

PROJECT SOA

Hierarchical clustering



INNOVATION ET AMOA
2019-2020

Réalisé par :

- ELMAGUI Fatiha
- JARMOUNE Mohamed

Encadré par :

Mr H. ELGHAZI

Remerciement

*Avant d'entamer ce rapport de mini-projet, nous tenons à exprimer nos sincères gratitudes envers notre professeur Mr **H. EL GHAZI** qui nous a donné l'opportunité de travailler sur ce mini-projet, pour sa générosité en transmettre son savoir et son savoir-faire dans les meilleures conditions possibles.*

Introduction :

Le web service est un protocole d'interface informatique qui a été développé pour permettre la communication et l'échange entre les applications et les systèmes hétérogènes dans des environnements distribués. Le protocole de communication est défini dans le cadre de la norme SOAP dans la signature du service exposé (WSDL).

Les services web permettent essentiellement aux différentes applications écrites dans des langages de programmation différents de communiquer entre elles.

Le service web nous disposons dans ce projet consiste à appliquer l'algorithme du Hierarchical clustering. Le problème que nous allons résoudre dans ce projet est de segmenter les clients en différents groupes en fonction de leurs tendances d'achat. (et cela en utilisant un dataset des clients)

Objectif du projet :

Dans ce projet nous allons travailler avec le web server qui est installé dans notre machine, Nous allons donc avoir une configuration un peu particulière puisque le client et le serveur vont se trouver sur la même machine pour objectif de réaliser deux interfaces permettant d'afficher le résultat de l'algorithme Hierarchical clustering (HC), dans cette configuration on désire à effectuer un test simple, nous aurons donc deux logiciels sur la même machine : client (navigateur web) et le serveur (server web), ces deux logiciels vont communiquer en utilisant le protocole HTTP. Nous avons travaillé avec le framework Python Flask afin de générer des pages web côté serveur à partir de son propre serveur web.

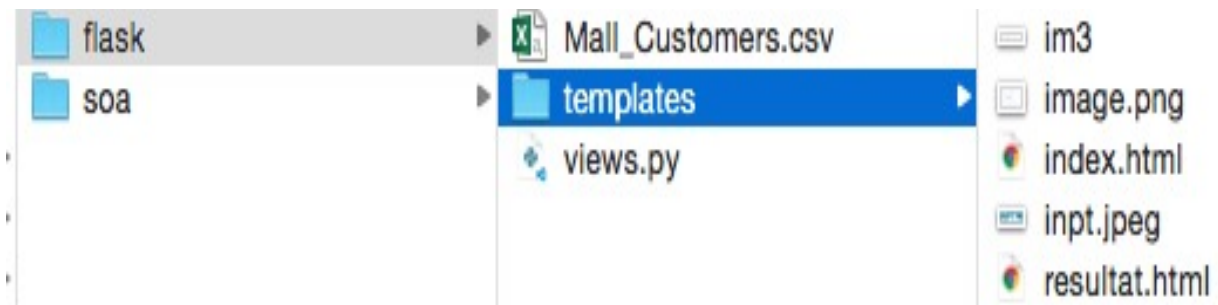
- **Langages de programmation utilisées :** framework Python Flask, Css et le langage de balises HTML
- **Environnements de travail :** Spyder, cmd, Visual studio
- **Librairies Python:** pandas, numpy, Flask, matplotlib, sklearn
- **Type de service web :** WS-* (reposant sur le standard SOAP)

I. La structure du projet :

Notre projet se compose de la structure suivante :

- **Dossier templates:** contient toutes les ressources liées à l'interface web en plus des interfaces elles-mêmes (html,css, et les images)
- **Fichier Mall_Customers.csv :** qui contient les données d'apprentissage de l'algorithme de clustering.

- **Fichier *views.py*** : contenant l'application de flask



II. Création de serveur web

Après la création d'un répertoire (flask) qui rassemble les fichiers de projet, on a créé par la suite un fichier d'extension python s'appelle "views.py", ce fichier contient les importations nécessaires : Flask, pandas numpy matplotlib..., il contient aussi deux méthodes, l'une nous permet d'avoir le dataset et l'autre d'afficher le résultat sur le navigateur. En plus, il contient le code python qui traitent le dataset puis retourner le résultat puis l'envoyer par le port 5000.

```

x temp.py x views.py
1 from flask import Flask, render_template, request
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import scipy.cluster.hierarchy as sch
6 import matplotlib.pyplot as plt
7 import pandas as pd
8 from sklearn import datasets, linear_model
9 from sklearn.model_selection import train_test_split
10 from matplotlib import pyplot as plt
11 import matplotlib.pyplot as plt
12 #%matplotlib inline
13 app = Flask(__name__)
14
15 @app.route('/', methods=['GET', 'POST'])
16 def index():
17     return render_template('index.html')
18 @app.route('/resultat', methods = ['GET'])
19 def resultat():
20
21     result=request.form
22     #data = pd.read_csv(result['nom'])
23     #p = result['prenom']
24
25     data = pd.read_csv('./Mall_Customers.csv');
26     df = pd.DataFrame(data, columns=data.columns)
27     data_train, data_test= train_test_split(df,test_size=0.2)
28     X = data_train.iloc[:, [3,4]].values
29     labels = range(1, 11)
30     plt.figure(figsize=(10, 7))
31     graph = plt.scatter(X[:,0],X[:,1], label='True Position')
32     fig = graph.get_figure()
33     filepath='./templates/image'
34     fig.savefig(filepath)
35     return render_template("resultat.html",nom=fig)
36
37 app.run(debug=True)

```

III. Interface d'input

Concernant le code de l'interface d'input, il contient deux parties : la première partie consiste à la partie CSS, et l'autre partie concerne la partie HTML. Cette dernière se compose de quatre divisions :

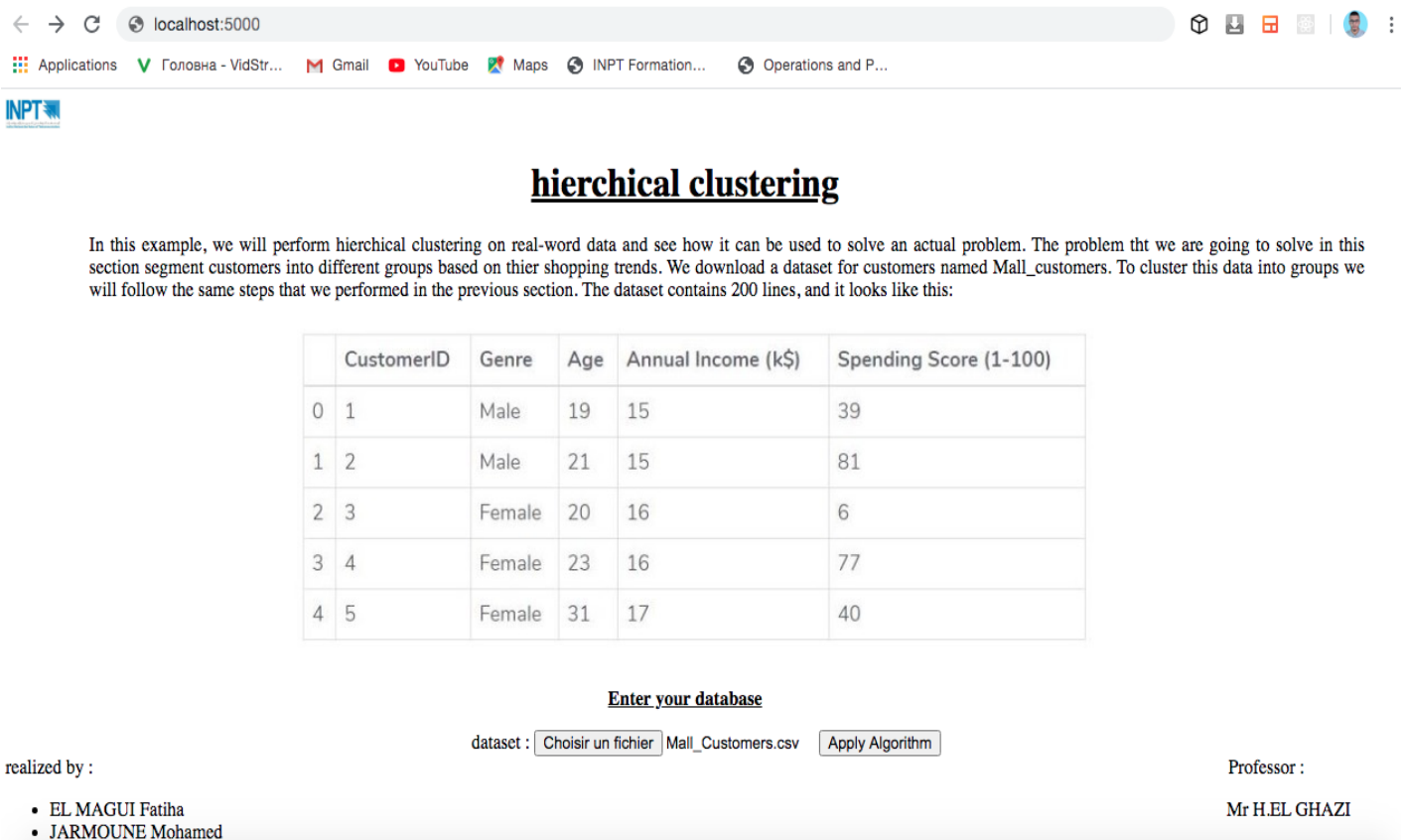
- **1ère division** : contient le titre et le logo de la page web.
- **2ème division** : se compose de la balise paragraph qui contient l'explication de l'exemple qu'on a choisi en plus de la balise image d'une capture sur notre dataset.
- **3ème division** : contient un fichier input dans lequel on choisit notre dataset dans la machine, et un bouton qui permet d'aller vers la page de résultat. Mais avant de ça, lorsqu'on clique sur le bouton, il envoie une requête http vers le serveur via le port 5000 pour traiter les dataset. Finalement, il contient les labels de notre nom et le nom de notre professeur.

```
Users > mac > Desktop > S4 > S4_P2 > SOA > flask > templates > <> index.html > html > body > div.isertion > form > inp
34     </style>
35 </head>
36 <body>
37     <div class="miseEnpage">
38
39         <div style="width: 100%;">
40             
41             <h1 style="text-align: center;"><u><strong>hierchical clustering</strong></u></h1>
42         </div>
43         <p style="margin-left: 2cm;">
44             In this example, we will perform hierchical clustering on real-word data and see how it can be us
45         </p>
46         <!--  -->
47         <!--  -->
48         
49         <br>
50         <p style="text-align: center;">
51             <u><strong> Enter your database</strong></u>
52         </p>
53     </div>
54     <div class="isertion">
55         <form action="http://localhost:5000/resultat" method="GET">
56             <label>dataset</label> : <input type="file" name="nom" />
57             <input type="submit" value="Apply Algorithm" />
58         </form>
59     </div>
60     <div class="realized">
61         realized by :
62         <ul>
63             <li>EL MAGUI Fatiha</li>
64             <li>JARMOUNE Mohamed</li>
65         </ul>
66     </div>
67     <div class="professor">
```

Par ailleurs lorsqu'on lance le serveur à partir de spyder comme le montre la figure ci-dessous :

```
In [81]: runfile('/Users/mac/Desktop/S4/S4_P2/S0A/flask/views.py', wdir='/Users/mac/Desktop/S4/S4_P2/S0A/flask')
* Serving Flask app "views" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with fsevents reloader
```

Et voici la page web de cette interface après l'envoi d'une requête http via le port 5000 de client vers le serveur, pour voir le résultat on navigue vers l'adresse `http://localhost:5000/` :



hierarchical clustering

In this example, we will perform hierarchical clustering on real-world data and see how it can be used to solve an actual problem. The problem that we are going to solve in this section segment customers into different groups based on their shopping trends. We download a dataset for customers named Mall_customers. To cluster this data into groups we will follow the same steps that we performed in the previous section. The dataset contains 200 lines, and it looks like this:

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Enter your database

dataset : Mall_Customers.csv

realized by :

- EL MAGUI Fatiha
- JARMOUNE Mohamed

Professor :

Mr H.EL GHAZI

IV. L'interface de l'output :

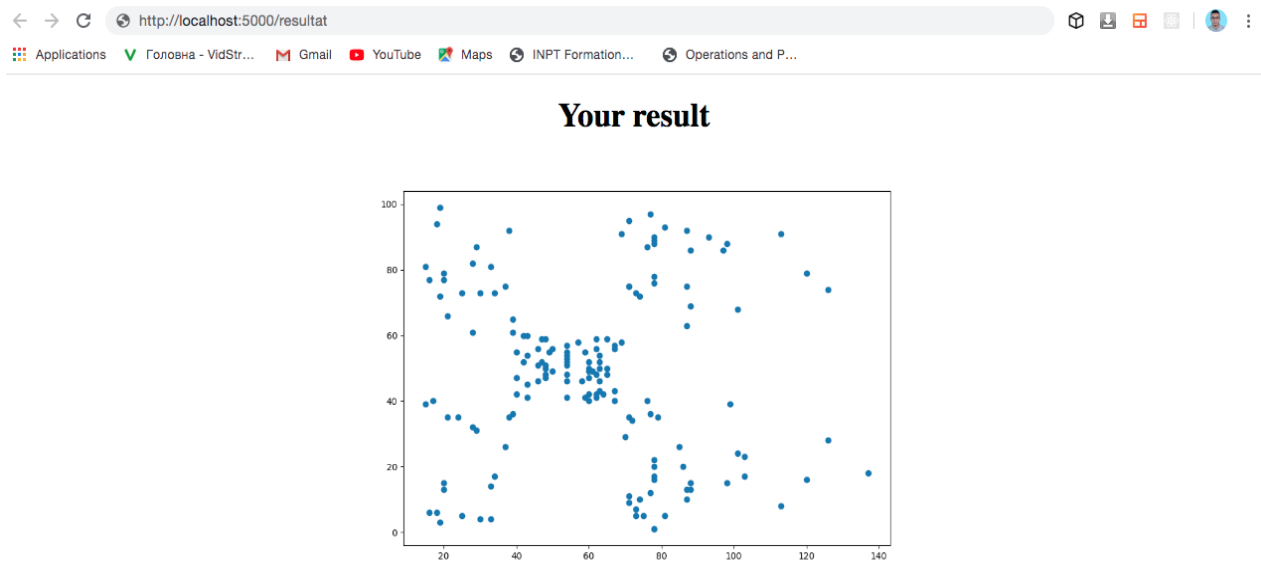
En ce qui concerne le code de l'interface d'output, elle se compose de la partie CSS et la partie HTML. Pour cette dernière, on peut seulement afficher le résultat sous forme d'une image, après d'avoir la récupérer à partir du serveur web via le port 5000 :

```

Users > mac > Desktop > S4 > S4_P2 > SOA > flask > templates > <> resultat.html > html > body > div.result
1  <!doctype html>
2  <html lang="fr">
3    <head>
4      <meta charset="utf-8">
5      <title>Résultat</title>
6      <style>
7        .result{
8          text-align: center;
9          height: 1cm;
10       }
11       .image{
12         width: 50%;
13         text-align: center;
14       }
15     </style>
16   </head>
17   <body>
18     <div class="result">
19       <h1>
20         Your result
21       </h1>
22       <!--  -->
23     </div>
24     <div class="banniere">
25       
26     </div>
27   </body>
28 </html>

```

le résultat s'affiche sur le navigateur via l'adresse <http://localhost:5000/resultat>:



Conclusion

Les clients ont bien été segmenté en différentes groupes, comme il apparut dans le graphe ci-dessus, selon leurs tendances d'achat. On peut voir que les points de données sous la forme de cinq clusters. Les points de données en bas à droite appartiennent aux clients avec des salaires élevés mais de faibles dépenses. Ce sont les clients qui dépensent soigneusement leur argent. De même, les clients en haut à droite, ce sont les clients avec des salaires élevés et des dépenses élevées. Ce sont les types de clients que les entreprises ciblent. Les clients du milieu sont ceux qui ont un revenu moyen et des salaires moyens. Le plus grand nombre de clients appartient à cette catégorie. Les entreprises peuvent également cibler ces clients étant donné qu'ils sont en grand nombre, etc.