DERIVANDO DADOS NO PROJETO KHIPO

Importando tabelas

```
import pandas as pd
```

```
dfProd=pd.read_csv("product.csv",sep=";")
dfProd.head(5)
```

	ProductID	ProviderID	CategoryID	Description
0	99POP	3	1	99POP
1	1	1	5	Taxi Comum
2	2	1	6	Executivo
3	46cec5c4d23e57bcba2122677eb8c759	4	5	Easy Taxi Corp (-15%)
4	5fc141256dc70a394d0ce4c5c1444dfc	4	5	Taxi

```
dfRide=pd.read_csv("ride_v2.csv",sep=";")
dfRide.head(1)
```

	RideID	UserID	Schedule	Create	RideStatusID	CompanyID	ProviderID	RideProviderID	price	Updated	C
0	1685755	e15b8cc3-5a67-4630-b89f-ee69f302b582	2025-02-10 14:31:10.8858446	2025-02-10 14:31:10.9084221	1	2	NaN	NaN	0.0	2025-02-10 14:31:10.9084233	

```
dfRideAdd=pd.read_csv("rideaddress_v1.csv",sep=";",low_memory=False)
dfRideAdd.head(1)
```

	RideAddressID	Address	Street	Number	Neighborhood	City	State	Lat	Lng	RideAddressTypeID	RideI
		Rua									

```
dfRideEst=pd.read_csv("rideestimative_v3.csv",sep=";",low_memory=False)
dfRideEst.head(1)
```

	RideEstimativeID	RideID	ProductID	WaitingTime	Price	FareID	Selected	RideReasonSelectedEstimativeID	Fee
0	8619946	1183200	Flash	8	80.0	c6aaac64-5f89-4fc4-8b66-	0	NaN	0.0

Tratando dados nulos e visualizando tamanho das bases

```
print(len(dfProd))
print(len(dfRide))
print(len(dfRideAdd))
print(len(dfRideEst))
print(len(dfProd.dropna()))
print(len(dfRide.dropna()))
print(len(dfRideAdd.dropna()))
print(len(dfRideEst.dropna()))
```

```
237
500000
1000000
2000000
237
14781
121971
132236
```

Criando coluna com dados da semana

```
import datetime
data=pd.to_datetime(dfRide.iloc[0]["Schedule"])
print(data)
print(data.day_name())
print(data.weekday())
```

```
2025-02-10 14:31:10.885844600
Monday
0
```

Criando a coluna dia para receber os dados (dia da semana{segunda= 0})

```
dia=[]
for i in range(len(dfRide)):
    data=pd.to_datetime(dfRide.iloc[i]["Schedule"])
    dia.append(data.weekday())

dfRide["Dia"]=dia
dfRide.head(5)
```

	RideID	UserID	Schedule	Create	RideStatusID	CompanyID	ProviderID	RideProviderID	price	Updated
0	1685755	e15b8cc3-5a67-4630-b89f-ee69f302b582	2025-02-10 14:31:10.8858446	2025-02-10 14:31:10.9084221	1	2	NaN	NaN	0.00	2025-02-10 14:31:10.9084233
1	1685754	5c3fb011-0aea-429a-8305-b88953b77df1	2025-02-10 14:26:35.3411403	2025-02-10 14:26:35.4169873	2	230	5.0	NaN	30.45	2025-02-10 14:28:02.4656963
2	1685753	d7e2f4dc-337f-45f5-b762-67b72b077abc	2025-02-10 14:23:45.2540905	2025-02-10 14:24:32.7058722	2	52	3.0	NaN	11.40	2025-02-10 14:24:46.5037165
3	1685752	2125ed9c-89b8-4df6-9be6-53195397a269	2025-02-10 14:23:12.9838635	2025-02-10 14:23:12.9975475	8	230	36.0	1589157	45.79	2025-02-10 14:30:30.6031123
4	1685751	72cbebfb-5d70-49ab-ab23-8e3b57c7e399	2025-02-10 14:19:30.5937678	2025-02-10 14:19:30.6117184	2	2	3.0	NaN	17.28	2025-02-10 14:24:45.9711764

DIMINUINDO O TAMANHO DA BASE PARA COMEÇAR A EXPLORAR E DERIVAR

```
dfRide["RideID"].median()
```

```
1434271.5
```

```
len(dfRide.query("RideID<=1300000"))
```

```
116782
```

```
dfRide=dfRide.query("RideID<=1300000")
dfRideAdd=dfRideAdd.query("RideID<=1300000")
dfRideEst=dfRideEst.query("RideID<=1300000")
print(len(dfRide))
print(len(dfRideAdd))
print(len(dfRideEst))
```

```
116782
233564
905383
```

RENOMIANDO COLUNAS DATAFRAME dfRideAdd TEM A COLUNA RideAddressTypeID QUE INDICA SE O ENDEREÇO É ORIGEM(1) OU DESTINO(2) PODEMOS RENOMEAR PARA OrigDest

```
dfRideAdd = dfRideAdd.rename(columns={'RideAddressTypeID': 'OrigDest'})
dfRideAdd.head(1)
```

RideAddressID	Address	Street	Number	Neighborhood	City	State	Lat	Lng	OrigDest	RideID
	Rua João	Rua								

SELECIONANADO AS COLUNAS QUE QUEREMOS TRABALHAR (dfRideAdd podemos querer trabalhar com RideID,OrigDest,Lat e Lng)

```
dfRideAdd=dfRideAdd[["RideID","OrigDest","Lat","Lng"]]
dfRideAdd.head(1)
```

	RideID	OrigDest	Lat	Lng
0	1183200	1	-26.329754299999998	-48.840427999999996

Colocando tudo em uma linha Latitude e Longitude Origem e Destino e a distancia Para isso vamos CRIAR dfRideAdd2 (Destino) e tirar o destino de dfRideAdd

```
dfRideAdd2=dfRideAdd.query("OrigDest==2")
dfRideAdd2.head(3)
```

	RideID	OrigDest	Lat	Lng
1	1183200	2	-26.2554657	-48.6434197
3	1183201	2	-27.4371486	-48.39824309999999
5	1183202	2	-19.936899	-43.9401603

```
dfRideAdd=dfRideAdd.query("OrigDest!=2")
dfRideAdd.head(3)
```

	RideID	OrigDest	Lat	Lng
0	1183200	1	-26.329754299999998	-48.840427999999996
2	1183201	1	-27.4919788	-48.528287999999996
4	1183202	1	-19.8495799	-44.019915999999995

Renomeando nome novamente as colunas para fazer o merge

```
dfRideAdd=dfRideAdd.rename(columns={"Lat":"Lat1","Lng":"Lng1"})
dfRideAdd2=dfRideAdd2.rename(columns={"Lat":"Lat2","Lng":"Lng2"})
```

Fazendo o join

```
dfJoinAdd = pd.merge(dfRideAdd,dfRideAdd2, left_on='RideID', right_on='RideID', how='inner')
dfJoinAdd.head(5)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197
1	1183201	1	-27.4919788	-48.528287999999996	2	-27.4371486	-48.39824309999999
2	1183202	1	-19.8495799	-44.019915999999995	2	-19.936899	-43.9401603
3	1183203	1	-23.9624233	-46.25465759999999	2	-23.8373074	-46.1321725
4	1183204	1	-10.9198019	-37.077441799999995	2	-10.9071288	-37.0877194

Lógica de distância

```
!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)
```

```
from geopy.distance import geodesic

# Pontos: (latitude, longitude)
ponto1 = (-27.4919788, -48.528287999999996)
ponto2 = (-27.4371486, -48.39824309999999)

# Distância em km
distancia_km = geodesic(ponto1, ponto2).kilometers
print(f"Distância: {distancia_km:.2f} km")
```

```
Distância: 14.22 km
```

Adicionando a distância em toda a tabela

```
Dist=[]

for i in range(len(dfJoinAdd)):
    lat1=float(dfJoinAdd.iloc[i]["Lat1"].replace(",","."))
    lng1=float(dfJoinAdd.iloc[i]["Lng1"].replace(",","."))
    lat2=float(dfJoinAdd.iloc[i]["Lat2"].replace(",","."))
    lng2=float(dfJoinAdd.iloc[i]["Lng2"].replace(",","."))
    ponto1 = (lat1, lng1)
    ponto2 = (lat2, lng2)
    distkm = geodesic(ponto1, ponto2).kilometers
    Dist.append(distkm)

dfJoinAdd["Distancia"]=Dist

dfJoinAdd.head(5)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2	Distancia
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087
1	1183201	1	-27.4919788	-48.528287999999996	2	-27.4371486	-48.39824309999999	14.217724
2	1183202	1	-19.8495799	-44.019915999999995	2	-19.936899	-43.9401603	12.774728
3	1183203	1	-23.9624233	-46.25465759999999	2	-23.8373074	-46.1321725	18.643943
4	1183204	1	-10.9198019	-37.077441799999995	2	-10.9071288	-37.0877194	1.796511

Separando o dia da semana vinculando pelo RideID Já usando o melhor modo de treinar a ML.

```
dfRide3 = dfRide[["RideID","Dia"]]
print(dfRide3)
```

```
         RideID  Dia
383218  1300000    3
383219  1299999    3
383220  1299998    3
383221  1299997    3
383222  1299996    3
...         ...  ...
499995  1183204    1
```

```
499996  1183203    1
499997  1183202    1
499998  1183201    1
499999  1183200    1

[116782 rows x 2 columns]
```

```
dfJoinAdd2 = pd.merge(dfJoinAdd, dfRide3, left_on = 'RideID', right_on ='RideID', how='inner')
dfJoinAdd2.head(5)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2	Distancia	Dia
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1
1	1183201	1	-27.4919788	-48.528287999999996	2	-27.4371486	-48.39824309999999	14.217724	1
2	1183202	1	-19.8495799	-44.019915999999995	2	-19.936899	-43.9401603	12.774728	1
3	1183203	1	-23.9624233	-46.25465759999999	2	-23.8373074	-46.1321725	18.643943	1
4	1183204	1	-10.9198019	-37.077441799999995	2	-10.9071288	-37.0877194	1.796511	1

Selecionando os produtos para treinar a ML e imprimindo a quantidade de corridas para treinamento.

```
dfprodutos = dfRideEst.query("ProductID in ['UberX', '99POP', 'Flash', 'Confort', 'pop99','poupa99', 'Bag', 'UberFlash', '', '', '7da40a
print(len(dfprodutos))
```

```
421534
```

Separando o preço por RideID

```
dftprodutos = dfprodutos[["RideID","Price"]]
print(dftprodutos)
```

```
         RideID   Price
0       1183200   89.00
1       1183200   89.00
3       1183200  170.21
4       1183200  170.21
7       1183201   31.50
...         ...     ...
905373  1299999   51.50
905376  1300000   20.33
905377  1300000   23.62
905380  1300000   20.00
905382  1300000   20.00

[421534 rows x 2 columns]
```

```
dftprodutos1 = dfRideEst[["RideID","ProductID"]]
print(dftprodutos1)
```

```
         RideID     ProductID
0       1183200         Flash
1       1183200         UberX
2       1183200       Comfort
3       1183200       poupa99
4       1183200         pop99
...         ...           ...
905378  1300000    turbo-taxi
905379  1300000  regular-taxi
905380  1300000         Flash
905381  1300000       Comfort
905382  1300000         UberX

[905383 rows x 2 columns]
```

```
dfJoinAdd3 = pd.merge(dfJoinAdd2, dftprodutos, left_on = 'RideID', right_on ='RideID', how='inner')
dfJoinAdd3.head(5)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2	Distancia	Dia	Price
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	89.00
1	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	89.00
2	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	170.21
3	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	170.21
4	1183201	1	-27.4919788	-48.528287999999996	2	-27.4371486	-48.39824309999999	14.217724	1	31.50

```
dfRide['schedule']= pd.to_datetime(dfRide['Schedule'])
dfRide['schedule_time'] = dfRide['schedule'].dt.time
dfRide.head(1)
```

	RideID	UserID	Schedule	Create	RideStatusID	CompanyID	ProviderID	RideProviderID	price	Update
383218	1300000	acbb3e9f-62a8-4f01-b04f-f58058728fc4	2021-09-30 16:01:11.5153413	2021-09-30 16:01:11.5488417	2	40	NaN	NaN	20.0	2021-09-3 16:01:23.458564

Separando a hora da coluna schedule da dfRideId

```
dfRideT=dfRide[["RideID","schedule_time"]]
dfRideT.head(1)
```

	RideID	schedule_time
383218	1300000	16:01:11.515341

Fazendo merge da schedulo(hora de inicio)

```
dfJoinAdd3 = pd.merge(dfJoinAdd3, dfRideT, left_on = 'RideID', right_on ='RideID', how='inner')
dfJoinAdd3.head(1)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2	Distancia	Dia	Price	schedule_tim
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	89.0	10:09:45.62817

```
dfJoinAdd3 = pd.merge(dfJoinAdd3, dftprodutos1, left_on = 'RideID', right_on ='RideID', how='inner')
dfJoinAdd3.head(1)
```

	RideID	OrigDest_x	Lat1	Lng1	OrigDest_y	Lat2	Lng2	Distancia	Dia	Price	schedule_tim
0	1183200	1	-26.329754299999998	-48.840427999999996	2	-26.2554657	-48.6434197	21.327087	1	89.0	10:09:45.62817

Criação do banco de dados

Extrair a base pronta para ML

```
dfJoinAdd3.to_csv('dfJoinAdd3.csv', index=False)
```

```
dfJoinAdd3=pd.read_csv("dfJoinAdd3.csv",sep=",")
dfJoinAdd3.head(2)
```

```
import sqlite3
```

```
conn = sqlite3.connect("base_corridas")
cursor = conn.cursor()
```

Criação da tabela no banco de Dados (não executar)

```
cursor.execute('''
DROP TABLE Corridas
''')

conn.commit()
```

```
cursor.execute('''
CREATE TABLE Corridas (
    RideID INT,
    OrigDest_x INT,
    Lat1 REAL,
    Lng1 REAL,
    OrigDest_y INT,
    Lat2 REAL,
    Lng2 REAL,
    Distancia REAL,
    Dia INT,
    Price DECIMAL(10, 2),
    schedule_time TIME
)
''')

conn.commit()
```

Populando a tabela com os dados

```
strSQL = '''
INSERT INTO Corridas (RideID, OrigDest_x, Lat1, Lng1, OrigDest_y, Lat2, Lng2, Distancia, Dia, Price, schedule_time,ProductID)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,?)'''


dfJoinAdd3.to_sql('Corridas', conn, if_exists='append', index=False)
```

```
dfJoinAdd3 = pd.read_sql_query("SELECT * FROM Corridas", conn)
dfJoinAdd3.head(2)
```

```
    RideID  OrigDest_x                Lat1                Lng1  OrigDest_y         Lat2         Lng2  Distancia  Dia  Price   schedule_tim
0  1183200           1  -26.329754299999998  -48.840427999999996           2  -26.2554657  -48.6434197  21.327087    1   89.0  10:09:45.62817
1  1183200           1  -26.329754299999998  -48.840427999999996           2  -26.2554657  -48.6434197  21.327087    1   89.0  10:09:45.62817
```

Transformando schedule em horas

```
from datetime import datetime
dfJoinAdd3 = pd.read_sql_query("SELECT * FROM Corridas", conn)
X = dfJoinAdd3[["OrigDest_x","Lat1","Lng1","OrigDest_y","Lat2","Lng2","Distancia","Dia","schedule_time","ProductID"]]
y = dfJoinAdd3["Price"]
time_str = X.iloc[0]['schedule_time']
# Converter para objeto time
time_obj = datetime.strptime(time_str, "%H:%M:%S.%f").time()
```

Para testar inclua o BD(base corridas) e Data frame (dfJoinAdd3.cv)

Treinamento da Machine Learning

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from datetime import datetime
import pandas as pd


def strtotime(time_str):
    time_obj = datetime.strptime(time_str, "%H:%M:%S.%f").time()
    return time_obj
```

```python

dfJoinAdd3 = pd.read_sql_query("SELECT * FROM Corridas", conn)

dfJoinAdd3 = dfJoinAdd3.sample(frac=0.5) #reduz em 50% da base para teste

X = dfJoinAdd3[["OrigDest_x","Lat1","Lng1","OrigDest_y","Lat2","Lng2","Distancia","Dia","schedule_time"]]
y = dfJoinAdd3["Price"]

SEED = 20

treino_x, teste_x, treino_y, teste_y = train_test_split(X, y, test_size=0.3, random_state=SEED) # Linha adicionada
#coluna limpa:

# Convertendo colunas de latitude e longitude para o formato correto
for col in ["Lat1", "Lng1", "Lat2", "Lng2"]:
    treino_x[col] = treino_x[col].astype(str).str.replace(',', '.').astype(float)
    teste_x[col] = teste_x[col].astype(str).str.replace(',', '.').astype(float)

# Convertendo 'schedule_time' para segundos desde a meia-noite
treino_x['schedule_time'] = treino_x['schedule_time'].apply(lambda x: strtotime(x).hour * 3600 + strtotime(x).minute * 60 + strtotime(x)
teste_x['schedule_time'] = teste_x['schedule_time'].apply(lambda x: strtotime(x).hour * 3600 + strtotime(x).minute * 60 + strtotime(x).s

#treino_x = pd.get_dummies(treino_x, columns=['ProductID'], prefix=['ProductID'])
#teste_x = pd.get_dummies(teste_x, columns=['ProductID'], prefix=['ProductID'])

# Alinhar as colunas após o get_dummies
treino_x, teste_x = treino_x.align(teste_x, join='left', axis=1, fill_value=0)

# Agora você pode treinar o modelo:
model = RandomForestRegressor(n_estimators=100, random_state=SEED)
model.fit(treino_x , treino_y)

predicao_y = model.predict(teste_x)
r2 = r2_score(teste_y, predicao_y)
mse = mean_squared_error(teste_y, predicao_y)
mae = mean_absolute_error(teste_y, predicao_y)

print(f"Erro médio absoluto: {mae}")
print(f"Erro quadrático médio: {mse}")
```

```
Erro médio absoluto: 4.989349073690697
Erro quadrático médio: 107.83222191258031
```

```python
from sklearn.ensemble import GradientBoostingRegressor

# 2 - Definindo o modelo:
modelo = GradientBoostingRegressor()
modelo.fit(treino_x, treino_y)
```

```
▾ GradientBoostingRegressor  ⓘ ?
GradientBoostingRegressor()
```

```python
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# 2 - Aplicação do Teste:
#predicao_y = modelo.predict(teste_x)
predicao_y = modelo.predict(teste_x)

# 3 - Avaliação do Teste (com todos os modelos de Metrics):
r2 = r2_score(teste_y, predicao_y)

mse = mean_squared_error(teste_y, predicao_y)
mae = mean_absolute_error(teste_y, predicao_y)

# 4 - Mostrar os resultados:

print(f"R2 (Acuracia): {r2*100:.2f}%")
print(f"MSE : {mse:.2f}")
print(f"MAE : {mae:.2f}")
```

```
R2 (Acuracia): 89.73%
MSE : 192.00
MAE : 7.44
```

Logica Final pedindo endereço

```
!pip install --upgrade WazeRouteCalculator
```

```
Collecting WazeRouteCalculator
  Downloading WazeRouteCalculator-0.15-py3-none-any.whl.metadata (448 bytes)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from WazeRouteCalculator) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->WazeRouteCalculator)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->WazeRouteCalculator) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->WazeRouteCalculator) (2.4.0
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->WazeRouteCalculator) (2025.
Downloading WazeRouteCalculator-0.15-py3-none-any.whl (16 kB)
Installing collected packages: WazeRouteCalculator
Successfully installed WazeRouteCalculator-0.15
```

```python
from WazeRouteCalculator import WazeRouteCalculator, WRCError
from geopy.geocoders import Nominatim
from datetime import datetime
import pandas as pd
import warnings
import unicodedata

warnings.filterwarnings("ignore")

# Inicializa geolocalizador
geolocator = Nominatim(user_agent="predictor-app")

def limpar_endereco(endereco):
    nfkd = unicodedata.normalize('NFKD', endereco)
    return u"".join([c for c in nfkd if not unicodedata.combining(c)])

def calcular_distancia_waze(origem, destino, region='EU'):
    try:
        lat1, lng1 = endereco_para_coordenadas(origem)
        lat2, lng2 = endereco_para_coordenadas(destino)

        origem_coord = f"{lat1},{lng1}"
        destino_coord = f"{lat2},{lng2}"

        rota = WazeRouteCalculator(origem_coord, destino_coord, region)
        tempo_min, distancia_km = rota.calc_route_info(real_time=False)  # ou real_time=True, se quiser tráfego atual

        return distancia_km, tempo_min
    except WRCError as e:
        raise ValueError(f"Erro ao calcular rota Waze: {e}")


def endereco_para_coordenadas(endereco):
    endereco_limpo = limpar_endereco(endereco)
    location = geolocator.geocode(endereco_limpo, timeout=15)
    if location:
        return location.latitude, location.longitude
    else:
        raise ValueError(f"Não foi possível geocodificar o endereço: {endereco}")

def gerar_features(end_origem, end_destino, horario=None):
    lat1, lng1 = endereco_para_coordenadas(end_origem)
    lat2, lng2 = endereco_para_coordenadas(end_destino)
    distancia, tempo = calcular_distancia_waze(end_origem, end_destino)

    now = horario or datetime.now()
    schedule_time = now.hour * 3600 + now.minute * 60 + now.second
    dia = now.weekday()

    orig_dest_x = abs(hash(end_origem)) % (10 ** 8)
    orig_dest_y = abs(hash(end_destino)) % (10 ** 8)

    df = pd.DataFrame([{
        "OrigDest_x": orig_dest_x,
        "Lat1": lat1,
        "Lng1": lng1,
        "OrigDest_y": orig_dest_y,
        "Lat2": lat2,
        "Lng2": lng2,
        "Distancia": distancia,
        "Tempo": tempo,
        "Dia": dia,
        "schedule_time": schedule_time
    }])

    return df

def prever_preco(end_origem, end_destino):
    dados = gerar_features(end_origem, end_destino)
    dados_para_prever = dados.drop(columns=["Tempo"])
    preco_estimado = model.predict(dados_para_prever)[0]

    dados["Preco_Previsto"] = preco_estimado
    dados["Endereco_Origem"] = end_origem
    dados["Endereco_Destino"] = end_destino
    return dados

# ================================

if __name__ == "__main__":
    print("=== Previsão de Preço Uber ===")
    origem = input("Digite o endereço de ORIGEM: ").strip()
    destino = input("Digite o endereço de DESTINO: ").strip()

    try:
        resultado = prever_preco(origem, destino)
        preco = resultado["Preco_Previsto"].values[0]
        distancia = resultado["Distancia"].values[0]
        tempo = resultado["Tempo"].values[0]

        preco_confort = preco * 1.25
        preco_black = preco_confort * 1.24

        print(f"\n🚗 Preço estimado da corrida (Ubex): R$ {preco:.2f}")
        print(f"\n🚗 Preço estimado (Uber Confort): R$ {preco_confort:.2f}")
        print(f"\n🚘 Preço estimado (Uber Black): R$ {preco_black:.2f}")
        print(f"\n📍 Distância estimada: {distancia:.2f} km")
        print(f"⏱️ Tempo estimado: {tempo:.0f} minutos")

    except Exception as e:
        print(f"\n❌ Erro ao calcular o preço: {e}")

```

```
=== Previsão de Preço Uber ===
Digite o endereço de ORIGEM: Rua Itajai 125 - Sao Paulo
Digite o endereço de DESTINO: Av Paulista 2000 - Sao Paulo

🚗 Preço estimado da corrida (Ubex): R$ 36.29

🚗 Preço estimado (Uber Confort): R$ 45.37

🚘 Preço estimado (Uber Black): R$ 56.25

📍 Distância estimada: 7.99 km
⏱️ Tempo estimado: 17 minutos
```