線形単回帰のプログラミング

ノートブックの作成

- cs3-07フォルダの下に新規ノートブックを作成し、ファイル名を linear_regression-simple.ipynb に変更します。
- 次ページからの内容にしたがって、各セルを作成、Shift+Enter で実行してください。そして、解説の内容をよく読み、各セルのプログラムと結果それぞれの意味について理解してください。
- まず、ipynb ファイルの先頭に、このノートブックの簡単な説明を入れておきましょう。

(Markdown)

```
### Simple linear regression
```

(Markdown)
```
#### Import libraries
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

必要なライブラリをインポートし、
CSVファイルを読み込みます。

(Markdown)
```
#### Read CSV file
```

```
csv_in = 'batting_npb2021.csv'
df = pd.read_csv(csv_in, sep=',', skiprows=0, header=0)
print(df.shape)
print(df.info())
display(df.head())
```

Weight列を説明変数、HomeRun列を目的変数にして、線形単回帰の計算を行い、回帰直線を描画してみましょう。

(Markdown)

```
#### Separate explanatory variable(s) and objective variable
説明変数と目的変数を分ける
```

```
X = df[ ['Weight'] ]  # explanatory variable, 2D
y = df[ 'HomeRun' ]  # objective variable, 1D
print('X:', X.shape)
display(X.head())
print('y:', y.shape)
print(y.head())
```

説明変数: X
目的変数: y

ポイント: **Xは2次元、yは1次元**にする。

Xは61行1列の**2次元**
データ(データフレーム)

X: (61, 1)

	Weight
0	85
1	85
2	104
3	87
4	85

yは61要素の**1次元**
データ(シリーズ)

```
y: (61,)
0      21
1      11
2      32
3      28
4      11
```

- 今回、説明変数Xは Weightの1列だけですが、

  X = df['Weight']

  と1次元のシリーズとして取り出すのではなく、

  X = df[['Weight']]

  と2次元のデータフレームとして取り出します。

- これは、あとで回帰の計算のために用いるOLS関数において、説明変数用の引数として2次元データを代入する必要があるためです。
  - 線形重回帰の場合は複数列なので2次元データになることから、線形単回帰の場合もデータ型をそれに合わせて2次元データとして与えます。

(Markdown)

```
#### MLR calculation
線形単回帰分析
```

```
X_c = sm.add_constant(X)
display(X.head())
display(X_c.head())
```

X

	Weight
0	85
1	85
2	104
3	87
4	85

X_c

	const	Weight
0	1.0	85
1	1.0	85
2	1.0	104
3	1.0	87
4	1.0	85

ポイント:
add_constant() で、説明変数 X に定数項 (a_0) 用の「1」だけの列を付加します(X_c)。

$$y = a_0 * 1 + a_1 * \mathbf{Weight}$$

という形式にするため。


```
model = sm.OLS(y, X_c)
results = model.fit()
print(results.summary())
```

モデルオブジェクト = sm.OLS(目的変数, 説明変数) でデータをモデルに与え、
結果オブジェクト = **モデルオブジェクト**.fit() で回帰の計算を実行する。
得られた偏回帰係数や決定係数などは、結果オブジェクトの中に格納され、
結果オブジェクト.summary() で概要を表示できる。

OLS()には、**目的変数(y)、説明変数(X_c)の順**で引数を与えることに注意。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                HomeRun   R-squared:                       0.315
Model:                            OLS   Adj. R-squared:                  0.304
Method:                 Least Squares   F-statistic:                     27.18
Date:                Tue, 10 May 2022   Prob (F-statistic):           2.49e-06
Time:                        16:16:57   Log-Likelihood:                -214.81
No. Observations:                  61   AIC:                             433.6
Df Residuals:                      59   BIC:                             437.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -32.2835      9.003     -3.586      0.001     -50.299     -14.268
Weight         0.5505      0.106      5.214      0.000       0.339       0.762
==============================================================================
Omnibus:                        6.163   Durbin-Watson:                   1.579
Prob(Omnibus):                  0.046   Jarque-Bera (JB):                5.551
Skew:                           0.729   Prob(JB):                       0.0623
Kurtosis:                       3.237   Cond. No.                         720.
==============================================================================
```

決定係数
自由度調整済み決定係数

偏回帰係数

(Markdown)

```
#### Check R2 and Adjusted R2
決定係数や自由度調整済み決定係数
```

```
print('R2:', results.rsquared)
print('Adj R2:', results.rsquared_adj)
```

```
R2: 0.3154251578093461
Adj R2: 0.3038221943823859
```

- 結果オブジェクト(results)のrsquared属性やrsquared_adj属性に、決定係数(R-squared, R2)や自由度調整済み決定係数(Adj. R-squared, Adj R2) の値が格納されている。
- 決定係数(0.3154…) の値は、たしかに correlation.ipynb で求めた Weight と HomeRun の間のピアソン相関係数(0.561627…) の2乗と一致している。
- 非常によく当てはまっているとはいえないが、一定の当てはまりはみられるといえる。

(Markdown)

```
#### Partial regression coefficient
偏回帰係数
```

```
print(results.params)
```

```
const     -32.283543
Weight      0.550493
dtype: float64
```

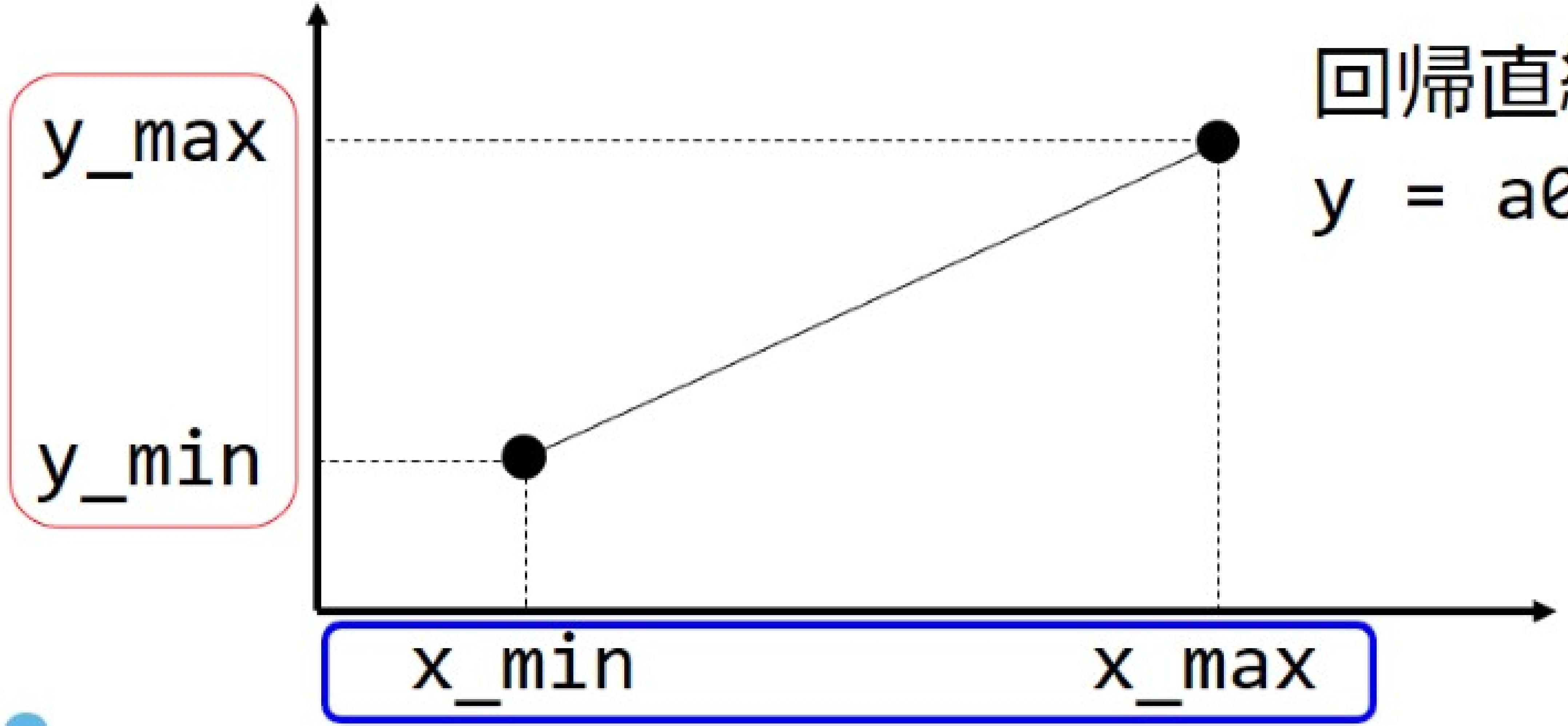
- 結果オブジェクト(results)のparams属性(データ型はSeries)に、偏回帰係数の値が格納されている。
- HomeRun = -32.28… + 0.5505… * Weight
  で、HomeRunとWeightの関係がモデル化できたといえる。
  つまり、体重が 1 kg 増えると、ホームラン数は 0.55本 増える。

(Markdown)

```
#### Plot of regression line
回帰直線の描画
```

```
a0 = results.params['const']
a1 = results.params['Weight']
x_min = X['Weight'].min()
x_max = X['Weight'].max()
x_min_max = np.array([x_min, x_max])
y_min_max = a0 + a1 * x_min_max
```

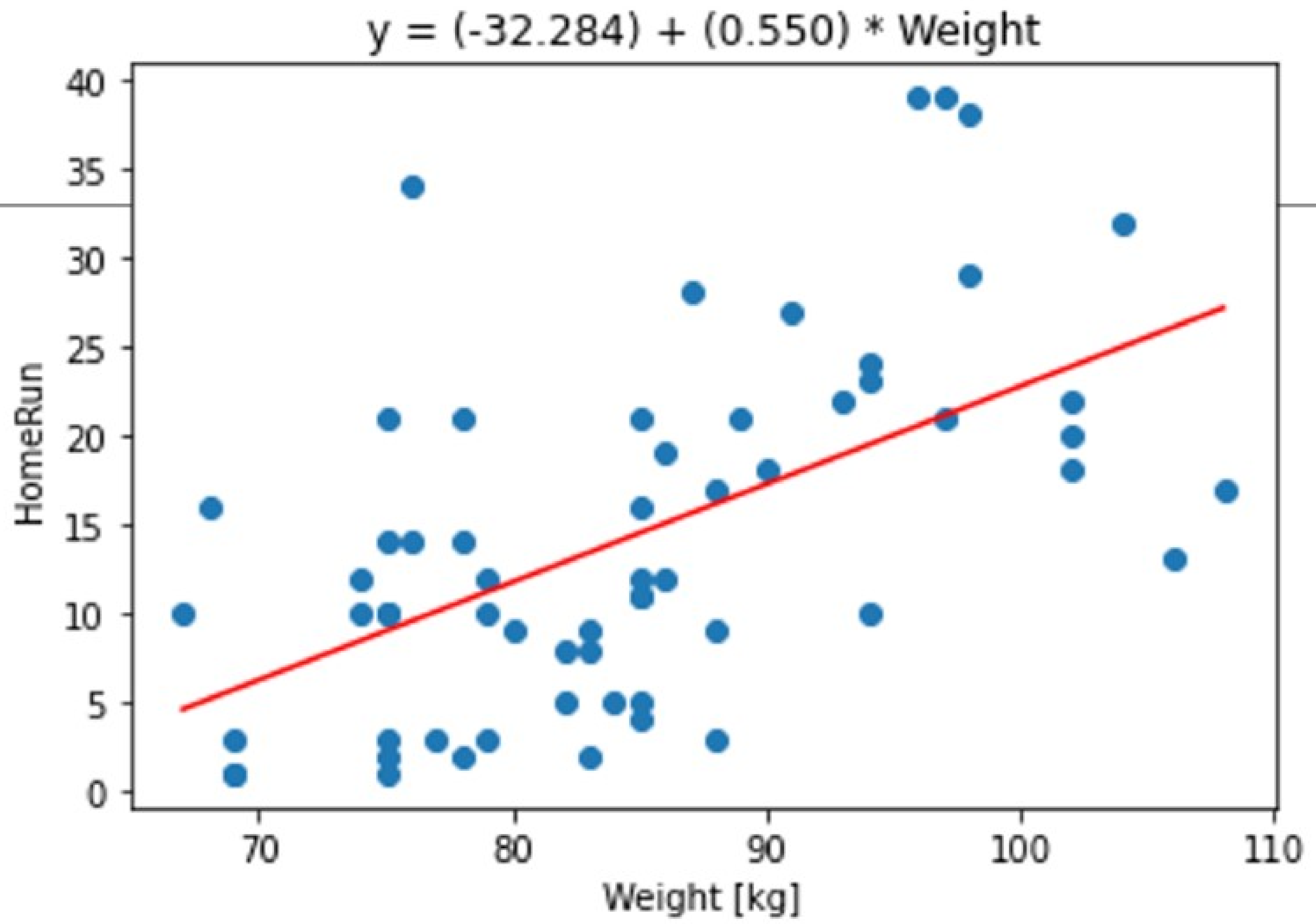
y_min と y_max をまとめて y_min_max とし、
y_min_max = a0 + a1 * x_min_max
で2つの値を一度に求めている。



回帰直線
y = a0 + a1 * x

x_min_max = [x_min, x_max]
y_min_max = [y_min, y_max]

```
plt.title('y = ({:.3f}) + ({:.3f}) * Weight'.format(a0, a1))
plt.scatter(X['Weight'], y)
plt.plot(x_min_max, y_min_max, c='red')
plt.xlabel('Weight [kg]')
plt.ylabel('HomeRun')
plt.show()
```



(Markdown)

```
#### Do prediction with obtained model
得られたモデルを用いて、予測を行う。
```

```
x_test = np.array([ 70.0, 85.0, 100.0 ])
y_test = a0 + a1 * x_test
print(y_test)
```

体重 70.0 kg, 85.0 kg, 100.0 kg の各選手は、それぞれ何本くらいホームランを打つと予想されるか？

```
[ 6.25096402 14.50835835 22.76575267]
```

3つの x に対する y を一度に求めた。

およそ 6.3本、14.5本、22.8本。

```
x_test_c = sm.add_constant(x_test, has_constant='add')
y_test = results.predict(x_test_c)
print(y_test)
```

```
[ 6.25096402 14.50835835 22.76575267]
```

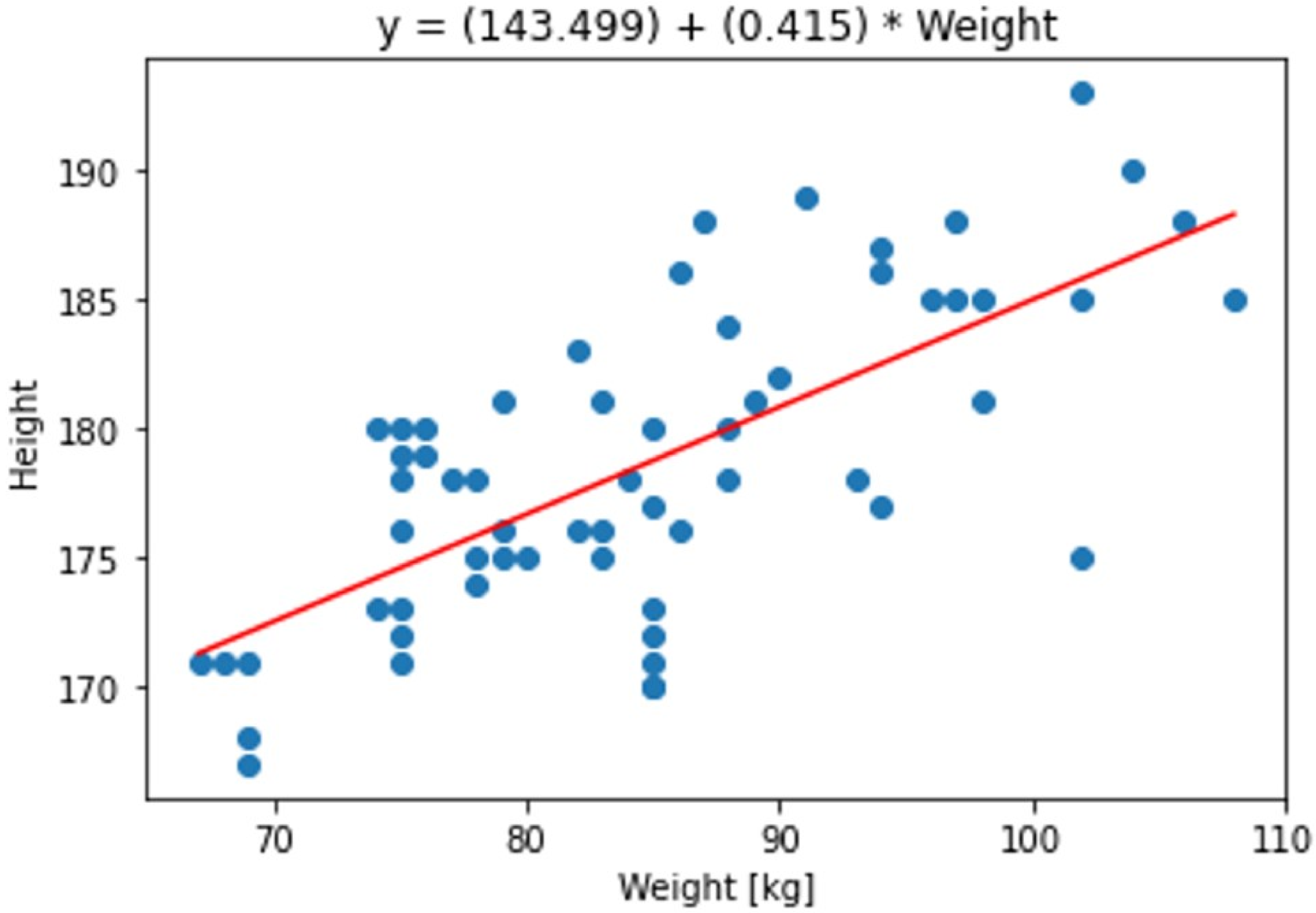
has_constant='add' オプションを付けた add_constant() で、 x_test に定数項用の「1」だけの列を追加した x_test_c を作成し、predict() に代入してもよい。

(Markdown) `#### Other columns`

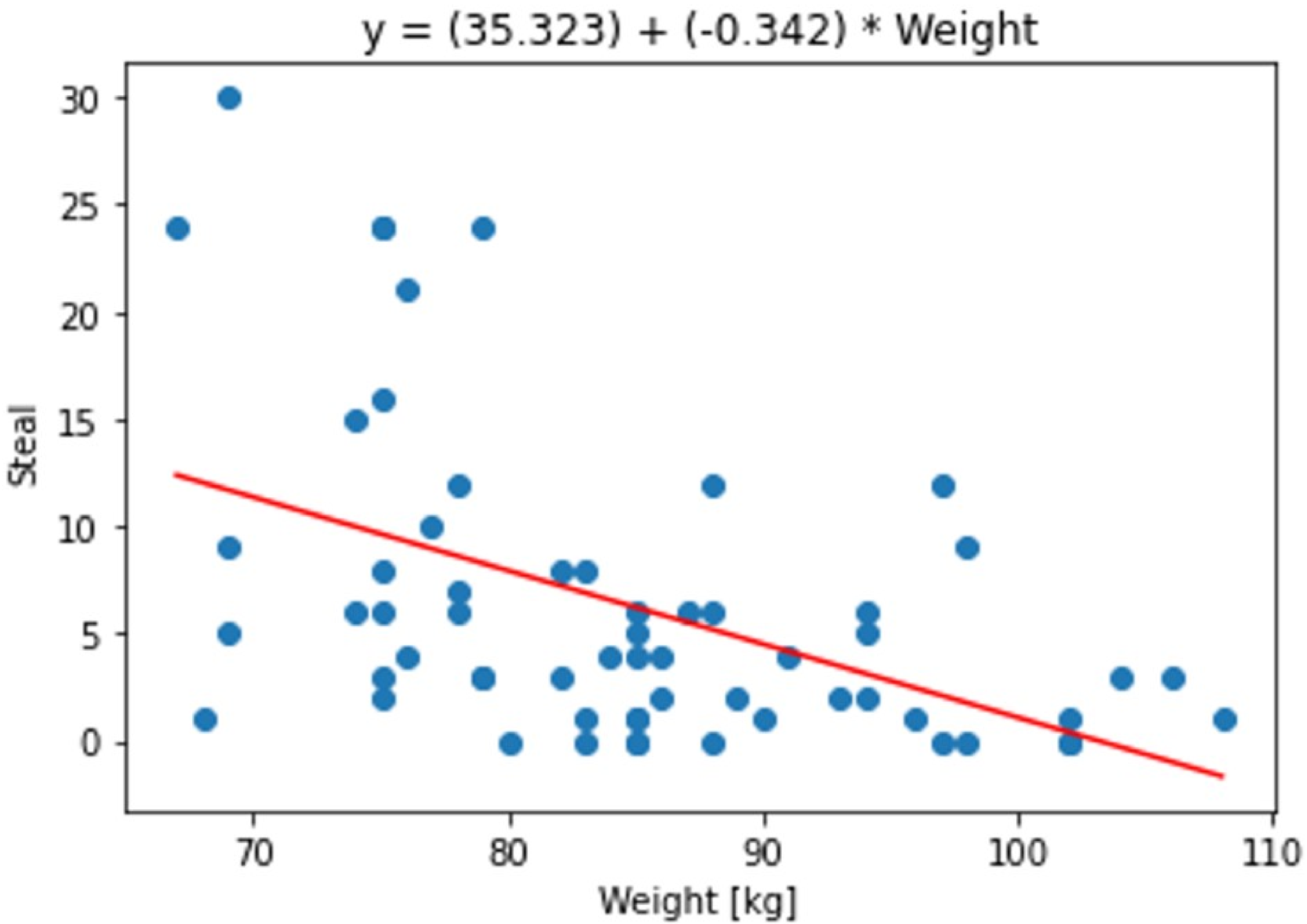
```
for c in ['Height', 'Steal', 'StrikeOut']:
    print(c, 'vs Weight')
    X = df[['Weight']]  # explanatory variable, 2D
    y = df[c]  # objective variable, 1D
    X_c = sm.add_constant(X)
    model = sm.OLS(y, X_c)
    results = model.fit()
    print('R2:', results.rsquared)
    print('Adj R2:', results.rsquared_adj)
    print(results.params)

    a0 = results.params['const']
    a1 = results.params['Weight']
    y_min_max = a0 + a1 * x_min_max

    plt.title('y = ({:.3f}) + ({:.3f}) * Weight'.format(a0, a1))
    plt.scatter(X['Weight'], y)
    plt.plot(x_min_max, y_min_max, c='red')
    plt.xlabel('Weight [kg]')
    plt.ylabel(c)
    plt.show()
```

これまでのプログラムを 1つのfor文のbodyにして、Height, Steal, StrikeOut の Weightとの線形単回帰の計算と回帰直線の描画を行う。

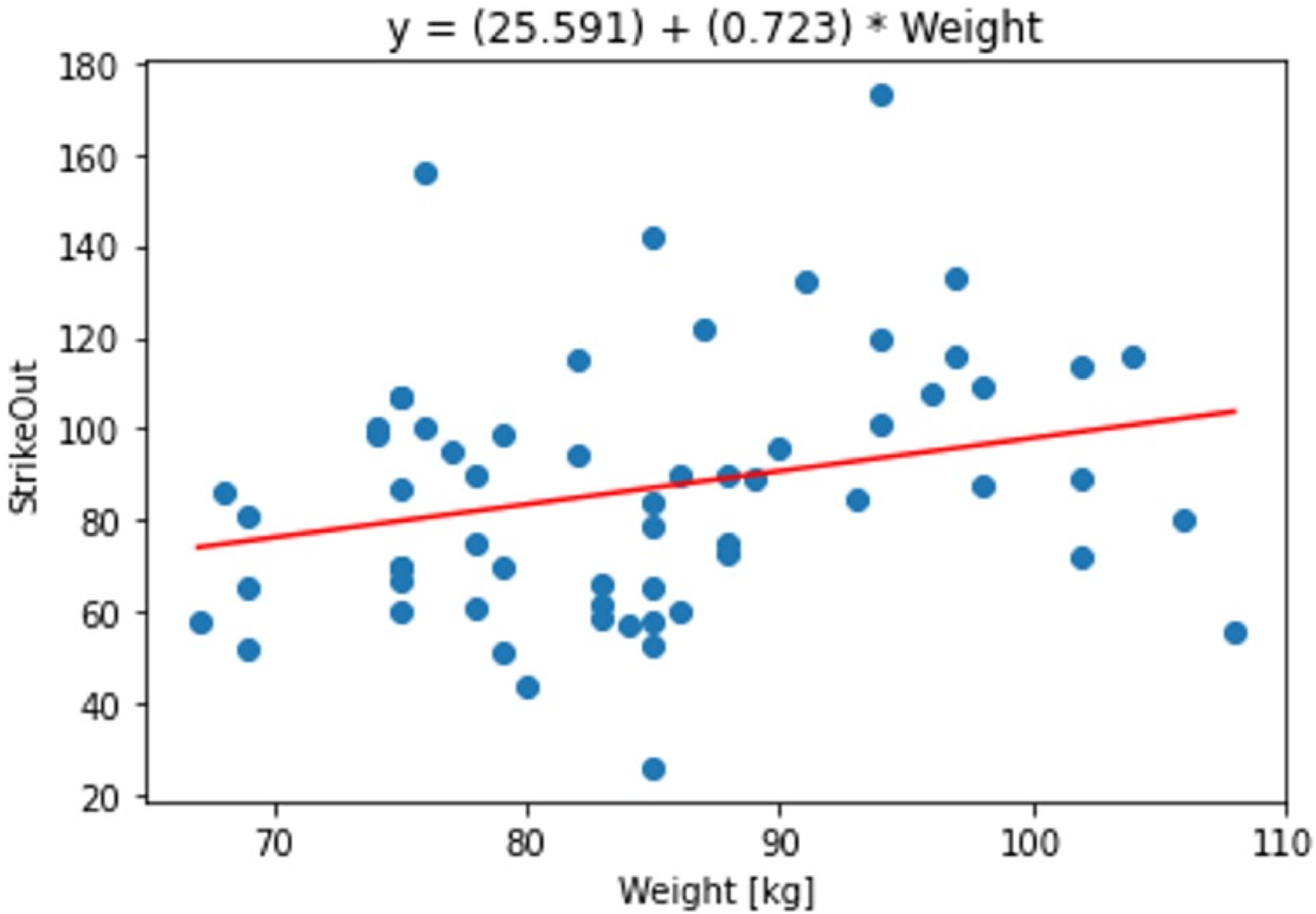
```
Height vs Weight
R2: 0.49004217569943875
Adj R2: 0.48139882274519197
const      143.498650
Weight       0.414634
dtype: float64
```



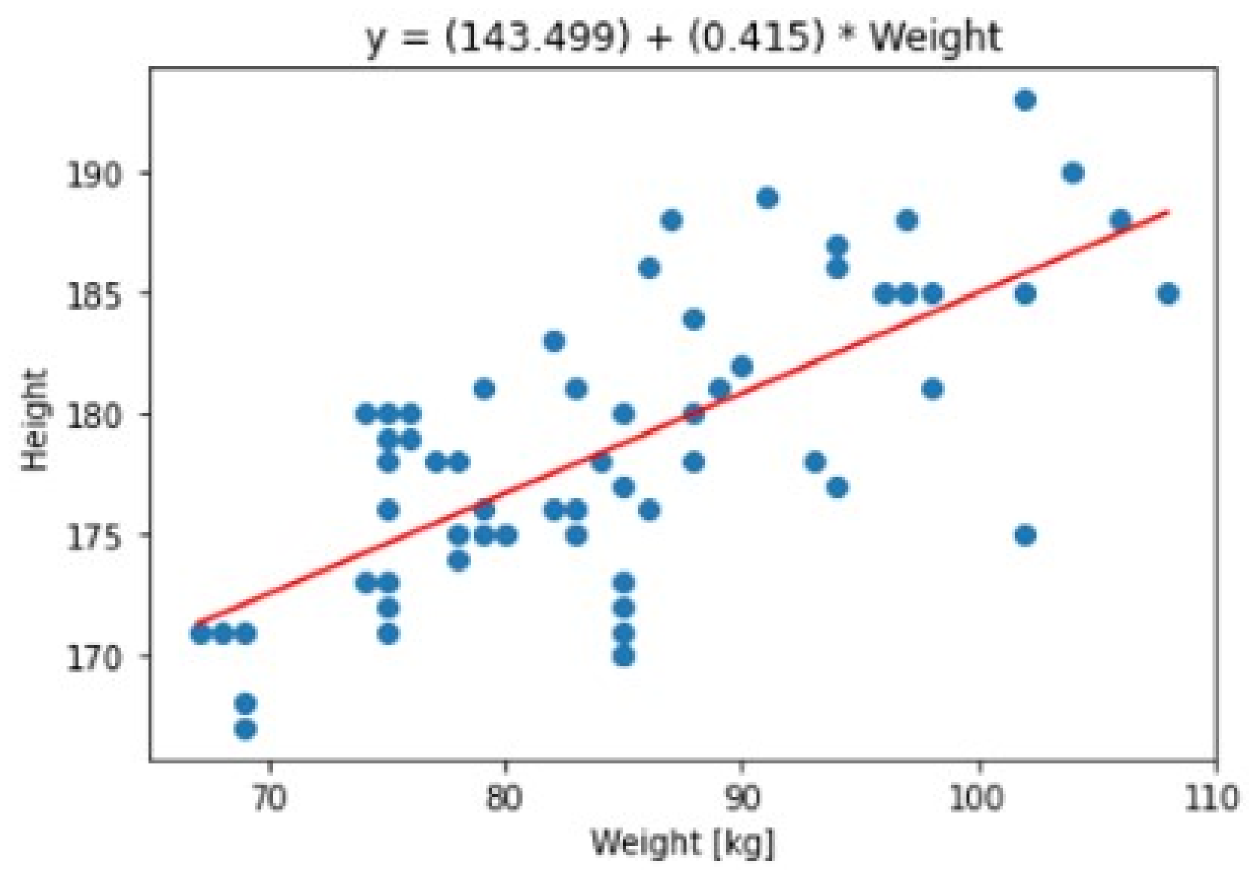
```
Steal vs Weight
R2: 0.23455717073053328
Adj R2: 0.2215835634547797
const      35.323354
Weight     -0.342444
dtype: float64
```



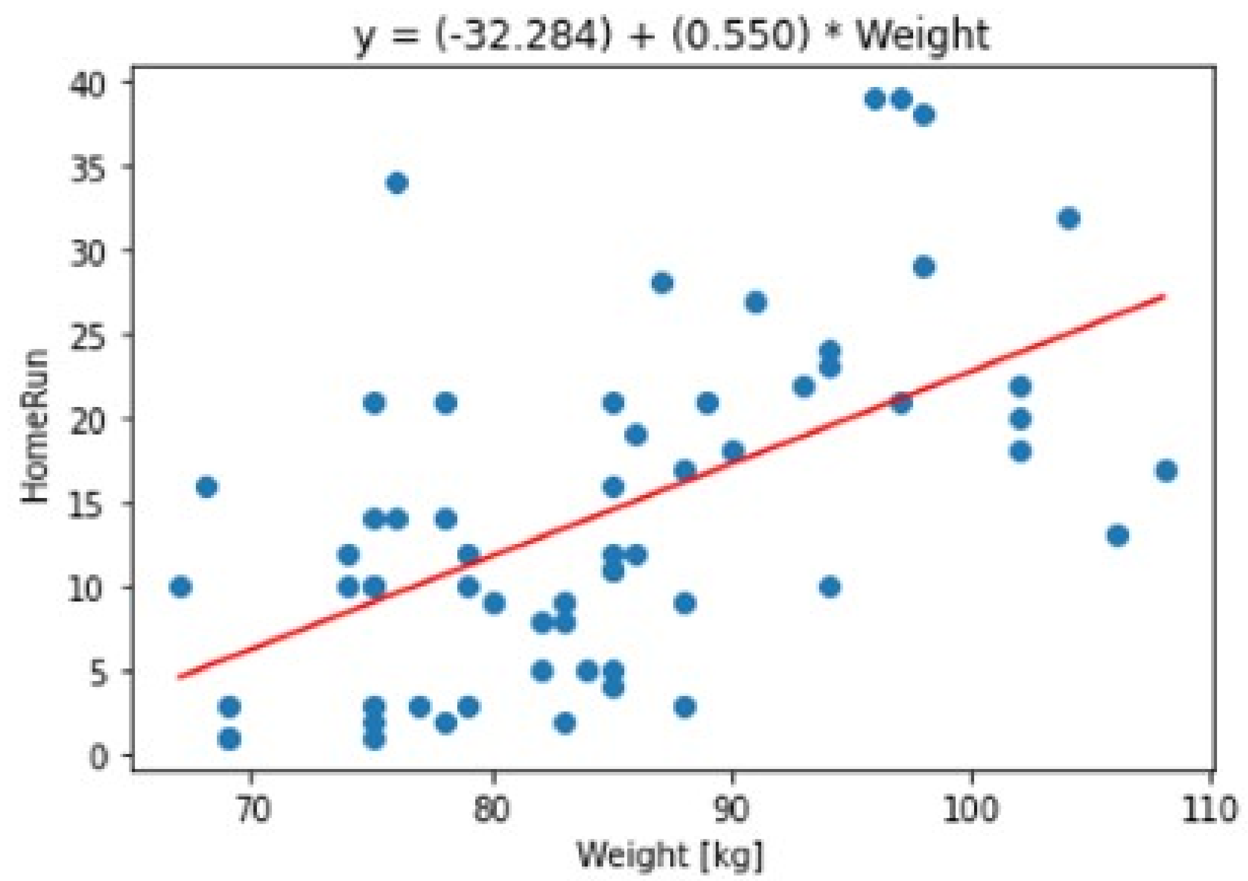
```
StrikeOut vs Weight
R2: 0.06847617985497023
Adj R2: 0.05268764053047825
const      25.591464
Weight      0.723315
dtype: float64
```



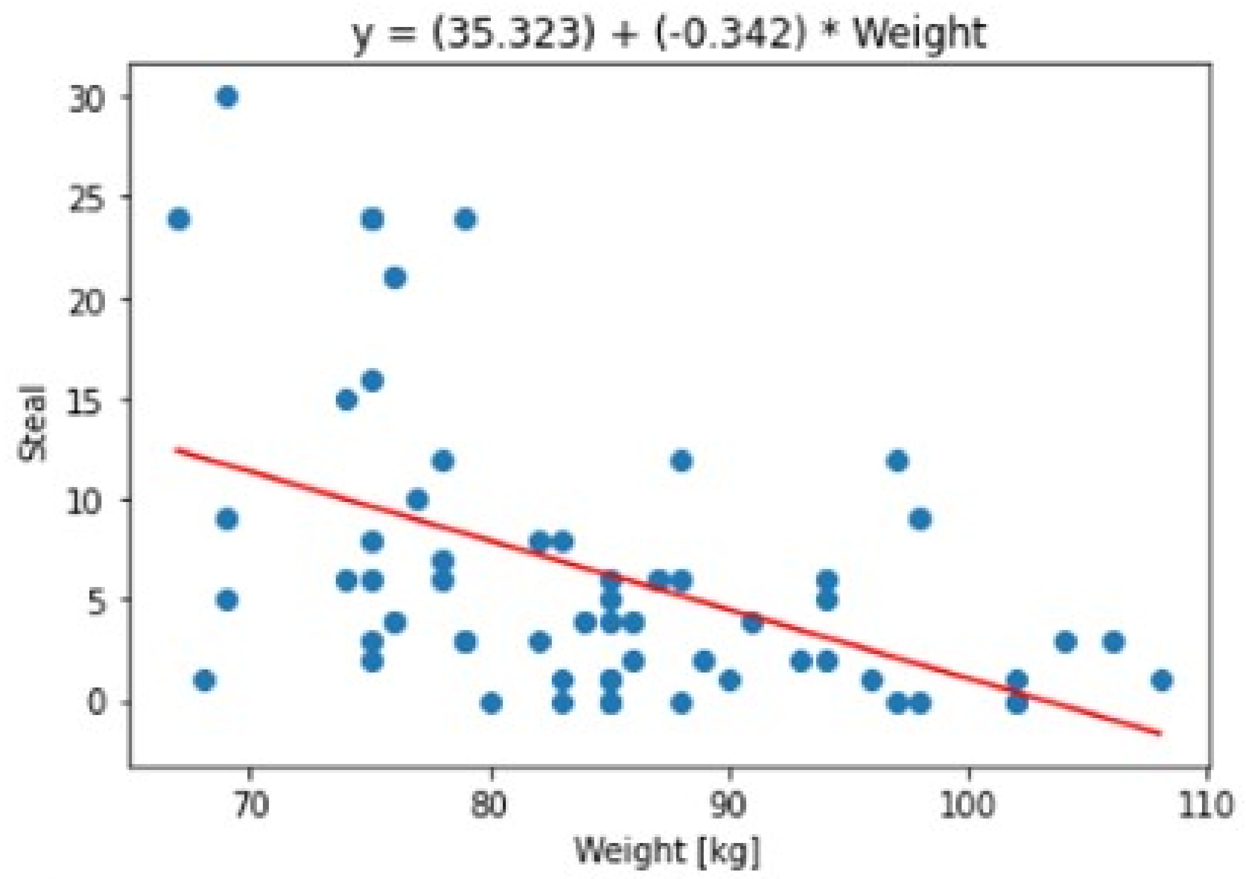
身長: 相関係数0.70, 決定係数0.49



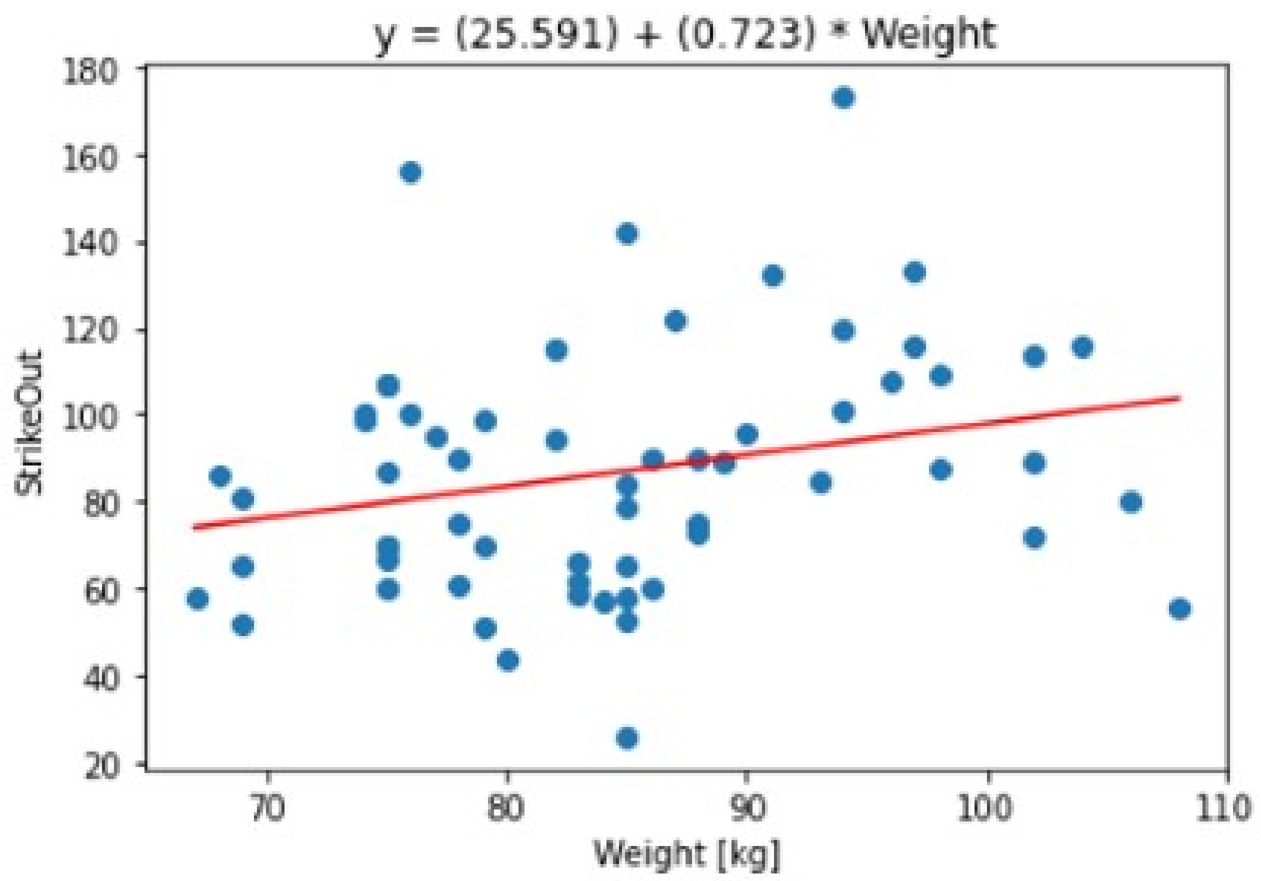
ホームラン: 相関係数0.56, 決定係数0.32



盗塁: 相関係数-0.48, 決定係数0.23



三振: 相関係数0.26, 決定係数0.07



説明変数(X)と目的変数(Y)の間の関係性が弱くても、回帰直線は必ず引けてしまう。

いつも、散布図＋回帰直線と相関係数、決定係数で、求めた回帰直線のデータへの当てはまりの良さを確認することが必要。