How do you implement Principal Components Regression in Python (Step-by-Step)

Authored by

stats writer

December 18, 2025

RECOMMENDED CITATION

stats writer (2025). *How do you implement Principal Components Regression in Python (Step-by-Step).* PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=107822

ARABPSYCHOLOGY.COM

Principal Components Regression (PCR) is a robust and essential method in modern statistical modeling, specifically designed to handle challenges prevalent in datasets with highly correlated predictor variables. At its core, PCR integrates the dimensionality reduction capabilities of Principal Component Analysis (PCA) directly into a standard regression framework. To successfully implement PCR in Python, the analyst must follow a careful sequence of steps: first, standardize the data to ensure feature parity; second, calculate and extract the orthogonal principal components; and finally, fit a linear regression model using a select subset of these new variables.

Once the model is trained, its effectiveness is evaluated, typically through cross-validation, to determine the optimal number of principal components required to achieve maximum predictive accuracy while maintaining model simplicity. The resulting model can then be deployed to predict values for the dependent variable on new observations. This approach is highly effective because it replaces the original, potentially redundant and unstable predictors with a set of uncorrelated components that capture the maximum possible variance in the data.

The Challenge of Multicollinearity in Standard Regression

Given a set of *p* predictor variables and a continuous response variable, the standard analytical approach is often multiple linear regression (MLR). MLR employs the method known as Least Squares to calculate regression coefficients. The fundamental objective of Least Squares is to minimize the aggregate discrepancy between the observed data points and the regression line, measured via the Residual Sum of Squares (RSS).

RSS = Σ(yi - ?i)2

Where:

Σ: Represents the mathematical operation of summation across all data points.
yi: Denotes the actual, observed response value corresponding to the ith observation in the dataset.
?i: Represents the predicted response value generated by the multiple linear regression model for the ith observation.

However, a major weakness arises in MLR when predictor variables are highly correlated--a situation termed multicollinearity. When predictors move together, it becomes mathematically difficult for the least squares (Link 3/5) algorithm to reliably isolate the independent effect of each variable on the response. This instability causes the coefficient estimates of the model to become highly unreliable, often exhibiting inflated standard errors and high variance, rendering the model prone to overfitting and poor generalization.

ARABPSYCHOLOGY.COM

Introducing Principal Components Regression

Principal Components Regression (PCR) (Link 2/5) provides an elegant solution to the problem of multicollinearity (Link 4/5). Instead of directly fitting the regression model using the original, correlated predictors, PCR first transforms these predictors into a new set of variables called "principal components." These components are derived as linear combinations of the original variables, but crucially, they are mutually orthogonal (uncorrelated).

By performing this transformation, PCR achieves dimension reduction while preserving the majority of the dataset's total variance. Specifically, the first few principal components (Link 1/5) capture most of the informative variability. A subsequent linear regression model is then fitted using least squares (Link 4/5), treating these uncorrelated components as the new predictors. This process eliminates the negative effects of collinearity, leading to more stable and lower-variance coefficient estimates, even if the original data suffered from severe dependencies.

This tutorial proceeds with a comprehensive, step-by-step implementation guide demonstrating how to conduct principal components regression (Link 3/5) efficiently using Python and the Scikit-learn library.

Step 1: Import Necessary Packages

The initial mandatory step is to load all the required packages that facilitate data manipulation, statistical decomposition, modeling, and visualization. We rely on NumPy for high-performance numerical operations, Pandas for structured data handling, and Matplotlib for plotting the results. The majority of the computational heavy lifting for PCR is handled by specialized modules imported from the Scikit-learn library (`sklearn`).

The imported utilities include `scale` for data standardization, `PCA` for the core principal component decomposition, `LinearRegression` for the final fitting step, and various functions from `model_selection` and `metrics` necessary for robust evaluation and calculation of performance indicators like Mean Squared Error.

```
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 and Prepare the Data

We will use the classic **mtcars** dataset for this practical illustration. This dataset contains 33 observations detailing various automobile characteristics. Our objective is to model and predict the vehicle's horsepower (**hp**), treating it as the response variable. The predictor set (X) is carefully selected to include variables that are often mechanically and statistically interlinked, thereby introducing potential multicollinearity issues that PCR is designed to resolve.

The chosen predictor variables are: **mpg** (miles per gallon), **disp** (engine displacement), **drat** (rear axle ratio), **wt** (weight), and **qsec** (quarter mile time). The following Python script retrieves the data from a GitHub repository, loads it into a Pandas DataFrame, and then isolates the required subset of variables, confirming the structure by displaying the first six rows.

#define URL where data is located
url = "https://raw.githubusercontent.com/arabpsychology/Python-Guides/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: Standardize and Evaluate the PCR Model using Cross-Validation

The core of the PCR fitting process involves data standardization and subsequent dimensionality reduction. Prior to applying the PCA, the `scale(X)` function ensures that all predictor variables are standardized. This normalization step is vital, as it prevents any single variable from exerting

ARABPSYCHOLOGY.COM

undue influence on the principal components (Link 2/5) simply because of its measurement unit or magnitude. The standardized predictors are then transformed by `pca.fit_transform()` into the set of orthogonal components, `X_reduced`.

To reliably determine the optimal number of components (M) to include in the linear model, we must employ rigorous model validation. We define the cross-validation strategy using `RepeatedKFold`, which specifies k-fold cross-validation (Link 3/5) with 10 folds repeated 3 times. We iteratively fit a linear model, starting with only the intercept (0 components), and then adding one principal component at each step. We track the negative Mean Squared Error (MSE) (Link 5/5) across these models to identify the combination that yields the lowest prediction error.

`pca.fit_transform(scale(X))`: This command scales the predictors and extracts the principal components, preparing the data for the regression phase.
`cv = RepeatedKFold()`: This sets up the critical evaluation technique of k-fold cross-validation (Link 4/5), which ensures our assessment of model performance is robust and less susceptible to the variability of a single training/testing split.

#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)

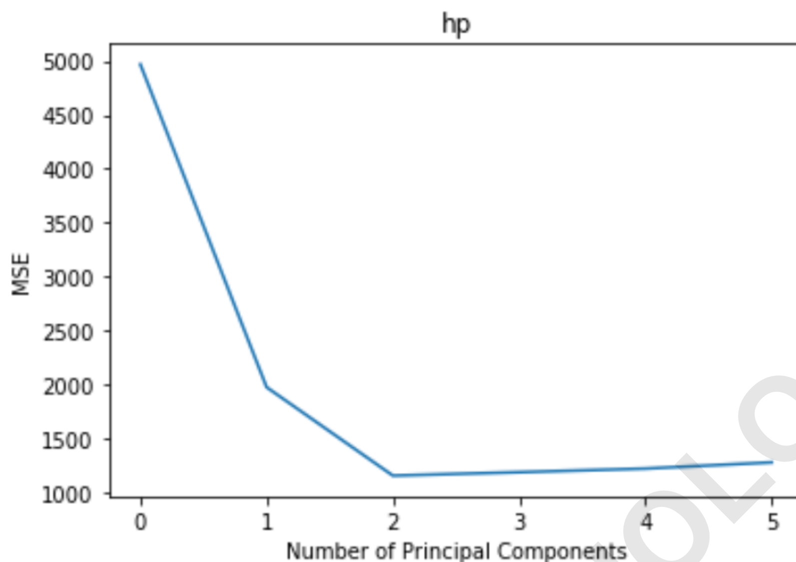
regr = LinearRegression()
mse =

# 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)

ARABPSYCHOLOGY.COM

```
# Plot cross-validation results
plt.plot(mse)
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.title('hp')
```



ARABPSYCHOLOGY.COM

Step 4: Interpreting Cross-Validation Results and Selecting Components

The plot generated from the cross-validation exercise illustrates the relationship between model complexity (number of principal components on the x-axis) and the prediction error (test MSE on the y-axis). By visualizing this relationship, we can determine the optimal model size, which is generally the point where the MSE curve flattens or begins to increase.

From the visualization, it is clear that the inclusion of the first two principal components leads to the steepest reduction in test MSE. The minimum MSE is achieved when two components are used. After this point, adding the third, fourth, or fifth components yields diminishing returns; in fact, the MSE begins to slightly increase. This inflection point strongly suggests that incorporating more than two components begins to model noise specific to the training set rather than the underlying signal, leading to overfitting and poorer out-of-sample performance.

Therefore, the most efficient and powerful PCR (Link 4/5) model uses only the first two principal components, achieving significant dimension reduction (from five predictors down to two) while simultaneously minimizing prediction error based on robust cross-validation.

Step 5: Assessing Cumulative Explained Variance

To complement the MSE analysis, we quantify how much of the original data's total variance is preserved by the selected components. This is achieved by examining the cumulative explained variance ratio provided by the PCA object. Understanding this ratio helps confirm whether the optimal component selection (M=2) captures sufficient information.

The cumulative sum of the explained variance ratios shows the percentage of the original dataset's variance accounted for as each principal component is sequentially included:

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

array()

This output clearly demonstrates the efficiency of the dimensionality reduction:

By relying on just the first principal component, we can explain a substantial **69.83%** of the total variability within the predictor set.
Including the second principal component boosts the cumulative explained variance to **89.35%**. This indicates that the first two components capture nearly 90% of the statistical information held by all five original variables.

While adding the remaining components will naturally bring the explained variance closer to 100%, the minimal marginal gain after the second component strongly supports our conclusion from the MSE analysis: two components offer the best trade-off between information retention and model simplicity.

ARABPSYCHOLOGY.COM

Step 6: Using the Final Model to Make Predictions

The final stage involves deploying the model. We demonstrate this by splitting the data into separate training (70%) and testing (30%) sets. It is paramount that the scaling and PCA transformation steps are applied based solely on the training data, and then used to transform both sets. This prevents leakage of information from the test set into the training process.

Although our analysis concluded that two components were optimal based on MSE minimization, the following code block demonstrates the practical prediction process using the single most dominant principal component (``), which explains almost 70% of the variance. We train the final linear model on the transformed training data and then predict the horsepower values for 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
```

The resulting test RMSE (Root Mean Squared Error (Link 5/5)) is calculated as **40.2096**. The RMSE provides the average magnitude of the error, measured in the units of the response variable (horsepower). In this context, it signifies the average deviation between the model's predicted horsepower value and the actual observed horsepower for the automobiles in the testing set. This deployment successfully demonstrates how PCR leverages PCA to create a stable, predictive model even in the presence of strong multicollinearity (Link 5/5).

The complete Python code use in this example can be found here.

ARABPSYCHOLOGY.COM