Variance Inflation Factor in Python

Asked 4 years, 3 months ago Active 8 months ago Viewed 65k times

41

27

I'm trying to calculate the variance inflation factor (VIF) for each column in a simple dataset in python:

```
a b c d
1 2 4 4
1 2 6 3
2 3 7 4
3 2 8 5
4 1 9 4
```

I have already done this in R using the vif function from the usdm library which gives the following results:

```
a <- c(1, 1, 2, 3, 4)
b <- c(2, 2, 3, 2, 1)
c <- c(4, 6, 7, 8, 9)
d <- c(4, 3, 4, 5, 4)

df <- data.frame(a, b, c, d)
vif_df <- vif(df)
print(vif_df)

Variables   VIF
  a       22.95
  b       3.00
  c       12.95
  d       3.00
```

However, when I do the same in python using the statsmodel vif function, my results are:

```
a = [1, 1, 2, 3, 4]
b = [2, 2, 3, 2, 1]
c = [4, 6, 7, 8, 9]
d = [4, 3, 4, 5, 4]

ck = np.column_stack([a, b, c, d])

vif = [variance_inflation_factor(ck, i) for i in range(ck.shape[1])]
print(vif)

Variables   VIF
  a       47.136986301369774
  b       28.931506849315081
  c       80.31506849315096
  d       40.438356164383549
```

The results are vastly different, even though the inputs are the same. In general, results from the statsmodel VIF function seem to be wrong, but I'm not sure if this is because of the way I am calling it or if it is an

issue with the function itself.

I was hoping someone could help me figure out whether I was incorrectly calling the statsmodel function or explain the discrepancies in the results. If it's an issue with the function then are there any VIF alternatives in python?

python | r | numpy | statistics | statsmodels

Share Improve this question Follow

asked Mar 7 '17 at 21:09

Nizag
689 1 7 15

8 Answers

Active | Oldest | Votes

31

I believe the reason for this is due to a difference in Python's OLS. OLS, which is used in the python variance inflation factor calculation, does not add an intercept by default. You definitely want an intercept in there however.

What you'd want to do is add one more column to your matrix, ck, filled with ones to represent a constant. This will be the intercept term of the equation. Once this is done, your values should match out properly.

Edited: replaced zeroes with ones

Share Improve this answer Follow

edited Aug 27 '20 at 20:47

HoldOffHunger
11.3k 6 54 102

answered Mar 20 '17 at 18:56

Drverzal
326 3 3

---

subtracting the mean from all variables would be similar. – Josef Mar 20 '17 at 20:42

3 typo: column for constant should be filled with ones (not zeros). – Josef Mar 20 '17 at 20:43

Good call on my typo. Edited my original post with the fix. – Drverzal Mar 20 '17 at 21:04

That makes sense. Adding a column of 1s did the trick. Thanks! – Nizag Mar 21 '17 at 20:24

46

As mentioned by others and in this post by Josef Perktold, the function's author, `variance_inflation_factor` expects the presence of a constant in the matrix of explanatory variables. One can use `add_constant` from statsmodels to add the required constant to the dataframe before passing its values to the function.

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

df = pd.DataFrame(
    {'a': [1, 1, 2, 3, 4],
```

```
    'b': [2, 2, 3, 2, 1],
    'c': [4, 6, 7, 8, 9],
    'd': [4, 3, 4, 5, 4]}
)

X = add_constant(df)
>>> pd.Series([variance_inflation_factor(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns)
const    136.875
a         22.950
b          3.000
c         12.950
d          3.000
dtype: float64
```

I believe you could also add the constant to the right most column of the dataframe using `assign` :

```
X = df.assign(const=1)
>>> pd.Series([variance_inflation_factor(X.values, i)
               for i in range(X.shape[1])],
              index=X.columns)
a         22.950
b          3.000
c         12.950
d          3.000
const    136.875
dtype: float64
```

The source code itself is rather concise:

```
def variance_inflation_factor(exog, exog_idx):
    """
    exog : ndarray, (nobs, k_vars)
        design matrix with all explanatory variables, as for example used in
        regression
    exog_idx : int
        index of the exogenous variable in the columns of exog
    """
    k_vars = exog.shape[1]
    x_i = exog[:, exog_idx]
    mask = np.arange(k_vars) != exog_idx
    x_noti = exog[:, mask]
    r_squared_i = OLS(x_i, x_noti).fit().rsquared
    vif = 1. / (1. - r_squared_i)
    return vif
```

It is also rather simple to modify the code to return all of the VIFs as a series:

```
from statsmodels.regression.linear_model import OLS
from statsmodels.tools.tools import add_constant

def variance_inflation_factors(exog_df):
    '''
    Parameters
```

```
    ----------
    exog_df : dataframe, (nobs, k_vars)
       design matrix with all explanatory variables, as for example used in
       regression.

    Returns
    -------
    vif : Series
       variance inflation factors
    '''
    exog_df = add_constant(exog_df)
    vifs = pd.Series(
       [1 / (1. - OLS(exog_df[col].values,
                      exog_df.loc[:, exog_df.columns != col].values).fit().rsquared)
        for col in exog_df],
       index=exog_df.columns,
       name='VIF'
    )
    return vifs
```

```
>>> variance_inflation_factors(df)
const    136.875
a         22.950
b          3.000
c         12.950
Name: VIF, dtype: float64
```

Per the solution of @T_T, one can also simply do the following:

```
vifs = pd.Series(np.linalg.inv(df.corr().to_numpy()).diagonal(),
                 index=df.columns,
                 name='VIF')
```

Share Improve this answer Follow

edited Oct 18 '20 at 17:11

answered Feb 16 '18 at 2:54

Alexander
88.4k 24 165 171

---

1 I think it is safe to add `X = add_constant(df.dropna())` in case of missing values. – steven Feb 11 '19 at 14:51

1 Thanks for this solution. I was extremely perplexed as to why I was getting such high VIF for my model's independent variables, which is how I ended up on this post. Much as I hate to do this, I'm almost tempted to complete my analysis in R. – horcle_buzz Sep 21 '20 at 14:53