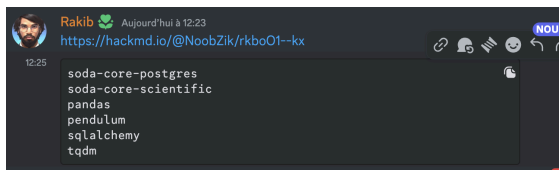


TP QUALITÉ DE DONNÉE

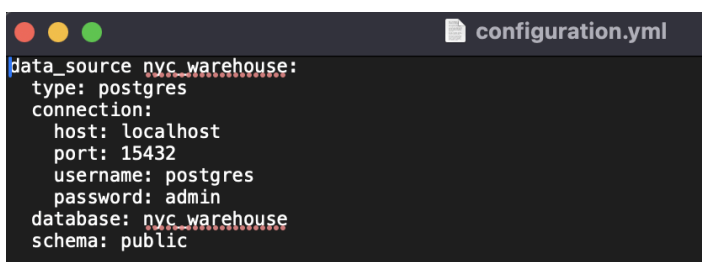
fichier requirements.txt



installer python via miniconda (environnement virtuelle)

Exercice 1 :

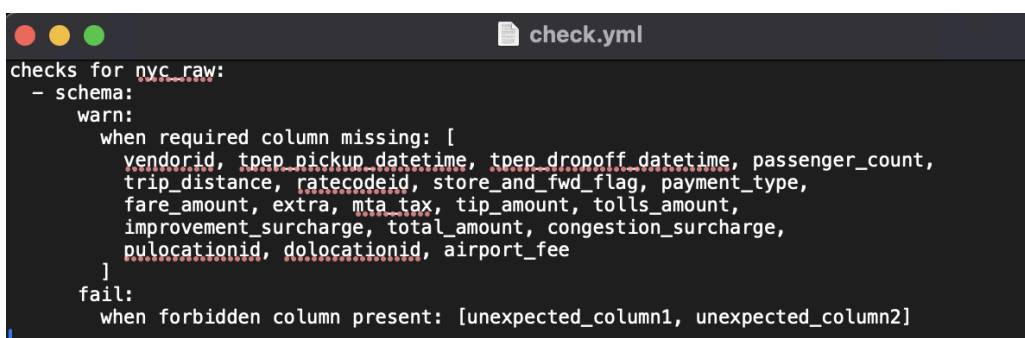
Création de configuration.yml :



Connexion et test de la bonne configuration de soda

```
[nicodeme@mbp-de-nicodeme Desktop % soda test-connection -d nyc_warehouse -c configuration.yml  
[15:04:03] Soda Core 3.4.1  
Successfully connected to 'nyc_warehouse'.  
Connection 'nyc_warehouse' is valid.  
nicodeme@mbp-de-nicodeme Desktop %
```

Création de check.yml :



Exécuter le check soda :

```
nicodeme@mbp-de-nicodeme Desktop % soda scan -d nyc_warehouse -c configuration.yml check.yml
```

```
[18:06:28] Soda Core 3.4.1
[18:06:28] Scan summary:
[18:06:28] 1/1 check PASSED:
[18:06:28]   nyc_raw in nyc_warehouse
[18:06:28]   Schema Check [PASSED]
[18:06:28] All is good. No failures. No warnings. No errors.
nicodeme@mbp-de-nicodeme Desktop %
```

```
nicodeme@mbp-de-nicodeme Desktop % soda scan -d nyc_warehouse -c configuration.yml check.yml -V
```

```
[18:06:54] Soda Core 3.4.1
[18:06:54] Reading configuration file "configuration.yml"
[18:06:54] Reading SodaCL file "check.yml"
[18:06:54] Scan execution starts
[18:06:54] Postgres connection properties: host="localhost", port="15432", database="nyc_warehouse", user="postgres", options="-c search_path=public", connection_timeout="None"
[18:06:54] Query 1.nyc_warehouse.nyc_raw.schema[nyc_raw]:
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE lower(table_name) = 'nyc_raw'
AND lower(table_catalog) = 'nyc_warehouse'
AND lower(table_schema) = 'public'
ORDER BY ORDINAL_POSITION
[18:06:54] Scan summary:
[18:06:54] 1/1 query OK
[18:06:54] 1.nyc_warehouse.nyc_raw.schema[nyc_raw] [OK] 0:00:00.007881
[18:06:54] 1/1 check PASSED:
[18:06:54]   nyc_raw in nyc_warehouse
[18:06:54]   Schema Check [check.yml] [PASSED]
[18:06:54]   schema_measured = [vendorid integer, tpep_pickup_datetime timestamp without time zone, tpep_dropoff_datetime timestamp without time zone, passenger_count integer, trip_distance double precision, ratecodeid integer, store_and_fwd_flag text, payment_type integer, fare_amount double precision, extra double precision, mta_tax double precision, tip_amount double precision, tolls_amount double precision, improvement_surcharge double precision, total_amount double precision, congestion_surcharge double precision, pulocationid integer, dolocationid integer, airport_fee double precision]
[18:06:54] All is good. No failures. No warnings. No errors.
nicodeme@mbp-de-nicodeme Desktop %
```

Que remarques-t-on par rapport à ces commandes ?

La commande sans le -V se concentre uniquement sur le résumé du scan (nombre de checks, erreurs..) ainsi que les détails du warning/errors (colonnes manquantes s'il y a). Donc elle sert à faire un résumé rapide pour une simple vérification.

La commande avec le -V permet quant à elle de fournir plus de détails sur le processus d'exécution comme la lecture des fichiers utilisés sous forme de confirmation; les détails de la connexion de PostgreSQL avec l'hôte, port, base de données, user, options ...) ;

Requêtes SQL exécutée et son temps d'exécution "0.007881":

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE lower(table_name) = 'nyc_raw'
AND lower(table_catalog) = 'nyc_warehouse'
AND lower(table_schema) = 'public'
ORDER BY ORDINAL_POSITION
```

Elle affiche aussi les colonnes présentes dans la table.

Le -V est idéal pour diagnostiquer des problèmes ou analyser en détail.

Le mot PASSED signifie que le check a réussi sans erreur ni warning, ce qui indique donc que la table nyc_raw contient toutes les colonnes attendues et définies dans le fichier check.yml.

Test d'une nouvelle règle "row_count" :

```
check.yml
#checks for nyc_raw:
# - schema:
#   warn:
#     when required column missing: [ column_name ]
#   fail:
#     when forbidden column present: [ column_name, column_name2 ]

checks for nyc_raw:
- row_count:
  warn: when > 90
  fail: when = 0
```

```
nicodeme@mbp-de-nicodeme Desktop % soda scan -d nyc_warehouse -c configuration.yml check.yml -V

[15:58:28] Soda Core 3.4.1
[15:58:28] Reading configuration file "configuration.yml"
[15:58:28] Reading SodaCL file "check.yml"
[15:58:28] Scan execution starts
[15:58:28] Postgres connection properties: host="localhost", port="15432", database="nyc_warehouse", user="postgres", options="--c search_path=public", connection_timeout="None"
[15:58:28] Query 1.nyc_warehouse.nyc_raw.aggregation[0]:
SELECT
  COUNT(*)
FROM public.nyc_raw
[15:58:33] Scan summary:
[15:58:33] 1/1 query OK
[15:58:33] 1.nyc_warehouse.nyc_raw.aggregation[0] [OK] 0:00:05.091240
[15:58:33] 1/1 check WARNED:
[15:58:33]   nyc_raw in nyc_warehouse
[15:58:33]     row_count warn when > 90 fail when = 0 [check.yml] [WARNED]
[15:58:33]     check.value: 26388179
[15:58:33] Only 1 warning. 0 failure. 0 errors. 0 pass.
nicodeme@mbp-de-nicodeme Desktop %
```

Que remarques-t-on ?

La règle a déclenché un warning car le nombre de lignes dépasse 90. J'ai eu 1 warning et 0 failure/ errors. Ce qui signifie que la règle a été analysée correctement sans problème et il n'y a pas eu d'échec car le nombre de lignes n'est pas égal à 0 , elle contient 26 388 179 lignes.

Le nombre de lignes de la table est connu grâce à check_value.

Cette règle permet de valider la qualité des données en vérifiant le contenu d'une table.

Exercice 2:

Ce premier check permet de vérifier la qualité des colonnes tpep_dropoff_datetime et tpep_pickup_datetime (les colonnes des dates et heures de début et de fin des trajets) et de s'assurer que moins de 1% des valeurs dans ces colonnes sont invalides. Une valeur est dite invalide si elle est nulle (manque de données), si elle contient un format incorrect ou qu'elle dépasse une plage attendue.

```
dateval.yml
1 checks for nyc_raw:
2   - invalid_percent(tpep_dropoff_datetime) < 1%
3   - invalid_percent(tpep_pickup_datetime) < 1%
```

```

nicodeme@mbp-de-nicodeme Desktop % soda scan -d nyc_warehouse -c /Users/nicodeme/ATL-Datamart/src/data/configuration.yml dateval.yml -V
[00:43:19] Soda Core 3.4.1
[00:43:19] Reading configuration file "/Users/nicodeme/ATL-Datamart/src/data/configuration.yml"
[00:43:19] Reading SodaCL file "dateval.yml"
[00:43:19] Scan execution starts
[00:43:19] Postgres connection properties: host="localhost", port="15432", database="nyc_warehouse", user="postgres", options="--c search_path=public", connection_timeout="None"
[00:43:19] Counting invalid without valid or invalid specification does not make sense. ("invalid_percent(tpdp_dropoff_datetime) < 1%" @ line=2,col=5 in dateval.yml)
[00:43:19] Counting invalid without valid or invalid specification does not make sense. ("invalid_percent(tpdp_pickup_datetime) < 1%" @ line=3,col=5 in dateval.yml)
[00:43:19] Query 1.nyc_warehouse.nyc_raw.aggregation[0]:
SELECT
  COUNT(*),
  COUNT(CASE WHEN FALSE THEN 1 END),
  COUNT(CASE WHEN FALSE THEN 1 END)
FROM public.nyc_raw
[00:43:28] Scan summary:
[00:43:28] 1/1 query OK
[00:43:28] 1.nyc_warehouse.nyc_raw.aggregation[0] [OK] 0:00:08.714609
[00:43:28] 2/2 checks PASSED:
[00:43:28]   nyc_raw in nyc_warehouse
[00:43:28]     invalid_percent(tpdp_dropoff_datetime) < 1% [dateval.yml] [PASSED]
[00:43:28]       check_value: 0.0
[00:43:28]       row_count: 26388179
[00:43:28]       invalid_count: 0
[00:43:28]     invalid_percent(tpdp_pickup_datetime) < 1% [dateval.yml] [PASSED]
[00:43:28]       check_value: 0.0
[00:43:28]       row_count: 26388179
[00:43:28]       invalid_count: 0
[00:43:28] All is good. No failures. No warnings. No errors.
nicodeme@mbp-de-nicodeme Desktop %

```

Ce check vérifie qu'il n'y ait aucune valeur manquante dans la colonne vendorid

```

Vendorval.yml
1 checks for nyc_raw:
2   - missing_count(vendorid) = 0

```

```

nicodeme@mbp-de-nicodeme Desktop % soda scan -d nyc_warehouse -c /Users/nicodeme/ATL-Datamart/src/data/configuration.yml Vendorval.yml -V
[01:55:04] Soda Core 3.4.1
[01:55:04] Reading configuration file "/Users/nicodeme/ATL-Datamart/src/data/configuration.yml"
[01:55:04] Reading SodaCL file "Vendorval.yml"
[01:55:04] Scan execution starts
[01:55:04] Postgres connection properties: host="localhost", port="15432", database="nyc_warehouse", user="postgres", options="--c search_path=public", connection_timeout="None"
[01:55:04] Query 1.nyc_warehouse.nyc_raw.aggregation[0]:
SELECT
  COUNT(CASE WHEN vendorid IS NULL THEN 1 END)
FROM public.nyc_raw
[01:55:10] Scan summary:
[01:55:10] 1/1 query OK
[01:55:10] 1.nyc_warehouse.nyc_raw.aggregation[0] [OK] 0:00:06.081488
[01:55:10] 1/1 check PASSED:
[01:55:10]   nyc_raw in nyc_warehouse
[01:55:10]     missing_count(vendorid) = 0 [Vendorval.yml] [PASSED]
[01:55:10]       check_value: 0
[01:55:10] All is good. No failures. No warnings. No errors.
nicodeme@mbp-de-nicodeme Desktop %

```

Ce check permet de vérifier qu'il n'y ait aucune valeur manquante dans les colonnes pulocationid et dolocationid.

```

checkidlocalisation.yml
1 checks for nyc_raw:
2   - missing_count(pulocationid) = 0
3   - missing_count(dolocationid) = 0
4

```

Ce check permet de vérifier qu'il n'y ait aucune valeur manquante dans la colonne passenger_count.

```

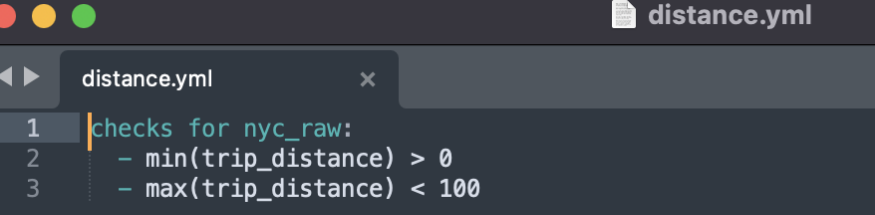
check_passengercount.yml
1 checks for nyc_raw:
2   - missing_count(passenger_count) = 0
3

```

Ce check garantit que les distances sont raisonnables et cohérentes avec ce que l'on peut attendre d'un service de taxi dans une ville comme New York. Cela permet de détecter des anomalies ou des erreurs dans les données.

`min(trip_distance) > 0`. Cette règle s'assure qu'aucune valeur de la colonne `trip_distance` n'est inférieure ou égale à zéro.

`max(trip_distance) < 100`. Cette règle vérifie que la distance maximale parcourue reste inférieure à 100 miles.



```
distance.yml
1 checks for nyc_raw:
2   - min(trip_distance) > 0
3   - max(trip_distance) < 100
```