

Transformando a tabela RideAdress em um sample de 1000 linhas.

```
In [2]: import pandas as pd

# Carregar o arquivo completo
arquivo = 'rideaddress_v1_tratado_sem_desconhecido.csv'
df = pd.read_csv(arquivo, delimiter=',', dtype={'Lat': str, 'Lng': str}, on_bad_

#Limpar os nomes das colunas para evitar espaços ou caracteres ocultos
df.columns = df.columns.str.strip()

#Limpar valores incorretos e converter para float
df['Lat'] = pd.to_numeric(df['Lat'].str.replace(',', '.'), errors='coerce')
df['Lng'] = pd.to_numeric(df['Lng'].str.replace(',', '.'), errors='coerce')

#Remover linhas com valores nulos em Lat e Lng
df = df.dropna(subset=['Lat', 'Lng'])

#Criar uma amostra aleatória de 1.000 linhas
amostra = df.sample(n=1000, random_state=42)

#Salvar o novo arquivo com a amostra
amostra.to_csv('rideaddress_v1_sample.csv', index=False)
print("Arquivo 'rideaddress_v1_sample.csv' salvo com sucesso!")
```

Arquivo 'rideaddress_v1_sample.csv' salvo com sucesso!

BUSCA GULOSA - TABELA RIDEADDRESS (Os locais a partir da latitude e longitude para verificar os lugares mais proximo de são paulo(Centro de Sp))

```
In [3]: import pandas as pd
import numpy as np

# Carregar a amostra com 1.000 linhas
arquivo = 'rideaddress_v1_sample.csv'
df = pd.read_csv(arquivo)

#Garantir que Lat e Lng estejam como float
df['Lat'] = pd.to_numeric(df['Lat'], errors='coerce')
df['Lng'] = pd.to_numeric(df['Lng'], errors='coerce')

#Remover linhas com valores nulos em Lat e Lng após conversão
df = df.dropna(subset=['Lat', 'Lng'])

# Ponto inicial (exemplo) – São Paulo
start_lat, start_lng = -23.550520, -46.633308

#Função para calcular a distância euclidiana entre dois pontos (aproximação)
def calcular_distancia(lat1, lng1, lat2, lng2):
    return np.sqrt((lat1 - lat2) ** 2 + (lng1 - lng2) ** 2)

#Algoritmo de busca gulosa
def busca_gulosa(lat_inicio, lng_inicio):
    df_copy = df.copy()
    caminho = []

    while not df_copy.empty:
        #Calcula a distância para cada ponto
```

```
df_copy['Distancia'] = calcular_distancia(lat_inicio, lng_inicio, df_cop

#Seleciona o ponto mais próximo (menor distância)
ponto_mais_proximo = df_copy.loc[df_copy['Distancia'].idxmin()]

#Adiciona ao caminho
caminho.append(ponto_mais_proximo)

# Atualiza o ponto inicial para o novo ponto escolhido
lat_inicio, lng_inicio = ponto_mais_proximo['Lat'], ponto_mais_proximo['

# Remove o ponto escolhido da lista
df_copy = df_copy.drop(ponto_mais_proximo.name)

# Retorna apenas colunas relevantes
return pd.DataFrame(caminho)[['Neighborhood', 'City', 'State', 'Lat', 'Lng']]

#Executar a busca gulosa
resultado_busca = busca_gulosa(start_lat, start_lng)

#Mostrar o resultado
print("\nResultado da Busca Gulosa:")
print(resultado_busca.head(20))

#Salvar o resultado em um novo CSV para análise futura (opcional)
resultado_busca.to_csv('resultado_busca_gulosa.csv', index=False)
print("\nArquivo 'resultado_busca_gulosa.csv' salvo com sucesso!")
```

Resultado da Busca Gulosa:

	Neighborhood	City \
925	Centro Histórico de São Paulo	São Paulo
332	Bela Vista	São Paulo
156	Bela Vista	São Paulo
376	R. Maria José, 475 - Bela Vista, São Paulo - S...	Bela Vista
655	R. Maria José, 475 - Bela Vista, São Paulo - S...	Bela Vista
452	Bela Vista	São Paulo
392	R. Martiniano de Carvalho, 969 - Bela Vista, S...	Bela Vista
712	Bela Vista	São Paulo
295	Av. Paulista - Bela Vista, São Paulo - SP, Brasil	São Paulo
791	Av. Paulista - Bela Vista, São Paulo - SP, Brasil	São Paulo
711	Alameda Santos, 721 - Jardim Paulista, São Pau...	Jardim Paulista
275	Jardim Paulista	São Paulo
962	Jardim Paulista	São Paulo
17	Jardim Paulista	São Paulo
743	R. José Maria Lisboa, 129 - Jardim Paulista, S...	Jardim Paulista
829	R. Cap. Pinto Ferreira, 62 - Jardim Paulista, ...	Jardim Paulista
735	Alameda Lorena, 521 - Jardim Paulista, São Pau...	Jardim Paulista
885	Jardim Paulista	São Paulo
482	Jardins	São Paulo
506	Jardim Paulista	São Paulo


	State	Lat	Lng
925	São Paulo	-23.546124	-46.635958
332	São Paulo	-23.559093	-46.642776
156	São Paulo	-23.559108	-46.642835
376	São Paulo	-23.559290	-46.643034
655	São Paulo	-23.559290	-46.643034
452	São Paulo	-23.565749	-46.642474
392	São Paulo	-23.567417	-46.643073
712	São Paulo	-23.565834	-46.650838
295	SP	-23.565739	-46.651238
791	SP	-23.565739	-46.651238
711	São Paulo	-23.568269	-46.651086
275	São Paulo	-23.567819	-46.653171
962	São Paulo	-23.567819	-46.653171
17	São Paulo	-23.567876	-46.653202
743	São Paulo	-23.571311	-46.654621
829	São Paulo	-23.571470	-46.655028
735	São Paulo	-23.570614	-46.659411
885	São Paulo	-23.569375	-46.659116
482	São Paulo	-23.569253	-46.659339
506	São Paulo	-23.569143	-46.659405


Arquivo 'resultado_busca_gulosa.csv' salvo com sucesso!


BUSCA GULOSA - TABELA RIDEESTIMATIVE_V3 (Verificando os Valores mais comuns)\



```
In [4]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# 1 Carregar o arquivo CSV completo
arquivo = 'rideestimative_v3_limpo.csv'
df = pd.read_csv(arquivo, delimiter=';', low_memory=False)
print("✅ Arquivo carregado com sucesso!")
```


```
#  Exibir valores nulos antes do tratamento
print("\n=====")
print("🔥 Valores Nulos ANTES do tratamento:")
print(df.isnull().sum())


#  Limpar valores incorretos e garantir que as colunas estejam como float
df['Price'] = pd.to_numeric(df['Price'], errors='coerce')
df['WaitingTime'] = pd.to_numeric(df['WaitingTime'], errors='coerce')
df['Fee'] = pd.to_numeric(df['Fee'], errors='coerce')

#  Remover valores nulos em colunas importantes
df = df.dropna(subset=['Price', 'WaitingTime', 'Fee'])

#  Criar uma amostra aleatória de 1.000 linhas para simplificar o processamento
amostra = df.sample(n=1000, random_state=42)
amostra.to_csv('rideestimative_v3_sample.csv', index=False)
print("\n Arquivo 'rideestimative_v3_sample.csv' salvo com sucesso!")

#  Ponto inicial (exemplo) – Preço médio como ponto de partida
start_price = amostra['Price'].mean()

#  Função para calcular distância entre dois preços (valor absoluto)
def calcular_distancia(price1, price2):
    return abs(price1 - price2)

#  Algoritmo de Busca Gulosa
def busca_gulosa_estimativa(preco_inicial):
    df_copy = amostra.copy()
    caminho = []

    while not df_copy.empty:
        # Calcula a distância com base em preço
        df_copy['Distancia'] = calcular_distancia(preco_inicial, df_copy['Price'])


        # Seleciona o ponto mais próximo (menor distância)
        ponto_mais_proximo = df_copy.loc[df_copy['Distancia'].idxmin()]



        # Adiciona ao caminho
        caminho.append(ponto_mais_proximo)



        # Atualiza o preço inicial para o novo ponto escolhido
        preco_inicial = ponto_mais_proximo['Price']

        # Remove o ponto escolhido da lista
        df_copy = df_copy.drop(ponto_mais_proximo.name)

    # Retorna apenas as colunas relevantes para análise
    return pd.DataFrame(caminho)[['RideID', 'ProductID', 'Price', 'WaitingTime', 'Fee']]

#  Executar o algoritmo de busca gulosa
resultado_estimativa = busca_gulosa_estimativa(start_price)

#  Mostrar os primeiros resultados
print("\n Resultado da Busca Gulosa na Tabela RideEstimative:")
print(resultado_estimativa.head(20))

#  Salvar os resultados em CSV
resultado_estimativa.to_csv('resultado_busca_gulosa_estimative.csv', index=False)
print("\n Arquivo 'resultado_busca_gulosa_estimative.csv' salvo com sucesso!")
```

```
# 🟢 Selecionar os três preços mais comuns para visualização
top_3_prices = resultado_estimativa['Price'].value_counts().nlargest(3)
print("\n📊 Três preços mais comuns após busca gulosa:")
print(top_3_prices)

# 📊 **Gerar gráfico com os três preços mais comuns**
plt.figure(figsize=(8, 5))

# Criar gráfico de barras
bars = plt.bar(top_3_prices.index, top_3_prices.values, color="#1f77b4")

# Adicionar valores sobre as barras
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, yval + 5, int(yval), ha='center')

# Melhorias visuais
plt.title("Três Preços Mais Frequentes Após Busca Gulosa", fontsize=16, pad=15)
plt.xlabel("Preço (R$)", fontsize=12)
plt.ylabel("Quantidade de Corridas", fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.ylim(0, top_3_prices.max() + 50)
plt.tight_layout()

# Exibir o gráfico
plt.show()
```

✅ Arquivo carregado com sucesso!

=====

🚩 Valores Nulos ANTES do tratamento:

RideEstimativeID	0
RideID	0
ProductID	0
WaitingTime	0
Price	0
FareID	1
Selected	1
RideReasonSelectedEstimativeID	1
Fee	1

dtype: int64

✅ Arquivo 'rideestimative_v3_sample.csv' salvo com sucesso!

✅ Resultado da Busca Gulosa na Tabela RideEstimative:

	RideID	ProductID	Price	WaitingTime	Fee
17911	1185432	Comfort	36.00	5	0.0
68989	1191923	UberX	36.00	3	0.0
24231	1186253	turbo-taxi	35.96	7	0.0
126288	1199507	pop99	35.96	5	0.0
59064	1190742	regular-taxi	35.84	7	0.0
19918	1185676	pop99	35.77	5	0.0
228252	1212793	regular-taxi	35.66	7	0.0
66529	1191630	UberX	35.50	4	0.0
113103	1197602	Bag	35.50	7	0.0
68394	1191856	Comfort	35.50	6	0.0
206513	1210089	Comfort	35.50	6	0.0
93609	1195065	UberX Promo	35.50	11	0.0
77336	1193033	Flash	35.50	2	0.0
219018	1211578	turbo-taxi	35.16	6	0.0
16732	1185289	UberFlash	35.00	5	0.0
63352	1191258	Uber Promo	35.00	3	0.0
150727	1202608	regular-taxi	34.91	7	0.0
139907	1201154	pop99	34.90	9	0.0
65986	1191564	pop99	34.67	5	0.0
1635	1183430	turbo-taxi	34.55	7	0.0

✅ Arquivo 'resultado_busca_gulosa_estimative.csv' salvo com sucesso!

🇧🇷 Três preços mais comuns após busca gulosa:

Price	
11.5	26
10.5	26
19.0	20

Name: count, dtype: int64

