R^2 is one of the most important metric to find the goodness of fit of a linear regression model

We have formulas to find R^2 & sklearn function to compute R^2. But all these can be used only after fitting a linear model on the dataset & obtaining predicted values of y. But there a cool way to find R^2 without building any model or making any predictions

Consider the following dataset. This is a synthetic dataset. But the phenomena that we're going to witness now holds true regardless of the choice of dataset.

```
df
```

target

	x1	x2	x3	x4	y
0	0.550930	-9.454285	6.877938	-1.511569	-1.252146
1	0.527074	-13.576427	6.835688	-0.008219	-6.221816
2	0.369241	-4.308927	3.194860	-5.950549	28.713653
3	0.071082	-16.544772	1.330686	-26.072143	638.541496
4	0.623815	-17.055896	4.763626	-16.086181	231.010593
...	...	...	...	...	...
95	0.093623	-15.511775	1.570773	-23.715717	524.872117
96	0.524613	-6.894327	3.352488	-14.983740	206.511490
97	0.771286	-2.999534	5.478539	-26.295293	668.397423
98	0.333856	-9.163442	4.602109	-15.970542	234.860202
99	0.133913	-2.821084	4.912822	-6.971142	43.851334

But R^2 is hidden within this correlation matrix

```
df.corr()
```

this inbuilt pandas method gives the correlation matrix. This is a basic step we do to capture obvious cases of multicollinearity.

	x1	x2	x3	x4	y
x1	1.000000	0.037797	0.086736	-0.097399	0.120096
x2	0.037797	1.000000	-0.005664	0.051457	-0.029573
x3	0.086736	-0.005664	1.000000	0.040384	0.026987
x4	-0.097399	0.051457	0.040384	1.000000	-0.968436
y	0.120096	-0.029573	0.026987	-0.968436	1.000000

this is usually denoted by R_{xx}

this is obviously c^T

this is usually denoted by c

Now consider the matrix product $c^T R_{xx}^{-1} c$ which we're going to compute using numpy's linear algebra functions as follows

Here we're trying to create a linear model to fit and predict on the same data & eventually obtain R^2

```
c=df.corr()[['y']][:-1].values
c
```

```
array([[ 0.12009648],
       [-0.02957306],
       [ 0.02698693],
       [-0.96843629]])
```

```
R_xx=X.corr().values
R_xx_inv=np.linalg.inv(R_xx)
R_xx_inv
```

Mind the inverse

```
array([[ 1.02001209, -0.04450372, -0.09298011,  0.10539336],
       [-0.04450372,  1.00465694,  0.01183274, -0.05650909],
       [-0.09298011,  0.01183274,  1.0101695 , -0.05045958],
       [ 0.10539336, -0.05650909, -0.05045958,  1.01521077]])
```

```
ct_R_xx_inv_c=np.dot(np.dot(c.transpose(),R_xx_inv),c)
ct_R_xx_inv_c
```

```
array([[0.94304025]])
```

This is where we're computing $c^T R_{xx}^{-1} c$ which is equal to

```
X=df.drop('y',axis=1)
y=df.y
```

```
from sklearn.linear_model import LinearRegression
```

```
line=LinearRegression()
```

```
line.fit(X,y)
```

```
LinearRegression()
```

```
y_pred=line.predict(X)
```

```
from sklearn.metrics import r2_score
```

```
r_sq=r2_score(y,y_pred)
```

```
r_sq
```

```
0.9430402485887793
```

They both are exactly same. There appears to be a small difference due to truncation.

What we observed is not a coincidence, because $\boxed{c^T R_{xx}^{-1} c = R^2}$ is a theorem.

LHS is what we obtain from the raw data

RHS is what we obtain after building & training a linear model

There are 2 limitations. This will not hold good in the presence of multicollinearity & it will give us the R^2 of the training set only