Table of Contents

[Exploratory Factor Analysis Introduction 1](#_Toc34208982)

[Bartlett’s Test of Sphericity 2](#_Toc34208983)

[Kaiser-Meyer-Olkin (KMO) Test 2](#_Toc34208984)

[Explanatory Analysis 4](#_Toc34208985)

[Conclusion about Explanatory Factor Analysis 6](#_Toc34208986)

[Building a Predictive Model with Our Factors 6](#_Toc34208987)

[Comparing PCA with Factor Analysis 8](#_Toc34208988)

[References 10](#_Toc34208989)

## Exploratory Factor Analysis Introduction

This discussion examines factor analysis to describe the variance in data sets which have complex interrelated features. In other words, we will use factor analysis to understand how groupings of labelled features move together. You could even say that **factor analysis allows you to explain complex correlations**. We experienced this difficulty earlier in the term when examining a correlation matrix for the housing data set.

**Factor analysis explains common variance amongst labelled features.** Factor analysis is often **applied to better understand** psychological tests, marketing surveys and other applications where the features are known and the relationships between features are highly correlated with high VIF scores.

Factor analysis considers unique variance of features. Unique variance is the independent variance of a feature and the variance due to error. PCA, on the other hand, assumes that all variance is common across all components. This is a difficult concept to grasp at first so I hope the explanation that follows will help to clarify your understanding.

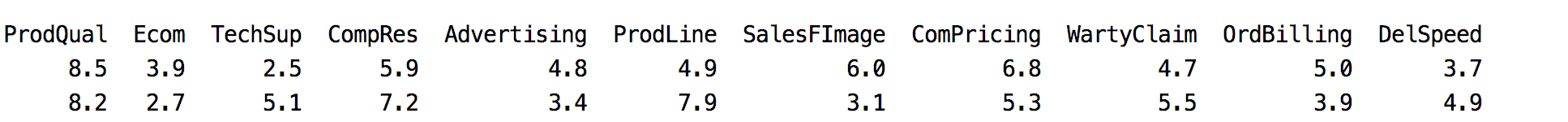
Example : Exploratory Factor Analysis

This example uses factor analysis to help interpret responses for a survey about customer satisfaction for the hair products of a company.. This example is based on the code presented at

<https://medium.com/analytics-vidhya/multiple-linear-regression-factor-analysis-in-r-35a26a2575cc>

After initially loading and preparing the data, Table 1 shows a snapshot of the predictor variables with corresponding ratings in the first two rows of our data set.

Table : Hair Products Survey Sample Data



Before using factor analysis, it is important to ensure enough variance exists amongst predictors.

### Bartlett’s Test of Sphericity

Bartlett’s test determines if enough correlation between predictors exists to enable a factor analysis. The null hypothesis, , suggests that the correlation matrix is equivalent to the identity matrix where no correlation exists. (See Figure 1 for an example of a correlation matrix.)

Figure : Sample Identity Matrix



Since the outcome is statistically significant, Bartlett’s test of sphericity shows enough correlation exists to enable the factor analysis:

|  |
| --- |
| Bartlett's test chi-square value:  629.7198819161315  Bartlett's test p-value:  6.884240423574735e-99 |

### Kaiser-Meyer-Olkin (KMO) Test

The Kaiser-Meyer-Olkin (KMO) Test **measures how well-suited your data is for factor analysis.** The statistic measures the proportion of common variance among variables. A lower proportion of common variance indicates that the data is suitable for factor analysis.

KMO returns values between 0 and 1. To interpret the statistic:

* **KMO values between 0.6 and 1 indicate the sampling is adequate**.
* KMO values less than 0.6 indicate the sampling is not adequate. Some authors put this value at 0.5, so use your own judgment for values between 0.5 and 0.6.

For our case, the KMO test is 0.65 so the sample appears to be suitable for factor analysis.

|  |
| --- |
| Kaiser-Meyer-Olkin (KMO) Test: Suitability of data for factor analysis.  0.6485745602894757 |

The **tests indicate that factor analysis can be applied**. Next, factors are generated and eigenvalues are displayed for each. Kaiser’s stopping rule suggests that four components can be selected since only the first four Eigenvalues are greater than 1. (You could also draw a scree plot or a cumulative variance plot but I won’t here to keep it easier.

|  |
| --- |
| Factors:  [[ 1.80481609e-01 -3.67732943e-01 -6.03448564e-02]  [ 2.95431966e-01 6.50921961e-01 2.85373682e-01]  [ 2.72717302e-01 -4.01519516e-01 7.08682189e-01]  [ 8.63026330e-01 -8.34810383e-04 -2.56871030e-01]  [ 2.93216384e-01 4.60430638e-01 9.71635543e-02]  [ 6.54067937e-01 -4.33889821e-01 -1.43221525e-01]  [ 3.92693321e-01 7.65702862e-01 3.48456376e-01]  [-2.17756271e-01 5.38417829e-01 -2.61572443e-02]  [ 3.78410549e-01 -3.52926759e-01 7.25703902e-01]  [ 7.48674912e-01 8.97496358e-03 -1.76304024e-01]  [ 8.91036030e-01 8.58153982e-02 -2.97368615e-01]]  Eignenvalues: # You can look at eignen values or use a scree or cumulative plot to pick values ( eignen values > 1 are desirable)  [**3.42697133 2.55089671 1.69097648 1.08655606** 0.60942409 0.55188378  0.40151815 0.24695154 0.20355327 0.13284158 0.09842702] |

Here is the code which generates the preliminary output:

|  |
| --- |
| import pandas as pd  import numpy as np  # Can get from PyCharm terminal.  from factor\_analyzer import FactorAnalyzer  # Read data.  from sklearn.linear\_model import LinearRegression  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "Factor-Hair-Revised.csv"  data = pd.read\_csv(PATH + CSV\_DATA, sep=',')  # Create data frame without ID and Satisfaction columns.  df = data.copy()  del df['ID']  del df['Satisfaction']  # Display all columns of the data frame.  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  print(df.head(2))  # Bartlett's test of sphericity tests the hypothesis that your correlation matrix  # is an identity matrix. If the correlation matrix is an identity matrix the  # columns are unrelated and are therefore unsuitable for structure detection.  # If the test is insignificant do not use factor analysis.  from factor\_analyzer.factor\_analyzer import calculate\_bartlett\_sphericity  chi\_square\_value, p\_value=calculate\_bartlett\_sphericity(df)  print("\nBartlett's test chi-square value: ")  print(chi\_square\_value)  print("\nBartlett's test p-value: ")  print(p\_value)  # Kaiser-Meyer-Olkin (KMO) test measures the proportion of variance among  # variables that might be common variance. The lower the proportion,  # the more suited your data is to Factor Analysis. Factor analysis is suitable  # for scores of 0.6 (and sometimes 0.5) and above.  from factor\_analyzer.factor\_analyzer import calculate\_kmo  kmo\_all,kmo\_model=calculate\_kmo(df)  print("\nKaiser-Meyer-Olkin (KMO) Test: Suitability of data for factor analysis.")  print(kmo\_model)  # Create factor analysis and examine loading vectors and Eigenvalues.  fa = FactorAnalyzer(rotation=None)  fa.fit(df)  print("\nFactors:")  print(fa.loadings\_)  ev, v = fa.get\_eigenvalues()  print("\nEignenvalues:")  print(ev) |

### Explanatory Analysis

Now that the first four factors have been selected, this next section interprets each factor’s representation of variance. The factor interpretation can be very subjective so be careful to mislead others with the interpretation. Whenever possible seek the opinion and confirmation of a subject matter expert when explaining factor variance.

Example : Generating and Interpreting Factors

In this example, we will start adding the code below at the end of the code from Example 1. Only four factors had eigenvalue scores that are greater than 1 so we will use those four factors. Usually a rotation is performed with the selected number of features to help clearly reveal each factor. There are many rotation types and we won’t get into the differences but you can experiment with them to see if they improve your results.

|  |
| --- |
| # Pick factors where eigenvalues are greater than 1.  fa = FactorAnalyzer(rotation="varimax",n\_factors=4)  fa.fit(df)  # Create formatted factor loading matrix.  dfFactors = pd.DataFrame(fa.loadings\_)  dfFactors['Categories'] = df.keys().values.tolist()  dfFactors = dfFactors.rename(columns={0:'Factor 1',  1:'Factor 2', 2:'Factor 3', 3:'Factor 4'})  print("\nFactors: ")  print(dfFactors)  # Display common variance.  print("\nVariance Explanation")  variances = fa.get\_factor\_variance()  varianceDf = pd.DataFrame(data=variances)  varianceDf = varianceDf.rename(columns={0:'Purchase',  1:'Marketing', 2:'Post Purchase', 3:'Position'})  varianceDf['Totals'] = ['Eigenvalues', '% variance', 'cumulative variance']  print(varianceDf) |

The result from running the code after applying the changes shows four special components called factors. Table 2 highlights the dominant correlations of each factor. Since we can extract these correlation patterns the factors are also called **latent variables** because they describe hidden relationships in the data. Each cell value is called a **loading**.

Note that competitive pricing is outlined in red because this variance is negative. Perhaps the high negative loading occurs because people report that they do not want to pay a higher price.

After the high correlations have been identified the factors can be interpretted and labelled. The interpretations are listed on the right of Table 2.

## Table 2: Explaining the Latent Variables

|  |  |
| --- | --- |
|  | **Factor 1 – Purchase**  Contains features related to purchasing which range from placing the order to billing and having it delivered. |
| **Factor 2 – Marketing**  Contains features related to marketing processes like sales force image and spending on advertising. |
| **Factor 3 – Post Purchase**  Includes warranty claims and technical support. |
| **Factor 4 – Product Position**  Covers product quality, product line and pricing. |

Table 3 shows the amount of common variance accounted for by each of the labelled factors. The total percent common variance is calculated by summing the squares of the loadings (cells) in each column. Each cell, loading, represents the correlation of that feature with the current factor.

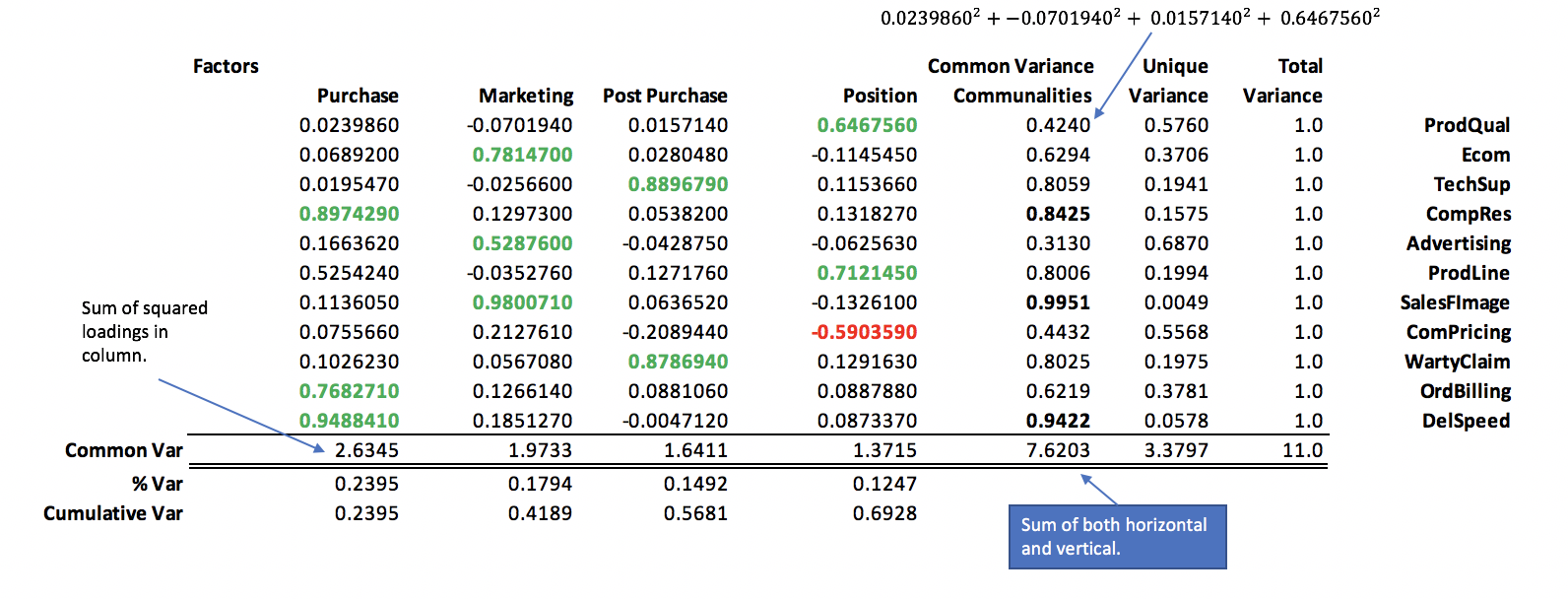
## Table 3: Common Variance Totals

|  |
| --- |
| Variance Explanation  Purchase Marketing Post Purchase Position Totals  0 2.634520 1.973267 1.641068 1.371469 Eigenvalues  1 0.239502 0.179388 0.149188 0.124679 % variance  2 0.239502 0.418890 0.568078 0.692757 cumulative variance |

Factor analysis provides a tidy summary common variance for all features and factors. For explanatory purposes, this analysis has reduced the explanation of survey features from 11 to 4 latent variables which account for 69.28% of the common variance (interrelated variance between predictors).

From our summary, we can say purchase and marketing factors account for the highest amount of variance in customer responses around customer satisfaction. When considering features we can say delivery speed has a strong correlation with the purchasing factor and sales force image has a strong correlation with the marketing factor. See Table 4 for a summary of common and unique variance.

## Table 4: Summarizing Common and Unique Variance



## Conclusion about Explanatory Factor Analysis

When **explaining survey variance, we have reduced 11 interrelated features to 4 latent variables** which is much simpler to comprehend. We also have a good understanding of the features that are strongly correlated with each of the four factors.

## Building a Predictive Model with Our Factors # Important

Now we have some latent variables, we could use them to build a linear regression model to predict customer satisfaction levels. Or we could combine them with other predictor variables. Or, we could use the information to simplify the model by removing variables which are not contributing to explaining the variance.

Example : Building a Linear Regression

This example builds a regression model entirely with the latent variables. However, the latent variables could be included with other variables to boost predictive power. When examining the scores, the first model looks decent but factor three is insignificant.

|  |
| --- |
| RMSE: 0.7110585593501577  R^2: 0.6191974230580022  Model Coefficients:  [0.58447535 0.56748657 0.00598941 0.48025292]  Model Intercept:  6.814285714285713  Model p-values:  const 8.404154e-65  x1 3.123768e-08  x2 5.155832e-08  x3 9.507624e-01  x4 2.713530e-06 |

The insignificant factor is then dropped to yield the following results.

|  |
| --- |
| RMSE: 0.7127093810693713  R^2: 0.6174271978326823  Model Coefficients:  [0.58449075 0.56754143 0.48039546]  Model Intercept:  6.814285714285713  Model p-values:  const 8.947633e-66  x1 2.445931e-08  x2 4.056186e-08  x3 2.246370e-06 |

Here is the code to build the remainder. To build it, add this code to the end of Example 2.

|  |
| --- |
| # Display RMSE, R^2, model coefficients and intercept.  def showModelSummary(model, y\_test, X\_test\_tranformed):  print("\n\*\*\*\*\*\* MODEL SUMMARY \*\*\*\*\*\*")  pred = model.predict(X\_test\_tranformed)  # Show stats about the regression.  from sklearn.metrics import mean\_squared\_error  mse = mean\_squared\_error(y\_test, pred)  RMSE = np.sqrt(mse)  print("\nRMSE: " + str(RMSE))  from sklearn.metrics import r2\_score  print("\nR^2: ",r2\_score(y\_test,pred))  print("\nModel Coefficients:")  print(model.coef\_)  print("\nModel Intercept:")  print(model.intercept\_)  # Display p-values for model coefficients.  def showCoefficientPValues(y\_train, X\_train\_transformed):  import statsmodels.api as sm  X2 = sm.add\_constant(X\_train\_transformed)  model = sm.OLS(y\_train, X2)  fii = model.fit()  p\_values = fii.summary2().tables[1]['P>|t|']  print("\nModel p-values: ")  print(p\_values)  # Split data before it is transformed.  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test , y\_train, y\_test = train\_test\_split(df, data['Satisfaction'],  test\_size=0.3, random\_state=1)  # Transform data with factor components.  X\_train\_transformed = fa.fit\_transform(X\_train)  X\_test\_tranformed = fa.transform(X\_test)  #----------------------------------------------------------  # Build first model  #----------------------------------------------------------  # Train regression model on training data  model = LinearRegression()  model.fit(X\_train\_transformed, y\_train)  # Show model statistics.  showModelSummary(model, y\_test, X\_test\_tranformed)  # Check coefficient significance.  showCoefficientPValues(y\_train, X\_train\_transformed)  #----------------------------------------------------------  # Build second model without insignificant variable.  #----------------------------------------------------------  # Builds labelled DataFrame with signficant latent variables from  # factor matrix.  def dropInsignificantX(X\_transformed):  # Builds DataFrame from matrix.  dfX = pd.DataFrame(data=X\_transformed)  # Labels columns and drops insignificant column.  dfX = dfX.rename(columns={0:'Purchase',  1:'Marking', 2:'Post Purchase', 3:'Product Position'})  del dfX['Post Purchase']  return dfX  # Prepare significant X values for regression.  trainDF = dropInsignificantX(X\_train\_transformed)  testDF = dropInsignificantX(X\_test\_tranformed)  # Train regression model on training data  model = LinearRegression()  model.fit(trainDF, y\_train)  # Show model statistics.  showModelSummary(model, y\_test, testDF)  # Check coefficient significance.  showCoefficientPValues(y\_train, trainDF.values) |

# Advantages and Disadvantages of PCA with Factor Analysis

Here is a quick summary that explains differences between PCA and Factor Analysis.

|  |  |
| --- | --- |
| PCA | Factor Analysis |
| * Excellent for reducing large unlabeled data such as images, near-infrared spectroscopy. * Poor explanatory value. * Only considers common variance between correlated variables. * Does not consider variance due to error or independent variance of the feature. | * Excellent for explaining labelled data such as housing prices and features, surveys, psychological and behavioral analysis. * Considers both common and unique variance between correlated variables. |

Exercise (10 mark)

The usCityData.csv file contains information that is used to predict home prices. These are the columns:

CRIM - per capita crime rate by town

ZN - proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS - proportion of non-retail business acres per town.

NOX - nitric oxides concentration (parts per 10 million)

RM - average number of rooms per dwelling

AGE - proportion of owner-occupied units built prior to 1940

DIS - weighted distances to five Boston employment centres

RAD - index of accessibility to radial highways

TAX - full-value property-tax rate per $10,000

PTRATIO - pupil-teacher ratio by town

LSTAT - % lower status of the population

MEDV - Median value of owner-occupied homes in $1000's

**Run Bartlett’s test and the KMO test to ensure the data is suitable for factor analysis.** Use exploratory data to create latent variables which describe the relationships between features in the usCityData.csv data set (in the data sets folder of the learning hub).

Kaiser-Meyer-Olkin (KMO) Test: Suitability of data for factor analysis.

0.8465709073764048

Determine the optimal number of factors. Then, create a data table in Excel like the one in Table 4. However, also highlight two to four dominant loadings in each factor in green for positive or red for negative. Generally, the dominant loadings should be greater than 0.5 or more. Then, label each factor.

# Answers

Calculate and show:

* Dominant positive loadings highlighted in green.
* Dominant negative loadings highlighted in red.
* The common variance explained by each factor.
* The % common variance explained by each factor.
* The cumulative common variance for each factor.
* The common variance for each feature.
* The unique variance for each feature.
* Suitable descriptive labels for each factor.

Hint: These functions may be handy:

print(fa.get\_factor\_variance())  
print(fa.get\_communalities())  
print(fa.get\_uniquenesses())

Provide a brief written summary that describes the three factors and their variance.

|  |
| --- |
| Factor 1 – Building Density / Type of Building  Based on the categories that heavily influence Factor 1 we can infer that the type of building influences housing price. This is shown by ZN category and AGE category. Since newer buildings  There is a positive relationship between indus, nox and age categories.  There is a negative relationship between zn and dis categories.  Factor 2 – Municipality/Region  There is a positive relationship between crim, rad and tax. This would indicate that the housing price is directly influenced by relative crime level per capita in an area, by proximity to commuting infrastructure, and by a region’s specific tax rate. This leads me to believe that this factor is related to specific characteristics of a region or municipality. This label makes sense for the correlation between these positively related categories.  Factor 3 – Economic Status / Income Level  The categories that heavily influence factor 3 include LSTAT, RM, and MEDV. Based on these influencers we can infer that the housing price is negatively impacted by lower RM (room) numbers and by high MEDV levels. The high MEDV levels can be explained with price and demand economics. Higher price levels lead to lower demand thereby lowering the house price to make the house more competitive. |

Show your code here:

|  |
| --- |
| import pandas as pd import numpy as np  # Can get from PyCharm terminal. from factor\_analyzer import FactorAnalyzer  # Read data. from sklearn.linear\_model import LinearRegression PATH = "../dataset/" CSV\_DATA = "usCityData.csv" data = pd.read\_csv(PATH + CSV\_DATA, sep=',')  # Create data frame without ID and Satisfaction columns. df = data.copy() # Columns # crim,zn,indus,nox,rm,age,dis,rad,tax,ptratio,lstat,medv  # Display all columns of the data frame. pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000) print(df.head(2))  # Bartlett's test of sphericity tests the hypothesis that your correlation matrix # is an identity matrix. If the correlation matrix is an identity matrix the # columns are unrelated and are therefore unsuitable for structure detection. # If the test is insignificant do not use factor analysis. from factor\_analyzer.factor\_analyzer import calculate\_bartlett\_sphericity chi\_square\_value, p\_value=calculate\_bartlett\_sphericity(df)  print("\nBartlett's test chi-square value: ") print(chi\_square\_value)  print("\nBartlett's test p-value: ") print(p\_value)  # Kaiser-Meyer-Olkin (KMO) test measures the proportion of variance among # variables that might be common variance. The lower the proportion, # the more suited your data is to Factor Analysis. Factor analysis is suitable # for scores of 0.6 (and sometimes 0.5) and above. from factor\_analyzer.factor\_analyzer import calculate\_kmo kmo\_all,kmo\_model=calculate\_kmo(df) print("\nKaiser-Meyer-Olkin (KMO) Test: Suitability of data for factor analysis.") print(kmo\_model)  # Create factor analysis and examine loading vectors and Eigenvalues. fa = FactorAnalyzer(rotation=None) fa.fit(df) print("\nFactors:") print(fa.loadings\_)  ev, v = fa.get\_eigenvalues() print("\nEignenvalues:") print(ev)  # Pick factors where eigenvalues are greater than 1. fa = FactorAnalyzer(rotation="varimax", n\_factors=3) fa.fit(df)  # Create formatted factor loading matrix. dfFactors = pd.DataFrame(fa.loadings\_) dfFactors['Categories'] = df.keys().values.tolist() dfFactors = dfFactors.rename(columns={0:'Factor 1',  1:'Factor 2', 2:'Factor 3'}) print("\nFactors: ") print(dfFactors)   # Display common variance. print("\nVariance Explanation") variances = fa.get\_factor\_variance() varianceDf = pd.DataFrame(data=variances) varianceDf = varianceDf.rename(columns={0:'F1',  1:'F2', 2:'F3'}) varianceDf['Totals'] = ['Eigenvalues', '% variance', 'cumulative variance'] print(varianceDf)  # Display RMSE, R^2, model coefficients and intercept. def showModelSummary(model, y\_test, X\_test\_tranformed):  print("\n\*\*\*\*\*\* MODEL SUMMARY \*\*\*\*\*\*")  pred = model.predict(X\_test\_tranformed)   # Show stats about the regression.  from sklearn.metrics import mean\_squared\_error  mse = mean\_squared\_error(y\_test, pred)  RMSE = np.sqrt(mse)  print("\nRMSE: " + str(RMSE))   from sklearn.metrics import r2\_score  print("\nR^2: ",r2\_score(y\_test,pred))   print("\nModel Coefficients:")  print(model.coef\_)   print("\nModel Intercept:")  print(model.intercept\_)  # Display p-values for model coefficients. def showCoefficientPValues(y\_train, X\_train\_transformed):  import statsmodels.api as sm  X2 = sm.add\_constant(X\_train\_transformed)  model = sm.OLS(y\_train, X2)  fii = model.fit()  p\_values = fii.summary2().tables[1]['P>|t|']  print("\nModel p-values: ")  print(p\_values)  # Split data before it is transformed. from sklearn.model\_selection import train\_test\_split X\_train, X\_test , y\_train, y\_test = train\_test\_split(df, data['medv'],  test\_size=0.3, random\_state=1) # Transform data with factor components. X\_train\_transformed = fa.fit\_transform(X\_train) X\_test\_tranformed = fa.transform(X\_test)  #---------------------------------------------------------- # Build first model #---------------------------------------------------------- # Train regression model on training data model = LinearRegression() model.fit(X\_train\_transformed, y\_train)  # Show model statistics. showModelSummary(model, y\_test, X\_test\_tranformed)  # Check coefficient significance. showCoefficientPValues(y\_train, X\_train\_transformed)  #---------------------------------------------------------- # Build second model without insignificant variable. #---------------------------------------------------------- # Builds labelled DataFrame with signficant latent variables from # factor matrix. def dropInsignificantX(X\_transformed):  # Builds DataFrame from matrix.  dfX = pd.DataFrame(data=X\_transformed)   # Labels columns and drops insignificant column.  dfX = dfX.rename(columns={0:'Purchase',  1:'Marking', 2:'Post Purchase', 3:'Product Position'})   del dfX['Post Purchase']  return dfX  # Prepare significant X values for regression. trainDF = dropInsignificantX(X\_train\_transformed) testDF = dropInsignificantX(X\_test\_tranformed)  # Train regression model on training data model = LinearRegression() model.fit(trainDF, y\_train)  # Show model statistics. showModelSummary(model, y\_test, testDF)  # Check coefficient significance. showCoefficientPValues(y\_train, trainDF.values)  print("FA Factor Variance") print(fa.get\_factor\_variance()) print("FA Get Communalities") print(fa.get\_communalities()) print("FA Uniqunesses") print(fa.get\_uniquenesses()) |

**Note:**

The Factor-Hair-Revised.csv sample had more clearly distinguished factors. Factors in the **usCityData.csv** data set may exhibit strong variance for the same feature in more than one factor.

## References

<https://stats.idre.ucla.edu/spss/seminars/introduction-to-factor-analysis/a-practical-introduction-to-factor-analysis/>

<https://datasciencetips.com/use-factor-analysis-to-better-understand-your-data/>

<https://www.datacamp.com/community/tutorials/introduction-factor-analysis>