QUESTÃO 1
Você precisa integrar e manter atualizada uma base de comércio exterior de 143 países. Para isso, você utilizará uma API onde pode realizar 2500 consultas diárias. Os parâmetros de consulta são: país_origem, país_destino e ano, exemplo:
(BRA, ARG, 2019) >> importações e exportações feitas do Brasil para Argentina em 2019
(BRA, ARG, 2020) >> // em 2020
(BRA, AFG, 2019) >> imp/exp feitas do Brasil para Afeganistão em 2019
(BRA, AFG, 2020) >> // em 2020
(BRA, WLD, 2019) >> imp/exp feitas do Brasil para o Mundo em 2019
(BRA, WLD, 2020) >> // em 2020
(BRA, BRA, 2019) >> -> imp/exp feitas do Brasil para Brasil em 2019
(WLD, BRA 2019) >> -> imp/exp feitas do Mundo para Brasil em 2019

Os dados serão utilizados em um painel que responde, por exemplo, às seguintes questões:
Qual a quantidade de soja que o Brasil exportou para China em 2020? (BRA, CHN, 2020)

Qual a quantidade de soja que a China importou do Brasil em 2020? (CHN, BRA, 2020)

Qual a quantidade de soja que o Brasil exportou para o Mundo em 2020? (BRA, WLD, 2020)

Qual a quantidade de soja que o Mundo importou do Brasil em 2020?

Sua tarefa é criar:
1.1 - Uma estrutura de diretórios e nomenclatura de arquivos que permita armazenar as consultas de modo coerente, ex:
Haverão diretórios: BRA, ARG, AFG, ...
E arquivos: BRA.txt, ARG.txt, AFG.txt, ...
Com isso, o arquivo /BRA/ARG.txt representa a imp/exp feita do BRA para ARG
***Obs: A estrutura listada pode ser insuficiente/incorreta**

1.2 - Liste as etapas necessárias para integração de 3 anos (2019~2021) de comércio exterior, ex:
1 - Fazer request em todos paises, parceiros e anos
```
#itera sobre 143 países origem
for pais_origem in lista_paises:
 #itera sobre os 143 países destino
for pais_destino in lista_paises:
#itera sobre os anos
for ano in ['2019', '2020', '2021']:
#consulta
res = get(pais_origem, pais_destino, ano)
```
***Obs: Esse exemplo é insuficiente/incompleto**

2- Salvar arquivos
```
res.write(f'{pais_origem}/{pais_destino}.txt')
#ex: BRA/ARG.txt
```

***Obs: Embora a estrutura listada em (1) tenha sido preservada, não funcionará corretamente. Etapas adicionais (ou uma estrutura distinta) são necessárias.**
***Obs2: Note que os códigos são apenas abstrações. Não se faz necessário detalhamento: explicitar função get; garantir que objeto res possua método write e assim por diante.**

1.3 - Qual/Quais arquivos da sua base de dados respondem a questão: 'Qual a quantidade de soja que o Mundo importou do Brasil em 2020?'
A resposta dependerá da estrutura proposta em (1.1)

1.4 – Sua base de dados alimenta um painel que possui informações de comércio exterior entre países – e também mundo. As consultas de comércio exterior podem ser feitas com os parâmetros M (‘mês’) ou A(‘ano’). A consulta M representa um report dos meses disponíveis no ano até então. A consulta A representa o report final/completo das imp/exp em determinado ano. Considere que cada país escolhe uma data “aleatória” do primeiro trimestre do novo ano para realizar o report final/completo das suas exportações e importações anuais do ano anterior. Para o melhor entendimento, considere as seguintes consultas:
(BRA, CHN, 2024, ‘A’) >> retorna vazio, pois ainda não houve report final
(BRA, CHN, 2024, ‘M’) >> retorna o imp/exp nos meses disponíveis
(BRA, CHN, 2024, ‘A’) >> retornar report final/completo do ano de 2024
(BRA, CHN, 2024, ‘M’) >> retorna meses disponíveis (porém é incompleto)
Vale ressaltar que o report final/completo pode ser diferente do mensal mesmo após o término do ano. Isso porque o país pode escolher reportar mensalmente até o penúltimo trimestre do ano e depois fazer apenas o report anual. Isso faria com que os dataset de todos os meses disponíveis em 2024 seja distinto do report anual do mesmo ano.
Com isso em mente, liste as etapas necessárias para manter as bases atualizadas de modo recorrente: considere que estamos no dia 1 de fevereiro de 2025.

1.1 Uma alteração interessante a ser feita na estrutura fornecida é separar os arquivos de texto de acordo com o ano, então a estrutura final se parece com:
pasta-mãe/
|_Brasil/
|__Argentina2019.csv
|__Argentina2020.csv
|__China2019.csv
|__…
|_Argentina/
|__Brasil2020.csv
…

1.2 Para integrar 3 anos de dados utilizando a estrutura proposta acima, podemos utilizar o seguinte pseudocódigo:

```
def getDados(origem, destino, ano):
    url = "urlDaAPI.com/"+origem+"/"+destino+"/"+ano
    resposta = get(url)
```

```
    return resposta
def salvarDados(dados, origem, destino):
    converterParaCSV(dados)
    se /origem/destino/dados.csv não exite:
        cria arquivo /origem/destino/dados.csv
    senão:
        escreve no arquivo /origem/destino/dados.csv

paises = ["Brasil", "Argentina", "China", ...]
anos = [2019, 2020, 2021]

para origem em paises:
    para destino em paises:
        para ano em anos:
            dados = getDados(origem, destino, ano)
            salvardDados = (dados, origem, destino)
```

1.3 Para responder essa pergunta podemos consultar todos os arquivos dentro da pasta ‘paisOrigem/’ por todos os países destino no ano de 2020. Após isso, basta somar todos os valores das importações.

1.4 Podemos seguir os passos abaixo para manter a base atualizada:

1. Definir a frequência de atualização para duas vezes ao mês, assim países que atualizarem a base no início e no fim de cada mês não são prejudicados (por exemplo, todo dia 15 e todo dia 28)
2. Criar o script de atualização que
    a. Consulta a API buscando dados novos
    b. Caso existam, atualizar os arquivos existentes
3. Existem relatórios mensais e um relatório final anual, portanto:
    a. Relatórios mensais são atualizados à medida que surjam dados novos
    b. Após a publicação do relatório final, atualizar o arquivo/criar o arquivo de relatório anual

Um possível pseudocódigo para esta tarefa é:

```
para origem em paises:
     para destino em paises:
       se relatorio_anual estiver disponivel
           pegue os dados
           salve o arquivo
       caso contrário
           atualize os dados mensais
```

==========

QUESTÃO 2

Considere os arquivos públicos da Receita Federal disponíveis em: http://200.152.38.155/CNPJ/. A página é atualizada, na média, 1 vez ao mês: os nomes dos arquivos são fixos. Suponha que você precise automatizar o processo de baixar, empilhar e manter atualizado os arquivos de empresas (0~9). Considere que o site é instável, portanto, o arquivo baixado pode estar corrompido. Liste as etapas necessárias para que o dataset de empresas esteja sempre atualizado.
Ex:
1 - Baixar os arquivos de empresas, usando requests
2 - Extrair arquivos de empresas, usando zipfile
3 - Empilhar os arquivos, usando pandas.concat
4 - ...

Obs1: Não é necessário codar!
Obs2: O exemplo acima é insuficiente para garantir a criação do dataset e a atualização contínua do mesmo

Uma das formas de fazer com que o dataset de empresas esteja sempre atualizado é:

1. Num primeiro momento, criar um script que faça o download dos arquivos da receita (Empresa1.zip, …, Empresa9.zip). Dada a instabilidade do site é interessante programar um mecanismo de retry.

2. Após isso é possível verificar a integridade dos arquivos utilizando técnicas de checksum, como por exemplo o MD5 ou o SHA256. Caso algum arquivo esteja corrompido, basta fazer o download deste novamente.
3. Como os arquivos estão no formato .zip, a utilização da biblioteca zipfile ou outra similar irá auxiliar na extração dos documentos.
4. É preciso garantir que os arquivos extraídos contém os dados esperados, para isso define se alguns critérios de análise, como por exemplo: olhar para o tipo, comprimento, faixa de valores e outras características dos dados.
5. Para empilhar os arquivos, podemos utilizar a biblioteca pandas, que possui a função concat. Essa função irá concatenar os dataframes, e para garantir índices únicos utilizamos o parâmetro ignore_index = True.
6. Salvar o dataframe no formato adequado (CSV, Parquet, …)
7. Configurar um cron jobs para executar este script uma vez ao mês

Algumas etapas extras podem ser implementadas para garantir a funcionalidade e continuidade do projeto, como por exemplo criar um sistema que informa o progresso das atualizações, implementar uma rotina de backups dos dados, manter uma documentação do sistema e muitos outros.

==========

QUESTÃO 3
Escreva uma consulta SQL que retorne, para cada funcionário, o nome do departamento em que ele trabalhou pela primeira vez (baseado na data de início de trabalho), o nome do departamento onde ele está atualmente trabalhando, e a quantidade de departamentos diferentes em que ele já trabalhou. Use as tabelas employees, departments, e employee_department_history.

Estrutura das tabelas:

- employees
  - employee_id (PK)
  - employee_name
- departments
  - department_id (PK)
  - department_name
- employee_department_history
  - employee_id (FK)
  - department_id (FK)
  - start_date
  - end_date ('NULL' para o departamento atual)

Primeiro podemos criar o banco de dados com a estrutura passada no enunciado:

```
CREATE DATABASE IF NOT EXISTS teste;

USE teste;

CREATE TABLE IF NOT EXISTS employees (
    employee_id int AUTO_INCREMENT NOT NULL UNIQUE,
    employee_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (employee_id)
);

CREATE TABLE IF NOT EXISTS departments (
    department_id int   AUTO_INCREMENT NOT NULL UNIQUE,
    department_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (department_id)
);

CREATE TABLE IF NOT EXISTS employee_department_history (
    employee_id int  NOT NULL,
    department_id INTEGER   NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE,
    PRIMARY KEY (employee_id, department_id, start_date),
   FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
   FOREIGN KEY (department_id) REFERENCES departments(department_id)
);
```

Após isso, podemos povoar este banco com dados falsos, a fim de validar a requisição no futuro:

```
USE teste;
INSERT INTO employees (employee_name) VALUES
    ('Joao Pedro'),
    ('Luciana'),
```

```
    ('Sandro'),
    ('Thereza'),
    ('Fatima'),
    ('Antonia'),
    ('Daniel'),
    ('Gabriel'),
    ('Samara'),
    ('Aline');

INSERT INTO departments (department_name) VALUES
    ('Engenharia'),
    ('Marketing'),
    ('Financeiro'),
    ('Recursos Humanos'),
    ('Vendas');

INSERT INTO employee_department_history (employee_id, department_id,
start_date, end_date) VALUES
    (1, 1, '2023-01-01', '2024-05-01'),
    (1, 3, '2024-05-01', NULL),
    (2, 2, '2023-03-15', '2024-08-31'),
    (3, 1, '2022-11-01', NULL),
    (4, 4, '2023-07-01', '2024-06-30'),
    (5, 5, '2024-02-10', NULL),
    (6, 2, '2024-04-20', '2023-09-01'),
    (6, 3, '2023-09-01', '2024-03-31'),
    (7, 1, '2024-01-15', NULL),
    (8, 5, '2023-06-01', NULL),
    (9, 4, '2023-10-05', NULL),
    (10, 2, '2024-03-01', NULL);
```

Por fim podemos fazer a requisição em si:

```
WITH firstDepartment AS (
      SELECT
        edh.employee.id,
        d.department_name AS first_department
    FROM
        employee_department_history edh
    INNER JOIN departments d ON edh.department_id = d.department_id
    WHERE (edh.employee_id, edh.start_date) IN (
        SELECT
            employee_id,
            MIN(start_date)
```

```
        FROM
            employee_department_history
        GROUP BY employee_id
    )
),
actualDepartment AS (
    SELECT
        edh.employee_id,
        d.department_name AS actual_department
    FROM
        employee_department_history edh
    INNER JOIN departments d ON edh.department_id = d.department_id
    WHERE edh.end_date IS NULL
)
SELECT
    e.employee_name,
    fd.first_department
    ad.actual_department
    COUNT(DISTINCT edh.department_id) AS quantidade_departamentos
FROM
    employees e
INNER JOIN firstDepartment fd ON e.employee_id = fd.employee_id
INNER JOIN actualDepartment ad ON e.employee_id = ad.employee_id
INNER JOIN employee_department_history edh ON e.employee_id =
edh.employee_id
GROUP BY e.employee_name, fd.first_department, ad.actual_department;
```

==========

QUESTÃO 4

O arquivo exemplo1.parquet possui 7GB. O tempo de leitura e retorno de 1 linha específica são 5s - utilizando processamento paralelo (pyspark) em cluster. Contudo, você precisa criar uma API com recursos bem mais modestos, isto é, menor poder de processamento - mantendo o tempo de consulta rápido. Liste as alterações e procedimentos necessários para realizar isso.

Obs: se necessário, considere que o arquivo exemplo1.parquet atualize 1 vez ao mês.

No primeiro momento as atenções devem ser voltadas ao arquivo exemplo1.parquet. Levando em consideração que este arquivo é atualizado uma vez ao mês podemos dividi-lo em porções menores de acordo com a data da atualização. Fora isso, criar índices irá tornar as consultas mais rápidas e eficientes, o Apache Parquet possui suporte para essa indexação. Dependendo das características dos dados podemos pensar em indexação via estruturas de dados mais complexas, que otimizam o tempo da busca.

Ainda podemos pensar em implementar um sistema de memória em cache, que salva os arquivos mais acessados para melhor eficiência de consulta.

Olhando diretamente para o código PySpark podemos otimizar ainda mais o tempo, tanto no momento em que fazemos a requisição ao banco, quanto ao carregar apenas os dados necessários na memória. Quando os dados são pequenos podemos evitar serialização repetidas vezes. Também é interessante reduzir os embaralhamentos e otimizar o dataframe.

Como as atualizações no arquivo acontecem de forma mensal, é interessante automatizar essa tarefa. Para isso é possível utilizar cron jobs, fazendo com que os arquivos sejam sempre atualizados em horários de pouco uso, minimizando impacto no desempenho.

==========
QUESTÃO 5
Você foi instruído a criar um modelo de regressão baseado em um modelo econométrico. Ao terminar a implementação você verificou que o resultado está abaixo do esperado. Liste o que poderia ser feito para melhorar o resultado.

Um modelo de regressão baseado em um modelo econométrico variáveis econômicas. Existem diferentes abordagens que melhoram este modelo.
Num primeiro momento é necessário dar atenção às variáveis. Elas são relevantes para a previsão do resultado? Existe multicolinearidade entre elas? Os erros são independentes entre si e possuem variância constante?

Feita essa análise, podemos analisar se existem transformações que podem ser aplicadas às variáveis, desde operações de normalização até transformações não lineares, como por exemplo, polinômios, que podem ajudar em relacionamentos mais complexos.
Também é importante estar atento aos resíduos do modelo, resíduos inconstantes exigem a aplicação de uma regressão ponderada. Caso exista correlação, alguns modelos específicos podem ser aplicados.

Modelos com muitas características podem impactar negativamente na regressão, aplicar técnicas de seleção de atributos pode ajudar a reduzir o overfitting.

Fora essas análises, muitos outros aspectos podem ser vistos com a intenção de melhorar o resultado de uma regressão, como por exemplo a validação cruzada estratificada, a limpeza e o aumento do conjunto de dados, a utilização de modelos alternativos, entre outros.
==========
QUESTÃO 6
ELT e ETL são termos comuns na rotina de trabalho de especialistas em Soluções de Tecnologia da Informação, justamente por representarem estratégias de pipelines de integração
de dados em um determinado projeto. Sobre esses procedimentos, é correto afirmar que:
a) Ambos necessitam de uma etapa de cópia dos dados em um armazenamento intermediário, antes de serem transferidos para o banco de dados consolidado.
b) Modelos locais, que utilizem dados relacionais e estruturados são ideais para estratégias

ELT, por apresentarem menores custos e necessidades de hardware.
c) Estratégias ETL realizam a etapa de tratamento diretamente no Banco de Dados Consolidado.
d) Estrutura de dados em nuvem são ideais para a adoção de estratégias ELT, devido a maior rapidez no carregamento dos dados e a adequada capacidade de processamento posterior.
e) Pipelines de integração do tipo ETL é indicada para casos em que a organização utilizada
dados não-estruturados.

Primeiro podemos fazer uma revisão dos conceitos abordados na questão.
ELT (Extract, Load e Transform) ⇒ Os dados são extraídos e carregados num banco de dados de destino, após isso podem ser transformados. Esse modelo costuma ser mais utilizado em Data Lakes (repositórios de dados estruturados e não estruturados) e Big Data, isso porque a capacidade de processamento pode ser melhor aproveitada após o carregamento. Essa estrutura costuma ser mais flexível e escalável, além
de contar tipicamente com estruturas em nuvem .

ETL (Extract, Transform, Load) ⇒ Os dados são extraídos, transformados num formato adequado e por fim carregados no banco de destino. Esse método é mais tradicional e costuma ser utilizado em cenários de menor complexidade.

Agora podemos analisar as alternativas, começando com as incorretas:
A alternativa ‘a’ é incorreta. O ELT não precisa utilizar o sistema de armazenamento intermediário, os dados podem ser carregados diretamente no banco de destino. Por outro lado, o ETL tem esta etapa obrigatória.

A alternativa ‘b’ também é incorreta. Os ELTs costumam necessitar de grandes capacidades de armazenamento e processamento, eles também podem utilizar modelos não estruturados. A escalabilidade da nuvem é ideal para estes conceitos.

A alternativa ‘c’ também está incorreta. A transformação dos dados acontece antes do carregamento no banco de dados, isso acontece para que evitemos impactos na performance do banco.

Por fim, a alternativa ‘e’ é incorreta. O ETL é utilizado em dados estruturados, já o ELT utiliza ambos, dados estruturados e não estruturados.

A única alternativa que é correta é a ‘d’, isso porque o ELT se beneficia de estruturas de dados em nuvem, tanto da sua capacidade de processamento, quanto da escalabilidade. Este tipo de arquitetura permite um rápido carregamento e uma transformação eficiente.