```
from geopy.distance import geodesic
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

# ============================================
# 1. CARREGAMENTO DOS ARQUIVOS
# ============================================

print("📦 Carregando arquivos...")

# Carregar com delimitador ponto-e-vírgula e tratamento de tipos
dfRide = pd.read_csv("ride_v2.csv", sep=";", dtype=str)
dfRideAdd = pd.read_csv("rideaddress_v1.csv", sep=";", dtype=str)
dfRideEst = pd.read_csv("rideestimative_v3.csv", sep=";", dtype=str)
dfProduct = pd.read_csv("product.csv", sep=";", dtype=str)

# ============================================
# 2. PRÉ-PROCESSAMENTO DOS DADOS
# ============================================

print("\n🔧 Pré-processamento dos dados...")

# Uniformização de RideID
for df in [dfRide, dfRideAdd, dfRideEst]:
    df["RideID"] = df["RideID"].astype(str).str.replace(".0", "", regex=False)
```

```
# Processamento de datas
dfRide["Schedule"] = pd.to_datetime(dfRide["Schedule"], errors="coerce")
dfRide = dfRide.dropna(subset=["Schedule"])

# Derivar colunas de tempo
dfRide["Dia"] = dfRide["Schedule"].dt.weekday
dfRide["Hora"] = dfRide["Schedule"].dt.hour
dfRide["Minuto"] = dfRide["Schedule"].dt.minute
dfRide["HoraDecimal"] = dfRide["Hora"] + dfRide["Minuto"] / 60
dfRide["Faixa15min"] = dfRide["Schedule"].dt.floor("15min")

# ============================================
# 3. PROCESSAMENTO DE COORDENADAS
# ============================================

print("\n📍 Processando coordenadas...")

# Extrair origem e destino
dfRideAdd = dfRideAdd.rename(columns={"RideAddressTypeID": "OrigDest"})
dfOrigem = dfRideAdd[dfRideAdd["OrigDest"] == "1"][["RideID", "Lat", "Lng",
"Address"]].rename(
    columns={"Lat": "Lat1", "Lng": "Lng1", "Address": "AddressOrig"}
)
dfDestino = dfRideAdd[dfRideAdd["OrigDest"] == "2"][["RideID", "Lat", "Lng",
"Address"]].rename(
    columns={"Lat": "Lat2", "Lng": "Lng2", "Address": "AddressDest"}
)

# Merge de coordenadas
dfCoords = pd.merge(dfOrigem, dfDestino, on="RideID", how="inner")

# Corrigir formato numérico
```

```
for col in ["Lat1", "Lng1", "Lat2", "Lng2"]:
    dfCoords[col] = dfCoords[col].str.replace(",", ".").astype(float).round(6)

# ============================================
# 4. INTEGRAÇÃO DAS ESTIMATIVAS
# ============================================

print("\n💰 Processando estimativas de preço...")

# Merge com produtos
dfRideEst["ProductID"] = dfRideEst["ProductID"].astype(str)
dfProduct["ProductID"] = dfProduct["ProductID"].astype(str)

dfEstimadaComProduto = pd.merge(dfRideEst, dfProduct, on="ProductID", how="left")
dfEstimadaComProduto["Price"] = dfEstimadaComProduto["Price"].str.replace(",", ".").astype(float)

# Criar dicionário de estimativas
dfEstimadaSelecionada = dfEstimadaComProduto.groupby("RideID").apply(
    lambda x: dict(zip(x["Description"], x["Price"]))
).reset_index().rename(columns={0: "Estimativas"})

# ============================================
# 5. CRIAÇÃO DO DATAFRAME FINAL
# ============================================

print("\n🧩 Criando dataframe final...")

# Filtrar por IDs comuns
dfTempo = dfRide[["RideID", "Dia", "Hora", "Minuto", "HoraDecimal", "Faixa15min"]].dropna()
ids_comuns = set(dfTempo["RideID"]) & set(dfCoords["RideID"]) & set(dfEstimadaSelecionada["RideID"])
```

```
dfTempo = dfTempo[dfTempo["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)

dfCoords = dfCoords[dfCoords["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)

dfEstimadaSelecionada = dfEstimadaSelecionada[dfEstimadaSelecionada["RideID"].isin(ids_comuns)].sort_values("RideID").reset_index(drop=True)


# Concatenar horizontalmente

dfDerivado = pd.concat([

    dfTempo,

    dfCoords.drop(columns=["RideID"]),

    dfEstimadaSelecionada.drop(columns=["RideID"])

], axis=1)


# Remover linhas com coordenadas inválidas

dfDerivado = dfDerivado.dropna(subset=["Lat1", "Lng1", "Lat2", "Lng2"]).reset_index(drop=True)


# Calcular distância

dfDerivado["Distancia_km"] = dfDerivado.apply(

    lambda row: geodesic((row["Lat1"], row["Lng1"]), (row["Lat2"], row["Lng2"])).kilometers,

    axis=1

)


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

# 6. TREINAMENTO DOS MODELOS (SEÇÃO CORRIGIDA)

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


print("\n🤖 Treinando modelos de previsão...")


# Serviços alvo
```

```
servicos_alvo = ["UberX", "Comfort", "Black"]

# Listas para resultados
resultados = []
modelos_treinados = {}

for servico in servicos_alvo:
    print(f"\n🔮 Processando serviço: {servico}")

    # CORREÇÃO: Filtro corrigido para evitar erro de sintaxe
    mask = dfDerivado["Estimativas"].apply(
        lambda d: isinstance(d, dict) and servico in d and isinstance(d[servico], (int, float))
    )
    df_modelo = dfDerivado[mask].copy()

    if df_modelo.empty:
        print(f"⚠️ Nenhum dado encontrado para {servico}")
        continue

    # Definir variável alvo
    df_modelo["y"] = df_modelo["Estimativas"].apply(lambda d: d[servico])

    # Features base
    features_base = ["Distancia_km", "Dia", "Hora", "HoraDecimal", "Lat1", "Lng1", "Lat2",
"Lng2"]

    # Features auxiliares (outros serviços)
    servicos_aux = set()
    df_modelo["Estimativas"].apply(lambda d: servicos_aux.update(d.keys()) if isinstance(d, dict)
else None)
    servicos_aux.discard(servico)
```

```
for s in servicos_aux:
    nome_coluna = f"aux_{s.lower().replace(' ', '_')}"
    df_modelo[nome_coluna] = df_modelo["Estimativas"].apply(lambda d: d.get(s, np.nan))

# Preparar dados de treino
features_auxiliares = [col for col in df_modelo.columns if col.startswith("aux_")]
X = df_modelo[features_base + features_auxiliares].fillna(-1)
y = df_modelo["y"]

# Verificar tamanho do dataset
if len(X) < 100:
    print(f"⚠ Dados insuficientes para {servico} ({len(X)} registros)")
    continue

# 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
modelo = RandomForestRegressor(random_state=42)
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

# Armazenar modelo
modelos_treinados[servico] = modelo

# Calcular métricas
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

resultados.append({
```

```
        "Serviço": servico,
        "Registros": len(X),
        "MAE": round(mae, 2),
        "RMSE": round(rmse, 2),
        "R²": round(r2, 4)
    })

    print(f"✅ Modelo treinado com {len(X)} registros")
    print(f"MAE: R${mae:.2f} | RMSE: R${rmse:.2f} | R²: {r2:.4f}")

# ============================================
# 7. RESULTADOS FINAIS
# ============================================

print("\n📊 RESUMO DOS MODELOS TREINADOS")
if resultados:
    df_resultados = pd.DataFrame(resultados).sort_values(by="R²", ascending=False)
    print(df_resultados.to_string(index=False))
else:
    print("❌ Nenhum modelo foi treinado com sucesso.")

# ============================================
# 8. FUNÇÃO DE PREVISÃO POR ENDEREÇO
# ============================================

from geopy.geocoders import Nominatim

def prever_por_endereco():
    print("\n📍 PREVISÃO POR ENDEREÇO")
    print("------------------------")
```

```
# Mostrar serviços disponíveis
servicos_disponiveis = list(modelos_treinados.keys())
print("Serviços disponíveis:", ", ".join(servicos_disponiveis))

# Input do usuário
servico = input("👉 Escolha o serviço: ").strip()
if servico not in modelos_treinados:
    print("❌ Serviço não disponível")
    return

origem = input("📌 Endereço de origem: ")
destino = input("🎯 Endereço de destino: ")

# Geocodificação
geolocator = Nominatim(user_agent="uber-predict")

try:
    loc_origem = geolocator.geocode(origem)
    loc_destino = geolocator.geocode(destino)

    if not loc_origem or not loc_destino:
        print("❌ Não foi possível geocodificar um ou ambos os endereços")
        return

    coord_origem = (loc_origem.latitude, loc_origem.longitude)
    coord_destino = (loc_destino.latitude, loc_destino.longitude)

    distancia = geodesic(coord_origem, coord_destino).kilometers
    print(f"\n📏 Distância calculada: {distancia:.2f} km")

    # Preparar dados para previsão
```

```
        agora = pd.Timestamp.now()
        dados = {
            'Distancia_km': distancia,
            'Dia': agora.weekday(),
            'Hora': agora.hour,
            'HoraDecimal': agora.hour + agora.minute/60,
            'Lat1': coord_origem[0],
            'Lng1': coord_origem[1],
            'Lat2': coord_destino[0],
            'Lng2': coord_destino[1]
        }

        # Preencher features auxiliares
        modelo = modelos_treinados[servico]
        for feature in modelo.feature_names_in_:
            if feature not in dados:
                dados[feature] = -1

        # Fazer previsão
        preco = modelo.predict(pd.DataFrame([dados]))[0]
        print(f"\n💰 Preço estimado para {servico}: R$ {preco:.2f}")

    except Exception as e:
        print(f"❌ Erro durante a previsão: {str(e)}")

# Exemplo de uso
# prever_por_endereco()

📦 Carregando arquivos...

🔧 Pré-processamento dos dados...
```

📍 Processando coordenadas...

💰 Processando estimativas de preço...

🧩 Criando dataframe final...

🤖 Treinando modelos de previsão...

🔮 Processando serviço: UberX

✅ Modelo treinado com 235601 registros

MAE: R$0.48 | RMSE: R$2.27 | R^2: 0.9904

🔮 Processando serviço: Comfort

✅ Modelo treinado com 192876 registros

MAE: R$2.20 | RMSE: R$4.88 | R^2: 0.9807

🔮 Processando serviço: Black

✅ Modelo treinado com 123666 registros

MAE: R$1.83 | RMSE: R$4.85 | R^2: 0.9852

📊 RESUMO DOS MODELOS TREINADOS

Serviço	Registros	MAE	RMSE	R^2
UberX	235601	0.48	2.27	0.9904
Black	123666	1.83	4.85	0.9852
Comfort	192876	2.20	4.88	0.9807
