

How to Perform Principal Components Regression in R (Step-by-Step)

Authored by
stats writer

December 18, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Perform Principal Components Regression in R (Step-by-Step)*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=107824>

Principal Components Regression (PCR) is a robust and essential method in regression analysis designed to handle datasets plagued by highly correlated predictor variables. Standard multiple linear regression often suffers when predictors exhibit strong dependency, leading to unstable and unreliable coefficient estimates. PCR tackles this challenge by intelligently reducing the dimensionality of the predictor space. It achieves this by transforming the original, often numerous, features into a smaller set of uncorrelated variables known as Principal Components. These components retain the maximum possible variance from the original data, ensuring that critical information is preserved while the issue of multicollinearity is mitigated. This tutorial provides a comprehensive, step-by-step guide on implementing PCR using the powerful statistical capabilities of the R programming language.

The practical application of PCR involves several critical stages within the R environment. Initially, the necessary statistical packages must be loaded. Subsequently, the model matrix is constructed, and the principal components are calculated. Finally, a standard linear regression is performed using these newly derived components as the independent variables, rather than the original predictors. Functions such as `prcomp()` (or `pcr()` from the `pls` package), `predict()`, and `lm()` facilitate this process within R, allowing data scientists to build more accurate and stable predictive models, especially in high-dimensional settings.

The Challenges of Traditional Multiple Linear Regression

When analyzing the relationship between a set of p predictor variables and a single response variable, multiple linear regression (MLR) is the conventional starting point. MLR relies on the method of least squares to determine the optimal coefficients that minimize the Sum of Squared Residuals (RSS). This minimization criterion seeks the line of best fit by calculating the vertical distance between the actual data points and the predicted values.

The mathematical foundation of minimizing prediction error is defined by the following equation:

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

Where the components of this formula represent:

Σ : The Greek symbol signifying the mathematical operation of **summation** across all observations.

y_i : The actual, observed response value for the i -th data point in the sample.

\hat{y}_i : The predicted response value generated by the multiple linear regression model for the i -th observation.

While MLR is highly effective, its assumptions can be severely violated when faced with high levels of correlation among the predictor variables--a phenomenon known as multicollinearity. When predictors are near-linearly dependent, the design matrix becomes unstable, causing the resulting

coefficient estimates to be highly sensitive to minor changes in the data. This leads to inflated standard errors and high variance in the coefficients, making the interpretation of feature importance unreliable and hindering the model's generalization ability.

Addressing this structural issue is paramount for stable modeling. Principal Components Regression (PCR) offers a sophisticated alternative. Instead of using the original predictors directly, PCR first extracts M linear combinations--the principal components--from the predictor matrix. Since these components are mathematically guaranteed to be orthogonal (uncorrelated), they eliminate the issue of multicollinearity by design. Subsequently, a standard linear regression model is fitted using only these M principal components as the new set of predictors. This strategic dimensionality reduction results in a model that is often more stable and provides better out-of-sample prediction performance.

Step 1: Loading Essential R Packages

To efficiently execute the Principal Components Regression process in R, the specialized `pls` package is the preferred tool. The `pls` package provides optimized functions, notably `pcr()`, which seamlessly integrates the steps of PCA extraction and subsequent linear regression fitting. Utilizing this package streamlines the workflow and ensures accurate calculation of cross-validation metrics necessary for component selection.

If the `pls` package has not yet been installed in your R environment, it must be loaded using the `install.packages()` function. Once installed, the `library()` function makes all the package's functions available for use in the current R session. It is crucial to perform these steps before attempting to fit any PCR models.

#install pls package (if not already installed)

```
install.packages("pls")
```

```
#load pls package
```

```
library(pls)
```

The `pls` package is exceptionally valuable because it handles the internal mechanics of centering and scaling the predictor variables, as well as the cross-validation process, all within a single function call, making the implementation of PCR highly accessible within the R programming language environment.

Step 2: Selecting and Preparing the Data

For this illustrative example, we will leverage the renowned built-in R dataset, `mtcars`, which contains extensive data characterizing 32 different automobiles across 11 key metrics. Analyzing

this dataset allows us to explore the relationship between various mechanical specifications and horsepower output. Understanding the structure of the data is the first step toward building a meaningful model.

We begin by displaying the initial rows of the dataset to familiarize ourselves with the variables and their respective scales. This initial examination confirms the presence of several potentially correlated variables, such as displacement (`disp`), weight (`wt`), and miles per gallon (`mpg`), which makes it an ideal candidate for PCR.

#view first six rows of mtcars dataset

head(mtcars)

```
mpg cyl disp hp drat wt  qsec vs am gear carb
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

Our objective is to model the car's horsepower (`hp`), which will serve as our response variable. We will utilize a subset of five other mechanical attributes as our predictor variables in the PCR model. The selected predictors are:

mpg: Miles per gallon (fuel efficiency)

disp: Displacement (engine size)

drat: Rear axle ratio

wt: Weight (in 1000 lbs)

qsec: Quarter mile time

Step 3: Implementing and Configuring the PCR Model in R

Fitting the PCR model using the `pcr()` function is straightforward, but careful consideration of its arguments is essential for accurate results. We define the model using the formula notation (`response ~ predictors`) and specify the dataset. The two most critical arguments are `scale=TRUE` and `validation="CV"`, which govern data preprocessing and model evaluation, respectively.

The argument `scale=TRUE` instructs R to standardize the predictor variables before calculating the principal components. This standardization process scales each predictor to have a mean of 0 and a standard deviation of 1. This step is crucial because Principal Component Analysis is sensitive to

the scale of the variables; without scaling, variables measured in larger units (e.g., displacement) would disproportionately influence the resulting components compared to variables measured in smaller units (e.g., rear axle ratio). By ensuring equal variance, we guarantee that the components accurately reflect the underlying structure of the correlation.

The argument `validation="cv"` specifies the method for assessing the model's predictive performance. In this case, we employ k-fold cross-validation (using the default $k=10$). Cross-validation systematically partitions the data into training and testing subsets, providing a robust estimate of the model's true prediction error on unseen data. Setting a seed ensures reproducibility of the cross-validation folds.

#make this example reproducible

set.seed(1)

#fit PCR model

```
model <- pcr(hp~mpg+dis+drat+wt+qsec, data=mtcars, scale=TRUE, validation="CV")
```

Step 4: Optimizing the Model by Selecting Principal Components

A defining challenge in PCR is determining the optimal number of principal components (\$M\$) to retain. Using too few components risks underfitting the model by discarding meaningful variance, while using too many components risks overfitting or reintroducing noise, potentially leading back to problems similar to those caused by multicollinearity. The standard practice for selecting \$M\$ relies on evaluating the model's performance on unseen data, which is provided by the cross-validation metrics.

We inspect the output of the `summary(model)` function, paying close attention to the Cross-Validated Root Mean Squared Error (RMSE) of Prediction (RMSEP) and the percentage of variance explained. The RMSEP measures the average magnitude of the errors in prediction, evaluated across the cross-validation folds. We seek the number of components (\$M\$) that yields the minimum RMSEP value, as this indicates the most effective balance between bias and variance.

#view summary of model fitting

summary(model)

Data: X dimension: 32 5

Y dimension: 32 1

Fit method: svdpc

Number of components considered: 5

VALIDATION: RMSEP

Cross-validated using 10 random segments.

(Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps

CV 69.66 44.56 35.64 35.83 36.23 36.67

adjCV 69.66 44.44 35.27 35.43 35.80 36.20

TRAINING: % variance explained

1 comps 2 comps 3 comps 4 comps 5 comps

X 69.83 89.35 95.88 98.96 100.00

hp 62.38 81.31 81.96 81.98 82.03

Step 5: Interpreting Cross-Validation Metrics for Component Selection

The output of the model summary provides two crucial tables for optimization. The first, `VALIDATION: RMSEP`, is paramount for selecting k . It shows how the test RMSE changes as we incrementally add more principal components to the regression model.

1. VALIDATION: RMSEP Analysis: This table quantifies the predictive error for models ranging from a simple intercept-only model up to a model utilizing all five possible components. We observe a dramatic reduction in error initially:

Starting point (Intercept only): Test RMSE is **69.66**.

Adding the first principal component (1 comp): RMSE drops significantly to **44.56**.

Adding the second principal component (2 comps): RMSE drops further to the minimum value of **35.64**.

Crucially, when we consider adding the third principal component (3 comps), the test RMSE begins to increase (35.83). This signals that components three, four, and five introduce more noise than signal, indicating the onset of overfitting relative to the cross-validated error metric. Therefore, the minimum RMSEP confirms that the most parsimonious and effective predictive model uses only the first **two principal components**.

2. TRAINING: % variance explained Analysis: This table provides context by showing how much cumulative variance in the predictor matrix (X) and the response variable (hp) is accounted for by the components. This metric helps confirm that a significant portion of the input data structure is captured by the selected components.

By using just the first Principal Component, we can explain **69.83%** of the variation in the predictor variables (X).

By adding in the second Principal Component, we can explain a combined **89.35%** of the variation in the predictor variables (X).

While using all five components will always explain 100% of the variance in the predictor matrix, the key insight here is efficiency: by retaining just the first two components, we retain nearly 90% of the variance while simultaneously minimizing predictive error. This demonstrates the powerful dimensionality reduction achieved by Principal Components Regression.

Step 6: Visualizing Model Performance Metrics

Visualizing the cross-validation results offers an intuitive method for confirming the optimal number of components identified in the summary tables. The `validationplot()` function allows us to plot key performance indicators--specifically the Root Mean Squared Error of Prediction (RMSEP), the Mean Squared Error of Prediction (MSEP), and the R^2 value--against the number of components used.

We generate three separate plots to cover these critical metrics. The optimal number of components corresponds to the point where the RMSEP or MSEP is minimized, and typically where the R^2 plateaus or starts diminishing in validation sets. This visual confirmation is often clearer than interpreting raw numerical output.

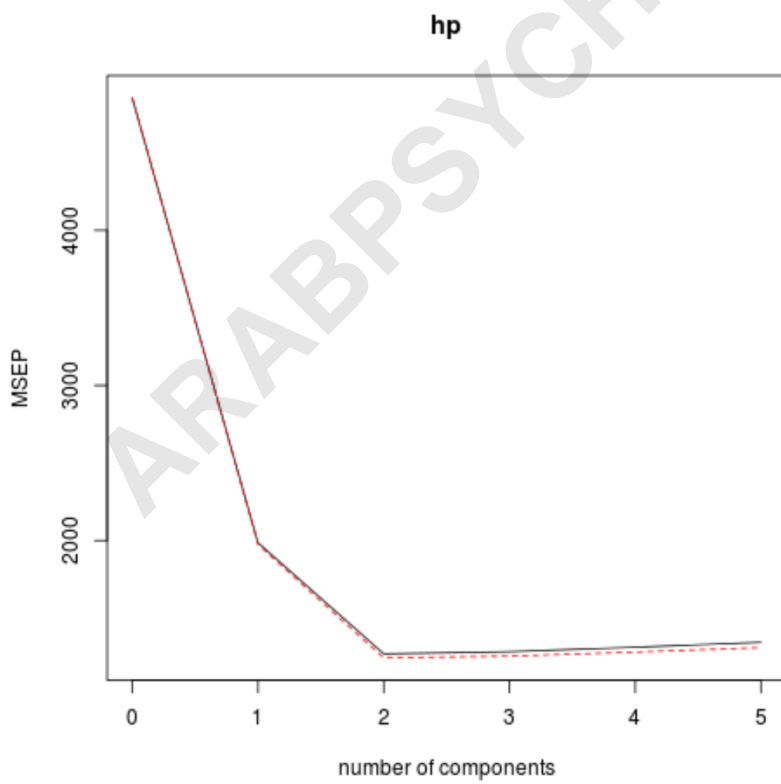
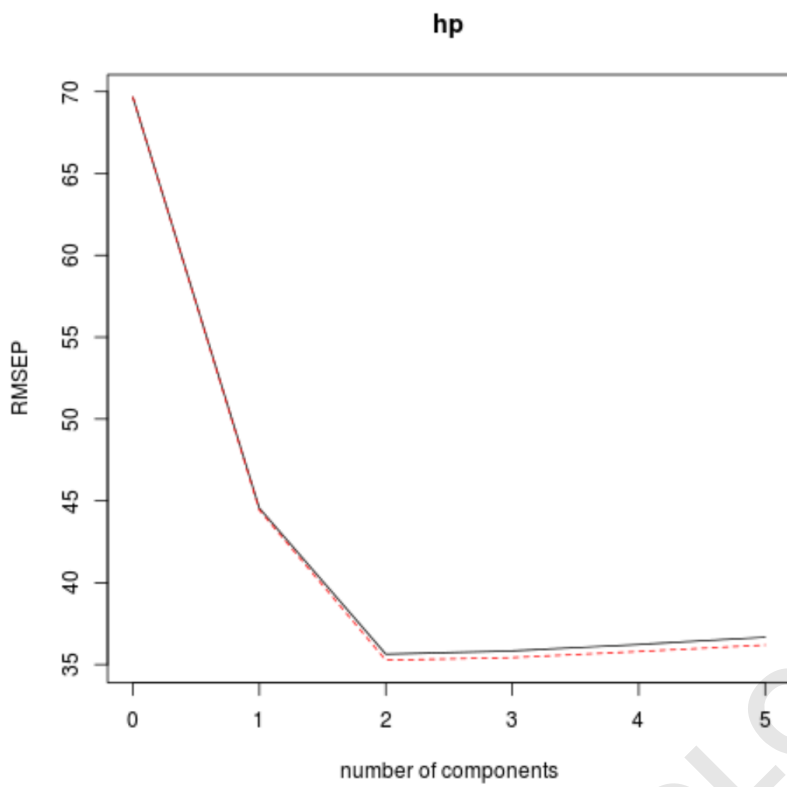
#visualize cross-validation plots

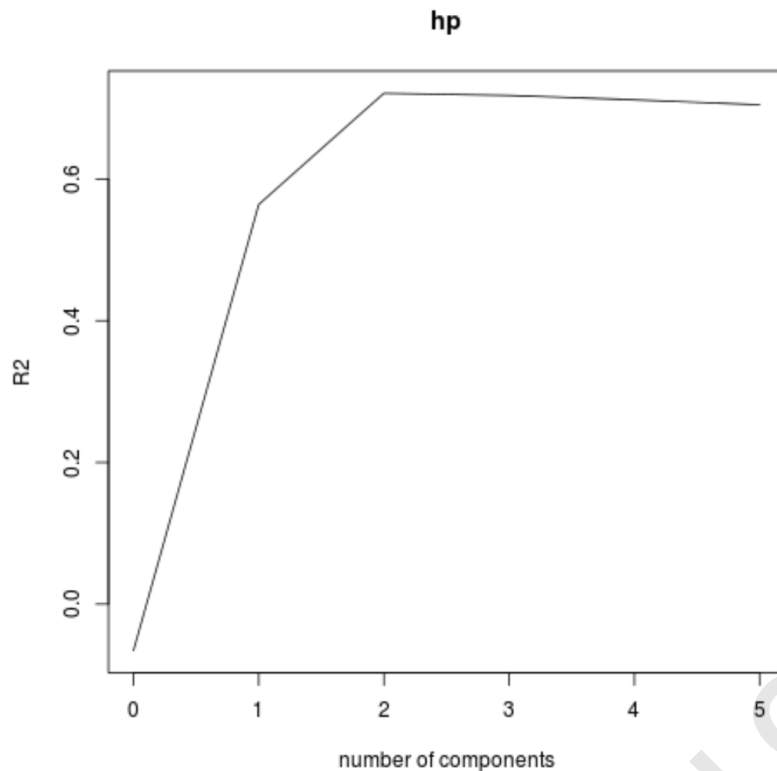
```
validationplot(model)
```

```
validationplot(model, val.type="MSEP")
```

```
validationplot(model, val.type="R2")
```

The generated plots clearly illustrate the trade-off inherent in model complexity. The first two components lead to a sharp decline in error and a rapid increase in R^2 . However, beyond the second component, the performance curves flatten or, more importantly, begin to rise again in terms of RMSEP, confirming the presence of minimal prediction gain or the introduction of noise.





In visual summary, each plot reinforces the finding derived from the numerical analysis: the model fit substantially improves by incorporating the first two principal components. Adding subsequent components provides negligible benefit or actively degrades the cross-validated predictive performance. Thus, we formally conclude that the optimal PCR model for predicting horsepower in the `mtcars` dataset utilizes just the first two principal components.

Step 7: Applying the Optimized PCR Model for Prediction

With the optimal number of components ($M=2$) selected, the final step involves using the refined PCR model to make predictions on a completely new, unseen dataset. To simulate real-world application and accurately assess generalization ability, we must first split our original `mtcars` data into distinct training and testing sets.

We will allocate the first 25 observations for training the model, allowing the PCR algorithm to learn the underlying relationships between the principal components and the response variable (`hp`). The remaining observations will form the testing set, which is used solely for evaluating the final predictive accuracy. It is crucial that the testing set is independent of the training process to ensure an unbiased estimate of the model's performance.

The `pcr()` function is used again to train the model, this time exclusively on the defined training data. Subsequently, the `predict()` function is utilized, passing the testing predictor variables

(`test`) and explicitly setting the argument `ncomp=2`, confirming that only the two optimal principal components are employed for the final prediction process.

#define training and testing sets

```
train <- mtcars
```

```
y_test <- mtcars
```

```
test <- mtcars
```

```
#use model to make predictions on a test set
```

```
model <- pcr(hp~mpg+disp+drat+wt+qsec, data=train, scale=TRUE, validation="CV")
```

```
pcr_pred <- predict(model, test, ncomp=2)
```

```
#calculate RMSE
```

```
sqrt(mean((pcr_pred - y_test)^2))
```

```
56.86549
```

The calculated test RMSE for the holdout set is **56.86549**. This figure represents the average magnitude of error, quantifying how far, on average, the predicted horsepower values deviated from the actual observed horsepower values in the testing data. This final metric provides a concrete measure of the predictive capability of the PCR model when applied to fresh, unanalyzed observations, confirming its utility in circumventing issues arising from multicollinearity.

By successfully implementing Principal Components Regression, we have created a robust statistical model that effectively manages the potential pitfalls of correlated predictors, demonstrating how to achieve stable and interpretable predictions through intelligent dimensionality reduction in R.

The complete and executable R code used throughout this detailed example can be found [here](#).