Data Engineer Test

Ajit Malik

For this assessment I have used Airflow to orchestrate pipelines to download raw data files and store them into postresDB.

Jupyter Notebook is used for Analysis and its the pdf of the same.

Docker Containers are used as infra and solution can be run on any machine with docker installed, below I will list the steps to run the solution.

All the ouput files will be generated in MightyDataEngineeringTest/output/

Steps to run solution using docker

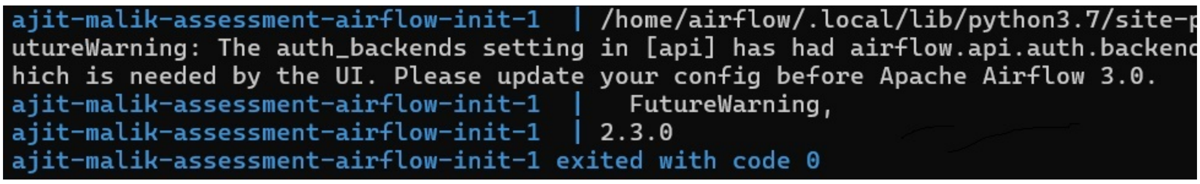
Navigate to " MightyDataEngineeringTest/ " folder

Run command : `"docker compose up airflow-init"`

Wait for it to exit with code 0

In [3]:
```
from IPython.display import Image
Image(filename=r'/home/jovyan/screenshots/airflow-init-op.jpg')
```

Out[3]:



```
ajit-malik-assessment-airflow-init-1  | /home/airflow/.local/lib/python3.7/site-p
utureWarning: The auth_backends setting in [api] has had airflow.api.auth.backend
hich is needed by the UI. Please update your config before Apache Airflow 3.0.
ajit-malik-assessment-airflow-init-1  |   FutureWarning,
ajit-malik-assessment-airflow-init-1  | 2.3.0
ajit-malik-assessment-airflow-init-1 exited with code 0
```

Run command : `"docker compose up -d --build"`

It will create all needed containers.

Airflow :

Access at http://localhost:8080/ , it might load after few mins after compose up command

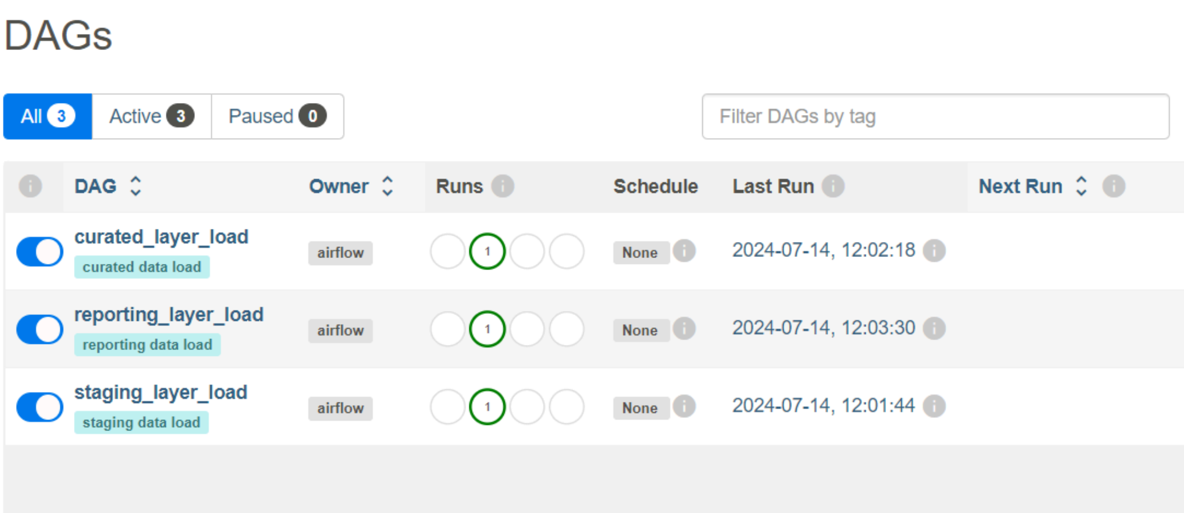
username : `airflow`

password : `airflow`

Trigger the staging_layer_load dag and let it comlete, This will load the files into PostgresDB for analysis.

In [4]:
```python
Image(filename=r'/home/jovyan/screenshots/airflow.png')
```

Out[4]:



PostgreDB Access : (from local machine)

host : `localhost`

port : `5431`

DB : `mydb`

username : `db_user`

password : `pwd`

Other containers use port 5432 to connect to this DB in same network.

Jupyter Notebook :

Now we can start with the Analysis !!

Access at http://localhost:8888/

password : `pwd`

If asking for token, please copy token from Jupyter container log, and login.

open file "notebook/code-assessment.ipynb"

Below code cells can be run by selecting the cell and `ctrl+enter`

To select next cell after selected cell run by `shift+enter`

some command can take long to execute, wait for execution to complete.

In [5]:
```python
from sqlalchemy import create_engine
import pandas as pd
from IPython.display import Image
import psycopg2
from configparser import ConfigParser

config_path=r'/home/jovyan/config/'  # DB creds stored in a config file

def config(filename, section):       # function read config file and op params
    parser = ConfigParser()
    parser.read(filename)

    db = {}
    if parser.has_section(section):
        params = parser.items(section)
        for param in params:
            db[param[0]] = param[1]

    return db


def get_engine(schema='public'):          # function returns sql connection used by
    dbschema=schema
    param=config(filename=config_path+r'database.ini', section='postgresql')
    db_uri='postgresql+psycopg2://'+param['user']+':'+param['password']+'@'+param['
    engine=create_engine(db_uri,connect_args={'options': '-csearch_path={}'.format(
    return engine
```

1. Data Cleaning

Data is cleaned and new columns generated using python and required final output generated.

I have written statements in Jupyter Notebook for easy debugging, These statements can be packaged in one script file.

In [6]:
```python
procedures=pd.read_csv(r'/home/jovyan/raw_files/procedures.csv') # Read Input data
```

In [7]:
```python
procedures.info() # checking counts and datatypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47701 entries, 0 to 47700
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   START              47701 non-null  object 
 1   STOP               47701 non-null  object 
 2   PATIENT            47701 non-null  object 
 3   ENCOUNTER          47701 non-null  object 
 4   CODE               47701 non-null  int64  
 5   DESCRIPTION        47701 non-null  object 
 6   BASE_COST          47701 non-null  int64  
 7   REASONCODE         10756 non-null  float64
 8   REASONDESCRIPTION  10756 non-null  object 
dtypes: float64(1), int64(2), object(6)
memory usage: 3.3+ MB
```

In [8]:
```python
procedures['ENCOUNTER'].value_counts()
```

Out[8]:

```
66b2ab44-a2cc-8053-8f4e-c5be57e50cc4    186
e995e5e9-5130-abad-9247-6b83858fe3f5    180
534c59b6-2b18-28b7-a276-97de5576738e    168
eb6f44a2-c321-fa31-8d31-171794271e11    142
353167e8-5e65-4fd8-e0a8-acce8d2812bd    132
                                       ...
97800f93-f2c6-7e69-01d7-2f9d04491829      1
655d12ee-df51-1c2f-1c33-02a6d13dab68      1
e5497644-0c95-d66c-a5f3-9eff6a9d311b      1
069c4311-e5d2-fb6e-05fe-b872f86e5dea      1
32c84703-2481-49cd-d571-3899d5820253      1
Name: ENCOUNTER, Length: 14670, dtype: int64
```

In [9]:

```
procedures[procedures['ENCOUNTER']=='66b2ab44-a2cc-8053-8f4e-c5be57e50cc4']
```

Out[9]:

	START	STOP	PATIENT	ENCOUNTER	CODE	DESCRIPTION	BASE_
28161	2017-07-11T14:22:57Z	2017-07-11T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
28164	2017-07-12T14:22:57Z	2017-07-12T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
28193	2017-07-13T14:22:57Z	2017-07-13T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
28201	2017-07-14T14:22:57Z	2017-07-14T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
28223	2017-07-15T14:22:57Z	2017-07-15T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
...	...	...	...	...	...	...	
30404	2018-01-08T14:22:57Z	2018-01-08T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
30416	2018-01-09T14:22:57Z	2018-01-09T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
30428	2018-01-10T14:22:57Z	2018-01-10T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
30449	2018-01-11T14:22:57Z	2018-01-11T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	385763009	Hospice care (regime/therapy)	
30463	2018-01-12T14:22:57Z	2018-01-12T14:37:57Z	5936f828-81d9-1a90-03b1-cfe49183dba8	66b2ab44-a2cc-8053-8f4e-c5be57e50cc4	58000006	Patient discharge (procedure)	

186 rows × 9 columns

Removing Duplicates

**NOTE : while working on ETL exercise after doing more exploratory analysis on data I feel like there are not any duplicates and one encounter can have multiple procedures performed on the patient. But in this exercise I remove duplicates based on encounter id.

Lots of procedures data have same encounter id diffrent start and stop date but same time.

For simplicity and keeping only procedures file in scope, I will select the first record for a given encounter ID to remove duplicates for this exercise.

Before doing any operation on data I am removing duplicates so that other operations are comparatively faster.

In [10]:
```
procedures.drop_duplicates(subset=['ENCOUNTER'],inplace=True) # Removing duplicate
```

Procedure Duration in Seconds

In [11]:
```
procedures['START']=pd.to_datetime(procedures['START']) # changing datatype to date
procedures['STOP']=pd.to_datetime(procedures['STOP'])
```

In [12]:
```
procedures.info() # checking counts and datatypes
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14670 entries, 0 to 47696
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   START              14670 non-null  datetime64[ns, UTC]
 1   STOP               14670 non-null  datetime64[ns, UTC]
 2   PATIENT            14670 non-null  object
 3   ENCOUNTER          14670 non-null  object
 4   CODE               14670 non-null  int64
 5   DESCRIPTION        14670 non-null  object
 6   BASE_COST          14670 non-null  int64
 7   REASONCODE         4632 non-null   float64
 8   REASONDESCRIPTION  4632 non-null   object
dtypes: datetime64[ns, UTC](2), float64(1), int64(2), object(4)
memory usage: 1.1+ MB
```

In [13]:
```
procedures['DURATION']=procedures['STOP']-procedures['START'] # calculating time di
```

In [14]:
```
procedures['DURATION']=procedures['DURATION'].apply(lambda x : x.seconds) # convert
```

In [15]:
```
procedures.head()
```

Out[15]:

	START	STOP	PATIENT	ENCOUNTER	CODE	DESCRIPTION	BASE_CC
0	2011-01-02 09:26:36+00:00	2011-01-02 12:58:36+00:00	3de74169-7f67-9304-91d4-757e0f3a14d2	32c84703-2481-49cd-d571-3899d5820253	265764009	Renal dialysis (procedure)	9
1	2011-01-03 05:44:39+00:00	2011-01-03 06:01:42+00:00	d9ec2e44-32e9-9148-179a-1653348cc4e2	c98059da-320a-c0a6-fced-c8815f3e3f39	76601001	Intramuscular injection	2
2	2011-01-04 14:49:55+00:00	2011-01-04 15:04:55+00:00	d856d6e6-4c98-e7a2-129b-44076c63d008	2cfd4ddd-ad13-fe1e-528b-15051cea2ec3	703423002	Combined chemotherapy and radiation therapy (p...	11
3	2011-01-05 04:02:09+00:00	2011-01-05 04:17:09+00:00	bc9d59c3-0a30-6e3b-f47d-022e4f03c8de	17966936-0878-f4db-128b-a43ae10d0878	173160006	Diagnostic fiberoptic bronchoscopy (procedure)	9
4	2011-01-05 12:58:36+00:00	2011-01-05 16:42:36+00:00	3de74169-7f67-9304-91d4-757e0f3a14d2	9de5f0b0-4ba4-ce6f-45fb-b55c202f31a5	265764009	Renal dialysis (procedure)	1

Extracting Columns

Generating new columns YEAR, MONTH, WEEK, DAY from START Timestamp

In [16]:

```
procedures['YEAR']=procedures['START'].apply(lambda x : x.year)
procedures['MONTH']=procedures['START'].apply(lambda x : x.month_name())
procedures['WEEK']=procedures['START'].apply(lambda x : x.isocalendar().week)
procedures['DAY']=procedures['START'].apply(lambda x : x.day_name())
```

In [17]:

```
procedures.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14670 entries, 0 to 47696
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   START              14670 non-null  datetime64[ns, UTC]
 1   STOP               14670 non-null  datetime64[ns, UTC]
 2   PATIENT            14670 non-null  object
 3   ENCOUNTER          14670 non-null  object
 4   CODE               14670 non-null  int64
 5   DESCRIPTION        14670 non-null  object
 6   BASE_COST          14670 non-null  int64
 7   REASONCODE         4632 non-null   float64
 8   REASONDESCRIPTION  4632 non-null   object
 9   DURATION           14670 non-null  int64
 10  YEAR               14670 non-null  int64
 11  MONTH              14670 non-null  object
 12  WEEK               14670 non-null  int64
 13  DAY                14670 non-null  object
dtypes: datetime64[ns, UTC](2), float64(1), int64(5), object(6)
memory usage: 1.7+ MB
```

Filtering procedures

Filtering data and based on BASE_COST and storing in file "ajit-malik-assessment/output/task_1_output.csv

In [18]:
```
procedures_costly=procedures[procedures['BASE_COST']>=30000] # Filtering for data w
```

In [19]:
```
procedures_costly.head()
```

Out[19]:

	START	STOP	PATIENT	ENCOUNTER	CODE	DESCRIPTION	BASE
287	2011-03-25 18:33:11+00:00	2011-03-25 18:48:11+00:00	624e6dad-69f7-1f89-ca0b-5f77d4415ada	0fc73f3a-ba35-a904-a964-b1b4b0612c1c	180325003	Electrical cardioversion	
587	2011-06-13 15:44:02+00:00	2011-06-13 15:59:02+00:00	49bc1d54-ed70-7ec5-02cb-76c178292427	6073cd4b-f273-2843-6862-48233312c1f0	180325003	Electrical cardioversion	
750	2011-07-19 09:04:16+00:00	2011-07-19 09:19:16+00:00	ded9d0c9-ae3c-a52a-2567-234bcf5a2294	415e557b-80ca-e2c4-4834-c176c6af25f1	180325003	Electrical cardioversion	
866	2011-08-22 15:44:02+00:00	2011-08-22 15:59:02+00:00	49bc1d54-ed70-7ec5-02cb-76c178292427	ed0dc71b-8a65-ae90-2767-23ba097ed3dc	180325003	Electrical cardioversion	
1121	2011-12-06 09:04:16+00:00	2011-12-06 09:19:16+00:00	ded9d0c9-ae3c-a52a-2567-234bcf5a2294	a6a7f084-ac18-5be0-ba0f-68ee4b1e15e4	180325003	Electrical cardioversion	

In [20]:
```
procedures_costly.to_csv(r'/home/jovyan/output/costly_procedures.csv', index=False)
```

Costly procedures records saved to a file.

---

2. ETL Pipeline

Data stored in Postgres schemas STAGING and CURATED (FACTS and DIM tables)

- Data is loaded to staging schema using airflow dag "staging_layer_load".
- After this, Using Postgres I load data into CURATED layer fact and dim tables orchestrated using airflow dag "curated_layer_load"

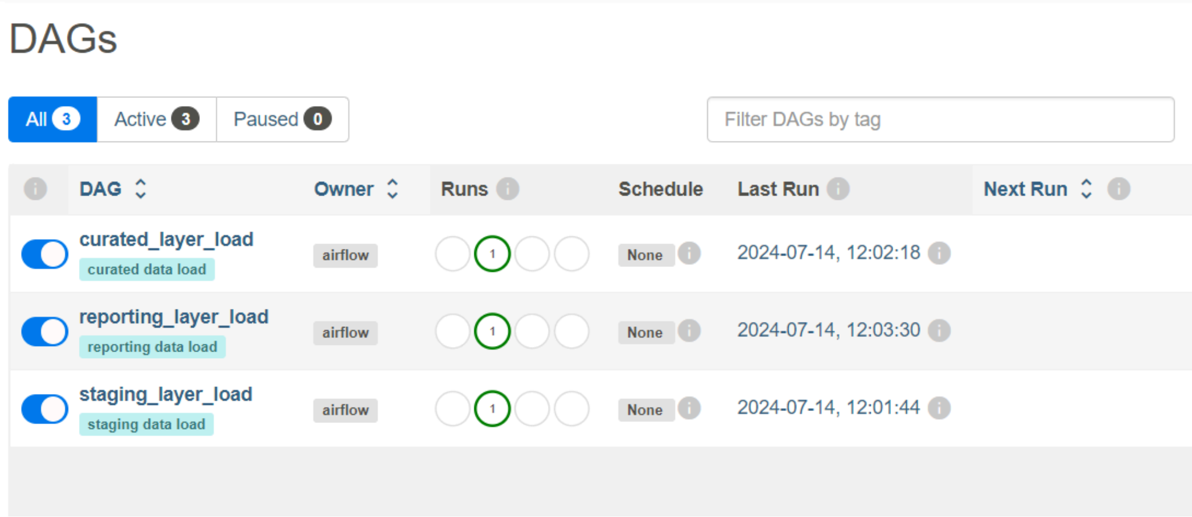
Assumption :

- Some encounters are withour any procedures performed so these are also loaded in fact table for analysis
- Relationships are not defined physically in database.

Below is the ER diagram of curated layer and dags

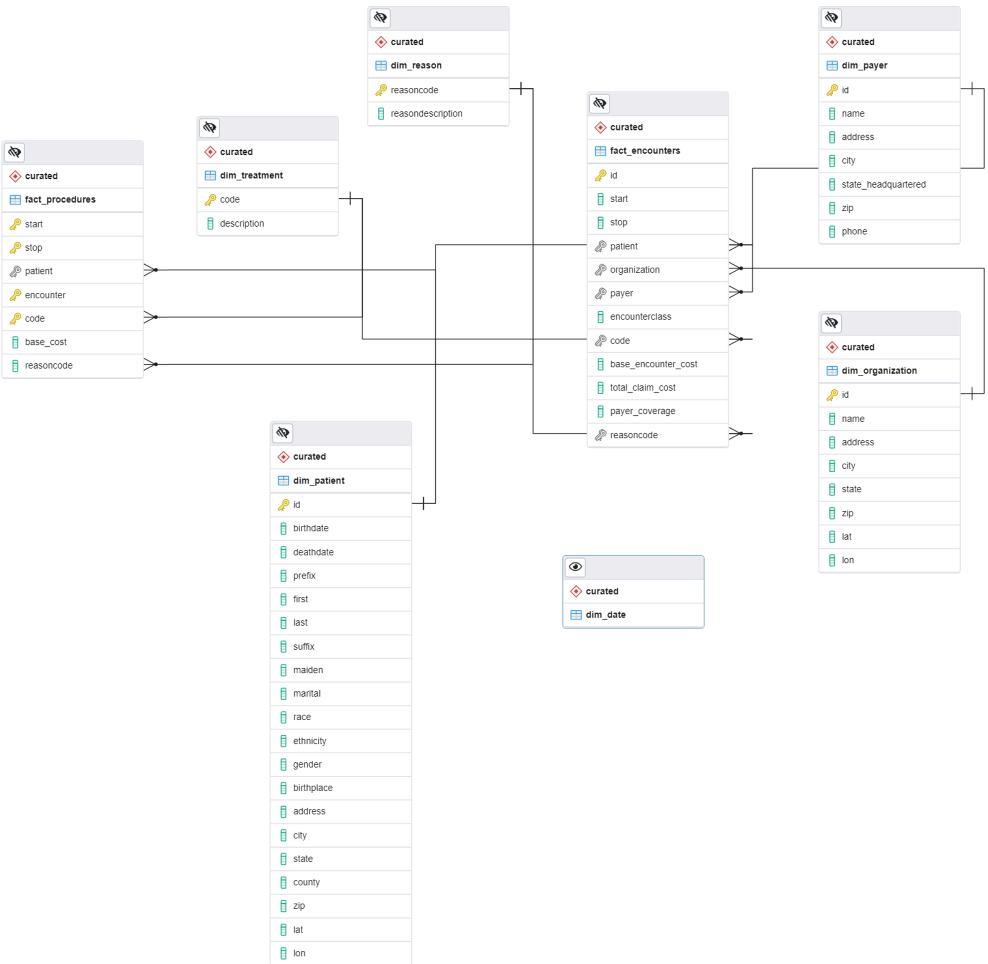
In [21]:
```
Image(filename=r'/home/jovyan/screenshots/airflow.png')
```

Out[21]:



In [22]:
```
Image(filename=r'/home/jovyan/screenshots/ER_dia.png')
```

Out[22]:



Rank the payer by cost paid

"payer_coverage" column is used to calculate cost paid by payer, considering company will pay only for covered amount not beyond that.

In [23]:
```
final_df=pd.read_sql("""
select p.name,sum(payer_coverage)::integer "cost payed by company" from curated.fac
left outer join curated.dim_payer p on fe.payer=p.id
group by 1
order by 2 desc;
"""
,con=get_engine())

final_df.to_csv(r'/home/jovyan/output/payer_by_cost.csv', index=False)  # store as

final_df
```

Out[23]:

	name	cost payed by company
0	Medicare	19215691
1	Medicaid	8417974
2	Blue Cross Blue Shield	2074496
3	Dual Eligible	1380706
4	UnitedHealthcare	3937
5	Humana	1954
6	Aetna	1780
7	Cigna Health	968
8	Anthem	0
9	NO_INSURANCE	0

Top 5 highest costing patients (from payer prespective)

"payer_coverage" column is used to calculate cost paid by payer for a patient.

In [24]:
```
final_df=pd.read_sql("""
select pat.id,pat.first||' '||pat.last "patient name",p.name "company name",sum(pay
left outer join curated.dim_payer p on fe.payer=p.id
left outer join curated.dim_patient pat on fe.patient=pat.id
group by 1,2,3
order by 4 desc limit 5;
"""
,con=get_engine())

final_df.to_csv(r'/home/jovyan/output/payer_by_cost_company.csv', index=False)  # s

final_df
```

Out[24]:

	id	patient name	company name	cost payed by company
0	ff331e5c-ab16-e218-f39a-63e11de1ed75	Eugene421 Abernathy524	Medicare	845233.69
1	5e055638-0dad-dfd5-005d-1e74b6fd29ac	Shani239 Parisian75	Medicare	628529.62
2	c83e8f1b-8f35-5855-0d38-6ea6509ec619	Ferdinand55 Goodwin327	Medicare	626448.92
3	5427845a-82ab-b6c9-e70b-aeb672ddd5d7	Arica110 McLaughlin530	Medicare	560859.04
4	49bc1d54-ed70-7ec5-02cb-76c178292427	Kurtis994 Bartell116	Medicare	559834.49

Top 5 highest costing patients (from Patient prespective)

"total_clain-cost" - "payer_coverage" column is used to calculate cost paid by patient from own pocket.

In [25]:

```
final_df=pd.read_sql("""
with base as (
select pat.id,pat.first||' '||pat.last "patient name",p.name,sum(total_claim_cost)
left outer join curated.dim_payer p on fe.payer=p.id
left outer join curated.dim_patient pat on fe.patient=pat.id
group by 1,2,3
)
select id,"patient name",sum(cost_payed_total) - sum(cost_payed_company) "cost paye
from base
group by 1,2
order by 3 desc limit 5
"""
,con=get_engine())

final_df.to_csv(r'/home/jovyan/output/payer_by_cost_self.csv', index=False)  # stor

final_df
```

Out[25]:

	id	patient name	cost payed by self
0	3f523789-55f3-bb31-2757-4803ca6a9c2a	Gail741 Glover433	9932262.99
1	ff331e5c-ab16-e218-f39a-63e11de1ed75	Eugene421 Abernathy524	5511431.97
2	a733bbc1-cbdf-992f-f1b7-bd230028fc4f	Columbus656 Wolf938	3014121.02
3	2aa3ac2a-88c6-3253-547d-6d8ed69790a3	Williams176 Harris789	2699823.10
4	1712d26d-822d-1e3a-2267-0a9dba31d7c8	Kimberly627 Collier206	2437415.31

top 5 most expensive procedures on daily basis

first I select top 5 costly procedures daily based on "base_cost" (there can be more than 5 procedures if cost is same), then I count the unique days those procedures were performed and sort them on mostly ocurred.

this is not perfect because costly procedures can be less frequent, and also if some treatment is happening frequently will affect the results.

In [26]:
```
final_df=pd.read_sql("""
with daily_top_5 as (
select t.description,DATE(start) "date" ,base_cost, rank() over (partition by DATE(
left outer join curated.dim_treatment t on fp.code=t.code
)
select description,count(distinct "date") freq
from daily_top_5
where rnk <=5
group by 1 order by 2 desc limit 5;
"""
,con=get_engine())

final_df.to_csv(r'/home/jovyan/output/top_5_procedures.csv', index=False)  # store

final_df
```

Out[26]:

	description	freq
0	Hospice care (regime/therapy)	1911
1	Assessment of health and social care needs (pr...	1833
2	Renal dialysis (procedure)	1782
3	Depression screening (procedure)	1685
4	Depression screening using Patient Health Ques...	1626

---

3. Datamarts

For our 2 clients we can make diffrent fact tables in "reporting" schema, containing only data from that Client.

These facts can be joined with dim tables to get reporting done !

- "fact_procedures_united" and "fact_encounters_united" tables for UNITED HEALTCARE.
- "fact_procedures_humana" and "fact_encounters_humana" tables for HUMANA.
- orchestrated using airflow dag "reporting_layer_load"

If these tables are access by client also we should put them in diffrent schemas for better access controls.

In [27]:
```
final_df=pd.read_sql("""
select 'fact_procedures_united' table_name, count(*) from reporting.fact_procedures
select 'fact_encounters_united' table_name, count(*) from reporting.fact_encounters
select 'fact_procedures_humana' table_name, count(*) from reporting.fact_procedures
select 'fact_encounters_humana' table_name, count(*) from reporting.fact_encounters
"""
,con=get_engine())

final_df
```

Out[27]:

	table_name	count
0	fact_encounters_humana	1084
1	fact_encounters_united	900
2	fact_procedures_humana	1999
3	fact_procedures_united	1262

Thank You

In [ ]: