Instruções do script:

1. Crie um banco de dados postgresql chamado clicksign, crie a role com nome clicksign e password clicksign 2. Garanta que os módulos que estão nas importações estejam presentes no env do seu python, caso não esteja instale com pip (veja no Google) 3. Estou usando o server 10.0.10.178 para executar o airflow e também o postgres altere para o seu servidor que contém estes instalados. 4. A variável local_csv aponta para o arquivo a ser carregado, mas poderia ser usado por variável de ambiente ou mesmo passado por parâmetro. 4.1. Ajuste para contemplar o caminho onde estará seu arquivo de dados.

Instruções do airflow:

1. A DAG está com configurações que uso em alguns projetos e pode ser configurada para alertar falhas no Teams, mas deixei apenas comentado. 2. A DAG contèm o env de onde está o chamador do Ariflow e precisa ser alterada para chamar do env de onde está configurado o seu python 3. A DAG aponta o arquivo que IngestaoAdult.py que está em um diretório de app que você precisa alterar para que o python chame no local que estará a aplicação IngestaoAdult.py.

Observações:

1. Devido ao meu tempo fiz o teste hoje 19/09/2022 e tive que instalar o postgresql e configurar no ambiente linux com acessos e firewall. 2. Tem várias formas de fazer esse teste com pyspark, validações de registro MD5, tratativas, entre várias outras possibilidades. 3. Como estou trabalhando e fazendo o teste em paralelo, procurei ser o mais rápido possível para entregar e fazer como uma primeira entrega. 4. Alguns pontos não compreendi e poderia ter perguntado, mas realmente o meu tempo tá limitadíssimo. Creio que é possível ver o conhecimento, embora acho bem pouco dado ao universo de coisas que posso fazer, mas espero ter valido como teste do meu conhecimento.

In [ ]:

```
import time
import pandas as pd
from sqlalchemy import create_engine
import psycopg2
```

In [ ]:

```
#Verifica se a tabela existe para appendar ou criar automaticamente
def table_exists(conn,table_str):
    exists = False
    try:
        result = conn.execute("select exists(select * from information_schema.tables where
        exists = result.fetchone()[0]
        #print(exists)
        result.close()
    except psycopg2.Error as e:
        print(e)
    return exists

#Remove espaço entre as strings
def trim_all_columns(df):
    trim_strings = lambda x: x.strip() if isinstance(x, str) else x
    return df.applymap(trim_strings)
```

In [ ]:

```
#String de conexão para esse teste usei linux CentOS 7, o usuario, senha e db são dos mesmo
#Precisei instalar o Postgresl PostgreSQL 9.2.24 on x86_64-redhat-linux-gnu,
#compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-44), 64-bit
conn_string = 'postgresql://clicksign:clicksign@10.0.10.178/clicksign'
conn  = create_engine(conn_string, echo=False)
conn = conn.execution_options(autocommit=True)
```

In [ ]:

```


local_csv = 'C:\\Users\\WalterFerreiradosSan\\Documents\\GitHub\\python-dev-test\\data\\Adu
#Como ainda não conheço o negócio a fundo da ClickSign trouxe todas as colunas afinal, com
#É possível vários insigths.
#Inclui o header conforme documentação
header_list = ["age", \
               "workclass", \
               "fnlwgt", \
               "education", \
               "education-num", \
               "marital-status", \
               "occupation", \
               "relationship", \
               "race", \
               "sex", \
               "capital-gain ", \
               "capital-loss", \
               "hours-per-week", \
               "native-country", \
               "class"
              ]
#grava no dataframe incluindo o header e removendo a primeira linha e garantindo somente in
#Troca o caractere inválido por nulo.
df = pd.read_csv(local_csv,header=None, names=header_list,  nrows=1630,skiprows=1, na_value
#Chama função para remover espaços vazios entre as strings
df = trim_all_columns(df)
df
```

In [ ]:

```
#observem que quem vai determinar a não duplicidade da informação será o código de origem,
#chave fica difícil fazer o incremental/diferencial de dados.
#também poderiamos add um hash md5 para validar o que é dado novo...

start_time = time.time()
if(table_exists(conn,'adult')==False):
    df.to_sql('adult', con=conn, if_exists='replace', index=False, method="multi")
else:
    df.to_sql('adult', con=conn, if_exists='append', index=False, method="multi")
print("to_sql duration: {} seconds".format(time.time() - start_time))
```

In [ ]:

```

```