Plano Estratégico para Análise de Vagas de Emprego no Brasil com Big Data e Python

1. Revisar e Refinar a Estratégia de Aquisição de Dados

Desafios Identificados na Aquisição de Dados Reais

Conforme a análise anterior, a obtenção de dados reais e atualizados sobre vagas de emprego no Brasil apresenta desafios significativos. As fontes governamentais diretas para dados de vagas individuais em formato aberto (APIs) são limitadas, desatualizadas ou inacessíveis. Os portais de emprego comerciais, por sua vez, geralmente não oferecem APIs públicas para coleta em larga escala, visando proteger seus dados e modelos de negócio. Isso nos leva a considerar abordagens alternativas e complementares.

Fontes de Dados Potenciais e Estratégias de Aquisição

Para o sucesso do projeto, a estratégia de aquisição de dados deve ser multifacetada e adaptável:

1. **Dados Agregados e Microdados Governamentais (MTE e IBGE)**:
    - **CAGED (Cadastro Geral de Empregados e Desempregados)**: Esta é uma fonte robusta para dados de emprego formal, incluindo admissões e desligamentos por setor, município, e tipo de contrato. Os dados são disponibilizados mensalmente e anualmente. Embora não forneça vagas individuais, é crucial para entender as tendências macro do mercado de trabalho, o que pode contextualizar a análise de vagas.
    - **RAIS (Relação Anual de Informações Sociais)**: Complementar ao CAGED, a RAIS oferece um panorama anual mais detalhado do emprego formal, com informações sobre salários, ocupações e características dos trabalhadores. Ideal para análises históricas e de longo prazo.
    - **PNAD Contínua (Pesquisa Nacional por Amostra de Domicílios Contínua) do IBGE**: Fornece microdados detalhados sobre a força de trabalho, ocupação, desocupação, rendimento e outras características demográficas e sociais. É uma fonte valiosa para análises aprofundadas do mercado de trabalho, incluindo o setor informal, mas requer processamento complexo para extrair informações relevantes para vagas.
    - **Estratégia de Aquisição**: Para CAGED e RAIS, os dados são geralmente baixados em formatos como CSV ou XLSX diretamente dos portais do MTE. Para a PNAD Contínua, os microdados são disponibilizados no site do IBGE, muitas vezes em

formatos como TXT ou CSV, acompanhados de dicionários de variáveis e programas para leitura em softwares estatísticos (SAS, SPSS, R). A automação do download pode ser feita com Python utilizando bibliotecas como `requests` e `BeautifulSoup` para web scraping de links de download, ou diretamente via APIs se disponíveis para datasets específicos (o que é raro para microdados).

2. **Web Scraping de Portais de Emprego (com cautela)**:
    - **Portais Alvo**: Indeed, Infojobs, Vagas.com.br, Catho, Glassdoor, LinkedIn Jobs, Trabalha Brasil, entre outros. Cada portal possui sua própria estrutura e termos de serviço.
    - **Estratégia de Aquisição**: O web scraping envolve o uso de bibliotecas Python como `requests` para fazer requisições HTTP e `BeautifulSoup` ou `Scrapy` para parsear o HTML e extrair as informações das vagas (título, descrição, empresa, localização, setor, requisitos, data de publicação). É fundamental:
        - **Verificar os Termos de Serviço (ToS)** de cada site para garantir que o scraping é permitido ou para entender as restrições. A violação dos ToS pode levar a bloqueios de IP ou ações legais.
        - **Respeitar o `robots.txt`**: Este arquivo indica quais partes do site podem ou não ser rastreadas por bots.
        - **Implementar atrasos (delays)** entre as requisições para não sobrecarregar os servidores do site e evitar ser detectado como bot.
        - **Usar User-Agents rotativos e proxies**: Para simular diferentes usuários e IPs, dificultando o bloqueio.
        - **Lidar com CAPTCHAs e JavaScript dinâmico**: Pode exigir o uso de ferramentas como Selenium para simular um navegador real.
    - **Considerações**: O web scraping é uma solução de alto custo de manutenção, pois a estrutura dos sites pode mudar frequentemente, quebrando os scripts de coleta. É uma abordagem viável para protótipos ou coleta de dados em menor escala, mas para um projeto de Big Data contínuo, parcerias ou APIs pagas seriam mais sustentáveis.
3. **APIs de Agregadores de Vagas ou Parceiros Comerciais (se disponíveis)**:
    - **Agregadores**: Alguns serviços podem consolidar vagas de diversos portais e oferecer acesso via API (muitas vezes paga). A pesquisa inicial não revelou opções gratuitas robustas no Brasil, mas uma investigação mais aprofundada ou a consideração de serviços pagos pode ser necessária.
    - **Parcerias Diretas**: Contatar diretamente os portais de emprego para negociar acesso aos dados via API. Esta é a abordagem mais ideal para um projeto de Big

Data que busca dados individuais em larga escala e de forma contínua, mas exige recursos e tempo para negociação.

- **Estratégia de Aquisição**: Se uma API for disponibilizada, a aquisição de dados será via requisições HTTP (GET/POST) usando a biblioteca `requests` em Python, processando as respostas JSON ou XML. A documentação da API será crucial para entender os endpoints, parâmetros e limites de requisição.

Priorização da Estratégia de Aquisição

Para iniciar, sugiro uma abordagem híbrida:

- **Prioridade 1**: Focar na coleta e integração dos dados agregados e microdados do **MTE (CAGED, RAIS)** e **IBGE (PNAD Contínua)**. Estes dados são oficiais, confiáveis e essenciais para análises macro e de tendências setoriais, mesmo que não forneçam vagas individuais.
- **Prioridade 2**: Desenvolver um **web scraper experimental** para um ou dois portais de emprego de menor porte ou com termos de serviço mais flexíveis, apenas para entender a dinâmica de coleta de vagas individuais e a estrutura dos dados. Isso servirá como prova de conceito e para identificar os desafios práticos.
- **Prioridade 3**: Paralelamente, iniciar a **investigação de APIs pagas ou potenciais parcerias** com grandes portais, caso a necessidade de dados de vagas individuais em tempo real e em larga escala seja crítica para o projeto.

Esta estratégia permite construir uma base sólida com dados oficiais enquanto exploramos as opções para dados de vagas individuais, gerenciando os riscos e custos associados à aquisição de dados em Big Data.

2. Definir Arquitetura de Big Data e Ferramentas

A definição da arquitetura de Big Data é um passo crucial para garantir a escalabilidade, eficiência e robustez do projeto. A escolha das ferramentas dependerá do volume de dados, da velocidade de processamento necessária, do orçamento disponível e da expertise da equipe. No contexto de análise de vagas de emprego, onde podemos lidar com volumes crescentes de dados semi-estruturados (web scraping) e estruturados (dados governamentais), uma arquitetura flexível e escalável é fundamental.

Componentes Essenciais de uma Arquitetura de Big Data

Uma arquitetura típica de Big Data para este tipo de projeto pode incluir os seguintes componentes:

1. **Camada de Ingestão de Dados (Data Ingestion Layer)**:

- **Função**: Responsável por coletar dados de diversas fontes (web scraping, APIs, arquivos CSV/XLSX) e movê-los para a camada de armazenamento de forma eficiente.
- **Ferramentas Python**: `requests` para APIs e HTTP, `BeautifulSoup` ou `Scrapy` para web scraping. Para orquestração e agendamento de tarefas de ingestão, ferramentas como Apache Airflow ou Prefect são altamente recomendadas. Elas permitem definir fluxos de trabalho (DAGs - Directed Acyclic Graphs) para automatizar a coleta, validação e carregamento dos dados.
- **Ferramentas de Streaming (Opcional)**: Se houver necessidade de processamento de dados em tempo real (ex: monitoramento de novas vagas instantaneamente), Apache Kafka pode ser utilizado como um barramento de mensagens para ingestão de streams de dados.

2. **Camada de Armazenamento de Dados (Data Storage Layer)**:
   - **Função**: Armazenar os dados brutos e processados de forma escalável e tolerante a falhas.
   - **Opções de Armazenamento**: A escolha depende do tipo e volume de dados:
     - **Data Lake (Armazenamento de Objetos)**: Para dados brutos e semi-estruturados (JSON, CSV, HTML de páginas web), serviços como Amazon S3, Google Cloud Storage (GCS) ou Azure Blob Storage são ideais. Eles oferecem armazenamento de baixo custo, alta durabilidade e escalabilidade ilimitada. Os dados podem ser armazenados em seu formato original (raw data).
     - **Bancos de Dados NoSQL**: Para dados semi-estruturados ou não estruturados que exigem flexibilidade de esquema e alta escalabilidade horizontal. Exemplos incluem MongoDB (documentos), Cassandra (colunas largas) ou Elasticsearch (busca e análise de texto). Útil para armazenar detalhes de vagas de emprego que podem ter campos variáveis.
     - **Data Warehouse (Bancos de Dados Colunares)**: Para dados estruturados e otimizados para consultas analíticas complexas. Exemplos incluem Google BigQuery, Amazon Redshift, Snowflake. Ideal para armazenar dados de vagas processados e agregados, bem como dados do CAGED/RAIS/PNAD, permitindo análises rápidas e relatórios.
   - **Considerações**: Uma arquitetura comum é usar um Data Lake para dados brutos e, após o processamento, carregar os dados limpos e estruturados em um Data Warehouse ou Banco de Dados NoSQL para consumo analítico.
3. **Camada de Processamento de Dados (Data Processing Layer)**:
   - **Função**: Transformar, limpar, enriquecer e agregar os dados brutos em formatos adequados para análise.

- **Ferramentas Python**: Python é a linguagem central aqui. Bibliotecas como `Pandas` são excelentes para manipulação e transformação de dados em memória para volumes menores. Para Big Data, frameworks como **Apache Spark com PySpark** são indispensáveis. Spark permite processamento distribuído em clusters, lidando com grandes volumes de dados de forma eficiente e escalável. Outras bibliotecas úteis incluem `NumPy` para operações numéricas e `Scikit-learn` para tarefas de Machine Learning (ex: classificação de vagas por setor, extração de entidades).
- **Ferramentas de Orquestração**: Apache Airflow ou Prefect também orquestram as etapas de processamento, garantindo que as transformações ocorram na ordem correta e que as dependências sejam satisfeitas.

4. **Camada de Análise e Modelagem (Analytics & Modeling Layer)**:
   - **Função**: Realizar análises exploratórias, construir modelos preditivos (ex: previsão de demanda por vagas, identificação de tendências), e gerar insights.
   - **Ferramentas Python**: `Jupyter Notebooks` para desenvolvimento interativo e exploração de dados. `Pandas`, `NumPy`, `SciPy` para análise estatística. `Scikit-learn`, `TensorFlow` ou `PyTorch` para Machine Learning. `Matplotlib`, `Seaborn`, `Plotly` para visualização de dados.
5. **Camada de Visualização e Relatórios (Visualization & Reporting Layer)**:
   - **Função**: Apresentar os insights de forma clara e interativa para os usuários finais.
   - **Ferramentas**: Ferramentas de Business Intelligence (BI) como Tableau, Power BI, Looker Studio (antigo Google Data Studio) ou Metabase. Para dashboards customizados, frameworks web como Flask ou Django (com bibliotecas de visualização Python) podem ser utilizados.

Exemplo de Arquitetura Simplificada para Início do Projeto

Para começar, uma arquitetura mais simplificada pode ser adotada e expandida conforme a necessidade:

- **Ingestão**: Scripts Python (`requests`, `BeautifulSoup`) agendados via `cron` (para protótipos) ou Apache Airflow (para produção).
- **Armazenamento**: Data Lake (Amazon S3/GCS) para dados brutos e um banco de dados relacional (PostgreSQL) ou NoSQL (MongoDB) para dados processados e estruturados, dependendo da flexibilidade de esquema desejada.
- **Processamento**: Scripts Python com `Pandas` para volumes menores ou `PySpark` para volumes maiores, executados em máquinas virtuais ou em um cluster Spark.
- **Análise/Visualização**: Jupyter Notebooks para exploração e dashboards simples com `Matplotlib` / `Seaborn` ou ferramentas de BI conectadas ao banco de dados processado.

À medida que o projeto cresce e o volume de dados aumenta, a arquitetura pode evoluir para incluir mais componentes de Big Data, como clusters Hadoop/Spark dedicados, Data Warehouses em nuvem e ferramentas de orquestração mais robustas. A chave é começar com uma solução que atenda às necessidades atuais e que seja escalável para o futuro.

3. Planejar Ingestão e Armazenamento de Dados

O planejamento da ingestão e armazenamento de dados é fundamental para garantir que os dados sejam coletados de forma eficiente, armazenados de maneira otimizada e estejam prontos para processamento e análise. Esta fase detalha as estratégias e ferramentas para lidar com os diferentes tipos de dados de vagas de emprego.

Estratégias de Ingestão de Dados

A ingestão de dados pode ser categorizada em batch (lotes) ou streaming (tempo real), dependendo da necessidade de atualização e do volume de dados. Para o projeto de análise de vagas, a ingestão em batch será predominante para dados governamentais e web scraping, enquanto o streaming pode ser uma consideração futura para fontes de dados em tempo real.

1. **Ingestão de Dados Governamentais (CAGED, RAIS, PNAD)**:
    - **Frequência**: Mensal (CAGED), Anual (RAIS), Trimestral/Anual (PNAD Contínua).
    - **Mecanismo**: Download direto dos portais do MTE e IBGE. Isso pode ser automatizado com scripts Python que utilizam a biblioteca `requests` para baixar os arquivos (CSV, XLSX, TXT) e, se necessário, `BeautifulSoup` para navegar nas páginas e encontrar os links de download. É crucial implementar verificações para novas versões dos arquivos e lidar com possíveis mudanças na estrutura dos sites.
    - **Orquestração**: Ferramentas como Apache Airflow são ideais para agendar e monitorar esses downloads. Um DAG (Directed Acyclic Graph) no Airflow pode ser configurado para:
        - Verificar a disponibilidade de novos arquivos.
        - Baixar os arquivos para um diretório temporário.
        - Mover os arquivos brutos para o Data Lake (ex: S3, GCS).
        - Acionar a próxima etapa de processamento.
2. **Ingestão via Web Scraping (Portais de Emprego)**:
    - **Frequência**: Diária ou semanal, dependendo da volatilidade das vagas e da necessidade de atualização.

- **Mecanismo**: Scripts Python utilizando `requests` e `BeautifulSoup` (para sites mais simples) ou `Scrapy` (para projetos de scraping mais complexos e escaláveis). Para lidar com JavaScript dinâmico, `Selenium` pode ser integrado. É vital:
  - **Gerenciamento de Proxies e User-Agents**: Para evitar bloqueios de IP.
  - **Tratamento de Erros e Retries**: Para lidar com falhas de rede ou bloqueios temporários.
  - **Extração de Dados Estruturados**: Identificar os seletores CSS ou XPath corretos para extrair título da vaga, descrição, empresa, localização, setor, requisitos, salário (se disponível), data de publicação, URL da vaga, etc.
- **Orquestração**: Novamente, Apache Airflow é a ferramenta preferencial para agendar e gerenciar esses crawlers, garantindo que eles rodem em intervalos definidos e que os dados coletados sejam armazenados corretamente.

3. **Ingestão via APIs (se disponíveis)**:
   - **Frequência**: Conforme a taxa de atualização da API (pode ser em tempo real ou batch).
   - **Mecanismo**: Utilizar a biblioteca `requests` em Python para fazer chamadas HTTP aos endpoints da API. É necessário gerenciar chaves de API, limites de requisição e paginação. As respostas (geralmente JSON) devem ser parseadas e validadas.
   - **Orquestração**: Airflow pode ser usado para agendar chamadas de API e processar as respostas.

Estratégias de Armazenamento de Dados

O armazenamento deve ser planejado para acomodar dados brutos (raw data) e dados processados (curated data), garantindo acessibilidade, durabilidade e custo-benefício.

1. **Data Lake para Dados Brutos (Raw Data)**:
   - **Propósito**: Armazenar todos os dados coletados em seu formato original, sem modificações. Isso é crucial para auditoria, reprocessamento e para suportar futuras análises que possam exigir os dados brutos.
   - **Tecnologia**: Serviços de armazenamento de objetos em nuvem como **Amazon S3**, **Google Cloud Storage (GCS)** ou **Azure Blob Storage**. São altamente escaláveis, duráveis e de baixo custo.
   - **Estrutura**: Organizar os dados em um esquema de pastas lógico, por exemplo: `/raw/<fonte>/<ano>/<mes>/<dia>/<arquivo>`. Isso facilita a governança e a recuperação de dados.
   - **Formato**: Manter o formato original (CSV, XLSX, TXT, JSON, HTML).

2. **Data Warehouse ou Banco de Dados NoSQL para Dados Processados (Curated Data)**:
    - **Propósito**: Armazenar dados limpos, transformados e estruturados, otimizados para consultas analíticas e consumo por aplicações de BI ou Machine Learning.
    - **Tecnologia**: A escolha depende da natureza dos dados e dos requisitos de consulta:
        - **Data Warehouse (ex: Google BigQuery, Amazon Redshift, Snowflake)**: Ideal para dados estruturados (CAGED, RAIS, PNAD processada, vagas de emprego com esquema fixo) que serão usados para análises complexas e relatórios. Oferecem alta performance para consultas SQL em grandes volumes de dados.
        - **Banco de Dados NoSQL (ex: MongoDB, Apache Cassandra, Elasticsearch)**: Adequado para dados semi-estruturados ou com esquema flexível, como detalhes de vagas de emprego que podem ter campos variados. MongoDB é uma boa opção para documentos JSON, enquanto Elasticsearch é excelente para busca e análise de texto em descrições de vagas.
    - **Estrutura**: Para Data Warehouses, definir um esquema de tabelas bem modelado (ex: esquema estrela ou floco de neve). Para NoSQL, planejar a estrutura dos documentos ou coleções para otimizar as consultas mais frequentes.
    - **Formato**: Dados limpos, padronizados e enriquecidos. Pode incluir campos como `id_vaga`, `titulo`, `descricao_limpa`, `empresa`, `localizacao`, `setor_padronizado`, `salario_min`, `salario_max`, `data_publicacao`, `url_origem`, `habilidades_extraidas`, etc.

Considerações de Segurança e Governança

- **Controle de Acesso**: Implementar políticas de acesso rigorosas para o Data Lake e o Data Warehouse, garantindo que apenas usuários e serviços autorizados possam acessar os dados.
- **Criptografia**: Criptografar dados em repouso (no armazenamento) e em trânsito (durante a ingestão e processamento).
- **Retenção de Dados**: Definir políticas de retenção de dados para gerenciar o ciclo de vida dos dados e cumprir requisitos regulatórios.
- **Qualidade de Dados**: Implementar verificações de qualidade de dados durante a ingestão para identificar e mitigar problemas como dados duplicados, incompletos ou inconsistentes. Isso pode ser feito com bibliotecas Python como `Great Expectations`.

Ao planejar cuidadosamente a ingestão e o armazenamento, garantimos que o projeto terá uma base sólida de dados confiáveis e acessíveis para as etapas subsequentes de processamento e análise.

4. Desenvolver Processamento e Transformação de Dados com Python

Após a ingestão, os dados brutos precisam ser processados e transformados para se tornarem úteis para análise. Esta etapa é crítica para garantir a qualidade, consistência e usabilidade dos dados. Python, com seu rico ecossistema de bibliotecas, é a ferramenta ideal para realizar essas operações, desde a limpeza básica até o enriquecimento complexo e a padronização de informações.

Etapas Chave de Processamento e Transformação

1. **Limpeza de Dados (Data Cleaning)**:
    - **Remoção de Duplicatas**: Identificar e remover registros duplicados, que podem ocorrer devido a múltiplas fontes de dados ou falhas na ingestão. Bibliotecas como `pandas` são eficientes para isso (`df.drop_duplicates()`).
    - **Tratamento de Valores Ausentes**: Decidir como lidar com dados faltantes (NaN, None). Opções incluem preenchimento com valores médios/medianos/moda, interpolação, ou remoção de linhas/colunas com muitos valores ausentes. A escolha depende do contexto e da porcentagem de dados faltantes.
    - **Padronização de Formatos**: Garantir que os dados estejam em formatos consistentes. Exemplos:
        - Datas: Converter todas as datas para um formato padrão (ex: YYYY-MM-DD).
        - Textos: Converter para minúsculas, remover caracteres especiais, espaços extras.
        - Numéricos: Garantir que campos numéricos sejam tratados como tal e não como strings.
    - **Correção de Erros Tipográficos**: Utilizar técnicas de correspondência fuzzy (`fuzzywuzzy`) ou dicionários de correção para padronizar nomes de empresas, cargos, etc., que podem ter variações.
2. **Enriquecimento de Dados (Data Enrichment)**:
    - **Extração de Entidades (Named Entity Recognition - NER)**: Usar bibliotecas de Processamento de Linguagem Natural (NLP) como `spaCy` ou `NLTK` para extrair entidades importantes de descrições de vagas, como habilidades técnicas (Python, SQL, AWS), ferramentas (Tableau, Power BI), certificações, e tecnologias específicas. Isso permite criar novas colunas estruturadas a partir de texto não estruturado.
    - **Classificação de Setores**: Atribuir um setor padronizado para cada vaga. Isso pode ser feito através de:

        - **Mapeamento baseado em palavras-chave**: Criar um dicionário de palavras-chave que mapeiam para setores específicos.
        - **Modelos de Machine Learning**: Treinar um classificador de texto (ex: Naive Bayes, SVM, modelos de deep learning com `scikit-learn` ou `TensorFlow` / `PyTorch`) usando descrições de vagas e setores conhecidos para prever o setor de novas vagas.
    - **Geocodificação**: Converter endereços ou nomes de cidades em coordenadas geográficas (latitude e longitude) para análises espaciais. Bibliotecas como `geopy` podem ser usadas com APIs de geocodificação (ex: Google Maps API, OpenStreetMap Nominatim).
    - **Normalização de Salários**: Se os salários forem extraídos em formatos variados (ex: R$ 2.000-3.000, A combinar, 5k por mês), normalizá-los para um formato numérico consistente (ex: salário mínimo e máximo em BRL). Isso pode envolver expressões regulares (`re`) e lógica condicional.
1. **Padronização e Categorização (Standardization & Categorization)**:
    - **Unificação de Termos**: Criar dicionários ou regras para unificar termos semelhantes (ex: "Desenvolvedor", "Dev", "Programador" para "Desenvolvedor").
    - **Categorização de Habilidades**: Agrupar habilidades extraídas em categorias mais amplas (ex: "Python", "Java", "C++" em "Linguagens de Programação").
    - **Mapeamento de Cargos**: Padronizar os títulos de cargos para um conjunto predefinido, facilitando a análise comparativa.
2. **Agregação de Dados (Data Aggregation)**:
    - **Criação de Métricas**: Calcular métricas relevantes, como número de vagas por setor, por localização, por nível de experiência, salário médio por cargo/setor, etc.
    - **Dados Temporais**: Agrupar dados por período (dia, semana, mês, ano) para análise de tendências temporais.

Ferramentas Python para Processamento e Transformação

- **Pandas**: Essencial para manipulação e análise de dados tabulares em memória. Ideal para tarefas de limpeza, transformação e agregação em datasets de tamanho médio. Permite operações rápidas e flexíveis com DataFrames.
- **NumPy**: Base para operações numéricas de alta performance, frequentemente usado em conjunto com Pandas.
- **Scikit-learn**: Para tarefas de Machine Learning, como classificação de texto (para categorização de setores ou habilidades), agrupamento (clustering) de vagas similares, ou extração de características.

- **NLTK (Natural Language Toolkit) / spaCy**: Para processamento de linguagem natural, incluindo tokenização, remoção de stopwords, lematização/stemming e NER (Named Entity Recognition) para extrair informações de texto livre nas descrições das vagas.
- **Regex (módulo `re`)**: Para extração de padrões específicos de texto, como valores salariais, datas ou códigos.
- **Apache Spark (PySpark)**: Para processamento de Big Data em larga escala. Quando os volumes de dados excedem a capacidade de memória de uma única máquina, o Spark permite distribuir o processamento em um cluster. Ele oferece APIs para Python (PySpark) que são muito semelhantes às do Pandas, facilitando a transição. Funções de transformação e limpeza podem ser aplicadas a DataFrames distribuídos.

Exemplo de Fluxo de Processamento com Python (Conceitual)

Python

```
import pandas as pd
import re
import spacy

# Carregar modelo spaCy para português
try:
    nlp = spacy.load("pt_core_news_sm")
except OSError:
    print("Baixando modelo pt_core_news_sm para spaCy...")
    spacy.cli.download("pt_core_news_sm")
    nlp = spacy.load("pt_core_news_sm")

def limpar_texto(texto):
    if pd.isna(texto): return ""
    texto = str(texto).lower() # Converter para minúsculas
    texto = re.sub(r'[^a-z0-9\s]', '', texto) # Remover caracteres especiais
    texto = re.sub(r'\s+', ' ', texto).strip() # Remover espaços extras
    return texto

def extrair_habilidades(descricao):
    doc = nlp(descricao)
    habilidades = []
    # Exemplo simplificado: identificar entidades que podem ser habilidades
    # Em um cenário real, seria necessário um dicionário de habilidades ou
um modelo mais sofisticado
    for ent in doc.ents:
        if ent.label_ in ["SKILL", "PRODUCT", "LANGUAGE"]:# Exemplo de
labels, ajustar conforme o modelo
            habilidades.append(ent.text)
    return list(set(habilidades))
```

```
def normalizar_salario(salario_str):
    if pd.isna(salario_str): return None, None
    salario_str = salario_str.lower().replace("r$", "").replace(".",
"").replace(",", ".").strip()
    min_salario, max_salario = None, None

    # Exemplo simplificado de extração de faixas salariais
    match = re.search(r'(\d+\.?\d*)\s*a\s*(\d+\.?\d*)', salario_str)
    if match:
        min_salario = float(match.group(1))
        max_salario = float(match.group(2))
    else:
        match = re.search(r'(\d+\.?\d*)', salario_str)
        if match:
            min_salario = float(match.group(1))
            max_salario = float(match.group(1))

    return min_salario, max_salario

# Exemplo de DataFrame (simulando dados brutos de vagas)
data = {
    'id_vaga': [1, 2, 3],
    'titulo': ['Analista de Dados Sênior', 'Desenvolvedor Python Jr.',
'Estágio em Marketing Digital'],
    'descricao': [
        'Buscamos analista com experiência em SQL, Python e Tableau.
Conhecimento em BigQuery é um diferencial.',
        'Vaga para desenvolvedor Python. Experiência com Flask e AWS.
Conhecimento em Docker.',
        'Estágio na área de marketing. Conhecimento em SEO, Google Analytics
e redes sociais.'
    ],
    'salario': ['R$ 5.000 - 8.000', 'A combinar', 'R$ 1.500,00']
}
df = pd.DataFrame(data)

# Aplicar limpeza e transformação
df['descricao_limpa'] = df['descricao'].apply(limpar_texto)
df['habilidades'] = df['descricao'].apply(extrair_habilidades)
df[['salario_min', 'salario_max']] = df['salario'].apply(lambda x:
pd.Series(normalizar_salario(x)))

print(df[['titulo', 'descricao_limpa', 'habilidades', 'salario_min',
'salario_max']].to_markdown(index=False))
```

Este exemplo ilustra como Python pode ser usado para limpar texto, extrair habilidades e normalizar salários. Em um ambiente de Big Data, essas funções seriam aplicadas a

DataFrames do PySpark, aproveitando o processamento distribuído. A complexidade das transformações pode variar, mas a flexibilidade do Python permite implementar lógicas sofisticadas para extrair o máximo valor dos dados de vagas.

5. Implementar Análise e Modelagem de Dados

Com os dados limpos, transformados e armazenados, a próxima etapa crucial é a implementação de análises e modelos que extraiam insights valiosos. Esta fase é onde o valor real do projeto de Big Data é gerado, permitindo entender tendências, prever comportamentos e apoiar decisões estratégicas. Python, com suas bibliotecas robustas para ciência de dados e machine learning, é a linguagem de escolha para esta etapa.

Tipos de Análise e Modelagem

1. **Análise Exploratória de Dados (EDA - Exploratory Data Analysis)**:
    - **Objetivo**: Entender a estrutura dos dados, identificar padrões, anomalias, relações entre variáveis e validar suposições. É a base para qualquer modelagem.
    - **Técnicas**: Estatísticas descritivas (média, mediana, desvio padrão), visualizações (histogramas, box plots, gráficos de dispersão, mapas de calor para correlações), agrupamento (clustering) para identificar segmentos de vagas ou perfis de empresas.
    - **Ferramentas Python**: `Pandas` para manipulação de dados, `Matplotlib`, `Seaborn` e `Plotly` para visualizações estáticas e interativas. `Jupyter Notebooks` ou `JupyterLab` são ambientes ideais para EDA, permitindo um fluxo de trabalho iterativo e documentado.
2. **Análise de Tendências e Padrões**:
    - **Tendências Temporais**: Analisar a evolução do número de vagas por setor, cargo, localização ao longo do tempo. Identificar sazonalidades, picos de demanda e declínios. Isso pode ser feito com dados do CAGED/RAIS (agregados) e, se disponíveis, dados de vagas individuais.
    - **Padrões Geográficos**: Mapear a distribuição de vagas por região, estado, cidade. Identificar polos de emprego e áreas com maior demanda por certas habilidades. Ferramentas como `Folium` ou `Geopandas` em Python podem ser usadas para criar mapas interativos.
    - **Análise de Habilidades**: Identificar as habilidades mais demandadas por setor, cargo ou nível de experiência. Analisar a co-ocorrência de habilidades para entender conjuntos de competências valorizados no mercado.
    - **Ferramentas Python**: `Pandas` para agregação e pivoteamento de dados,

`Matplotlib` / `Seaborn` para gráficos de linha, barras e mapas.

3. **Modelagem Preditiva (Machine Learning)**:
    - **Previsão de Demanda por Vagas**: Construir modelos de séries temporais (ex: ARIMA, Prophet) para prever a demanda futura por vagas em setores específicos ou para cargos específicos. Isso pode ajudar empresas a planejar recrutamento e instituições de ensino a ajustar currículos.
    - **Classificação de Vagas/Candidatos**: Desenvolver modelos para classificar automaticamente vagas em categorias (ex: setor, nível de experiência) ou para classificar candidatos com base em seu perfil e requisitos da vaga. Algoritmos como Regressão Logística, SVM, Random Forest, Gradient Boosting (`XGBoost` , `LightGBM`) são comuns.
    - **Recomendação de Vagas/Candidatos**: Criar sistemas de recomendação que sugerem vagas para candidatos com base em seu perfil e histórico, ou sugerem candidatos para recrutadores com base nos requisitos da vaga. Técnicas de filtragem colaborativa ou baseadas em conteúdo podem ser aplicadas.
    - **Extração de Tópicos (Topic Modeling)**: Usar técnicas de NLP como Latent Dirichlet Allocation (LDA) para identificar tópicos predominantes nas descrições das vagas, revelando tendências de mercado e novas áreas de atuação.
    - **Ferramentas Python**: `Scikit-learn` para uma vasta gama de algoritmos de ML, `TensorFlow` ou `PyTorch` para modelos de Deep Learning (especialmente para NLP em textos de vagas), `Statsmodels` para modelos estatísticos e séries temporais. `Gensim` para modelagem de tópicos.

Processo de Modelagem de Dados

O processo de modelagem de dados em um ambiente de Big Data geralmente segue estas etapas:

1. **Definição do Problema**: Claramente definir o que se deseja prever ou analisar (ex:

prever a demanda por cientistas de dados nos próximos 6 meses). 2. **Coleta e Preparação de Dados**: Utilizar os dados limpos e transformados da fase anterior. Isso inclui seleção de features, engenharia de features e divisão dos dados em conjuntos de treino, validação e teste. 3. **Seleção e Treinamento do Modelo**: Escolher o algoritmo de ML mais adequado para o problema e treinar o modelo com os dados de treino. 4. **Avaliação do Modelo**: Avaliar o desempenho do modelo usando métricas apropriadas (ex: acurácia, precisão, recall, F1-score para classificação; RMSE, MAE para regressão). 5. **Otimização e Ajuste de Hiperparâmetros**: Refinar o modelo para melhorar seu desempenho. 6. **Implantação (Deployment)**: Integrar o modelo treinado em um ambiente de produção, onde ele pode fazer previsões ou classificações em novos dados. Isso pode envolver a criação de APIs com

Flask ou FastAPI para servir o modelo. 7. **Monitoramento e Manutenção**: Monitorar continuamente o desempenho do modelo em produção e retreiná-lo periodicamente com novos dados para garantir sua relevância e precisão.

Considerações para Big Data

Em um ambiente de Big Data, a modelagem pode exigir ferramentas distribuídas:

- **PySpark MLlib**: Para algoritmos de Machine Learning escaláveis que rodam em clusters Spark. Isso é essencial quando os datasets são muito grandes para caber na memória de uma única máquina.
- **Cloud ML Platforms**: Plataformas como Google AI Platform, AWS SageMaker ou Azure Machine Learning oferecem serviços gerenciados para construir, treinar e implantar modelos de ML em escala, abstraindo a complexidade da infraestrutura subjacente.

Ao seguir estas diretrizes, o projeto poderá extrair insights profundos e construir modelos preditivos robustos a partir dos dados de vagas de emprego, transformando dados em conhecimento acionável.

6. Planejar Visualização e Relatórios

A visualização e a geração de relatórios são as etapas finais do pipeline de dados, onde os insights extraídos são comunicados de forma clara e eficaz aos stakeholders. Um bom planejamento nesta fase garante que as informações sejam acessíveis, compreensíveis e acionáveis, transformando dados complexos em narrativas visuais impactantes.

Princípios de Visualização de Dados

1. **Clareza e Simplicidade**: As visualizações devem ser fáceis de entender, sem sobrecarga de informações. O objetivo é transmitir uma mensagem clara rapidamente.
2. **Relevância**: Cada gráfico ou relatório deve responder a uma pergunta de negócio específica ou destacar um insight importante.
3. **Precisão**: As visualizações devem representar os dados de forma precisa, evitando distorções ou interpretações errôneas.
4. **Interatividade (Opcional)**: Para análises mais aprofundadas, a interatividade permite que os usuários explorem os dados por conta própria, filtrando e detalhando informações.
5. **Consistência**: Usar cores, fontes e layouts consistentes em todos os relatórios e dashboards para facilitar a leitura e a compreensão.

Tipos de Visualizações e Relatórios para Vagas de Emprego

Considerando os dados de vagas de emprego, algumas visualizações e relatórios chave podem incluir:

- **Tendências de Vagas por Setor/Cargo**: Gráficos de linha mostrando a evolução do número de vagas ao longo do tempo para diferentes setores ou cargos. Isso pode ser complementado com gráficos de área para mostrar a proporção de cada setor no total de vagas.
- **Top N Habilidades/Tecnologias Demandadas**: Gráficos de barras ou nuvens de palavras (word clouds) para destacar as habilidades e tecnologias mais frequentemente mencionadas nas descrições das vagas.
- **Distribuição Geográfica de Vagas**: Mapas de calor ou coropléticos (choropleth maps) mostrando a concentração de vagas por estado, cidade ou região. Isso pode ser combinado com filtros para setor ou cargo.
- **Salário Médio por Cargo/Setor/Experiência**: Gráficos de barras agrupadas ou box plots para comparar faixas salariais em diferentes categorias.
- **Análise de Sentimento em Descrições de Vagas (se aplicável)**: Gráficos de pizza ou barras para mostrar a proporção de vagas com descrições positivas, neutras ou negativas (se um modelo de sentimento for implementado).
- **Painéis de Controle (Dashboards)**: Combinar várias visualizações interativas em um único painel para fornecer uma visão abrangente do mercado de trabalho, permitindo que os usuários filtrem por data, localização, setor, etc.

Ferramentas para Visualização e Relatórios

1. **Bibliotecas Python para Visualização**: Para análises exploratórias e prototipagem de visualizações, as seguintes bibliotecas são excelentes:
    - **Matplotlib**: A biblioteca fundamental para gráficos estáticos em Python. Oferece controle granular sobre todos os elementos do gráfico.
    - **Seaborn**: Construída sobre Matplotlib, oferece uma interface de alto nível para criar gráficos estatísticos atraentes e informativos com menos código.
    - **Plotly / Bokeh**: Para visualizações interativas que podem ser incorporadas em aplicações web ou Jupyter Notebooks. Ideais para dashboards e exploração de dados.
    - **Folium / Geopandas**: Para criar mapas interativos, especialmente úteis para visualizar a distribuição geográfica de vagas.
2. **Ferramentas de Business Intelligence (BI)**: Para dashboards e relatórios de produção, ferramentas de BI são mais adequadas, pois oferecem interfaces amigáveis para usuários de negócio e recursos avançados de compartilhamento e governança:

- **Tableau**: Uma das ferramentas de BI mais populares, conhecida por suas capacidades de visualização e facilidade de uso.
- **Power BI**: Solução da Microsoft, bem integrada com o ecossistema Microsoft e com forte capacidade de modelagem de dados.
- **Looker Studio (antigo Google Data Studio)**: Uma opção gratuita e baseada em nuvem do Google, ideal para integração com fontes de dados do Google Cloud (BigQuery, Google Sheets).
- **Metabase / Superset**: Ferramentas de BI open-source que podem ser auto-hospedadas, oferecendo flexibilidade e controle.

3. **Desenvolvimento Web Customizado**: Para requisitos de visualização muito específicos ou integração profunda com aplicações existentes, pode-se desenvolver dashboards customizados usando frameworks web como Flask ou Django, combinados com bibliotecas de visualização Python (Plotly Dash, Streamlit) ou bibliotecas JavaScript (D3.js, React com bibliotecas de gráficos).

Processo de Geração de Relatórios

1. **Identificação de KPIs (Key Performance Indicators)**: Definir as métricas mais importantes que precisam ser monitoradas e reportadas (ex: número de vagas abertas, tempo médio para preenchimento de vaga, habilidades mais procuradas).
2. **Design do Relatório/Dashboard**: Esboçar o layout e o tipo de visualizações que serão usadas para cada KPI, considerando o público-alvo.
3. **Desenvolvimento**: Implementar as visualizações e o layout usando as ferramentas escolhidas.
4. **Validação**: Garantir que os relatórios sejam precisos, consistentes e fáceis de interpretar.
5. **Automação**: Configurar a atualização automática dos relatórios e dashboards, seja via agendamento (Airflow) ou conexão direta com as fontes de dados atualizadas.
6. **Distribuição**: Definir como os relatórios serão compartilhados (e-mail, portal web, ferramenta de BI).

Um planejamento cuidadoso da visualização e dos relatórios é essencial para transformar os dados brutos em insights acionáveis, permitindo que o projeto entregue valor real aos seus usuários e stakeholders.

7. Considerações Finais e Boas Práticas

Para garantir o sucesso e a sustentabilidade do projeto de análise de vagas de emprego no Brasil, é fundamental adotar um conjunto de boas práticas e considerar aspectos

importantes que permeiam todas as fases do desenvolvimento.

Desenvolvimento Iterativo e Ágil

Projetos de Big Data são complexos e evoluem rapidamente. Adotar uma metodologia ágil, com ciclos de desenvolvimento curtos (sprints), permite a entrega contínua de valor, a adaptação a novas necessidades e a correção de rotas de forma eficiente. Começar com um protótipo (MVP - Minimum Viable Product) e expandir gradualmente as funcionalidades é uma abordagem recomendada.

Qualidade e Governança de Dados

A qualidade dos dados é a base para qualquer análise ou modelo. Investir em ferramentas e processos para monitorar a qualidade dos dados desde a ingestão até o consumo é crucial. Isso inclui:

- **Validação de Dados**: Implementar regras de validação em cada etapa do pipeline para garantir que os dados estejam nos formatos esperados e dentro dos limites aceitáveis.
- **Linhagem de Dados (Data Lineage)**: Documentar a origem, as transformações e o destino de cada conjunto de dados. Isso é vital para auditoria, depuração e compreensão do fluxo de dados.
- **Metadados**: Manter um catálogo de metadados que descreva os dados (definição de campos, tipos, fontes, frequência de atualização, etc.).
- **Segurança e Privacidade**: Garantir a conformidade com a LGPD (Lei Geral de Proteção de Dados) e outras regulamentações de privacidade. Isso envolve anonimização, criptografia e controle de acesso rigoroso aos dados.

Monitoramento e Alerta

Um pipeline de Big Data é um sistema complexo que requer monitoramento constante. Implementar sistemas de monitoramento e alerta para:

- **Falhas na Ingestão**: Ser notificado imediatamente se um script de web scraping falhar ou se uma API retornar erros.
- **Qualidade de Dados**: Alertar sobre anomalias ou desvios significativos nos dados (ex: queda abrupta no número de vagas coletadas, valores fora do padrão).
- **Performance**: Monitorar o desempenho dos clusters de processamento (Spark) e dos bancos de dados para identificar gargalos e otimizar recursos.
- **Modelos de ML**: Acompanhar o desempenho dos modelos preditivos em produção para detectar degradação (drift) e acionar retreinamentos.

Documentação Abrangente

Documentar todas as etapas do projeto é essencial para a manutenção, escalabilidade e para que novos membros da equipe possam entender e contribuir. A documentação deve incluir:

- **Arquitetura do Sistema**: Diagramas e descrições dos componentes e seus fluxos.
- **Dicionário de Dados**: Detalhes sobre cada campo nos datasets.
- **Código**: Comentários claros, padrões de codificação e READMEs para cada módulo ou script.
- **Processos Operacionais**: Guias para implantação, monitoramento e solução de problemas.

Escalabilidade e Otimização de Custos

- **Design para Escalabilidade**: Desde o início, projetar o sistema para lidar com o crescimento do volume de dados e da complexidade das análises. Isso significa escolher tecnologias e arquiteturas que possam escalar horizontalmente.
- **Otimização de Recursos**: Em ambientes de nuvem, monitorar e otimizar o uso de recursos (CPU, memória, armazenamento) para controlar os custos. Utilizar instâncias sob demanda, reservas ou funções serverless quando apropriado.

Colaboração e Comunicação

O sucesso de um projeto de Big Data depende da colaboração entre diferentes perfis (engenheiros de dados, cientistas de dados, analistas de negócio). Manter canais de comunicação abertos e garantir que todos os stakeholders estejam alinhados com os objetivos e o progresso do projeto é fundamental.

Ao seguir estas considerações e boas práticas, o projeto de análise de vagas de emprego estará bem posicionado para superar os desafios técnicos, entregar insights valiosos e se manter relevante e eficaz a longo prazo.

2. Definir a Arquitetura de Dados e Ferramentas para o ASCENDE

Com base nos requisitos funcionais e não funcionais do projeto ASCENDE, conforme detalhado no documento fornecido e na análise anterior, podemos agora definir uma arquitetura de dados e selecionar as ferramentas mais adequadas. Esta arquitetura será projetada para ser escalável, confiável e eficiente, utilizando as melhores práticas de Big Data e Python.

2.1. Visão Geral da Arquitetura Proposta

A arquitetura do ASCENDE seguirá um padrão de pipeline de dados, com camadas distintas para ingestão, armazenamento, processamento, análise e visualização. A ênfase será na utilização de tecnologias de código aberto e serviços de nuvem (assumindo uma plataforma de nuvem como GCP, AWS ou Azure para escalabilidade e gerenciamento simplificado, embora possa ser adaptada para on-premise).

mermaid

Failed to render mermaid

2.2. Detalhamento dos Componentes e Ferramentas

2.2.1. Fontes de Dados

- **Portais de Emprego**: LinkedIn, Gupy, Indeed, CIEE. Estas serão as fontes primárias para vagas individuais de tecnologia em Fortaleza. A coleta será via web scraping, devido à ausência de APIs públicas para consumo em massa.
- **Dados Governamentais**: CAGED, RAIS (Ministério do Trabalho e Emprego) e PNAD Contínua (IBGE). Essenciais para contextualizar o mercado de trabalho, identificar tendências macro e validar insights obtidos das vagas individuais. Serão utilizados para enriquecer a análise, fornecendo dados agregados sobre setores, ocupações e demografia do emprego formal e informal.

2.2.2. Camada de Ingestão

- **Web Scrapers Python**: Desenvolvidos especificamente para cada portal de emprego. Utilizarão bibliotecas como `Scrapy` para estruturar a coleta, `requests` para requisições HTTP, `BeautifulSoup` para parsing HTML e, se necessário, `Selenium` ou `Playwright` para

interagir com páginas dinâmicas. Serão configurados para filtrar por localização (Fortaleza, Ceará) sempre que possível na fonte.

- **Scripts Python de Download**: Para automatizar a obtenção de arquivos (CSV, TXT, XLSX) dos portais do MTE e IBGE. Utilizarão `requests` e `BeautifulSoup` para navegar e baixar os arquivos.
- **Orquestração de Pipeline (Apache Airflow / Prefect)**: Ferramentas robustas para agendar, monitorar e gerenciar os fluxos de trabalho (DAGs) de ingestão. Elas garantirão que os web scrapers e scripts de download sejam executados periodicamente, lidem com retries, e notifiquem em caso de falhas. Airflow é uma escolha popular para orquestração de Big Data devido à sua flexibilidade e comunidade.

2.2.3. Camada de Armazenamento Bruto (Data Lake)

- **Tecnologia**: **Amazon S3, Google Cloud Storage (GCS) ou Azure Blob Storage**. A escolha dependerá da plataforma de nuvem preferencial. Estes serviços oferecem armazenamento de objetos altamente escalável, durável e de baixo custo, ideal para armazenar dados brutos em seu formato original (HTML, JSON, CSV, TXT) antes de qualquer transformação.
- **Estrutura**: Os dados serão organizados em um esquema de pastas lógico, por exemplo: `/raw/<fonte>/<ano>/<mes>/<dia>/<timestamp>/<arquivo>`. Isso facilita a governança, a auditoria e o reprocessamento de dados.

2.2.4. Camada de Processamento e Transformação

- **Processamento Distribuído (Apache Spark / PySpark)**: Para lidar com o volume e a velocidade dos dados de vagas, especialmente quando o web scraping estiver em plena operação, o Apache Spark com sua API Python (PySpark) será a ferramenta central. Ele permite processamento distribuído em clusters, garantindo escalabilidade e performance para tarefas como limpeza, extração de entidades e transformações complexas.
- **Scripts Python (Pandas, NLTK, spaCy)**: Embora o Spark seja para escala, bibliotecas como `Pandas` podem ser usadas para prototipagem e para processamento de dados menores ou resultados intermediários. `NLTK` e `spaCy` serão cruciais para o Processamento de Linguagem Natural (NLP) nas descrições das vagas, realizando:
  - **Limpeza de Texto**: Remoção de HTML, caracteres especiais, pontuação, stopwords.
  - **Lematização/Stemming**: Redução de palavras à sua forma base.
  - **Extração de Entidades (NER)**: Identificação de habilidades, tecnologias, ferramentas, cargos e localizações.

- **Categorização de Vagas**: Implementação de lógica baseada em palavras-chave ou modelos de ML para classificar vagas como

estágio, jovem aprendiz ou efetivo (CLT). * **Normalização de Salários**: Extração e padronização de faixas salariais.

2.2.5. Camada de Armazenamento Curado

- **Data Warehouse (Google BigQuery / Amazon Redshift)**: Para armazenar os dados estruturados e limpos das vagas de emprego (título, empresa, localização, categoria, salário, URL, data de publicação, etc.) e os dados agregados do CAGED/RAIS/PNAD. Estes Data Warehouses em nuvem são otimizados para consultas analíticas complexas e de alta performance, sendo ideais para alimentar dashboards e relatórios.
- **Banco de Dados NoSQL (MongoDB / Elasticsearch)**: Pode ser utilizado para armazenar informações mais flexíveis ou semi-estruturadas, como as descrições originais das vagas, as habilidades e tecnologias extraídas (em formato de listas ou arrays), ou para servir como um índice de busca rápida para as vagas. Elasticsearch seria particularmente útil para buscas textuais avançadas nas descrições das vagas.

2.2.6. Camada de Análise e Modelagem

- **Jupyter Notebooks / JupyterLab**: Ambiente interativo para cientistas de dados realizarem Análise Exploratória de Dados (EDA), desenvolverem e testarem modelos de Machine Learning. Permite um fluxo de trabalho iterativo e documentado.
- **Bibliotecas Python de Machine Learning**: `Scikit-learn` para algoritmos de classificação (categorização de vagas, identificação de setores), regressão (previsão de salários), agrupamento (segmentação de vagas). `TensorFlow` ou `PyTorch` para modelos de Deep Learning, especialmente para tarefas de NLP mais avançadas (ex: embeddings de texto para similaridade de vagas).
- **PySpark MLlib**: Para aplicar algoritmos de Machine Learning em larga escala diretamente nos DataFrames do Spark, aproveitando o processamento distribuído.

2.2.7. Camada de Visualização e Relatórios

- **Ferramentas de Business Intelligence (BI)**: **Looker Studio (Google Data Studio)**, **Tableau** ou **Power BI**. Estas ferramentas se conectarão diretamente ao Data Warehouse (BigQuery/Redshift) para criar dashboards interativos e relatórios que apresentarão os insights do projeto ASCENDE. Elas permitirão que os usuários finais (estudantes, empresas, instituições de ensino) explorem os dados com filtros por categoria de vaga, habilidades, localização, etc.

- **Aplicação Web Customizada (Flask/Django + Plotly Dash/Streamlit)**: Para uma experiência mais integrada e customizada, pode-se desenvolver um site próprio utilizando frameworks web Python como `Flask` ou `Django`, e bibliotecas como `Plotly Dash` ou `Streamlit` para criar dashboards e visualizações interativas diretamente na aplicação. Isso oferece maior controle sobre a interface do usuário e a integração com outras funcionalidades do site.

2.3. Considerações de Infraestrutura

- **Plataforma de Nuvem**: Recomenda-se a utilização de uma plataforma de nuvem (GCP, AWS ou Azure) para hospedar os componentes da arquitetura. Isso oferece escalabilidade, alta disponibilidade, segurança e serviços gerenciados que simplificam a operação.
- **Contêineres (Docker)**: Empacotar os web scrapers, scripts de processamento e até mesmo a aplicação web em contêineres Docker facilita a implantação, o gerenciamento de dependências e a portabilidade entre ambientes.
- **Orquestração de Contêineres (Kubernetes)**: Para ambientes de produção com múltiplos serviços e alta demanda, Kubernetes pode ser usado para orquestrar os contêineres, garantindo escalabilidade e resiliência.

Esta arquitetura fornece um roteiro robusto para construir o projeto ASCENDE, abordando os requisitos funcionais e não funcionais com tecnologias de Big Data e Python. A flexibilidade da nuvem e das ferramentas de código aberto permitirá que o sistema evolua e se adapte às necessidades futuras do mercado de trabalho de tecnologia em Fortaleza.

3. Planejar a Estratégia de Aquisição de Dados para o ASCENDE (Web Scraping e Fontes Governamentais)

A aquisição de dados é a base do projeto ASCENDE. Dada a natureza das fontes (portais de emprego comerciais e dados governamentais), a estratégia deve ser robusta, legalmente consciente e tecnicamente eficiente. Esta seção detalha como os dados serão coletados, considerando os desafios e as melhores práticas.

3.1. Aquisição de Dados de Vagas Individuais (Web Scraping)

Conforme identificado nos requisitos, a principal forma de obter dados de vagas individuais de plataformas como LinkedIn, Gupy, Indeed e CIEE será através de web scraping. Esta abordagem exige um planejamento cuidadoso para garantir a eficácia e a conformidade.

3.1.1. Ferramentas e Tecnologias

- **Scrapy**: Framework Python de alto nível para web scraping e rastreamento de sites. É ideal para projetos complexos e escaláveis, oferecendo funcionalidades como gerenciamento de requisições, parsing de HTML/XML, pipelines de itens e middlewares para lidar com cookies, sessões e User-Agents. [1]
- **Requests**: Biblioteca HTTP para Python, utilizada para fazer requisições web. Pode ser usada em conjunto com Scrapy ou de forma independente para requisições mais simples. [2]
- **BeautifulSoup**: Biblioteca Python para parsing de HTML e XML. Facilita a navegação e busca de elementos na árvore DOM. Geralmente usada com `requests` para scraping mais leve. [3]
- **Selenium / Playwright**: Ferramentas de automação de navegador. Essenciais para sites que dependem fortemente de JavaScript para carregar conteúdo dinamicamente ou que possuem proteções anti-bot que exigem a simulação de um navegador real. [4], [5]
- **Proxies e VPNs**: Para rotacionar endereços IP e evitar bloqueios. Serviços de proxy pagos oferecem maior confiabilidade e variedade de IPs.
- **User-Agents Rotativos**: Para simular diferentes navegadores e sistemas operacionais, dificultando a detecção como bot.

3.1.2. Estratégia por Plataforma (Exemplo)

- **LinkedIn Jobs**: É um dos portais mais desafiadores para scraping devido às suas robustas proteções anti-bot. Pode exigir o uso de Selenium/Playwright e uma estratégia agressiva de rotação de IPs e User-Agents. A busca por vagas em Fortaleza deve ser configurada diretamente na URL de busca, se possível. A extração de detalhes da vaga (título, descrição, empresa, localização, requisitos, data de publicação) será feita a partir das páginas de detalhe.
- **Gupy**: Muitas empresas utilizam a Gupy para gerenciar suas vagas. A Gupy geralmente tem URLs padronizadas para páginas de carreira de empresas. O scraping pode focar em identificar essas páginas e, em seguida, extrair as vagas listadas. Pode ser menos complexo que o LinkedIn, mas ainda exige cuidado.
- **Indeed**: Oferece uma interface de busca relativamente amigável para scraping, mas também implementa proteções. A filtragem por localização (Fortaleza) e termos de busca pode ser feita via parâmetros de URL. A extração de dados é similar aos outros portais.
- **CIEE**: Focado em estágio e jovem aprendiz. A estrutura pode ser mais simples, mas a necessidade de login ou formulários específicos pode complicar o scraping. Selenium pode ser útil aqui.

3.1.3. Boas Práticas e Considerações Legais para Web Scraping

- **Termos de Serviço (ToS)**: **É imperativo ler e entender os Termos de Serviço de cada plataforma antes de iniciar o scraping.** Muitos ToS proíbem explicitamente o scraping automatizado. A violação pode resultar em bloqueio de IP, ações legais ou outras penalidades. Se o scraping for proibido, a alternativa seria buscar parcerias ou APIs pagas.
- **`robots.txt`**: Respeitar o arquivo `robots.txt` de cada site. Este arquivo indica quais partes do site podem ou não ser rastreadas por bots. [6]
- **Frequência e Atrasos**: Implementar atrasos (`time.sleep()` em Python) entre as requisições para não sobrecarregar os servidores dos sites. Um atraso aleatório (ex: entre 5 e 15 segundos) é melhor do que um fixo. Evitar picos de requisições.
- **Identificação**: Usar User-Agents que simulem navegadores reais. Evitar se identificar como um bot de scraping, a menos que explicitamente permitido.
- **Volume de Dados**: Começar com um volume pequeno de dados para testes e aumentar gradualmente. Monitorar o impacto no site alvo.
- **Armazenamento de Dados Brutos**: Os dados coletados (HTML das páginas, JSON de APIs) devem ser armazenados no Data Lake (S3/GCS) em seu formato bruto, para permitir reprocessamento ou auditoria futura.

3.2. Aquisição de Dados Governamentais (CAGED, RAIS, PNAD Contínua)

Essas fontes fornecem dados agregados e microdados cruciais para contextualizar o mercado de trabalho e identificar tendências macro.

3.2.1. Ferramentas e Tecnologias

- **Requests**: Para baixar arquivos diretamente de URLs. [2]
- **BeautifulSoup**: Para navegar em páginas web e extrair links de download, caso não sejam diretos. [3]
- **Pandas**: Para ler e processar arquivos CSV, XLSX, TXT após o download. [7]

3.2.2. Estratégia de Aquisição

- **CAGED**: Os dados mensais e anuais são disponibilizados no portal do Ministério do Trabalho e Emprego. Scripts Python serão desenvolvidos para:
  - Navegar até a seção de download do CAGED.
  - Identificar os links para os arquivos mais recentes (por mês/ano).

    - Baixar os arquivos (geralmente CSV ou TXT compactados).
    - Descompactar e mover para o Data Lake.
- **RAIS**: Similar ao CAGED, os dados anuais da RAIS são disponibilizados no mesmo portal. O processo de download e armazenamento será análogo.
- **PNAD Contínua (IBGE)**: Os microdados da PNAD Contínua são disponibilizados no site do IBGE. O processo envolve:
    - Navegar até a seção de microdados da PNAD Contínua.
    - Identificar os links para os arquivos de dados e dicionários de variáveis mais recentes.
    - Baixar os arquivos (geralmente TXT ou CSV compactados).
    - Descompactar e mover para o Data Lake.
    - **Importante**: Os dicionários de variáveis são cruciais para interpretar os microdados e devem ser baixados e armazenados junto com os dados.

3.2.3. Orquestração da Aquisição de Dados

- **Apache Airflow / Prefect**: A orquestração é fundamental para automatizar a execução periódica dos scripts de aquisição de dados. Serão criados DAGs (Directed Acyclic Graphs) para:
    - **Web Scraping**: DAGs separados para cada portal de emprego, executados diariamente ou semanalmente.
    - **Dados Governamentais**: DAGs para CAGED (mensal), RAIS (anual) e PNAD Contínua (trimestral/anual).
    - **Monitoramento e Alerta**: Configurar alertas (e-mail, Slack) para notificar a equipe em caso de falhas na coleta de dados.
    - **Retries**: Implementar lógica de retry para lidar com falhas temporárias de rede ou de acesso a sites.

3.3. Armazenamento Inicial dos Dados Brutos

Todos os dados coletados, sejam de web scraping ou downloads governamentais, serão primeiramente armazenados no **Data Lake (Amazon S3 / GCS / Azure Blob Storage)**. Isso garante que tenhamos uma cópia fiel dos dados originais, permitindo reprocessamento e auditoria. A estrutura de pastas será organizada para facilitar a identificação da fonte, data e tipo de dado.

Referências:

[1] Scrapy. Disponível em: https://scrapy.org/ [2] Requests: HTTP for Humans. Disponível em: https://requests.readthedocs.io/en/latest/ [3] Beautiful Soup Documentation. Disponível em: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ [4] Selenium. Disponível em: https://www.selenium.dev/ [5] Playwright. Disponível em: https://playwright.dev/ [6] The Web Robots Pages. Disponível em: https://www.robotstxt.org/robotstxt.html [7] Pandas. Disponível em: https://pandas.pydata.org/

4. Desenvolver o Processamento e Transformação de Dados com Python para o ASCENDE

Com os dados brutos coletados e armazenados no Data Lake, a próxima etapa crítica é transformá-los em informações valiosas e estruturadas que possam ser utilizadas para análise e modelagem. Esta fase é o coração do pipeline de dados, onde Python, com seu ecossistema robusto de bibliotecas, desempenha um papel central. O objetivo é garantir que os dados sejam limpos, consistentes, enriquecidos e prontos para gerar os insights que o projeto ASCENDE propõe.

4.1. Etapas Essenciais de Processamento e Transformação

O processo de transformação de dados para o ASCENDE envolverá várias sub-etapas, cada uma com um propósito específico para refinar os dados brutos.

4.1.1. Limpeza de Dados (Data Cleaning)

A limpeza de dados é a primeira linha de defesa contra a "sujeira" inerente aos dados brutos, especialmente aqueles provenientes de web scraping. Dados inconsistentes, incompletos ou incorretos podem levar a análises falhas e insights enganosos. As principais atividades incluem:

- **Remoção de Duplicatas**: Vagas podem ser coletadas múltiplas vezes de uma mesma fonte ou de fontes diferentes. É crucial identificar e remover registros duplicados com base em critérios como URL da vaga, título e empresa. Bibliotecas como `pandas` oferecem métodos eficientes como `df.drop_duplicates()` para essa finalidade.
- **Tratamento de Valores Ausentes (Missing Values)**: Campos como salário, nível de experiência ou até mesmo partes da descrição podem estar ausentes. As estratégias incluem:
  - **Remoção**: Descartar registros com valores ausentes em campos críticos (ex: título da vaga).
  - **Preenchimento**: Substituir valores ausentes por um valor padrão (ex: "Não Informado"), a média/mediana (para campos numéricos), ou por inferência a partir

de outros campos.
    - **Imputação**: Utilizar modelos mais sofisticados para prever valores ausentes, embora isso possa ser um passo mais avançado.
- **Padronização de Formatos**: Garantir que todos os dados sigam um formato consistente:
    - **Datas**: Converter todas as datas de publicação para um formato uniforme (ex: `YYYY-MM-DD`).
    - **Textos**: Converter todo o texto para minúsculas, remover espaços em branco extras (`strip()`), e lidar com quebras de linha e tabulações. Isso é vital para análises de texto subsequentes.
    - **Numéricos**: Assegurar que campos como salários ou contagens sejam tratados como tipos numéricos, e não como strings.
- **Remoção de Caracteres Especiais e HTML**: Descrições de vagas frequentemente contêm tags HTML, entidades HTML (ex: `&`), e caracteres especiais que precisam ser removidos ou normalizados para que o texto seja limpo para NLP. Expressões regulares (`re` em Python) e bibliotecas como `BeautifulSoup` (para remover tags HTML) são ferramentas essenciais aqui.

4.1.2. Enriquecimento de Dados (Data Enrichment)

O enriquecimento de dados visa adicionar valor aos dados brutos, extraindo informações implícitas ou combinando-os com outras fontes para criar novos atributos. Para o ASCENDE, isso é fundamental para atender aos requisitos de identificação de habilidades, tecnologias e categorização de vagas.

- **Extração de Entidades (Named Entity Recognition - NER)**: Utilizar técnicas de Processamento de Linguagem Natural (NLP) para identificar e extrair entidades relevantes das descrições das vagas. Para o ASCENDE, isso significa extrair:
    - **Habilidades Técnicas**: Python, SQL, Java, React, AWS, Azure, Docker, Kubernetes, etc.
    - **Habilidades Comportamentais (Soft Skills)**: Comunicação, Liderança, Trabalho em Equipe, Resolução de Problemas.
    - **Ferramentas e Plataformas**: Tableau, Power BI, Jira, Git, Figma.
    - **Tecnologias Específicas**: Machine Learning, Inteligência Artificial, Big Data, Cloud Computing.
    - **Localização**: Refinar a extração de cidades, estados e bairros, se necessário.
    - **Bibliotecas**: `spaCy` e `NLTK` são as principais bibliotecas Python para NER. Pode ser necessário treinar modelos customizados ou usar dicionários extensos de termos

para melhorar a precisão da extração.

- **Classificação de Setores/Áreas**: Embora o foco seja tecnologia, as vagas podem ter nuances. Pode-se classificar a vaga em sub-áreas de tecnologia (ex: Desenvolvimento Web, Data Science, DevOps, Segurança da Informação) com base em palavras-chave ou modelos de ML.
- **Normalização de Salários**: Vagas podem apresentar salários de diversas formas (ex: "R$ 3.000-5.000", "5k", "A combinar", "Salário competitivo"). Esta etapa envolve:
  - **Extração**: Usar expressões regulares para extrair valores numéricos e faixas salariais.
  - **Padronização**: Converter todos os valores para uma unidade monetária e escala padrão (ex: BRL, mensal). Converter "5k" para "5000".
  - **Inferência**: Para vagas com "A combinar" ou "Salário competitivo", pode-se tentar inferir uma faixa salarial com base em cargos e habilidades similares, usando modelos preditivos.
- **Enriquecimento Geográfico**: Se a localização for muito genérica (ex: "Fortaleza"), pode-se usar APIs de geocodificação (ex: Google Maps API, OpenStreetMap Nominatim via `geopy`) para obter coordenadas geográficas (latitude, longitude) e até mesmo informações de bairros ou regiões administrativas.
- **Adição de Metadados**: Incluir metadados como a data de coleta, a fonte original da vaga, e um identificador único gerado pelo sistema.

4.1.3. Padronização e Categorização (Standardization & Categorization)

Esta etapa visa criar consistência e facilitar a análise ao agrupar informações semelhantes.

- **Categorização de Vagas (Estágio, Jovem Aprendiz, Efetivo/CLT)**: Conforme requisito do ASCENDE, esta categorização será feita com base em regras de palavras-chave no título e/ou descrição da vaga. Um conjunto de regras bem definido e testado será implementado. Ex: se "estágio" ou "intern" no título -> Estágio; se "jovem aprendiz" -> Jovem Aprendiz; caso contrário, e se houver termos como "CLT", "efetivo" -> Efetivo.
- **Normalização de Habilidades e Tecnologias**: Após a extração, é comum ter variações (ex: "Python", "python", "Py"). É necessário normalizar esses termos para uma representação única (ex: sempre "Python"). Isso pode ser feito com um dicionário de sinônimos ou técnicas de agrupamento de strings.
- **Padronização de Títulos de Cargo**: Criar um conjunto limitado de títulos de cargo padronizados (ex: "Desenvolvedor Frontend", "Cientista de Dados", "Engenheiro de DevOps") e mapear os títulos originais das vagas para esses padrões. Isso facilita a análise comparativa entre cargos.

4.1.4. Agregação de Dados (Data Aggregation)

Após a limpeza e enriquecimento, os dados podem ser agregados para gerar métricas e resumos que alimentam as visualizações e análises.

- **Contagem de Vagas**: Total de vagas por categoria (estágio, jovem aprendiz, CLT), por setor, por localização, por empresa.
- **Frequência de Habilidades**: Contagem de quantas vezes cada habilidade/tecnologia aparece nas vagas.
- **Salário Médio/Mediana**: Calcular o salário médio, mediano, mínimo e máximo por cargo, setor, nível de experiência.
- **Tendências Temporais**: Agregação de dados por dia, semana ou mês para analisar a evolução das métricas ao longo do tempo.

4.2. Ferramentas Python para Processamento e Transformação

- **Pandas**: A biblioteca fundamental para manipulação e análise de dados tabulares em Python. Ideal para operações de limpeza, transformação e agregação em datasets que cabem na memória de uma única máquina. Será amplamente utilizada para prototipagem e para processamento de dados governamentais e de vagas individuais, caso o volume não exija processamento distribuído.
- **NumPy**: Base para operações numéricas de alta performance, frequentemente usado em conjunto com Pandas.
- **Scikit-learn**: Para tarefas de Machine Learning, como classificação de texto (para categorização de vagas ou setores), agrupamento (clustering) de vagas similares, ou para engenharia de features.
- **NLTK (Natural Language Toolkit) / spaCy**: Bibliotecas essenciais para Processamento de Linguagem Natural (NLP). `spaCy` é geralmente preferido para tarefas de produção devido à sua velocidade e modelos pré-treinados eficientes para NER e outras tarefas. `NLTK` oferece uma gama mais ampla de algoritmos e recursos para pesquisa e desenvolvimento.
- **Regex (módulo `re`)**: Para extração de padrões específicos de texto, como valores salariais, datas, códigos, ou para limpeza de strings.
- **Apache Spark (PySpark)**: Para processamento de Big Data em larga escala. Quando os volumes de dados excederem a capacidade de memória de uma única máquina, o PySpark será indispensável. Ele permite aplicar as mesmas lógicas de limpeza, enriquecimento e transformação de forma distribuída em um cluster, garantindo escalabilidade e performance. As APIs de DataFrame do PySpark são muito semelhantes às do Pandas, facilitando a transição.

4.3. Exemplo de Fluxo de Processamento com Python (Conceitual para ASCENDE)

Python

```
import pandas as pd
import re
import spacy
from collections import Counter

# Carregar modelo spaCy para português
try:
    nlp = spacy.load("pt_core_news_sm")
except OSError:
    print("Baixando modelo pt_core_news_sm para spaCy...")
    spacy.cli.download("pt_core_news_sm")
    nlp = spacy.load("pt_core_news_sm")

# Dicionário de habilidades e tecnologias (exemplo simplificado)
habilidades_tecnicas = [
    "python", "java", "javascript", "sql", "aws", "azure", "gcp", "docker",
    "kubernetes", "react", "angular", "vue", "node.js", "spring boot",
    "machine learning", "inteligencia artificial", "big data", "devops",
    "git", "jira", "tableau", "power bi", "excel", "linux", "cloud"
]

def limpar_texto(texto):
    if pd.isna(texto): return ""
    texto = str(texto).lower() # Converter para minúsculas
    texto = re.sub(r'<.*?>', '', texto) # Remover tags HTML
    texto = re.sub(r'&[a-z]+;', '', texto) # Remover entidades HTML
    texto = re.sub(r'[^a-z0-9áéíóúâêîôûãõç\s]', '', texto) # Remover
caracteres especiais, manter acentos
    texto = re.sub(r'\s+', ' ', texto).strip() # Remover espaços extras
    return texto

def categorizar_vaga(titulo, descricao):
    titulo_lower = titulo.lower() if pd.notna(titulo) else ""
    descricao_lower = descricao.lower() if pd.notna(descricao) else ""

    if "estágio" in titulo_lower or "intern" in titulo_lower or "estagiário"
in descricao_lower:
        return "Estágio"
    if "jovem aprendiz" in titulo_lower or "aprendiz" in titulo_lower or
"jovem aprendiz" in descricao_lower:
        return "Jovem Aprendiz"
    if "clt" in titulo_lower or "efetivo" in titulo_lower or "full-time" in
```

```
descricao_lower:
        return "Efetivo (CLT)"
    # Lógica adicional pode ser implementada aqui para inferir CLT se não
houver termos explícitos
    return "Não Categorizado"

def extrair_e_normalizar_habilidades(descricao_limpa):
    doc = nlp(descricao_limpa)
    habilidades_encontradas = []
    for habilidade_padrao in habilidades_tecnicas:
        if habilidade_padrao in descricao_limpa:
            habilidades_encontradas.append(habilidade_padrao)

    # Usar NER do spaCy para extrair entidades que podem ser habilidades
    # Isso requer um modelo treinado para identificar habilidades de forma
mais robusta
    # Exemplo simplificado: apenas para ilustrar a ideia
    for ent in doc.ents:
        # Em um cenário real, você teria uma lista de labels relevantes ou
um modelo customizado
        if ent.label_ in ["ORG", "PRODUCT", "LANGUAGE", "SKILL"] and
ent.text.lower() not in habilidades_encontradas:
            # Filtrar por termos relevantes e evitar duplicatas
            if len(ent.text) > 2 and ent.text.lower() not in ["e", "o", "a",
"de", "para"]:
                habilidades_encontradas.append(ent.text.lower())

    return list(set(habilidades_encontradas)) # Remover duplicatas e
retornar lista

def normalizar_salario(salario_str):
    if pd.isna(salario_str): return None, None
    salario_str = salario_str.lower().replace("r$", "").replace(".",
"").replace(",", ".").strip()
    min_salario, max_salario = None, None

    # Tentar extrair faixas salariais (ex: 3000 a 5000)
    match_range = re.search(r'(\d+\.?\d*)\s*(?:a|-)\s*(\d+\.?\d*)',
salario_str)
    if match_range:
        min_salario = float(match_range.group(1))
        max_salario = float(match_range.group(2))
    else:
        # Tentar extrair um único valor (ex: 3000)
        match_single = re.search(r'(\d+\.?\d*)', salario_str)
        if match_single:
            min_salario = float(match_single.group(1))
            max_salario = float(match_single.group(1))
```

```
    return min_salario, max_salario

# Exemplo de DataFrame (simulando dados brutos de vagas coletadas)
data = {
    'id_vaga': [1, 2, 3, 4],
    'titulo': [
        'Estágio em Desenvolvimento Python',
        'Desenvolvedor Fullstack Java/React CLT',
        'Jovem Aprendiz em TI',
        'Analista de Dados Sênior - Power BI e SQL'
    ],
    'descricao': [
        'Buscamos estudante de TI com conhecimento em Python e SQL. Vaga de
estágio em Fortaleza.',
        'Experiência com Spring Boot e React. Conhecimento em AWS e Docker.
Contratação CLT. Salário R$ 6.000 - 9.000.',
        'Oportunidade para primeiro emprego na área de TI. Não exige
experiência. Programa de jovem aprendiz.',
        'Analista de dados com experiência em Power BI, SQL e Big Data.
Conhecimento em Python é um diferencial. Salário 8k.'
    ],
    'localizacao': ['Fortaleza, CE', 'Fortaleza', 'Fortaleza - Ceará',
'Remoto (Fortaleza)']
}
df_vagas = pd.DataFrame(data)

# 1. Limpeza de Dados
df_vagas['descricao_limpa'] = df_vagas['descricao'].apply(limpar_texto)

# 2. Categorização de Vagas
df_vagas['categoria_vaga'] = df_vagas.apply(lambda row:
categorizar_vaga(row['titulo'], row['descricao']), axis=1)

# 3. Extração e Normalização de Habilidades
df_vagas['habilidades_extraidas'] =
df_vagas['descricao_limpa'].apply(extrair_e_normalizar_habilidades)

# 4. Normalização de Salários
df_vagas[['salario_min', 'salario_max']] = df_vagas['salario'].apply(lambda
x: pd.Series(normalizar_salario(x)) if 'salario' in df_vagas.columns else
pd.Series([None, None]))

# 5. Padronização de Localização (exemplo simples)
df_vagas['localizacao_padronizada'] = df_vagas['localizacao'].apply(lambda
x: 'Fortaleza, CE' if 'fortaleza' in x.lower() else x)

# Exibindo o DataFrame processado
```

```
print("\nDataFrame Processado:")
print(df_vagas[[
    'id_vaga', 'titulo', 'descricao_limpa', 'categoria_vaga',
    'habilidades_extraidas', 'salario_min', 'salario_max',
'localizacao_padronizada'
]].to_markdown(index=False))

# Exemplo de Agregação: Contagem de vagas por categoria
contagem_categorias = df_vagas['categoria_vaga'].value_counts().reset_index()
contagem_categorias.columns = ['Categoria', 'Numero_Vagas']
print("\nContagem de Vagas por Categoria:")
print(contagem_categorias.to_markdown(index=False))

# Exemplo de Agregação: Top N habilidades
lista_habilidades = [h for sublist in df_vagas['habilidades_extraidas'] for
h in sublist]
contagem_habilidades =
pd.DataFrame(Counter(lista_habilidades).most_common(5), columns=
['Habilidade', 'Frequencia'])
print("\nTop 5 Habilidades Mais Demandadas:")
print(contagem_habilidades.to_markdown(index=False))
```

Este exemplo conceitual demonstra como as etapas de limpeza, categorização, extração de habilidades e normalização de salários podem ser implementadas em Python. Em um ambiente de Big Data com PySpark, a lógica seria similar, mas aplicada a DataFrames distribuídos, aproveitando a capacidade de processamento paralelo do Spark.

4.4. Considerações de Qualidade de Dados e Monitoramento

- **Validação Contínua**: Implementar verificações de qualidade de dados em cada etapa do pipeline. Isso pode incluir a verificação de esquemas, a detecção de anomalias e a garantia de que os dados transformados atendam aos requisitos de negócios.
- **Testes Automatizados**: Escrever testes unitários e de integração para os scripts de processamento e transformação para garantir que as mudanças no código não introduzam erros.
- **Monitoramento**: Configurar ferramentas de monitoramento para acompanhar a execução dos jobs de processamento, o volume de dados processados e a qualidade dos dados de saída. Alertas devem ser configurados para notificar a equipe em caso de falhas ou desvios significativos.

Ao seguir estas diretrizes, o projeto ASCENDE terá um pipeline de processamento e transformação de dados robusto e eficiente, capaz de converter dados brutos em insights acionáveis e confiáveis.

5. Implementar Análise e Modelagem de Dados para o ASCENDE

Com os dados limpos, transformados e armazenados em formatos otimizados, a fase de análise e modelagem de dados é onde os insights propostos pelo projeto ASCENDE são gerados. Esta etapa é fundamental para traduzir os dados brutos em conhecimento acionável, permitindo que estudantes, empresas e instituições de ensino tomem decisões informadas. Python, com seu vasto ecossistema de bibliotecas de ciência de dados e machine learning, será a espinha dorsal desta fase.

5.1. Análise Exploratória de Dados (EDA)

A Análise Exploratória de Dados (EDA) é o primeiro passo para entender a estrutura, os padrões e as anomalias presentes nos dados. Ela serve como base para a formulação de hipóteses e para guiar a escolha de modelos mais complexos.

5.1.1. Objetivos da EDA no ASCENDE

- **Compreensão da Distribuição**: Analisar a distribuição de vagas por categoria (estágio, jovem aprendiz, CLT), por localização (Fortaleza e arredores), por empresa e por data de publicação.
- **Identificação de Habilidades Chave**: Visualizar as habilidades e tecnologias mais frequentes, bem como suas co-ocorrências.
- **Análise de Faixas Salariais**: Entender a distribuição de salários por cargo, nível de experiência e conjunto de habilidades.
- **Detecção de Anomalias**: Identificar vagas com descrições incomuns, salários muito discrepantes ou padrões de publicação irregulares.

5.1.2. Ferramentas Python para EDA

- **Pandas**: Essencial para manipulação e agregação de dados. Permite calcular estatísticas descritivas (média, mediana, desvio padrão, contagens) e realizar agrupamentos (`groupby`) para sumarizar informações.
- **Matplotlib e Seaborn**: Para criar visualizações estáticas de alta qualidade. Histogramas para distribuições, gráficos de barras para contagens e frequências, gráficos de dispersão para relações entre variáveis, e box plots para distribuições salariais são exemplos de gráficos que serão utilizados.
- **Plotly e Bokeh**: Para visualizações interativas que permitem aos usuários explorar os dados em maior profundidade, com funcionalidades de zoom, filtros e tooltips. Isso é particularmente útil para o desenvolvimento de protótipos de dashboards.

- **Jupyter Notebooks/Lab**: Ambiente interativo para realizar a EDA de forma iterativa, documentando o processo com código, texto e visualizações.

5.2. Análise de Tendências e Padrões

Esta etapa foca em extrair insights sobre a dinâmica do mercado de trabalho de tecnologia em Fortaleza, utilizando tanto os dados de vagas individuais quanto os dados agregados governamentais.

5.2.1. Tipos de Análise de Tendências

- **Tendências Temporais de Vagas**: Analisar a evolução do número de vagas por categoria, setor e habilidades ao longo do tempo (mensal, trimestral). Isso pode revelar sazonalidades, crescimento ou declínio de certas áreas. Os dados do CAGED e RAIS serão cruciais para contextualizar essas tendências com o mercado formal geral.
- **Padrões de Demanda por Habilidades**: Identificar como a demanda por habilidades específicas (ex: Python, Cloud, Machine Learning) muda ao longo do tempo e em diferentes setores. Isso ajudará a guiar estudantes e profissionais sobre quais habilidades focar.
- **Análise Geográfica Detalhada**: Mapear a concentração de vagas em Fortaleza, identificando bairros ou regiões com maior oferta. Isso pode ser feito com `Folium` ou `Geopandas` para criar mapas interativos que mostram a densidade de vagas.
- **Análise de Co-ocorrência**: Identificar quais habilidades e tecnologias são frequentemente mencionadas juntas nas descrições das vagas. Isso pode indicar conjuntos de competências valorizados ou pilhas tecnológicas comuns.

5.2.2. Ferramentas Python para Análise de Tendências

- **Pandas**: Para manipulação de séries temporais, reamostragem de dados e cálculos de médias móveis.
- **Statsmodels**: Para análises estatísticas mais avançadas e modelagem de séries temporais (ex: ARIMA, SARIMA) para prever a demanda futura por vagas ou habilidades.
- **NetworkX**: Para analisar redes de co-ocorrência de habilidades, visualizando as conexões entre diferentes competências.

5.3. Modelagem Preditiva e Machine Learning

Para ir além da análise descritiva, o projeto ASCENDE pode incorporar modelos de Machine Learning para prever e classificar informações, agregando um valor significativo.

5.3.1. Aplicações de Machine Learning no ASCENDE

- **Classificação de Vagas**: Refinar a categorização de vagas (estágio, jovem aprendiz, CLT) usando modelos de classificação de texto. Isso pode melhorar a precisão em comparação com regras baseadas em palavras-chave. Algoritmos como Naive Bayes, SVM, Random Forest ou modelos baseados em embeddings de texto podem ser utilizados.
- **Previsão de Demanda**: Construir modelos de séries temporais para prever a demanda futura por cargos específicos ou habilidades em Fortaleza. Isso pode ser valioso para instituições de ensino e para o planejamento de carreira.
- **Recomendação de Vagas/Habilidades**: Desenvolver um sistema de recomendação que sugira vagas para candidatos com base em seu perfil (habilidades, experiência) ou que sugira habilidades a serem desenvolvidas com base nas tendências do mercado.
- **Extração de Tópicos (Topic Modeling)**: Utilizar técnicas como Latent Dirichlet Allocation (LDA) ou Non-negative Matrix Factorization (NMF) para descobrir os principais tópicos ou temas subjacentes nas descrições das vagas, revelando áreas de foco emergentes no mercado de tecnologia.
- **Previsão Salarial**: Construir modelos de regressão para estimar a faixa salarial de uma vaga com base em suas características (cargo, habilidades, nível de experiência, empresa).

5.3.2. Ferramentas Python para Machine Learning

- **Scikit-learn**: A biblioteca padrão para Machine Learning em Python. Oferece uma vasta gama de algoritmos para classificação, regressão, clustering e redução de dimensionalidade, além de ferramentas para pré-processamento e avaliação de modelos.
- **TensorFlow / PyTorch**: Para modelos de Deep Learning, especialmente para tarefas de NLP mais avançadas, como o uso de redes neurais recorrentes (RNNs) ou transformadores (Transformers) para entender o contexto das descrições das vagas e extrair informações mais ricas.
- **Gensim**: Para modelagem de tópicos (LDA) e para trabalhar com embeddings de palavras (Word2Vec, Doc2Vec).
- **PySpark MLlib**: Para aplicar algoritmos de Machine Learning em larga escala em DataFrames distribuídos do Spark, essencial quando o volume de dados excede a capacidade de uma única máquina.

5.4. Considerações para Big Data na Análise e Modelagem

- **Processamento Distribuído**: Para datasets muito grandes, a análise e a modelagem precisarão ser realizadas em um ambiente distribuído. O **PySpark** será fundamental

para isso, permitindo que as operações de Pandas e os algoritmos de Scikit-learn sejam executados em clusters.

- **Feature Engineering em Escala**: A criação de novas features a partir dos dados brutos (ex: contagem de habilidades, indicadores de senioridade) deve ser feita de forma eficiente e escalável, utilizando as capacidades do Spark.
- **Gerenciamento de Modelos (MLOps)**: Para modelos em produção, será necessário um sistema para versionar modelos, monitorar seu desempenho, retreiná-los periodicamente e implantá-los de forma automatizada. Ferramentas como MLflow ou serviços de MLOps em plataformas de nuvem (ex: AWS SageMaker, Google AI Platform) podem ser exploradas.
- **Recursos Computacionais**: A execução de modelos de Machine Learning, especialmente Deep Learning, pode exigir recursos computacionais significativos (GPUs). A utilização de serviços de nuvem com instâncias otimizadas para ML será vantajosa.

Ao implementar esta fase com rigor e utilizando as ferramentas adequadas, o projeto ASCENDE será capaz de gerar os insights profundos e as previsões acuradas que são a essência de sua proposta de valor, transformando dados em um guia estratégico para o mercado de trabalho de tecnologia em Fortaleza.

6. Planejar a Visualização e Relatórios (Site e Dashboards) para o ASCENDE

A fase de visualização e relatórios é onde os resultados do projeto ASCENDE se tornam tangíveis e acessíveis ao público-alvo. O documento do projeto enfatiza a criação de um "guia prático e visual" e um "buscador inteligente" com "gráficos e painéis". Esta seção detalha como esses requisitos serão atendidos, transformando dados complexos em insights claros e acionáveis.

6.1. Objetivos da Visualização e Relatórios no ASCENDE

Os objetivos principais da camada de visualização e relatórios são:

- **Democratizar o Acesso à Informação**: Tornar os insights do mercado de trabalho de tecnologia em Fortaleza acessíveis a estudantes, profissionais, empresas e instituições de ensino.
- **Facilitar a Tomada de Decisão**: Fornecer dados concretos para que os usuários possam planejar suas carreiras, alinhar currículos, e direcionar estratégias de recrutamento.

- **Interatividade**: Permitir que os usuários explorem os dados por conta própria, aplicando filtros e detalhando informações conforme suas necessidades.
- **Atualização Contínua**: Garantir que as visualizações e relatórios sejam atualizados regularmente para refletir as tendências mais recentes do mercado.

6.2. Tipos de Visualizações e Relatórios Chave

Com base nos requisitos funcionais e nos objetivos do ASCENDE, as seguintes visualizações serão prioritárias:

1. **Visão Geral do Mercado de Vagas em Fortaleza**:
    - **Gráfico de Barras/Pizza**: Distribuição de vagas por categoria (Estágio, Jovem Aprendiz, Efetivo/CLT).
    - **Gráfico de Linha**: Tendência do número total de vagas de tecnologia em Fortaleza ao longo do tempo (mensal/trimestral).
    - **Cartão de Indicadores (KPIs)**: Número total de vagas ativas, média salarial geral, principais empresas contratando.
2. **Habilidades e Tecnologias Mais Demandadas**:
    - **Gráfico de Barras Horizontal**: Top N habilidades técnicas mais exigidas, com a possibilidade de filtrar por categoria de vaga, setor ou nível de experiência.
    - **Nuvem de Palavras (Word Cloud)**: Representação visual das habilidades e tecnologias mais frequentes, com o tamanho da palavra indicando a frequência. Útil para uma visão rápida das tendências.
    - **Gráfico de Co-ocorrência**: Visualização de redes que mostra quais habilidades são frequentemente exigidas em conjunto, ajudando a identificar conjuntos de competências valorizados.
3. **Análise Salarial**:
    - **Gráfico de Barras Agrupadas/Box Plot**: Comparação da faixa salarial média/mediana por cargo, nível de experiência, e habilidades específicas. Isso ajudará candidatos a entender o valor de mercado de suas competências.
4. **Distribuição Geográfica das Vagas**:
    - **Mapa Interativo (Coroplético/Heatmap)**: Visualização da concentração de vagas em diferentes bairros ou regiões de Fortaleza. Isso pode ser crucial para estudantes e profissionais que buscam oportunidades próximas a sua residência ou instituições de ensino.
5. **Tendências por Setor/Área de Tecnologia**:

- **Gráfico de Linha/Área Empilhada**: Evolução da demanda por vagas em diferentes sub-áreas da tecnologia (ex: Desenvolvimento Web, Data Science, DevOps, Cibersegurança) ao longo do tempo.

6.3. Ferramentas para Implementação

A escolha das ferramentas para visualização e relatórios dependerá do nível de customização e interatividade desejado, bem como da plataforma de implantação (site próprio vs. ferramenta de BI).

1. **Para o Site Interativo (Buscador Inteligente)**:
   - **Framework Web**: **Flask** ou **Django** (Python) para construir o backend da aplicação web que servirá os dados e as visualizações. O documento menciona "desenvolvimento do site é a etapa final", indicando a necessidade de uma aplicação web customizada.
   - **Bibliotecas de Visualização Interativa**: **Plotly Dash** ou **Streamlit** são excelentes opções para construir dashboards interativos diretamente em Python, que podem ser incorporados ou servidos por um framework web. Eles permitem criar interfaces de usuário complexas com pouco código, conectando diretamente aos dados processados.
   - **Frontend (HTML, CSS, JavaScript)**: Para maior flexibilidade e controle sobre a interface do usuário, pode-se usar um frontend mais tradicional com HTML, CSS e JavaScript, e bibliotecas JS como **D3.js**, **Chart.js** ou **React/Vue/Angular** para renderizar os gráficos, consumindo dados via APIs RESTful expostas pelo backend Python.
2. **Para Relatórios e Dashboards Internos/Gerenciais (Opcional, mas Recomendado)**:
   - **Ferramentas de Business Intelligence (BI)**: **Looker Studio (Google Data Studio)**, **Tableau** ou **Power BI**. Estas ferramentas são ideais para criar dashboards de monitoramento para a equipe do projeto ou para stakeholders que precisam de acesso rápido e fácil aos dados sem a necessidade de interagir com o site principal. Elas se conectam diretamente ao Data Warehouse (BigQuery/Redshift) e oferecem recursos robustos de compartilhamento e governança.

6.4. Processo de Desenvolvimento da Visualização

1. **Design da Interface (UI/UX)**: Criar wireframes e mockups do site e dos dashboards, definindo o layout, a navegação, os filtros e os tipos de gráficos para cada seção. O foco deve ser na simplicidade e clareza para o público-alvo.
2. **Desenvolvimento do Backend da API**: Se um frontend separado for usado, desenvolver endpoints de API (com Flask/FastAPI) que sirvam os dados agregados e as

análises pré-calculadas do Data Warehouse/NoSQL para o frontend.

3. **Implementação das Visualizações**: Utilizar as bibliotecas Python ou frameworks web escolhidos para construir os gráficos e painéis interativos. Garantir que os filtros funcionem corretamente e que as visualizações respondam dinamicamente às interações do usuário.
4. **Integração com o Backend**: Conectar as visualizações aos dados atualizados, garantindo que o site reflita as informações mais recentes do pipeline de dados.
5. **Testes de Usabilidade e Performance**: Realizar testes com usuários reais para garantir que o site seja intuitivo e que as visualizações carreguem rapidamente. Otimizar o desempenho do frontend e do backend conforme necessário.
6. **Implantação**: Publicar o site em um ambiente de produção (servidores web, serviços de hospedagem de nuvem como Vercel, Netlify, ou instâncias EC2/GCE com Docker/Kubernetes).

6.5. Atualização e Manutenção

- **Pipeline de Atualização**: O pipeline de dados (orquestrado pelo Airflow) deve garantir que os dados no Data Warehouse/NoSQL sejam atualizados regularmente (ex: diariamente ou semanalmente). O site e os dashboards devem refletir essas atualizações automaticamente.
- **Monitoramento**: Monitorar a disponibilidade e o desempenho do site e dos dashboards. Configurar alertas para falhas ou lentidão.
- **Feedback Contínuo**: Coletar feedback dos usuários para identificar melhorias e novas funcionalidades para as visualizações e relatórios.

Ao planejar cuidadosamente esta fase, o projeto ASCENDE não apenas processará e analisará dados, mas também os transformará em uma ferramenta poderosa e acessível para todos os envolvidos no mercado de trabalho de tecnologia em Fortaleza.

7. Considerações Finais e Boas Práticas para o Projeto ASCENDE

O desenvolvimento do projeto ASCENDE, com sua complexidade e abrangência, exige a adoção de um conjunto de boas práticas e a atenção a considerações cruciais para garantir seu sucesso, sustentabilidade e impacto a longo prazo. Como especialista em Big Data e Python, destaco os seguintes pontos para guiar a execução do projeto.

7.1. Desenvolvimento Iterativo e Ágil

Projetos de Big Data e Machine Learning são inerentemente iterativos. A metodologia ágil, com ciclos de desenvolvimento curtos (sprints), é a mais adequada para o ASCENDE. Isso permite:

- **Entrega Contínua de Valor**: Funcionalidades são entregues em incrementos, permitindo que os stakeholders vejam o progresso e forneçam feedback regularmente.
- **Adaptação a Mudanças**: O mercado de trabalho e as tecnologias evoluem rapidamente. Uma abordagem ágil permite que o projeto se adapte a novos requisitos ou desafios (ex: mudanças na estrutura de um site de vagas, novas tecnologias emergentes).
- **Gerenciamento de Riscos**: Problemas são identificados e resolvidos mais cedo, reduzindo o risco de falhas maiores no final do projeto.
- **Começar Pequeno**: Iniciar com um Mínimo Produto Viável (MVP) que atenda aos requisitos mais críticos (ex: coleta de vagas de uma única fonte, análise básica de habilidades, dashboard simples) e expandir gradualmente. Isso valida a abordagem e gera valor rapidamente.

7.2. Qualidade e Governança de Dados

A qualidade dos dados é a espinha dorsal do ASCENDE. Análises e insights só serão confiáveis se os dados subjacentes forem precisos e consistentes. A governança de dados é essencial para manter essa qualidade ao longo do tempo.

- **Validação de Dados em Cada Etapa**: Implementar verificações de qualidade de dados (Data Quality Checks) em cada fase do pipeline:
  - **Ingestão**: Verificar se os dados foram coletados integralmente, se os formatos estão corretos e se não há dados corrompidos.
  - **Processamento**: Validar se as transformações estão produzindo os resultados esperados, se não há valores nulos inesperados após a limpeza, e se as entidades extraídas são relevantes.
  - **Armazenamento**: Garantir que os dados no Data Warehouse/NoSQL estejam em conformidade com o esquema definido.
- **Linhagem de Dados (Data Lineage)**: Documentar a origem de cada dado, as transformações aplicadas e seu destino final. Isso é crucial para auditoria, depuração e para entender o impacto de mudanças em qualquer parte do pipeline.
- **Catálogo de Metadados**: Manter um repositório centralizado de metadados que descreva os datasets, seus campos, tipos de dados, fontes, frequência de atualização e proprietários. Isso facilita a descoberta e o entendimento dos dados por toda a equipe.

- **Monitoramento de Qualidade de Dados**: Utilizar ferramentas (ex: Great Expectations em Python) para definir expectativas sobre os dados e monitorar continuamente se essas expectativas são atendidas. Gerar alertas em caso de desvios.

7.3. Monitoramento, Alerta e Observabilidade

Um pipeline de Big Data é um sistema dinâmico. A capacidade de monitorar seu desempenho e ser alertado sobre problemas é vital para a confiabilidade.

- **Monitoramento de Pipeline**: Acompanhar a execução dos DAGs do Airflow/Prefect, o tempo de execução de cada tarefa, o uso de recursos (CPU, memória) e o volume de dados processados.
- **Alertas Proativos**: Configurar alertas para:
  - **Falhas de Web Scraping**: Bloqueios de IP, mudanças na estrutura do site, erros de conexão.
  - **Anomalias de Dados**: Quedas abruptas no volume de vagas coletadas, valores fora do padrão em campos numéricos, aumento de valores nulos.
  - **Desempenho de Modelos de ML**: Degradação da acurácia de modelos de classificação ou previsão.
- **Observabilidade**: Ir além do monitoramento, permitindo entender o "porquê" de um problema. Isso envolve a coleta de logs detalhados, métricas e traces distribuídos para depuração eficiente.

7.4. Segurança e Privacidade de Dados

Dado que o projeto lida com informações do mercado de trabalho, a segurança e a privacidade são primordiais.

- **Conformidade com LGPD**: Garantir que todas as etapas do pipeline estejam em conformidade com a Lei Geral de Proteção de Dados (LGPD) e outras regulamentações relevantes. Isso inclui:
  - **Anonimização/Pseudonimização**: Se houver dados pessoais (ex: nomes de empresas em descrições de vagas que possam ser sensíveis), considerar técnicas para proteger a privacidade.
  - **Controle de Acesso**: Implementar políticas de acesso rigorosas aos dados brutos e processados, garantindo que apenas usuários e serviços autorizados possam acessá-los.
  - **Criptografia**: Criptografar dados em repouso (no Data Lake e Data Warehouse) e em trânsito (durante a comunicação entre os componentes do pipeline).

- **Segurança da Infraestrutura**: Proteger os ambientes de nuvem e os servidores contra acessos não autorizados, utilizando firewalls, grupos de segurança e gerenciamento de identidades e acessos (IAM).

7.5. Escalabilidade e Otimização de Custos

O projeto ASCENDE deve ser capaz de crescer com o volume de dados e a demanda por análises.

- **Design para Escalabilidade Horizontal**: Utilizar tecnologias que permitam adicionar mais recursos de computação e armazenamento conforme a necessidade (ex: Apache Spark, Data Lakes e Data Warehouses em nuvem).
- **Otimização de Recursos em Nuvem**: Monitorar e otimizar continuamente o uso de recursos em plataformas de nuvem para controlar os custos. Isso pode envolver o uso de instâncias spot, instâncias reservadas, ou o dimensionamento automático de clusters.
- **Arquitetura Serverless (Opcional)**: Para algumas partes do pipeline (ex: funções de processamento pontuais), considerar o uso de funções serverless (AWS Lambda, Google Cloud Functions) para reduzir custos operacionais e gerenciar automaticamente a escala.

7.6. Documentação Abrangente

Uma documentação clara e atualizada é essencial para a manutenção, evolução e para a colaboração da equipe.

- **Documentação Técnica**: Incluir diagramas de arquitetura, descrições detalhadas dos componentes, esquemas de banco de dados, dicionários de dados e guias de implantação.
- **Documentação de Código**: Comentários claros no código, docstrings para funções e classes, e READMEs para cada módulo ou repositório.
- **Documentação de Processos Operacionais**: Guias passo a passo para a execução de tarefas comuns, como a implantação de novas versões, a resolução de problemas e a adição de novas fontes de dados.

7.7. Colaboração e Comunicação

O sucesso do ASCENDE dependerá da colaboração eficaz entre engenheiros de dados, cientistas de dados, desenvolvedores web e stakeholders de negócio.

- **Comunicação Regular**: Estabelecer canais de comunicação claros e reuniões regulares para discutir o progresso, desafios e feedback.

- **Ferramentas de Colaboração**: Utilizar ferramentas de controle de versão (Git/GitHub/GitLab), gerenciamento de projetos (Jira, Trello) e comunicação (Slack, Microsoft Teams).
- **Alinhamento de Expectativas**: Garantir que todos os stakeholders tenham uma compreensão clara dos objetivos do projeto, das limitações e do cronograma.

Ao integrar essas considerações e boas práticas em todas as fases do projeto, o ASCENDE não apenas alcançará seus objetivos iniciais, mas também se tornará uma solução robusta, sustentável e valiosa para o mercado de trabalho de tecnologia em Fortaleza.

8. Primeiros Passos Práticos para Aquisição de Dados

Com o plano estratégico e a arquitetura definidos, é hora de colocar a mão na massa e iniciar a fase de aquisição de dados. Esta seção detalha os primeiros passos práticos, focando em uma abordagem incremental e de prova de conceito (PoC) para validar as ferramentas e técnicas antes de escalar.

8.1. Configuração do Ambiente de Desenvolvimento Python

Antes de escrever qualquer código, é fundamental ter um ambiente de desenvolvimento Python bem configurado. Isso garante que as dependências sejam gerenciadas corretamente e que o projeto seja replicável.

1. **Instalação do Python**: Certifique-se de ter o Python 3.8+ instalado. Você pode baixar a versão mais recente do site oficial do Python [1].
2. **Gerenciamento de Ambientes Virtuais**: É altamente recomendável usar ambientes virtuais para isolar as dependências do seu projeto. Isso evita conflitos entre diferentes projetos.
   - `venv` **(módulo padrão do Python)**: Para criar um ambiente virtual:
   - `conda` **(Anaconda/Miniconda)**: Se você já usa Conda, pode criar um ambiente assim:
3. **Instalação de Bibliotecas Essenciais**: Com o ambiente virtual ativado, instale as bibliotecas Python que serão usadas para a aquisição de dados:
4. **IDE (Integrated Development Environment)**: Utilize uma IDE como VS Code [2] ou PyCharm [3] para facilitar a escrita, depuração e gerenciamento do código.

8.2. Prova de Conceito (PoC) para Web Scraping de um Portal de Empregos

Para começar com o web scraping, selecione um portal de empregos com uma estrutura relativamente simples e que permita o scraping (verifique sempre o `robots.txt` e os Termos

de Serviço). Para esta PoC, vamos focar em extrair informações básicas de uma página de listagem de vagas.

8.2.1. Escolha do Portal para PoC

Escolha um portal que seja relevante para vagas de tecnologia em Fortaleza e que pareça ter uma estrutura HTML mais previsível. Evite LinkedIn ou Gupy para a primeira PoC, pois são mais complexos. Um site como o CIEE (para estágios) ou um portal local menor pode ser um bom ponto de partida.

8.2.2. Passos Práticos

1. **Análise da Página**: Abra a página de busca de vagas do portal no seu navegador (ex: busque por "desenvolvedor fortaleza"). Use as ferramentas de desenvolvedor do navegador (F12) para inspecionar o HTML da página. Identifique:
   - Onde estão os títulos das vagas.
   - Onde estão os links para as páginas de detalhe das vagas.
   - Onde estão as informações de empresa, localização, data de publicação.
   - Identifique os seletores CSS ou XPath que podem ser usados para extrair essas informações.
2. **Desenvolvimento do Script Básico com `requests` e `BeautifulSoup`**:
3. **Refinamento para Páginas de Detalhe**: Se o script acima funcionar, o próximo passo é visitar cada `link` de vaga para extrair a descrição completa e outras informações detalhadas. Isso envolverá um loop adicional e a aplicação de `requests` e `BeautifulSoup` para cada página de detalhe.
4. **Armazenamento Temporário**: Salve os dados extraídos em um arquivo CSV ou JSON localmente para inspeção. Isso valida que os dados estão sendo coletados corretamente.

8.3. Prova de Conceito (PoC) para Download de Dados Governamentais

Para os dados governamentais, a PoC focará no download de um arquivo específico do CAGED ou RAIS.

8.3.1. Escolha do Dataset para PoC

Selecione um arquivo de dados do CAGED (ex: o mais recente disponível) no portal do Ministério do Trabalho e Emprego. Procure por um link direto para download de um arquivo CSV ou TXT.

8.3.2. Passos Práticos

1. **Identificação da URL de Download**: Navegue até a página de dados do CAGED (ex: https://www.gov.br/trabalho-e-emprego/pt-br/servicos/empregador/caged/estatisticas) e encontre o link direto para o arquivo que deseja baixar. Pode ser necessário inspecionar o HTML para encontrar o URL exato do arquivo (geralmente um `.csv`, `.zip` ou `.xlsx`).
2. **Desenvolvimento do Script Básico com `requests` e `pandas`**:
3. **Tratamento de Arquivos Compactados**: Se o arquivo for um `.zip`, você precisará descompactá-lo após o download. A biblioteca `zipfile` do Python pode ser usada para isso.

8.4. Próximos Passos após as PoCs

Após validar a coleta de dados com essas Provas de Conceito, os próximos passos seriam:

- **Escalar o Web Scraping**: Desenvolver os scrapers completos para todos os portais de emprego definidos, incorporando as boas práticas (rotação de User-Agents, proxies, atrasos, tratamento de erros).
- **Automatizar Downloads Governamentais**: Criar scripts robustos para baixar e processar os dados do CAGED, RAIS e PNAD de forma automatizada e periódica.
- **Integração com Data Lake**: Modificar os scripts para armazenar os dados brutos diretamente no Data Lake (S3/GCS).
- **Orquestração**: Começar a implementar os DAGs no Apache Airflow para agendar e monitorar todas as tarefas de aquisição de dados.

Esses primeiros passos práticos são cruciais para construir uma base sólida para o projeto ASCENDE, permitindo que você ganhe experiência com as ferramentas e valide a estratégia de aquisição de dados antes de avançar para as fases de processamento e análise em larga escala.

Referências:

[1] Python.org. Disponível em: https://www.python.org/downloads/ [2] Visual Studio Code. Disponível em: https://code.visualstudio.com/ [3] PyCharm. Disponível em: https://www.jetbrains.com/pycharm/

4. Detalhar Coleta e Integração de Dados Governamentais (CAGED, RAIS, PNAD)

A coleta e integração de dados governamentais como o Cadastro Geral de Empregados e Desempregados (CAGED), a Relação Anual de Informações Sociais (RAIS) e a Pesquisa Nacional por Amostra de Domicílios Contínua (PNAD Contínua) do IBGE são cruciais para o projeto ASCENDE. Eles fornecem um panorama macro do mercado de trabalho, permitindo contextualizar e validar os insights obtidos das vagas individuais. No entanto, a forma de disponibilização desses dados pode ser um desafio. Esta seção detalha como realizar essa coleta e integração usando Python.

4.1. Coleta e Integração de Dados do CAGED

O CAGED registra as admissões e desligamentos de trabalhadores com carteira assinada, sendo uma fonte valiosa para entender a dinâmica do emprego formal. Os dados são disponibilizados mensalmente pelo Ministério do Trabalho e Emprego (MTE).

4.1.1. Localização dos Dados

Os dados do CAGED podem ser encontrados no portal do Ministério do Trabalho e Emprego, geralmente na seção de estatísticas ou acesso à informação. O link principal para as estatísticas do CAGED é: https://www.gov.br/trabalho-e-emprego/pt-br/servicos/empregador/caged/estatisticas [1].

Nesta página, você geralmente encontrará links para arquivos mensais e anuais, muitas vezes compactados em formato `.zip` e contendo arquivos `.csv` ou `.txt`.

4.1.2. Estratégia de Coleta com Python

1. **Identificação da URL de Download**: A primeira etapa é navegar programaticamente até a página do CAGED e identificar os links para os arquivos de dados desejados. Isso pode ser feito com `requests` e `BeautifulSoup`.
    - **Exemplo de Código (Identificação de Links)**:

".zip" ou ".csv" e "caged" download_links = [] for a_tag in soup.find_all("a", href=True): href = a_tag["href"] if (".zip" in href or ".csv" in href) and "caged" in href.lower(): # Construir URL completa se for relativa if not href.startswith("http"): href = requests.compat.urljoin(url_caged_estatisticas, href) download_links.append(href)

Plain Text

```
    print("Links de download do CAGED encontrados:")
    for link in download_links:
        print(link)
    ```
```
```

2. **Download dos Arquivos**: Uma vez que os links de download são identificados, o próximo passo é baixar os arquivos. Muitos arquivos do CAGED são grandes e compactados. * **Exemplo de Código (Download e Descompactação)**: ```python import requests import zipfile import os

Plain Text

```
    def download_file(url, destination_folder):
        os.makedirs(destination_folder, exist_ok=True)
        local_filename = os.path.join(destination_folder, url.split("/")[-1])
        print(f"Baixando {url} para {local_filename}...")
        with requests.get(url, stream=True) as r:
            r.raise_for_status()
            with open(local_filename, "wb") as f:
                for chunk in r.iter_content(chunk_size=8192):
                    f.write(chunk)
        return local_filename

    def unzip_file(zip_path, extract_to_folder):
        print(f"Descompactando {zip_path} para {extract_to_folder}...")
        with zipfile.ZipFile(zip_path, "r") as zip_ref:
            zip_ref.extractall(extract_to_folder)
        print("Descompactação concluída.")

    # Exemplo de uso (substitua com um link real encontrado)
    caged_zip_url = "https://www.gov.br/trabalho-e-emprego/pt-
br/servicos/empregador/caged/estatisticas/arquivos/CAGED_Movimentacao_012023.
zip" # Exemplo
    raw_data_folder = "./data/raw/caged"

    downloaded_zip = download_file(caged_zip_url, raw_data_folder)
    unzip_file(downloaded_zip, raw_data_folder)
    ```
```

3. **Leitura e Pré-processamento com Pandas**: Os arquivos do CAGED geralmente são `.csv` ou `.txt` com delimitadores específicos (ponto e vírgula, tabulação) e codificação `latin1` ou `cp1252`. * **Exemplo de Código (Leitura e Inspeção)**: ```python import pandas as pd import os

Plain Text

```
    # Assumindo que o arquivo CSV foi descompactado para raw_data_folder
    caged_csv_path = os.path.join(raw_data_folder,
"CAGED_Movimentacao_012023.csv") # Exemplo

    try:
```
```

```
        # Pode ser necessário ajustar o separador (sep) e a codificação
(encoding)
        df_caged = pd.read_csv(caged_csv_path, sep=";", encoding="latin1",
low_memory=False)
        print("\nPrimeiras 5 linhas do DataFrame CAGED:")
        print(df_caged.head().to_markdown(index=False))
        print("\nInformações do DataFrame CAGED:")
        df_caged.info()
    except FileNotFoundError:
        print(f"Arquivo não encontrado: {caged_csv_path}")
    except Exception as e:
        print(f"Erro ao ler o arquivo CSV: {e}")
    ```
```

4. **Integração e Armazenamento no Data Lake**: Após a leitura, os dados brutos devem ser armazenados no Data Lake (S3/GCS) em um formato otimizado como Parquet, mantendo a estrutura original.

## 4.2. Coleta e Integração de Dados da RAIS

A RAIS (Relação Anual de Informações Sociais) é um registro administrativo que coleta informações sobre o vínculo empregatício no Brasil. Os dados são anuais e complementam o CAGED, fornecendo um retrato mais completo do emprego formal.

### 4.2.1. Localização dos Dados

Os dados da RAIS também são disponibilizados pelo MTE, muitas vezes na mesma seção de estatísticas ou em um portal dedicado. O acesso pode ser via: https://www.gov.br/trabalho-e-emprego/pt-br/acesso-a-informacao/dados-abertos/rais [2].

Os arquivos da RAIS são geralmente muito grandes e vêm em formato `.txt` ou `.csv` compactados por UF (Unidade da Federação) e ano.

### 4.2.2. Estratégia de Coleta com Python

O processo é muito similar ao do CAGED:

1. **Identificação da URL de Download**: Usar `requests` e `BeautifulSoup` para navegar na página da RAIS e identificar os links para os arquivos `.zip` ou `.txt` por ano e UF.
2. **Download e Descompactação**: Utilizar a função `download_file` e `unzip_file` (adaptadas) para baixar e descompactar os arquivos da RAIS para uma pasta específica no Data Lake (ex: `./data/raw/rais`).
3. **Leitura e Pré-processamento com Pandas**: Os arquivos da RAIS são geralmente `.txt` com um layout fixo ou delimitados por ponto e vírgula. A leitura exige atenção ao `sep` e `encoding`.
```

- **Exemplo de Código (Leitura e Inspeção)**:
- **Dicionário de Dados**: A RAIS possui um dicionário de dados extenso que é **fundamental** para interpretar as colunas. Este dicionário também deve ser baixado e armazenado.

4.3. Coleta e Integração de Dados da PNAD Contínua (IBGE)

A PNAD Contínua é uma pesquisa domiciliar que coleta informações sobre o mercado de trabalho, educação, renda e outras características demográficas. Ela fornece microdados que permitem análises detalhadas sobre a força de trabalho, ocupação e desocupação.

4.3.1. Localização dos Dados

Os microdados da PNAD Contínua são disponibilizados no site do IBGE: https://www.ibge.gov.br/estatisticas/sociais/trabalho/17270-pnad-continua.html?t=microdados [3].

Os arquivos são geralmente compactados (ZIP) e contêm arquivos `.txt` ou `.csv` com os microdados, além de dicionários de variáveis e layouts de registro.

4.3.2. Estratégia de Coleta com Python

O processo é similar aos anteriores, mas com uma atenção especial aos dicionários de dados.

1. **Identificação da URL de Download**: Usar `requests` e `BeautifulSoup` para navegar na página da PNAD Contínua e identificar os links para os arquivos `.zip` dos microdados e seus respectivos dicionários de variáveis por trimestre/ano.
2. **Download e Descompactação**: Utilizar as funções de download e descompactação para baixar os arquivos para uma pasta específica no Data Lake (ex: `./data/raw/pnad`).
3. **Leitura e Pré-processamento com Pandas**: Os microdados da PNAD Contínua são geralmente arquivos `.txt` com layout fixo, o que exige uma leitura mais cuidadosa. O dicionário de variáveis é **indispensável** para saber a posição e o significado de cada campo.
    - **Exemplo de Código (Leitura de Layout Fixo)**:
    - **Dicionário de Variáveis**: O IBGE fornece dicionários de variáveis em PDF ou XLSX. É crucial tê-los para interpretar os códigos e valores nos microdados. Eles devem ser armazenados junto com os dados brutos.

4.4. Orquestração e Automação

Para todas as fontes governamentais, a orquestração com **Apache Airflow** ou **Prefect** é fundamental. Serão criados DAGs para:

- **CAGED**: Um DAG mensal para baixar o arquivo mais recente, descompactar e armazenar no Data Lake.
- **RAIS**: Um DAG anual para baixar os arquivos por UF e ano, descompactar e armazenar.
- **PNAD Contínua**: Um DAG trimestral/anual para baixar os microdados e seus dicionários, descompactar e armazenar.

Cada DAG deve incluir etapas para:

- **Verificação de Novas Versões**: Antes de baixar, verificar se há uma versão mais recente do arquivo disponível.
- **Download Robusto**: Com retries e tratamento de erros.
- **Armazenamento no Data Lake**: Salvar os arquivos brutos e descompactados em pastas organizadas por fonte, ano, mês/trimestre.
- **Notificação**: Enviar alertas em caso de falhas ou sucesso da operação.

4.5. Considerações Finais para Dados Governamentais

- **Documentação**: A documentação fornecida pelos órgãos (MTE, IBGE) é a sua melhor amiga. Ela detalha o formato dos arquivos, o significado das colunas e as codificações.
- **Volume de Dados**: Especialmente RAIS e PNAD Contínua podem gerar arquivos muito grandes. Esteja preparado para lidar com isso, utilizando `low_memory=False` no Pandas ou, para volumes muito grandes, considere usar PySpark para a leitura e processamento inicial.
- **Atualização Contínua**: Os portais governamentais podem mudar suas URLs ou formatos de disponibilização. Os scripts de coleta precisarão de manutenção periódica.
- **Licença de Uso**: Verifique sempre as licenças de uso dos dados para garantir a conformidade com os termos de redistribuição e análise.

Ao seguir esses passos, você terá uma base sólida para coletar e integrar os dados governamentais no seu projeto ASCENDE, enriquecendo significativamente a análise do mercado de trabalho de tecnologia em Fortaleza.

4.5. Considerações de Qualidade de Dados e Monitoramento

- **Validação Contínua**: Implementar verificações de qualidade de dados em cada etapa do pipeline. Isso pode incluir a verificação de esquemas, a detecção de anomalias e a garantia de que os dados transformados atendam aos requisitos de negócios.

- **Testes Automatizados**: Escrever testes unitários e de integração para os scripts de processamento e transformação para garantir que as mudanças no código não introduzam erros.
- **Monitoramento**: Configurar ferramentas de monitoramento para acompanhar a execução dos jobs de processamento, o volume de dados processados e a qualidade dos dados de saída. Alertas devem ser configurados para notificar a equipe em caso de falhas ou desvios significativos.

Ao seguir estas diretrizes, o projeto ASCENDE terá um pipeline de processamento e transformação de dados robusto e eficiente, capaz de converter dados brutos em insights acionáveis e confiáveis.

5. Desenvolver o Processamento e Transformação de Dados com Python para o ASCENDE

Com os dados brutos coletados e armazenados no Data Lake, a próxima etapa crítica é transformá-los em informações valiosas e estruturadas que possam ser utilizadas para análise e modelagem. Esta fase é o coração do pipeline de dados, onde Python, com seu ecossistema robusto de bibliotecas, desempenha um papel central. O objetivo é garantir que os dados sejam limpos, consistentes, enriquecidos e prontos para gerar os insights que o projeto ASCENDE propõe.

5.1. Etapas Essenciais de Processamento e Transformação

O processo de transformação de dados para o ASCENDE envolverá várias sub-etapas, cada uma com um propósito específico para refinar os dados brutos.

5.1.1. Limpeza de Dados (Data Cleaning)

A limpeza de dados é a primeira linha de defesa contra a "sujeira" inerente aos dados brutos, especialmente aqueles provenientes de web scraping. Dados inconsistentes, incompletos ou incorretos podem levar a análises falhas e insights enganosos. As principais atividades incluem:

- **Remoção de Duplicatas**: Vagas podem ser coletadas múltiplas vezes de uma mesma fonte ou de fontes diferentes. É crucial identificar e remover registros duplicados com base em critérios como URL da vaga, título e empresa. Bibliotecas como `pandas` oferecem métodos eficientes como `df.drop_duplicates()` para essa finalidade.
- **Tratamento de Valores Ausentes (Missing Values)**: Campos como salário, nível de experiência ou até mesmo partes da descrição podem estar ausentes. As estratégias incluem:

    - **Remoção**: Descartar registros com valores ausentes em campos críticos (ex: título da vaga).
    - **Preenchimento**: Substituir valores ausentes por um valor padrão (ex: "Não Informado"), a média/mediana (para campos numéricos), ou por inferência a partir de outros campos.
    - **Imputação**: Utilizar modelos mais sofisticados para prever valores ausentes, embora isso possa ser um passo mais avançado.
- **Padronização de Formatos**: Garantir que todos os dados sigam um formato consistente:
    - **Datas**: Converter todas as datas de publicação para um formato uniforme (ex: `YYYY-MM-DD`).
    - **Textos**: Converter todo o texto para minúsculas, remover espaços em branco extras (`strip()`), e lidar com quebras de linha e tabulações. Isso é vital para análises de texto subsequentes.
    - **Numéricos**: Assegurar que campos como salários ou contagens sejam tratados como tipos numéricos, e não como strings.
- **Remoção de Caracteres Especiais e HTML**: Descrições de vagas frequentemente contêm tags HTML, entidades HTML (ex: `&amp;`), e caracteres especiais que precisam ser removidos ou normalizados para que o texto seja limpo para NLP. Expressões regulares (`re` em Python) e bibliotecas como `BeautifulSoup` (para remover tags HTML) são ferramentas essenciais aqui.

5.1.2. Enriquecimento de Dados (Data Enrichment)

O enriquecimento de dados visa adicionar valor aos dados brutos, extraindo informações implícitas ou combinando-os com outras fontes para criar novos atributos. Para o ASCENDE, isso é fundamental para atender aos requisitos de identificação de habilidades, tecnologias e categorização de vagas.

- **Extração de Entidades (Named Entity Recognition - NER)**: Utilizar técnicas de Processamento de Linguagem Natural (NLP) para identificar e extrair entidades relevantes das descrições das vagas. Para o ASCENDE, isso significa extrair:
    - **Habilidades Técnicas**: Python, SQL, Java, React, AWS, Azure, Docker, Kubernetes, etc.
    - **Habilidades Comportamentais (Soft Skills)**: Comunicação, Liderança, Trabalho em Equipe, Resolução de Problemas.
    - **Ferramentas e Plataformas**: Tableau, Power BI, Jira, Git, Figma.

    - **Tecnologias Específicas**: Machine Learning, Inteligência Artificial, Big Data, Cloud Computing.
    - **Localização**: Refinar a extração de cidades, estados e bairros, se necessário.
    - **Bibliotecas**: `spaCy` e `NLTK` são as principais bibliotecas Python para NER. Pode ser necessário treinar modelos customizados ou usar dicionários extensos de termos para melhorar a precisão da extração.
- **Classificação de Setores/Áreas**: Embora o foco seja tecnologia, as vagas podem ter nuances. Pode-se classificar a vaga em sub-áreas de tecnologia (ex: Desenvolvimento Web, Data Science, DevOps, Segurança da Informação) com base em palavras-chave ou modelos de ML.
- **Normalização de Salários**: Vagas podem apresentar salários de diversas formas (ex: "R$ 3.000-5.000", "5k", "A combinar", "Salário competitivo"). Esta etapa envolve:
    - **Extração**: Usar expressões regulares para extrair valores numéricos e faixas salariais.
    - **Padronização**: Converter todos os valores para uma unidade monetária e escala padrão (ex: BRL, mensal). Converter "5k" para "5000".
    - **Inferência**: Para vagas com "A combinar" ou "Salário competitivo", pode-se tentar inferir uma faixa salarial com base em cargos e habilidades similares, usando modelos preditivos.
- **Enriquecimento Geográfico**: Se a localização for muito genérica (ex: "Fortaleza"), pode-se usar APIs de geocodificação (ex: Google Maps API, OpenStreetMap Nominatim via `geopy`) para obter coordenadas geográficas (latitude, longitude) e até mesmo informações de bairros ou regiões administrativas.
- **Adição de Metadados**: Incluir metadados como a data de coleta, a fonte original da vaga, e um identificador único gerado pelo sistema.

5.1.3. Padronização e Categorização (Standardization & Categorization)

Esta etapa visa criar consistência e facilitar a análise ao agrupar informações semelhantes.

- **Categorização de Vagas (Estágio, Jovem Aprendiz, Efetivo/CLT)**: Conforme requisito do ASCENDE, esta categorização será feita com base em regras de palavras-chave no título e/ou descrição da vaga. Um conjunto de regras bem definido e testado será implementado. Ex: se "estágio" ou "intern" no título -> Estágio; se "jovem aprendiz" -> Jovem Aprendiz; caso contrário, e se houver termos como "CLT", "efetivo" -> Efetivo.
- **Normalização de Habilidades e Tecnologias**: Após a extração, é comum ter variações (ex: "Python", "python", "Py"). É necessário normalizar esses termos para uma representação única (ex: sempre "Python"). Isso pode ser feito com um dicionário de sinônimos ou técnicas de agrupamento de strings.

- **Padronização de Títulos de Cargo**: Criar um conjunto limitado de títulos de cargo padronizados (ex: "Desenvolvedor Frontend", "Cientista de Dados", "Engenheiro de DevOps") e mapear os títulos originais das vagas para esses padrões. Isso facilita a análise comparativa entre cargos.

5.1.4. Agregação de Dados (Data Aggregation)

Após a limpeza e enriquecimento, os dados podem ser agregados para gerar métricas e resumos que alimentam as visualizações e análises.

- **Contagem de Vagas**: Total de vagas por categoria (estágio, jovem aprendiz, CLT), por setor, por localização, por empresa.
- **Frequência de Habilidades**: Contagem de quantas vezes cada habilidade/tecnologia aparece nas vagas.
- **Salário Médio/Mediana**: Calcular o salário médio, mediano, mínimo e máximo por cargo, setor, nível de experiência.
- **Tendências Temporais**: Agregação de dados por dia, semana ou mês para analisar a evolução das métricas ao longo do tempo.

5.2. Ferramentas Python para Processamento e Transformação

- **Pandas**: A biblioteca fundamental para manipulação e análise de dados tabulares em Python. Ideal para operações de limpeza, transformação e agregação em datasets que cabem na memória de uma única máquina. Será amplamente utilizada para prototipagem e para processamento de dados governamentais e de vagas individuais, caso o volume não exija processamento distribuído.
- **NumPy**: Base para operações numéricas de alta performance, frequentemente usado em conjunto com Pandas.
- **Scikit-learn**: Para tarefas de Machine Learning, como classificação de texto (para categorização de vagas ou setores), agrupamento (clustering) de vagas similares, ou para engenharia de features.
- **NLTK (Natural Language Toolkit) / spaCy**: Bibliotecas essenciais para Processamento de Linguagem Natural (NLP). `spaCy` é geralmente preferido para tarefas de produção devido à sua velocidade e modelos pré-treinados eficientes para NER e outras tarefas. `NLTK` oferece uma gama mais ampla de algoritmos e recursos para pesquisa e desenvolvimento.
- **Regex (módulo `re`)**: Para extração de padrões específicos de texto, como valores salariais, datas, códigos, ou para limpeza de strings.

- **Apache Spark (PySpark)**: Para processamento de Big Data em larga escala. Quando os volumes de dados excederem a capacidade de memória de uma única máquina, o PySpark será indispensável. Ele permite aplicar as mesmas lógicas de limpeza, enriquecimento e transformação de forma distribuída em um cluster, garantindo escalabilidade e performance. As APIs de DataFrame do PySpark são muito semelhantes às do Pandas, facilitando a transição.

5.3. Exemplo de Fluxo de Processamento com Python (Conceitual para ASCENDE)

Python

```
import pandas as pd
import re
import spacy
from collections import Counter

# Carregar modelo spaCy para português
try:
    nlp = spacy.load("pt_core_news_sm")
except OSError:
    print("Baixando modelo pt_core_news_sm para spaCy...")
    spacy.cli.download("pt_core_news_sm")
    nlp = spacy.load("pt_core_news_sm")

# Dicionário de habilidades e tecnologias (exemplo simplificado)
habilidades_tecnicas = [
    "python", "java", "javascript", "sql", "aws", "azure", "gcp", "docker",
    "kubernetes", "react", "angular", "vue", "node.js", "spring boot",
    "machine learning", "inteligencia artificial", "big data", "devops",
    "git", "jira", "tableau", "power bi", "excel", "linux", "cloud"
]

def limpar_texto(texto):
    if pd.isna(texto): return ""
    texto = str(texto).lower() # Converter para minúsculas
    texto = re.sub(r'<.*?>', '', texto) # Remover tags HTML
    texto = re.sub(r'&[a-z]+;', '', texto) # Remover entidades HTML
    texto = re.sub(r'[^a-z0-9áéíóúâêîôûãõç\s]', '', texto) # Remover
caracteres especiais, manter acentos
    texto = re.sub(r'\s+', ' ', texto).strip() # Remover espaços extras
    return texto

def categorizar_vaga(titulo, descricao):
    titulo_lower = titulo.lower() if pd.notna(titulo) else ""
    descricao_lower = descricao.lower() if pd.notna(descricao) else ""
```

```
    if "estágio" in titulo_lower or "intern" in titulo_lower or "estagiário"
in descricao_lower:
        return "Estágio"
    if "jovem aprendiz" in titulo_lower or "aprendiz" in titulo_lower or
"jovem aprendiz" in descricao_lower:
        return "Jovem Aprendiz"
    if "clt" in titulo_lower or "efetivo" in titulo_lower or "full-time" in
descricao_lower:
        return "Efetivo (CLT)"
    # Lógica adicional pode ser implementada aqui para inferir CLT se não
houver termos explícitos
    return "Não Categorizado"

def extrair_e_normalizar_habilidades(descricao_limpa):
    doc = nlp(descricao_limpa)
    habilidades_encontradas = []
    for habilidade_padrao in habilidades_tecnicas:
        if habilidade_padrao in descricao_limpa:
            habilidades_encontradas.append(habilidade_padrao)

    # Usar NER do spaCy para extrair entidades que podem ser habilidades
    # Isso requer um modelo treinado para identificar habilidades de forma
mais robusta
    # Exemplo simplificado: apenas para ilustrar a ideia
    for ent in doc.ents:
        # Em um cenário real, você teria uma lista de labels relevantes ou
um modelo customizado
        if ent.label_ in ["ORG", "PRODUCT", "LANGUAGE", "SKILL"] and
ent.text.lower() not in habilidades_encontradas:
            # Filtrar por termos relevantes e evitar duplicatas
            if len(ent.text) > 2 and ent.text.lower() not in ["e", "o", "a",
"de", "para"]:
                habilidades_encontradas.append(ent.text.lower())

    return list(set(habilidades_encontradas)) # Remover duplicatas e
retornar lista

def normalizar_salario(salario_str):
    if pd.isna(salario_str): return None, None
    salario_str = salario_str.lower().replace("r$", "").replace(".",
"").replace(",", ".").strip()
    min_salario, max_salario = None, None

    # Tentar extrair faixas salariais (ex: 3000 a 5000)
    match_range = re.search(r'(\d+\.?\d*)\s*(?:a|-)\s*(\d+\.?\d*)',
salario_str)
    if match_range:
```

```
        min_salario = float(match_range.group(1))
        max_salario = float(match_range.group(2))
    else:
        # Tentar extrair um único valor (ex: 3000)
        match_single = re.search(r'(\d+\.?\d*)', salario_str)
        if match_single:
            min_salario = float(match_single.group(1))
            max_salario = float(match_single.group(1))

    return min_salario, max_salario

# Exemplo de DataFrame (simulando dados brutos de vagas coletadas)
data = {
    'id_vaga': [1, 2, 3, 4],
    'titulo': [
        'Estágio em Desenvolvimento Python',
        'Desenvolvedor Fullstack Java/React CLT',
        'Jovem Aprendiz em TI',
        'Analista de Dados Sênior - Power BI e SQL'
    ],
    'descricao': [
        'Buscamos estudante de TI com conhecimento em Python e SQL. Vaga de
estágio em Fortaleza.',
        'Experiência com Spring Boot e React. Conhecimento em AWS e Docker.
Contratação CLT. Salário R$ 6.000 - 9.000.',
        'Oportunidade para primeiro emprego na área de TI. Não exige
experiência. Programa de jovem aprendiz.',
        'Analista de dados com experiência em Power BI, SQL e Big Data.
Conhecimento em Python é um diferencial. Salário 8k.'
    ],
    'localizacao': ['Fortaleza, CE', 'Fortaleza', 'Fortaleza - Ceará',
'Remoto (Fortaleza)']
}
df_vagas = pd.DataFrame(data)

# 1. Limpeza de Dados
df_vagas['descricao_limpa'] = df_vagas['descricao'].apply(limpar_texto)

# 2. Categorização de Vagas
df_vagas['categoria_vaga'] = df_vagas.apply(lambda row:
categorizar_vaga(row['titulo'], row['descricao']), axis=1)

# 3. Extração e Normalização de Habilidades
df_vagas['habilidades_extraidas'] =
df_vagas['descricao_limpa'].apply(extrair_e_normalizar_habilidades)

# 4. Normalização de Salários
df_vagas[['salario_min', 'salario_max']] = df_vagas['salario'].apply(lambda
```

```
x: pd.Series(normalizar_salario(x)) if 'salario' in df_vagas.columns else
pd.Series([None, None]))

# 5. Padronização de Localização (exemplo simples)
df_vagas['localizacao_padronizada'] = df_vagas['localizacao'].apply(lambda
x: 'Fortaleza, CE' if 'fortaleza' in x.lower() else x)

# Exibindo o DataFrame processado
print("\nDataFrame Processado:")
print(df_vagas[[
    'id_vaga', 'titulo', 'descricao_limpa', 'categoria_vaga',
    'habilidades_extraidas', 'salario_min', 'salario_max',
'localizacao_padronizada'
]].to_markdown(index=False))

# Exemplo de Agregação: Contagem de vagas por categoria
contagem_categorias = df_vagas['categoria_vaga'].value_counts().reset_index()
contagem_categorias.columns = ['Categoria', 'Numero_Vagas']
print("\nContagem de Vagas por Categoria:")
print(contagem_categorias.to_markdown(index=False))

# Exemplo de Agregação: Top N habilidades
lista_habilidades = [h for sublist in df_vagas['habilidades_extraidas'] for
h in sublist]
contagem_habilidades =
pd.DataFrame(Counter(lista_habilidades).most_common(5), columns=
['Habilidade', 'Frequencia'])
print("\nTop 5 Habilidades Mais Demandadas:")
print(contagem_habilidades.to_markdown(index=False))
```

Este exemplo conceitual demonstra como as etapas de limpeza, categorização, extração de habilidades e normalização de salários podem ser implementadas em Python. Em um ambiente de Big Data com PySpark, a lógica seria similar, mas aplicada a DataFrames distribuídos, aproveitando a capacidade de processamento paralelo do Spark.

5.4. Considerações de Qualidade de Dados e Monitoramento

- **Validação Contínua**: Implementar verificações de qualidade de dados em cada etapa do pipeline. Isso pode incluir a verificação de esquemas, a detecção de anomalias e a garantia de que os dados transformados atendam aos requisitos de negócios.
- **Testes Automatizados**: Escrever testes unitários e de integração para os scripts de processamento e transformação para garantir que as mudanças no código não introduzam erros.
- **Monitoramento**: Configurar ferramentas de monitoramento para acompanhar a execução dos jobs de processamento, o volume de dados processados e a qualidade

dos dados de saída. Alertas devem ser configurados para notificar a equipe em caso de falhas ou desvios significativos.

Ao seguir estas diretrizes, o projeto ASCENDE terá um pipeline de processamento e transformação de dados robusto e eficiente, capaz de converter dados brutos em insights acionáveis e confiáveis.

4.2. Coleta e Integração de Dados da RAIS

A RAIS (Relação Anual de Informações Sociais) é um registro administrativo que coleta informações sobre o vínculo empregatício no Brasil. Os dados são anuais e complementam o CAGED, fornecendo um retrato mais completo do emprego formal.

4.2.1. Localização dos Dados

Os dados da RAIS também são disponibilizados pelo MTE, muitas vezes na mesma seção de estatísticas ou em um portal dedicado. O acesso pode ser via: https://www.gov.br/trabalho-e-emprego/pt-br/acesso-a-informacao/dados-abertos/rais [2].

Os arquivos da RAIS são geralmente muito grandes e vêm em formato `.txt` ou `.csv` compactados por UF (Unidade da Federação) e ano.

4.2.2. Estratégia de Coleta com Python

O processo é muito similar ao do CAGED:

1. **Identificação da URL de Download**: Usar `requests` e `BeautifulSoup` para navegar na página da RAIS e identificar os links para os arquivos `.zip` ou `.txt` por ano e UF.
2. **Download e Descompactação**: Utilizar a função `download_file` e `unzip_file` (adaptadas) para baixar e descompactar os arquivos da RAIS para uma pasta específica no Data Lake (ex: `./data/raw/rais`).
3. **Leitura e Pré-processamento com Pandas**: Os arquivos da RAIS são geralmente `.txt` com um layout fixo ou delimitados por ponto e vírgula. A leitura exige atenção ao `sep` e `encoding`.
    - **Exemplo de Código (Leitura e Inspeção)**:
    - **Dicionário de Dados**: A RAIS possui um dicionário de dados extenso que é **fundamental** para interpretar as colunas. Este dicionário também deve ser baixado e armazenado.

4.3. Coleta e Integração de Dados da PNAD Contínua (IBGE)

A PNAD Contínua é uma pesquisa domiciliar que coleta informações sobre o mercado de trabalho, educação, renda e outras características demográficas. Ela fornece microdados

que permitem análises detalhadas sobre a força de trabalho, ocupação e desocupação.

4.3.1. Localização dos Dados

Os microdados da PNAD Contínua são disponibilizados no site do IBGE: https://www.ibge.gov.br/estatisticas/sociais/trabalho/17270-pnad-continua.html?t=microdados [3].

Os arquivos são geralmente compactados (ZIP) e contêm arquivos `.txt` ou `.csv` com os microdados, além de dicionários de variáveis e layouts de registro.

4.3.2. Estratégia de Coleta com Python

O processo é similar aos anteriores, mas com uma atenção especial aos dicionários de dados.

1. **Identificação da URL de Download**: Usar `requests` e `BeautifulSoup` para navegar na página da PNAD Contínua e identificar os links para os arquivos `.zip` dos microdados e seus respectivos dicionários de variáveis por trimestre/ano.
2. **Download e Descompactação**: Utilizar as funções de download e descompactação para baixar os arquivos para uma pasta específica no Data Lake (ex: `./data/raw/pnad`).
3. **Leitura e Pré-processamento com Pandas**: Os microdados da PNAD Contínua são geralmente arquivos `.txt` com layout fixo, o que exige uma leitura mais cuidadosa. O dicionário de variáveis é **indispensável** para saber a posição e o significado de cada campo.
    - **Exemplo de Código (Leitura de Layout Fixo)**:
    - **Dicionário de Variáveis**: O IBGE fornece dicionários de variáveis em PDF ou XLSX. É crucial tê-los para interpretar os códigos e valores nos microdados. Eles devem ser armazenados junto com os dados brutos.

4.4. Orquestração e Automação

Para todas as fontes governamentais, a orquestração com **Apache Airflow** ou **Prefect** é fundamental. Serão criados DAGs para:

- **CAGED**: Um DAG mensal para baixar o arquivo mais recente, descompactar e armazenar no Data Lake.
- **RAIS**: Um DAG anual para baixar os arquivos por UF e ano, descompactar e armazenar.
- **PNAD Contínua**: Um DAG trimestral/anual para baixar os microdados e seus dicionários, descompactar e armazenar.

Cada DAG deve incluir etapas para:

- **Verificação de Novas Versões**: Antes de baixar, verificar se há uma versão mais recente do arquivo disponível.
- **Download Robusto**: Com retries e tratamento de erros.
- **Armazenamento no Data Lake**: Salvar os arquivos brutos e descompactados em pastas organizadas por fonte, ano, mês/trimestre.
- **Notificação**: Enviar alertas em caso de falhas ou sucesso da operação.

4.5. Considerações Finais para Dados Governamentais

- **Documentação**: A documentação fornecida pelos órgãos (MTE, IBGE) é a sua melhor amiga. Ela detalha o formato dos arquivos, o significado das colunas e as codificações.
- **Volume de Dados**: Especialmente RAIS e PNAD Contínua podem gerar arquivos muito grandes. Esteja preparado para lidar com isso, utilizando `low_memory=False` no Pandas ou, para volumes muito grandes, considere usar PySpark para a leitura e processamento inicial.
- **Atualização Contínua**: Os portais governamentais podem mudar suas URLs ou formatos de disponibilização. Os scripts de coleta precisarão de manutenção periódica.
- **Licença de Uso**: Verifique sempre as licenças de uso dos dados para garantir a conformidade com os termos de redistribuição e análise.

Referências:

[1] Ministério do Trabalho e Emprego - Estatísticas CAGED. Disponível em: https://www.gov.br/trabalho-e-emprego/pt-br/servicos/empregador/caged/estatisticas [2] Ministério do Trabalho e Emprego - Dados Abertos RAIS. Disponível em: https://www.gov.br/trabalho-e-emprego/pt-br/acesso-a-informacao/dados-abertos/rais [3] IBGE - PNAD Contínua - Microdados. Disponível em: https://www.ibge.gov.br/estatisticas/sociais/trabalho/17270-pnad-continua.html?t=microdados