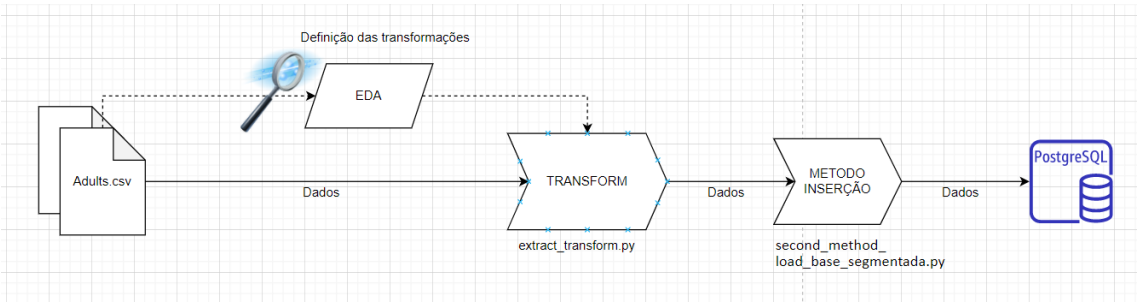
A Proposta da Solução.

A resolução final deste projeto possui quatro arquivos principais, pensados de modo a ilustrar uma parte do horizonte de conhecimentos que compõe meu background profissional:

- EDA.ipynb
    - Jupyter Notebook utilizado para realizar a análise exploratória de dados, elaboração dos processos de transformação dos atributos da base e desenvolvimento de dois modelos de machine learning
- extract_transform.py;
    - Arquivo responsável por aplicar as transformações definidas através do processo de EDA na base de dados de forma produtizada
- first_method_full_load.py;
    - Arquivo que um método de inserção da base de dados tratada após as transformações feitas por extract_transform.py em um banco de dados Postgresql. A ideia central deste método é carregar todo o conjunto de dados na memória, segmentar a base e inserir cada um dos lotes na tabela final a cada X segundos. Este método considera que a base original dos dados transformados é estática.
- second_method_load_base_segmentada.py.
    - Arquivo que um método de inserção da base de dados tratada após as transformações feitas por extract_transform.py em um banco de dados Postgresql. A ideia central deste método é mimetizar um tráfego parcial (em lote) dos dados da base transformada, simulando uma área de staging. Tal método considera que a base de dados transformada pode ser dinâmica de modo a aceitar que novos registros possam ser adicionados (‘appendados’) na área de staging. Este foi o arquivo utilizado como método final da solução do problema proposto.



A proposta de agendamento é cumprida através do encapsulamento do projeto em um Docker, através do dockerfile.yaml, na raíz do projeto. De tal modo, o projeto pode ser executado através de um agendamento do tipo CronJob em um cluster kubernetes ou executado diretamente através do Docker.

O Desenvolvimento do Projeto.

1 – Análise Exploratória de Dados (EDA)

Assim que uma base de dados é recebida para ser trabalhada, a primeira coisa que se deve fazer é analisá-la de forma ampla de modo a extrair informações que descrevam a estrutura e qualidade destes dados.

De tal modo, pode se verificar que a base não possuía nenhum dado faltante para cada um dos atributos.

2 Attributes check

In [5]:
```
df.info() ## nenhuma linha vazia
```

executed in 99ms, finished 20:25:45 2022-07-28

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48842 entries, 0 to 48841
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             48842 non-null  object
 1   workclass       48842 non-null  object
 2   fnlwgt          48842 non-null  object
 3   education       48842 non-null  object
 4   education-num   48842 non-null  int64 
 5   marital-status  48842 non-null  object
 6   occupation      48842 non-null  object
 7   relationship    48842 non-null  object
 8   race            48842 non-null  object
 9   sex             48842 non-null  object
 10  capital-gain    48842 non-null  object
 11  capital-loss    48842 non-null  int64 
 12  hours-per-week  48842 non-null  object
 13  native-country  48842 non-null  object
 14  class           48842 non-null  object
dtypes: int64(2), object(13)
memory usage: 5.6+ MB
```

A partir daí, cada um dos atributos foi estudado individualmente para que pudesse ser entendido e definido quais tratamentos seriam necessários realizar para normalizar efetivamente cada uma das séries de dados. Vale ressaltar que tal estratégia pode ser adotada devido o tamanho e estrutura do dataset, que possibilitaram a análise detalhada e individual de forma rápida. Datasets com dezenas, centenas (ou até mesmo milhares) de colunas poderiam ter sido abordados de forma diferente e mais genérica, por exemplo. Enfim, cada caso é um caso.

Para mais detalhes das transformações realizadas no processo de EDA, visite o Jupyter Notebook.

2.1 age

- Age of an individual.
- Continuous.

In [6]:
```
df['age'].unique() #mais de um tipo de dado: STR e int; nenhuma idade negativa.
```
executed in 28ms, finished 20:25:45 2022-07-28

Out[6]:
```
array(['39', '50', '38', '53', '28', '37', '49', '52', '31', '42', '30',
       '23', '32', '40', '34', '25', '43', '54', '35', '59', '56', '19',
       '20', '45', '22', '48', '21', '24', '57', '44', '41', '29', '18',
       '47', '46', '36', '79', '27', '67', '33', '76', '17', '55', '61',
       '70', '64', '71', '68', '66', '51', '58', '26', '60', '90', '75',
       '65', '77', '62', '63', '80', '72', '74', '69', '73', '81', '78',
       '88', '82', '83', '84', '85', 'B', '86', '87', 'D', 25, 38, 28, 44,
       18, 34, 29, 63, 24, 55, 65, 36, 26, 58, 48, 43, 20, 37, 40, 72, 45,
       22, 23, 54, 32, 46, 56, 17, 39, 52, 21, 42, 33, 30, 47, 41, 19, 69,
       50, 31, 59, 49, 51, 27, 57, 61, 64, 79, 73, 53, 77, 80, 62, 35, 68,
       66, 75, 60, 67, 71, 70, 90, 81, 74, 78, 82, 83, 85, 76, 84, 89, 88,
       87], dtype=object)
```

In [7]:
```
df['age']=df['age'].astype(str) #ETL
```
executed in 25ms, finished 20:25:45 2022-07-28

In [8]:
```
df['age_is_numeric']=df['age'].apply(lambda x : x.isnumeric() ) #ETL
```
executed in 40ms, finished 20:25:45 2022-07-28

In [9]:
```
df[df['age_is_numeric']==False] #Apenas duas linhas onde a iddade não é numérica
```
executed in 175ms, finished 20:25:45 2022-07-28

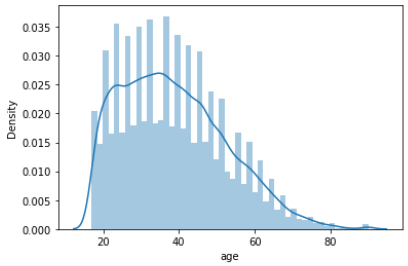
Out[9]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	class	age_is
22197	B	Self-emp-not-inc	182771	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Asian-Pac-Islander	Male	0	0	48	South	>50K	
32540	D	State-gov	252208	HS-grad	9	Separated	Adm-clerical	Own-child	White	Female	0	0	40	United-States	<=50K	

In [10]:
```
sns.distplot(df[df['age_is_numeric']==True]['age'])
```
executed in 1.11s, finished 20:25:47 2022-07-28

```
C:\Anaconda\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be rem
oved in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or
`histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

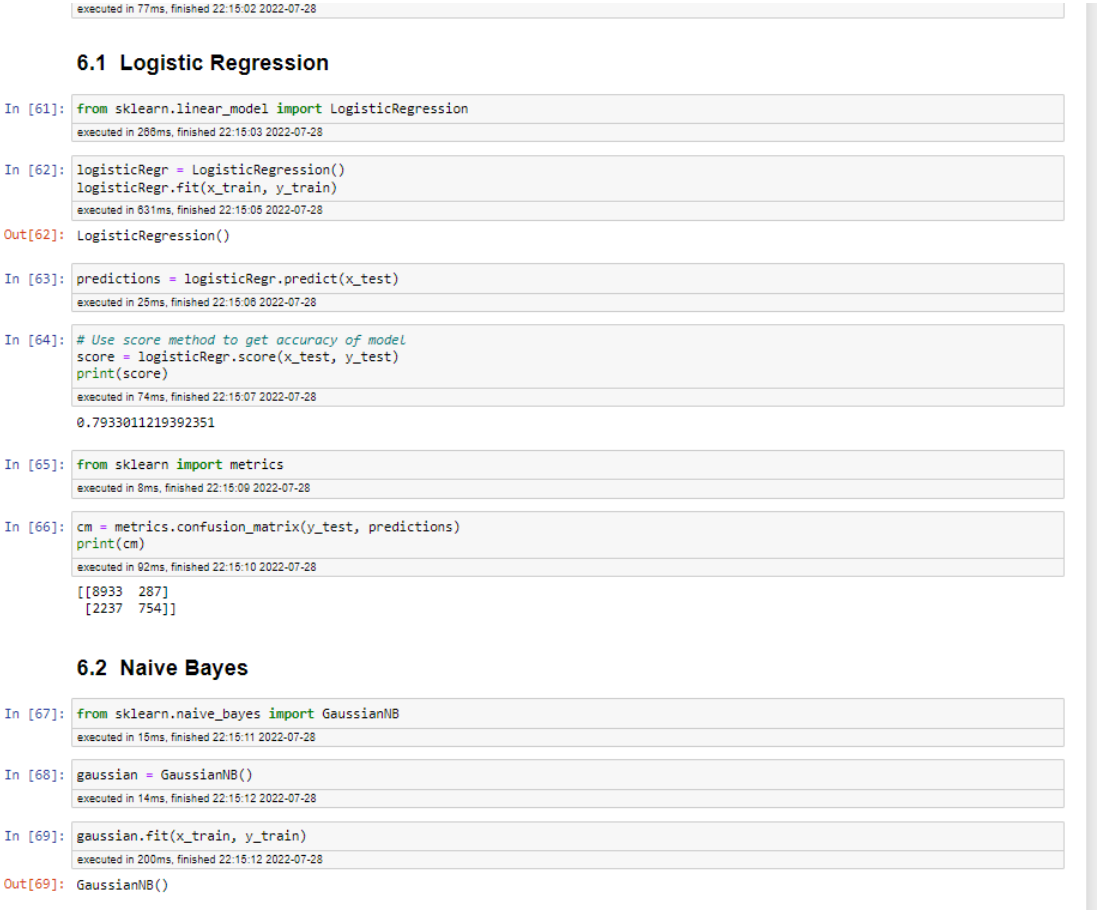
Out[10]:
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```



- Uma vez que a distribuição dos dados não está muito distorcida, e a quantidade de linhas do dataset a serem tratadas é pequena se comparada ao tamanho total da base, uma tratativa plausível é substituir tais valores não numéricos pela média das idades, ao invés de, por exemplo, deletar tais linhas, evitando assim a perda de dados das outras colunas.

Machine Learning

Para fins de demonstração de conhecimento, após o tratamento devido dos dados e análise exploratória, foram criados dois modelos de Machine Learning com o intuido de prever (Classificar) qual a faixa de faturamento anual de uma pessoa (<=50k;>50k)



executed in 77ms, finished 22:15:02 2022-07-28

6.1 Logistic Regression

In [61]:
```
from sklearn.linear_model import LogisticRegression
```
executed in 286ms, finished 22:15:03 2022-07-28

In [62]:
```
logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)
```
executed in 831ms, finished 22:15:05 2022-07-28

Out[62]:
```
LogisticRegression()
```

In [63]:
```
predictions = logisticRegr.predict(x_test)
```
executed in 25ms, finished 22:15:06 2022-07-28

In [64]:
```
# Use score method to get accuracy of model
score = logisticRegr.score(x_test, y_test)
print(score)
```
executed in 74ms, finished 22:15:07 2022-07-28

```
0.7933011219392351
```

In [65]:
```
from sklearn import metrics
```
executed in 8ms, finished 22:15:09 2022-07-28

In [66]:
```
cm = metrics.confusion_matrix(y_test, predictions)
print(cm)
```
executed in 92ms, finished 22:15:10 2022-07-28

```
[[8933  287]
 [2237  754]]
```

6.2 Naive Bayes

In [67]:
```
from sklearn.naive_bayes import GaussianNB
```
executed in 15ms, finished 22:15:11 2022-07-28

In [68]:
```
gaussian = GaussianNB()
```
executed in 14ms, finished 22:15:12 2022-07-28

In [69]:
```
gaussian.fit(x_train, y_train)
```
executed in 200ms, finished 22:15:12 2022-07-28

Out[69]:
```
GaussianNB()
```

Os modelos utilizados foram Regressão Logística e o Classificador de Naive Bayes, ambos performando com 79% de acurácia.

2 – A Produtização das Transformações

Após conseguir definir quais seriam as transformações necessárias no dataset, cada conjunto de transformações para cada atributo foi encapsulado em uma classe, de modo a refletir, organizar e limitar o escopo de execução do código responsável por executar a correção dos dados.

```python
class AttCapitalGain:
    def __init__(self):
        pass

    @staticmethod
    def corrigir(df):
        df['capital-gain'] = df['capital-gain'].astype(str)
        df['capital-gain'] = df['capital-gain'].apply(lambda x: x.replace(' ', ''))
        df['capital-gain_is_numeric'] = df['capital-gain'].apply(lambda x: x.isnumeric())
        media_fnlwgt = df[df['capital-gain_is_numeric'] == True]['capital-gain'].astype(float).mean()
        df['capital-gain'] = df['capital-gain'].apply(lambda x: float(x) if x.isnumeric() else media_fnlwgt)
        df.drop('capital-gain_is_numeric', axis=1, inplace=True)
        return df
```

Cada uma das Classes possui a responsabilidade única de tratar e tansformar cada um dos atributos da base de forma individual, considerando as particularidades dos dados em cada um dos casos. Esse modelo de arquitetura, embora implique em mais linhas de código, traz a possibilidade de atuar em um escopo reduzido, garantindo assim a qualidade dos processos de transformação, a facilitando a realização de testes unitários e de integração. Extrapolando o conceito, tal arquitetura, para além de tratar individualmente cada um dos atributos do um dataframe de um problema específico, pode ser utilizada para absorver dados de origens diversas e com diferentes modelagens, assegurando o escopo reduzido e organizado, auxiliando na manutenção de um código limpo e etc para um projeto mais complexo. Não obstante, tal arquitetura possui uma estrutura que facilita a implementação de versionamento do método de tratamento para os diferentes Atributos/Fontes caso seja necessário, por exemplo, conviver com mais de uma versão de modelo de dados transeuntes.

```python
print('Iniciando Transformação de Base')
df = DataFetcher().fetch()
df = AttAge().corrigir(df)
df = AttWorkClass().corrigir(df)
df = AttFnlWgt().corrigir(df)
df = AttEducation().corrigir(df)
df = AttMartialStatus().corrigir(df)
df = AttOccupation().corrigir(df)
df = AttRealtionship().corrigir(df)
df = AttSex().corrigir(df)
df = AttCapitalGain().corrigir(df)
df = AttHoursPerWeek().corrigir(df)
df = AttNativeCountry().corrigir(df)
df = AttClass().corrigir(df)

df.to_csv('outputs/base_tratada_para_inserir.csv', index=False)

print("Transformação Realizada com sucesso")
```

Uma vez que transformada, a base corrigida é salva no formato CSV para a execução do método de inserção. Vale ressaltar que a base não PRECISARIA ser salva em disco, uma vez que o processo de inserção na base de dados poderia ter sido feito na mesma execução do processo de limpeza. Contudo, A ideia por trás da segmentação dos passos é mimetizar processos de ETL mais reais, com mais de um passo e etc.

3 - Os Métodos de Inserção

First Method: Full Load

O Intuito deste método é demonstrar a resolução pragmática do problema: Inserir a cada X segundos Y linhas de da base de dados tratado em uma nova tabela no bando de dados, respeitando os datatypes do dataset. Sendo assim, a lógica por trás desse arquivo se baseia em dividir o dataset em vários slices de tamanho definido e inseri-los na base.

```
#
if __name__ == "__main__":
    df = pd.read_csv('outputs/base_tratada_para_inserir.csv')

    obj_insertor = InsercaoAdults()
    obj_insertor.run_insert(df, slice_len=1630, time_to_sleep=10)
```

Para isso, foi criado uma classe capaz de realizar a conexão e operação no banco de dados e abarcar a modelagem de dado com a tipificação das colunas do dataset.

```
class InsercaoAdults:

    def __init__(self):
        self.engine = create_engine(DB_CONN,
                                    connect_args={"application_name": "ETL_TEST"})
        self.dtypes = {
            'age': Numeric(),
            'workclass': VARCHAR(),
            'fnlwgt': Numeric(),
            'education': VARCHAR(),
            'education-num': Numeric(),
            'marital-status': VARCHAR(),
            'occupation': VARCHAR(),
            'relationship': VARCHAR(),
            'race': VARCHAR(),
            'sex': VARCHAR(),
            'capital-gain': Numeric(),
            'capital-loss': Numeric(),
            'hours-per-week': Numeric(),
            'native-country': VARCHAR(),
            'class': VARCHAR()
        }
```

A conexão com o banco é feita através do Engine Connection da biblioteca SQLAlchemy.

Este método foi desenvolvido apenas como modo de demonstração técnica, sendo assim não é executado na produtização da solução.

Second Method: Load Base Segmentada

A ideia central deste método é mimetizar um tráfego parcial (em lote) dos dados da base transformada, simulando uma área de staging. Tal método considera que a base de dados transformada pode ser dinâmica de modo a aceitar que novos registros possam ser adicionados (‘appendados’) na área de staging.

```
class InsercaoStagedAdults(InsercaoAdults, StageData):

    def __init__(self):   Arruda, Yesterday • first commit
        self.engine = create_engine(DB_CONN,
                                    connect_args={"application_name": "ETL_TEST"})
        self.dtypes = {
            'age': Numeric(),
            'workclass': VARCHAR(),
            'fnlwgt': Numeric(),
            'education': VARCHAR(),
            'education-num': Numeric(),
            'marital-status': VARCHAR(),
            'occupation': VARCHAR(),
            'relationship': VARCHAR(),
            'race': VARCHAR(),
            'sex': VARCHAR(),
            'capital-gain': Numeric()
```

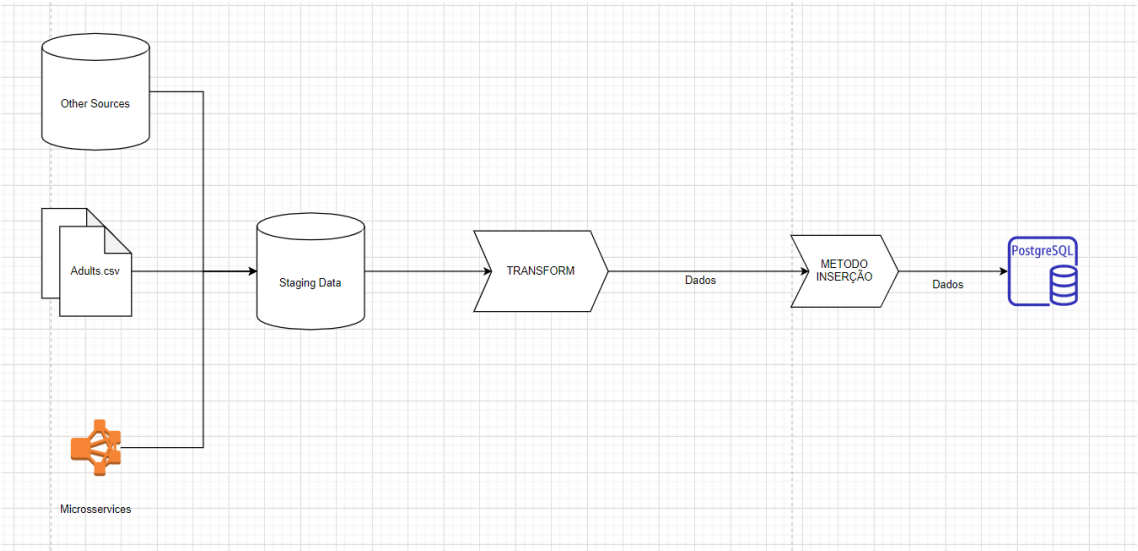
Assim como a classe InsercaoAdults, a classe principal deste método tem o intuito de inserir os dados em um banco de dados Postgres, de tal modo possui a engine de conexão com o baco e a modelagem dos datatypes dos atributos da tabela. Como modo de demonstração de conceitos, a classe InsercaoStageAdults herda as classes InsercaoAdults (onde realiza a sobrescrita do método Insert_in_database, de modo a se adequar ao seu contexto – conceito de polimorfismo).

A classe StageData foi construída com o de carregar memória um lote limitada de informações, não tendo necessidade de carregar a base inteira. Tal funcionamento mimetiza um tráfego de dados em uma tabela com dados dinâmicos, isto é, uma base que pode receber constantemente dados appendados, como por exemplo uma tabela em um datalake ou uma tabela em um ecossistema microserviço. Obviamente, o arquivo em texto funciona como um simulacro de uma tabela do banco de dados, a fim de demonstrar horizontes de conhecimento. Uma consulta realizada uma tabela Postgres de banco de dados real pode ser realizada do Engine Connection do SQLAlchemy ou conexão de Cursor do pacote Psycopg2, seguida da execução de query tabela.

Para ficar mais claro, a lógica acontece da seguinte maneira:

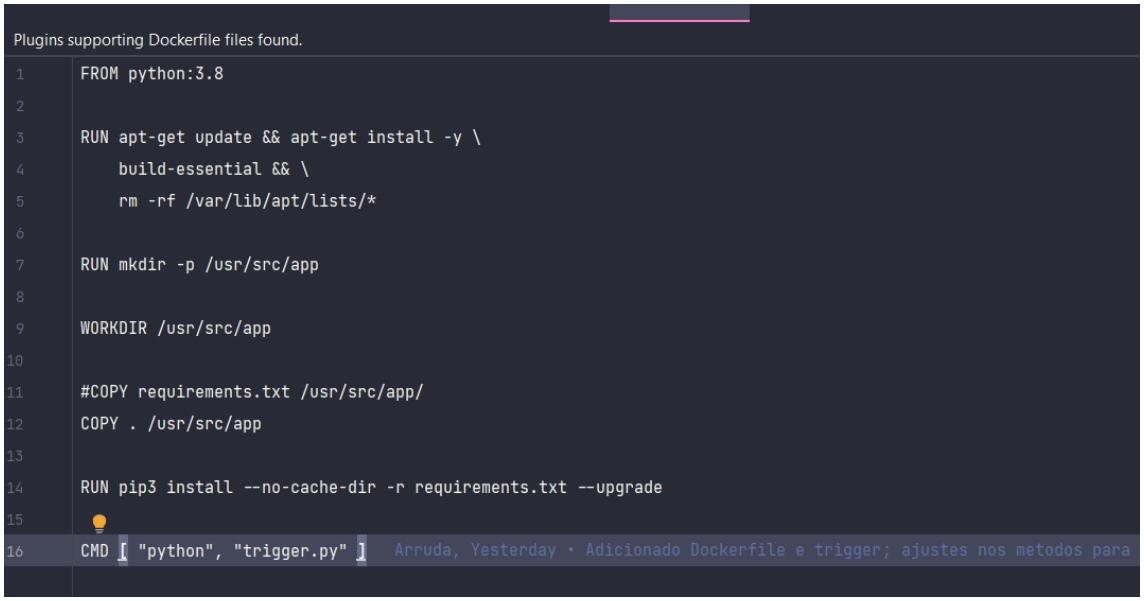
```
def run_insert(self, slice_len):   Arruda, Yesterday • first commit
    index_atualizado, slice_para_inserir = self.select_stage_data_to_insert(slice_len)
    self.insert_in_database(slice_para_inserir)
    self.atualizar_indice(index_atualizado)
    print(index_atualizado)
```

- Durante a execução, o código resgatar qual o índice da ultima informação inserida da base de dados final.
- Uma vez identificado qual a última linha inserida, o algoritmo irá entender e selecionar qual o devido lote de informações deve ser transacionado para o banco de dados final.
    - Tal seleção mimetiza a chegada do lote de informações por algum canal de entrada. Na vida real, o canal poderia ser uma mensagem em uma fila, um lote inserido em um serviço de streaming (AWS Kinesis), uma lote de mensagens de um Plugin CDC em um banco de dados, um evento s3 que dispara um lambda, ou qualquer outro trigger que dispare o ETL, por exemplo.
- Uma vez carregado em memória o lote de informações e retornado o índice final do conjunto, os dados são inseridos na tabela final no banco de dados Postgres e o índice é salvo na tabela de controle. Este passo é executado com um intervalor egula definido previamente (no caso do problema proposto, a cada 10 segundos).
    - Tal arquitetura permite permite o reprocessamento das informações faltantes na base de dados final caso a conexão com o banco falhe sem que haja a necessidade do reprocessamento da base por completo.
    - Esse tipo de controle permite que mais dados possam ser adicionados na tabela de origem sem que haja a interrupção do sistema.

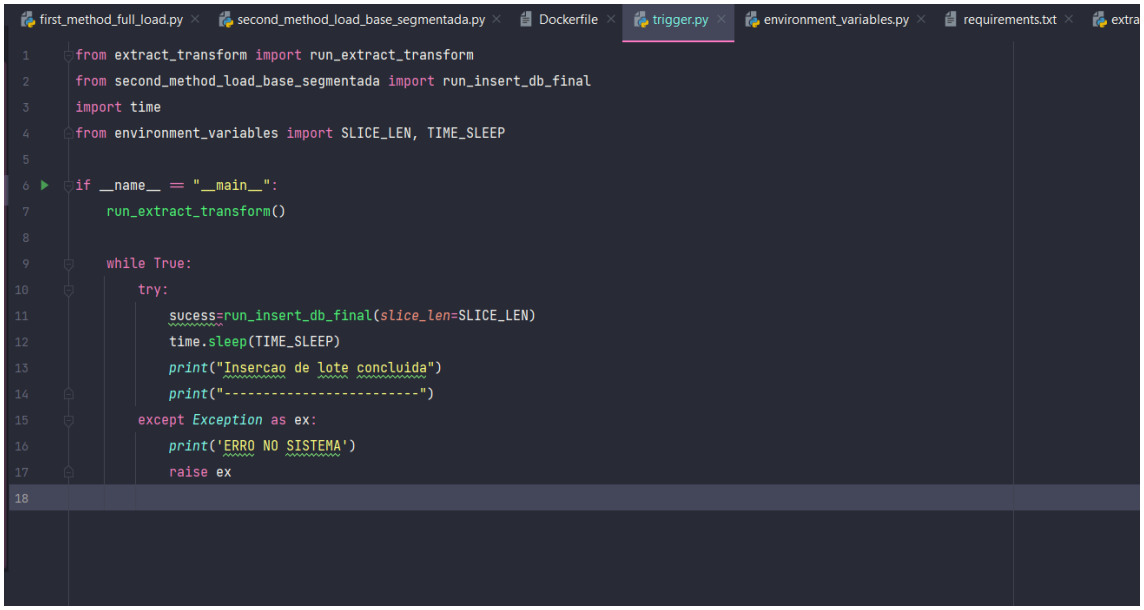


4 - A Produtização da proposta

Para atender a proposta e existir uma solução encapsulada e executável de forma agendada, o projeto foi estruturado com a utilização de um Dockerfile, responsável por encapsular e estruturar o projeto em um container de modo que se possa executá-lo através do Docker ou em um CronJob em um cluster Kubernetes.

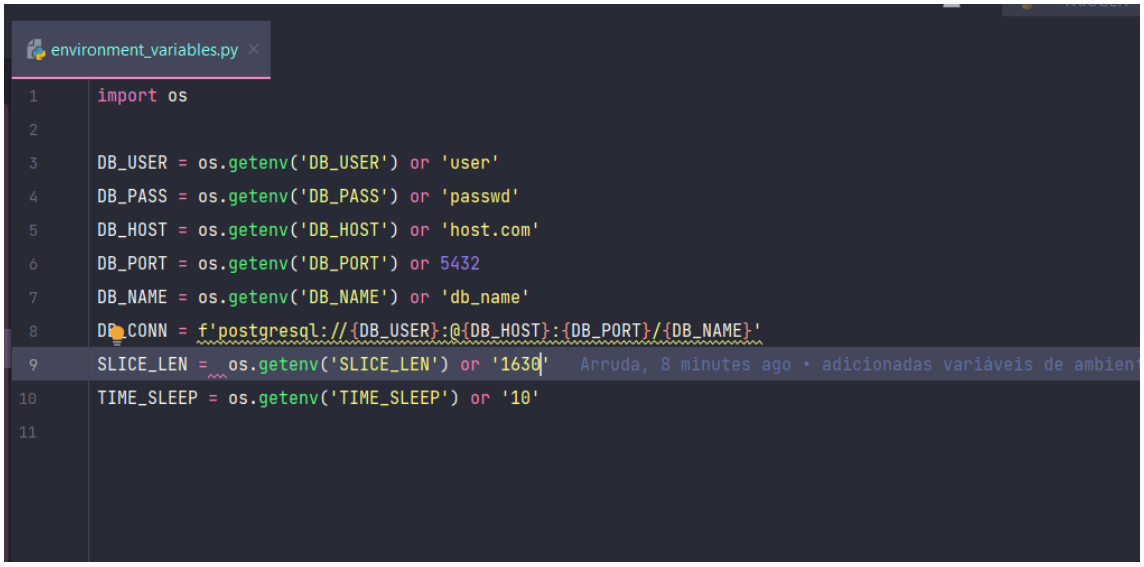


O Dockerfile estrutura as pastas do projeto, instala os requirements e executa o script contido no arquivo trigger.py



O arquivo trigger.py é responsável por executar o passo de ExtractTransform, descrito anteriormente, e executar em loop a função run_insert_db_final, função esta que redireciona o script para a execução do método de inserção do banco Second Method: Load Base Segmentada descrito anteriormente.

Naturalmente, para no setup de um projeto, são definidas variáveis de ambiente que condicionam os parâmetros de execução do sistema, tais como o tamanho dos lotes para inserção, o tempo de intervalo de cada execução e as credenciais de conexão com o banco. Contudo, para fins didáticos, cada uma das variáveis de ambiente, descritas e organizadas em environment_variables.py possuem valores padrão



Ademais, seria um prazer poder discutir tecnicamente o projeto e mais nuances sobre o Engenharia de Software orientada a Dados.

Obrigado pela atenção.