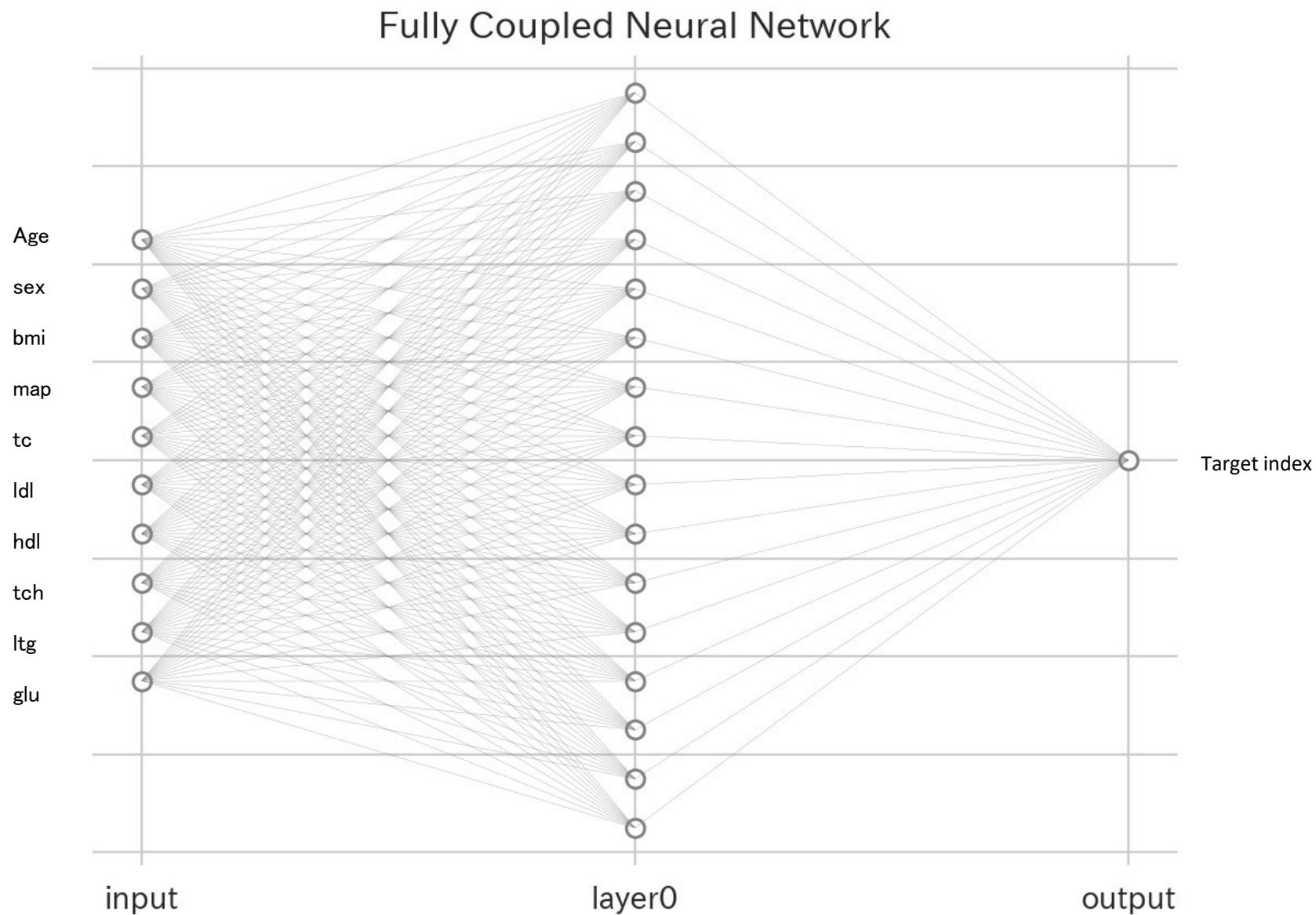
1 foldでのsplit



```
kf = KFold(n_splits=5, random_state=0, shuffle=True)
kf.get_n_splits(X, Y)
X = pds.DataFrame(X)
Y = pds.DataFrame(Y)
#参加者番号を取得するためにPandasのDataFrameに変換する
for i, (train_index, test_index) in enumerate(kf.split(X, Y)):
    print(f"Fold {i}:")
    X_train = X.iloc[train_index]
    X_test = X.iloc[test_index]
    y_train = Y.iloc[train_index]
    y_test = Y.iloc[test_index]
#以下略
```



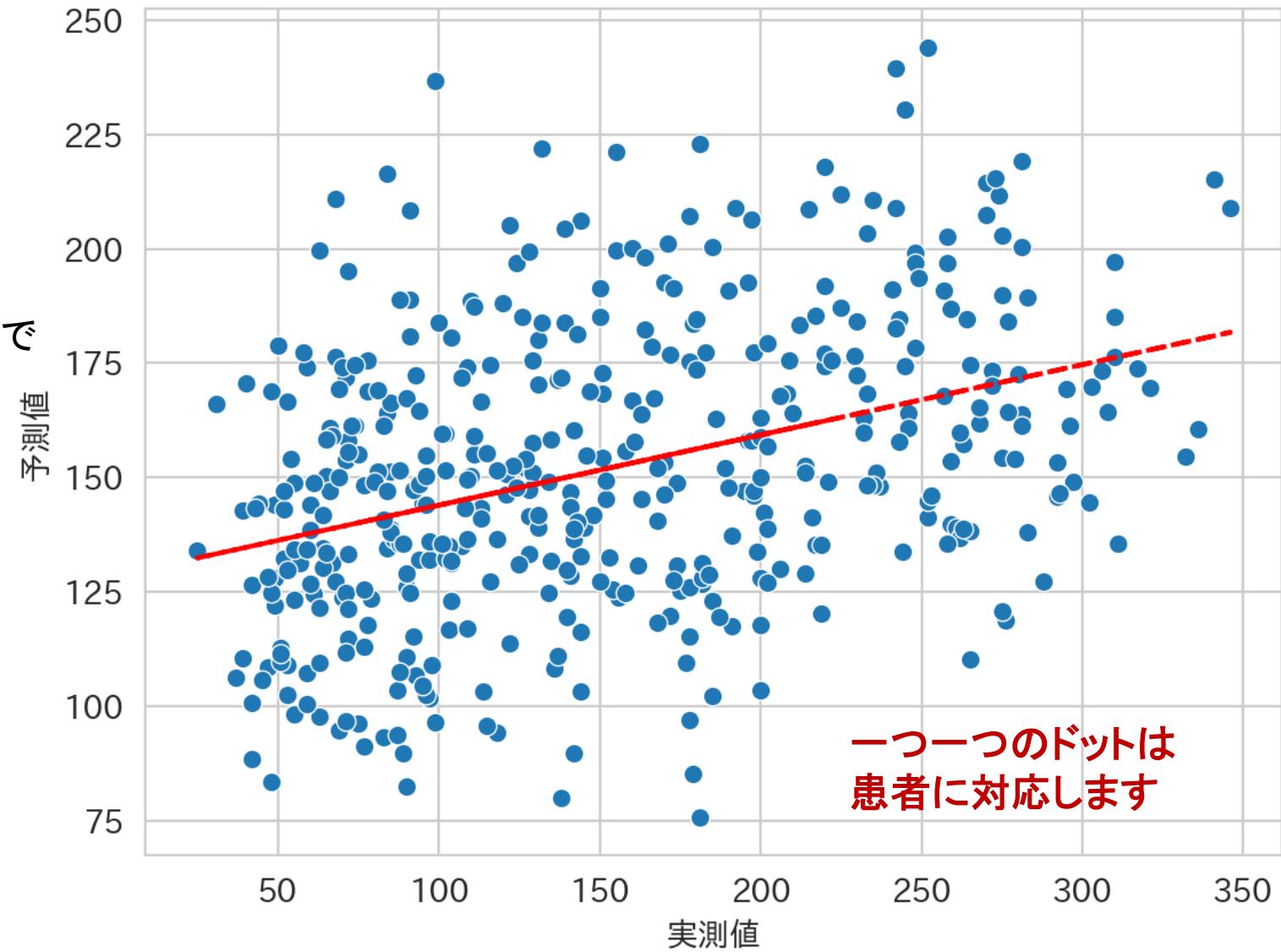
# モデル 隠れ層1層だけの単純なニューラルネットワークモデル



## 実測値と予測値の散布図

普通に  
問題の無い答え

ただ中間層1層  
だけ、ノイズも  
大きいデータなので  
精度はさほど  
上がらない



回帰直線:  $y = 0.1537x + 128.6$

相関係数 (実測値と予測値の間の): **0.376345**

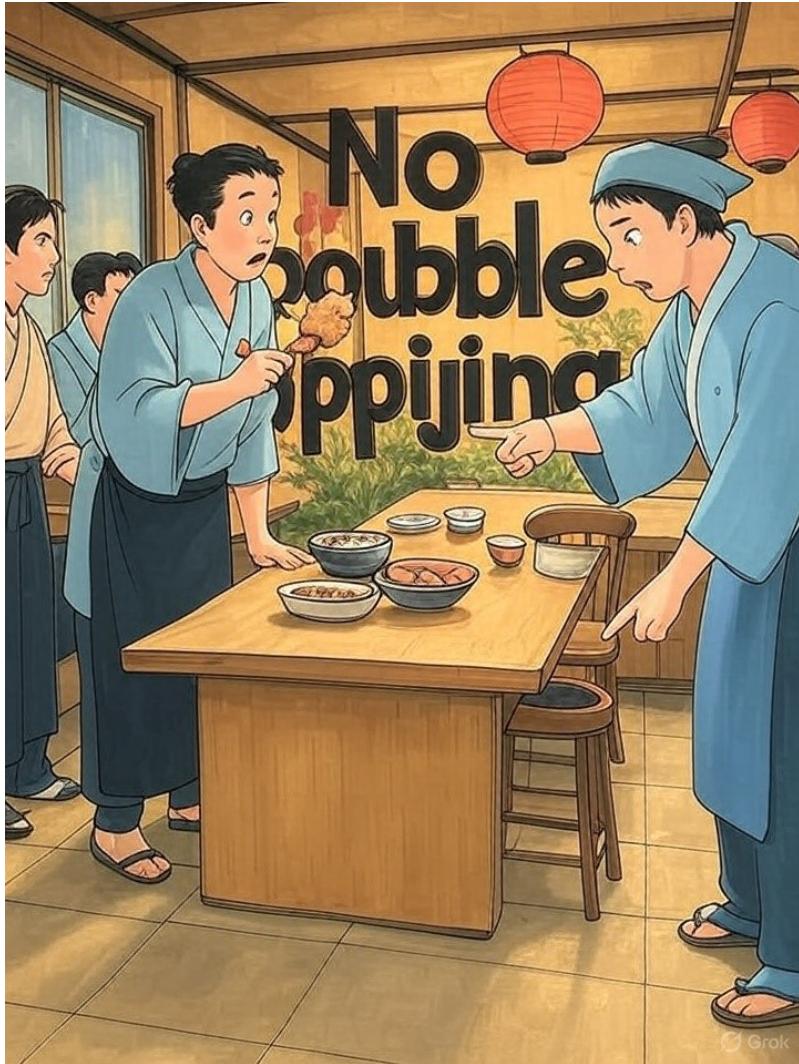
p value: 2.549145649539668e-16

まあまあ普通にRは有意です。

機械学習の基礎の基礎を忘れるとな人工  
知能(AI)など全く無意味になります

## 誤ったモデリングの例

# Double dipping 二度漬けから 魔術的ブードゥー相関へ



その告発は脳画像解析から始まった

# 機械学習のイカサマ

- 脇坂 崇平(理化学研究所脳科学総合研究センター), 『魔術的相関と仮説形成』 <https://phsc.jp/dat/rsm/2009061404.pdf>
- “Voodoo correlation in social neuroscience” --Vulらは、多くのfMRI研究に見受けられる不自然なまでに高い(故に”魔術”によるかの様な)相関は、実験データの不適切な取り扱いに起因していると主張した。
  - 仮説形成(“どの”部位が関連しているか)に用いたデータを仮説検証(その部位が”どれ程”関連しているか)に用いる、という一種の循環論法が行われている、というのである。
- 推論と検証に同一データを使うことを NIHのKriegeskorte, Bakerらは”double dipping(ソースの二度漬け)”と呼んでいる。
  - 1. Vul, E., Harris, et. al., Puzzlingly high correlations in fMRI studies of emotion, personality, and social cognition. Psychological Science. in the press (2008).
  - 2. Kriegeskorte, N., et. al. Circular analysis in systems neuroscience: the dangers of double dipping. Nat Neurosci 12, 535-40 (2009).

# Double dipping 二度漬け

禁止！



トレーニングセットにあるものが

テストセットにも浸かっている

同じ実験参加者(患者)から取ったデータをトレーニングセットにもテストセットにも入れるのがまさしくこれ



データ拡張(data augmentation)の際にやりがち

答えを見て問題を解いているカンニングです

以下間違ったモデリングを紹介します。さらに

[\*\*How2avoid\\_doubledipping\\_keras\\_regression\\_noisy\\_diabetes.ipynb/py\*\*](#)  
を読み進めてください。

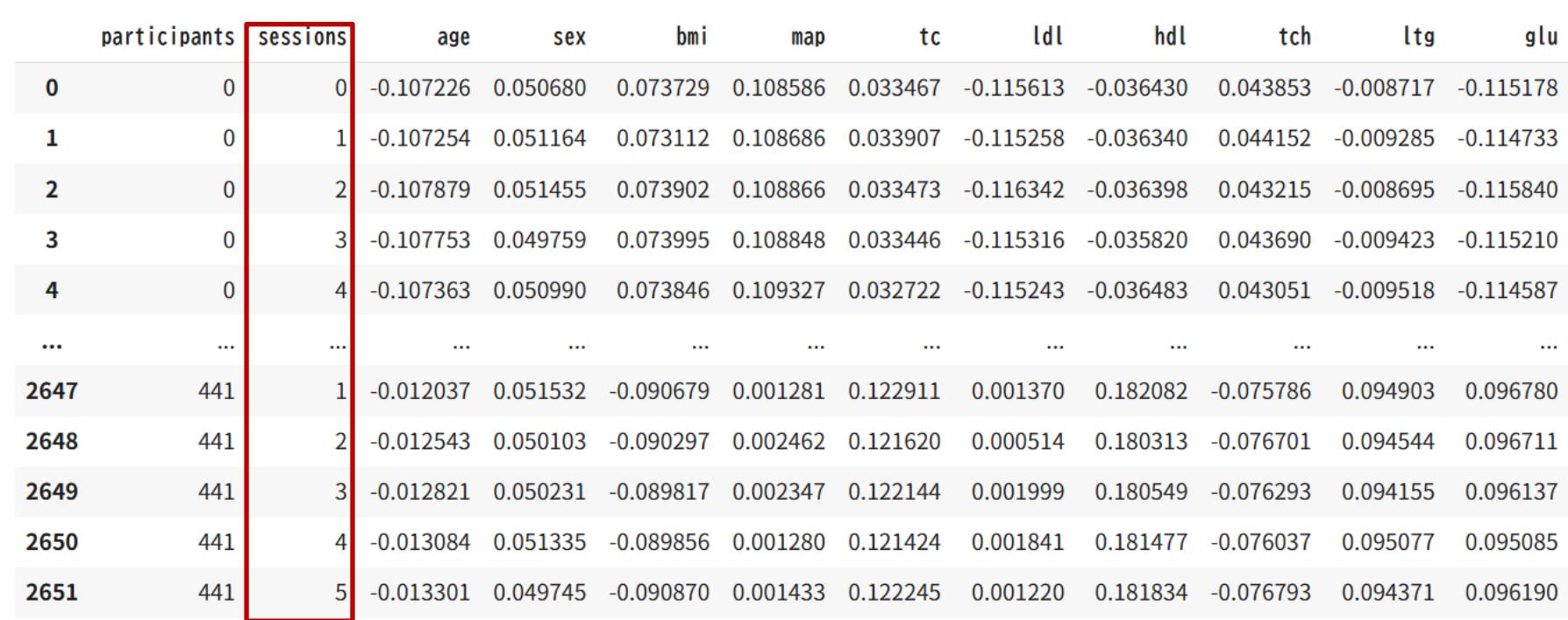
# ふたつの魔術的相関導出法



実験参加者の情報はトレーニングからテストに忍び込み、二度漬けを誘う

データ拡張のため、反復測定のすべてのデータを実験参加者の区別なく投入  
**魔術相関解その1**

# データ拡張1



	participants	sessions	age	sex	bmi	map	tc	ldl	hdl	tch	ltg	glu
0	0	0	-0.107226	0.050680	0.073729	0.108586	0.033467	-0.115613	-0.036430	0.043853	-0.008717	-0.115178
1	0	1	-0.107254	0.051164	0.073112	0.108686	0.033907	-0.115258	-0.036340	0.044152	-0.009285	-0.114733
2	0	2	-0.107879	0.051455	0.073902	0.108866	0.033473	-0.116342	-0.036398	0.043215	-0.008695	-0.115840
3	0	3	-0.107753	0.049759	0.073995	0.108848	0.033446	-0.115316	-0.035820	0.043690	-0.009423	-0.115210
4	0	4	-0.107363	0.050990	0.073846	0.109327	0.032722	-0.115243	-0.036483	0.043051	-0.009518	-0.114587
...	...	...	...	...	...	...	...	...	...	...	...	...
2647	441	1	-0.012037	0.051532	-0.090679	0.001281	0.122911	0.001370	0.182082	-0.075786	0.094903	0.096780
2648	441	2	-0.012543	0.050103	-0.090297	0.002462	0.121620	0.000514	0.180313	-0.076701	0.094544	0.096711
2649	441	3	-0.012821	0.050231	-0.089817	0.002347	0.122144	0.001999	0.180549	-0.076293	0.094155	0.096137
2650	441	4	-0.013084	0.051335	-0.089856	0.001280	0.121424	0.001841	0.181477	-0.076037	0.095077	0.095085
2651	441	5	-0.013301	0.049745	-0.090870	0.001433	0.122245	0.001220	0.181834	-0.076793	0.094371	0.096190

2652 rows × 12 columns

- A君: 今回のデータを取る前に、6年前から同じ患者のデータを反復して探っていたことを思い出した！それを入れたらデータは6倍に増えて2652になる！これをいっぺんに交差評価用にsplitすればよい！

## 実測値と予測値の散布図

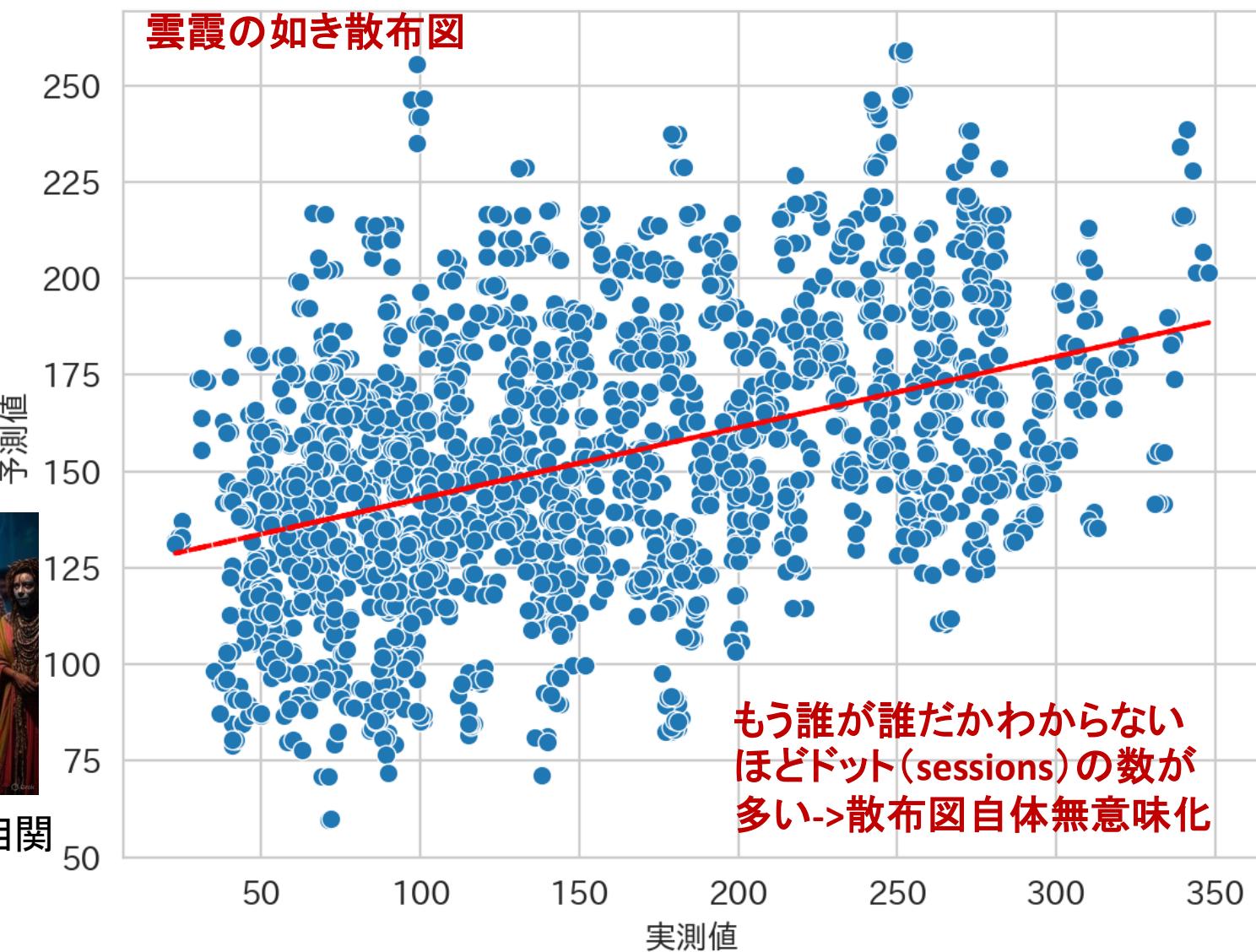
### 雲霞の如き散布図

A君の答え

どこが問題でしょう



魔術的ブードゥー相関



# これは良くない！

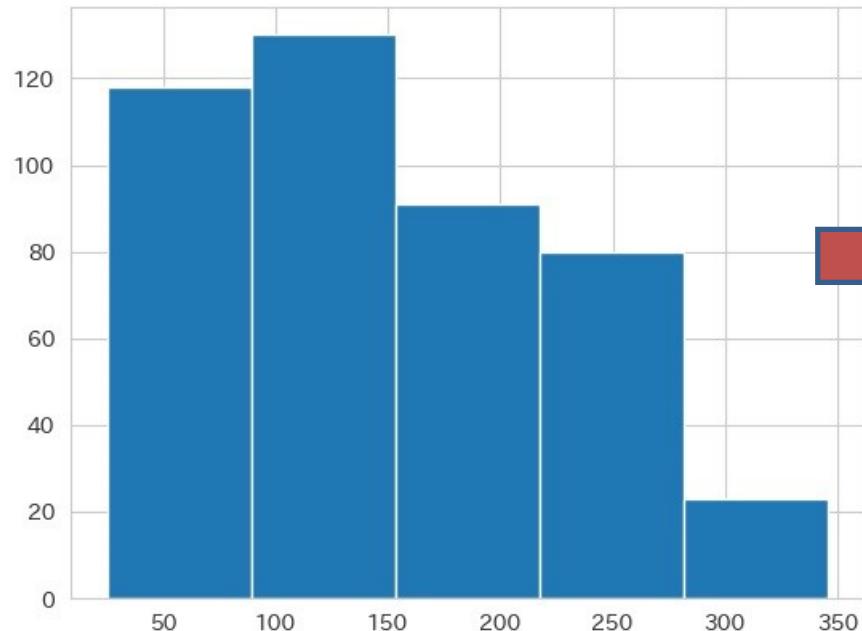
- 確かに患者数 \* セッション数(データ拡張の効果)でデータサイズは大きくなつたので精度は上がる筈
- しかし散布図のドットはセッション単位だが、**どの実験参加者のセッションかわからず、同じ人のセッションと異なる人のセッションが同条件で混在**している
- 同じ人の反復測定によるセッションがトレーニングセットとテストセットに混在している以上、そちらは**患者の特徴情報として、前者から後者へ情報漏洩**(information leakage)が起り、見かけの精度が上がりやすい
- 昔の機械学習の分野ではこれでさえ**double dipping(二度漬け)**と呼んでいた
- information leakage (情報漏洩)は、生温(ナマヌル)過ぎ！



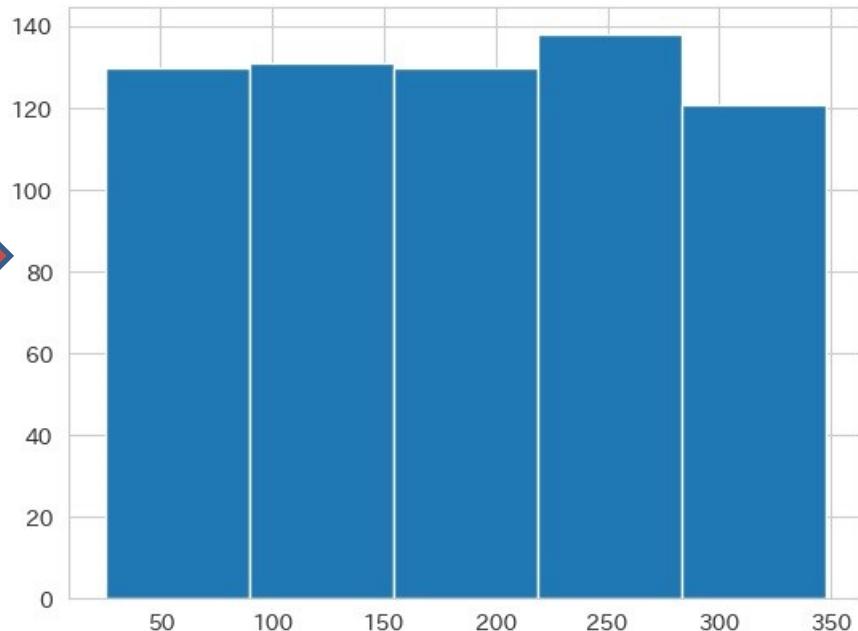
度数分布表のすべてのbinsの高さを揃え、稀少でクリティカルな関心領域のみデータ拡張

## 魔術相関解その2

# ターゲット値の度数分布表を作った



データ拡張前の度数分布表



データ拡張後の度数分布表

B君の発想：

BINごとに度数のばらつきがある！

→関心のある高い数値の患者が少ない！

→そのあたりを中心にBIN毎にデータ拡張の規模を変えよう！

[`How2avoid\_doubledipping\_keras\_regression\_noisy\_diabetes.ipynb/py`](#)  
に禁断のcodingを載せています

## 実測値と予測値の散布図

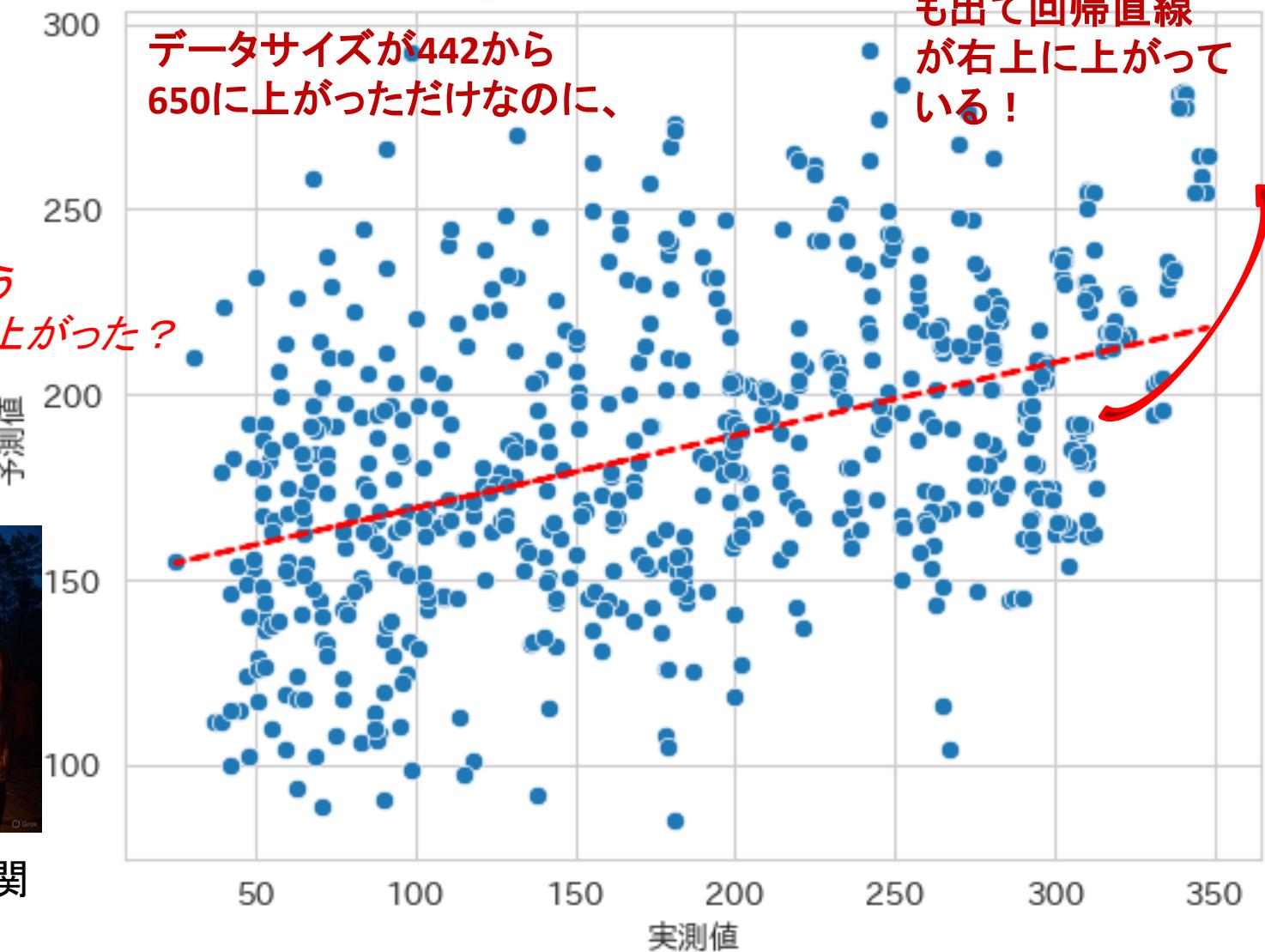
一見すごい???  
B君の答え

どこが問題でしょう

データ分布変更を伴う  
データ拡張で精度が上がった?



魔術的ブードゥー相関



回帰直線:  $0.1965x + 149.7$

相関係数 (実測値と予測値の間の): **0.438819**

p value:  $5.679333879882503e-32$

A君より良い結果!

# この方法は最悪！



- テストセットの値の分散が前もってわかっている  
=>情報漏洩(information leakage)
- 情報漏洩によって、関心の対象である高い数値が出る確率が作為的に高まる
  - 回帰曲線の傾きが1に向かって人為的に大きくなる
- しかも高い数値が出る患者に関してのdouble dipping(二度漬け)が他の患者と比べて酷くなる
  - この領域でのデータの稀少性を考えるとdouble dippingを回避する方法は存在しない
  - 最初から諦めるべき

# さらに...

- 度数分布表を作つて、特定の関心データ領域(しばしば裾野の外れ値領域)に焦点を当て、選択的にデータ拡張を適用する**部分調整型ダブルディッピング**(B君)は、
- データセット全体に一様にデータ補強を適用する従来のダブルディッピング(A君)よりも、
- 魔法的ブードゥー相関を成功裏にいかさま達成する上で効果的である!

# どうしてもデータ拡張したい場合は

- 交差評価はまずデータ拡張前の患者(実験参加者)レベルでデータのsplitを行う
- 交差評価の各foldでのトレーニングセットのみ初めてデータ拡張の対象とし、テストセットは絶対に拡張しない。
  - 全n個のfoldsにおいて、どのデータも( $n - 1$ )回データ拡張したことになる
- 患者(実験参加者)レベルの情報遺漏はない。
- ただし、元々十分大きく、ばらつきの少ないデータセットでないとさほどあまり効果はない。

# 結論

- データ内で最も関心のある値域は、しばしばデータ内で値が非常に大きい／小さいもので、値分布の裾野にあって、サイズが小さい(稀少な)ため、そこに絞ってデータ拡張をかけるときわめて効果的だが、これは機械学習における最大の不正、究極の詐欺である。
- これを*partially adjusted double dipping*(偏調整ダブルディッピング、偏調整二重湿潤)とでも呼んでみよう。
- なぜダブルディッピングなのかというと、
  - 1)モデルを計算する前に、データの値分布の事前知識をトレーニングセット、テストセットの双方に跨らせてしまう。これは今は情報漏洩(information leakage)と言うが、このことが問題になった当時はこれをダブルディッピングと呼んでいた。
  - 2)増殖した拡張データはほぼすべて出元(同じ実験参加者、患者)が同じでその特徴を共有するわけだが、サイズが小さい(稀少な)データ領域にデータ拡張をかけると、元々拡張前の拡張元データが極めて少ないため、(ほぼ)すべてトレーニングセット、テストセットに跨って配置される。これは狭義のダブルディッピングである。

# 結論に付け加えて



- 偏調整ダブルディッピングは、データ分布の裾野にかけるので、実測値対予測値の散布図に対する回帰直線の傾きを端から上向きに揚げ、1に近づける効果がある
- そのため、この部分的なデータ拡張は限定的だが、全サイズがさほど増えない割にはモデルの見かけの精度を一挙に大きくする
- おそらくこのトリックは、サイズが十分大きい大規模データでも本質的に変わりはなく使えてしまうだろう。

# 蛇足

- これほど問題を指摘しても、学の世界によっては、偏調整ダブルディッピングを止める方法は無いかもしれない。
- 偏調整ダブルディッピングがデファクトスタンダードとして大手を振ってまかり通り、それをしない論文が逆に精度不足で落とされる倒錯が、自然に慣行化されるかもしれない。
  - 学問の世界なんて所詮そんなもの
- そもそも中心極限定理によって、データサイズが大きくなれば（巨大になっても）分布が正規分布に近づくだけで、データ内で最も関心のある裾野の値域の相対的稀少性に変わりはない。
- だから偏調整ダブルディッピングの誘惑を止める方法は無い。
- 唯一の根本的な解決策は、Scikit-Learnの開発者であるGael Varoquauxが常々主張するように！回帰モデルを使わないことだけである（Varoquauxのような天才のみ理解する本質論）。
- 回帰モデルが存在する限り研究不正は続く。
  - それだけの話。