


Transformando a tabela RideAdress em um sample de 1000 linhas.

```

1 import pandas as pd
2
3 # Carregar o arquivo completo
4 arquivo = 'rideaddress_v1_tratado_sem_desconhecido.csv'
5 df = pd.read_csv(arquivo, delimiter=',', dtype={'Lat': str, 'Lng': str}, on_bad_lines='skip')
6
7 #Limpar os nomes das colunas para evitar espaços ou caracteres ocultos
8 df.columns = df.columns.str.strip()
9
10 #Limpar valores incorretos e converter para float
11 df['Lat'] = pd.to_numeric(df['Lat'].str.replace(',', '.'), errors='coerce')
12 df['Lng'] = pd.to_numeric(df['Lng'].str.replace(',', '.'), errors='coerce')
13
14 #Remover linhas com valores nulos em Lat e Lng
15 df = df.dropna(subset=['Lat', 'Lng'])
16
17 #Criar uma amostra aleatória de 1.000 linhas
18 amostra = df.sample(n=1000, random_state=42)
19
20 #Salvar o novo arquivo com a amostra
21 amostra.to_csv('rideaddress_v1_sample.csv', index=False)
22 print("Arquivo 'rideaddress_v1_sample.csv' salvo com sucesso!")
23

```

 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))

```

1 import pandas as pd
2 import numpy as np
3
4 # Carregar a amostra com 1.000 linhas
5 arquivo = 'rideaddress_v1_sample.csv'
6 df = pd.read_csv(arquivo)
7
8 #Garantir que Lat e Lng estejam como float
9 df['Lat'] = pd.to_numeric(df['Lat'], errors='coerce')
10 df['Lng'] = pd.to_numeric(df['Lng'], errors='coerce')
11
12 #Remover linhas com valores nulos em Lat e Lng após conversão
13 df = df.dropna(subset=['Lat', 'Lng'])
14
15 # Ponto inicial (exemplo) – São Paulo
16 start_lat, start_lng = -23.550520, -46.633308
17
18 #Função para calcular a distância euclidiana entre dois pontos (aproximação)
19 def calcular_distancia(lat1, lng1, lat2, lng2):
20     return np.sqrt((lat1 - lat2) ** 2 + (lng1 - lng2) ** 2)
21
22 #Algoritmo de busca gulosa
23 def busca_gulosa(lat_inicio, lng_inicio):
24     df_copy = df.copy()
25     caminho = []
26
27     while not df_copy.empty:
28         #Calcula a distância para cada ponto
29         df_copy['Distancia'] = calcular_distancia(lat_inicio, lng_inicio, df_copy['Lat'], df_copy['Lng'])
30
31         #Seleciona o ponto mais próximo (menor distância)
32         ponto_mais_proximo = df_copy.loc[df_copy['Distancia'].idxmin()]
33
34         #Adiciona ao caminho
35         caminho.append(ponto_mais_proximo)
36
37         # Atualiza o ponto inicial para o novo ponto escolhido
38         lat_inicio, lng_inicio = ponto_mais_proximo['Lat'], ponto_mais_proximo['Lng']
39

```

```

40     # Remove o ponto escolhido da lista
41     df_copy = df_copy.drop(ponto_mais_proximo.name)
42
43     # Retorna apenas colunas relevantes
44     return pd.DataFrame(caminho[['Neighborhood', 'City', 'State', 'Lat', 'Lng']])
45
46 #Executar a busca gulosa
47 resultado_busca = busca_gulosa(start_lat, start_lng)
48
49 #Mostrar o resultado
50 print("\nResultado da Busca Gulosa:")
51 print(resultado_busca.head(20))
52
53 #Salvar o resultado em um novo CSV para análise futura (opcional)
54 resultado_busca.to_csv('resultado_busca_gulosa.csv', index=False)
55 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)\

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # 1 Carregar o arquivo CSV completo
6 arquivo = 'rideestimative_v3_limpo.csv'
7 df = pd.read_csv(arquivo, delimiter=';', low_memory=False)
8 print("✅ Arquivo carregado com sucesso!")
9
10 # ✅ Exibir valores nulos antes do tratamento

```

```


11 print("\n=====")
12 print("🔴 Valores Nulos ANTES do tratamento:")
13 print(df.isnull().sum())
14
15 # ✅ Limpar valores incorretos e garantir que as colunas estejam como float
16 df['Price'] = pd.to_numeric(df['Price'], errors='coerce')
17 df['WaitingTime'] = pd.to_numeric(df['WaitingTime'], errors='coerce')
18 df['Fee'] = pd.to_numeric(df['Fee'], errors='coerce')
19
20 # ✅ Remover valores nulos em colunas importantes
21 df = df.dropna(subset=['Price', 'WaitingTime', 'Fee'])
22
23 # 2 Criar uma amostra aleatória de 1.000 linhas para simplificar o processamento
24 amostra = df.sample(n=1000, random_state=42)
25 amostra.to_csv('rideestimative_v3_sample.csv', index=False)
26 print("\n✅ Arquivo 'rideestimative_v3_sample.csv' salvo com sucesso!")
27
28 # ✅ Ponto inicial (exemplo) – Preço médio como ponto de partida
29 start_price = amostra['Price'].mean()
30
31 # ✅ Função para calcular distância entre dois preços (valor absoluto)
32 def calcular_distancia(price1, price2):
33     return abs(price1 - price2)
34
35 # ✅ Algoritmo de Busca Gulosa
36 def busca_gulosa_estimativa(preco_inicial):
37     df_copy = amostra.copy()
38     caminho = []
39
40     while not df_copy.empty:
41         # Calcula a distância com base em preço
42         df_copy['Distancia'] = calcular_distancia(preco_inicial, df_copy['Price'])
43
44         # Seleciona o ponto mais próximo (menor distância)
45         ponto_mais_proximo = df_copy.loc[df_copy['Distancia'].idxmin()]
46
47         # Adiciona ao caminho
48         caminho.append(ponto_mais_proximo)
49
50         # Atualiza o preço inicial para o novo ponto escolhido
51         preco_inicial = ponto_mais_proximo['Price']
52
53         # Remove o ponto escolhido da lista
54         df_copy = df_copy.drop(ponto_mais_proximo.name)
55
56     # Retorna apenas as colunas relevantes para análise
57     return pd.DataFrame(caminho)[['RideID', 'ProductID', 'Price', 'WaitingTime', 'Fee']]
58
59 # 3 Executar o algoritmo de busca gulosa
60 resultado_estimativa = busca_gulosa_estimativa(start_price)
61
62 # ✅ Mostrar os primeiros resultados
63 print("\n✅ Resultado da Busca Gulosa na Tabela RideEstimative:")
64 print(resultado_estimativa.head(20))
65
66 # ✅ Salvar os resultados em CSV
67 resultado_estimativa.to_csv('resultado_busca_gulosa_estimative.csv', index=False)
68 print("\n✅ Arquivo 'resultado_busca_gulosa_estimative.csv' salvo com sucesso!")
69
70 # ✅ Selecionar os três preços mais comuns para visualização
71 top_3_prices = resultado_estimativa['Price'].value_counts().nlargest(3)
72 print("\n📊 Três preços mais comuns após busca gulosa:")
73 print(top_3_prices)
74
75 # 📊 **Gerar gráfico com os três preços mais comuns**
76 plt.figure(figsize=(8, 5))
77
78 # Criar gráfico de barras
79 bars = plt.bar(top_3_prices.index, top_3_prices.values, color="#1f77b4")
80
81 # Adicionar valores sobre as barras
82 for bar in bars:
83     yval = bar.get_height()
84     plt.text(bar.get_x() + bar.get_width() / 2, yval + 5, int(yval), ha='center', fontsize=12)


```

```
85
86 # Melhorias visuais
87 plt.title("Três Preços Mais Frequentes Após Busca Gulosa", fontsize=16, pad=15)
88 plt.xlabel("Preço (R$)", fontsize=12)
89 plt.ylabel("Quantidade de Corridas", fontsize=12)
90 plt.xticks(rotation=45, ha='right')
91 plt.grid(axis="y", linestyle="--", alpha=0.7)
92 plt.ylim(0, top_3_prices.max() + 50)
93 plt.tight_layout()
94
95 # Exibir o gráfico
96 plt.show()
97
```

 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