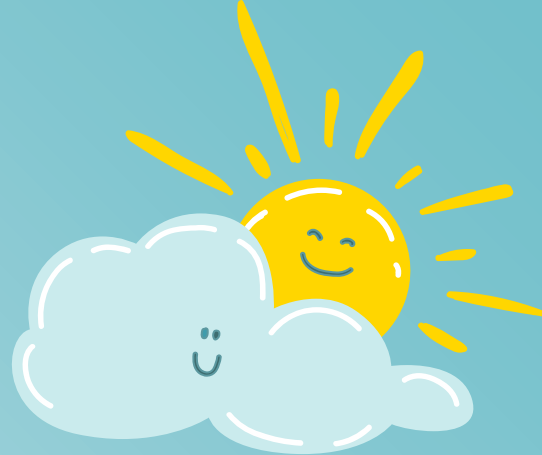


# 機械学習による天気予測



# 発表の流れ

1

機械学習と本開発の概要

2

開発プロセスの説明

3

実践

4

開発の振り返りと参考資料について

# 1

## 機械学習と本開発の概要

# 機械学習とは

機械学習は、コンピューターが大量のデータを分析して傾向などを学習する技術

データを分析する方法の1つで、データから機会が自動で学習し、データの背景にあるルールやパターンを発見する方法。学習した結果に基づいて、予測・判断する。

## 機械学習のイメージ

機械学習＝ビッグデータと呼ばれる大量のデータを用いることで、  
人工知能が自ら知識を獲得する



ビッグデータ  
(インターネットの成長とともに蓄積された  
大容量のデータ)



人工知能

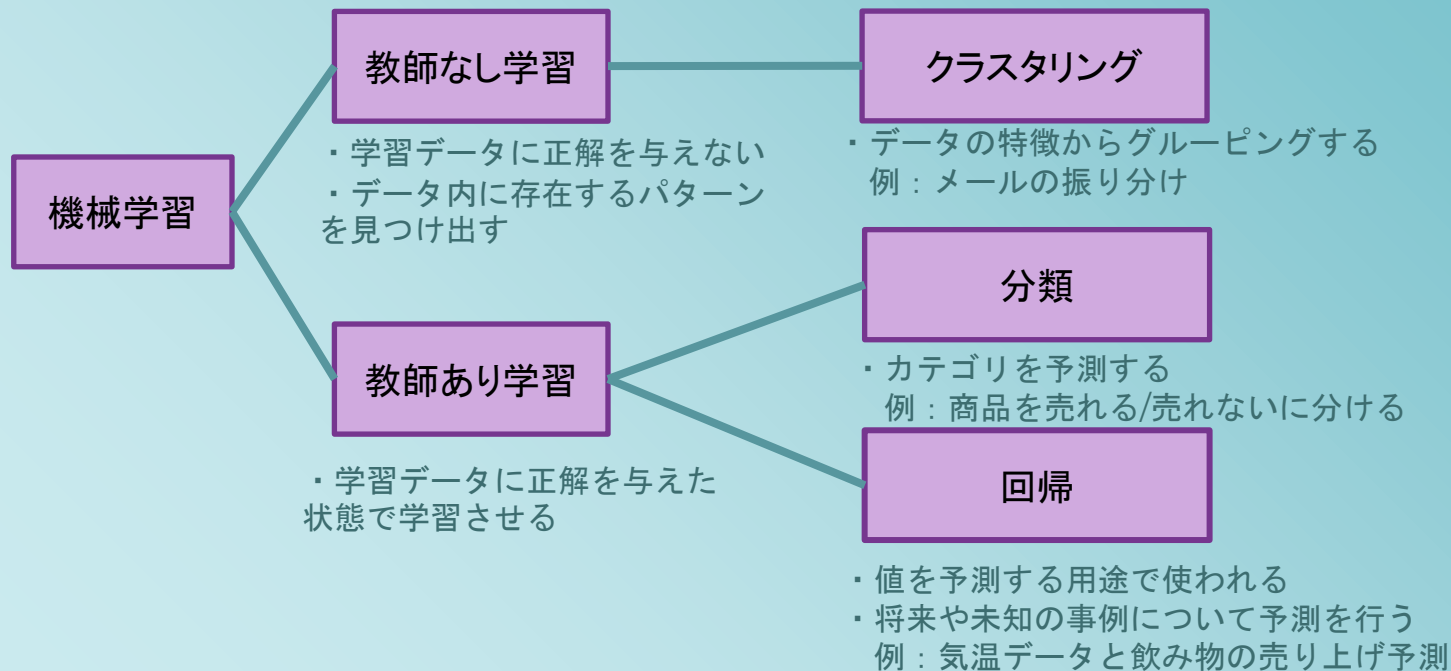


これまでのデータを分析すると、  
この家の家賃は〇〇万円になる  
だろう…

データを反復学習し、  
その中に潜むパターンを学習する！

# 機械学習とは

- ・ 教師あり学習は、学習データに正解を与えた状態で学習させる方法
- ・ 回帰モデルは将来に関して値を予測するため使用する



# 本開発について

- ・ 本開発では、線形回帰の一種であるリッジ回帰という手法を使用する
- ・ リッジ回帰は過学習を抑えるため正則化の概念を入れた線形回帰

線形回帰：

- ・ データの分布に合う直線を求めること

リッジ回帰：

- ・ 正則化された線形回帰の一つで、線形回帰に「学習した重みの二乗を加えたもの」
- ・ 係数をできるだけ小さくしつつ、予測と実際の誤差を小さくするというコンセプトの上で成り立つ

過学習：

- ・ 学習時に利用したデータのみで過剰に適合してしまうため、汎化性能が失われてしまう現象
- ・ 未知データに対する推測性能が落ちる

正則化

- ・ 過学習を防ぐための代表的な手法
- ・ モデルの複雑さを表す指標を加え、これを最小化するように学習すれば、性能と複雑さ、過学習と未学習のバランスをとった学習が実現できる

# 本開発について

本開発では、前日の情報を入力することで、次の日の最高気温を予測してくれる機械学習モデルを作成する

## 方法



過去の天気データ

学習



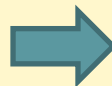
## 完成したシステムのイメージ

インプット: 今日の天気情報



これまでのデータを分析  
するに、次の日の最高  
気温は〇〇だと予測

アウトプット:  
次の日の最高気温





2



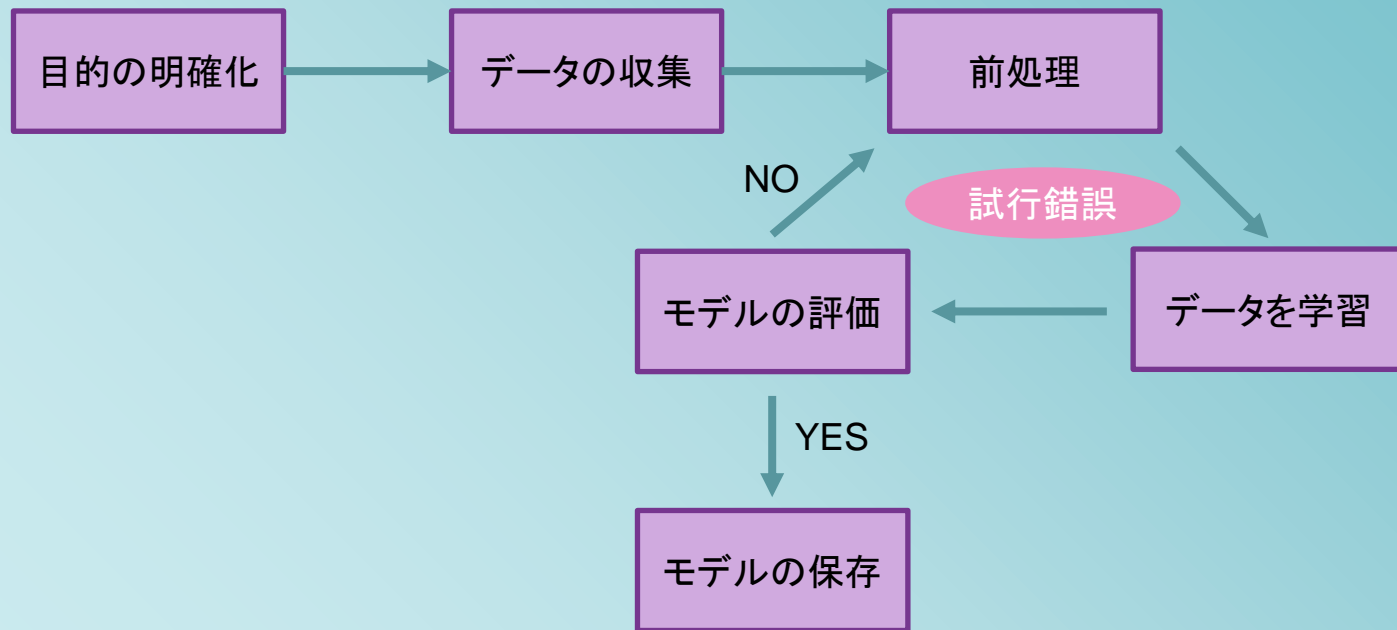
開発プロセス





# 開発プロセス

- ・ 学習させたモデルの評価がよくなければ、学習内容を変更し、データの前処理から再度行う
- ・ モデルの精度が妥当だと判断されるまで、前処理、学習、評価を繰り返す



# 開発プロセス

1. データの収集
  - a) データのスクレイピング
2. データの前処理
  - a) 欠損値の穴埋め、適切なデータへの変換、
  - b) データの把握と分析
3. 機械の学習
4. 学習モデルの評価と改善
  - a) 評価結果の出し方
  - b) 学習モデルの改善: 特徴データの追加
5. モデルの保存

# 2.1

## データ収集

# データ収集

スクレイピングにより気象庁のサイトの必要なデータのみ抜き出して、csvファイルに落とし込む

国土交通省

気象庁

Japan Meteorological Agency

ENGLISH

Other Languages

Google

提供

検索

ホーム

防災情報

各種データ・資料

地域の情報

知識・解説

各種申請・ご案内

ホーム

>

各種データ・資料

>

過去の気象データ検索

>

日ごとの値

日ごとの値

一覧表

グラフ

見出しの固定

メニューに戻る

主要要素

詳細(気圧・降水量)

詳細(気温・蒸気圧・湿度)

詳細(風)

詳細(日照・雪・その他)

前年

前月

前日

翌日

翌年

月ごとの値

旬ごとの値

半旬ごとの値

日ごとの値

東京

2023年1月(日ごとの値)

主要要素

日	気圧(hPa)		降水量(mm)			気温(℃)			湿度(%)		風向・風速(m/s)					日照時間(h)	雪(cm)		天気概況	
	現地	海面	合計	最大	平均	最高	最低	平均	最小	平均風速	最大風速	風向	最大瞬間風速	風速	風向		降雪合計	最深積雪	昼	夜
	平均	平均		1時間		10分間	1時間		10分間	1時間	10分間	1時間	10分間					1時間	10分間	1時間
1	1015.8	1018.8	—	—	—	6.6	13.0	0.9	62	28	1.7	4.1	南南東	6.6	北西	8.8	—	—	快晴	快晴
2	1015.7	1018.7	—	—	—	6.2	12.1	2.0	51	29	2.1	4.2	北西	9.0	北	6.3	—	—	晴一時曇	晴
3	1015.2	1018.2	—	—	—	5.8	11.0	0.5	42	23	2.2	4.9	北西	9.2	北北西	8.9	—	—	快晴	快晴
4	1015.2	1018.2	—	—	—	5.6	11.0	1.3	47	27	2.6	4.7	北西	8.8	北北西	8.4	—	—	晴	快晴
5	1018.8	1021.8	—	—	—	5.9	10.6	2.6	36	21	3.2	6.3	北西	12.0	北西	8.8	—	—	快晴	晴
6	1017.0	1020.1	—	—	—	5.3	9.9	0.0	47	34	1.7	3.2	西北西	5.2	北北東	7.8	—	—	薄曇時々晴	曇
7	1010.4	1013.4	—	—	—	6.0	10.4	3.0	59	44	1.7	3.9	北西	5.9	北西	8.8	—	—	晴	晴
8	1015.7	1018.7	—	—	—	7.0	12.5	1.6	58	35	1.8	3.2	南南東	5.7	東北東	8.8	—	—	快晴	晴時々曇
9	1013.5	1016.4	—	—	—	8.0	13.9	2.4	61	41	1.9	3.3	南東	4.9	南東	8.8	—	—	快晴	晴

[https://www.data.jma.go.jp/obd/stats/etrn/view/daily\\_s1.php?prec\\_no=44&block\\_no=47662&year=2019&month=01&day=1&view=p1](https://www.data.jma.go.jp/obd/stats/etrn/view/daily_s1.php?prec_no=44&block_no=47662&year=2019&month=01&day=1&view=p1)

スクレイピング: ウェブサイトを経由して情報を収集し、活用すること

# データ収集

1950年～2022年の日ごとの最高気温等の天気に関するデータをcsvファイルに落とし込んだ

東京 2019年1月(日ごとの値) 主要要素

日	気圧(hPa)		降水量(mm)			気温(℃)			湿度(%)		風向・風速(m/s)					日照時間 (h)	雪(cm)	
	現地	海面	合計	最大		平均	最高	最低	平均	最小	平均風速	最大風速		最大瞬間風速			降雪 合計	最深積雪 値
	平均	平均		1時間	10分間							風速	風向	風速	風向			
1	1019.6	1022.7	---	---	---	5.3	10.7	0.9	54	28	2.3	4.7	西北西	7.4	北西	8.9	---	---
2	1017.7	1020.7	---	---	---	6.2	10.9	2.2	47	31	2.9	5.8	北西	10.8	北西	8.7	---	---
3	1020.2	1023.2	---	---	---	4.9	10.9	-0.3	49	24	3.1	6.6	北西	13.2	北西	8.9	---	---
4	1023.0	1026.0	---	---	---	5.1	9.8	0.2	59	34	2.0	4.1	南	6.3	南	8.9	---	---
5	1010.6	1013.6	---	---	---	7.4	14.0	0.9	52	36	2.1	5.4	北	10.5	北	8.9	---	---
6	1015.2	1018.2	---	---	---	5.3	8.0	2.4	44	35	3.0	5.2	北西	11.1	北北西	2.7	---	---
7	1016.4	1019.4	---	---	---	5.4	10.6	0.4	42	24	2.8	6.8	北西	11.5	北西	8.8	---	---
8	1012.3	1015.3	---	---	---	4.6	9.8	0.7	59	41	1.6	3.2	南東	5.4	北北西	8.5	---	---
9	1016.6	1019.6	---	---	---	4.5	8.0	1.0	46	29	4.7	9.3	北西	16.9	北西	9.0	---	---
10	1014.8	1017.9	---	---	---	2.0	5.2	-1.2	48	34	1.8	3.8	北西	6.3	北西	1.8	---	---
11	1014.6	1017.6	---	---	---	6.4	12.2	-0.2	47	29	2.9	7.1	北西	13.1	北西	8.8	---	---
12	1016.7	1019.7	0.0	0.0	0.0	4.6	6.2	2.7	67	48	2.0	4.7	北西	6.7	北西	0.0	---	---
13	1016.1	1019.1	0.0	0.0	0.0	6.2	10.1	3.9	62	38	2.2	4.2	北東	8.2	北東	6.6	---	---
14	1020.0	1023.0	---	---	---	5.6	9.8	1.3	59	39	2.0	3.5	東北東	6.5	北東	9.1	---	---
15	1016.3	1019.3	0.5	0.5	0.5	5.6	10.5	0.4	76	53	2.3	7.3	南南西	14.7	南南西	2.9	---	---
16	1009.1	1012.1	---	---	---	6.6	11.3	1.8	50	31	2.9	6.0	北北西	10.4	北北西	6.8	---	---



要素	CSVファイル の列名
年月日	Date
降水量	precipitation
平均気温	avg_temp
最高気温	max_temp
最低気温	min_temp
平均風速	windspeed
降雪合計	snowfall

# データ収集

Tokyo.csv

	Date	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall
1	1950/1/1	16.7	3.5	4.7	2.2	0.0	0.0
2	1950/1/2	0.1	5.5	8.3	3.4	0.0	0.0
3	1950/1/3	0.0	7.2	11.7	4.1	0.0	0.0
4	1950/1/4	0.0	5.8	9.2	2.3	0.0	0.0
5	1950/1/5	0.0	2.6	5.8	0.6	0.0	0.0
6	1950/1/6	0.0	0.5	7.5	-2.8	0.0	0.0
7	1950/1/7	0.0	1.0	7.3	-3.3	0.0	0.0
8	1950/1/8	0.0	1.6	9.8	-4.6	0.0	0.0
9	1950/1/9	0.0	3.9	9.6	-2.4	0.0	0.0
10	1950/1/10	20.6	2.0	5.8	0.4	0.0	0.0
11	1950/1/11	0.0	2.4	7.4	-2.0	0.0	0.0
12	1950/1/12	0.0	4.0	10.8	-2.4	0.0	0.0
13	1950/1/13	0.0	4.3	9.8	0.9	0.0	0.0
14	1950/1/14	8.5	0.1	1.9	-1.6	0.0	0.0
15	1950/1/15	1.2	6.6	10.4	1.7	0.0	0.0
16	1950/1/16	0.0	5.6	13.7	0.4	0.0	0.0
17	1950/1/17	0.0	5.9	14.9	-1.3	0.0	0.0
18	1950/1/18	0.6	6.3	8.4	2.8	0.0	0.0
19	1950/1/19	0.3	8.1	9.7	6.2	0.0	0.0
20	1950/1/20	0.0	6.3	11.8	2.4	0.0	0.0
21	1950/1/21	0.0	5.8	12.0	1.2	0.0	0.0
22	1950/1/22	0.0	4.3	8.1	1.7	0.0	0.0
23	1950/1/23	0.0	5.3	11.5	-1.4	0.0	0.0



2.2

データの预处理



# データの事前処理

- ・ 前処理: 用意したデータを、アルゴリズムが学習できるよう“きれいな”データに整えること
- ・ データを効率よく操作するため、pandasを利用する
- ・ Pandasは、データの読み込み、抽出、整理などするために使用することができる

```
import pandas as pd
```

```
weather = pd.read_csv("Tokyo.csv", index_col="Date")
```

Tokyo.csvというファイルを読み込んで、Date列をインデックスとして指定したpandasのweatherという名前のデータフレームを作成

```
weather.loc["2022/5/1":"2022/5/10"]
```

2022/5/1~2022/5/10までの  
データを表示

	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall
Date						
2022/5/1	0.5	8.3	11.2	7.2	5.0	0.0
2022/5/2	0.0	9.2	13.3	5.3	2.7	0.0
2022/5/3	0.0	9.6	13.9	5.5	2.9	0.0
2022/5/4	3.5	9.6	14.6	6.8	5.5	0.0
2022/5/5	0.0	12.1	18.7	5.9	6.2	0.0
2022/5/6	0.0	18.9	27.4	11.1	5.5	0.0
2022/5/7	1.5	15.9	23.7	11.5	5.5	0.0
2022/5/8	0.0	11.2	15.6	8.5	4.0	0.0
2022/5/9	0.0	13.9	18.6	7.9	4.3	0.0
2022/5/10	0.0	16.6	24.8	8.7	2.1	0.0



# ★ データの前処理

データを把握し、必要に応じて数字を導入したり、列を削除したりする

```
weather["snowfall"].value_counts()
```

0.0	26538
1.0	35
2.0	24
3.0	12
5.0	9
9.0	7
7.0	5
6.0	5
4.0	5
10.0	3
16.0	2
15.0	2
21.0	2
8.0	2
11.0	2
13.0	2
18.0	2
14.0	1
33.0	1
26.0	1
17.0	1
27.0	1
23.0	1

Name: snowfall, dtype: int64

snowfallの列で、どんなデータがどれくらいあるのか表示

```
weather.apply(pd.isnull).sum()
```

precipitation	0
temp_avg	0
temp_max	0
temp_min	0
windspeed	0
snowfall	0
dtype:	int64

欠損値の数を表示する  
(存在する場合、削除するか、数字を代入するか対応する)

```
weather.apply(lambda x: (x==0).sum())
```

precipitation	17776
temp_avg	9
temp_max	6
temp_min	90
windspeed	4190
snowfall	26538
dtype:	int64

値が0である要素の数を表示  
(分析に影響する場合は、代わりの数字をを代入するなどして対応する)

# ★ データの前処理

データの型を、機械学習に使用するため適切な型に変更する

`weather.dtypes`

```
precipitation    float64
temp_avg         float64
temp_max         float64
temp_min         float64
windspeed        float64
snowfall         float64
dtype: object
```

各列のデータの型を表示

`weather.index`

```
Index(['1950/1/1', '1950/1/2', '1950/1/3', '1950/1/4', '1950/1/5', '1950/1/6',
      '1950/1/7', '1950/1/8', '1950/1/9', '1950/1/10',
      ...,
      '2022/12/22', '2022/12/23', '2022/12/24', '2022/12/25', '2022/12/26',
      '2022/12/27', '2022/12/28', '2022/12/29', '2022/12/30', '2022/12/31'],
      dtype='object', name='Date', length=26663)
```

インデックスに指定したデータ (Date) の表示  
→Dateはobject型

`weather.index = pd.to_datetime(weather.index)`

インデックスをdatetime (日時を表すための型) に変更

`weather.index.year`

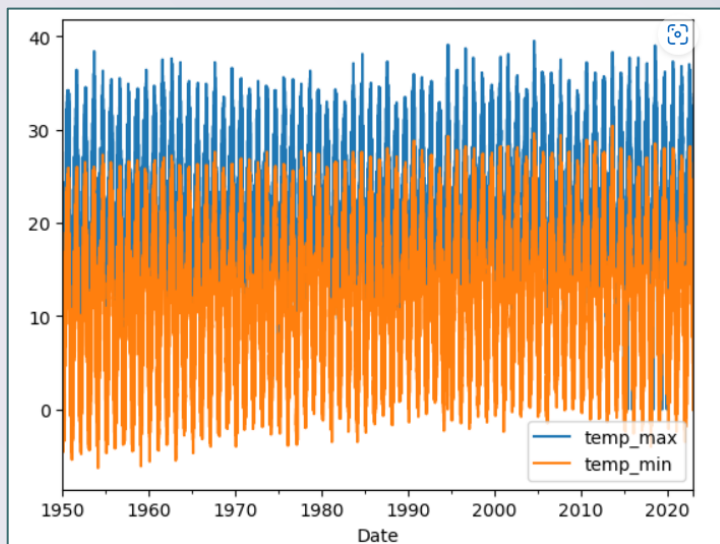
インデックスのyearのみ表示

```
Int64Index([1950, 1950, 1950, 1950, 1950, 1950, 1950, 1950, 1950, 1950, 1950,
            ...,
            2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022],
            dtype='int64', name='Date', length=26663)
```

# ★ データの前処理

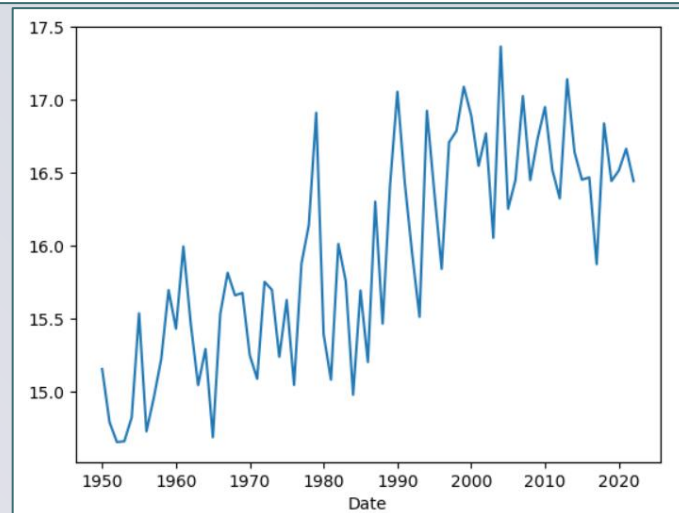
- ・ グラフを作成し、データの可視化を行うことで全体の傾向を把握する
- ・ 全体を見ることで、分析に影響するようなデータがないか確認する

```
weather[["temp_max", "temp_min"]].plot()
```



最高気温と最低気温を示す図を表示

```
weather.groupby(weather.index.year).mean()["temp_avg"].plot()
```



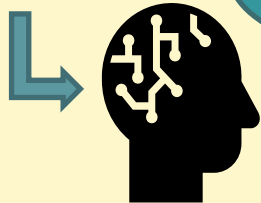
- ・ 平均気温の列を年毎にグループ化し、年ごとの平均値を示す図を表示
- ・ グラフは年ごとの平均気温の変化を示している

# ★ データの前処理

- ・ 次の日の最高気温を示した列(target)を追加する(targetの数字が正解データとなる)
- ・ 本開発では、過去のデータを使用しtargetの値を予測するモデルを作成する

## 完成したシステムのイメージ

入力: 今日の天気情報



これまでのデータを  
分析するに、次の日  
の最高気温は〇〇  
だと予測

出力:  
次の日の最高気温

```
weather["target"] = weather.shift(-1)["temp_max"]
```

	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall	target
Date							
1950-01-01	16.7	3.5	4.7	2.2	0.0	0.0	8.3
1950-01-02	0.1	5.5	8.3	3.4	0.0	0.0	11.7
1950-01-03	0.0	7.2	11.7	4.1	0.0	0.0	9.2
1950-01-04	0.0	5.8	9.2	2.3	0.0	0.0	5.8
1950-01-05	0.0	2.6	5.8	0.6	0.0	0.0	7.5
...	...	...	...	...	...	...	...
2022-12-27	0.0	6.8	12.3	2.0	2.7	0.0	11.9
2022-12-28	0.0	6.5	11.9	2.8	1.4	0.0	13.7
2022-12-29	0.0	8.1	13.7	2.9	2.0	0.0	11.8
2022-12-30	0.0	6.6	11.8	2.8	1.9	0.0	10.5
2022-12-31	0.0	5.7	10.5	3.1	1.6	0.0	NaN

- ・ target追加し、temp\_max列を1行ずらした値に設定
- ・ target列には、temp\_max列のその行の次の行の値が格納される(次の日の最高気温)



# 2.3

## 機械の学習



# 機械の学習

リッジ回帰分析を行うため、重回帰モデルのRidge関数をインポートする

```
from sklearn.linear_model import Ridge  
reg = Ridge(alpha=.1)
```

リッジ回帰モデルのインポート

Scikit-learn: 機械学習を行うためのプログラムを簡単に実装できるライブラリ

Ridge回帰のインスタンスを作成し、\*alphaを0.1に設定

\*alphaは正則化項の重みを制御するパラメータで、値が大きくなるほど正則化が強くなる。alphaが小さい場合、正則化の効果が小さくなり、過学習のリスクが高くなり、alphaが大きすぎる場合は、モデルの複雑性が低すぎて、予測力が低下する可能性がある。適切なalphaの値を見つけるには、交差検証などの手法を用いて評価する必要があるが、初めてリッジ回帰を使用する場合0.1から始めることが一般的

# 機械の学習

- ・ 学習用データとテストデータに分割する
- ・ 予測したい数値(最高気温)を出すための特徴量(予測の手がかりとなる数値)を設定する
- ・ fitメソッドに学習用データを指定して学習させる

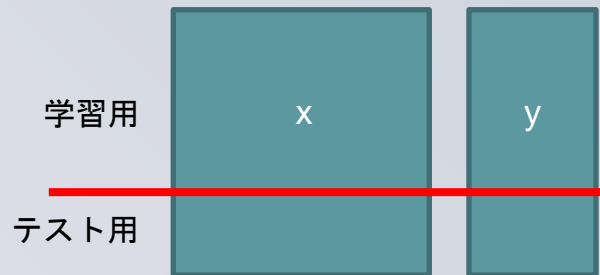
## データの分割

```
train = weather.loc[:"2008/12/31"]
```

2008年12月31日より前のデータを学習用データとして設定

```
test = weather.loc["2009/1/1":]
```

2009年1月1日より先のデータをテスト用データとして設定



## 特徴量の設定

```
predictors = ["precipitation", "temp_max", "temp_min"]
```

最高気温を予測するための特徴量を設定する

## データの学習

```
reg.fit(train[predictors], train["target"])
```

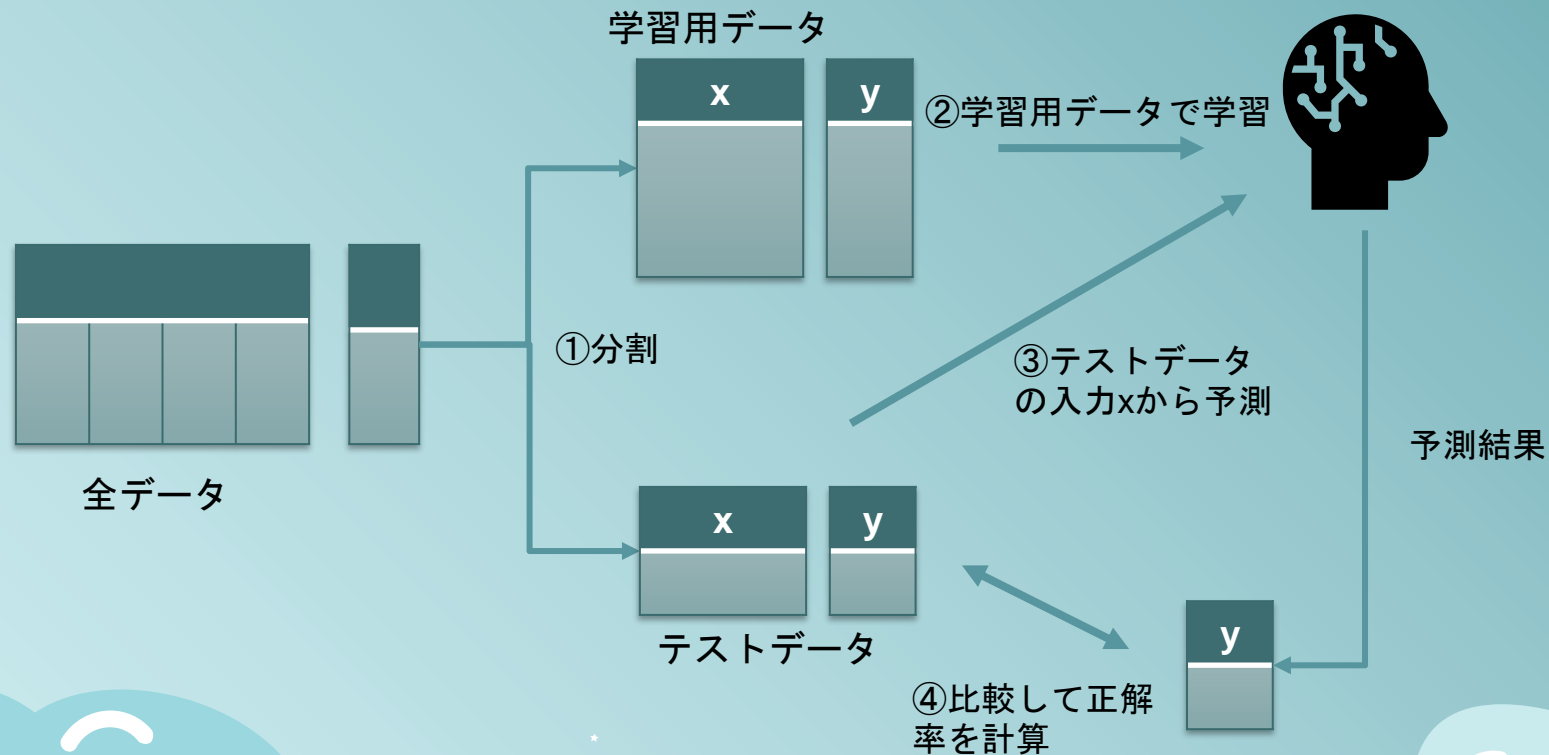
特徴量データx

正解データy

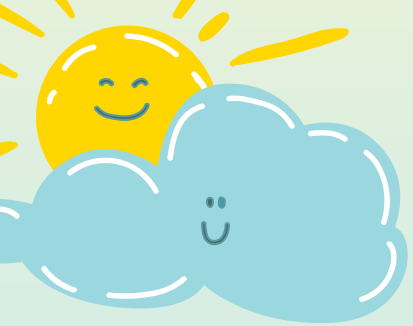
- ・ fitメソッドを使用し、学習を実行
- ・ 訓練データの特徴量と、正解データを引数として設定する

# 機械の学習

- ・ 学習用データは学習に利用するデータ
- ・ テストデータは正解率など予測性能を評価するデータ







# 2.4

## 機械モデルの評価

You can enter a subtitle here if you need it



# 機械モデルの評価

- ・ 評価指標を用いて、モデルの精度を判別する
- ・ 予測した値がどれだけ正解値とずれているか確認するためMAEを使用する

```
from sklearn.metrics import mean_absolute_error
```

平均絶対誤差 (MAE: Mean Absolute Error): 各データに対し予測値と正解値の差 (= 誤差) の絶対値を計算し、その総和をデータ数で割った値を出力する

```
mean_absolute_error(test["target"], predictions)
```

この時点でのMAE :

2.422452483454122

モデルの精度を上げ、この数字を小さくしたい



2.5

モデルの改善

# モデルの改善

モデルの精度を高めるため、予測のため使用する特徴量を追加する

```
weather["month_max"] = weather["temp_max"].rolling(30).mean()
```

	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall	target	month_max
Date								
1950-01-01	16.7	3.5	4.7	2.2	0.0	0.0	8.3	NaN
1950-01-02	0.1	5.5	8.3	3.4	0.0	0.0	11.7	NaN
1950-01-03	0.0	7.2	11.7	4.1	0.0	0.0	9.2	NaN
1950-01-04	0.0	5.8	9.2	2.3	0.0	0.0	5.8	NaN
1950-01-05	0.0	2.6	5.8	0.6	0.0	0.0	7.5	NaN
...	...	...	...	...	...	...	...	...
2022-12-26	0.0	6.3	11.4	1.8	3.6	0.0	12.3	13.130000
2022-12-27	0.0	6.8	12.3	2.0	2.7	0.0	11.9	12.900000
2022-12-28	0.0	6.5	11.9	2.8	1.4	0.0	13.7	12.820000
2022-12-29	0.0	8.1	13.7	2.9	2.0	0.0	11.8	12.566667
2022-12-30	0.0	6.6	11.8	2.8	1.9	0.0	10.5	12.273333

過去30日間の最高気温の平均値を示すmonth\_max列を追加

```
weather["month_day_max"] = weather["month_max"] / weather["temp_max"]
```

	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall	target	month_max	month_day_max
Date									
1950/1/1	16.7	3.5	4.7	2.2	0.0	0.0	8.3	NaN	NaN
1950/1/2	0.1	5.5	8.3	3.4	0.0	0.0	11.7	NaN	NaN
1950/1/3	0.0	7.2	11.7	4.1	0.0	0.0	9.2	NaN	NaN
1950/1/4	0.0	5.8	9.2	2.3	0.0	0.0	5.8	NaN	NaN
1950/1/5	0.0	2.6	5.8	0.6	0.0	0.0	7.5	NaN	NaN
...	...	...	...	...	...	...	...	...	...
2022/12/26	0.0	6.3	11.4	1.8	3.6	0.0	12.3	13.130000	1.151754
2022/12/27	0.0	6.8	12.3	2.0	2.7	0.0	11.9	12.900000	1.048780
2022/12/28	0.0	6.5	11.9	2.8	1.4	0.0	13.7	12.820000	1.077311
2022/12/29	0.0	8.1	13.7	2.9	2.0	0.0	11.8	12.566667	0.917275
2022/12/30	0.0	6.6	11.8	2.8	1.9	0.0	10.5	12.273333	1.040113

その日の最高気温が過去30日の平均最高気温と比べどの程度高いか表すmonth\_day\_maxを追加

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max"]
```

追加した変数

# モデルの改善

特徴量データ:

MAE:

```
predictors = ["precipitation", "temp_max", "temp_min"]
```

降水量

最高気温

最低気温

2.422452483454122

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max"]
```

+

平均最高気温と最高気温  
の差を示す値

2.399197746370611

-0.029

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max", "monthly_avg"]
```

+

該当月の最高気温  
の平均値

2.356300161342552

-0.043

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max", "monthly_avg",
```

+

"day\_of\_year\_avg"]

その日の最高気温の平均値

例: その日が1月1日であれば、過去の1月1日のデータから最高気温の平均値を出したもの

2.343558824556225

-0.012

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max", "monthly_avg", "day_of_year_avg",
```

+

"humidity\_avg"]

★ 平均湿度

2.3231266809229667

-0.021



モデルの保存と読み込み



**2.6**



# モデルの保存と読み込み

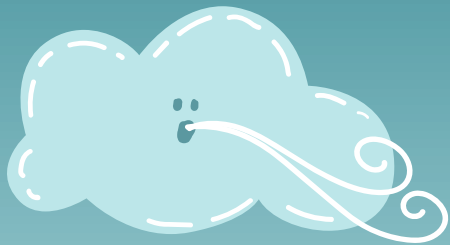
モデルの保存 :

```
import pickle
with open ('tenkiyosoku.pkl', 'wb') as f:
    pickle.dump(reg,f)
```

モデルの読み込み :

```
import pickle

with open('tenkiyosoku.pkl', mode='rb') as f:
    clf = pickle.load(f)
```



3

实践





# 今日の天気データ

※4月5日の天気予報を参考にしたデータ

定義	降水量 (mm)	最高気温 (°C)	最低気温 (°C)	4月平均最高気温 と最高気温の差 (°C)	4月の 最高平均気温(°C)	4月6日の 平均気温(°C)	平均湿度(%)
開発で使 いたラベル	precipitation	temp_max	temp_min	month_day_max	monthly_avg	day_of_year _avg	humidity_avg
4月6日の 情報	0	21	16	0.78	18.7	17.8	60

明日(4月7日)の最高気温は？

エラーについて：

Ridgeモデルが適合する際に使用された特徴量名が、予測の際に使用されたデータと一致しない場合に発生するもの。警告は問題ない場合でも表示されるため、無視しても構わない



# 4

開発の振り返りと参考資料について





# 開発の振り返り

## 開発で難しかったこと：

- ・ 分析モデルを決めること  
→適切なモデルだと思っけていても、モデルの評価結果がよくなかったり、扱っているデータが適切でない可能性がある

## 開発で工夫したこと：

- ・ 1つの進め方にこだわらず、分析モデルそのものを変更したり、使用するデータを変更したりと、色々試すようにした





## 参考文献



気象庁のホームページ

[https://www.data.jma.go.jp/obd/stats/etrn/view/daily\\_s1.php?prec\\_no=44&block\\_no=47662&year=2019&month=01&day=1&view=p](https://www.data.jma.go.jp/obd/stats/etrn/view/daily_s1.php?prec_no=44&block_no=47662&year=2019&month=01&day=1&view=p)  
(1950~2022年のデータを使用)

★



スクレイピング

<https://qiita.com/data-man/items/66f5fa8a7cacc529071c>



気象予想の機械学習

<https://www.youtube.com/watch?v=km95-NMT6IU>



スライドの引用先


<https://slidesgo.com/>



Pythonでデータ解析

[http://nfunao.web.fc2.com/files/python\\_intro\\_04.pdf](http://nfunao.web.fc2.com/files/python_intro_04.pdf)





ご清聴ありがとうございました

# データ収集

HTMLからデータを取得するためBeautifulSoupを使用し、スクレイピングを行う

```
import requests
from bs4 import BeautifulSoup
import csv
```

- requests : webページにHTTPリクエストを送信し、レスポンスを受信するために使用
- BeautifulSoup: HTMLなどを解析するために使用
- csv: csvファイルを読み書きするために使用

異なる年月のデータの取得はサイトのURLの記述を変更することで取得することができる

表示する年月を変更する場合：

サイト  
URL

[https://www.data.jma.go.jp/obd/stats/etrn/view/daily\\_s1.php?prec\\_no=44&block\\_no=47662&year=2019&month=01&day=1&view=p1](https://www.data.jma.go.jp/obd/stats/etrn/view/daily_s1.php?prec_no=44&block_no=47662&year=2019&month=01&day=1&view=p1)

[/daily\\_s1.php?prec\\_no=44&block\\_no=47662&year=2019&month=01&day=1&view=p1](#)

## 次にモデルの精度を上げるためにできること

- 実績データとテストデータを比べ、差が大きい場合の原因を調べる

```
combined.sort_values("diff", ascending=False).head()
```

	actual	predictions	diff
Date			
2022-10-05	15.0	24.216493	9.216493
2021-01-15	18.7	9.706619	8.993381
2021-08-14	20.2	29.158802	8.958802
2021-02-28	19.1	11.521126	7.578874
2022-03-13	24.1	16.527758	7.572242

?

- 使用するデータの地域を増やす
- 風速や積雪など他の特徴データを追加する
- 1950年以前のデータを追加する

# データの収集



URLのyear=とmonth=に年と月の数字を入れ替えながら、1950年~2022の1月~12月の引き出したいデータをCSVファイルに落とし込む

```
place_name = ["Tokyo"]

# URL で年と月ごとの設定ができるので%sで指定した英数字を埋め込めるようにする
base_url = "http://www.data.jma.go.jp/obd/stats/etrn/view/daily_s1.php?prec_no=44&block_no=47662&year=%s&month=%s&day=1&view=p1"

# 取ったデータをfloat型に変える
def str2float(str):
    try:
        return float(str)
    except:
        return 0.0

if __name__ == "__main__":
    # 都市を網羅
    for place in place_name:
        # 最終的にデータを集めるリスト
        All_list = [['Date', 'precipitation', 'temp_avg', 'temp_max', 'temp_min', 'windspeed', 'snowfall']]
```

```
# for文で該当期間抽出
for year in range(1950,2023):
    print(year)
    # その年の1月~12月の12回を網羅する
    for month in range(1,13):
        # 年と月を当てはめる
        r = requests.get(base_url%(year, month))
        r.encoding = r.apparent_encoding

        # サイトごとスクレイピング
        soup = BeautifulSoup(r.text)
```



# 機械モデルの評価

- 実績値と予測値の違いの可視化を行い、どれくらいの差があるか把握

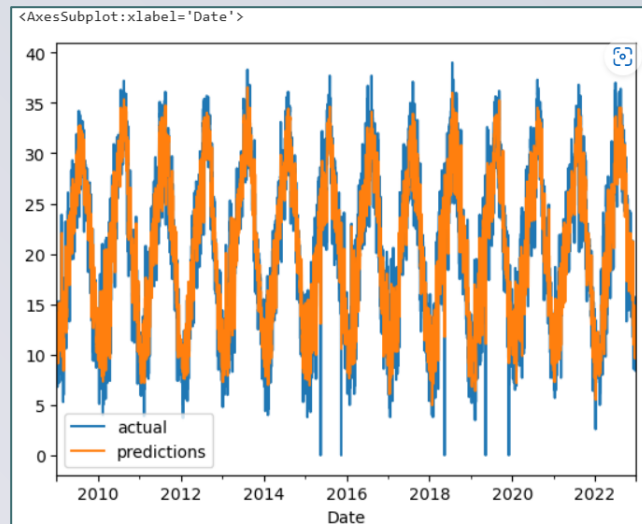
```
combined = pd.concat([test["target"], pd.Series(predictions, index=test.index)], axis=1)  
combined.columns = ["actual", "predictions"]
```

combined

	actual	predictions
Date		
2009-01-01	11.6	10.583273
2009-01-02	10.7	11.731744
2009-01-03	11.1	10.812795
2009-01-04	11.8	12.051978
2009-01-05	12.5	12.970583
...	...	...
2022-12-26	12.3	11.043348
2022-12-27	11.9	11.549465
2022-12-28	13.7	11.732259
2022-12-29	11.8	12.606882
2022-12-30	10.5	11.686217

5112 rows x 2 columns

combined.plot()



# データの収集



- 1950年から2023年の1月から12月のサイトを表示
- 表のうち、年月日、降水量、平均気温、最低気温、最高気温、平均風速、降雪量の列の値を取得
- 取得したデータをCSVファイルに保存し、「Tokyo」という名前のファイルで保存

```
# for文で該当期間抽出
for year in range(1950,2023):
    print(year)
    # その年の1月~12月の12回を網羅する
    for month in range(1,13):
        # 年と月を当てはめる
        r = requests.get(base_url%(year, month))
        r.encoding = r.apparent_encoding

    # サイトごとスクレイピング
    soup = BeautifulSoup(r.text)
    # findAllで条件に一致するものをすべて抜き出す
    # 今回の条件はtrタグでclassがmtxになっているもの
    rows = soup.findAll('tr',class_='mtx')

    # 表の最初の1~4行目はカラム情報なのでスライスする
    rows = rows[4:]
```

```
# 1日~最終日までの1行を取得
for row in rows:
    # trのなかのtdをすべて抜き出す
    data = row.findAll('td')

    # 1行の中には様々なデータがあるので全部取り出す
    rowData = [] #初期化
    rowData.append(str(year) + "/" + str(month) + "/" + str(data[0].string))
    rowData.append(str2float(data[3].string))
    rowData.append(str2float(data[6].string))
    rowData.append(str2float(data[7].string))
    rowData.append(str2float(data[8].string))
    rowData.append(str2float(data[11].string))
    rowData.append(str2float(data[17].string))

    # 次の行にデータを追加
    All_list.append(rowData)

# ファイルを生成
with open(place + '.csv', 'w',encoding="utf_8_sig") as file: #文字化け防止
    writer = csv.writer(file, lineterminator='\n')
    writer.writerows(All_list)
```

# WEATHER JOURNAL

```
missing = weather.isna().any()
print(missing[missing == True])
```

```
import numpy as np

infs = np.isinf(weather)
inf_rows = infs.any(axis=1)
print(inf_rows)
```

```
Series([], dtype: bool)
```

Date	
1950/1/31	False
1950/2/1	False
1950/2/2	False
1950/2/3	False
1950/2/4	False
...	
2022/12/26	False
2022/12/27	False
2022/12/28	False
2022/12/29	False
2022/12/30	False
Length: 26632, dtype: bool	

## エラー処理の例

```
: import numpy as np

# float64の最大値と最小値を取得
float64_max = np.finfo(np.float64).max
float64_min = np.finfo(np.float64).min

# データフレーム内の値が最大値と最小値の範囲外かどうかを検出
is_out_of_range = (weather > float64_max) | (weather < float64_min)

# 範囲外の値がある行と列を取得
rows_with_out_of_range = weather.index[is_out_of_range.any(axis=1)]
cols_with_out_of_range = weather.columns[is_out_of_range.any(axis=0)]

if len(rows_with_out_of_range) == 0 and len(cols_with_out_of_range) == 0:
    print("データフレームにfloat64の範囲外の値はありません。")
else:
    print("以下の行、列にfloat64の範囲外の値があります。")
    print("行:", rows_with_out_of_range)
    print("列:", cols_with_out_of_range)
```

以下の行、列にfloat64の範囲外の値があります。

```
行: DatetimeIndex(['1951-12-21', '1952-01-07', '1952-01-09', '1952-12-07',  
                  '1952-12-12', '1953-01-06', '1954-02-16', '1954-03-14',  
                  '1955-02-15', '1955-12-20', '1956-01-15', '1956-02-15',  
                  '1956-12-19', '1957-02-15', '1958-01-28', '1959-02-28',  
                  '1959-12-24', '1959-12-28', '1960-01-10', '1961-02-01',  
                  '1961-02-10', '1961-03-10', '1962-01-13', '1962-02-05',  
                  '1962-03-25', '1963-02-06', '1963-02-10', '1963-02-24',  
                  '1963-03-08', '1964-02-05', '1964-02-18', '1964-03-01',  
                  '1965-02-08', '1965-12-25', '1967-02-11', '1967-12-13',  
                  '1969-01-15', '1969-01-17', '1969-02-26', '1969-12-22',  
                  '1972-01-03', '1972-01-08', '1972-01-19', '1972-01-28',  
                  '1972-02-24', '1972-12-14', '1973-12-26', '1975-01-04',  
                  '1975-01-21', '1976-01-14', '1977-01-26', '1977-02-09',  
                  '1977-02-22', '1978-01-03', '1978-01-19', '1978-12-21',  
                  '1980-01-15', '1980-01-22', '1981-01-05', '1982-01-28',  
                  '1984-01-28', '1984-12-31', '1986-02-11', '1986-02-18',  
                  '1986-03-04', '1987-03-03', '1992-02-02', '1992-02-23',  
                  '1994-07-18', '1995-01-17', '1996-01-10', '1996-02-21',  
                  '1997-01-26', '1998-01-15', '2000-02-16', '2001-02-15',  
                  '2002-12-29', '2006-01-03', '2006-01-21', '2008-02-14',  
                  '2009-01-13', '2011-01-14', '2012-01-13', '2012-02-18',  
                  '2014-12-19', '2015-05-18', '2015-11-16', '2016-12-17',  
                  '2017-03-08', '2018-01-04', '2018-05-15', '2019-01-28',  
                  '2019-02-15', '2019-05-13', '2019-12-03', '2022-12-19'],  
                  dtype='datetime64[ns]', name='Date', freq=None)  
列: Index(['month_day_max', 'max_min'], dtype='object')
```

```
weather.loc['1951-12-21']
```

```
precipitation    0.000000  
temp_avg         7.300000  
temp_max        16.900000  
temp_min         0.000000  
windspeed        0.000000  
snowfall         0.000000  
target          10.500000  
month_max        14.496667  
month_day_max    0.857791  
max_min          inf  
Name: 1951-12-21 00:00:00, dtype: float64
```

```
weather["max_min"].value_counts()
```

```
2.000000    97  
inf          90  
1.250000    53  
1.500000    52  
1.333333    38  
..          ..  
-52.000000    1  
2.914286     1  
-5.111111    1  
-8.846154    1  
4.214286     1  
Name: max_min, Length: 13739, dtype: int64
```

```
inf_rows = weather[weather.isin([np.inf, -np.inf]).any(axis=1)]
print(inf_rows)
```




Date	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall \
1951-12-21	0.0	7.3	16.9	0.0	0.0	0.0
1952-01-07	0.0	3.8	9.5	0.0	0.0	0.0
1952-01-09	0.0	3.3	8.6	0.0	0.0	0.0
1952-12-07	0.0	6.0	12.6	0.0	0.0	0.0
1952-12-12	0.0	3.8	9.3	0.0	0.0	0.0
...	...	...	...	...	...	...
2019-01-28	0.0	6.5	13.2	0.0	2.6	0.0
2019-02-15	0.0	3.0	5.3	0.0	2.0	0.0
2019-05-13	0.0	17.7	0.0	13.3	0.0	0.0
2019-12-03	0.0	0.0	0.0	9.1	0.0	0.0
2022-12-19	0.0	4.9	10.1	0.0	2.0	0.0


Date	target	month_max	month_day_max	max_min
1951-12-21	10.5	14.496667	0.857791	inf
1952-01-07	8.2	12.656667	1.332281	inf
1952-01-09	6.1	12.250000	1.424419	inf
1952-12-07	11.9	15.123333	1.200265	inf
1952-12-12	13.1	13.336667	1.434050	inf
...	...	...	...	...
2019-01-28	9.9	10.253333	0.776768	inf
2019-02-15	13.9	10.283333	1.940252	inf
2019-05-13	19.3	21.383333	inf	0.0
2019-12-03	15.3	16.510000	inf	0.0
2022-12-19	10.9	14.683333	1.453795	inf

[96 rows x 10 columns]






```
inf_count = np.isinf(weather['month_day_max']).sum()
print("The number of inf values in 'month_day_max' column is:", inf_count)
```



```
The number of inf values in 'month_day_max' column is: 6
```

```
weather = weather[~np.isinf(weather['month_day_max'])].copy()
```



```
inf_count = np.isinf(weather['month_day_max']).sum()
print("The number of inf values in 'month_day_max' column is:", inf_count)
```

```
The number of inf values in 'month_day_max' column is: 0
```

# OUR TEAM

```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max"]
```

```
error, combined = create_predictions(predictors, weather, reg)
```

```
error
```

```
2.447137636660573
```



## 改善策②



```
weather["monthly_avg"] = weather["temp_max"].groupby(weather.index.month).apply(lambda x: x.expanding(1).mean())
```

Date	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall	target	month_max	month_day_max	monthly_avg
1950-03-02	0.0	6.0	14.4	-1.7	0.0	0.0	9.9	9.593333	0.666204	14.400000
1950-03-03	0.7	4.9	9.9	1.8	0.0	0.0	16.3	9.640000	0.973737	12.150000
1950-03-04	0.0	7.7	16.3	0.3	0.0	0.0	17.7	9.803333	0.601431	13.533333
1950-03-05	0.0	8.6	17.7	0.8	0.0	0.0	16.3	10.233333	0.578154	14.575000
1950-03-06	0.0	9.8	16.3	2.2	0.0	0.0	17.9	10.506667	0.644581	14.920000
...	...	...	...	...	...	...	...	...	...	...
2022-12-26	0.0	6.3	11.4	1.8	3.6	0.0	12.3	13.130000	1.151754	12.225742
2022-12-27	0.0	6.8	12.3	2.0	2.7	0.0	11.9	12.900000	1.048780	12.225775
2022-12-28	0.0	6.5	11.9	2.8	1.4	0.0	13.7	12.820000	1.077311	12.225631
2022-12-29	0.0	8.1	13.7	2.9	2.0	0.0	11.8	12.566667	0.917275	12.226283
2022-12-30	0.0	6.6	11.8	2.8	1.9	0.0	10.5	12.273333	1.040113	12.226095

## 改善策②



```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max", "monthly_avg"]
```

```
error, combined = create_predictions(predictors, weather, reg)
```

```
error
```

2.4378684928940575



## 改善策③

```
weather["day_of_year_avg"] = weather["temp_max"].groupby(weather.index.day_of_year).apply(lambda x: x.expanding(1).mean())
```

	precipitation	temp_avg	temp_max	temp_min	windspeed	snowfall	target	month_max	month_day_max	monthly_avg	day_of_year_avg
Date											
1950-03-02	0.0	6.0	14.4	-1.7	0.0	0.0	9.9	9.593333	0.666204	14.400000	14.400000
1950-03-03	0.7	4.9	9.9	1.8	0.0	0.0	16.3	9.640000	0.973737	12.150000	9.900000
1950-03-04	0.0	7.7	16.3	0.3	0.0	0.0	17.7	9.803333	0.601431	13.533333	16.300000
1950-03-05	0.0	8.6	17.7	0.8	0.0	0.0	16.3	10.233333	0.578154	14.575000	17.700000
1950-03-06	0.0	9.8	16.3	2.2	0.0	0.0	17.9	10.506667	0.644581	14.920000	16.300000
...	...	...	...	...	...	...	...	...	...	...	...
2022-12-26	0.0	6.3	11.4	1.8	3.6	0.0	12.3	13.130000	1.151754	12.225742	10.835616
2022-12-27	0.0	6.8	12.3	2.0	2.7	0.0	11.9	12.900000	1.048780	12.225775	11.064384
2022-12-28	0.0	6.5	11.9	2.8	1.4	0.0	13.7	12.820000	1.077311	12.225631	10.663014
2022-12-29	0.0	8.1	13.7	2.9	2.0	0.0	11.8	12.566667	0.917275	12.226283	10.719178
2022-12-30	0.0	6.6	11.8	2.8	1.9	0.0	10.5	12.273333	1.040113	12.226095	10.976712

# OUR TEAM



```
predictors = ["precipitation", "temp_max", "temp_min", "month_day_max", "monthly_avg", "day_of_year_avg"]
```


```
error, combined = create_predictions(predictors, weather, reg)
```

```
error
```

```
2.4345675101677244
```

```
reg.coef_
```

```
array([0.01323776, 0.3858106 , 0.21204349, 0.1737103 , 0.10150869,  
       0.28266144])
```



# OUR TEAM

```
weather.corr()["target"]
```

```
precipitation    0.081305
temp_avg         0.930376
temp_max         0.911751
temp_min         0.915462
windspeed        -0.034095
snowfall         -0.070541
target           1.000000
month_max        0.881085
month_day_max    -0.191018
max_min          -0.077584
monthly_avg      0.901146
day_of_year_avg  0.912965
Name: target, dtype: float64
```

```
combined["diff"]=(combined["actual"] - combined["predictions"]).abs()
```

```
combined.sort_values("diff", ascending=False).head()
```

	actual	predictions	diff
<b>2021-01-15</b>	18.7	9.613553	9.086447
<b>2021-08-14</b>	20.2	29.199112	8.999112
<b>2022-10-05</b>	15.0	23.764906	8.764906
<b>2022-03-13</b>	24.1	16.305107	7.794893
<b>2021-02-28</b>	19.1	11.417991	7.682009