```
import pandas as pd
df = pd.read_csv("Resumo_v2.csv",sep=";")
df.head(5)
```

	RideID	Create	Updated	ProductID	Car	price	Tempo_Viagem_Min	Dia	Distância_km	Coordenadas_1	Coordenadas_2
0	1425243	14/06/2022	14/06/2022	pop99	NaN	11.05	0.20	Terça-feira	2.03	-22.9505577, -43.1826401	-22.9687019, -43.1852154
1	1425241	14/06/2022	14/06/2022	UberX	NaN	28.50	0.46	Terça-feira	5.13	-23.5996561, -46.6655571	-23.5605827, -46.6925092
2	1425240	14/06/2022	14/06/2022	Comfort	NaN	22.00	0.14	Terça-feira	4.76	-22.847022, -47.053341499999995	-22.8851178, -47.0319086
3	1425239	14/06/2022	14/06/2022	pop99	NaN	20.69	0.27	Terça-feira	4.64	-25.4472447, -49.245239399999996	-25.4073641, -49.2593377
4	1425238	14/06/2022	14/06/2022	pop99	NaN	35.87	0.07	Terça-feira	4.91	-23.5488482, -46.7149299	-23.5058317, -46.7265643

Próximas etapas: Ver gráficos recomendados | New interactive sheet

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error, r2_score

# 🔹 Carregar os dados
df = pd.read_csv("Resumo_v2.csv", sep=";", na_values=["NaN", "nan"])

# 🔹 Remover linhas com preço ausente
df.dropna(subset=["price"], inplace=True)

# 🔹 Converter colunas categóricas
categorical_cols = ["ProductID", "Car", "Dia"]
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# 🔹 Criar features para tempo da viagem
df["Tempo_Viagem_Min"] = df["Tempo_Viagem_Min"].astype(float)

# 🔹 Criar features de localização (tratando erros na separação das coordenadas)
df["Lat1"], df["Lon1"] = zip(*df["Coordenadas_1"].apply(lambda x: x.split(",") if "," in x else ["NaN", "NaN"]))
df["Lat2"], df["Lon2"] = zip(*df["Coordenadas_2"].apply(lambda x: x.split(",") if "," in x else ["NaN", "NaN"]))

# 🔹 Converter coordenadas para float
df["Lat1"] = pd.to_numeric(df["Lat1"], errors='coerce')
df["Lon1"] = pd.to_numeric(df["Lon1"], errors='coerce')
df["Lat2"] = pd.to_numeric(df["Lat2"], errors='coerce')
df["Lon2"] = pd.to_numeric(df["Lon2"], errors='coerce')

# 🔹 Criar as diferenças entre latitude e longitude
df["Delta_Lat"] = df["Lat2"] - df["Lat1"]
df["Delta_Lon"] = df["Lon2"] - df["Lon1"]

# 🔹 Converter colunas de horário para formato numérico (minutos desde a meia-noite)
df["H_Inicio"] = pd.to_datetime(df["H_Inicio"], format="%H:%M:%S.%f", errors='coerce')
df["H_Fim"] = pd.to_datetime(df["H_Fim"], format="%H:%M:%S.%f", errors='coerce')

df["H_Inicio_Min"] = df["H_Inicio"].dt.hour * 60 + df["H_Inicio"].dt.minute
df["H_Fim_Min"] = df["H_Fim"].dt.hour * 60 + df["H_Fim"].dt.minute

# 🔹 Criar feature de duração da viagem
df["Duração_Viagem"] = df["H_Fim_Min"] - df["H_Inicio_Min"]

# 🔹 Selecionar features
features = ["ProductID", "Car", "Dia", "Tempo_Viagem_Min", "Distância_km", "Delta_Lat", "Delta_Lon",
            "H_Inicio_Min", "H_Fim_Min", "Duração_Viagem"]
X = df[features]
y = df["price"]

# 🔹 Remover possíveis valores NaN de forma segura
X = X.dropna().copy()
```

```
y = y.loc[X.index]  # Garantir alinhamento entre X e y

# 🔹 Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 🔹 Treinar modelo Random Forest
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# 🔹 Fazer previsões
y_pred = model.predict(X_test)

# 🔹 Avaliação do modelo
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# 🔹 Exibir métricas
print(f"Erro médio absoluto (MAE): {mae:.2f}")
print(f"Acurácia (R² score): {r2:.2f}")
```

```
Erro médio absoluto (MAE): 6.31
Acurácia (R² score): 0.94
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
from datetime import datetime
import locale
import joblib # Importar joblib para salvar o modelo e encoders

# Configurar o locale para português para obter o nome do dia da semana correto
# Tentar configurar o locale. Mesmo que falhe, o mapeamento manual lidará com o nome do dia.
try:
    locale.setlocale(locale.LC_TIME, 'pt_BR.UTF8')
except locale.Error:
    print("Aviso: Não foi possível configurar o locale 'pt_BR.UTF8'. Nomes de dias da semana podem não estar em português.")
    try:
        locale.setlocale(locale.LC_TIME, 'Portuguese_Brazil')
    except locale.Error:
        print("Aviso: Não foi possível configurar o locale 'Portuguese_Brazil'. Nomes de dias da semana podem não estar em português.")


# Função para mapear nomes de dias da semana para números (0-6)
def map_day_to_number(day_name):
    """Mapeia o nome de um dia da semana para um número (0 para Segunda, 6 para Domingo)."""
    # Mapeamento em inglês e português para maior robustez
    day_mapping = {
        'monday': 0, 'segunda-feira': 0, 'segunda': 0,
        'tuesday': 1, 'terça-feira': 1, 'terça': 1,
        'wednesday': 2, 'quarta-feira': 2, 'quarta': 2,
        'thursday': 3, 'quinta-feira': 3, 'quinta': 3,
        'friday': 4, 'sexta-feira': 4, 'sexta': 4,
        'saturday': 5, 'sábado': 5,
        'sunday': 6, 'domingo': 6
    }
    # Retorna o número mapeado ou None se o dia não for reconhecido
    return day_mapping.get(str(day_name).lower(), None)


# ================================================================================
# 🔹 CARREGAR E PRÉ-PROCESSAR OS DADOS, E TREINAR O MODELO
# ================================================================================

print("Carregando e pré-processando os dados, e treinando o modelo...")

# 🔹 Carregar os dados
try:
    df = pd.read_csv("Resumo_v2.csv", sep=";", na_values=["NaN", "nan"])
except FileNotFoundError:
    print("Erro: Arquivo 'Resumo_v2.csv' não encontrado. Certifique-se de que o arquivo está no mesmo diretório do script.")
    exit()

# 🔹 Remover linhas com preço ausente
df.dropna(subset=["price"], inplace=True)

# 🔹 Tratar a coluna 'Dia' primeiro: Mapear para números e remover a coluna original
print("Valores únicos na coluna 'Dia' antes do mapeamento:", df['Dia'].unique()) # Debugging
df['Dia_Numero'] = df['Dia'].apply(map_day_to_number)
# Remover a coluna 'Dia' original após mapear para evitar conflitos
df.drop('Dia', axis=1, inplace=True)
```

```
# Remover linhas onde o mapeamento do dia falhou (resultou em None)
df.dropna(subset=['Dia_Numero'], inplace=True)
# Converter para int (map_day_to_number retorna int ou None)
df['Dia_Numero'] = df['Dia_Numero'].astype(int)


# 🔹 Converter colunas categóricas (usando LabelEncoder apenas para ProductID)
categorical_cols_label_encoder = ["ProductID"]
label_encoders = {}
for col in categorical_cols_label_encoder:
    le = LabelEncoder()
    # Fit o encoder nos dados originais para aprender todas as categorias possíveis
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le


# 🔹 Criar features para tempo da viagem
df["Tempo_Viagem_Min"] = df["Tempo_Viagem_Min"].astype(float)

# 🔹 Criar features de localização (tratando erros na separação das coordenadas)
# Preservar colunas originais caso sejam necessárias para depuração
df['Coordenadas_1_split'] = df["Coordenadas_1"].apply(lambda x: x.split(",") if isinstance(x, str) and "," in x else ["NaN", "NaN"])
df['Coordenadas_2_split'] = df["Coordenadas_2"].apply(lambda x: x.split(",") if isinstance(x, str) and "," in x else ["NaN", "NaN"])

df["Lat1"] = df['Coordenadas_1_split'].apply(lambda x: x[0])
df["Lon1"] = df['Coordenadas_1_split'].apply(lambda x: x[1])
df["Lat2"] = df['Coordenadas_2_split'].apply(lambda x: x[0])
df["Lon2"] = df['Coordenadas_2_split'].apply(lambda x: x[1])

# 🔹 Converter coordenadas para float
df["Lat1"] = pd.to_numeric(df["Lat1"], errors='coerce')
df["Lon1"] = pd.to_numeric(df["Lon1"], errors='coerce')
df["Lat2"] = pd.to_numeric(df["Lat2"], errors='coerce')
df["Lon2"] = pd.to_numeric(df["Lon2"], errors='coerce')

# 🔹 Criar as diferenças entre latitude e longitude
df["Delta_Lat"] = df["Lat2"] - df["Lat1"]
df["Delta_Lon"] = df["Lon2"] - df["Lon1"]

# 🔹 Converter colunas de horário para formato numérico (minutos desde a meia-noite)
# Tratar horários que podem não estar no formato esperado
df["H_Inicio_dt"] = pd.to_datetime(df["H_Inicio"], format="%H:%M:%S.%f", errors='coerce')
df["H_Fim_dt"] = pd.to_datetime(df["H_Fim"], format="%H:%M:%S.%f", errors='coerce')

# Calcular minutos desde a meia-noite, tratando NaT
df["H_Inicio_Min"] = np.nan
valid_inicio_mask = df["H_Inicio_dt"].notna()
df.loc[valid_inicio_mask, "H_Inicio_Min"] = df.loc[valid_inicio_mask, "H_Inicio_dt"].dt.hour * 60 + df.loc[valid_inicio_mask, "H_Inicio_

df["H_Fim_Min"] = np.nan
valid_fim_mask = df["H_Fim_dt"].notna()
df.loc[valid_fim_mask, "H_Fim_Min"] = df.loc[valid_fim_mask, "H_Fim_dt"].dt.hour * 60 + df.loc[valid_fim_mask, "H_Fim_dt"].dt.minute

# 🔹 Criar feature de duração da viagem
df["Duração_Viagem"] = df["H_Fim_Min"] - df["H_Inicio_Min"]

# 🔹 Selecionar features
# Usar 'Dia_Numero' em vez de 'Dia'
features = ["ProductID", "Dia_Numero", "Tempo_Viagem_Min", "Distância_km", "Delta_Lat", "Delta_Lon",
            "H_Inicio_Min", "H_Fim_Min", "Duração_Viagem"]
X = df[features]
y = df["price"]

# 🔹 Remover possíveis valores NaN de forma segura
# Remover linhas onde qualquer uma das features selecionadas ou o target é NaN
# Usar apenas as features selecionadas para o dropna
df_cleaned = df.dropna(subset=features + ["price"]).copy()
X_cleaned = df_cleaned[features]
y_cleaned = df_cleaned["price"]

# 🔹 Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# 🔹 Treinar modelo Random Forest
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

print("Modelo treinado com sucesso!")

# ================================================================================
# 🔹 SALVAR MODELO E ENCODERS
# ================================================================================
```

```
MODEL_FILENAME = "modelo_treinado.joblib"
ENCODERS_FILENAME = "encoders.joblib"
FEATURES_FILENAME = "features.joblib" # Salvar também a lista de features

print(f"Salvando modelo, encoders e lista de features em '{MODEL_FILENAME}', '{ENCODERS_FILENAME}' e '{FEATURES_FILENAME}'...")

try:
    joblib.dump(model, MODEL_FILENAME)
    joblib.dump(label_encoders, ENCODERS_FILENAME)
    joblib.dump(features, FEATURES_FILENAME)
    print("Arquivos salvos com sucesso!")
except Exception as e:
    print(f"Erro ao salvar arquivos: {e}")
```

```
Aviso: Não foi possível configurar o locale 'pt_BR.UTF8'. Nomes de dias da semana podem não estar em português.
Aviso: Não foi possível configurar o locale 'Portuguese_Brazil'. Nomes de dias da semana podem não estar em português.
Carregando e pré-processando os dados, e treinando o modelo...
Valores únicos na coluna 'Dia' antes do mapeamento: ['Terça-feira' 'Segunda-feira' 'Domingo' 'Sábado' 'Sexta-feira'
 'Quinta-feira' 'Quarta-feira']
Modelo treinado com sucesso!
Salvando modelo, encoders e lista de features em 'modelo_treinado.joblib', 'encoders.joblib' e 'features.joblib'...
Arquivos salvos com sucesso!
```

```
import pandas as pd
import numpy as np
# Não precisamos mais importar train_test_split, RandomForestRegressor, LabelEncoder aqui
# se carregarmos o modelo e os encoders salvos
from datetime import datetime
import locale
from geopy.geocoders import Nominatim
import requests
import joblib # Importar joblib para carregar o modelo e encoders salvos

# Configurar o locale para português para obter o nome do dia da semana correto
# Tentar configurar o locale. Mesmo que falhe, o mapeamento manual lidará com o nome do dia.
try:
    locale.setlocale(locale.LC_TIME, 'pt_BR.UTF8')
except locale.Error:
    print("Aviso: Não foi possível configurar o locale 'pt_BR.UTF8'. Nomes de dias da semana podem não estar em português.")
    try:
        locale.setlocale(locale.LC_TIME, 'Portuguese_Brazil')
    except locale.Error:
        print("Aviso: Não foi possível configurar o locale 'Portuguese_Brazil'. Nomes de dias da semana podem não estar em português.")


# Inicializar o geocodificador Nominatim
geolocator = Nominatim(user_agent="previsao_viagem_script")

# ==============================================================================
# 🔹 Configuração da API de Roteamento HERE
# ==============================================================================
HERE_API_KEY = "zK9f9fu8wVtDG9dCY4fUvfpO6K4LDdRYtQupwo89fQ4" # Sua chave de API HERE
HERE_ROUTING_URL = "https://router.hereapi.com/v8/routes"

# Função para obter coordenadas de um endereço usando geopy
def get_coordinates(address):
    """Converte um endereço em coordenadas (latitude, longitude) usando Nominatim."""
    try:
        location = geolocator.geocode(address)
        if location:
            return (location.latitude, location.longitude)
        else:
            return None
    except Exception as e:
        print(f"Erro ao geocodificar o endereço '{address}': {e}")
        return None

# Função para obter dados de rota (distância e duração) usando a API HERE
def get_route_data(origin_coords, destination_coords, api_key):
    """
    Obtém a distância e a duração da rota entre dois pontos usando a API HERE.

    Args:
        origin_coords (tuple): Tupla (latitude, longitude) do ponto de origem.
        destination_coords (tuple): Tupla (latitude, longitude) do ponto de destino.
        api_key (str): Sua chave de API HERE.

    Returns:
        tuple: Uma tupla (distancia_km, duracao_min) ou (None, None) em caso de erro.
    """
    origin_str = f"{origin_coords[0]},{origin_coords[1]}"
    destination_str = f"{destination_coords[0]},{destination_coords[1]}"
```

```
        params = {
            "transportMode": "car",
            "origin": origin_str,
            "destination": destination_str,
            "return": "summary", # Pedimos apenas o resumo da rota
            "apikey": api_key
        }

        try:
            response = requests.get(HERE_ROUTING_URL, params=params)
            response.raise_for_status() # Levanta um erro para códigos de status HTTP ruins (4xx ou 5xx)
            data = response.json()

            if data and "routes" in data and data["routes"] and data["routes"][0] and "sections" in data["routes"][0] and data["routes"][0]["
                total_distance_meters = 0
                total_duration_seconds = 0
                for section in data["routes"][0]["sections"]:
                    if "summary" in section:
                        total_distance_meters += section["summary"].get("length", 0)
                        total_duration_seconds += section["summary"].get("duration", 0)

                distancia_km = total_distance_meters / 1000.0
                duracao_min = total_duration_seconds / 60.0

                return (distancia_km, duracao_min)
            else:
                print("Aviso: A API HERE não retornou dados de rota válidos.")
                return (None, None)

        except requests.exceptions.RequestException as e:
            print(f"Erro ao chamar a API HERE: {e}")
            return (None, None)
        except Exception as e:
            print(f"Erro ao processar a resposta da API HERE: {e}")
            return (None, None)

    # Função para mapear nomes de dias da semana para números (0-6)
    def map_day_to_number(day_name):
        """Mapeia o nome de um dia da semana para um número (0 para Segunda, 6 para Domingo)."""
        day_mapping = {
            'monday': 0, 'segunda-feira': 0, 'segunda': 0,
            'tuesday': 1, 'terça-feira': 1, 'terça': 1,
            'wednesday': 2, 'quarta-feira': 2, 'quarta': 2,
            'thursday': 3, 'quinta-feira': 3, 'quinta': 3,
            'friday': 4, 'sexta-feira': 4, 'sexta': 4,
            'saturday': 5, 'sábado': 5,
            'sunday': 6, 'domingo': 6
        }
        return day_mapping.get(str(day_name).lower(), None)


    # ================================================================================
    # 🔹 CARREGAR MODELO E ENCODERS SALVOS
    # ================================================================================
    # Estes arquivos devem ter sido criados por um script de treinamento separado.
    MODEL_FILENAME = "modelo_treinado.joblib"
    ENCODERS_FILENAME = "encoders.joblib"
    FEATURES_FILENAME = "features.joblib" # Salvar também a lista de features

    print("Carregando modelo e encoders salvos...")

    try:
        model = joblib.load(MODEL_FILENAME)
        label_encoders = joblib.load(ENCODERS_FILENAME)
        features = joblib.load(FEATURES_FILENAME)
        print("Modelo e encoders carregados com sucesso!")
    except FileNotFoundError:
        print(f"Erro: Arquivos '{MODEL_FILENAME}', '{ENCODERS_FILENAME}' ou '{FEATURES_FILENAME}' não encontrados.")
        print("Por favor, execute primeiro um script de treinamento para criar esses arquivos.")
        exit()
    except Exception as e:
        print(f"Erro ao carregar modelo ou encoders: {e}")
        exit()


    # ================================================================================
    # 🔹 Seção de Interação com o Usuário e Previsão
    # ================================================================================

    print("\n" + "="*40)
    print(" PREVISÃO DE PREÇO DE VIAGEM ")
```

```
print( = "40 + "\n")

# 🔹 Obter dados do usuário (endereços e categoria)
try:
    endereco_origem = input("Digite o endereço de origem (ex: Av. Paulista, 1000, São Paulo): ").strip()
    endereco_destino = input("Digite o endereço de destino (ex: Parque Ibirapuera, São Paulo): ").strip()

    categoria_input = input("Escolha a categoria (UberX, Comfort, Black): ").strip()
    categoria_input_lower = categoria_input.lower()

    # Mapear entrada do usuário para o que o LabelEncoder de ProductID espera (case-insensitive)
    # Precisamos encontrar a string exata que o encoder aprendeu para ProductID
    # Verificar se label_encoders["ProductID"] existe antes de usar .classes_
    if "ProductID" in label_encoders:
        categorias_validas = {str(cat).lower(): str(cat) for cat in label_encoders["ProductID"].classes_}
        if categoria_input_lower in categorias_validas:
            categoria_produto_id = categorias_validas[categoria_input_lower]
        else:
            print(f"Erro: Categoria '{categoria_input}' inválida.")
            print(f"Categorias válidas são: {', '.join(label_encoders['ProductID'].classes_)}")
            exit()
    else:
        print("Erro: LabelEncoder para 'ProductID' não encontrado nos arquivos salvos.")
        exit()


except Exception as e:
    print(f"Ocorreu um erro ao obter as entradas do usuário: {e}")
    exit()


# 🔹 Geocodificar endereços
print("\nGeocodificando endereços...")
coords_origem = get_coordinates(endereco_origem)
coords_destino = get_coordinates(endereco_destino)

if not coords_origem:
    print(f"Não foi possível encontrar coordenadas para o endereço de origem: '{endereco_origem}'.")
    exit()

if not coords_destino:
    print(f"Não foi possível encontrar coordenadas para o endereço de destino: '{endereco_destino}'.")
    exit()

lat_origem, lon_origem = coords_origem
lat_destino, lon_destino = coords_destino

# 🔹 Obter dados de rota (distância e duração) da API HERE
print("Obtendo dados de rota da API HERE...")
distancia_calculada_km, duracao_calculada_min = get_route_data(coords_origem, coords_destino, HERE_API_KEY)

if distancia_calculada_km is None or duracao_calculada_min is None:
    print("Não foi possível obter dados de rota precisos da API. Saindo.")
    exit()

print(f"Distância da rota (API): {distancia_calculada_km:.2f} km")
print(f"Duração da rota (API): {duracao_calculada_min:.1f} minutos")


# 🔹 Preparar os dados de entrada para a previsão
now = datetime.now()
# Obter o nome do dia da semana
dia_actual_name = now.strftime('%A')
hora_inicio_min = now.hour * 60 + now.minute

# Calcular horário de fim estimado (baseado na duração da API)
hora_fim_min = hora_inicio_min + duracao_calculada_min

# Mapear o nome do dia da semana para número para a previsão
dia_numero_prever = map_day_to_number(dia_actual_name)
if dia_numero_prever is None: # Verificar se o mapeamento retornou None
     print(f"Erro: Não foi possível reconhecer o nome do dia da semana '{dia_actual_name}'.")
     print("Verifique se o nome do dia retornado pelo sistema é válido (ex: Monday, Segunda-feira).")
     exit()


# Criar um DataFrame com os dados do usuário e dados calculados pela API
dados_para_prever = {
    "ProductID": [categoria_produto_id],
    "Dia_Numero": [dia_numero_prever],
    "Tempo_Viagem_Min": [duracao_calculada_min],
    "Distância_km": [distancia_calculada_km],
    "Delta_Lat": [lat_destino - lat_origem],
```

```
    "Delta_Lon": [lon_destino - lon_origem],
    "H_Inicio_Min": [hora_inicio_min],
    "H_Fim_Min": [hora_fim_min],
    "Duração_Viagem": [duracao_calculada_min]
}

df_prever = pd.DataFrame(dados_para_prever)

# 🔹 Pré-processar os dados de entrada usando os encoders carregados (apenas ProductID)

# Codificar ProductID usando o encoder carregado
# Verificar se o encoder para ProductID existe antes de usar
if "ProductID" in label_encoders:
    try:
        df_prever["ProductID"] = label_encoders["ProductID"].transform(df_prever["ProductID"])
    except ValueError as e:
        print(f"Erro ao codificar a coluna 'ProductID' para os dados de previsão: {e}")
        print(f"Verifique se o valor de entrada para 'ProductID' ('{categoria_produto_id}') existe nos dados de treino originais.")
        exit()
else:
     print("Erro: LabelEncoder para 'ProductID' não encontrado nos arquivos salvos.")
     exit()


# Selecionar as features na ordem correta
# Garantir que as colunas estejam na mesma ordem das features usadas no treino
X_prever = df_prever[features]


# 🔹 Fazer a previsão
try:
    preco_estimado = model.predict(X_prever)

    # 🔹 Exibir o resultado
    print("\n" + "="*40)
    print(f" PREVISÃO PARA: {categoria_input.upper()} ")
    print("="*40)
    print(f"Endereço de Origem: {endereco_origem}")
    print(f"Endereço de Destino: {endereco_destino}")
    print(f"Distância (API): {distancia_calculada_km:.2f} km")
    print(f"Duração (API): {duracao_calculada_min:.1f} minutos")
    print(f"Horário de Início (aprox): {now.strftime('%H:%M')}")
    print(f"Dia: {dia_actual_name} ({dia_numero_prever})")
    print("-" * 40)
    print(f"Preço Estimado: R$ {preco_estimado[0]:.2f}")
    print("="*40)

except Exception as e:
    print(f"Ocorreu um erro ao fazer a previsão: {e}")
```

```
Aviso: Não foi possível configurar o locale 'pt_BR.UTF8'. Nomes de dias da semana podem não estar em português.
Aviso: Não foi possível configurar o locale 'Portuguese_Brazil'. Nomes de dias da semana podem não estar em português.
Carregando modelo e encoders salvos...
Modelo e encoders carregados com sucesso!

========================================
 PREVISÃO DE PREÇO DE VIAGEM
========================================

Digite o endereço de origem (ex: Av. Paulista, 1000, São Paulo): Av. Paulista, 1000, São Paulo
Digite o endereço de destino (ex: Parque Ibirapuera, São Paulo): Parque Ibirapuera, São Paulo
Escolha a categoria (UberX, Comfort, Black): UberX

Geocodificando endereços...
Obtendo dados de rota da API HERE...
Distância da rota (API): 3.88 km
Duração da rota (API): 20.7 minutos

========================================
 PREVISÃO PARA: UBERX
========================================
Endereço de Origem: Av. Paulista, 1000, São Paulo
Endereço de Destino: Parque Ibirapuera, São Paulo
Distância (API): 3.88 km
Duração (API): 20.7 minutos
Horário de Início (aprox): 18:44
Dia: Sunday (6)
----------------------------------------
Preço Estimado: R$ 20.34
========================================
```

```
!pip install dash
```

```
Collecting dash
  Downloading dash-3.0.4-py3-none-any.whl.metadata (10 kB)
Collecting Flask<3.1,>=1.0.4 (from dash)
  Downloading flask-3.0.3-py3-none-any.whl.metadata (3.2 kB)
Collecting Werkzeug<3.1 (from dash)
  Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.11/dist-packages (from dash) (5.24.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.11/dist-packages (from dash) (8.7.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.11/dist-packages (from dash) (4.13.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from dash) (2.32.3)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.11/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from dash) (75.2.0)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.2.0)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.11/dist-packages (from Flask<3.1,>=1.0.4->dash) (1.9.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly>=5.0.0->dash) (9.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly>=5.0.0->dash) (24.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dist-packages (from Werkzeug<3.1->dash) (3.0.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.11/dist-packages (from importlib-metadata->dash) (3.21.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->dash) (2025.4.26)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from retrying->dash) (1.17.0)
Downloading dash-3.0.4-py3-none-any.whl (7.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.9/7.9 MB 38.5 MB/s eta 0:00:00
Downloading flask-3.0.3-py3-none-any.whl (101 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 6.4 MB/s eta 0:00:00
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 14.9 MB/s eta 0:00:00
Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Installing collected packages: Werkzeug, retrying, Flask, dash
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.1
    Uninstalling Flask-3.1.1:
      Successfully uninstalled Flask-3.1.1
Successfully installed Flask-3.0.3 Werkzeug-3.0.6 dash-3.0.4 retrying-1.3.4
```

```
import pandas as pd
import numpy as np
import joblib
from sklearn.preprocessing import LabelEncoder
import plotly.express as px
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

print("Iniciando carregamento e pré-processamento de dados...")

# Tenta carregar o DataFrame original
try:
    df_original = pd.read_csv("Resumo_v2.csv", sep=";", na_values=["NaN", "nan"])
    print(f"Arquivo 'Resumo_v2.csv' carregado com {len(df_original)} linhas.")
except FileNotFoundError:
    print("ERRO CRÍTICO: Arquivo 'Resumo_v2.csv' não encontrado. Verifique o caminho do arquivo.")
    raise

df = df_original.copy()

# 1. Remover linhas onde o preço (variável alvo) é ausente
df.dropna(subset=["price"], inplace=True)
print(f"Linhas após remover preços ausentes: {len(df)}")

# 2. Mapear 'Dia' para 'Dia_Numero' (como no script de treino)
def map_day_to_number(day_name_series):
    """Mapeia nomes de dias da semana para números (Segunda=0, Domingo=6)."""
    day_mapping = {
        'monday': 0, 'segunda-feira': 0, 'segunda': 0,
        'tuesday': 1, 'terça-feira': 1, 'terça': 1,
        'wednesday': 2, 'quarta-feira': 2, 'quarta': 2,
        'thursday': 3, 'quinta-feira': 3, 'quinta': 3,
        'friday': 4, 'sexta-feira': 4, 'sexta': 4,
        'saturday': 5, 'sábado': 5,
        'sunday': 6, 'domingo': 6
    }
    return day_name_series.astype(str).str.lower().map(day_mapping)
```

```
if 'Dia' in df.columns:
    df['Dia_Numero'] = map_day_to_number(df['Dia'])
    df.dropna(subset=['Dia_Numero'], inplace=True) # Remove linhas onde o mapeamento falhou
    df['Dia_Numero'] = df['Dia_Numero'].astype(int)
    print(f"Coluna 'Dia_Numero' criada e processada. Linhas restantes: {len(df)}")
else:
    print("AVISO: Coluna 'Dia' não encontrada. 'Dia_Numero' não será criado a partir dela.")
    if 'Dia_Numero' not in df.columns:
        df['Dia_Numero'] = -1 # Placeholder para evitar erros nos callbacks
        print("AVISO: Coluna 'Dia_Numero' também não existe. Criada como placeholder (-1).")

# Mapeamento para exibição dos dias nos dropdowns e gráficos
dia_map_display = {
    0: 'Segunda-feira', 1: 'Terça-feira', 2: 'Quarta-feira',
    3: 'Quinta-feira', 4: 'Sexta-feira', 5: 'Sábado', 6: 'Domingo',
    -1: 'Dia N/A' # Para o placeholder
}
dia_options = []
if 'Dia_Numero' in df.columns and df['Dia_Numero'].nunique() > 0 and not (df['Dia_Numero'].nunique() == 1 and df['Dia_Numero'].iloc[0] =
    unique_dias = sorted(df['Dia_Numero'].dropna().unique().astype(int))
    dia_options = [{'label': dia_map_display.get(i, f'Dia Cód. {i}'), 'value': i} for i in unique_dias]
if not dia_options: # Fallback se não houver dias válidos
    dia_options = [{'label': 'Nenhum dia disponível', 'value': -1}]


# 3. Tratar ProductID (tentar usar encoders.joblib para nomes originais)
product_id_options = []
product_id_display_map = {}

try:
    label_encoders = joblib.load("encoders.joblib")
    product_id_encoder = label_encoders.get('ProductID')
    if product_id_encoder:
        print("Encoder 'ProductID' carregado de 'encoders.joblib'.")
        if 'ProductID' in df.columns and pd.api.types.is_string_dtype(df['ProductID']):
            # Filtra valores que não estão nas classes conhecidas pelo encoder para evitar erros no transform
            known_products_mask = df['ProductID'].isin(product_id_encoder.classes_)
            df_known = df[known_products_mask].copy() # Trabalha em uma cópia para evitar SettingWithCopyWarning
            if not df_known.empty:
                 df_known.loc[:, 'ProductID_encoded'] = product_id_encoder.transform(df_known['ProductID'])
                 df = df.merge(df_known[['ProductID_encoded']], left_index=True, right_index=True, how='left')
            df['ProductID_encoded'] = df.get('ProductID_encoded', pd.Series(dtype=int)).fillna(-1).astype(int)
        elif 'ProductID' in df.columns and pd.api.types.is_numeric_dtype(df['ProductID']):
            df['ProductID_encoded'] = df['ProductID'].fillna(-1).astype(int) # Assume que já está codificado
        else:
            df['ProductID_encoded'] = -1 # Placeholder
            print("AVISO: Coluna 'ProductID' não encontrada ou tipo inesperado para aplicar encoder salvo.")

        # Cria mapa e opções para dropdown usando as classes do encoder carregado
        temp_map = {idx: cls_name for idx, cls_name in enumerate(product_id_encoder.classes_)}
        unique_encoded_ids_in_df = sorted(df['ProductID_encoded'].dropna().unique())

        product_id_options = [{'label': temp_map.get(i, f'Prod. Cód. {i}'), 'value': i} for i in unique_encoded_ids_in_df if i != -1]
        product_id_display_map = {i: temp_map.get(i, f'Prod. Cód. {i}') for i in unique_encoded_ids_in_df if i != -1}
    else:
        # Se 'ProductID' não está no dicionário de encoders, levanta exceção para cair no except
        raise FileNotFoundError("Encoder para 'ProductID' não encontrado dentro de encoders.joblib")
except FileNotFoundError:
    print("AVISO: Arquivo 'encoders.joblib' ou encoder 'ProductID' nele não encontrado. Tentando LabelEncoder nos dados atuais de 'Produ
    if 'ProductID' in df.columns:
        if pd.api.types.is_string_dtype(df['ProductID']):
            le_temp = LabelEncoder()
            df['ProductID_encoded'] = le_temp.fit_transform(df['ProductID'].astype(str))
            temp_map = {idx: cls_name for idx, cls_name in enumerate(le_temp.classes_)}
            unique_encoded_ids_in_df = sorted(df['ProductID_encoded'].dropna().unique())
            product_id_options = [{'label': temp_map.get(i, f'Prod. {cls_name}'), 'value': i} for i, cls_name in temp_map.items() if i :
            product_id_display_map = {i: temp_map.get(i, f'Prod. {cls_name}') for i, cls_name in temp_map.items() if i in unique_encoded
        elif pd.api.types.is_numeric_dtype(df['ProductID']): # Se já for numérico, usa como está
            df['ProductID_encoded'] = df['ProductID'].fillna(-1).astype(int)
            unique_ids = sorted(df['ProductID_encoded'].dropna().unique())
            product_id_options = [{'label': f'Prod. Cód. {i}', 'value': i} for i in unique_ids if i != -1]
            product_id_display_map = {i: f'Prod. Cód. {i}' for i in unique_ids if i != -1}
        else:
             df['ProductID_encoded'] = -1 # Placeholder
    else:
        df['ProductID_encoded'] = -1 # Placeholder
        print("AVISO: Coluna 'ProductID' não encontrada para fallback de LabelEncoding.")

if not product_id_options: # Garante que não está vazio para os dropdowns
    product_id_options = [{'label': 'Nenhuma categoria disponível', 'value': -1}]
if -1 not in product_id_display_map: # Garante que o valor de fallback tenha um mapa
    product_id_display_map[-1] = 'Categoria N/A'
```

```
# 4. Features Numéricas para correlação e outros usos
numerical_features_options = []
# Lista baseada no seu script de treino, ajuste conforme necessário
features_to_process_numeric = ['Tempo_Viagem_Min', 'Distância_km']

# Engenharia para H_Inicio_Min (como no seu script de treino)
if 'H_Inicio' in df.columns:
    try:
        # Tenta múltiplos formatos ou infere
        df["H_Inicio_dt"] = pd.to_datetime(df["H_Inicio"], errors='coerce')
        # Se a conversão falhar para todos, H_Inicio_dt terá NaT
        valid_times_mask = df["H_Inicio_dt"].notna()
        df.loc[valid_times_mask, "H_Inicio_Min"] = df.loc[valid_times_mask, "H_Inicio_dt"].dt.hour * 60 + \
                                                   df.loc[valid_times_mask, "H_Inicio_dt"].dt.minute
        if "H_Inicio_Min" in df.columns and df["H_Inicio_Min"].notna().any():
            features_to_process_numeric.append('H_Inicio_Min')
        else:
            if "H_Inicio_Min" not in df.columns: df['H_Inicio_Min'] = np.nan # Cria a coluna se não existir
            print("AVISO: Não foi possível converter 'H_Inicio' para 'H_Inicio_Min' ou resultou em todos NaNs.")

    except Exception as e:
        print(f"AVISO: Erro ao processar H_Inicio para H_Inicio_Min: {e}")
        if 'H_Inicio_Min' not in df.columns: df['H_Inicio_Min'] = np.nan


for feat in features_to_process_numeric:
    if feat in df.columns:
        df[feat] = pd.to_numeric(df[feat], errors='coerce')
        if df[feat].notna().any(): # Só adiciona se tiver algum valor não NaN
            numerical_features_options.append({'label': feat, 'value': feat})
    else:
        print(f"AVISO: Feature numérica '{feat}' esperada não encontrada no DataFrame.")

if not numerical_features_options: # Fallback
    numerical_features_options = [{'label': 'Nenhuma feature numérica disponível', 'value': 'none'}]


# 5. Limpeza final de NaNs para colunas que serão efetivamente usadas nos dashboards
#    Isso garante que os callbacks não quebrem por NaNs inesperados nas colunas principais.
cols_for_dashboard_check = ['price', 'Dia_Numero', 'ProductID_encoded'] + \
                           [opt['value'] for opt in numerical_features_options if opt['value'] != 'none']
if 'Distância_km' in df.columns and 'Distância_km' not in cols_for_dashboard_check:
    cols_for_dashboard_check.append('Distância_km')

# Remove duplicatas e garante que as colunas existem no df antes de tentar o dropna
cols_existing_in_df_for_dropna = [col for col in list(set(cols_for_dashboard_check)) if col in df.columns]

if cols_existing_in_df_for_dropna:
    df.dropna(subset=cols_existing_in_df_for_dropna, inplace=True)
    print(f"Linhas após dropna final das colunas do dashboard: {len(df)}")
else:
    print("AVISO: Nenhuma das colunas chave para o dashboard existe para checagem final de NaN.")


print("Pré-processamento de dados concluído.")
if df.empty:
    print("ALERTA: DataFrame está vazio após o pré-processamento! Os dashboards podem não funcionar ou exibir dados.")
else:
    print(f"DataFrame final para dashboards com {len(df)} linhas.")
    # print("\nOpções de Dia para Dropdowns:")
    # print(dia_options)
    # print("\nOpções de Categoria de Produto (ID) para Dropdowns:")
    # print(product_id_options)
    # print("\nOpções de Features Numéricas para Correlação:")
    # print(numerical_features_options)
```

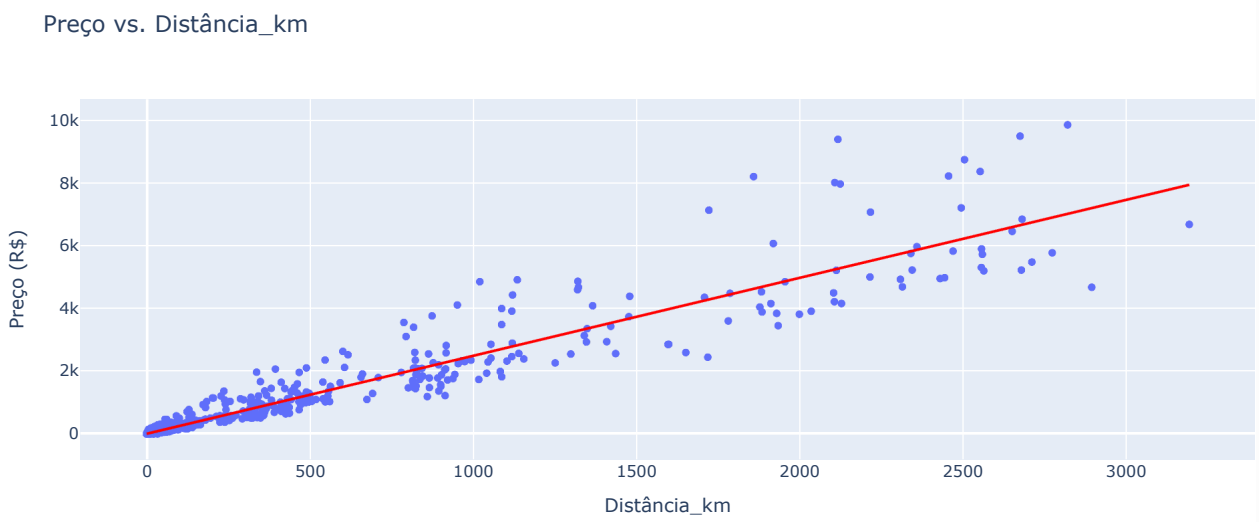
```
Iniciando carregamento e pré-processamento de dados...
Arquivo 'Resumo_v2.csv' carregado com 234021 linhas.
Linhas após remover preços ausentes: 234021
Coluna 'Dia_Numero' criada e processada. Linhas restantes: 234021
Encoder 'ProductID' carregado de 'encoders.joblib'.
<ipython-input-8-58f4f2ef3319>:133: UserWarning: Could not infer format, so each element will be parsed individually, falling back t
  df["H_Inicio_dt"] = pd.to_datetime(df["H_Inicio"], errors='coerce')
Linhas após dropna final das colunas do dashboard: 234021
Pré-processamento de dados concluído.
DataFrame final para dashboards com 234021 linhas.
```

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

# Verifica se as variáveis globais necessárias existem
if 'df' not in globals() or 'numerical_features_options' not in globals():
    print("ERRO: DataFrame 'df' ou 'numerical_features_options' não definidos.")
    print("Por favor, execute a célula de preparação de dados (dash_data_prep_viagens_v2) primeiro.")
    # Placeholders para evitar que o Dash quebre na definição do layout
    if 'df' not in globals():
        df = pd.DataFrame({'price': [0], 'none': [0]})
    if 'numerical_features_options' not in globals():
        numerical_features_options = [{'label': 'Nenhuma feature disponível', 'value': 'none'}]

# CORREÇÃO APLICADA AQUI:
app_corr = dash.Dash(__name__)
app_corr.title = "Dashboard Correlação" # Define o título da aba do navegador

app_corr.layout = html.Div([
    html.H3("Dashboard 1: Correlação do Preço com Outras Features Numéricas"),
    dcc.Dropdown(
        id='corr-feature-dropdown-d1',
        options=numerical_features_options,
        value=next((opt['value'] for opt in numerical_features_options if opt['value'] != 'none'), None),
        clearable=False,
        style={'marginBottom': '20px', 'width': '70%'}
    ),
    dcc.Graph(id='correlation-scatter-plot-d1'),
    html.Div(id='correlation-text-d1', style={'marginTop': '10px', 'fontWeight': 'bold'})
], style={'padding': '20px', 'border': '1px solid lightgrey', 'borderRadius': '5px'})

@app_corr.callback(
    [Output('correlation-scatter-plot-d1', 'figure'),
     Output('correlation-text-d1', 'children')],
    [Input('corr-feature-dropdown-d1', 'value')]
)
def update_correlation_plot_d1(selected_feature):
    if selected_feature is None or selected_feature == 'none' or selected_feature not in df.columns:
        fig_empty = px.scatter(title="Por favor, selecione uma feature numérica válida.")
        return fig_empty, "Coeficiente de Correlação: N/A"

    if df['price'].isna().all() or df[selected_feature].isna().all():
        fig_empty = px.scatter(title=f"Dados ausentes para 'price' ou '{selected_feature}'.")
        return fig_empty, "Coeficiente de Correlação: Dados ausentes"

    df_filtered = df[['price', selected_feature]].dropna()

    if df_filtered.empty or len(df_filtered) < 2:
        fig_empty = px.scatter(title=f"Não há dados suficientes para a correlação entre Preço e {selected_feature} após remover NaNs.")
        return fig_empty, "Coeficiente de Correlação: Dados insuficientes"

    try:
        fig = px.scatter(df_filtered, x=selected_feature, y='price',
                         title=f'Preço vs. {selected_feature}',
                         labels={'price': 'Preço (R$)', selected_feature: selected_feature},
                         trendline="ols",
                         trendline_color_override="red")

        correlation = df_filtered['price'].corr(df_filtered[selected_feature])
        correlation_text = f"Coeficiente de Correlação (Pearson) entre Preço e {selected_feature}: {correlation:.3f}"
    except Exception as e:
        print(f"Erro ao gerar gráfico de correlação para {selected_feature}: {e}")
        fig_empty = px.scatter(title=f"Erro ao gerar gráfico para {selected_feature}.")
        correlation_text = "Erro ao calcular correlação."

    return fig, correlation_text

if __name__ == '__main__':
    print("Para rodar o Dashboard 1 (Correlação):")
    print("Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.")
    print("Caso contrário, abra http://127.0.0.1:8051/ no seu navegador após a mensagem 'Dash is running'.")
    app_corr.run(mode='inline', port=8051, dev_tools_ui=True, dev_tools_props_check=True)
    # Alternativa: app_corr.run_server(debug=True, port=8051)
```

```
Para rodar o Dashboard 1 (Correlação):
Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.
Caso contrário, abra http://127.0.0.1:8051/ no seu navegador após a mensagem 'Dash is running'.
```

Dashboard 1: Correlação do Preço com Outras Features Numéricas

Distância_km



Coeficiente de Correlação (Pearson) entre Preço e Distância_km: 0.955

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

if 'df' not in globals() or 'dia_map_display' not in globals() or 'product_id_display_map' not in globals():
    print("ERRO: DataFrame 'df' ou mapas de display não definidos.")
    print("Por favor, execute a célula de preparação de dados (dash_data_prep_viagens_v2) primeiro.")
    # Placeholders para evitar que o Dash quebre na definição do layout
    if 'df' not in globals():
        df = pd.DataFrame({'price': [0], 'Dia_Numero': [-1], 'ProductID_encoded': [-1]})
    if 'dia_map_display' not in globals():
        dia_map_display = {-1: "N/A"}
    if 'product_id_display_map' not in globals():
        product_id_display_map = {-1: "N/A"}


# CORREÇÃO APLICADA AQUI:
app_levels = dash.Dash(__name__)
app_levels.title = "Dashboard Níveis de Preço" # Define o título da aba do navegador

app_levels.layout = html.Div([
    html.H3("Dashboard 2: Níveis de Preço por Característica Categórica"),
    dcc.Dropdown(
        id='levels-feature-dropdown-d2', # ID único
        options=[
            {'label': 'Dia da Semana', 'value': 'Dia_Numero'},
            {'label': 'Categoria do Produto', 'value': 'ProductID_encoded'}
        ],
        value='Dia_Numero', # Valor inicial
        clearable=False,
        style={'marginBottom': '20px', 'width': '70%'}
    ),
    dcc.Graph(id='price-levels-plot-d2') # ID único
], style={'padding': '20px', 'border': '1px solid lightgrey', 'borderRadius': '5px', 'marginTop': '30px'})

@app_levels.callback(
    Output('price-levels-plot-d2', 'figure'),
    [Input('levels-feature-dropdown-d2', 'value')]
)
def update_price_levels_plot_d2(selected_categorical_feature):
    if selected_categorical_feature not in df.columns or df[selected_categorical_feature].isna().all():
```

```
            return px.scatter(title=f"Feature '{selected_categorical_feature}' não encontrada ou sem dados.")

        df_display = df.copy()
        fig_title = "Distribuição de Preços"
        x_axis_label = selected_categorical_feature
        display_col = selected_categorical_feature
        category_orders_map = None

        if selected_categorical_feature == 'Dia_Numero':
            df_display['Dia_Display'] = df_display['Dia_Numero'].map(dia_map_display).fillna(df_display['Dia_Numero'].astype(str))
            x_axis_label, display_col = 'Dia da Semana', 'Dia_Display'
            unique_days_in_data = sorted(df_display['Dia_Numero'].dropna().unique().astype(int))
            ordered_days_display = [dia_map_display[i] for i in unique_days_in_data if i in dia_map_display]
            category_orders_map = {display_col: ordered_days_display}
            fig_title = 'Distribuição de Preços por Dia da Semana'

        elif selected_categorical_feature == 'ProductID_encoded':
            df_display['Produto_Display'] = df_display['ProductID_encoded'].map(product_id_display_map).fillna(df_display['ProductID_encoded
            x_axis_label, display_col = 'Categoria do Produto', 'Produto_Display'
            # Ordena pelos nomes decodificados para consistência, apenas os presentes nos dados
            present_products_display = sorted(df_display['Produto_Display'].dropna().unique())
            category_orders_map = {display_col: present_products_display}
            fig_title = 'Distribuição de Preços por Categoria do Produto'

        else:
            return px.scatter(title=f"Tipo de gráfico não implementado para {selected_categorical_feature}")

        if df_display[display_col].nunique() == 0:
            return px.scatter(title=f"Sem dados ou categorias válidas para a feature '{x_axis_label}'")

        try:
            fig = px.box(df_display, x=display_col, y='price',
                         title=fig_title,
                         labels={'price': 'Preço (R$)', display_col: x_axis_label},
                         category_orders=category_orders_map,
                         points="all",
                         notched=True)
            fig.update_xaxes(type='category')
        except Exception as e:
            print(f"Erro ao gerar boxplot para {x_axis_label}: {e}")
            fig = px.scatter(title=f"Erro ao gerar gráfico para {x_axis_label}")

        return fig

    if __name__ == '__main__':
        print("Para rodar o Dashboard 2 (Níveis de Preço):")
        print("Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.")
        print("Caso contrário, abra http://127.0.0.1:8052/ no seu navegador após a mensagem 'Dash is running'.")
        app_levels.run(mode='inline', port=8052, dev_tools_ui=True, dev_tools_props_check=True)
        # Alternativa: app_levels.run_server(debug=True, port=8052)
```

```
Para rodar o Dashboard 2 (Níveis de Preço):
Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.
Caso contrário, abra http://127.0.0.1:8052/ no seu navegador após a mensagem 'Dash is running'.
```


```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
import numpy as np

# Verifica se as variáveis globais necessárias existem
if 'df' not in globals() or 'dia_options' not in globals() or 'product_id_options' not in globals() or \
   'dia_map_display' not in globals() or 'product_id_display_map' not in globals():
    print("ERRO: DataFrame 'df' ou opções/mapas de display não definidos.")
    print("Por favor, execute a célula de preparação de dados (dash_data_prep_viagens_v2) primeiro.")
    # Placeholders para evitar que o Dash quebre na definição do layout
    if 'df' not in globals():
        df = pd.DataFrame({'price': [0], 'Distância_km': [0], 'Dia_Numero': [-1], 'ProductID_encoded': [-1], 'Tempo_Viagem_Min': [0]})
    if 'dia_options' not in globals(): dia_options = [{'label': 'N/A', 'value': -1}]
    if 'product_id_options' not in globals(): product_id_options = [{'label': 'N/A', 'value': -1}]
    if 'dia_map_display' not in globals(): dia_map_display = {-1: "N/A"}
    if 'product_id_display_map' not in globals(): product_id_display_map = {-1: "N/A"}


# CORREÇÃO APLICADA AQUI:
app_dist = dash.Dash(__name__)
app_dist.title = "Dashboard Preço vs Distância" # Define o título da aba do navegador

# Preparação dos valores para o RangeSlider de Distância
distancia_km_min_val = 0.0
distancia_km_max_val = 100.0
distancia_km_default_range_val = [0.0, 75.0] # Valor default se a coluna não existir ou for vazia
distancia_marks_step_val = 10 # Step default para as marcas

if 'Distância_km' in df.columns and not df['Distância_km'].empty and df['Distância_km'].notna().any():
    valid_distances = df['Distância_km'].replace([np.inf, -np.inf], np.nan).dropna()
    if not valid_distances.empty:
        distancia_km_min_val = float(valid_distances.min())
        distancia_km_max_val = float(valid_distances.max())
        if distancia_km_min_val < distancia_km_max_val: # Garante que min é menor que max
            distancia_km_default_range_val = [distancia_km_min_val, float(valid_distances.quantile(0.75))]
            range_span = distancia_km_max_val - distancia_km_min_val
            if range_span > 0: # Evita divisão por zero ou steps inválidos
                if range_span <= 10: distancia_marks_step_val = 1
                elif range_span <= 50: distancia_marks_step_val = 5
                else: distancia_marks_step_val = max(1, int(range_span / 10))
```

```
            else: # Se min == max
                distancia_marks_step_val = 1
        else: # Caso min >= max (ex: todos os valores são iguais ou erro nos dados)
            distancia_km_default_range_val = [distancia_km_min_val, distancia_km_max_val] # Ou valores fixos como [0,10]
            distancia_marks_step_val = 1 # Ou um step fixo
            if distancia_km_min_val > distancia_km_max_val: # Corrige se min > max
                distancia_km_max_val = distancia_km_min_val
else:
    print("AVISO: 'Distância_km' não encontrada, vazia ou toda NaN. Slider usará valores default.")


app_dist.layout = html.Div([
    html.H3("Dashboard 3: Análise Interativa - Preço vs. Distância"),
    html.Div([
        html.Div([
            html.Label("Dia da Semana:", style={'marginRight': '5px'}),
            dcc.Dropdown(
                id='dist-dia-dropdown-d3',
                options=dia_options,
                value=dia_options[0]['value'] if dia_options else None, # Pega o primeiro valor da lista de opções
                clearable=False,
                style={'width': '100%'}
            )], style={'width': '30%', 'display': 'inline-block', 'paddingRight': '10px', 'verticalAlign': 'top'}),

        html.Div([
            html.Label("Categoria do Produto:", style={'marginRight': '5px'}),
            dcc.Dropdown(
                id='dist-categoria-dropdown-d3',
                options=product_id_options,
                value=product_id_options[0]['value'] if product_id_options else None, # Pega o primeiro valor
                clearable=False,
                style={'width': '100%'}
            )], style={'width': '40%', 'display': 'inline-block', 'paddingRight': '10px', 'verticalAlign': 'top'}),
    ], style={'display': 'flex', 'marginBottom': '20px'}),

    html.Div([
        html.Label("Intervalo de Distância (km):"),
        dcc.RangeSlider(
            id='dist-slider-d3',
            min=distancia_km_min_val,
            max=distancia_km_max_val,
            step=0.5,
            # Lógica para as marcas, garantindo que o step não seja zero e que min <= max
            marks={
                i: {'label': str(i)}
                for i in range(int(round(distancia_km_min_val)),
                               int(round(distancia_km_max_val)) + 1,
                               max(1, distancia_marks_step_val)) # Garante step >= 1
            } if distancia_km_min_val <= distancia_km_max_val else {int(distancia_km_min_val): str(int(distancia_km_min_val))},
            value=distancia_km_default_range_val,
            tooltip={"placement": "bottom", "always_visible": True},
            allowCross=False,
            pushable=0.5
        )
    ], style={'marginBottom': '20px'}),
    dcc.Graph(id='interactive-price-distance-plot-d3')
], style={'padding': '20px', 'border': '1px solid lightgrey', 'borderRadius': '5px', 'marginTop': '30px'})

@app_dist.callback(
    Output('interactive-price-distance-plot-d3', 'figure'),
    [Input('dist-dia-dropdown-d3', 'value'),
     Input('dist-categoria-dropdown-d3', 'value'),
     Input('dist-slider-d3', 'value')]
)
def update_interactive_distance_plot_d3(selected_dia, selected_categoria, distance_range):
    # Checagens iniciais para valores nulos ou formatos inesperados
    if selected_dia is None or selected_categoria is None or distance_range is None or \
       not (isinstance(distance_range, list) and len(distance_range) == 2):
        return px.scatter(title="Aguardando seleção de todos os filtros ou intervalo de distância inválido.")

    # Checa se as colunas necessárias existem e não são todas NaN
    required_cols = ['Distância_km', 'Dia_Numero', 'ProductID_encoded', 'price']
    for col in required_cols:
        if col not in df.columns or df[col].isna().all():
            return px.scatter(title=f"Dados necessários (Coluna '{col}') ausentes ou inválidos.")

    # Filtra o DataFrame
    # Converte tipos para garantir a comparação correta se os valores do dropdown forem de tipos diferentes
    try:
        current_df = df[
            (df['Dia_Numero'].astype(type(selected_dia)) == selected_dia) &
            (df['ProductID_encoded'].astype(type(selected_categoria)) == selected_categoria) &
            (df['Distância_km'] >= distance_range[0]) &
```

```
            (df['Distância_km'] <= distance_range[1])
        ]
    except TypeError as e:
        print(f"Erro de tipo na filtragem: {e}. Verifique os tipos de dados e dos filtros.")
        return px.scatter(title="Erro de tipo ao filtrar dados. Verifique o console.")


    dia_nome = dia_map_display.get(selected_dia, str(selected_dia))
    categoria_nome = product_id_display_map.get(selected_categoria, str(selected_categoria))

    if current_df.empty:
        return px.scatter(title=f"Nenhum dado para: {categoria_nome} na {dia_nome} (Dist: {distance_range[0]:.1f}-{distance_range[1]:.1f}

    try:
        fig = px.scatter(current_df, x='Distância_km', y='price',
                         title=f'Preço vs. Distância para {categoria_nome} na {dia_nome} (Dist: {distance_range[0]:.1f}-{distance_range[1
                         labels={'price': 'Preço (R$)', 'Distância_km': 'Distância (km)'},
                         hover_data=['Tempo_Viagem_Min'] if 'Tempo_Viagem_Min' in current_df.columns else None,
                         trendline="ols", trendline_color_override="green")
    except Exception as e:
        print(f"Erro ao gerar gráfico interativo de distância: {e}")
        fig = px.scatter(title="Erro ao gerar gráfico. Verifique os dados e filtros.")

    return fig

if __name__ == '__main__':
    print("Para rodar o Dashboard 3 (Preço vs. Distância Interativo):")
    print("Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.")
    print("Caso contrário, abra http://127.0.0.1:8053/ no seu navegador após a mensagem 'Dash is running'.")
    app_dist.run(mode='inline', port=8053, dev_tools_ui=True, dev_tools_props_check=True)
    # Alternativa: app_dist.run_server(debug=True, port=8053)
```

```
Para rodar o Dashboard 3 (Preço vs. Distância Interativo):
Se estiver em um notebook que suporta 'inline', ele deve aparecer abaixo.
Caso contrário, abra http://127.0.0.1:8053/ no seu navegador após a mensagem 'Dash is running'.
```