

✓ **GERANDO A TABELA**

```
1 pip install geopy
```

⇒ Requirement already satisfied: geopy in /usr/local/lib/python3.11/dist-packages (2.4.1)
Requirement already satisfied: geographiclib<3,>=1.52 in /usr/local/lib/python3.11/dist-packages (from geopy) (2.0)

```
1 import pandas as pd
2 from geopy.distance import geodesic
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.model_selection import train_test_split
5 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
6 import numpy as np
7
8 # 1. Carregamento dos arquivos
9 dfRide = pd.read_csv("ride_v2.csv", sep=";", dtype=str)
10 dfRideAdd = pd.read_csv("rideaddress_v1.csv", sep=";", dtype=str)
11 dfRideEst = pd.read_csv("rideestimative_v3.csv", sep=";", dtype=str)
12 dfProduct = pd.read_csv("product.csv", sep=";", dtype=str)
13
14 # 2. Uniformização: datas e RideID
15 dfRide["Schedule"] = pd.to_datetime(dfRide["Schedule"], errors="coerce")
16 for df in [dfRide, dfRideAdd, dfRideEst]:
17     df["RideID"] = df["RideID"].astype(str).str.replace(".0", "", regex=False)
18
19 # 3. Derivar colunas de tempo
20 dfRide["Dia"] = dfRide["Schedule"].dt.weekday
21 dfRide["Hora"] = dfRide["Schedule"].dt.hour
22 dfRide["Minuto"] = dfRide["Schedule"].dt.minute
23 dfRide["HoraDecimal"] = dfRide["Hora"] + dfRide["Minuto"] / 60
24 dfRide["Faixa15min"] = dfRide["Schedule"].dt.floor("15min")
25 dfTempo = dfRide[["RideID", "Dia", "Hora", "Minuto", "HoraDecimal", "Faixa15min"]].dropna()
26
27 # 4. Extrair origem e destino (Lat, Lng, Address)
28 dfRideAdd = dfRideAdd.rename(columns={"RideAddressTypeID": "OrigDest"})
29 dfOrigem = dfRideAdd[dfRideAdd["OrigDest"] == "1"][["RideID", "Lat", "Lng", "Address"]].rename(
```

```
30     columns={"Lat": "Lat1", "Lng": "Lng1", "Address": "AddressOrig"}
31 )
32 dfDestino = dfRideAdd[dfRideAdd["OrigDest"] == "2"][["RideID", "Lat", "Lng", "Address"]].rename(
33     columns={"Lat": "Lat2", "Lng": "Lng2", "Address": "AddressDest"}
34 )
35 dfCoords = pd.merge(dfOrigem, dfDestino, on="RideID", how="inner")
36
37 # Corrige vírgulas e converte coordenadas
38 for col in ["Lat1", "Lng1", "Lat2", "Lng2"]:
39     dfCoords[col] = dfCoords[col].str.replace(",", ".").astype(float).round(6)
40
41 # 5. Integrar todas as estimativas com produtos em UMA COLUNA tipo dicionário
42 dfRideEst["ProductID"] = dfRideEst["ProductID"].astype(str)
43 dfProduct["ProductID"] = dfProduct["ProductID"].astype(str)
44
45 dfEstimadaComProduto = pd.merge(dfRideEst, dfProduct, on="ProductID", how="left")
46 dfEstimadaComProduto["Price"] = dfEstimadaComProduto["Price"].str.replace(",", ".").astype(float)
47
48 dfEstimadaSelecionada = dfEstimadaComProduto.groupby("RideID").apply(
49     lambda x: dict(zip(x["Description"], x["Price"])))
50 ).reset_index().rename(columns={0: "Estimativas"})
51
52 # 6. Refiltra pelos RideID em comum
53 dfCoords["RideID"] = dfCoords["RideID"].astype(str)
54 dfEstimadaSelecionada["RideID"] = dfEstimadaSelecionada["RideID"].astype(str)
55 ids_comuns = set(dfTempo["RideID"]) & set(dfCoords["RideID"]) & set(dfEstimadaSelecionada["RideID"])
56
57 dfTempo = dfTempo[dfTempo["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)
58 dfCoords = dfCoords[dfCoords["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)
59 dfEstimadaSelecionada = dfEstimadaSelecionada[dfEstimadaSelecionada["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)
60
61 # 7. Junta tudo sem merge (concatenando os DataFrames horizontalmente)
62 dfDerivado = pd.concat([
63     dfTempo,
64     dfCoords.drop(columns=["RideID"]),
65     dfEstimadaSelecionada.drop(columns=["RideID"])
66 ], axis=1)
67
68 # 8. Remove NaNs nas coordenadas
69 dfDerivado = dfDerivado.dropna(subset=["Lat1", "Lng1", "Lat2", "Lng2"]).reset_index(drop=True)
```

```

/0
71 # 9. Cálculo da distância
72 dfDerivado["Distancia_km"] = dfDerivado.apply(
73     lambda row: geodesic((row["Lat1"], row["Lng1"]), (row["Lat2"], row["Lng2"])).kilometers,
74     axis=1
75 )
76
77 # Mostrar amostra do DataFrame final com estimativas e valor_uber
78 print("\n🇧🇷 Exemplo de dados do dfDerivado com estimativas:")
79 print(dfDerivado.head(10))

```

↳ <ipython-input-3-3a1a1636f833>:48: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated
dfEstimadaSelecionada = dfEstimadaComProduto.groupby("RideID").apply(

🇧🇷 Exemplo de dados do dfDerivado com estimativas:

	RideID	Dia	Hora	Minuto	HoraDecimal	Faixa15min	Lat1	\
0	1183200	1	10	9	10.150000	2021-08-17 10:00:00	-26.329754	
1	1183201	1	10	9	10.150000	2021-08-17 10:00:00	-27.491979	
2	1183202	1	10	10	10.166667	2021-08-17 10:00:00	-19.849580	
3	1183203	1	10	10	10.166667	2021-08-17 10:00:00	-23.962423	
4	1183204	1	10	10	10.166667	2021-08-17 10:00:00	-10.919802	
5	1183205	1	10	10	10.166667	2021-08-17 10:00:00	-22.873502	
6	1183206	1	10	10	10.166667	2021-08-17 10:00:00	-23.554281	
7	1183207	1	10	10	10.166667	2021-08-17 10:00:00	-23.962423	
8	1183208	1	10	10	10.166667	2021-08-17 10:00:00	-19.849539	
9	1183209	1	10	10	10.166667	2021-08-17 10:00:00	-8.025771	

	Lng1	AddressOrig	Lat2	\
0	-48.840428	Rua João Pinheiro, 585 - Rua João Pinheiro - B...	-26.255466	
1	-48.528288	Rodovia Rafael da Rocha Pires, 1883 - Rodovia ...	-27.437149	
2	-44.019916	Rua Barão do Rio Branco, 12 - Rua Barão do Rio...	-19.936899	
3	-46.254658	Tv. Duzentos e Sessenta e Um, 72, 72	-23.837307	
4	-37.077442	Rua Argentina, 160 - Rua Argentina - Brasil	-10.907129	
5	-43.571402	Rua João Cirílo de Oliveira, 5 - Rua João Cirí...	-22.917373	
6	-46.662732	Avenida Angélica, 2573 - Avenida Angélica - Br...	-23.734290	
7	-46.254658	Tv. Duzentos e Sessenta e Um, 72, 72	-23.837307	
8	-44.019929	R. Barão do Rio Branco, 12 - Nacional, Contage...	-19.936899	
9	-34.874475	Rua João Alfredo, 83 - Rua João Alfredo - Brasil	-8.029684	

	Lng2	AddressDest	\
0	-48.643420	Av. Dr. Nereu Ramos, 450 - Rocio Grande, São F...	
1	-48.398243	Angeloni Ingleses (Florianópolis) - Supermerca...	
2	-43.940160	R. Antônio de Albuquerque, 1080 - Funcionários...	

```

3 -46.132172          Semar Supermercados Bertioga, 2141
4 -37.087719  R. Simeão Aguiar, 430 - Novo Paraíso, Aracaju ...
5 -43.633044  Av. Cesário de Melo, 10809 - Paciência, Rio de...
6 -46.698628  Av. Sen. Teotônio Vilela, 4029 - Parque Cocaia...
7 -46.132172          Semar Supermercados Bertioga, 2141
8 -43.940160  R. Antônio de Albuquerque, 1080 - Funcionários...
9 -34.944037  Av. Professor Joaquim Cavalcanti, 721 - Iputin...

```

	Estimativas	Distancia_km
0	{'Flash': 89.0, 'UberX': 89.0, 'Comfort': 116....	21.327034
1	{'Flash': 31.5, 'Comfort': 33.5, '99POUPA': 26...	14.217724
2	{'Moto': 25.5, 'Comfort': 44.0, 'UberFlash': 4...	12.774740
3	{'Flash': 47.5, '99POP': 63.69, 'Comfort': 68....	18.644013
4	{'99POUPA': 7.88, '99POP': 7.88, '99TAXI': 17....	1.796461
5	{'99POUPA': 14.69, '99POP': 19.51, '99TAXI': 5...	7.975203
6	{'UberX': 46.5, 'Comfort': 71.5, 'Flash': 43.0...	20.270171
7	{'99TAXI': 118.16, 'Táxi Comum': 138.95, 'Flas...	18.644013
8	{'99POUPA': 71.3, '99POP': 71.3, '99TAXI': 62....	12.779065
9	{'Flash': 18.0, 'Flash Moto': 12.0, 'Comfort':...	7.680426

✓ **TREINANDO O MODELO**

```

1 from sklearn.ensemble import RandomForestRegressor
2 from sklearn.model_selection import train_test_split
3 from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
4 import numpy as np
5 import pandas as pd
6
7 # 🚗 Lista de serviços da Uber a prever
8 servicos_alvo = ["UberX", "Comfort", "Black"]
9
10 print("📁 Treinando modelos para serviços da Uber (com features auxiliares):\n")
11
12 # Listas para armazenar os resultados e os modelos treinados
13 resultados = []
14 modelos_treinados = {} # <= AQUI salvaremos os modelos
15
16 for servico in servicos_alvo:


```

```
17 print(f"🚀 Iniciando modelo para: {servico}")
18
19 # Filtra somente onde o serviço alvo tem valor
20 df_modelo = dfDerivado[dfDerivado["Estimativas"].apply(
21     lambda d: isinstance(d, dict) and servico in d and isinstance(d[servico], (int, float))
22 )].copy()
23
24 if df_modelo.empty:
25     print(f"⚠️ Nenhum dado encontrado para {servico}. Ignorado.\n")
26     continue
27
28 # Define a variável alvo
29 df_modelo["y"] = df_modelo["Estimativas"].apply(lambda d: d[servico])
30
31 # Features base
32 features_base = ["Distancia_km", "Dia", "Hora", "HoraDecimal", "Lat1", "Lng1", "Lat2", "Lng2"]
33
34 # Adiciona colunas auxiliares com os demais serviços (exceto o alvo)
35 servicos_aux = set()
36 df_modelo["Estimativas"].apply(lambda d: servicos_aux.update(d.keys()) if isinstance(d, dict) else None)
37 servicos_aux.discard(servico)
38
39 for s in servicos_aux:
40     nome_coluna = f"aux_{s.lower().replace(' ', '_')}"
41     df_modelo[nome_coluna] = df_modelo["Estimativas"].apply(lambda d: d.get(s) if isinstance(d, dict) else np.nan)
42
43 # Prepara X e y
44 features_auxiliares = [col for col in df_modelo.columns if col.startswith("aux_")]
45 X = df_modelo[features_base + features_auxiliares].fillna(-1)
46 y = df_modelo["y"]
47
48 # Verificação mínima de dados
49 if len(X) < 100:
50     print(f"⚠️ Serviço '{servico}' com poucos dados após preparação ({len(X)} linhas) – ignorado.\n")
51     continue
52
53 # Treinamento
54 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
55 modelo = RandomForestRegressor(random_state=42)
56 modelo.fit(X_train, y_train)
```

```

57     y_pred = modelo.predict(X_test)
58
59     # ✅ Salvando o modelo treinado
60     modelos_treinados[servico] = modelo
61
62     # Métricas
63     mae = mean_absolute_error(y_test, y_pred)
64     rmse = np.sqrt(mean_squared_error(y_test, y_pred))
65     r2 = r2_score(y_test, y_pred)
66
67     resultados.append({
68         "Serviço": servico,
69         "Registros": len(X),
70         "MAE": round(mae, 2),
71         "RMSE": round(rmse, 2),
72         "R²": round(r2, 4)
73     })
74
75     print(f"✅ Modelo treinado para: {servico}")
76     print(f"→ MAE : R${mae:.2f}")
77     print(f"→ RMSE : R${rmse:.2f}")
78     print(f"→ R² : {r2:.4f}\n")
79
80 # 🏁 Resumo final
81 if resultados:
82     df_resultados = pd.DataFrame(resultados).sort_values(by="R²", ascending=False)
83     print("📊 Resumo Final dos Modelos:")
84     print(df_resultados.to_string(index=False))
85 else:
86     print("❌ Nenhum modelo foi treinado.")

```

🔗  Treinando modelos para serviços da Uber (com features auxiliares):

🤖 Iniciando modelo para: UberX
 ✅ Modelo treinado para: UberX
 → MAE : R\$0.48
 → RMSE : R\$2.27
 → R² : 0.9904

🤖 Iniciando modelo para: Comfort
 ✅ Modelo treinado para: Comfort

→ MAE : R\$2.20
 → RMSE : R\$4.88
 → R² : 0.9807

🤖 Iniciando modelo para: Black

✅ Modelo treinado para: Black

→ MAE : R\$1.82
 → RMSE : R\$4.86
 → R² : 0.9852

📊 Resumo Final dos Modelos:

Serviço	Registros	MAE	RMSE	R ²
UberX	235601	0.48	2.27	0.9904
Black	123666	1.82	4.86	0.9852
Comfort	192876	2.20	4.88	0.9807

✓ **INPUT CLIENTE**

```

1 from geopy.geocoders import Nominatim
2 from geopy.distance import geodesic
3 import pandas as pd
4 from datetime import datetime
5
6 # Inicializa o geocodificador
7 geolocator = Nominatim(user_agent="uber-predict")
8
9 def endereco_para_coordnadas(endereco):
10     try:
11         local = geolocator.geocode(endereco)
12         if local:
13             return (local.latitude, local.longitude)
14         else:
15             return None
16     except:
17         return None
18
19 def calcular_distancia_km(coord1, coord2):
20     return geodesic(coord1, coord2).km

```

```
21
22 def prever_por_endereco():
23     print("📍 Previsão de Preço com Endereços")
24
25     servico = input("👉 Escolha o serviço (UberX, Comfort, Black): ").strip()
26     if servico not in modelos_treinados:
27         print("❌ Modelo para esse serviço não foi treinado.")
28         return
29
30     origem = input("📍 Endereço de origem: ")
31     destino = input("📍 Endereço de destino: ")
32
33     coord_origem = endereco_para_coordenadas(origem)
34     coord_destino = endereco_para_coordenadas(destino)
35
36     if not coord_origem or not coord_destino:
37         print("❌ Não foi possível localizar um ou ambos os endereços.")
38         return
39
40     distancia_km = calcular_distancia_km(coord_origem, coord_destino)
41     print(f"📏 Distância estimada: {distancia_km:.2f} km")
42
43     # Hora e dia atuais
44     agora = datetime.now()
45     hora = agora.hour
46     hora_decimal = agora.hour + agora.minute / 60
47     dia_semana = agora.weekday() # segunda = 0
48
49     # Criar dados de entrada
50     dados = {
51         'Distancia_km': distancia_km,
52         'Dia': dia_semana,
53         'Hora': hora,
54         'HoraDecimal': hora_decimal,
55         'Lat1': coord_origem[0],
56         'Lng1': coord_origem[1],
57         'Lat2': coord_destino[0],
58         'Lng2': coord_destino[1],
59     }
60
```



```
61     modelo = modelos_treinados[servico]
62     for col in modelo.feature_names_in_:
63         if col not in dados:
64             dados[col] = -1 # preencher colunas auxiliares com -1
65
66     df_input = pd.DataFrame([dados])
67     preco = modelo.predict(df_input)[0]
68
69     print(f"\n💰 Preço estimado para o serviço {servico}: R$ {preco:.2f}")
70
71 # ✅ Execute a função para testar
72 prever_por_endereco()
```



📍 Previsão de Preço com Endereços
👉 Escolha o serviço (UberX, Comfort, Black): Black
📌 Endereço de origem: Avenida Paulista, São Paulo
🎯 Endereço de destino: Parque Ibirapuera, São Paulo
🗺️ Distância estimada: 3.52 km

💰 Preço estimado para o serviço Black: R\$ 22.06

