République algérienne démocratique et populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Benyoucef BENKHEDDA-ALGIERS 1

Faculté des sciences

Département de Mathématiques et d'Informatique

Spécialité : Ingénierie des Systèmes d'Information Intelligents

---

Project Entrepôt de données

Professeur :

- Dr.Drias

Membres du groupe :

- LOUNIS Amar

- MEDJBER Hamza

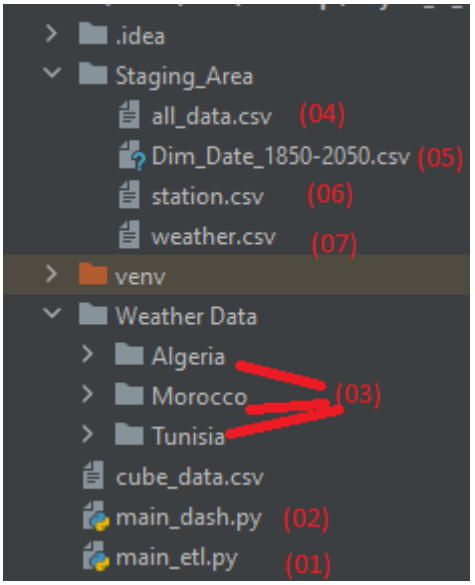
- KESRI Ahlem

- BEKHOUCHE Racha

2021/2022

Introduction

La qualité et le contenu des données concernant certains types d'informations peuvent varier dans le temps. Ces données peuvent être des données sur la santé, des données démographiques , des données sur des consommateurs ou dans notre cas de figure des données climatiques.



Pour utiliser efficacement ces données, elles doivent être collectées, traitées, stockées et fournies aux consommateurs sous forme d'informations sous différents formats. Ces derniers utiliserons ces informations et ces connaissances lors de la prise de décision dans des sujets variés comme l'agriculture, pour la mise en place de plans d'urgence, ou bien pour la recherche sur les changements climatiques.

Afin de ne pas se perdre dans la complexité du projet, et pour une compréhension simple, ci dessous une image présentant les différents fichiers python que nous avons utilisé :

I. Analyse des données :

1. Description de données

La première étape dans la construction d'un entrepôt de données est l'étude des données sources.
Notre dataset provient du "National Centers for Environmental Information" et comporte des données climatiques de trois pays d'Afrique : l'Algérie, la Tunisie et le Maroc.
Avant de commencer la conception d'un Datawarehouse, il est judicieux de commencer par analyser et améliorer la qualité des datasets.
Nous commençons l'étude de notre dataset avec l'exploration du schéma relationnel qui nous aidera à mieux comprendre nos données.
Au chargement de données nous remarquons que :

- Les données ne sont pas regroupées dans un seul fichier csv.
- Certains fichiers csv contiennent 48 attributs alors que d'autres en contiennent 12.
- Nous avons énormément de valeurs nulles (TMAX, TMIN, TAVG, RTCP).
- L'attribut "Name" de station contient deux attributs : le nom de la ville et le code du pays
- Certains fichiers Csv contiennent des attributs "WT**" qui n'existent pas dans d'autres fichiers.

2. Modélisation du schéma étoile

2.1 Diagramme de schéma en étoile

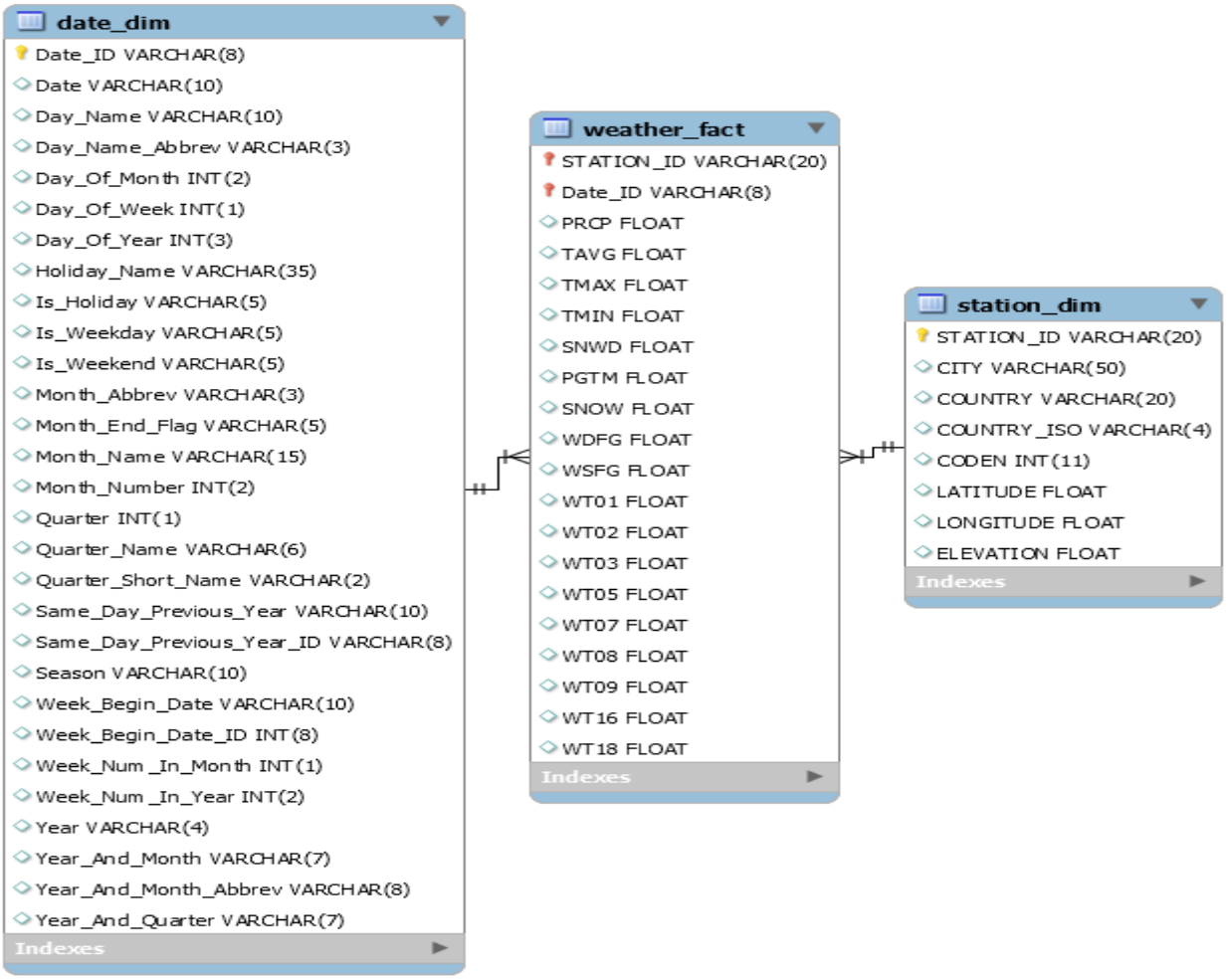


Diagramme de schéma en étoile

2.2 Description schéma en Étoile :

Notre schéma comporte une table de fait nommée **“Weather fact”** et deux tables de dimension “**Station_Dim**” et “**Date_Dim**”.

- “ Date_Dim ” est la dimension temps contenant les attributs relatif au temps. Les attributs les plus utilisés seront : La saison, le trimestre, le mois et l'année. Son niveau de détail s'arrête aux jours, ce qui ne nécessite qu’une table représentant la dimension temps (Granularité 24h).
- “Station_dim” est la dimension géographique qui contient tous les attributs décrivants la station où les faits ont été enregistrés dont les dimensions , la ville et le pays.
- “Weather fact” est la table des faits. Elle contient les mesures, c'est-à- dire, les informations enregistrées en relation avec notre sujet initial, qui est les conditions climatiques, en plus des différentes clés étrangères des dimensions (leur concaténation constitue alors la clé primaire de la table des faits.
- La granularité des dimensions temporelles : quotidienne..
- Les mesures : station_id, date_id, PRCPR, TAVG, TMAX, TMIN, SNWD, PGTM, SNOW, WDFG et WSFG.
- WTXX Il contient les valeurs 0, 1 et NULL C'est pourquoi nous l'avons mis avec les mesures

3. Implémentation d’un entrepôt de données

Notre processus d’implémentation est divisé en deux étapes :

1. extraction , traitement , chargement des données et enfin la création de cubes pour faciliter l'accès aux données (main_etl.py).
2. Création du Dash , ce dernier interrogera le cube de données(main_dash.py).

1. Processus ETL

Le processus débutera par l'importation des différents modules nécessaires pour nos processeurs puis on spécifie les différents chemins menant vers les fichiers csv nécessaires.

```
dir_weather_data = './Weather Data'
dir_ataging_area = './Staging_Area/'
if not os.path.exists(dir_ataging_area):
    os.makedirs(dir_ataging_area)

file_csv_path_all_data = './Staging_Area/all_data.csv'

file_csv_path_station = './Staging_Area/station.csv'
file_csv_path_date = './Staging_Area/Dim_Date_1850-2050.csv'
file_csv_path_weather = './Staging_Area/weather.csv'

file_csv_path_cube_data ="./cube_data.csv"
```

Suite à cela, nous allons établir la connexion vers la base de données (spécifiez votre mot de passe), toujours dans le but de faciliter les différentes manipulations nous allons créer une classe Database.

- La classe Database :

Les tables de données seront réalisées automatiquement en faisant appel à la classe "Database" qui a comme but de gérer la partie SQL en utilisant différentes fonctions, ces dernières seront définies au saint de cette classe et seront appelées à partir d'autres fonctions.

Les fonctions utilisées sont :

- def __init__(self) : le constructeur.
- def __del__(self) : suppression d'objet.
- def close(self) : fermer la base de données.
- def connect(self) : connexion à la base de données.
- def execute(self,_sql) : executer une requete sql
- def query(self,_sql) : definir la requete sql.
- def commit(self) : exécuter le commit.
- def create_db(self) : créer la base de données si elle n'existe pas.
- def drop_db(self) : supprimer la base de données.
- def create_table(self,_schema,_tableName) : créer une table.
- def create_FK(self,table_source,fk,table_ref,column_ref) : créer et relier les clés étrangères.
- def NAN_NULL(self,o) : passer du NAN au NULL.
- def insert_in_table(self,_tableName,_tableKeys,_values) : Insérer les données dans les tables.
- def populate_table(self,_csv_path,_csv_keys,_tableName,_tableKeys)

1.1 Extraction de données : Le problème que nous avons rencontré en premier lieu est que nos données sont dispersées et ont des structures différentes. Par exemple, certains fichiers possèdent des attributs alors que

d'autres non. Cela nous mène à combiner toutes les données dans un seul fichier csv que nous allons placer dans le dossier **Staging_Area.**

Pour ce faire, nous avons réalisé une fonction FilesYield() qui va elle lire tous nos fichiers csv , les extraire puis une fonction getAllFiles qui va les regrouper dans une liste.

```
def FilesYield(_dir_path,_extension):
    for root, dirs, files in os.walk(os.path.abspath(_dir_path)):
        for file in files:
            file = str(file)
            if file.endswith(_extension):
                yield os.path.join(root, file)

# ---------------------------------------------------------------------- function get list files from dir
def getAllFiles(_dir_path,_extension):
    return [f for f in FilesYield(_dir_path,_extension)]
```

Puis on passe à la fonction etl_extract() qui elle va récupérer la liste de tous les dataframes , les concaténer en un seul (All_Data) en faisant appel aux fonctions déjà mentionnées et au chemin défini en haut et qui défini le chemin du data frame file_csv_path_all_data:

```
def etl_extract():
    list_csv_path = getAllFiles(dir_weather_data,'csv')
    list_dataframes = []
    for csv_path in list_csv_path:
        print('read:' + csv_path)
        data = pd.read_csv(csv_path, sep=",", encoding='utf-8-sig', low_memory=False)
        list_dataframes.append(data)
    all_datafram = pd.concat(list_dataframes)
    print('save all file in one file')
    all_datafram.to_csv(file_csv_path_all_data, index=False, encoding='utf-8-sig')
```

1.2 Transformation de données :

Afin de détailler les informations relatives à la position géographique, nous avons créé la fonction **“etl_transformation_station ()**”. Cette dernière traitera les attributs décrivant la station géographique et en tirera de nouveaux qui nous seront utiles dans l'étape de visualisation :

```python
def etl_transformation_station():
    cols = ['STATION', 'NAME', 'LATITUDE', 'LONGITUDE', 'ELEVATION']
    data = pd.read_csv(file_csv_path_all_data, sep=",", encoding='utf-8-sig', usecols=cols, low_memory=False)
    # get all station without duplicate
    all_datafram = data.drop_duplicates(subset=['STATION'])
    # split CITY
    all_datafram.insert(1, 'CITY', all_datafram['NAME'].str.split(',', expand=True)[0])
    # split COUNTRY_CODE
    all_datafram.insert(1, 'COUNTRY_CODE', all_datafram['NAME'].str.split(',', expand=True)[1])
    # split COUNTRY_ISO
    all_datafram.insert(1, 'COUNTRY_ISO', all_datafram['COUNTRY_CODE']\
                        .replace(' AG', 'DZ')\
                        .replace(' TS', 'TN')\
                        .replace(' MO', 'MA')\
                        .replace(' SP', 'MA'))
    # extract COUNTRY_ISO
    all_datafram.insert(1, 'COUNTRY_ISO3', all_datafram['COUNTRY_CODE']\
                        .replace(' AG', 'DZA')\
                        .replace(' TS', 'TUN')\
                        .replace(' MO', 'MAR')\
                        .replace(' SP', 'MAR'))

    # # extract COUNTRY_NUMBER
    all_datafram.insert(1, 'COUNTRY_NUMBER', all_datafram['COUNTRY_CODE']\
                        .replace(' AG', '012')\
                        .replace(' TS', '788')\
                        .replace(' MO', '504')\
                        .replace(' SP', '504'))
    # extract COUNTRY_NAME
    all_datafram.insert(1, 'COUNTRY_NAME', all_datafram['COUNTRY_CODE']\
                        .replace(' AG', 'Algeria')\
                        .replace(' TS', 'Tunisia')\
                        .replace(' MO', 'Morocco')\
                        .replace(' SP', 'Morocco'))
    # save file
    all_datafram.to_csv(file_csv_path_station, index=False, encoding='utf-8-sig')
```

1- Tout d'abord, dans une liste nous mentionnons les attributs nécessaires à la description de la station, puis à l'aide de ces derniers on effectue une extraction depuis le fichier AllData.csv. Enfin, nous éliminons les lignes dupliquées.
2- Comme nous l'avons mentionné précédemment, l'attribut "Name" contient deux informations. On le divise en deux et on en extrait deux nouveaux attributs : le nom de la ville 'City' et le code du pays 'Country_Code'.
3- Grâce au code du pays que nous avons extrait, nous allons définir pour chaque pays son code ISO ,son code ISO3 , un nombre le représentant ainsi que son nom du pays.
Enfin nous allons regrouper toutes ces informations dans un fichier csv "**station.csv**" (04)

	A	B	C	D	E	F	G	H	I	J	K	L
1	STATION	COUNTRY	COUNTRY	COUNTRY	COUNTRY	COUNTRY	CITY	NAME	LATITUDE	LONGITUD	ELEVATION	
2	AGE00147	Algeria	12	DZA	DZ	AG	FORT NAT	FORT NAT	36.63	4.2	942	
3	AGE00147	Algeria	12	DZA	DZ	AG	TIZI OUZO	TIZI OUZO	36.72	4.05	222	
4	AGE00147	Algeria	12	DZA	DZ	AG	LAGHOUA	LAGHOUA	33.7997	2.89	767	
5	AGE00147	Algeria	12	DZA	DZ	AG	ORLEANS\	ORLEANS\	36.17	1.34	112	

Pour les données relatives au temps, nous avons créé un fonction "**etl_transformation_weather()"** qui va traiter et stocker ces dernières dans le fichier "weather.csv" (05) comme suit :

```python
def etl_transformation_weather():
    cols = ['STATION', 'DATE', 'PRCP', 'TAVG', 'TMAX', 'TMIN', 'SNWD', 'PGTM', 'SNOW', 'WDFG', 'WSFG']
    cols = cols + ['WT01','WT02','WT03','WT05','WT07','WT08','WT09','WT16','WT18'] # wtXX
    data = pd.read_csv(file_csv_path_all_data, sep=",", encoding='utf-8-sig', usecols=cols, low_memory=False)
    # replae NAN values in colun PRCP by mean
    data['PRCP'].fillna(0, inplace=True)
    # replae NAN values in colun TMAX by mean
    data['TMAX'].fillna(data['TMAX'].mean(), inplace=True)
    # replae NAN values in colun TMIN by mean
    data['TMIN'].fillna(data['TMIN'].mean(), inplace=True)
    # replae NAN values in colun TAVG by TMAX+TMIN /2
    data['TAVG'].fillna(data['TMAX'] + data['TMIN'] / 2, inplace=True)
    data.insert(1, 'Date_ID', data['DATE'].str.replace('-', ''))
    all_datafram = data.round(1)

    # save file
    all_datafram.to_csv(file_csv_path_weather, index=False, encoding='utf-8-sig')
    print('all success weather ')
```

1- Premièrement nous allons définir des vecteurs contenant les attributs nécessaires que nous allons extraire de notre fichier csv AllData.
2- Nous remplaçons les valeurs manquantes dans ‘PRCP’, ‘TMAX’ et ‘TMIN’ avec la moyenne.
3- Nous remplaçons les données manquantes dans ‘TAGV’ la moyenne de température en effectuant une addition entre TMAX(température maximale) et TMIN(température minimale) que nous divisons par 2.
4- La création de l’identifiant de la date est une concaténation entre le mois, le jour et l'année que nous réalisons en éliminant le ‘-’ .
5- Enfin nous allons regrouper tout cela dans un seul fichier weather.csv.

1.3 Création du Datawarehouse :

Nous allons utiliser pour la création du datawarehouse des schémas de tables que nous avons défini avant et qui sont les suivants:

Schéma de la table WEATHER_FACT

```
'STATION_ID':'VARCHAR(20)',
'Date_ID':'VARCHAR(8)',
'PRCP':'float',
'TAVG':'float',
'TMAX':'float',
'TMIN':'float',
'SNWD':'float',
'PGTM':'float',
'SNOW':'float',
'WDFG':'float',
'WSFG':'float',
'WT01':'float',
'WT02':'float',
'WT03':'float',
'WT05':'float',
'WT07':'float',
'WT08':'float',
'WT09':'float',
'WT16':'float',
'WT18':'float',
'':'PRIMARY KEY(STATION_ID,Date_ID)'
}
```

Schéma de la table STATION_DIM

```
schema_table_station_dim = {
'STATION_ID':'VARCHAR(20)',
'CITY':'VARCHAR(50)',
'COUNTRY':'VARCHAR(20)',
'COUNTRY_ISO':'VARCHAR(4)',
'CODEN':'INT',
'LATITUDE':'float',
'LONGITUDE':'float',
'ELEVATION':'float',
'':'PRIMARY KEY(STATION_ID)'
}
```

Schéma de la table DATE_DIM

```
schema_table_date_dim = {
'Date_ID':'VARCHAR(8)',
'Date':'VARCHAR(10)',
'Day_Name':'VARCHAR(10)',
'Day_Name_Abbrev':'VARCHAR(3)',
'Day_Of_Month':'INT(2)',
'Day_Of_Week':'INT(1)',
'Day_Of_Year':'INT(3)',
'Holiday_Name':'VARCHAR(35)',
'Is_Holiday':'VARCHAR(5)',
'Is_Weekday':'VARCHAR(5)',
'Is_Weekend':'VARCHAR(5)',
'Month_Abbrev':'VARCHAR(3)',
'Month_End_Flag':'VARCHAR(5)',
'Month_Name':'VARCHAR(15)',
'Month_Number':'INT(2)',
'Quarter':'INT(1)',
'Quarter_Name':'VARCHAR(6)',
'Quarter_Short_Name':'VARCHAR(2)',
'Same_Day_Previous_Year':'VARCHAR(10)',
'Same_Day_Previous_Year_ID':'VARCHAR(8)',
'Season':'VARCHAR(10)',
'Week_Begin_Date':'VARCHAR(10)',
'Week_Begin_Date_ID':'INT(8)',
'Week_Num_In_Month':'INT(1)',
'Week_Num_In_Year':'INT(2)',
'Year':'VARCHAR(4)',
'Year_And_Month':'VARCHAR(7)',
'Year_And_Month_Abbrev':'VARCHAR(8)',
'Year_And_Quarter':'VARCHAR(7)',
'':'PRIMARY KEY(Date_ID)'
}
```

au sein de la fonction create_datawarhouse() qui est la suivante:

```
def create_datawarehows():
    db = Database()
    db.drop_db()
    db.create_db()
    db.connect()

    db.create_table(schema_table_station_dim,'station_dim')
    db.create_table(schema_table_weather_fact,'weather_fact')
    db.create_table(schema_table_date_dim,'date_dim')

    db.create_FK('weather_fact','STATION_ID','station_dim','STATION_ID')
    db.create_FK('weather_fact','DATE_ID','date_dim','DATE_ID')

# ------------------------------------------------------------------------
```

On commence par instancier un objet db de la classe Database que nous avons défini en haut. Cet objet va nous permettre d'exécuter les différentes méthodes que comporte cette classe et qui permettent de :
1- effectuer une connexion avec la base de données.
2-créer les différentes dimensions ‘station_dim’ , ‘weather_fact’ et ‘date_dim’ tout en faisant appel aux shemas de tables déjà définis.
3- enfin nous allons établir les relations entre les dimensions et la table de faits à l'aide de la fonction crete_FK qui elle va définir les clés étrangères.

1.4. Remplissage du datawarehouse:

Nous procédons maintenant au remplissage des tables etl_load(), toujours à l'aide de la classe Database qui va permettre la connexion a ;la base de données le remplissage des dimensions et de la tables de faits se fait à partir des différentes tables que nous avons défini lors du traitement de données il suffit juste de mentionner le chemin menant vers ses derniers , chemins que nous avons déjà défini en haut.

```
def etl_load():
    db = Database()
    db.connect()

    # --------------------------
    csv_keys_station_dim = ['STATION', 'CITY', 'COUNTRY_NAME', 'COUNTRY_ISO3', 'COUNTRY_NUMBER', 'LATITUDE',
                            'LONGITUDE', 'ELEVATION']
    table_keys_station_dim = list(schema_table_station_dim.keys())[:-1]
    db.populate_table(file_csv_path_station, csv_keys_station_dim, 'station_dim', table_keys_station_dim)
    # --------------------------
    csv_keys_date_dim = list(schema_table_date_dim.keys())[:-1]
    table_keys_date_dim = list(schema_table_date_dim.keys())[:-1]

    db.populate_table(file_csv_path_date, csv_keys_date_dim, 'date_dim', table_keys_date_dim)
    # --------------------------
    csv_keys_weather_fact = ['STATION', 'Date_ID', 'PRCP', 'TAVG', 'TMAX', 'TMIN', 'SNWD', 'PGTM', 'SNOW', 'WDFG',
    csv_keys_weather_fact = csv_keys_weather_fact + ['WT01', 'WT02', 'WT03', 'WT05', 'WT07', 'WT08', 'WT09', 'WT16'
    table_keys_weather_fact = list(schema_table_weather_fact.keys())[:-1]

    db.populate_table(file_csv_path_weather, csv_keys_weather_fact, 'weather_fact', table_keys_weather_fact)
    # --------------------------
    db.close()
```

1.5 Création des cubes de données :

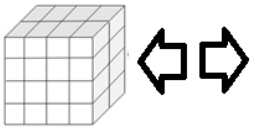
En raison de la grande quantité de données les requêtes prennent beaucoup de temps à s'exécuter , nous avons pensé à la création d'un cube multidimensionnel afin réduire la pression sur le SGBD.

```
def create_multidimensional_cube():
    db = Database()
    db.connect()
    db.execute("SET GLOBAL sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY',''));")
    db.commit()

    sql = 'SELECT station_dim.*,date_dim.Year,date_dim.Season, ' \
          'ROUND(AVG(weather_fact.PRCP),2) as AVG_PRCP , ' \
          'ROUND(AVG(weather_fact.TAVG),2) as AVG_TAVG , ' \
          'ROUND(AVG(weather_fact.TMAX),2) as AVG_TMAX , ' \
          'ROUND(AVG(weather_fact.TMIN),2) as AVG_TMIN   ' \
          'FROM weather_fact,station_dim ,date_dim ' \
          'WHERE weather_fact.STATION_ID = station_dim.STATION_ID ' \
          'AND weather_fact.Date_ID = date_dim.Date_ID ' \
          'group BY weather_fact.STATION_ID ,date_dim.Year,date_dim.Season;'

    df = db.query(sql)
    db.close()

    print('cube : size :', df.size)
    print('cube : columns :', df.columns)
    df.to_csv(file_csv_path_cube_data, index=False, encoding='utf-8-sig')
```

1- A l'aide de la classe Database on effectue une connexion à la base de données. Puis on utilise une configuration du **"GROUP BY global".**

2- Dû au fait que le volume de données présent dans notre entrepôt est important, on sélectionne que les informations nécessaires avec une requête sql.

3- On enregistre les données dans le fichier : "**Cube_Data.csv".**



STATION_	CITY	COUNTRY	COUNTRY	CODEN	LATITUDE	LONGITUD	ELEVATIO	Year	Season	AVG_PRC	AVG_TAV	AVG_TMA	AVG_TMIN
AG000060	ALGER DA	Algeria	DZA	12	36.7167	3.25	24	1940	Fall	1.78	27.47	21.98	10.98
AG000060	ALGER DA	Algeria	DZA	12	36.7167	3.25	24	1940	Spring	1.57	27.75	22.84	9.82
AG000060	ALGER DA	Algeria	DZA	12	36.7167	3.25	24	1940	Summer	0.15	37.81	29.64	16.34
AG000060	ALGER DA	Algeria	DZA	12	36.7167	3.25	24	1940	Winter	2.61	21.27	17.83	6.88
dimension station								dimension temporal		les mesure			

enfin nous allons effectuer un appel de toutes les fonctions déjà définies afin de lancer leur exécution.

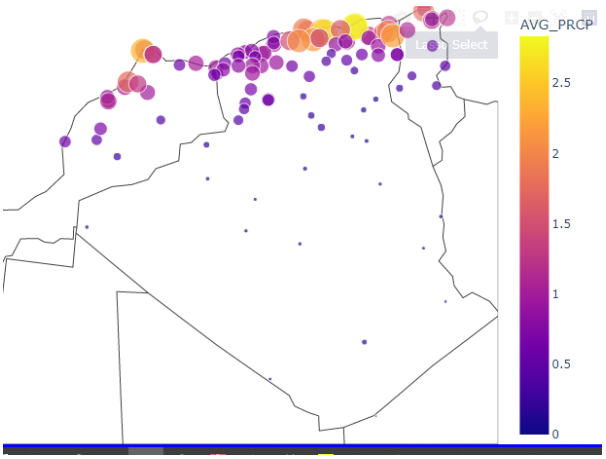
2. Visualisation des données avec dash:

Après l'importation des différents modules nécessaires aux différentes étapes de la création du dashboard, on commence par importer le fichier csv cube_data.csv , le lire puis On crée des listes pour interroger le cube et ainsi faciliter la création des différents filtres. On a besoin alors d'un filtre pour le pays, la ville, la saison et l'année.

```
file_csv_path_cube_data ="./cube_data.csv"

data = pd.read_csv(file_csv_path_cube_data)

selected_as = 'AVG_PRCP'

list_COUNTRY_GLOBAL = data.loc[~data['COUNTRY_ISO'].duplicated(), 'COUNTRY_ISO'].tolist()
list_CITY_GLOBAL = data.loc[~data['CITY'].duplicated(), 'CITY'].tolist()
list_YEAR_GLOBAL = data.loc[~data['Year'].duplicated(), 'Year'].tolist()
list_SEASON_GLOBAL = data.loc[~data['Season'].duplicated(), 'Season'].tolist()
```

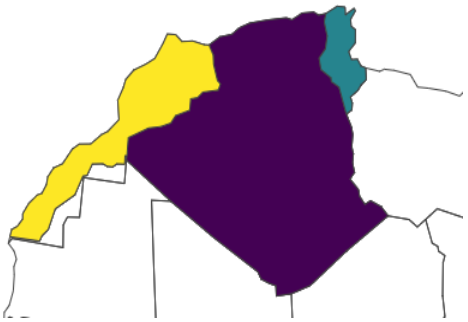
```python
def select_data(_data, list_country,list_CITY, list_Season, years):
    dff = _data.copy()
    #
    if (list_country == [] or list_country == None):
        list_country = list_COUNTRY_GLOBAL
    dff = dff[dff['COUNTRY_ISO'].isin(list_country)]          (01)
    #
    if (list_CITY == [] or list_CITY == None):
        list_CITY = list_CITY_GLOBAL                          (02)
    dff = dff[dff['CITY'].isin(list_CITY)]
    #
    if (years == [] or years == None):
        years = list_YEAR_GLOBAL                              (03)
    dff = dff[dff['Year'].isin(years)]
    #
    if (list_Season == [] or list_Season == None):
        list_Season = list_SEASON_GLOBAL                      (04)
    dff = dff[dff['Season'].isin(list_Season)]
    #
    return dff
```

Le reste du code nous permet de mettre en place le dashboard
Ci dessous, des parties du code qui ont permis de mettre en place les différentes représentation :



Filtrage par ville

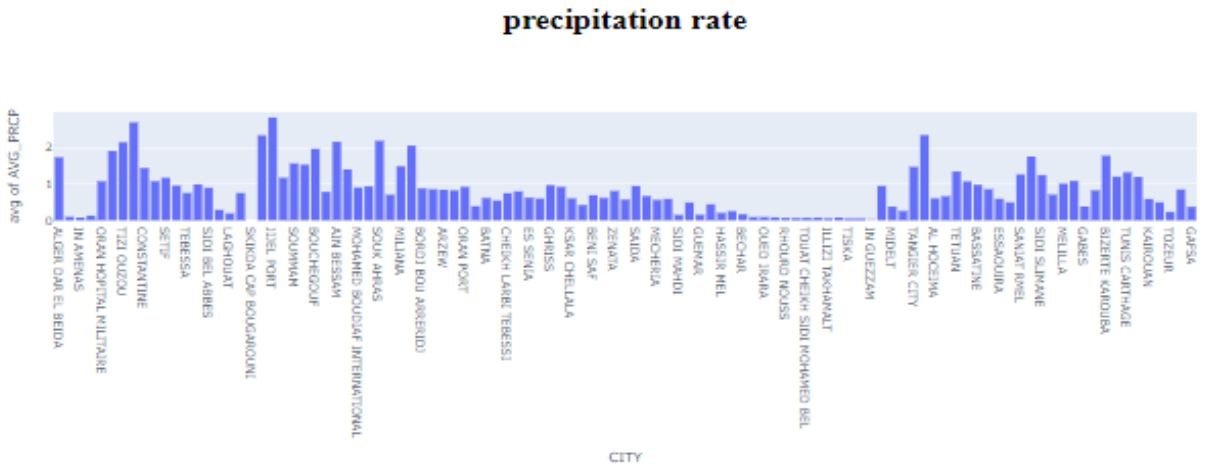
```python
fig = px.scatter_geo(dff,
    lat='LATITUDE',
    lon='LONGITUDE',
    scope="africa",
    size=selected_as,
    color=selected_as,
    template='plotly_white'
    )
```



Filtrage par pays

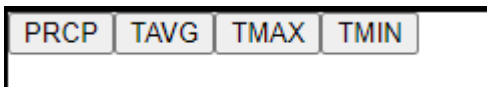
```python
fig = px.choropleth(
    data_frame=dff,
    locations="COUNTRY_ISO",
    color=selected_as,
    scope="africa",
    hover_name="COUNTRY",
    template='plotly_white',
    color_continuous_scale=
    px.colors.sequential.Viridis_r)
```

```
fig = px.histogram(dff, x="CITY", y=selected_as, histfunc='avg')
```



l'histogramme

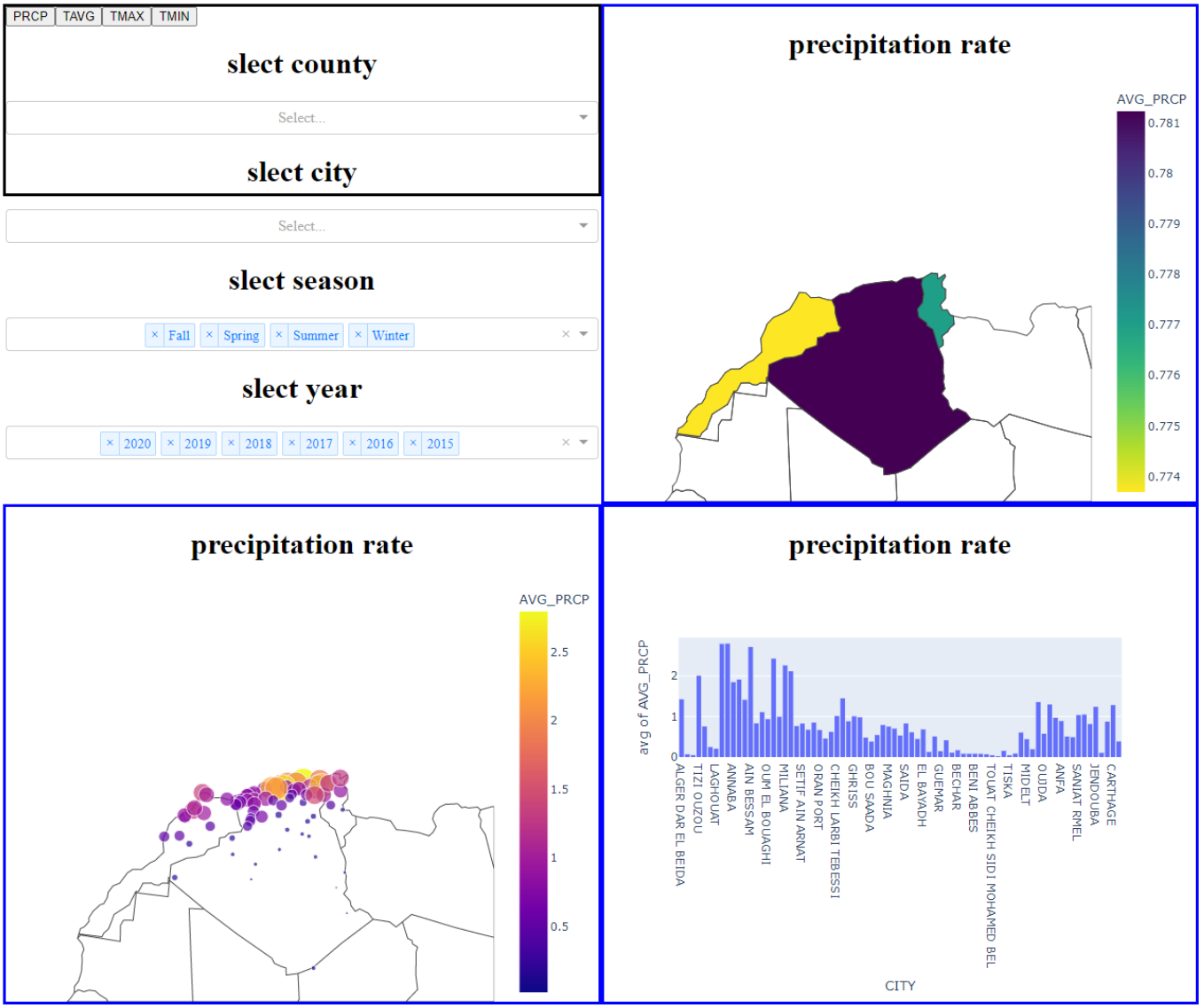
On cherche alors à calculer les mesures suivantes:



- la moyenne de Précipitation (PRCP)
- la moyenne de Température moyenne (TAVG)
- la moyenne de Température maximale (TMAX)
- la moyenne de Température minimale (TMIN)

Si on ne spécifie rien, cela signifie que toutes les mesures seront sélectionné

Exemple d'exécution :



Conclusion :

La réalisation d'un entrepôt de données est un processus contenant plusieurs étapes :

En commençant par l'extraction et le nettoyage de données, qui permettra de déterminer la qualité de la décision finale, vers le schéma en étoile et le chargement dans le datawarehouse. Enfin la visualisation de ces données traitées, sous différentes formes représentatives comme les cartes et les courbes, pourra apporter de nouvelles informations et connaissances à l'utilisateur dans le but d'améliorer ses décisions.

ENVIRONNEMENT DE DÉVELOPPEMENT :
PYTHON 3.10
MYSQL SERVER 5.7

BIBLIOTHÈQUE UTILISÉE :
PYMYSQL

DASH
PLOTLY 5.8.2
PANDAS
NUMPY

Contribution de chaque étudiant :

Etape du projet	Les étudiants
Conception du schéma en étoile	**KESRI Ahlem +** **BEKHOUCHE Racha +** **LOUNIS Amar +** **MEDJBER Hamza +**
etl_extract(); **etl_transformation_station()** **etl_transformation_weather()**	**LOUNIS Amar +** **MEDJBER Hamza +** **BEKHOUCHE Racha**
Class Database	**LOUNIS Amar**
create_datawarehows() **etl_load()** **create_multidimensional_cube()** **main_dash.py**	**LOUNIS Amar +** **KESRI Ahlem**
Rapport	**KESRI Ahlem +** **BEKHOUCHE Racha +** **LOUNIS Amar +** **MEDJBER Hamza +**