API Documentation

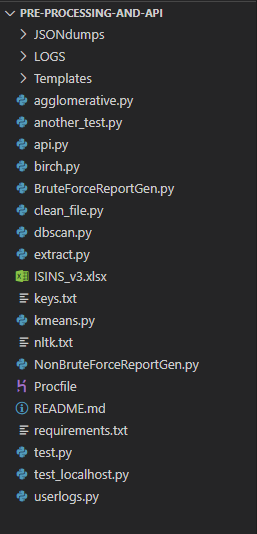
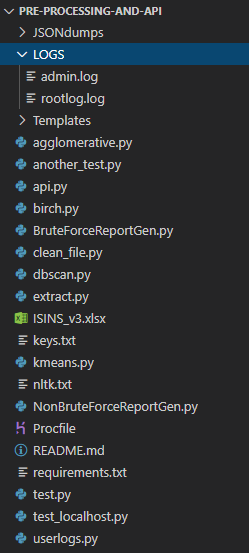
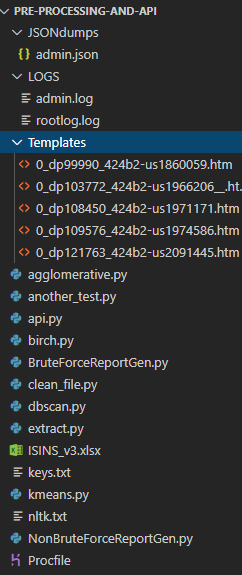
Required Packages:

**flask:** It is a micro web framework written in Python. It is required for request and response operations. To install, enter **: pip install Flask**

**flask\_restful:** It is an extension for Flask that adds support for quickly building REST APIs. It is required for Api, Resource and reqparse modules to make RESTful API development easier.

To install, enter : **pip install flask-restful**

Directory Structure:

**All files and folders LOGS folder to store logs JSONdumps and Templates**

**(rootlog.log stores server logs) (for template matching)**

Code Insight:

Required imports:

# Modules imported

from flask import Flask, request, make\_response, send\_file, jsonify

from matplotlib.backends.backend\_agg import FigureCanvasAgg as FigureCanvas

from flask\_restful import Api, Resource, reqparse

from urllib.request import urlopen

import json

import io

import os

import concurrent.futures

# Modules for template matching

import BruteForceReportGen

import NonBruteForceReportGen

import extract as ex        # Module for web scrapping and file operations

import clean\_file as cf     # Module for pre-processing

# Modules for clustering algorithms

import kmeans

import dbscan

import birch

import agglomerative as ag

# Module for creating log files

import userlogs as ul

import logging

Defining route for log files:

# Set directory for all log files

cwd = os.getcwd()

LOG\_FOLDER = os.path.join(cwd,'LOGS')

Format for log file:

formatter=logging.Formatter('%(asctime)s %(name)s %(levelname)s %(message)s')

Extensions created:

# Adding all URL paths

api.add\_resource(ReportGeneration, "/report")

api.add\_resource(ExtractData, '/extract')

api.add\_resource(ExportExtractedData, '/extract/export')

api.add\_resource(PreProcess, '/preprocess')

api.add\_resource(ExportPrepData, '/preprocess/export')

api.add\_resource(Kmeans, '/clustering/kmeans')

api.add\_resource(DBSCAN, '/clustering/dbscan')

api.add\_resource(Agglomerative, '/clustering/agglomerative')

api.add\_resource(Birch, '/clustering/birch')

api.add\_resource(ClusterSummary, '/clustering/summary')

api.add\_resource(Elbow, '/clustering/elbow')

api.add\_resource(Silhouette, '/clustering/silhouette')

api.add\_resource(Clear, '/clear')

api.add\_resource(SendLogs, '/getlog')

api.add\_resource(ClearLogs, '/clearlog')

Naming Convention for files:

To identify and retain data files for each user uniquely, the API uses naming convention of all the output files which include username and filename (file on which we are processing). Additionally, a timestamp is added to each filename.

**Note : Although the API is created to handle multiple users (with their own usernames), the web application that primarily uses this API is a single user application.**



All these files except for extracted and pre-processed will be cleared at the time of logout.

Running API locally:

For Windows users: Open command prompt and go to the directory where this API along with the modules that it imports is located. Then, set the environment variables "FLASK\_APP" and "FLASK\_ENV" and run the flask API as follows-

$ set FLASK\_APP = api.py

$ set FLASK\_ENV = development

$ flask run

For MAC/Linux users: Open terminal window and go to the directory where this API along with the modules that it imports is located. Then, set the environment variables "FLASK\_APP" and "FLASK\_ENV" and run the flask API as follows-

$ export FLASK\_APP = api.py

$ export FLASK\_ENV = development

$ flask run

API Deployment

The API is deployed on the Heroku Cloud Application Platform. This platform was chosen since it is very easy and less time consuming to deploy our applications. Our deployed repository is linked with a local git repository. Hence, making changes to our project and deploying them can be easily achieved by pushing our changes to our Heroku’s master branch.

Although, one major disadvantage of Heroku platform is that the Heroku router terminates a request made if that request takes more than 30 seconds to get processed. Since the functions performed by the API take a long time, this server is not suitable to match the API’s needs.

So, this API can be deployed on other platforms that can allow long requests. Some platforms are Google App Engine, Amazon Web Services.

API Functions

Following is a detailed description of all the functions that can be performed by the API.

1. Extracting data from URLs

URL: <https://preprocess-and-cluster-api.herokuapp.com/extract>

Methods allowed: GET, POST

1. POST request

Example in python:

#Extract data

url = "http://127.0.0.1:5000/extract?no\_of\_docs=10&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url, json=datajson)

Input Arguments:

1. no\_of\_docs (int/string, default = ‘all’) – Number of URLs from which text is to be extracted
2. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
3. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user

Input JSON object : It is a JSON object (named datajson in this case) which consists of ISIN numbers (unique code for every term sheet) as keys and the URLs of term sheets as values. API will extract text only from term sheets that are either “.htm” or “.html” files.

Example of input JSON object :

{

    'US17326YZV19': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm',

    'US17326YJJ64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm',

    'US17326YU388': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm',

    'US17326YNL64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm',

}

If the input object is not in JSON format, then the API will return the following response with error code 400 :

{

'data':'',

'message':'Error in json object parameter',

'status':'error'

}

Output of POST request

It will extract all the text present at the URLs given in the input JSON object and stores all the text in a “.json” file. Response is a JSON object indicating whether operation has succeeded or not.

If operation has succeeded, following response with error code 200 is returned :

{

'data': '',

'message': 'Data extracted',

'status': 'success'

}

1. GET Request

Prerequisite : Before making a GET request, POST request has to be made since GET request only reads the “.json” file created during the POST request and sends it to the user.

Example in Python :

url = "http://127.0.0.1:5000/extract?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments :

1. uname – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request

Response is a JSON object of the format :

{

data: result,

message: ‘’,

status: ‘Success’

}

Where “result” is a JSON object of the format :

{

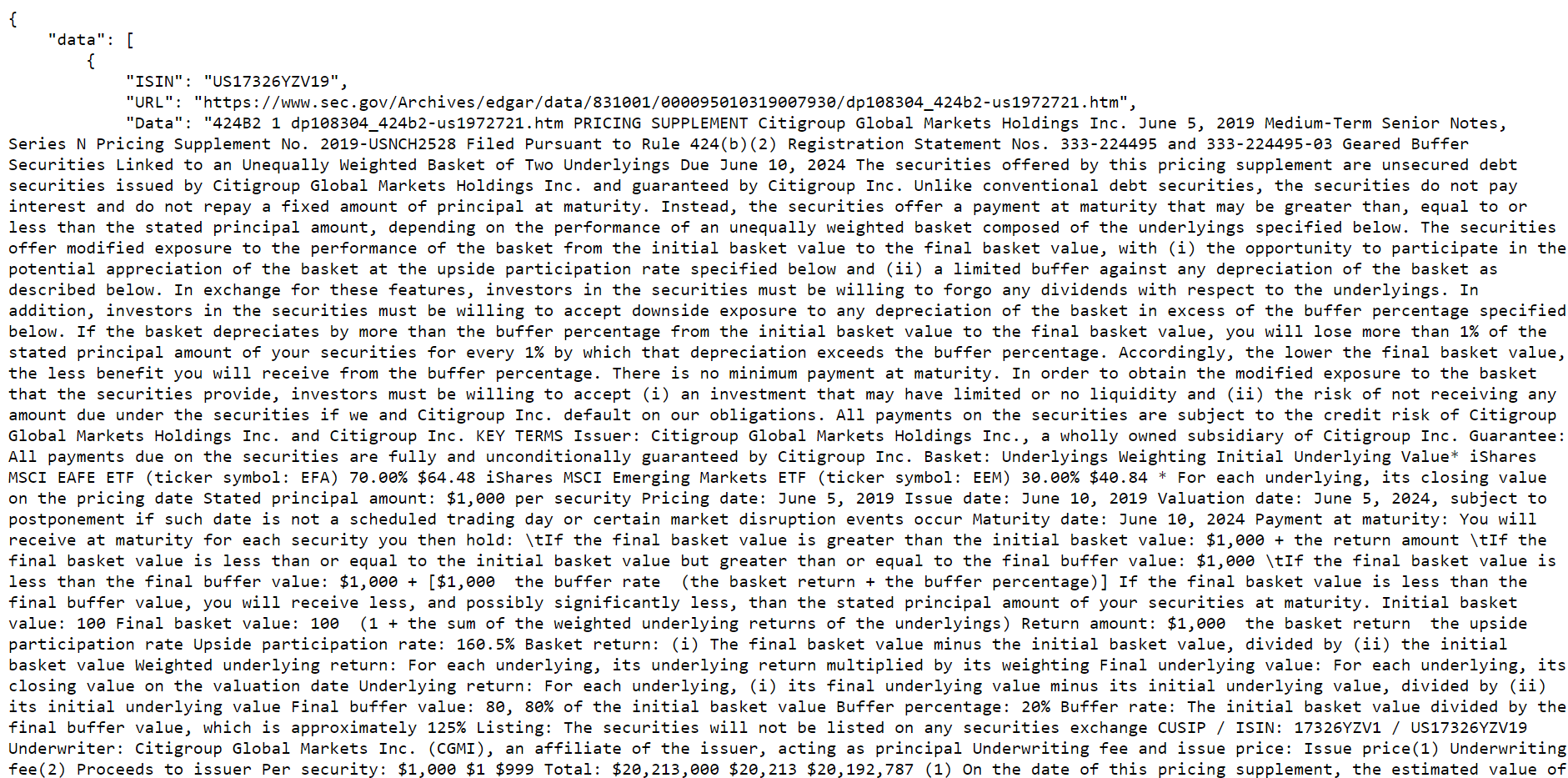
ISIN: isin number,

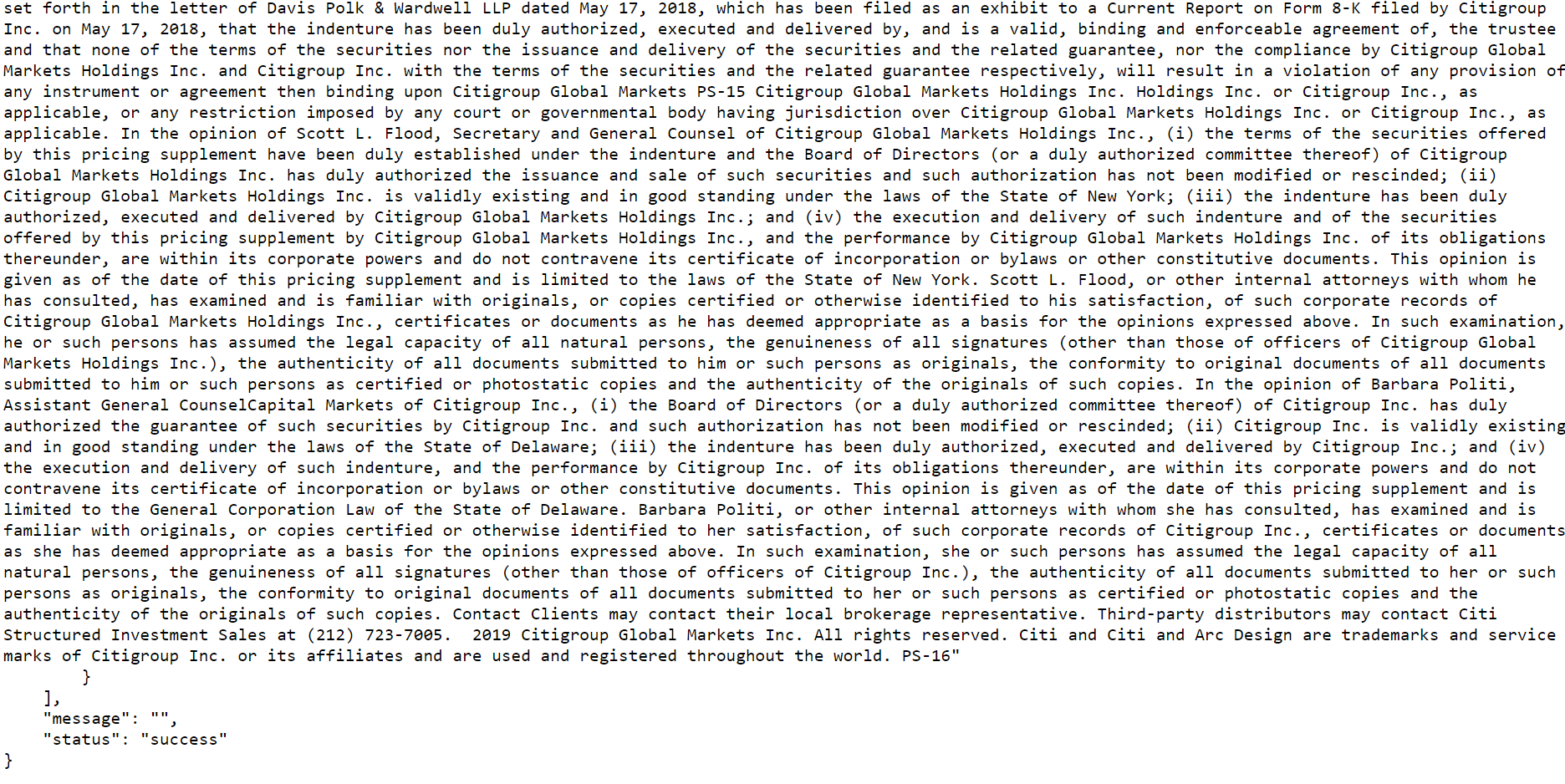
URL: url of term sheet

Data: extracted text

}

Example of response :





If GET request is made before corresponding POST request for extraction, error will be generated since the API could not read the file containing extracted data. In this case, output is JSON object of the format :

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

1. Pre-processing extracted text

URL : <https://preprocess-and-cluster-api.herokuapp.com/preprocess>

Methods Allowed : GET, POST

1. POST Request

Prerequisite : Extracted text json file must be present since the pre-processing function takes that file as input and pre-processes the text present.

Example in Python :

url = "http://127.0.0.1:5000/preprocess?steps=url&steps=stemming&steps=lemmatization&steps=stopwords&steps=unusual&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments :

1. uname – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.
3. steps (optional, default = [ ] ) – Different pre-processing functions that are to be applied on the extracted text. Values of steps can be one or more of the following :
4. url – removes URLs (if any) from the text
5. stemming – process of reducing words to their root form. For e.g., argue, argued, argues, arguing are reduced to “argu”.
6. lemmatization – process of grouping together the inflected forms of a word so they can be analysed as a single item
7. unusual – removes words with no meaning
8. stop words – removes stop words like “will”, “has”, “a”, “the”, “to” etc.

By default, the API will perform the following pre-processing functions on the text :

1. convert all text to lowercase
2. remove punctuations and numbers
3. removes end of line character ( \n ) and non-breaking space ( \xa0 )

Output of POST request

A successful operation will return status code 200 with the following response :

{

'data': '',

'message': 'Pre-processed!',

'status': 'success'

}

If data isn’t extracted before calling this request, then an error will be generated since API would not be able to find the file containing the extracted data. In this case, the status code will be 400 with the following response :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

Example in Python :

url = "http://127.0.0.1:5000/preprocess?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments :

1. uname – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request :

Response is a JSON object of the format :

{

data: result,

message: ‘’,

status: ‘Success’

}

Where “result” is a JSON object of the format :

{

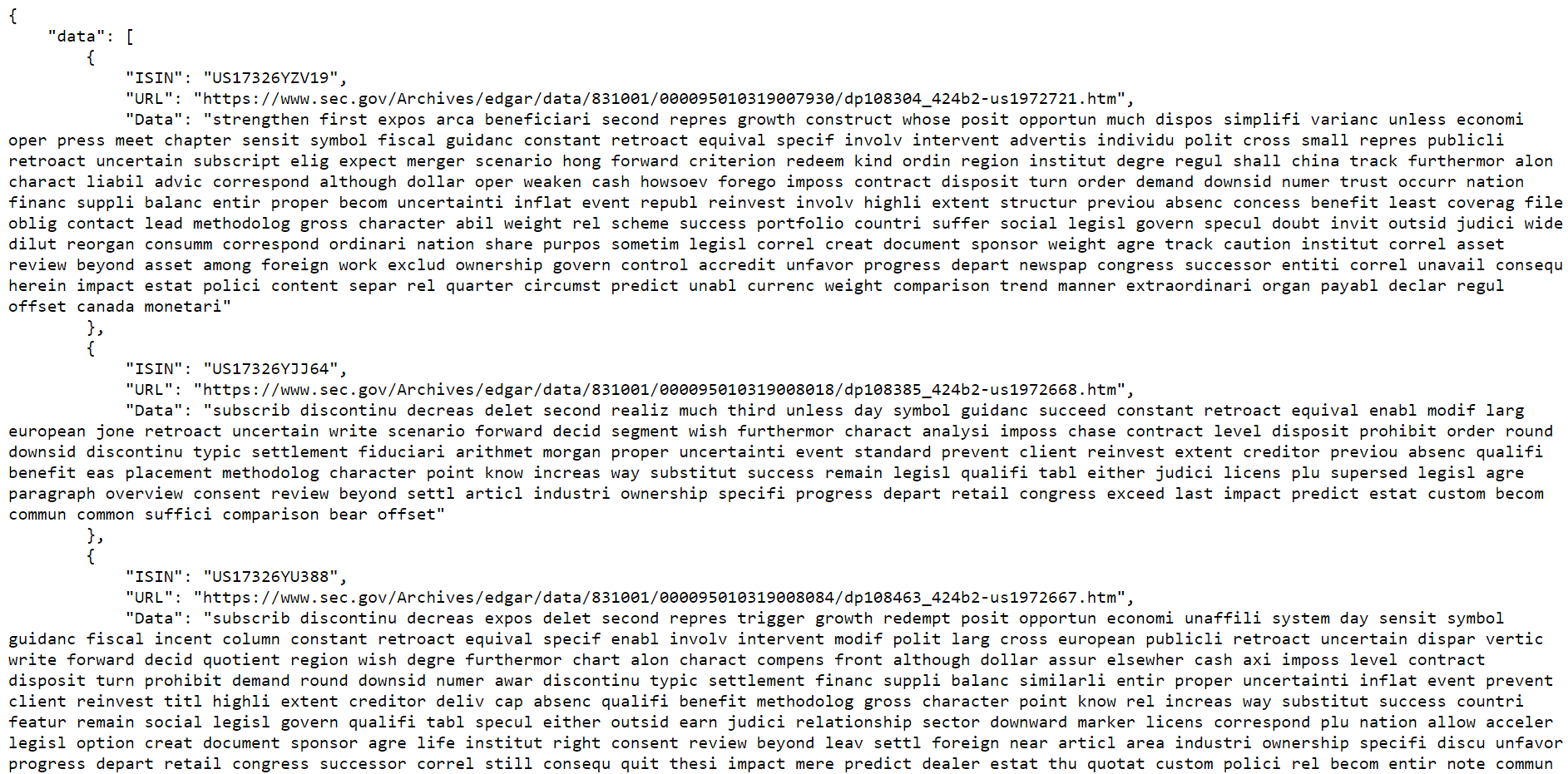
ISIN: isin number,

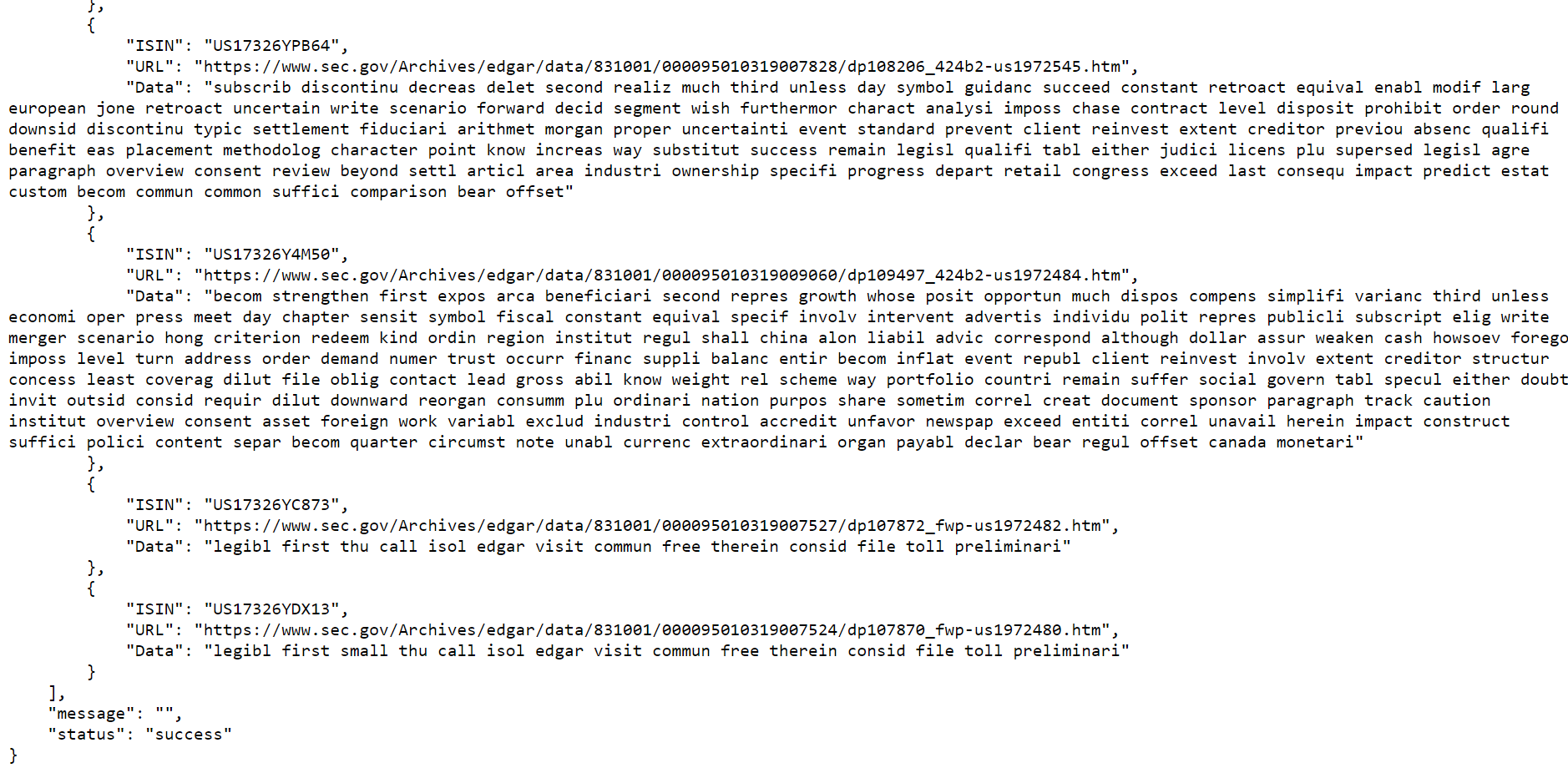
URL: url of term sheet

Data: pre-processed text

}

Example of response :





If GET request is made before corresponding POST request for extraction, error will be generated since the API could not read the file containing extracted data. In this case, output is JSON object of the format :

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

1. Exporting pre-processed data

URL: <https://preprocess-and-cluster-api.herokuapp.com/preprocess/export>

Methods allowed: POST

POST request

Example in python:

#Export pre-processed data

url = "https://preprocess-and-cluster-api.herokuapp.com/preprocess/export?filepath=prep.xlsx&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments:

1. filepath (string) – Name of the file to which pre-processed data is to be exported
2. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
3. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of POST request

It reads stored pre-processed data from “.json” file and copies to a excel file and this excel file is sent in response.

The file can be saved as :

open('prep.xlsx', 'wb').write(r.content)

The saved file will have name “prep.xlsx”. The name of the saved file can be anything according to the user’s needs. Format of the above command is :

open(filename, 'wb').write(r.content)

where **filename** is the name of the output file that the user wants.

Clustering

We have implemented 4 clustering algorithms. Which clustering algorithm is to be used depends on the user. The clustering algorithms used are :

1. K means clustering
2. DBSCAN clustering
3. Agglomerative clustering
4. Birch clustering

Clustering involves converting the text data into vectors by calculating the “TFIDF matrix” (Term Frequency Inverse Distribution Frequency matrix). Order of the TFIDF matrix is (no\_of\_docs x no. of features), where features are distinct words present in the pre-processed text. In order to reduce the order of this matrix and thus, making the process of manipulating the TFIDF matrix faster, two more methods are used :

1. PCA (Principal Component Analysis)
2. Variance Threshold Method

Both these methods aim in reducing the number of features.

1. K means clustering

URL: <https://preprocess-and-cluster-api.herokuapp.com/clustering/kmeans>

Methods allowed: GET, POST

1. POST request

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in python:

import requests

#Perform K-Means clustering

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/kmeans?k=4&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments:

1. k (int) – Number of clusters to be made
2. format (optional, default = excel) – Output format of file containing clustering results. Valid values are:
3. excel – Output file containing clustering results is an excel file
4. csv – Output file containing clustering results in a csv file
5. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
6. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.
7. pca\_comp (optional, int/float, default = 0.8) – Number of components to keep in PCA
8. thresh (float, optional, default = 0.001) – Features with a training-set variance lower than this threshold will be removed

Output of POST request

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted on a scatter plot. Every point represents a document.
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response:

>>> import requests

>>> url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/kmeans?k=4&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

>>> r = requests.post(url=url)

>>> r.json()

{

'data': [

{'x': -4.0923180789736096, 'y': 15.48886673311059},

{'x': 10.944620935488446, 'y': -8.387811565258191},

{'x': 16.687021530485232, 'y': -1.0435402576253514}, {'x': -12.966191003661837, 'y': -8.866649041131147}, {'x': 17.6188275360211, 'y': -6.533462791613075},

{'x': -12.536286343388568, 'y': -8.46616045934616},

{'x': 10.927797327309953, 'y': -8.490494343413197},

{'x': -1.3363433167846401, 'y': 14.750268378226355}, {'x': 1.6268598858615237, 'y': -3.883111134632197},

{'x': 1.4989502606046063, 'y': -3.821575244128336},

{'x': -3.7154567606941273, 'y': 2.826824624996808},

{'x': 1.6268598858615246, 'y': -3.8831111346321987}, {'x': 4.795273234517435, 'y': 9.57296145246006},

{'x': 14.202003687049162, 'y': 1.5926134999349106},

{'x': -12.417963035163314, 'y': -7.9222489003517955}, {'x': -8.866024786866431, 'y': -4.622541386417186},

{'x': -11.982024450916994, 'y': -8.440747410795273}, {'x': -6.242285854800599, 'y': 2.646624737386714},

{'x': -4.757072998126867, 'y': 15.130439274908914},

{'x': -1.0162476538220064, 'y': 12.352854968319763}

],

'message': '',

'status': 'success'

}

The clustering output is stored in a “.json” file on the server.

If the file containing pre-processed text is not present, then POST request will give a response with status code 400 as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

Example in Python:

import requests

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/kmeans?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments:

1. uname – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request

GET request sends a JSON object with keys :

1. data – value is a dictionary with showing all documents belonging to various clusters after performing k means clustering
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

{

'data':{'Cluster 0': [

{'ISIN': 'US17326YZV19', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm'},

{'ISIN': 'US17326YJJ64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm'},

{'ISIN': 'US17326YU388', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm'},

{'ISIN': 'US17326YNL64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm'},

{'ISIN': 'US17326YRV01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009050/dp109447\_424b2-us1972547.htm'}],

'Cluster 1': [

{'ISIN': 'US17326YPB64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007828/dp108206\_424b2-us1972545.htm'},

{'ISIN': 'US17326YC873', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007527/dp107872\_fwp-us1972482.htm'},

{'ISIN': 'US17326YDX13', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007524/dp107870\_fwp-us1972480.htm'},

{'ISIN': 'US17326YFJ01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008572/dp109026\_424b2-us1972369.htm'}, {'ISIN': 'US17326YTD84', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007525/dp107868\_fwp-us1972350.htm'}],

'message': '',

'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

<Response [400]>

1. DBSCAN clustering

URL: <https://preprocess-and-cluster-api.herokuapp.com/clustering/dbscan>

Methods allowed: GET, POST

1. POST request

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in python:

import requests

#Perform DBSCAN clustering

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/dbscan?eps=0.18&min=3&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments:

1. eps (float) – The maximum distance between two samples for one to be considered as in the neighbourhood of the other. This is not a maximum bound on the distances of points within a cluster.
2. min (int) – The number of samples (or total weight) in a neighbourhood for a point to be considered as a core point.
3. format (optional, default = excel) – Output format of file containing clustering results. Valid values are:
   1. excel – Output file containing clustering results is an excel file
   2. csv – Output file containing clustering results in a csv file
4. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
5. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.
6. pca\_comp (optional, int/float, default = 0.8) – Number of components to keep in PCA
7. thresh (float, optional, default = 0.001) – Features with a training-set variance lower than this threshold will be removed

Output of POST request

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted on a scatter plot. Every point represents a document.
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response:

>>> import requests

>>> url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/dbscan?eps=0.18&min=3&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

>>> r = requests.post(url=url)

>>> r.json()

{

'data': [

{'x': -4.0923180789736096, 'y': 15.48886673311059},

{'x': 10.944620935488446, 'y': -8.387811565258191},

{'x': 16.687021530485232, 'y': -1.0435402576253514}, {'x': -12.966191003661837, 'y': -8.866649041131147}, {'x': 17.6188275360211, 'y': -6.533462791613075},

{'x': -12.536286343388568, 'y': -8.46616045934616},

{'x': 10.927797327309953, 'y': -8.490494343413197},

{'x': -1.3363433167846401, 'y': 14.750268378226355}, {'x': 1.6268598858615237, 'y': -3.883111134632197},

{'x': 1.4989502606046063, 'y': -3.821575244128336},

{'x': -3.7154567606941273, 'y': 2.826824624996808},

{'x': 1.6268598858615246, 'y': -3.8831111346321987}, {'x': 4.795273234517435, 'y': 9.57296145246006},

{'x': 14.202003687049162, 'y': 1.5926134999349106},

{'x': -12.417963035163314, 'y': -7.9222489003517955}, {'x': -8.866024786866431, 'y': -4.622541386417186},

{'x': -11.982024450916994, 'y': -8.440747410795273}, {'x': -6.242285854800599, 'y': 2.646624737386714},

{'x': -4.757072998126867, 'y': 15.130439274908914},

{'x': -1.0162476538220064, 'y': 12.352854968319763}

],

'message': '',

'status': 'success'

}

The clustering output is stored in a “.json” file on the server.

If the file containing pre-processed text is not present, then POST request will give response with status code 400 as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

Example in Python:

import requests

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/dbscan?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string)– The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request

GET request sends a JSON object with keys :

1. data – value is a dictionary with showing all documents belonging to various clusters after performing DBSCAN clustering
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

{'data':

{'Cluster 0': [{'ISIN': 'US17326YZV19', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm'},

{'ISIN': 'US17326YJJ64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm'}],

'Cluster 1': [{'ISIN': 'US17326YU388', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm'},

{'ISIN': 'US17326YNL64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm'}],

'Cluster 2': [{'ISIN': 'US17326YUM64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007911/dp108280\_424b2-us1972550.htm'},

{'ISIN': 'US17326YRV01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009050/dp109447\_424b2-us1972547.htm'},

{'ISIN': 'US17326YPB64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007828/dp108206\_424b2-us1972545.htm'}],

'Cluster 3': [{'ISIN': 'US17326Y4M50', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009060/dp109497\_424b2-us1972484.htm'},

{'ISIN': 'US17326YC873', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007527/dp107872\_fwp-us1972482.htm'},

{'ISIN': 'US17326YDX13', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007524/dp107870\_fwp-us1972480.htm'},

{'ISIN': 'US17326YFJ01', 'URL':

'https://www.sec.gov/Archives/edgar/data/831001/000095010319008572/dp109026\_424b2-us1972369.htm'},

{'ISIN': 'US17326YTD84', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007525/dp107868\_fwp-us1972350.htm'}],

'Cluster 4': [{'ISIN': 'US17326YAH99', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009080/dp109492\_424b2-us1972281.htm'},

{'ISIN': 'US17326YMQ60', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007870/dp108232\_424b2-us1972280.htm'},

{'ISIN': 'US17326YBP07', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007155/dp107611\_424b2-us1972269.htm'}],

'Cluster 5': [{'ISIN': 'US17326Y2L95', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009048/dp109444\_424b2-us1972158.htm'},

{'ISIN': 'US17326YN854', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008567/dp108880\_424b2-us1972037.htm'},

{'ISIN': 'US17326YK215', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008005/dp108381\_424b2-us1971957.htm'}],

'Cluster 6': [{'ISIN': 'US17326YRK46', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007683/dp108134\_424b2-us1971956.htm'}],

'Cluster 7': [{'ISIN': 'US17326Y5P72', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319006794/dp107171\_424b2-us1971861.htm'}]},

'message': '', 'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output. Status code of response is 400.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

1. Agglomerative clustering

URL: <https://preprocess-and-cluster-api.herokuapp.com/clustering/agglomerative>

Methods allowed: GET, POST

1. POST request

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in python:

import requests

#Perform K-Means clustering

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/agglomerative?k=5&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments:

1. k (int) – The number of clusters to find
2. format (optional, default = excel) – Output format of file containing clustering results. Valid values are:

a) excel – Output file containing clustering results is an excel file

1. csv – Output file containing clustering results in a csv file
2. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
3. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.
4. pca\_comp (optional, int/float, default = 0.8) – Number of components to keep in PCA
5. thresh (float, optional, default = 0.001) – Features with a training-set variance lower than this threshold will be removed

Output of POST request

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted on a scatter plot. Every point represents a document.
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response:

>>> import requests

>>> url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/kmeans?k=4&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

>>> r = requests.post(url=url)

>>> r.json()

{

'data': [

{'x': -4.0923180789736096, 'y': 15.48886673311059},

{'x': 10.944620935488446, 'y': -8.387811565258191},

{'x': 16.687021530485232, 'y': -1.0435402576253514}, {'x': -12.966191003661837, 'y': -8.866649041131147}, {'x': 17.6188275360211, 'y': -6.533462791613075},

{'x': -12.536286343388568, 'y': -8.46616045934616},

{'x': 10.927797327309953, 'y': -8.490494343413197},

{'x': -1.3363433167846401, 'y': 14.750268378226355}, {'x': 1.6268598858615237, 'y': -3.883111134632197},

{'x': 1.4989502606046063, 'y': -3.821575244128336},

{'x': -3.7154567606941273, 'y': 2.826824624996808},

{'x': 1.6268598858615246, 'y': -3.8831111346321987}, {'x': 4.795273234517435, 'y': 9.57296145246006},

{'x': 14.202003687049162, 'y': 1.5926134999349106},

{'x': -12.417963035163314, 'y': -7.9222489003517955}, {'x': -8.866024786866431, 'y': -4.622541386417186},

{'x': -11.982024450916994, 'y': -8.440747410795273}, {'x': -6.242285854800599, 'y': 2.646624737386714},

{'x': -4.757072998126867, 'y': 15.130439274908914},

{'x': -1.0162476538220064, 'y': 12.352854968319763}

],

'message': '',

'status': 'success'

}

The clustering output is stored in a “.json” file on the server.

If the file containing pre-processed text is not present, then POST request will give response with status code 400 as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

Example in Python:

import requests

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/agglomerative?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request

GET request sends a JSON object with keys :

1. data – value is a dictionary with showing all documents belonging to various clusters after performing agglomerative clustering
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

{

'data':{'Cluster 0': [

{'ISIN': 'US17326YZV19', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm'},

{'ISIN': 'US17326YJJ64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm'},

{'ISIN': 'US17326YU388', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm'},

{'ISIN': 'US17326YNL64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm'},

{'ISIN': 'US17326YRV01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009050/dp109447\_424b2-us1972547.htm'}],

'Cluster 1': [

{'ISIN': 'US17326YPB64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007828/dp108206\_424b2-us1972545.htm'},

{'ISIN': 'US17326YC873', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007527/dp107872\_fwp-us1972482.htm'},

{'ISIN': 'US17326YDX13', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007524/dp107870\_fwp-us1972480.htm'},

{'ISIN': 'US17326YFJ01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008572/dp109026\_424b2-us1972369.htm'}, {'ISIN': 'US17326YTD84', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007525/dp107868\_fwp-us1972350.htm'}],

'message': '',

'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output. Status code of response is 400.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

1. Birch clustering

URL : <https://preprocess-and-cluster-api.herokuapp.com/clustering/birch>

Methods allowed: GET, POST

1. POST request

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in python:

import requests

#Perform K-Means clustering

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/birch?k=5&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments:

1. k (int) – Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples.
2. format (optional, default = excel) – Output format of file containing clustering results. Valid values are:
3. excel – Output file containing clustering results is an excel file
4. csv – Output file containing clustering results in a csv file
5. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
6. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.
7. pca\_comp (optional, int/float, default = 0.8) – Number of components to keep in PCA
8. thresh (float, optional, default = 0.001) – Features with a training-set variance lower than this threshold will be removed

Output of POST request

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted on a scatter plot. Every point represents a document.
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response:

>>> import requests

>>> url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/birch?k=5&format=csv&uname=admin&fname=ISINS\_v3.xlsx"

>>> r = requests.post(url=url)

>>> r.json()

{

'data': [

{'x': -4.0923180789736096, 'y': 15.48886673311059},

{'x': 10.944620935488446, 'y': -8.387811565258191},

{'x': 16.687021530485232, 'y': -1.0435402576253514}, {'x': -12.966191003661837, 'y': -8.866649041131147}, {'x': 17.6188275360211, 'y': -6.533462791613075},

{'x': -12.536286343388568, 'y': -8.46616045934616},

{'x': 10.927797327309953, 'y': -8.490494343413197},

{'x': -1.3363433167846401, 'y': 14.750268378226355}, {'x': 1.6268598858615237, 'y': -3.883111134632197},

{'x': 1.4989502606046063, 'y': -3.821575244128336},

{'x': -3.7154567606941273, 'y': 2.826824624996808},

{'x': 1.6268598858615246, 'y': -3.8831111346321987}, {'x': 4.795273234517435, 'y': 9.57296145246006},

{'x': 14.202003687049162, 'y': 1.5926134999349106},

{'x': -12.417963035163314, 'y': -7.9222489003517955}, {'x': -8.866024786866431, 'y': -4.622541386417186},

{'x': -11.982024450916994, 'y': -8.440747410795273}, {'x': -6.242285854800599, 'y': 2.646624737386714},

{'x': -4.757072998126867, 'y': 15.130439274908914},

{'x': -1.0162476538220064, 'y': 12.352854968319763}

],

'message': '',

'status': 'success'

}

The clustering output is stored in a “.json” file on the server.

If the file containing pre-processed text is not present, then POST request will give an error as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

<Response [400]>

1. GET Request

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

Example in Python:

import requests

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/birch?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of GET request

GET request sends a JSON object with keys :

1. data – value is a dictionary with showing all documents belonging to various clusters
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

{

'data':{'Cluster 0': [

{'ISIN': 'US17326YZV19', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm'},

{'ISIN': 'US17326YJJ64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm'},

{'ISIN': 'US17326YU388', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm'},

{'ISIN': 'US17326YNL64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm'},

{'ISIN': 'US17326YRV01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009050/dp109447\_424b2-us1972547.htm'}],

'Cluster 1': [

{'ISIN': 'US17326YPB64', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007828/dp108206\_424b2-us1972545.htm'},

{'ISIN': 'US17326YC873', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007527/dp107872\_fwp-us1972482.htm'},

{'ISIN': 'US17326YDX13', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007524/dp107870\_fwp-us1972480.htm'},

{'ISIN': 'US17326YFJ01', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008572/dp109026\_424b2-us1972369.htm'}, {'ISIN': 'US17326YTD84', 'URL': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007525/dp107868\_fwp-us1972350.htm'}],

'message': '',

'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

<Response [400]>

Getting optimal value of k for k means clustering

Two methods are adopted to get an optimal value of k such that clustering is as accurate as possible. These are :

1. By plotting Elbow Curve
2. By plotting variation of Silhouette score for different values of k
3. By plotting Elbow Curve

URL : <https://preprocess-and-cluster-api/clustering/elbow>

Methods allowed : GET, POST

1. POST Request :

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in Python :

#Returns elbow curve plot

url = "http://127.0.0.1:5000/clustering/elbow?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of POST request :

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted. The keys are the x co-ordinates and values are the y-co-ordinates
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response :

{

'data': {

'1': 12443.112924127114, '2': 9473.239099554934,

'3': 7082.909947113216, '4': 5630.211732054874,

'5': 4108.545152659102, '6': 2818.5104416655163,

'7': 1844.3882383872287, '8': 1306.5392775933428,

'9': 763.8806690767134, '10': 542.2995166018684,

'11': 333.0735688122333, '12': 228.78763716422958,

'13': 137.824767137972, '14': 93.96549258707911,

'15': 55.23547955503728, '16': 24.67058853071315,

'17': 9.969219674783792, '18': 5.071144223766692,

'19': 3.9062461284575782, '20': 2.934753700091844,

'21': 2.003294355660035, '22': 1.3129347424520112,

'23': 0.9380908438217259, '24': 0.5850218820788547,

'25': 0.2957420773488286, '26': 0.16146596010669914,

'27': 0.04648242986898675, '28': 1.583761526747622e-28,

'29': 1.279495410413549e-28

},

'message': '',

'status': 'success'

}

If the file containing pre-processed text is not present, then POST request will give response with status code 400 as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request :

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

#Returns optimal value of k by plotting elbow curve

url = "http://127.0.0.1:5000/clustering/elbow?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

GET request sends a JSON object with keys :

1. data – optimal k value from elbow curve
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of response from GET request :

{

'data': 7,

'message': '',

'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output. Status code of response is 400.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

1. By plotting Silhouette score variation with k

URL : <https://preprocess-and-cluster-api/clustering/silhouette>

Methods allowed : GET, POST

1. POST Request :

Prerequisite : File containing pre-processed text should be present since this algorithm takes the pre-processed text as input.

Example in Python :

#Returns variation of silhouette score with k

url = "http://127.0.0.1:5000/clustering/silhouette?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.post(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

Output of POST request :

POST request sends a JSON object with keys :

1. data – value is a dictionary with co-ordinates of all the points to be plotted. The keys are the x co-ordinates and values are the y-co-ordinates
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of POST request response :

{

'data': {

'2': 0.3698893233576973, '3': 0.38475223027180155,

'4': 0.43594660680183367, '5': 0.5522857801885965,

'6': 0.6933446926915032, '7': 0.7336542283129397,

'8': 0.7479672818766534, '9': 0.8020823118030215,

'10': 0.8280903465176426, '11': 0.8453225162071671,

'12': 0.8592583729555385, '13': 0.8459485980814488,

'14': 0.8202532605978171, '15': 0.7793402849566312,

'16': 0.7220073388998267, '17': 0.6805276821024768,

'18': 0.6182607468213898

},

'message': '',

'status': 'success'

}

Silhouette score is a measurement of how efficiently clustering is done.

If the file containing pre-processed text is not present, then POST request will give response with status code 400 as follows :

{

'data': '',

'message': 'Failed to read data!',

'status': 'error'

}

1. GET Request :

Pre-requisite : POST request should be called before GET request. GET request reads the “.json” file created during the POST request and sends it to the user.

#Returns optimal value of k from Silhouette score variation

url = "http://127.0.0.1:5000/clustering/silhouette?uname=admin&fname=ISINS\_v3.xlsx"

r = requests.get(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results
2. fname (string) – The name of the pre-processed file user has either uploaded or generated. This is again used to correctly identify which clustered output file belongs to which user.

GET request sends a JSON object with keys :

1. data – optimal k value from plotting variation of Silhouette score with k
2. Message – an empty message
3. Status – if the response code is 200 i.e. satisfactory, then status value is “Success”

Example of response from GET request :

{

'data': 9,

'message': '',

'status': 'success'

}

If GET request is made before the POST request on this extension, then API will give an error since it could not read the “.json” file containing the clustering output.

{

'data': '',

'message': 'Error in reading file',

'status': 'error'

}

<Response [400]>

1. Getting a summary of clustering performed

After clustering is performed, the API creates and stores a basic summary of clustering.

URL – <https://preprocess-and-cluster-api/clustering/summary>

Methods allowed – GET

Prerequisite : Clustering must be performed first in order to get the summary. The API will return summary of the latest clustering that is performed.

A) GET Request

Example in Python :

# Get summary of clustering

url = "https://preprocess-and-cluster-api.herokuapp.com/clustering/summary?uname=admin&content\_type=summary”

r = requests.get(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results.
2. content\_type (string, optional, default = summary) – Takes two values :
3. summary – Returns a JSON object where the key “data” contains cluster numbers as keys and number of documents in each cluster as values.

Example of response when content\_type = summary :

{

'data': {

'Cluster 0': 15,

'Cluster 1': 17,

'Cluster 2': 16,

'Cluster 3': 2

},

'status': 'success'

}

1. clust – Returns a JSON object where the key “data” contains the cluster number of every document in the order of the documents passed in the input JSON object.

Example of response when content\_type = clust :

{

'data': [

2, 0, 0, 1, 0, 1, 0, 2, 1, 1, 2, 1, 2, 3, 1, 1, 1, 1, 2, 2, 0, 2, 0, 0, 3, 0, 2, 0, 0, 0, 1, 2, 0, 2, 1, 2, 1, 1, 1, 2, 0, 0, 1, 2, 1, 1, 2, 2, 2, 0

],

'status': 'success'

}

1. Template Matching

This function maps the templates to the various term-sheets and gets the number of matches and miss-matches along with getting the useful attributes in the term-sheets and the templates.

URL – <https://preprocess-and-cluster-api/report>

Methods Allowed – GET, POST

1. POST Request

Example in Python :

# Generating report on Template Matching

url = "https://preprocess-and-cluster-api/report?uname=admin&kind=1"

r = requests.post(url=url, json=datajson)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results.
2. kind (int, optional, default = 1) – The algorithm to be used in order to map the templates to the term sheets. Valid values are :
3. 1 – Uses Brute-Force algorithm
4. 2 – Uses Non Brute-Force algorithm

Input JSON object : It is a JSON object (named datajson in this case) which consists of ISIN numbers (unique code for every term sheet) as keys and the URLs of term sheets as values.

Example of input JSON object :

datajson = {

'US17326YZV19': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007930/dp108304\_424b2-us1972721.htm',

'US17326YJJ64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008018/dp108385\_424b2-us1972668.htm',

'US17326YU388': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008084/dp108463\_424b2-us1972667.htm',

'US17326YNL64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009058/dp109430\_424b2-us1972617.htm',

'US17326YUM64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007911/dp108280\_424b2-us1972550.htm',

'US17326YRV01': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009050/dp109447\_424b2-us1972547.htm',

'US17326YPB64': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007828/dp108206\_424b2-us1972545.htm',

'US17326Y4M50': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009060/dp109497\_424b2-us1972484.htm',

'US17326YC873': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007527/dp107872\_fwp-us1972482.htm',

'US17326YDX13': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007524/dp107870\_fwp-us1972480.htm',

'US17326YFJ01': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319008572/dp109026\_424b2-us1972369.htm',

'US17326YTD84': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007525/dp107868\_fwp-us1972350.htm',

'US17326YAH99': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009080/dp109492\_424b2-us1972281.htm',

'US17326YMQ60': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007870/dp108232\_424b2-us1972280.htm',

'US17326YBP07': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319007155/dp107611\_424b2-us1972269.htm',

'US17326Y2L95': 'https://www.sec.gov/Archives/edgar/data/831001/000095010319009048/dp109444\_424b2-us1972158.htm'

}

POST request will map the term sheets given in the input JSON object to templates that are manually generated and are already present on the server. The generated report is then stored in an output “.json” file which can be accessed by making a GET request on this url.

Output of POST request

Response is a JSON object with status code 200 with the following message :

{

'data': '',

'message': 'Template generated',

'status': 'success'

}

If the argument “kind” has any other value than 1 or 2, following is the response :

{

'data': '',

'message': 'Invalid value of argument "kind". Valid values are "1" or "2".',

'status': 'error'

}

1. GET Request

Prerequisite : At least one POST request on this URL must be made before any GET request is made since the file gets prepared by sending the POST request. Once, an adequate amount of time has passed to prepare the file, any number of GET requests may be sent to get the JSON dump which can then converted to Excel.

Example in Python :

url = "https://preprocess-and-cluster-api/report?uname=admin"

r = requests.get(url=url)

Input Arguments :

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results.

GET request returns a JSON object with keys :

1. Miss-match matrix
2. Attributes from sheet
3. Attributes from template
4. All attributes from sheet

Example of response :





This returned object cannot be read easily by human but can be easily interpreted and converted into an excel file.

If GET request is made before the corresponding POST request, the API would not be able to read the file containing the report generated. Hence, it will return the following message :

{

"data": "",

"message": "Could not find the JSON Dump, the file is not prepared yet",

"status": "error"

}

1. Clearing summary and clustering files at logout:

URL: [https://preprocess-and-cluster-api.herokuapp.com/clear](https://preprocess-and-cluster-api.herokuapp.com/preprocess/export)

Methods allowed: DELETE

1. DELETE request

Example in python:

# Clear summary and cluster files at the time of log out

url = "http://127.0.0.1:5000/clear?uname=admin"

r = requests.delete(url=url)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results

Output of DELETE request

It will delete all the data files created except for extracted and pre-processed files of that user. Response is a JSON object indicating whether operation has succeeded or not.

If operations has succeeded, response is :

{

'data': '',

'status': 'success'

}

<Response [200]>

1. Get log file:

URL: https://preprocess-and-cluster-api.herokuapp.com/getlog

Methods allowed: GET

1. GET request

Example in python:

url = "http://127.0.0.1:5000/getlog?uname=admin"

r = requests.get(url=url)

open('admin\_'+'log.log', 'wb').write(r.content)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results

Output of GET request

It will send the log file of the requesting user in response.

1. Clearing logs:

URL: [https://preprocess-and-cluster-api.herokuapp.com/clear](https://preprocess-and-cluster-api.herokuapp.com/preprocess/export)log

Methods allowed: DELETE

1. DELETE request

Example in python:

# Clear log file of requesting user

url = "http://127.0.0.1:5000/clearlog?uname=admin"

r = requests.delete(url=url)

Input Arguments:

1. uname (string) – Username of user who is requesting for the data. This is given as an input argument so that the API will return correct results to the correct user and avoids jumbling up of the results

Output of DELETE request

It will clear all logs of the requesting user. Response is a JSON object indicating whether operation has succeeded or not.

If operations has succeeded, response is the following with status code 200 :

{

'data': '',

'status': 'success'

}

Note – In all the above requests, if “uname” and/or “fname” arguments are not given, response is as follows :

{

'data': '',

'message': 'Give user name',

'status': 'error'

}

{

'data': '',

'message': 'Give file name',

'status': 'error'

}

Since the API would store output files for many users on the server, in order to distinguish uniquely which file belongs to which user, all the file names are given the following :

1. Time stamp – date and time when request was processed
2. Username (uname argument) – Username
3. File name (fname argument) – The name of the file which user uploaded (excel file containing the ISIN number and URLs of term sheets)

Example of an output files created –



Note that the API does not access the file with name “fname” and uses the argument just to name the output files to correctly identify which file belongs to which user.

Future Prospects of this API

* Deploying it on a more flexible and reliable platform which can mainly handle long requests. Some platforms are Google App Engine, Amazon Web Services
* Make the API faster by using “Threading” so that it could be able to run/processes multiple tasks/functions at the same time.
* Make previous results available to the user so that he/she can compare changes