1) Respostas

1.1 –

ComercioExterior/
- BRA/
  - ARG/
    - 2019.txt
    - 2020.txt
    - 2021.txt
  - AFG/
    - 2019.txt
    - 2020.txt
    - 2021.txt
- ARG/
  - BRA/
    - 2019.txt
    - 2020.txt
    - 2021.txt
  - AFG/
    - 2019.txt
    - 2020.txt
    - 2021.txt

1.2 –

Cada arquivo conterá os dados de comércio exterior entre os países indicados no nome do diretório de origem/destino e no ano correspondente.

Integração de 3 anos de comércio exterior
1 – Criar a estrutura de diretórios
2 – Repetir sobre todos os países de origem/destino válidas
3 – Para cada combinação de país de origem, país destino e ano, fazer a consulta e salvar os dados no arquivo correspondente
4 – Garantir haja somente consultas válidas, exemplo de consultas inválidas: (BRA, BRA, ano) ou (WLD, BRA, ano)

Para a pergunta “Qual a quantidade de soja o mundo importou do Brasil em 2020?”, o arquivo é: “ComercioExterior/BRA/WLD/2020.txt”.

Código para realizar as consultas e salvar os dados
Em python:

```
Import os
Import requests
```

```
# Configurações da API

API_KEY = “sua_chave_api”
BASE_URL = https://api.comercio_exter.com

# Lista de países

Lista_paises = [“BRA”, “ARG”, “AFG”, ...]                # Completar os 143
países

# Anos para consulta

anos = [“2019”, “2020”, “2021”]

# Função para criar diretórios
def criar_diretorios(base_path, anos, lista_paises):
        for pais_origem in lista_paises:
                for pais_destino in lista_paises:
                        if pais_origem != pais_destino and pais_origem != "WLD"
                and pais_destino != "WLD": for ano in anos:
                        path = os.path.join(base_path, pais_origem, pais_destino)
                os.makedirs(path, exist_ok=True)


# Função para realizar consultas e salvar dados
def consultar_e_salvar(base_path, lista_paises, anos):
        for pais_origem in lista_paises:
                for pais_destino in lista_paises:
                        if pais_origem != pais_destino and pais_origem != "WLD"
                and pais_destino != "WLD": for ano in anos:
                                response = requests.get( f"{BASE_URL}/data",
                                        params={ "api_key": API_KEY, "pais_origem":
                                pais_origem, "pais_destino": pais_destino, "ano":
                                ano
                                         }
                                )
                                if response.status_code == 200:
                                        data = response.json()
                                        file_path = os.path.join(base_path,
                                pais_origem, pais_destino, f"{ano}.txt")
                                        with open(file_path, 'w') as file:
                                        file.write(str(data))

# Base para os dados
base_path = "./ComercioExterior"

# Criação dos  diretórios
```

criar_diretorios(base_path, anos, lista_paises)

# Consultar API e salvar dados
consultar_e_salvar(base_path, lista_paises, anos)

1.3 –

ComercioExterior/BRA/WLD/2020.txt

1.4 –

Considerando os parâmetros mensais e anuais, levando em conta que estamos em 1 de fevereiro de 2025, podemos seguir as seguintes etapas:

1.4.1. Configurar a API e autenticação

Obter a chave API e configure as credenciais para acessar API de comércio exterior

1.4.2. Preparar a lista de países

Definir e obter a lista completa dos 143 países

1.4.3. Criar a estrutura de diretórios

° Criar diretórios para cada ano e mês, seguindo a estrutura definida
° Dentro de cada diretório de ano, crie subdiretórios para cada país de origem/destino

1.4.4. Análise sobre os parâmetros da consulta

° Para cada país de origem, repetir para cada país de destino
° Para cada combinação de país origem e destino, repetir os anos

1.4.5. Realizar consultas mensais

° Verificar se a consulta é válida
° Realizar a consulta mensal para cada mês disponível
° Salve os dados retornados pela API nos arquivos correspondentes na estrua de diretórios

1.4.6. Realizar consultas anuais

° Verificar se a consulta é válida
° Para cada ano até o ano anterior ao atual, realize a consulta anual

° Salve os dados retornados pela API nos arquivos correspondentes na estrutura de diretórios

1.4.7. Tratamento de consultas inválidas

Garanta que não haja consultas do tipo (BRA, BRA, ano) ou (WLD, BRA,ano)

1.4.8. Agendar atualizações periódicas

Agendo a execução do script de atualização regularmente para garantir que os dados estejam sempre atualizados.

2. Resposta

Automatizar o processo de download, extração, empilhamento e atualização contínua do conjunto de dados de empresas (0-9) da Receita Federal, mesmo com a instabilidade do site.

Etapas:

2.1. Configurar o Ambiente e Dependências:

Primeiro, eu instalo todas as bibliotecas necessárias, como requests, zipfile, pandas e retrying, e configuro um ambiente virtual para isolar as dependências do projeto.

2.2. Definir Variáveis e URLs:

Em seguida, defino as URLs fixas dos arquivos de empresas e crio variáveis para os diretórios de download e armazenamento dos arquivos extraídos.

2.3. Verificar Atualização:

Eu acesso a página principal da Receita Federal e obtenho a data da última atualização. Comparo essa data com a data da última atualização local, que está armazenada em um arquivo ou banco de dados. Se a data da última atualização for mais recente, sei que há novos arquivos para baixar.

2.4. Baixar Arquivos:

Utilizo a biblioteca requests para baixar os arquivos compactados (CNPJ0.zip, CNPJ1.zip, ..., CNPJ9.zip). Implemento uma lógica de tentativas com recomeço, utilizando a biblioteca retrying, para lidar com falhas na conexão ou instabilidade do site. Salvo os arquivos baixados em um diretório local específico.

2.5. Verificar Integridade dos Arquivos:

Verifico se os arquivos baixados estão corrompidos. Se algum arquivo estiver corrompido, rebaixo o arquivo até obter uma versão íntegra.

2.6. Extrair Arquivos:

Uso a biblioteca zipfile para extrair os arquivos CSV de cada arquivo compactado baixado. Salvo os arquivos CSV extraídos em outro diretório local.

2.7. Empilhar Arquivos:

Utilizo a biblioteca pandas para ler cada arquivo CSV extraído e concatená-los em um único DataFrame. Salvo o DataFrame empilhado em um arquivo mestre.

2.8. Atualizar Data de Última Atualização:

Atualizo a data da última atualização local com a data da última atualização obtida na etapa de verificação.

2.9. Automatizar o Processo:

Configuro um agendador de tarefas (como cron jobs no Linux ou Task Scheduler no Windows) para executar o script em intervalos regulares, como semanalmente ou mensalmente. Monitoro a execução do script e envio notificações em caso de falhas, por exemplo, por email.

2.10. Documentar o Processo:

Documentar todas as etapas do processo, incluindo dependências, configurações e instruções de execução. Manter um log das execuções, registrando sucessos e falhas.

2.11. Observações Finais:

Verificação de Integridade:

Adiciono verificações para garantir que os arquivos baixados e extraídos não estejam corrompidos, utilizando checksums ou verificação de tamanho de arquivos.

2.12. Tratamento de Erros:

Implemento tratamentos de exceção robustos para capturar e lidar com erros durante o download, extração e empilhamento. Adiciono logs para monitorar o processo e facilitar a identificação de problemas.

2.13. Notificações:

Considero integrar notificações por email ou outro meio para alertar sobre falhas no processo de atualização.

2.14. Monitoramento e Manutenção:

Monitoro periodicamente o script para garantir que ele esteja funcionando corretamente e atualizando os dados conforme o esperado.

3. Respostas

3.1. Consulta SQL

Para obter o nome do departamento onde cada funcionário trabalhou pela primeira vez, o nome do departamento onde ele está atualmente, e a quantidade de departamentos diferentes em que ele já trabalhou, a consulta SQL que eu escreveria é a seguinte:

```
SELECT
    e.employee_name,
    d_first.department_name AS first_department,
    d_current.department_name AS current_department,
    COUNT(DISTINCT edh.department_id) AS num_departments
FROM
    employees e
JOIN
    employee_department_history edh ON e.employee_id = edh.employee_id
JOIN
    departments d_first ON d_first.department_id = (
        SELECT department_id
        FROM employee_department_history
        WHERE employee_id = e.employee_id
        ORDER BY start_date
        LIMIT 1
    )
LEFT JOIN
    departments d_current ON d_current.department_id = (
        SELECT department_id
        FROM employee_department_history
        WHERE employee_id = e.employee_id
```

```
        AND end_date IS NULL
        LIMIT 1
    )
GROUP BY
    e.employee_id, e.employee_name, d_first.department_name,
d_current.department_name;
```

Essa consulta une as tabelas employees, employee_department_history, e departments. A junção com employee_department_history garante que estou considerando apenas os registros relevantes para o histórico de departamentos de cada funcionário. Uso um JOIN com departments para obter o primeiro departamento e um LEFT JOIN para obter o departamento atual. O COUNT(DISTINCT edh.department_id) calcula a quantidade de departamentos diferentes em que o funcionário trabalhou.

3.4. Otimização para API com Menor Poder de Processamento

Para criar uma API que consiga manter o tempo de consulta rápido para um arquivo Parquet de 7GB, mesmo com recursos modestos, eu consideraria as seguintes alterações e procedimentos:

Compactação: Reduzir o tamanho do arquivo Parquet usando técnicas de compactação como Snappy ou Gzip.

Particionamento: Dividir o arquivo em partições menores, facilitando a leitura e evitando a necessidade de carregar todo o arquivo de uma vez.

Cache: Implementar cache para armazenar os resultados frequentemente acessados, evitando leituras repetidas.

Otimização de Consultas: Usar índices e otimizar consultas para evitar varreduras completas do arquivo.

Processamento Assíncrono: Executar a leitura e o processamento em segundo plano, para não bloquear a API.

Atualização Programada: Agendar a atualização do arquivo Parquet uma vez por mês, conforme necessário.

3.5. Melhorando o Resultado de um Modelo de Regressão

Para melhorar os resultados de um modelo de regressão baseado em um modelo econométrico, eu faria o seguinte:

Exploração de Dados: Analisar os dados para entender melhor as relações entre as variáveis.

Seleção de Variáveis: Escolher as variáveis mais relevantes e eliminar as redundantes.

Transformação de Variáveis: Aplicar transformações (como log ou raiz quadrada) para melhorar a linearidade.

Tratamento de Outliers: Identificar e tratar valores extremos.

Validação Cruzada: Usar validação cruzada para avaliar o desempenho do modelo.

Regularização: Experimentar técnicas como LASSO ou Ridge para evitar overfitting.

Teste de Hipóteses: Verificar a significância estatística das variáveis.

Diagnóstico de Resíduos: Avaliar a adequação do modelo aos dados observados.

3.6. ELT vs. ETL

Sobre ELT e ETL, a afirmativa correta é:

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.