

# How can I perform Principal Components Regression in Python using a step-by-step approach?

Authored by  
**stats writer**

April 22, 2024

## RECOMMENDED CITATION

stats writer (2024). *How can I perform Principal Components Regression in Python using a step-by-step approach?*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=138010>

Principal Components Regression (PCR) is a statistical method that combines the features of Principal Component Analysis (PCA) and linear regression to improve the accuracy of regression models. It is commonly used in data analysis and machine learning to reduce the dimensionality of a large dataset while retaining important information.

To perform PCR in Python, a step-by-step approach can be followed. Firstly, the dataset should be preprocessed by handling missing values, scaling the data, and splitting it into training and testing sets. Next, the PCA algorithm can be applied to the training set to identify the principal components. Then, the number of principal components to be used in the regression model can be determined through cross-validation.

Once the number of components is finalized, the regression model can be built using the selected components from the PCA. This can be done by fitting a linear regression model on the transformed dataset. Finally, the model can be evaluated using the testing set and the results can be interpreted to make predictions.

In summary, to perform PCR in Python, the dataset should be preprocessed, PCA should be applied, the number of components should be determined, and a regression model should be built and evaluated. This step-by-step approach can help in effectively using PCR for data analysis and prediction tasks.

## Principal Components Regression in Python (Step-by-Step)

**Given a set of  $p$  predictor variables and a response variable, multiple linear regression uses a method known as least squares to minimize the sum of squared residuals (RSS):**

$$RSS = \sum (y_i - \hat{y}_i)^2$$

**where:**

$\Sigma$ : A greek symbol that means *sumyi*: The actual response value for the *i*th observation?  
 $\hat{y}_i$ : The predicted response value based on the multiple linear regression model

However, when the predictor variables are highly correlated then multicollinearity can become a problem. This can cause the coefficient estimates of the model to be unreliable and have high variance.

One way to avoid this problem is to instead use principal components regression, which finds *M* linear combinations (known as "principal components") of the original *p* predictors and then uses least squares to fit a linear regression model using the principal components as predictors.

This tutorial provides a step-by-step example of how to perform principal components regression in Python.

Step 1: Import Necessary Packages

First, we'll import the necessary packages to perform principal components regression (PCR) in Python:

```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

### Step 2: Load the Data

For this example, we'll use a dataset called mtcars, which contains information about 33 different cars. We'll use hp as the response variable and the following variables as the predictors:

```
mpg disp drat wt qsec
```

The following code shows how to load and view this dataset:

```
#define URL where data is located
url =
"https://raw.githubusercontent.com/Statology/Python-G
```

```
uides/main/mtcars.csv"
```

```
#read in data
```

```
data_full = pd.read_csv(url)
```

```
#select subset of data
```

```
data = data_full]
```

```
#view first six rows of data
```

```
data
```

```
mpg disp drat wt qsec hp
```

```
0 21.0 160.0 3.90 2.620 16.46 110
```

```
1 21.0 160.0 3.90 2.875 17.02 110
```

```
2 22.8 108.0 3.85 2.320 18.61 93
```

```
3 21.4 258.0 3.08 3.215 19.44 110
```

```
4 18.7 360.0 3.15 3.440 17.02 175
```

```
5 18.1 225.0 2.76 3.460 20.22 105
```

Step 3: Fit the PCR Model

The following code shows how to fit the PCR model to this data. Note the following:

`pca.fit_transform(scale(X))`: This tells Python that each of the predictor variables should be scaled to have a

mean of 0 and a standard deviation of 1. This ensures that no predictor variable is overly influential in the model if it happens to be measured in different units.  
`cv = RepeatedKFold():` This tells Python to use **k-fold cross-validation** to evaluate the performance of the model. For this example we choose `k = 10` folds, repeated 3 times.

```
#define predictor and response variables
```

```
X = data]
```

```
y = data]
```

```
#scale predictor variables
```

```
pca = PCA()
```

```
X_reduced = pca.fit_transform(scale(X))
```

```
#define cross validation method
```

```
cv = RepeatedKFold(n_splits=10, n_repeats=3,  
random_state=1)
```

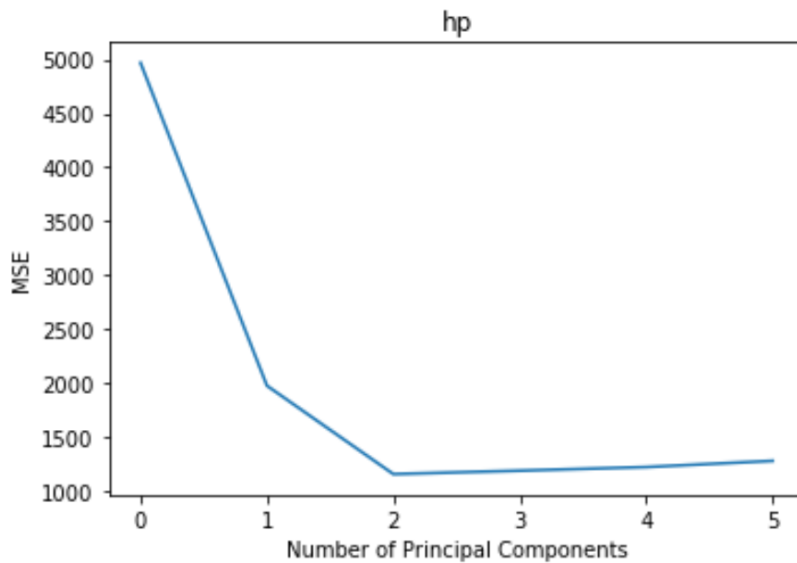
```
regr = LinearRegression()
```

```
mse =
```

```
# Calculate MSE with only the intercept
```

```
score = -1*model_selection.cross_val_score(regr,
```

```
np.ones((len(X_reduced),1)), y, cv=cv,  
scoring='neg_mean_squared_error').mean()  
mse.append(score)  
  
# Calculate MSE using cross-validation, adding one  
component at a time  
for i in np.arange(1, 6):  
score = -1*model_selection.cross_val_score(regr,  
X_reduced, y, cv=cv,  
scoring='neg_mean_squared_error').mean()  
mse.append(score)  
  
# Plot cross-validation results  
plt.plot(mse)  
plt.xlabel('Number of Principal Components')  
plt.ylabel('MSE')  
plt.title('hp')
```



The plot displays the number of principal components along the x-axis and the test MSE (mean squared error) along the y-axis.

From the plot we can see that the test MSE decreases by adding in two principal components, yet it begins to increase as we add more than two principal components.

Thus, the optimal model includes just the first two principal components.

We can also use the following code to calculate the percentage of variance in the response variable explained by adding in each principal component to the model:

```
np.cumsum(np.round(pca.explained_variance_ratio_,  
decimals=4)*100)
```

```
array()
```

**We can see the following:**

**By using just the first principal component, we can explain 69.83% of the variation in the response variable. By adding in the second principal component, we can explain 89.35% of the variation in the response variable.**

**Note that we'll always be able to explain more variance by using more principal components, but we can see that adding in more than two principal components doesn't actually increase the percentage of explained variance by much.**

**Step 4: Use the Final Model to Make Predictions**

**We can use the final PCR model with two principal components to make predictions on new observations.**

**The following code shows how to split the original dataset into a training and testing set and use the PCR**

model with two principal components to make predictions on the testing set.

```
#split the dataset into training (70%) and testing (30%) sets
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=0)
```

```
#scale the training and testing data
```

```
X_reduced_train = pca.fit_transform(scale(X_train))
```

```
X_reduced_test = pca.transform(scale(X_test))
```

```
#train PCR model on training data
```

```
regr = LinearRegression()
```

```
regr.fit(X_reduced_train, y_train)
```

```
#calculate RMSE
```

```
pred = regr.predict(X_reduced_test)
```

```
np.sqrt(mean_squared_error(y_test, pred))
```

40.2096

We can see that the test RMSE turns out to be 40.2096. This is the average deviation between the predicted value for *hp* and the observed value for *hp* for the

**observations in the testing set.**

**The complete Python code use in this example can be found [here](#).**

ARABPSYCHOLOGY.COM