

|  |
| --- |
| Technical documentation  Open Source recommender |
|  |

|  |  |
| --- | --- |
| Abbreviations and Acronyms | |
| SVD | Singular Value Decomposition |
| MAP | Mean Average Precision |

Contents

[1 Introduction 4](#_Toc482275288)

[2 Architecture overview 4](#_Toc482275289)

[3 Technical prerequisites 5](#_Toc482275290)

[3.1 Knime installation GUIDE 5](#_Toc482275291)

[3.2 Python installation guide 8](#_Toc482275292)

[3.3 NOTE 11](#_Toc482275293)

[4 how to use the tool 12](#_Toc482275294)

[4.1 importing the workflow file 12](#_Toc482275295)

[4.2 set Input data 12](#_Toc482275296)

[4.3 set OUTPUT DATA location 12](#_Toc482275297)

[4.4 execution 12](#_Toc482275298)

[5 Workflow elements 13](#_Toc482275299)

[5.1 Annotations and nodes 13](#_Toc482275300)

[5.2 Node settings 13](#_Toc482275301)

[5.2.1 File Reader 13](#_Toc482275302)

[5.2.2 Column Filter 14](#_Toc482275303)

[5.2.3 Date Handler 14](#_Toc482275304)

[5.2.4 Sorter 14](#_Toc482275305)

[5.2.5 Column Rename 14](#_Toc482275306)

[5.2.6 Rule-based Row Filter 14](#_Toc482275307)

[5.2.7 Column Filter 14](#_Toc482275308)

[5.2.8 GroupBy 15](#_Toc482275309)

[5.2.9 File Reader 15](#_Toc482275310)

[5.2.10 Table Row to Variable 15](#_Toc482275311)

[5.2.11 Python Script (1 => 2) 15](#_Toc482275312)

[5.2.12 Column Resorter 15](#_Toc482275313)

[5.2.13 Variable to Table Row 15](#_Toc482275314)

[5.2.14 Row Filter 15](#_Toc482275315)

[5.2.15 CSV Writer 1 15](#_Toc482275316)

[5.2.16 CSV Writer 2 15](#_Toc482275317)

[6 Python script overview 16](#_Toc482275318)

[7 Notes 19](#_Toc482275319)

[7.1 Licensing 19](#_Toc482275320)

# Introduction

The Open Source Recommender is a KNIME solution, which generates product recommendations using historical point-of-sales data. Its main purpose is to identify which products are likely to be sold in a store with a certain level of confidence. The approach is similar to collaborative filtering, where the goal is to predict user preferences (ratings) for items.

In this tool, stores represent users and purchased quantities serve as preference indicators for a product. Recommendations are generated by an embedded Python script, which utilizes a form of matrix factorization, called truncated singular value decomposition. The output of the workflow is a table with scored recommendations for each store. In addition, users can optionally group the stores using k-means clustering.

The Open Source Recommender is released under MIT License and it is publicly available on Github.

# Architecture overview

The KNIME workflow consists of three main modules: an ETL component for data preparation, one Python recommender engine, and a post-processing component. The workflow only interacts with local files on the user’s computer.

Recommedations

**KNIME workflow**

Transactions

Post Processing Component

Python Recommender Component

ETL

component

Figure 1. High-level overview of Open Source Recommender architecture.

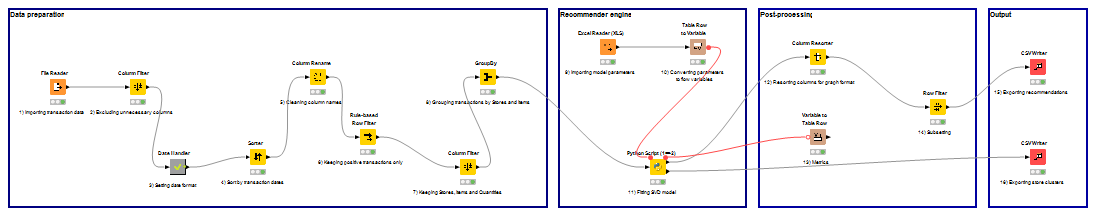


Figure 2. Outline of the KNIME workflow.

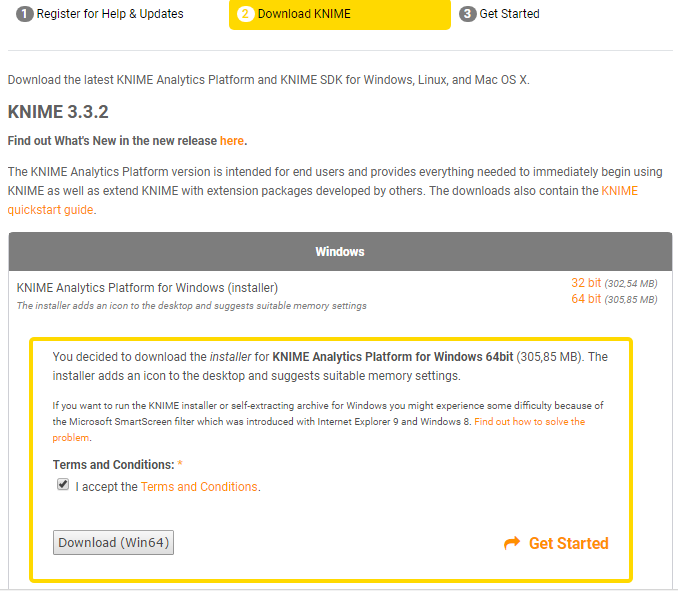
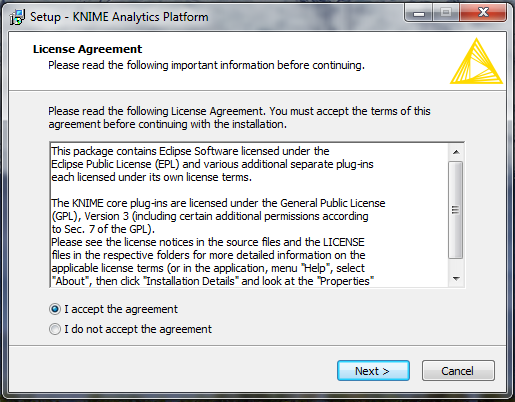
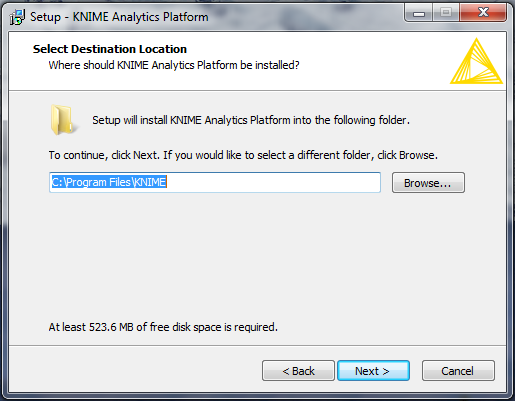
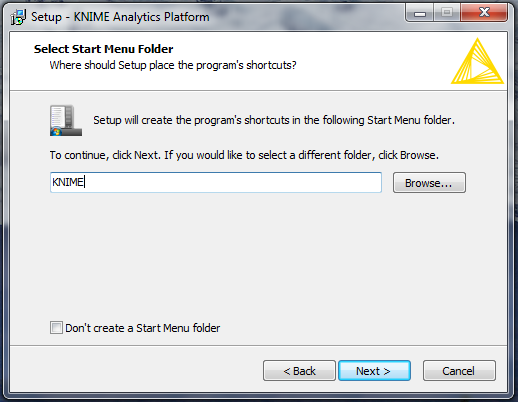
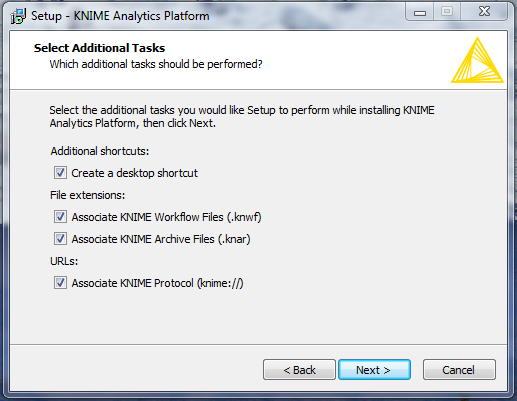
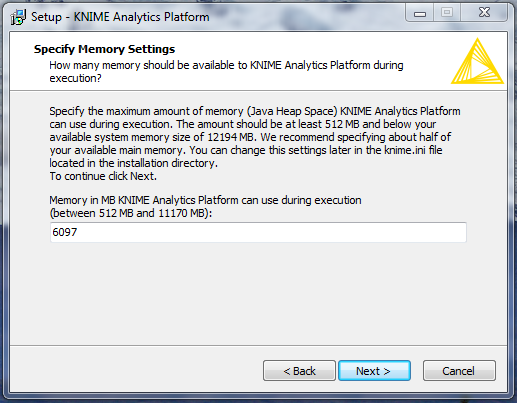
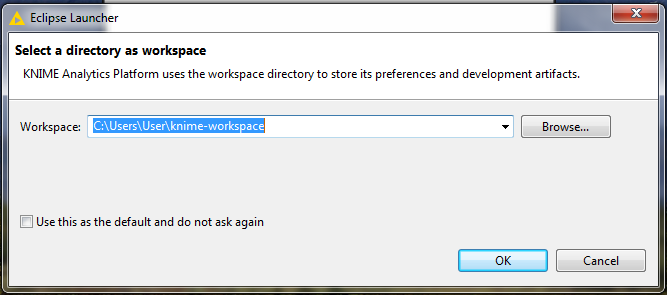
# Technical prerequisites

The open source recommender has the following software prerequisites:

* KNIME Analytics Platform (version 3.3.2 or higher)
* Python (version 2.7)

## Knime installation GUIDE

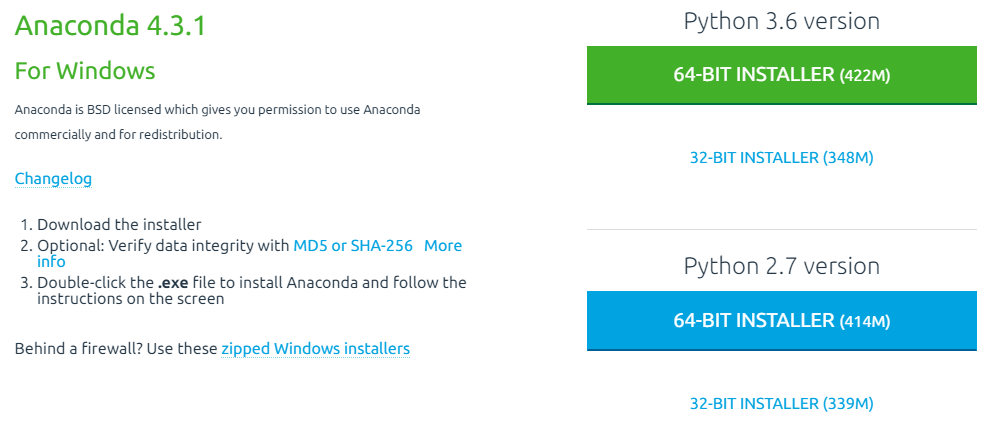
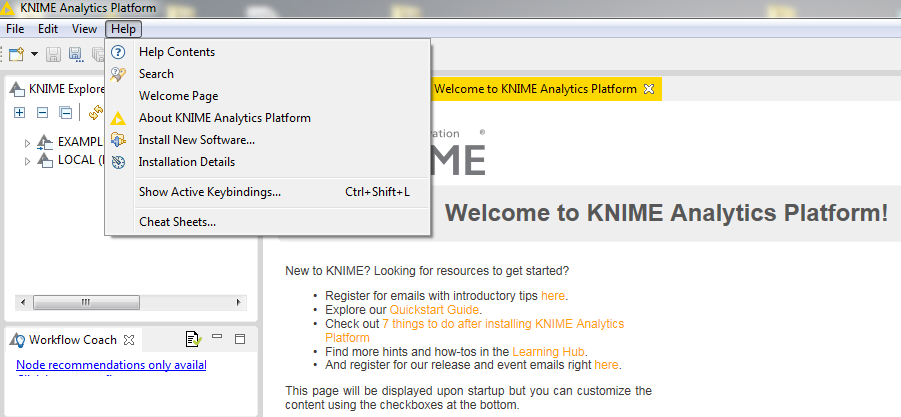
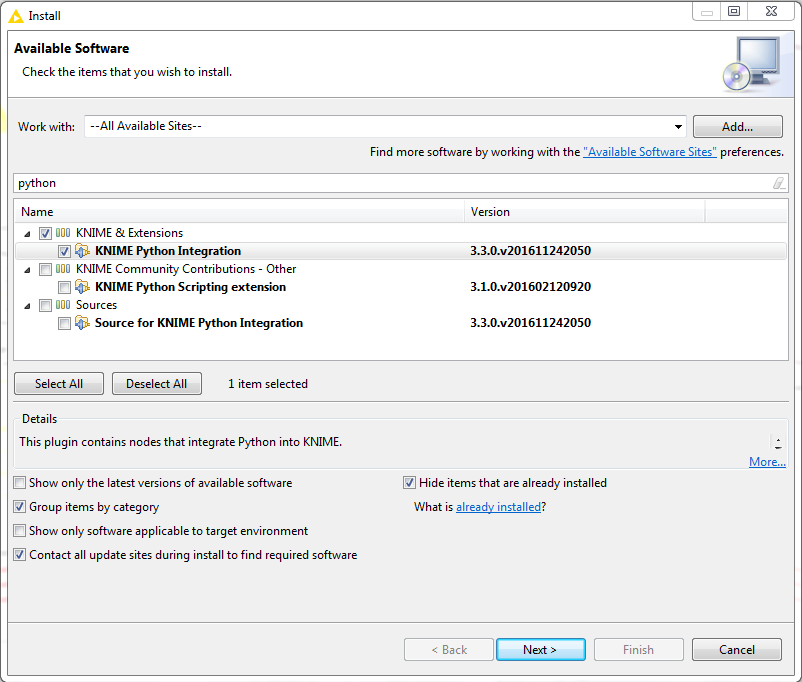
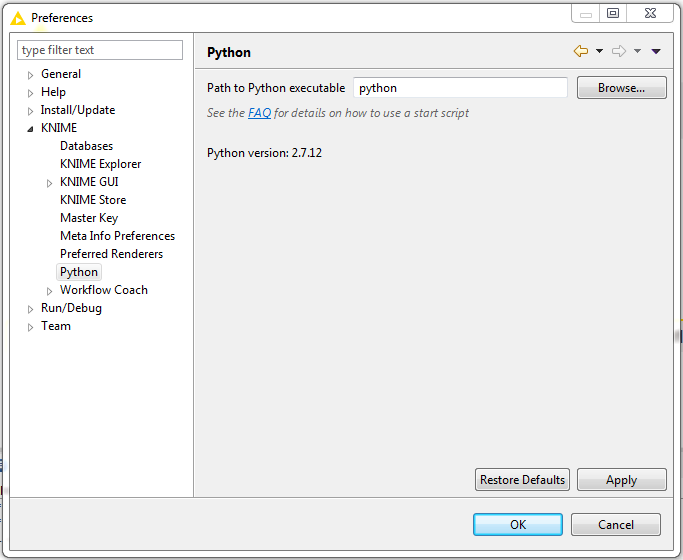
Installation steps:

1. Download KNIME Analytics Platform for Windows from the official website: <http://www.knime.org/downloads/overview>.   
     
   On downloads page, click **Download KNIME** (the first step “Register for Help & Updates” can be skipped) then choose a suitable installer.  
   Note: installing KNIME with all free extensions may increase application startup time.
2. Run the downloaded KNIME installer.
3. Accept the License Agreement.
4. Select destination location.   
     
   
5. Select start menu folder.
6. Select additional tasks.
7. Specify memory limit in MB. By default, it is half of the available physical memory.
8. After installation, launch KNIME and choose a directory as workspace.

## Python installation guide

It is recommended to install the Anaconda platform since it contains the most important Python packages for data manipulation.

Installation steps:

1. Download the latest Anaconda installer with Python 2.7 from the following website: <https://www.continuum.io/downloads#windows>
2. Install Anaconda using the downloaded .exe file.
3. Open KNIME, click Help on the menu bar and select Install New Software.
4. On the dialog panel, type “python” into the search bar and select **KNIME Python Integration** in KNIME & Extensions category.   
     
   
5. Click next, then follow the instructions.
6. Once the Python extension has been installed, restart KNIME.
7. Go to File and select Preferences. On the dialog panel click KNIME and select Python. If the installation is process successful, Python version will be shown below the executable.

## NOTE

In some cases, Python nodes can not be executed if the google-protobuf package is missing. To solve this error, users should take the following the steps:

1. Open start menu and launch CMD.

2. Write the following statement into the command line: pip install protobuf.

3. Wait until the installation process is completed.

4. Restart KNIME.

5. In case a package is still missing, repeat the installation process using *pip install <name of missing package>* command.

# how to use the tool

## importing the workflow file

The Open Source Recommender can be downloaded as a knwf file. It can be imported by selecting *File -> Import KNIME Worfklow*… in the menu bar.

## set Input data

The workflow needs two data tables:

Table of transactions (at node #1)

Table of model parameters (at node #9)

There are no restrictions on the format of the transactions table; however, it should have at least four columns, including TRANS\_DATE, DMS\_STORE\_ID, PGITEMCODE, OFFTAKE and TOTAL\_PIECE.

The table of model parameters has only three columns: N\_RECS, N\_CLUSTERS and N\_COMPS.

Default values of the above parameters are set to 10, 10 and 5, respectively.

Default locations:

* Input 1: file:/pg-recommender/data/data.csv (node #1)
* Input 2: file:/pg-recommender/data/op\_src\_parameters.csvcsv (node #9)

Note: pg-recommender is the default name of the repository.

## set OUTPUT DATA location

The locations of the output files must be set before executing the workflow.

The last two nodes will generate two output files in CSV format:

1. Table of recommendations
2. Table of store clusters

Default locations:

* Output 1: /pg-recommender/data/recommendations.csv (node #15)
* Output 2: /pg-recommender/data/store\_clusters.csv (node #16)

Note: pg-recommender is the default name of the repository.

## execution

Once the locations of the files have been configured, the workflow can be executed by pressing Shift + F7 or by clicking on Execute all nodes icon.

# Workflow elements

## Annotations and nodes

The workflow consists of 15 nodes that are grouped into 6 different workflow annotations.

|  |  |
| --- | --- |
| **Workflow annotation** | **Node Type** |
| Data preparation | 1. File Reader 2. Column Filter 3. Date Handler (Metanode)   3A) Number To String  3B) String to Date/Time   1. Sorter 2. Column Rename 3. Rule-based Row Filter 4. Column Filter 5. GroupBy |
| Recommender engine | 1. File Reader 2. Table Row to Variable 3. Python Script (1 => 2) |
| Post-processing | 1. Column Resorter 2. Variable to Table Row 3. Row Filter |
| Output | 1. CSV Writer #1 2. CSV Writer #2 |

## Node settings

### File Reader

The File Reader node is the first element of the workflow and it imports raw Transaction Data. This node can handle multiple file formats.

Note: make sure to check *read column headers* option.

### Column Filter

The Column Filter is the first node in the Data Preparation section. Its main purpose is to exclude columns that are not necessary in the analysis.

Included columns:

* TRANS\_DATE
* DMS\_STORE\_ID
* PGITEMCODE
* OFFTAKE\_TOTAL\_PIECE

Excluded columns:

* DMS\_ID
* DMS\_DSR\_ID
* OFFTAKE\_RMB(NIV)

### Date Handler

The Date Handler metanode consists of a *Number To String* and a *String to Date/Time node*. These nodes ensure that the type of TRANS\_DATE column is Date.

### Sorter

This node sorts the transactions by date (in ascending order).

Note: the use of this node is optional. It can be used for

### Column Rename

The Column Rename node is used to simply column names.

|  |  |
| --- | --- |
| **Old Name** | **New Name** |
| TRANS\_DATE | Transaction Date |
| DMS\_STORE\_ID | Store |
| PGITEMCODE | Item |
| OFFTAKE\_TOTAL\_PIECE | Quantity |

### Rule-based Row Filter

The Rule-based Row Filter excludes the negative transactions from the analysis. TRUE row matches for the following formula will be dismissed:

*$Quantity$ <= 0 => TRUE*

### Column Filter

As a last data preparation step, a Column Filter will only keep Store, Item and Quantity.

### GroupBy

The GroupBy node aggregates purchase quantities by stores and items.

### File Reader

This node imports model parameters from the second input file.

The parameters are stored in three separate columns, called *N\_OF\_RECS*, *N\_CLUSTERS, N\_COMPS*. All columns should have only one value.

N\_OF\_RECS controls the number of recommended items.

N\_CLUSTERS controls the number of store clusters.

N\_COMPS controls the number of the components of the SVD model.

Note: it is recommended to reset this node whenever the model parameters are changed.

### Table Row to Variable

The Table Row to Variable node converts the model parameters to flow variables, so they can be passed to the Python script as input parameters.

### Python Script (1 => 2)

The Python Script is the core component of the workflow.

The node has one input and two outputs. In addition, the three model parameters are passed to this script.

The first output table contains the recommended items for each store in graph format.

The second output table contains the results of KMeans clustering.

Note: a detailed summary of the Python script can be found in chapter 5.

### Column Resorter

This node sets the sorting order for the graph table. Default sorting order is Store -> Item -> Score.

### Variable to Table Row

Performance metrics of the recommender and the k-means algorithm (MAP and Silhouette score) are passed to a Variable to Table Row node. The metrics can be viewed by right-clicking on the node and selecting *Variable table*.

### Row Filter

The Row Filter enables the users to subset the output.

### CSV Writer 1

The first CSV Writer note exports the recommendations for each store in CSV format.

### CSV Writer 2

The second CSV Writer node exports the store clusters.

# Python script overview

**Lines 1-7**

import pandas as pd

import numpy as np

import operator as op

from scipy.sparse import csr\_matrix

from sklearn.decomposition import TruncatedSVD

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

In the first seven lines, Python imports the libraries and functions that are necessary for the analysis. All of these libraries are part of the Anaconda distribution.

**Lines 8-29**

# Getting aggregated transaction data

store\_items\_df = input\_table

items\_ordered = {x: y["Item"].tolist() for x,y in store\_items\_df.groupby("Store")}

# Getting model parameters

nr\_of\_recs = flow\_variables["N\_RECS"]

nr\_of\_clusters = flow\_variables["N\_CLUSTERS"]

nr\_of\_comps = flow\_variables["N\_COMPS"]

# Getting stores, items and quantities

stores = list(np.sort(store\_items\_df.Store.unique()))

items = list(store\_items\_df.Item.unique())

quantity = store\_items\_df.Quantity

# Creating store-item matrix

X\_rows = store\_items\_df.Store.astype('category', categories = stores).cat.codes

X\_cols = store\_items\_df.Item.astype('category', categories = items).cat.codes

X\_store\_item = csr\_matrix((quantity, (X\_rows, X\_cols)), shape = (len(stores), len(items)))

The second section saves stores, items and quantities in separate lists and dictionaries for future reference. The operations at the last 3 lines create the store-item sparse matrix in Row-Compressed format.

**Lines 30-43**

# Fitting SVD model on store-item matrix

svd = TruncatedSVD(n\_components = 100, n\_iter = 15, random\_state = 42)

svd.fit(X\_store\_item)

# Creating component matrices

item\_component\_matrix = svd.components\_

store\_component\_matrix = svd.transform(X\_store\_item)

# Kmeans on store component matrix

kmeans = KMeans(n\_clusters = nr\_of\_clusters)

kmeans.fit(store\_component\_matrix)

# Calculating matrix of SVD-based recommendations

M = np.dot(store\_component\_matrix, item\_component\_matrix)

The code between line 27 and 41 is the core element of the Python script. Here, the sparse matrix is passed to the *TruncatedSVD,* method, which performs Truncated Singular Value Decomposition. Store clustering with the k-means algorithm is performed at this secition as well. The decomposition model is then used for item recommendation.

**Lines 44-67**

# Functions for getting the results

def select\_top\_items(score\_set):

recs = dict(zip(items, score\_set))

elements = sorted(recs.items(), key = op.itemgetter(1), reverse = True)

top\_items = [item\_score[0] for item\_score in elements]

return top\_items

def select\_top\_recs(model\_matrix):

top\_recs = pd.DataFrame()

for i, score\_set in enumerate(model\_matrix):

current\_recs = dict(zip(items, score\_set))

existing\_items = items\_ordered[stores[i]]

for item in existing\_items:

if item in current\_recs:

del current\_recs[item]

sorted\_recs = sorted(current\_recs.items(), key=op.itemgetter(1), reverse = True)

current\_items = [item\_score[0] for item\_score in sorted\_recs][0:nr\_of\_recs]

current\_scores = [item\_score[1] for item\_score in sorted\_recs][0:nr\_of\_recs]

res\_chunk = pd.DataFrame({"Store": stores[i], "Item": map(str, current\_items), "Score": current\_scores})

top\_recs = top\_recs.append(res\_chunk, ignore\_index = True)

return top\_recs

The functions defined between line 42-67 are responsible for creating the output tables and for evaluating the recommender model.

**Lines 68 – 96**

# Predicted and actual items for scoring

predicted\_items = np.apply\_along\_axis(select\_top\_items, 1, M).tolist()

actual\_items = [items\_ordered[store] for store in sorted(items\_ordered)]

# Average Precision at k (k = number of recommended items)

def apk(actual, predicted, k = nr\_of\_recs):

if len(predicted)>k:

predicted = predicted[:k]

score = 0.0

num\_hits = 0.0

for i,p in enumerate(predicted):

if p in actual and p not in predicted[:i]:

num\_hits += 1.0

score += num\_hits / (i+1.0)

if not actual:

return 0.0

return score / min(len(actual), k)

# Mean Average Precision at k (k = number of recommended items)

def mapk(actual, predicted, k = nr\_of\_recs):

return np.mean([apk(a,p,k) for a,p in zip(actual, predicted)])

# Output tables

output\_table\_1 = select\_top\_recs(M)

output\_table\_2 = pd.DataFrame({"Store": stores, "Cluster": kmeans.labels\_})

# Output variables

flow\_variables['MAPscore'] = mapk(actual\_items, predicted\_items)

flow\_variables['KMscore'] = silhouette\_score(store\_component\_matrix, kmeans.labels\_)

In the last code block, the recommender and the cluster model will be evaluated on the original dataset. Output tables and variables are created at the last lines.

# Notes

## Licensing

The Open Source Recomender is released under MIT License: <https://opensource.org/licenses/MIT>.

| REVISION HISTORY | | | | | |
| --- | --- | --- | --- | --- | --- |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| n.n |  |  | dd-Mmm-yyyy |  | dd-Mmm-yyyy |
|  |  |  |  |  |  |
|  |  |  |  |  |  |