Python Packages for Exploratory Factor Analysis

Isaiah Persson & Jam Khojasteh

STRUCTURAL EQUATION MODELING: A MULTIDISCIPLINARY JOURNAL
https://doi.org/10.1080/10705511.2021.1910037

REVIEW

Python Packages for Exploratory Factor Analysis

Isaiah Persson and Jam Khojasteh

Oklahoma State University

ABSTRACT
Exploratory Factor Analysis (EFA) is a widely used statistical technique for reducing data dimensionality and representing latent constructs via observed variables. Different software offer toolsets for performing this analysis. While Python's statistical computing ecosystem is less developed than that of R, it is growing in popularity as a platform for data analysis and now offers several packages that perform EFA. This article reviews EFA modules in the *statsmodels, FactorAnalyzer*, and *scikit-learn* Python packages. These packages are discussed with regard to official documentation, features, and performance on an applied example.

KEYWORDS
Exploratory factor analysis (EFA); Python; statsmodels; FactorAnalyzer; scikit-learn

Introduction

Factor analysis is commonly used for data reduction in academic fields of educational measurement and psychology to describe constructs that cannot be directly observed (e.g., intelligence and happiness), and it is also used in fields such as marketing research to measure customer attitudes and other industry-relevant latent variables (B2B International, 2021; Costello & Osborne, 2005; Pohlmann, 2004; Watkins, 2018). Specifically, exploratory factor analysis (EFA) is a common way to model observed items (i.e., variables) in terms of a smaller number of unobserved factors (i.e., latent constructs) (Fabrigar et al., 1999; Watkins, 2018). This procedure essentially expresses each observed value as a linear combination of different factors plus error (Fabrigar et al., 1999; Preacher et al., 2013; Watkins, 2018). Each item's observed variance is partitioned into communality, variance that is shared with other items and explained by underlying factors, and uniqueness, also known as error, noise, or unexplained variance.

Python has become an important language within the data analytic community (Ayer et al., 2014; Bajuk, 2019). While the R programming language has dominated statistical computing in academic research due to its well-developed ecosystem of specialized packages, Python has emerged as an alternative language that offers versatility and integrates well with other applications (Bajuk, 2019; Ozgur et al., 2017). According to the major software development and version control site GitHub, Python is the second most popular language for software development and the preferred language for developing machine learning applications among its user-base (Elliot, 2019; GitHub, 2020). In addition, many users regard Python as one of the simplest programming languages to learn (Ayer et al., 2014). Given its pervasive use and ease of integration, researchers may benefit from familiarizing themselves with Python. Using this language for analyses will make academic research more accessible to application developers in industry, thus enhancing the probability for collaboration and exchange across domains. For those who wish to use Python for EFA, there are a few packages available. While multiple publications have discussed conducting EFA with R packages, it seems that there has been no such endeavor regarding Python packages (Kabacoff, 2011, pp. 342–351; Luo et al., 2019; Mair, 2018; pp. 23–34). This article attempts to close this gap by reviewing the statsmodels (version 0.12.2), FactorAnalyzer (version 0.3.2), and scikit-learn (version 0.24.2) packages, with respect to their documentation, features, and performance using a sample dataset (Biggs & Madnani, 2019; Pedregosa et al., 2011; Perktold et al., 2010).

Python packages and review framework

Currently, at least three Python packages (i.e., *statsmodels, FactorAnalyzer*, and *scikit-learn*) offer modules for conducting exploratory factor analyses (Biggs & Madnani, 2019; Pedregosa et al., 2011; Perktold et al., 2010). To start, this article reviews each package's official documentation for overall clarity and comprehensiveness and discusses the availability of informal resources on platforms such as blogs and forums. Then, the packages are compared based on their documented features, after which they are tested by conducting an EFA with each package on a sample dataset. Finally, this article concludes by providing recommendations to users and package developers.

All analyses with Python (version 3.8.8) are run in Jupyter Notebook (version 6.3.0) within the Anaconda open-source toolkit (Anaconda Inc., 2020; Project Jupyter, 2020). Jupyter Notebook is an integrated development environment (IDE), similar to RStudio, which operates as a web application and allows users to seemlessly edit, run, and present code (Project Jupyter, 2020; RStudio Team, 2020). Necessary software packages for this paper's analyses are retrieved and managed via Anaconda. For those who are unfamiliar with this toolkit, Anaconda "makes it easy to manage multiple data environments that can be maintained and run separately without interference from each other" (Anaconda Inc., 2020). Along with the three primary packages, users may also need to load

CONTACT Isaiah Persson isaiah.persson@okstate.edu

supporting packages, such as *pandas, NumPy, SciPy* and *Matplotlib* to manipulate and visualize data (Harris et al., 2020; Hunter, 2007; Krekel & Pytest-Dev Team, 2020; Smith, 2015; The Pandas Development Team, 2020; Virtanen et al., 2020). Each of the reviewed packages provides further information in their official documentation concerning software dependencies that are necessary to run the EFA modules. Figure 1 displays the code that loads the necessary primary and supporting packages for the analyses discussed in this article.[1]

Overview of Python packages with EFA capabilities

statsmodels is an expansive package in Python "that provides classes and functions for the estimation of many different statistical models" (Perktold et al., 2010). The package's authors attempt to accommodate individuals who are familiar with programming in R, by allowing users to define model variables for many statistical functions and classes with R-style formulas. The "Getting Started" and "User Guide" sections of the *statsmodels* website provide an introduction to this and general guidance on how to use the package. The *statsmodels* documentation details the input parameters that one may specify for the class that estimates an EFA model along with ways to report and modify the results. The documentation provides a thorough outline of intended functionality and limitations. Unfortunately, there are no examples of the code being implemented on a dataset. This may hinder someone who is new to this package or to Python, resulting in a trial and error process.

FactorAnalyzer, as the name suggests, is a package developed by ETS solely for performing exploratory factor analysis and confirmatory factor analysis (CFA) (Biggs & Madnani, 2019). The official documentation provides a clear and concise explanation of factor analysis and its application to modeling and measuring latent variables via observed variables. This is followed by instructions on how to use each of the package's modules for EFA and CFA. The package's documentation explains its EFA and CFA toolset in terms of psychometric application and provides a conceptual overview that avoids mathematical terminology and equations.

scikit-learn is one of the most comprehensive and influential machine learning packages in the Python programming ecosystem. Along with the other two packages, it provides a purpose-built class that performs EFA. The package's "User Guide" provides a conceptual overview of EFA that focuses on mathematical descriptions, presenting it as an alternative to principal components analysis (PCA) for matrix factorization. The code documentation outlines how to implement the EFA class, however many of the parameters and attributes are described with machine learning terminology that may not be familiar to users from behavioral and social sciences. The examples that the package uses, such as image processing, focus on predictive accuracy over interpretable model building. While informative, *scikit-learn's* approach may seem less relatable and even a bit inaccessible to users from backgrounds other than machine learning.

Informal documentation and help from user-base

There are a number of blogs and web tutorials that demonstrate how to perform EFA with Python. Most of these utilize the *FactorAnalyzer* package and, to a lesser extent, *scikit-learn*. A cursory web search did not find any user examples of EFA performed with *statsmodels*. This may reflect the limited popularity of Python for statistical computing compared to R. Due to this relative lack of popularity, it is difficult to find user-generated solutions when dealing with implementation challenges.

Package features

Next, each package's documented functionality is reviewed, by comparing input data requirements (e.g., raw datasets or correlation matrices), tests of assumptions, estimation methods, tools for choosing factors (e.g., scree plots and eigenvalue tables), rotation options, and reporting formats.

Specifying an EFA model

Each of the three packages provides a purpose-built class for specifying parameters and estimating an EFA model (see

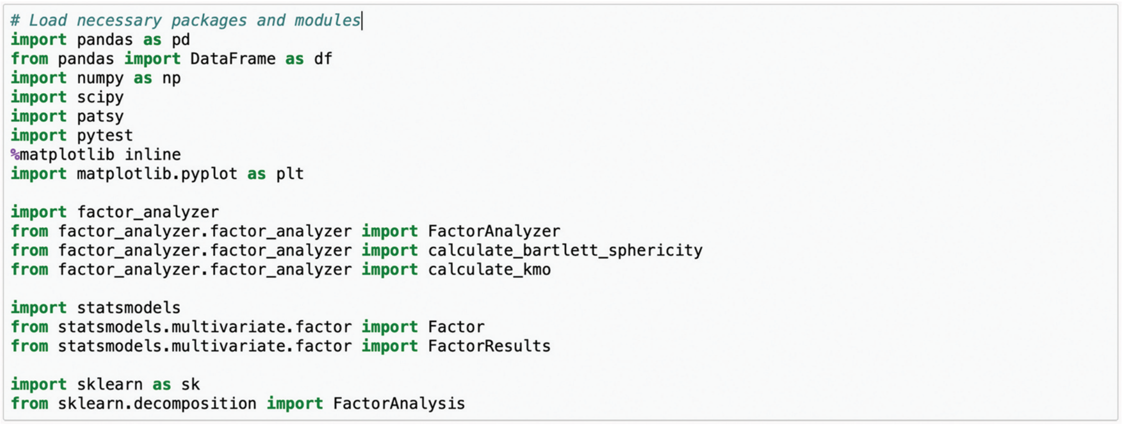
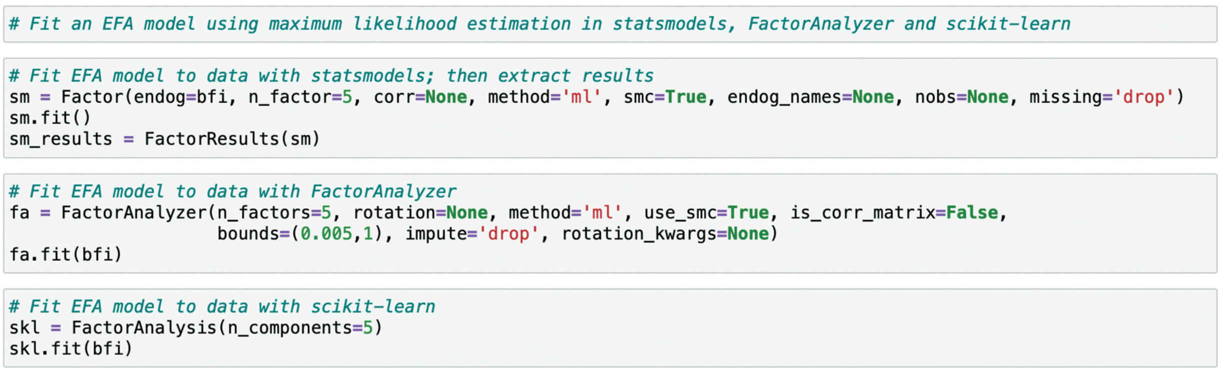

```
# Load necessary packages and modules
import pandas as pd
from pandas import DataFrame as df
import numpy as np
import scipy
import patsy
import pytest
%matplotlib inline
import matplotlib.pyplot as plt

import factor_analyzer
from factor_analyzer.factor_analyzer import FactorAnalyzer
from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
from factor_analyzer.factor_analyzer import calculate_kmo

import statsmodels
from statsmodels.multivariate.factor import Factor
from statsmodels.multivariate.factor import FactorResults

import sklearn as sk
from sklearn.decomposition import FactorAnalysis
```

Figure 1. This screenshot shows all the Python packages and modules for performing EFA in this article.

[1]The Python and R code that support the findings of this study are openly available on the Open Science Framework website (DOI: 10.17605/OSF.IO/XPMUZ).



```
# Fit an EFA model using maximum likelihood estimation in statsmodels, FactorAnalyzer and scikit-learn
```

```
# Fit EFA model to data with statsmodels; then extract results
sm = Factor(endog=bfi, n_factor=5, corr=None, method='ml', smc=True, endog_names=None, nobs=None, missing='drop')
sm.fit()
sm_results = FactorResults(sm)
```

```
# Fit EFA model to data with FactorAnalyzer
fa = FactorAnalyzer(n_factors=5, rotation=None, method='ml', use_smc=True, is_corr_matrix=False,
                    bounds=(0.005,1), impute='drop', rotation_kwargs=None)
fa.fit(bfi)
```

```
# Fit EFA model to data with scikit-learn
skl = FactorAnalysis(n_components=5)
skl.fit(bfi)
```

Figure 2. This screen shot shows the code used to specify and fit an EFA model using maximum likelihood estimation in *statsmodels*, *FactorAnalyzer*, and *scikit-learn*.

Figure 2). *FactorAnalyzer* and *scikit-learn* allow users to retrieve results directly from the fitted class, by calling attributes and methods that are associated with it. On the other hand, *statsmodels* requires users to then specify a separate class that uses the fitted model as its only parameter, to retrieve results. Figure 2 displays examples of code from each package for fitting an EFA model to data.

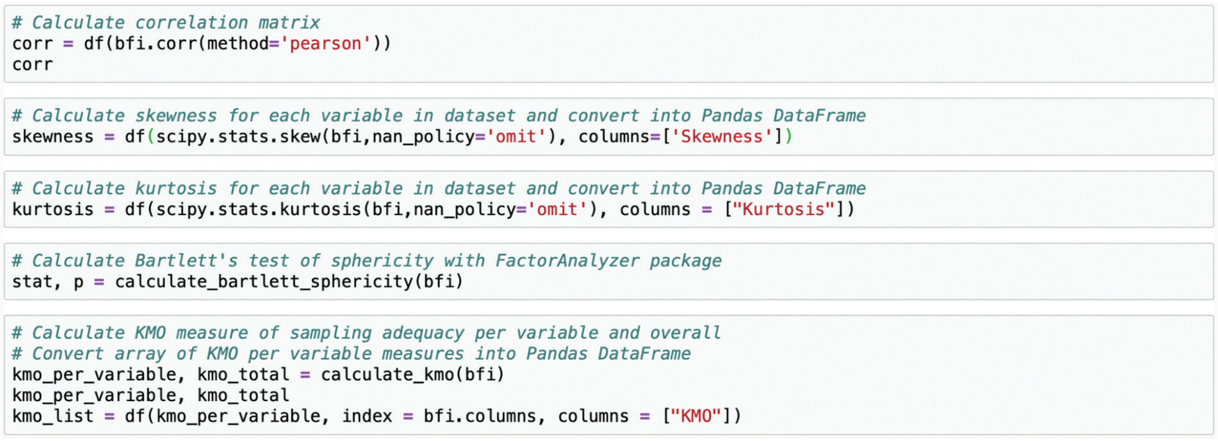
Input data

All three packages allow users to conduct an EFA on a raw dataset with observations organized by row and items (i.e., variables) by column. Alternatively, *FactorAnalyzer* and *statsmodels* also give users the option of using a correlation matrix as input data. As shown in Figure 2, *FactorAnalyzer* and *scikit-learn* require users to enter the dataset as a parameter to the ".fit()" method after specifying the other class parameters, whereas *statsmodels* requires the dataset to be specified as a parameter within the class.

Testing assumptions

Before starting an EFA, it is necessary to test basic assumptions about the data. One should evaluate measures of normality (i.e., skewness and kurtosis), the Kaiser-Meyer-Olkin (KMO) measure of sampling adequacy, and Bartlett's test of sphericity. Unfortunately, none of the packages provides a comprehensive set of functions or classes to test assumptions for EFA. While *statsmodels* provides a class for users to calculate descriptive statistics, such as skewness and kurtosis, an error message is generated when executing the code and little guidance is found from the official documentation or from searching user platforms. Neither *FactorAnalyzer* nor *scikit-learn* offer the option to generate descriptive statistics. Rather, users must look to other packages, such as *Scipy* or *pandas* to obtain these figures. *Scipy's* kurtosis and skewness functions are clear and easy to implement.

FactorAnalyzer uniquely provides classes to compute Bartlett's test of sphericity and the KMO test for sampling adequacy. After much searching, it appears that this is the only Python package that provides a built-in approach to calculate both test statistics. *FactorAnalyzer* also provides a built-in attribute to the "FactorAnalyzer()" class that computes a correlation matrix for the original data, which can be called by attaching ".corr_" as a suffix to the class command. Alternatively, one can simply call the *pandas* "corr()" function on the data frame being analyzed, to generate the data correlation matrix. Figure 3 demonstrates how to test the assumptions using *Scipy, FactorAnalyzer* and *pandas*.



```
# Calculate correlation matrix
corr = df(bfi.corr(method='pearson'))
corr
```

```
# Calculate skewness for each variable in dataset and convert into Pandas DataFrame
skewness = df(scipy.stats.skew(bfi,nan_policy='omit'), columns=['Skewness'])
```

```
# Calculate kurtosis for each variable in dataset and convert into Pandas DataFrame
kurtosis = df(scipy.stats.kurtosis(bfi,nan_policy='omit'), columns = ["Kurtosis"])
```

```
# Calculate Bartlett's test of sphericity with FactorAnalyzer package
stat, p = calculate_bartlett_sphericity(bfi)
```

```
# Calculate KMO measure of sampling adequacy per variable and overall
# Convert array of KMO per variable measures into Pandas DataFrame
kmo_per_variable, kmo_total = calculate_kmo(bfi)
kmo_per_variable, kmo_total
kmo_list = df(kmo_per_variable, index = bfi.columns, columns = ["KMO"])
```

Figure 3. This screenshot shows the Python functions used to calculate a correlation matrix, skewness, kurtosis, Bartlett's test of sphericity, and the KMO measure of sampling adequacy.

Estimation methods

Each of the packages offers different methods for estimating an EFA model. For example, *statsmodels* offers both principal axis factoring and maximum likelihood, however the latter method does not return eigenvalues for the correlation matrix. *FactorAnalyzer* documentation says that it offers minimum residual, maximum likelihood and "principal factor extraction" methods. While a search of EFA literature reveals no mention of the last method, we presume that it refers to principal axis factoring. The *scikit-learn* package offers the smallest amount of flexibility by utilizing only the maximum likelihood method, which it applies by default. For *statsmodels* and *FactorAnalyzer*, a user simply selects the estimation method as an argument in the class that fits an EFA model.

Choosing the number of factors

Scree plots, original eigenvalues (i.e., Kaiser criterion), and parallel analyses are some of the commonly used methods for determining the number of factors in a model (Costello & Osborne, 2005, pp. 1-2; Fabrigar et al., 1999; Watkins, 2018). Surprisingly, only the *statsmodels* "FactorResults()" class provides both an attribute for listing eigenvalues and a method for visualizing them in a scree plot. FactorAnalyzer returns only eigenvalues via an attribute of its "FactorAnalyzer()" class, while scikit-learn's "FactorAnalysis()" class provides neither eigenvalues nor scree plot. None of these packages offers a method to estimate the number of factors via parallel analysis.

Rotation methods

FactorAnalyzer and *statsmodels* provide considerable functionality for performing factor rotations. They both offer varimax, promax, oblimin, quartimax, and equamax rotations. Individually, *FactorAnalyzer* provides oblimax, quartimin, "geomin_obl", and "geomin_ort" rotations, while *statsmodels* offers biquartimax, parsimax, parsimony, biquartimin options. Of important note, the *statsmodels* documentation warns that only "'varimax', 'quartimax' and 'oblimin' are verified against R or Stata," while the other rotations may produce different results (Perktold et al., 2010). *scikit-learn* again provides the fewest options, by offering only varimax and quartimax rotations. The *FactorAnalyzer* and *scikit-learn* packages allow users to specify the rotation method as a parameter in their respective classes for fitting an EFA model, whereas statsmodels requires users to do this via the ".rotate()" method from the "FactorResults()" class. In addition, *FactorAnalyzer* and *statsmodels* each offer an optional class for performing rotations on an already fitted model. See Table 1 for a summary of EFA capabilities listed in each package.

Reporting options

When it comes to reporting results, *statsmodels* offers a very convenient summary function that returns data frames for the eigenvalues, communalities, pre-rotation loading matrix, and post-rotation loading matrix, much like the output from the fitted "fa()" function in R's *psych* package (Revelle, 2020). These and other results can be individually called and returned as data arrays in both *FactorAnalyzer* and *statsmodels*. One can then use *pandas* to convert the arrays into data frames, to help interpret results. For oblique rotations, *FactorAnalyzer* returns both pattern and structure matrices while *statsmodels* only offers rotated factor "loadings" without specifying which type. *FactorAnalyzer's* documentation also mentions a ".psi_" attribute, which purportedly returns the factor correlation matrix for oblique rotations. However, the command only returns an error message stating that the "FactorAnalyzer()" class has no such attribute. Of the three packages, *scikit-learn* offers the fewest class attributes for returning conventional EFA model results. While it provides methods for reporting a loading matrix, factor scores for observations, and a reproduced covariance matrix, the package does not list any methods for reporting eigenvalues or communalities.

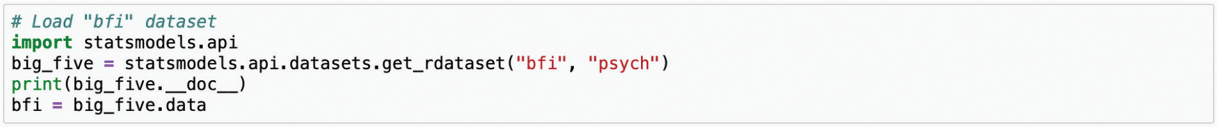
Applied example

In addition to examining each package's intended functionalities, it is important to evaluate their use in application. To accomplish this, an EFA is conducted with *FactorAnalyzer*, *statsmodels* and *scikit-learn* on the same dataset and results are compared against output from a similar analysis with R's *psych* package (Revelle, 2020).

Table 1. EFA capabilities listed within each Python package.

Capability	*statsmodels*	*FactorAnalyzer*	*scikit-learn*
Input Data	Raw dataset (observation by row & item by column) & correlation matrix	Raw dataset (observation by row & item by column) & correlation matrix	Raw dataset (observation by row & item by column)
Test of Assumptions	Descriptive statistics (including kurtosis & skewness)	Correlation matrix, Kaiser-Meyer-Olkin (KMO) test, & Bartlett's test of sphericity	None
Estimation Methods	Principal axis factoring & maximum likelihood	Minimum residual, maximum likelihood, & principal factor extraction*	Maximum likelihood
Methods for Identifying Number of Factors	Eigenvalues & scree plot	Eigenvalues	None
Rotation Methods	Varimax, quartimax, biquartimax, equamax, oblimin, parsimax, parsimony, biquartimin, & promax	Varimax, promax (default), oblimin, oblimax, quartimin, quartimax, equamax, geomin_obl, & geomin_ort	Varimax & quartimax

*"Principal factor extraction" appears to be alternative name for principal axis factoring.



```
# Load "bfi" dataset
import statsmodels.api
big_five = statsmodels.api.datasets.get_rdataset("bfi", "psych")
print(big_five.__doc__)
bfi = big_five.data
```

Figure 4. This screenshot shows the code from *statsmodels* that loads the "bfi" dataset, on which the three Python packages are tested in relationship to R's *psych* package.

Example dataset

In order to easily compare results, the same analysis is attempted with each of the three Python packages on the "bfi" dataset from R's *psych* package (Revelle, 2020). The *statsmodels* package provides convenient access to this file through a function that retrieves and loads any of the datasets from R packages that have been aggregated via the *Rdatasets* project (Arel-Bundock, 2020; Perktold et al., 2010). Figure 4 displays the code that loads this as a data frame and that returns a document describing the data.

The resulting data frame is prepared for analysis by using functions from the *pandas* data manipulation package to remove the three demographic columns (The Pandas Development Team, 2020). The remaining data frame contains 25 columns of items and 2800 rows of observations.

Example results

Results from the three packages, given similar model specifications, show varying levels of performance. The initial eigenvalues and scree plot are computed via principal axis factoring in *statsmodels*, since the package does not compute eigenvalues when applying maximum likelihood estimation. Likewise, *FactorAnalyzer's* "principal factor extraction" method is used to estimate initial eigenvalues. This portion of the example analysis is not performed with *scikit-learn*, since that package's "FactorAnalysis()" class does not output eigenvalues.

Upon comparison with results from R's *psych* package, the *statmodels* eigenvalue attribute and scree plot function evidently return the reproduced rather than initial eigenvalues. This oversight effectively prevents users from utilizing the tool to aid in choosing the number of factors for the model. *FactorAnalyzer's* "principal factor extraction" method returns eigenvalues that are similar, but not identical, to those estimated via the principal axis factoring method in the *psych* package. The discrepancy is large enough to question whether "principal factor extraction" is a different estimation method altogether.

After observing the initial eigenvalues, a five-factor model is specified with each of the packages using maximum likelihood estimation and varimax rotation. The packages are subsequently assessed by comparing the resulting factor loading matrices with the corresponding output from a similarly specified model in R's psych package. FactorAnalyzer maintains a strong performance, by returning factor loadings that closely resemble those from the psych package. Interestingly, statsmodels and scikit-learn each returns a loading matrix that uniquely differs from the other packages' matrices.

Conclusion

The three Python packages offer a range of functionality, both in specifying and reporting EFA models. The *FactorAnalyzer* and *statsmodels* packages present large toolsets for conducting EFA and reporting interpretable measurement models. On the other hand, *scikit-learn* offers more limited EFA functionality that seems to be primarily geared toward reducing dimensionality of data and enhancing predictive capabilities of machine learning models.

At present, *FactorAnalyzer* stands out as the most comprehensive and reliable Python package for conducting EFA, because it offers necessary tests of assumptions that are overlooked by other packages and its EFA results align with those from the *psych* package in R. While *statsmodels'* documentation describes similar functionality for EFA, it struggles to deliver accurate results. Finally, *scikit-learn* comes in at third place due to its limited set of options for estimating, modifying and reporting EFA models.

Regarding ease-of-use, package developers should consider adding the option to output results as data frames instead of arrays. Data frames are more interpretable since they organize data into tabular form with descriptive metadata such as column and row names. The *statsmodels* package outperforms the others in this regard, by offering a summary function that returns a selection of commonly reported EFA results as data frames like the *psych* package does in R. Otherwise, users must use the *pandas* package to convert the packages' output arrays into a data frame format. While not particularly difficult, manually converting arrays into data frames with corresponding metadata can present an unnecessarily tedious intermediate step in the data analytic workflow. This extra step may discourage new Python users who are not well-versed in data manipulation techniques and users who wish to perform quick analyses with minimal code modification.

In terms of intended functionality, all of the packages could be improved by adding methods for identifying the optimal number of factors for EFA models. None of the packages provide tools for conducting a parallel analysis, and only one offers a scree plot function. Fortunately, programming a scree plot does not require extensive coding ability or custom-built functions. Users who wish to code a scree plot using eigenvalues from their output can try the *Matplotlib* visualization package, which has a number of online tutorials (Hunter, 2007; Navlani, 2019; St-Amant, 2020; Toth, 2020).

Lastly, we recommend that package developers verify that all the methods work as stated, given the discrepancies between documented functionality and results. These package errors are probably simple oversights by the developers and can presumably be fixed with a few lines of code or clearer documentation.

For example, both statsmodels and scikit-learn return loading matrices that differ significantly from each other and the psych package's output even though the same estimation and rotation methods are used. Likewise, FactorAnalyzer's ".psi_" attribute for reporting the factor correlation matrix returns an error message and statsmodels's scree plot function visualizes the wrong set of eigenvalues. Such methods should be tested to make sure they return values that align with results from established programs such as R's *psych* package. Hopefully, development and quality control will accelerate as more users integrate these packages into their data analytic projects and contribute insights from their experiences.

Disclosure statement

We have no conflicts of interest to report.

References

Anaconda Inc. (2020). *Individual edition*. Anaconda. Retrieved January 1, 2021, from https://www.anaconda.com/products/individual

Arel-Bundock, V. (2020). A collection of datasets originally distributed in various R packages. *Rdatasets*. Vincent Arel-Bundock. Retrieved January 1, 2021, from https://vincentarelbundock.github.io/Rdatasets/

Ayer, V., Miguez, S., & Toby, B. (2014). Why scientists should learn to program in Python. *Powder Diffraction*, *29*, S48–S64. https://doi.org/10.1017/S0885715614000931

B2B International. (2021). *Factor analysis in marketing research*. B2B International. Retrieved February 1, 2021, from https://www.b2binternational.com/research/methods/statistical-techniques/factor-analysis/

Bajuk, L. (2019, December 17). *R vs. Python: What's the best language for data science?* R Studio Blog. Retrieved February 1, 2021, from R Studio Blog https://blog.rstudio.com/2019/12/17/r-vs-python-what-s-the-best-for-language-for-data-science/

Biggs, J., & Madnani, N. (2019). *Factor_analyzer documentation*. Release 0.3.1. Jeremy Biggs. Retrieved February 2, 2021, from http://factor-analyzer.readthedocs.io/en/latest/index.html

Costello, A. B., & Osborne, J. (2005). Best practices in exploratory factor analysis: Four recommendations for getting the most from your analysis. *Practical Assessment, Research, and Evaluation*, *10*, Article 7. https://doi.org/10.7275/jyj1-4868

Elliot, T. (2019, January 24). *The state of the octoverse: Machine learning*. The GitHub Blog. Retrieved February 1, 2020, from https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/

Fabrigar, L. R., Wegener, D. T., MacCallum, R. C., & Strahan, E. J. (1999). Evaluating the use of exploratory factor analysis in psychological research. *Psychological Methods*, *4*, 272–299. https://doi.org/10.1037/1082-989X.4.3.272

GitHub. (2020). *The 2020 state of the octoverse*. GitHub. Retrieved February 1, 2021, from https://octoverse.github.com

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*, 90–95. https://doi.org/10.1109/MCSE.2007.55

Kabacoff, R. (2011). *R in action*. Shelter Island, NY: Manning publications.

Krekel, H. and Pytest-Dev Team. (2020). Full pytest documentation. *pytest*. Retrieved February 2, 2021, from https://docs.pytest.org/en/stable/contents.html

Luo, L., Arizmendi, C., & Gates, K. M. (2019). Exploratory factor analysis (EFA) programs in R. *Structural Equation Modeling*, *26*, 819–826. https://doi.org/10.1080/10705511.2019.1615835

Mair, P. (2018). *Modern psychometrics with R*. Springer. https://doi.org/10.1007/978-3-319-93177-7

Navlani, A. (2019, April). *Introduction to factor analysis in Python*. datacamp. DataCamp, Inc. Retrieved February 2, 2021, from https://www.datacamp.com/community/tutorials/introduction-factor-analysis

Ozgur, C. C., Rogers, G., Hughes, Z., & Myer-Tyson, E. (2017). MatLab vs. Python vs. R. *Journal of Data Science*, *15*, 355–371. https://doi.org/10.6339/JDS.201707_15(3).0001

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

Pohlmann, J. T. (2004). Use and interpretation of factor analysis in the Journal of Educational Research: 1992–2002. *The Journal of Educational Research*, *98*, 14–23. https://doi.org/10.3200/JOER.98.1.14-23

Preacher, K. J., Zhang, G., Kim, C., & Mels, G. (2013). Choosing the optimal number of factors in exploratory factor analysis: A model selection perspective. *Multivariate Behavioral Research*, *48*, 28–56. https://doi.org/10.1080/00273171.2012.710386

Project Jupyter. (2020, November 18). Home. Jupyter. Project Jupyter. Retrieved January 1, 2021, from https://jupyter.org

Revelle, W. (2020). psych: Procedures for Psychological, Psychometric, and Personality Research. R package version 2.0.12. Northwestern University, Evanston, IL. http://cran.r-project.org/package=psych

RStudio Team. (2020). *RStudio: Integrated development for R*. RStudio, PBC. Retrieved February 19, 2021, from http://www.rstudio.com/ .

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. *Proceedings of the 9th Python in Science Conference*. https://doi.org/10.25080/Majora-92bf1922-011

Smith, N. (2015). *patsy – Describing statistical models in Python*. Authored/published by Nathaniel J. Smith. Retrieved February 2, 2021, from https://patsy.readthedocs.io/en/latest/index.html

St-Amant, F. (2020, May 13). *Factor analysis tutorial*. Towards data science. Retrieved February 2, 2021, from https://towardsdatascience.com/factor-analysis-a-complete-tutorial-1b7621890e42

The Pandas Development Team. (2020, February). *pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.3509134

Toth, G. (2020). *Factor analysis*. DataSklr. Retrieved Febuary 2, 2021, from Mair, 2018https://www.datasklr.com/principal-component-analysis-and-factor-analysis/factor-analysis

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Watkins, M. W. (2018). Exploratory factor analysis: A guide to best practice. *Journal of Black Psychology*, *44*, 219–246. https://doi.org/10.1177/0095798418771807