Documentação Sprint 3

Product Owner
Gustavo Rodrigo Morais Araújo - LinkedIn

Scrum Master
Mayara Carolina da Costa Gomes - LinkedIn

Team Members
Bruna Martins dos Santos - E-mail

Davi Augusto Nascimento Santos - LinkedIn

Juliana Alves Pascuti - LinkedIn

Marcos Vinícius Camargo Santana - LinkedIn

Samantha Giovanna Martins de Lima - LinkedIn

Professor Orientador
Marcus Vinicius do Nascimento

Professor Coorientador
Jean Carlos Lourenço Costa

Objetivo

Este projeto visa oferecer percepções valiosas e soluções para melhorar a eficiência operacional e a rentabilidade da empresa parceira.

- Análise de Produtividade das rotas.
- Criação e modelagem de um banco de dados em SQL.
- Criação de um visualizador de indicadores em BI.
- Aplicação do método de transportes para otimização da distribuição.

Cronograma

Projeto pedagógico fundamentado na Metodologia API (Aprendizagem por Projetos Integradores), visando o ensino e aprendizado. Ele se baseia nos pilares de aprendizado com problemas reais, focando no desenvolvimento de competências, validação externa e mentalidade ágil. Utilizamos estratégias para compreender o problema, conceber uma solução viável durante o desenvolvimento e implementar o MVP (Mínimo Produto Viável).

Link do GitHub

Sprint	Data	Status
Kick Off	13/03/2024	Concluído
1	17/04/2024	Concluída
2	08/05/2024	Concluída
3	29/05/2024	Concluída
4	19/06/2024	A fazer
Feira de Soluções	27/06/2024	A fazer

Mínimo Produto Viável – MVP

Sprint	MVP
1	Um **preview da amostra** de dados permitindo os primeiros inputs e uma análise preliminar das rotas.
2	Propõe-se a criação de um banco de dados dedicado à armazenagem, organização e controle de dados, visando garantir a integridade da base. Tal estrutura possibilitará consultas e exibições detalhadas, adaptáveis às necessidades do usuário. Em paralelo, será desenvolvido um visualizador no Power BI, direcionado aos principais KPI's identificados na sprint anterior.
3	Realização de uma análise detalhada da amostra, incluindo uma **avaliação mês a mês** para fornecer insights sobre as tendências e variações ao longo do tempo. Além disso, criação de um código em Python para identificar o melhor cenário visando a minimização dos custos da empresa, e estabelecimento de metas futuras da organização.
4	Ajustes necessários para alinhar o projeto com a realidade operacional da organização.

Tecnologias da Informação

O **GitHub** é uma plataforma em nuvem para hospedagem e colaboração de código-fonte, amplamente utilizada por desenvolvedores e equipes de projeto. Por meio dela, é possível gerenciar e compartilhar código, conduzir revisões, e rastrear problemas de forma eficiente. Os projetos são armazenados em repositórios, permitindo que os membros da equipe visualizem e editem o código de seus trabalhos.

O **Jira Software** é uma ferramenta de gerenciamento de projetos voltada para equipes de desenvolvimento de software. Ela é essencial no planejamento, gestão e acompanhamento de projetos, tanto ágeis quanto tradicionais.

O **Microsoft Excel** é uma planilha eletrônica amplamente utilizada para organizar, manipular e analisar dados numéricos e textuais. Nele, é possível criar planilhas distribuídas em colunas e linhas, realizar cálculos, criar gráficos e tabelas dinâmicas, além de utilizar uma variedade de funções e fórmulas para processar e analisar os dados.

O **Slack** é uma plataforma de comunicação empresarial desenvolvida para facilitar a conversação entre equipes de trabalho. Projetado para substituir e otimizar a comunicação por e-mail e outras formas internas. É essencial para empresas com equipes distribuídas ou trabalhando remotamente, proporcionando uma comunicação eficiente e colaborativa, independentemente da localização física dos membros da equipe.

O **Power BI** é uma plataforma de análise de negócios que permite aos usuários visualizarem e compartilharem compreensões a partir de dados de maneira rápida e fácil. Com diversas ferramentas para coletar, transformar, visualizar e compartilhar informações de fontes diferentes, locais ou na nuvem.

O **SQL** trata-se de uma linguagem de programação utilizada para gerenciar e manipular banco de dados relacionais. Ele permite que os usuários realizem diversas consultas e operações, como inserir, atualizar, excluir e recuperar dados. É uma ferramenta poderosa para lidar com grandes conjuntos de dados de maneira eficiente. A linguagem SQL é relativamente semelhante entre os principais Sistemas Gerenciadores de Banco de Dados (SGBDs) do mercado.

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral. É conhecida por sua sintaxe simples e legibilidade, o que a torna popular entre iniciantes e experientes em programação. Python é amplamente utilizado em uma variedade de domínios, como desenvolvimento web, análise de dados, inteligência artificial e automação de tarefas. Sua flexibilidade e facilidade de aprendizado contribuem para sua popularidade e ampla adoção em diferentes setores.

Integração do MySQL com Power BI

Os dados deste trabalho tendo sido inseridos em um banco de dados no MySQL permite que seja possível conectá-lo ao power bi para a criação de visualizações em relatórios interativos no power BI, proporcionando uma análise mais clara e detalhada dos dados.

Para realizar o processo de integração do banco de dados do MySQL com o Power BI, foi necessário um estudo sobre esta ação, a qual foi realizada através do passo a passo a seguir:

1. Instalar o conector MySQL-connector-net-8.4.0, essa ação é realizada para o Power BI entender de onde vem a conexão dos dados.
2. Após instalar o conector, foi selecionada a opção Banco de dados MySQL na seleção da obtenção da fonte de dados no Power BI.

3. Na caixa de diálogo banco de dados MySQL, foi inserido o nome do servidor e do banco de dados api_pbi.
4. Com a conexão realizada, é possível iniciar o desenvolvimento dos dashboards referente aos dados trabalhados no MySQL.

É notória a grande eficácia da junção destas ferramentas, a partir do banco de dados já conectado com Power bi é possível ter acesso em tempo real de todos os dados tratados no MySQL com muita rapidez, facilidade na hora de analisar os dados além de possuir uma visualização interativa.

Otimização das rotas utilizando Python

O desenvolvimento deste projeto tem como intuito fundamental a otimização do método de transportes efetuado entre as fábricas e os clientes atendidos por elas. O objetivo é a minimização do custo total de transporte, analisando as despesas nas rotas utilizadas, os gastos com frete por unidade transportada e despesas por km rodado desde a fábrica até o cliente. Verificando ainda a produtividade mensal que o veículo apresenta, buscando melhorar esse transporte em todos os aspectos, diminuindo quaisquer custos.

Para executar a otimização do custo total de transporte trabalhando-se com grande número de dados, utilizou-se a linguagem python, garantindo um processamento de alto volume de informações de maneira rápida e explicativa, assegurando uma análise detalhada de todo o processo.

Modelagem dos dados em Python

Para o desenvolvimento e realização do código em Python, foi utilizado o Google Colab, permitindo a visualização imediata e explicativa na ocorrência de erros no código, além de também assegurar o salvamento de todo o trabalho, graças à funcionalidade em nuvem do colab. A modelagem dos dados trabalhados para obter a otimização do custo de transporte foi estabelecida da seguinte maneira:

Passo 1

Inicialmente, foi necessário instalar bibliotecas específicas do python para que o código pudesse manipular e analisar dados, além de realizar operações matemáticas essenciais no tratamento

das informações. Estas bibliotecas foram: "Pandas" e "Numpy".

O Código para estabelecer acesso à estas bibliotecas foi o seguinte: "import pandas as pd" e "import numpy as np".

A terminologia adotada no final dos comandos, como "pd" e "np", facilitou o acesso às funções e objetos dessas bibliotecas na estruturação do código, possibilitando que seu uso fosse invocado utilizando essas abreviações, o que assegurou um código mais conciso e simples. Tendo como objetivo a minimização de custos, foi necessária a instalação da biblioteca: "Pulp", responsável por possibilitar a modelagem e resolução de problemas em pesquisa operacional. Sua instalação foi dada por: "!pip install pulp" fornecendo acesso às funções pertencentes a esta biblioteca.

Foi inserida a função: "from pulp import" responsável por importar todas as funções desta biblioteca para o código. Possibilitando o uso das funcionalidades desta biblioteca sem necessário adicionar: "pulp" no início de outro comando. Após isso, foi adicionado o comando: "from google.colab import files", permitindo a realização de upload e download de arquivos em ambientes como o Google colab.

Figura 1: Instalação das bibliotecas

1 Parte: Adicionando as bibliotecas, inserindo os dataframes e fazendo alguns tratamentos iniciais, como por exemplo, renomear algumas colunas e consolidar em um único arquivo.

```
[1] #Bibliotecas para rodar o código
    import pandas as pd
    import numpy as np

    !pip install pulp
    from pulp import * #Todas as funções da biblioteca

    from google.colab import files #Fazer o download do arquivo CSV
```

Passo 2

Em seguida, foi inserido comando para estabelecer o google drive no ambiente do colab, permitindo acesso e manipulação de quaisquer arquivos armazenados dentro do google drive diretamente no ambiente do colab.

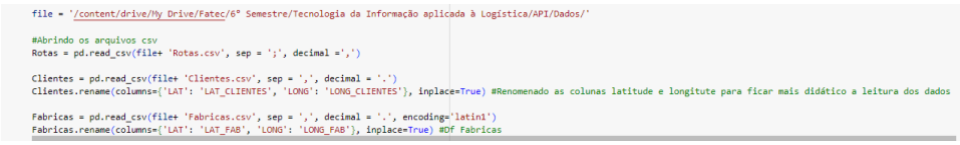
O comando foi:

"from google.colab import drive
drive.mount ('/content/drive')".

Figura 2: Acesso aos arquivos no drive

Foi criada uma variável nomeada: "file" para definir o caminho para a pasta onde os arquivos estão localizados no Google drive. Após isso, para realização da abertura dos arquivos csv no colab, importados pelo drive, foram estabelecidos DataFrame para cada um dos arquivos trabalhados. Sendo estes arquivos:
Rotas, clientes, fábricas, capacidade, demanda e custo ajustado.
Um DataFrame em python é uma estrutura de dados semelhante a uma tabela do Excel, onde existem linhas e colunas. Ele permite a leitura de dados de arquivos csv. O DataFrame permite também a filtragem de dados e realização de cálculos de forma organizada e fácil. O DataFrame foi escrito da seguinte forma: "df = pd.read_csv (file+ 'Rotas.csv', sep = ';', decimal = '.').
Este comando faz a leitura do arquivo csv e armazenamento dos dados deste arquivo em um DataFrame, nomeado como: "df", utilizando o; ';' como delimitador e o: '.' como separador decimal. Este comando foi modelado da mesma maneira para os outros 5 arquivos csv.

Figura 3: DataFrame dos arquivos CSV



Foi utilizada a função:

df2.rename(columns={'LAT': 'LAT_CLIENTES', 'LONG': 'LONG_CLIENTES'}, inplace=True)
Esta função possibilitou a renomeação das colunas: "Lat" e "Long" do DataFrame: "df2" para: "LAT_CLIENTES" e "LONG_CLIENTES".

Foi repetido o processo de renomeação para o DataFrame: "df3": df3.rename(columns={'LAT': 'LAT_FAB', 'LONG': 'LONG_FAB'}, inplace=True). Resultando em: "LAT_FAB" e "LONG_FAB".

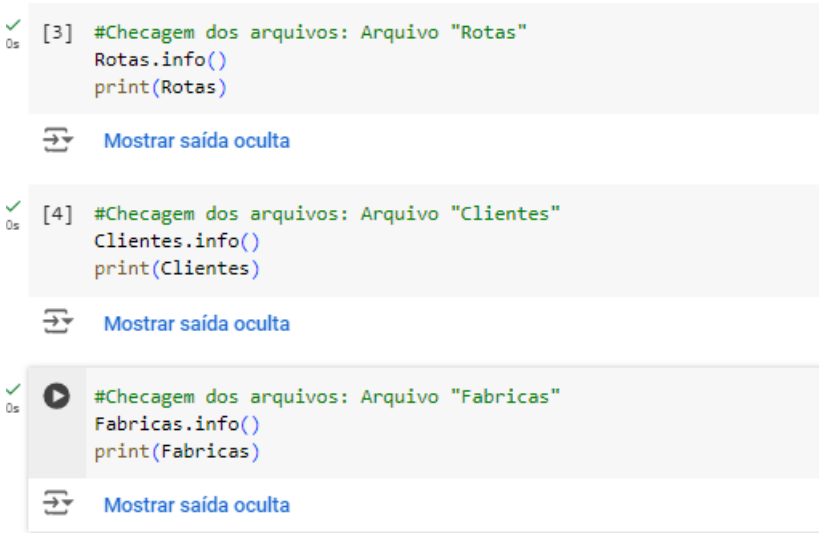
Passo 3

Após abertura de todos os arquivos a serem utilizados, foi inserido comando de verificação de cada documento. Estas funções foram inseridas da seguinte maneira: "df.info ( )" print (df).

Figura 4: Código de verificação

```
#Checagem dos arquivos: Arquivo "Rotas"
Rotas.info()
print(Rotas)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106262 entries, 0 to 106261
Data columns (total 13 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Dt.Emissao   106262 non-null  object
 1   Dt.Entrega   106262 non-null  object
 2   Mes.Base     106262 non-null  int64
 3   Ano.Exec     106262 non-null  int64
 4   CO.Fabrica   106262 non-null  int64
 5   CO.Cliente   106262 non-null  int64
 6   Incoterm     106262 non-null  object
 7   Veiculo      106262 non-null  object
 8   Qtd/pallets  106262 non-null  int64
 9   Qtd.Transp   106262 non-null  int64
 10  Moeda        106262 non-null  object
 11  Vlr.Frete    106262 non-null  object
 12  Dist         106262 non-null  float64
dtypes: float64(1), int64(6), object(6)
memory usage: 10.5+ MB
        Dt.Emissao  Dt.Entrega  Mes.Base  Ano.Exec  CO.Fabrica  CO.Cliente  \
0       01/01/2023  05/01/2023         1      2023     3423909        2311
1       01/01/2023  03/01/2023         1      2023     3424402        2333
2       01/01/2023  04/01/2023         1      2023     3403208        2347
3       01/01/2023  06/01/2023         1      2023     3424402        2332
```



Por meio dessa função, tornou-se possível visualizar um resumo conciso das informações relacionadas ao DataFrame, incluindo o número de entradas, o número de colunas, os tipos de dados em cada coluna etc. O comando "print (df)" foi empregado para exibir o DataFrame completo, sendo útil para análise detalhada dos dados ali presentes.

Passo 4

Para unificar informações em um único campo/base, foi empregado o uso da mesclagem dos DataFrames: "df", "df2" e "df3", formando assim, um novo DataFrame nomeado: "Base". Essa mesclagem ocorreu da seguinte forma:

Base = df.merge (df2, left_on='CO.CLIENTE',right_on='CO_CLIENTE',how='inner').merge(df3,left_on='CO.FABRICA',right_on='CO.FABRICA',how=inner')

Esta linha de código executa a mesclagem dos DataFrames: "df", "df2" e "df3" com base nas colunas 'CO.CLIENTE' e 'CO.FABRICA', utilizando a mesclagem interna ('inner join'). Isso significa que apenas as linhas com valores correspondentes em ambas as colunas serão incluídas no DataFrame resultante.

Figura 5: Mesclagem de DataFrame

```
[6] #Mesclar a base
    Base = Rotas.merge(Clientes, left_on='CO.Cliente',
                       right_on='CO.Cliente',
                       how='inner').merge(Fabricas, left_on='CO.Fabrica',
                                          right_on='CO.Fabrica',
                                          how='inner')
```

Ao término desta funcionalidade, foi adicionado código exibindo informações a respeito do DataFrame Base, como número de entradas e tipos de dados existentes nas colunas, sendo utilizado o seguinte comando: “base.info ( )”. Para visualização do DataFrame final, ou seja, DataFrame Base, foi adicionado o código: “base.head(5)”, demonstrando as primeiras 5 linhas do DataFrame Base.

Figura 6: Verificação do DataFrame Base

```
[7] # Checagem final
    Base.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106262 entries, 0 to 106261
Data columns (total 21 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   Dt.Emissao     106262 non-null  object
 1   Dt.Entrega     106262 non-null  object
 2   Mes.Base       106262 non-null  int64
 3   Ano.Exec       106262 non-null  int64
 4   CO.Fabrica     106262 non-null  int64
 5   CO.Cliente     106262 non-null  int64
 6   Incoterm       106262 non-null  object
 7   Veiculo        106262 non-null  object
 8   Qtd/pallets    106262 non-null  int64
 9   Qtd.Transp     106262 non-null  int64
 10  Moeda          106262 non-null  object
 11  Vlr.Frete      106262 non-null  object
 12  Dist           106262 non-null  float64
 13  MUN            106262 non-null  object
 14  LAT_CLIENTES   106262 non-null  float64
 15  LONG_CLIENTES  106262 non-null  float64
 16  NO_MUN         106262 non-null  object
 17  NO_MUN_MIN     106262 non-null  object
 18  SG_UF          106262 non-null  object
 19  LAT_FAB        106262 non-null  float64
 20  LONG_FAB       106262 non-null  float64
dtypes: float64(5), int64(6), object(10)
```

Passo 5

Como parte fundamental do objetivo em otimizar o método de transporte, foi estabelecido código condicional referente as capacidades dos veículos utilizados pela empresa. Essa função

foi inserida do seguinte modo:

Base [‘Capacidade’] = None
Base.loc [Base[Veículo’] == ‘p24’, ‘Capacidade’] = 3600
Base.loc[base[‘Veículo’] == ‘p12’, ‘Capacidade’] = 1800

Figura 7: Coluna condicional (capacidade dos veículos)

```
[ ]  #Cria nova: coluna condicional (Capacidade dos veículos)
     Base['Capacidade'] = None
     Base.loc[Base['Veiculo'] == 'P24', 'Capacidade'] = 3600
     Base.loc[Base['Veiculo'] == 'P12', 'Capacidade'] = 1800
```

O objetivo deste código é adicionar uma nova coluna chamada "Capacidade" ao DataFrame Base e preencher essa coluna com valores específicos dependendo do tipo de veículo presente na coluna "Veiculo". Para prosseguimento da otimização, foi realizado cálculo de produtividade, através da fórmula: “Base[‘produtividade’] = pd.to_numeric(base[‘Qtd.Transp’] / Base[‘Capacidade’] * 100).round(2)
print(Base)

Assim, calculando a produtividade que determinada quantidade transportada e escolha de véiculo apresentam, arredondando o resultado encontrado para duas casas decimais, as novas colunas adicionadas no DataFrame Base.

Figura 8: Cálculo de produtividade

```
[ ]  #Cria nova coluna: Produtividade
     Base['Produtividade'] = pd.to_numeric(Base['Qtd.Transp'] / Base['Capacidade'] * 100).round(2)
     print(Base)
```

Um outro cálculo necessário foi empregado ao código, nomeado como: “Frete/unidade” Essa operação serviu para analisar o custo presente nas rotas através da relação

“frete/unidade”, ou seja, o valor de frete dividido pela quantidade transportada. Essa nova coluna foi adicionada ao DataFrame Base. Este cálculo foi realizado da seguinte maneira: “Base[‘Frete/unidade’] = (Base[‘Vlr.Frete’] / Base[Qtd.Transp’].round(2)
Print(Base).

Figura 9: Cálculo de Frete/unidade

```
[14] #Adicionar coluna: Frete/unidade
     Base['Vlr.Frete'] = Base['Vlr.Frete'].astype(float)
     Base['Frete/unidade'] = (Base['Vlr.Frete'] / Base['Qtd.Transp']).round(2)

     pd.set_option('display.max_columns', None)
     print(Base)
```

```
        Dt.Emissao  Dt.Entrega  Mes.Base  Ano.Exec  CO.Fabrica  CO.Cliente  \
0       01/01/2023  05/01/2023         1      2023           1          11
1       02/01/2023  05/01/2023         1      2023           1          11
2       02/01/2023  05/01/2023         1      2023           1          11
3       02/01/2023  06/01/2023         1      2023           1          11
4       03/01/2023  07/01/2023         1      2023           1          11
...            ...         ...       ...       ...         ...         ...
106257  16/11/2023  21/11/2023        11      2023           3          36
106258  23/11/2023  28/11/2023        11      2023           3          36
106259  23/11/2023  28/11/2023        11      2023           3          36
106260  24/11/2023  27/11/2023        11      2023           3          36
106261  01/12/2023  05/12/2023        12      2023           3          36

       Incoterm Veiculo  Qtd/pallets  Qtd.Transp Moeda  Vlr.Frete    Dist  \
0           FOB     P24           24        3600   BRL       0.00   71.38
1           FOB     P12           12        1500   BRL       0.00   71.38
2           FOB     P24           24        3600   BRL       0.00   71.38
3           FOB     P24           24        3600   BRL       0.00   71.38
```

Passo 6

Foi utilizado um código com intuito de exportar o DataFrame Base para um arquivo CSV chamado: “Base_tratada.csv”, em seguida realizando um comando que realizou o dowload deste arquivo para o ambiente de execução atual, assim sendo feito seu salvamento no dispositivo. O parâmetro: “index=False” indica que o índice presente no DataFrame não será incluído no arquivo CSV resultante.

Figura 10: Exportar para CSV

```
# Exportar a base para CSV
Base.to_csv('Base_tratada.csv', index=False)
files.download('Base_tratada.csv')
```

Passo 7

Nesta etapa, a definição do problema a ser resolvido foi estabelecida no código.
Foi realizado um código padrão pertencente a biblioteca PuLP, utilizado para resolução de PO (Pesquisa Operacional). A estrutura do código foi realizada da seguinte forma: “problem = LpProblem(‘Minimização de custos’, LpMinimize)”. Este código foi escrito retratando o problema que está sendo resolvido, a de minimização de custos. Esta função é utilizada para criar um problema de programação linear ou inteira e define o tipo de problema, como: “minimização” ou “maximização”.

Figura 11: Definindo o problema

```
#Criando e definindo o tipo de problema
problem = LpProblem('Minimização de custos', LpMinimize)
```

Feito isso, foram inseridas as restrições do problema, escritas da seguinte forma:

```
“variables = {}
    for i in range(1, 4):  # Fábricas de 1 a 3
        for j in range(1, 52):  # Clientes de 1 a 51
            variable_name = f'X{i}{j}'
            variables[variable_name] = LpVariable(variable_name, lowBound=0)”
```

Neste código, o termo inicial; “variables = { }” é inserido para armazenar as váriaveis do problema. O termo “For i in range (1, 4)” faz o código percorrer cada número dentro do intervalo especificado de ‘i’ de 1 a 3, representando as 3 fábricas do problema. A linha: “for j in range(1, 52) faz o código percorrer também cada número dentro do intervalo especificado em sequência. O termo: “Variable_name = f’x{i} {j} serve para criar o nome da váriável combinando o índice da fábrica: “i” e o índice do cliente: “j”, tendo como exemplo: “para i = 1” e “j = 1” o nome da variável será: “X11”. A linha: “variables[variable_name] = LpVariable(variable_name, lowBound=0), cria uma variável de programação linear utilizando a biblioteca: “PuLP”. “LpVariable” se trata de uma função pertencente a esta biblioteca que cria uma variável com o nome especificado e um limite inferior, neste caso, sendo 0. A variável é adicionada ao dicionário “variables”, com o nome como chave e a variável como

valor. Após a elaboração das variáveis, o código efetua a verificação das primeiras variáveis criadas, sendo estas, para os primeiros 5 clientes de cada uma das fábricas, e as imprime para assegurar que tudo está devidamente correto. Isso é realizado pelos loops "for" aninhados, criando: "variable_name", em seguida, realiza a impressão do nome da variável e seu valor correspondente no dicionário: "variables". Esta parte do código serve para dar início e verificar as variáveis de decisão em um problema de programação linear.

Figura 12: Variáveis da PO

```
# Definir variáveis do problema
variables = {}
for i in range(1, 4):  # Fábricas de 1 a 3
    for j in range(1, 52):  # Clientes de 1 a 51
        variable_name = f'X{i}{j}'
        variables[variable_name] = LpVariable(variable_name, lowBound=0)

# Verificação das primeiras variáveis para garantir que tudo está correto
print("Primeiras variáveis:")
for i in range(1, 4):
    for j in range(1, 6):  # Verifica apenas os primeiros 5 clientes para evitar excesso de saída
        variable_name = f'X{i}{j}'
        print(f'{variable_name}: {variables[variable_name]}')
```

Foi colocado um código para verificação da existência de uma variável específica chamada: "variavel_procurada" no dicionário: "variables". Este código foi escrito da seguinte forma:

```
variavel_procurada = 'x126'
if variavel_procurada in variables:
    print(f'A variável {variavel_procurada} existe!')
else:
    print(f'A variável {variavel_procurada} não existe!')
```

Onde a linha: "variavel_procurada" define uma variável nomeada: "variavel_procurada" e atribui a ela o valor: "x126". Sendo este o nome da variável a ser verificada a cerca de sua existência no dicionário: "variables". O termo: "if variable_procurada in variables" é um código condicional que verifica se a: "variavel_procurada" está presente no dicionário: "variables". O operador "in" é utilizado para verificar a existência de uma chave no dicionário. A linha: "print(f" A variavel {variavel_procurada} existe!) realiza a impressão caso a variável exista. Se a variável procurada existir no dicionário: "variables", este código será executado e uma

mensagem será impressa avisando que a variável existe. O termo: “else” é um código condicional e é executado se a condição do “if” não for satisfeita, ou seja, se a variável procurada não existir no dicionário: “variables”. O termo: “print(f” A variavel {variavel_procurada} não existe!), o código é executado demonstrando uma mensagem indicando que a variável não existe no dicionário, caso a variável procurada não esteja presente no dicionário: “variables”.

Figura 13: Verificando a existência da variável

```
#Verificar se a variável existe
variavel_procurada = 'x126'
if variavel_procurada in variables:
    print(f'A variável {variavel_procurada} existe!')
else:
    print(f'A variável {variavel_procurada} não existe!')
```

Passo 8

A modelagem da função objetivo, ou, FO, foi desenvolvida da seguinte forma:

```
 # Adicionar a função objetivo ao problema
for _, row in df6.iterrows():
    variable_name = row['Xij']
    custo = row['Custo_ajustado']
  if variable_name in variables:
     problem += custo * variables[variable_name]

# Verificar a função objetivo
print("Função objetivo:")
print(problem)
```

Inicialmente, foi utilizado o loop: “for” que percorre as linhas do “df6”. Dentro do loop, duas variáveis são extraídas de cada linha do DataFrame, sendo essas: “variable_name”, que é a variável de decisão e “custo”, sendo este o custo associado a essa variável de decisão, conforme indicado pela coluna: “Custo_ajustado” no DataFrame. Seguidamente, a linha: “if

variable_name in variables", verifica se o nome da variável está presente no dicionário: "variables". Caso o nome da variável estiver presente no dicionário a função correspondente na função objetivo é construída. A função objetivo é composta pela soma dos custos ajustados de todas as variáveis de decisão presentes no DataFrame. Assim, a função objetivo foi adicionada ao problema de otimização: "problem" multiplicando o custo de cada variável de decisão pelo seu respectivo coeficiente e adicionando essa expressão à função objetivo. Por fim, foi expressa a impressão da função objetivo construída até o momento.

Figura 14: Função Objetivo

```
# Adicionar a função objetivo ao problema
for _, row in df6.iterrows():
    variable_name = row['Xij']
    custo = row['Custo_ajustado']
    if variable_name in variables:
        problem += custo * variables[variable_name]

# Verificar a função objetivo
print("Função objetivo:")
print(problem)
```

Passo 9

Para construção das restrições do problema, o código foi escrito da seguinte forma:

```
# Variável para rastrear os nomes exclusivos das restrições
constraint_counter = {}

# Restrições de Capacidade das Fábricas
for fabrica_id, capacidade in zip(df4['CO.Fabrica'], df4['Capacidade']):
    constraint_name = f"Capacidade_Fabrica_{fabrica_id}"
    for xij in Base_xij['xij']:
        if str(fabrica_id) in xij:
      if xij in variables:
            # Verificar se já existe uma restrição com o mesmo nome
            if constraint_name not in constraint_counter:
                constraint_counter[constraint_name] = 1
```

```
                else:
                    constraint_counter[constraint_name] += 1
                # Adicionar um sufixo único ao nome da restrição
                constraint_name_unique =
f"{constraint_name}_{constraint_counter[constraint_name]}"
                problem += variables[xij] <= capacidade, constraint_name_unique
            else:
                print(f"A variável {xij} não foi encontrada no dicionário de variáveis.")

    # Restrições de Demanda dos Clientes
    for cliente, demanda in zip(df5['CO.Cliente'], df5['Demanda']):
        constraint_name = f"Demanda_Cliente_{cliente}"
        if any(str(cliente) in xij for xij in Base_xij['xij'].values):
        problem += lpSum(variables[xij] for xij in Base_xij['xij'] if str(cliente) in xij) ==
demanda, constraint_name
        else:
            print(f"A variável {xij} não foi encontrada no dicionário de variáveis para atender
a demanda do cliente {cliente}.")
```

Onde a linha: “constraint_counter” trata-se de um dicionário que foi utilizado para rastrear quantas vezes uma determinada restrição apareceu. Seguidamente, na linha: “for fabrica_id, capacidade in zip(df4[‘CO.Fabrica’], df4[‘Capacidade]) percorre sobre os valores de “CO.Fabrica” e “Capacidade” do DataFrame: “df4”, percorrendo cada fábrica e sua capacidade correspondente. O código: “constraint_name = f”Capacidade_Fabrica_{fabrica_id}, cria um nome para a restrição de capacidade da fábrica atual, combinando o prefixo: “Capacidade_Fabrica_ com o ID da fábrica. A linha: for xij in Base_xij [‘xij’] trata-se de um loop aninhado que percorre sobre os valores de "xij” em “Base_xij” [‘xij’]. O código: “if str(fabrica_id) in xij” realiza a verificação se o ID da fábrica atual está presente na variável de decisão: “xij”. A linha: “if xij in variables” verifica se a variável de decisão: “xij” está presente no dicionário: “variables”. O código: “ if constraint_name not in constraint_counter:

Constraint_counter [constraint_name] = 1

else:

constraint_counter[constraint_name] += 1, realizam a atualização do contador de

restrições: “constraint_counter” assegurando que cada restrição tenha um nome único. Caso a restrição já exista no dicionário, incrementa o contador. O termo: “constraint_name_unique = f”{constraint_name}_{constraint_counter[constrain_name]}, cria um nome único para a restrição combinando o nome original com o valor atual do contador. A linha: “problem += variables[xij] <= capacidade, constraint_name_unique”, adiciona uma restrição ao problema de otimização, garantindo que a variável de decisão: “xij” associada à capacidade da fábrica atual não exceda a capacidade especificada. O código: “else: print (f”A variável {xij} não foi encontrada no dicionário de variáveis.”, realiza a impressão de uma mensagem de aviso caso a variável de decisão: “xij” não estiver presente no dicionário: “variables”. Na linha: “for cliente, demanda in zip(df5[‘CO.Cliente’], df5[‘Demanda], o código percorre sobre os valores de: “CO.Cliente” e “Demanda” do DataFrame: “df5”. No termo: “constraint_name = f”Demana_Cliente_ {cliente}”, a linha cria um nome para a restrição de demanda do cliente atual, combinando o prefixo: “Demanda_Cliente_” com o ID do cliente. Na linha: “if any(str(cliente) in xij for xij in Base_xij[‘xij’] if str(cliente) in xij) == demanda, constraint_name, foi adicionado uma restrição ao problema de otimização. Garantindo que a soma das variáveis de decisão associadas ao cliente atual seja igual à demanda especificada para esse cliente. O código: “else: print (f”A variável {xij} não foi encontrada no dicionário de variáveis para atender a demanda do cliente {cliente}. “), foi usado para exibir uma mensagem de aviso caso a variável de decisão associada a um cliente não for encontrada.

Figura 15: Criação das restrições do problema

```
# Variável para rastrear os nomes exclusivos das restrições
constraint_counter = {}

# Restrições de Capacidade das Fábricas
for fabrica_id, capacidade in zip(df4['CO.Fabrica'], df4['Capacidade']):
    constraint_name = f"Capacidade_Fabrica_{fabrica_id}"
    for xij in Base_xij['xij']:
        if str(fabrica_id) in xij:
            if xij in variables:
                # Verificar se já existe uma restrição com o mesmo nome
                if constraint_name not in constraint_counter:
                    constraint_counter[constraint_name] = 1
                else:
                    constraint_counter[constraint_name] += 1
                # Adicionar um sufixo único ao nome da restrição
                constraint_name_unique = f"{constraint_name}_{constraint_counter[constraint_name]}"
                problem += variables[xij] <= capacidade, constraint_name_unique
            else:
                print(f"A variável {xij} não foi encontrada no dicionário de variáveis.")

# Restrições de Demanda dos Clientes
for cliente, demanda in zip(df5['CO.Cliente'], df5['Demanda']):
    constraint_name = f"Demanda_Cliente_{cliente}"
    if any(str(cliente) in xij for xij in Base_xij['xij'].values):
        problem += lpSum(variables[xij] for xij in Base_xij['xij'] if str(cliente) in xij) == demanda, constraint_name
    else:
        print(f"A variável {xij} não foi encontrada no dicionário de variáveis para atender a demanda do cliente {cliente}.")
```

Passo 10

Para realização da solução do problema foi empregado o seguinte código:

#Resolver o problema (exemplo sem restrições)

```
problem.solve()
```

Esta linha executou a solução do problema de otimização nomeado: “problem”. O algoritimo correspondente foi executado com base nos parâmetros e configurações do problema, buscando solução ótima de acordo com os critérios definidos no problema, sendo neste caso, de minimização. O termo: “print(f’Status da solução: {LpProblem.solve}”) foi empregado para imprimir uma mensagem na tela a respeito do status da solução. Foi usado uma f-string para incluir o resultado da chamada: “LpProblem.solve” dentro da string. A linha: “for v in problem. Variables( ):

print(f”{v.name} = {v.varValue}”), cria um loop que percorre sobre todas as variáveis de decisão do problema: “problem”. A segunda linha desta parte do código (print), foi usada para imprimir o nome e valor de cada variável de decisão, onde “v.name” é o nome da variável e “v.varValue” é o valor atribuido à variável após a solução do problema. Por fim, foi utilizado o código: “print(f’Valor da função objetivo: {problem.objective.value( )}”) para imprimir na tela o valor atual da função objetivo do problema de otimização definido como: “problem”.

Figura 16: Resolução do problema

```
#Resolver o problema (exemplo sem restrições)
problem.solve()

#Verificar o status da solução
print(f"Status da solução: {LpProblem.solve}")

# Exibir os valores das variáveis de decisão
for v in problem.variables():
    print(f"{v.name} = {v.varValue}")

# Exibir o valor da função objetivo
print(f"Valor da função objetivo: {problem.objective.value()}")
```