☑ 1. **Entendimento do Problema**

- **Tarefa**: Regressão (prever número de horas online).
- **Métrica**: RMSE (Root Mean Squared Error).
- **Inputs de produção**: apenas `gender`, `age`, `numberofkids`.
- **Input para treinamento**: `pings.csv` (dados reais de uso), agregados por dia, por cliente.

🗀 2. **Arquivos e Estrutura Esperada**

Arquivo	Descrição
customers.csv	Informações demográficas dos usuários
pings.csv	Timestamp de cada acesso dos usuários
test.csv	Data + ID dos usuários do período de teste

🧠 3. **Estratégia para Construir o Dataset de Treinamento**

Você precisa transformar os `pings.csv` em um dataset com:

- **Chave**: `id + date`
- **Target**: número de horas online por dia (sua variável `y`)
- **Features**: `gender`, `age`, `numberofkids`

💡 **Como calcular "horas online por dia" com base nos `pings.csv`:**

Você pode assumir que cada *ping* indica que o usuário estava ativo. Por exemplo:

- Se pings são espaçados por 1 minuto e houver 30 pings, você pode assumir 30 minutos de atividade.
- Para facilitar, você pode **agrupar os pings por id + data**, contar quantos pings ocorreram, e assumir uma relação proporcional com horas (por exemplo: `1 ping = 1 minuto online` → `horas = count / 60`).

🛠 4. **Pré-processamento com Pandas (Exemplo)**

```
import pandas as pd
import numpy as np

# Carregar os dados
customers = pd.read_csv("customers.csv")
pings = pd.read_csv("pings.csv")

# Converter timestamp para datetime
pings['datetime'] = pd.to_datetime(pings['timestamp'], unit='s')
pings['date'] = pings['datetime'].dt.date

# Contar pings por cliente por dia
```

```
daily_pings = pings.groupby(['id', 'date']).size().reset_index(name='ping_count')

# Converter pings para horas (exemplo: 1 ping = 1 minuto online)
daily_pings['hours_online'] = daily_pings['ping_count'] / 60

# Juntar com dados dos clientes
df = daily_pings.merge(customers, on='id', how='left')

# Features finais
df_final = df[['gender', 'age', 'numberofkids', 'hours_online']]

# Converter gênero para numérico
df_final['gender'] = df_final['gender'].map({'Male': 0, 'Female': 1})

# Separar X e y
X = df_final[['gender', 'age', 'numberofkids']]
y = df_final['hours_online']
```

🤖 5. **Modelo de Machine Learning (Baseline simples)**

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

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

# Modelo baseline
model = RandomForestRegressor()
model.fit(X_train, y_train)

# Avaliar
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse:.2f}")
```

📦 6. **Usando no backend com FastAPI**

Como você só tem `gender`, `age` e `numberofkids` no formulário do frontend, o modelo será treinado usando somente essas features. No backend:

```
@app.post("/predict_hours")
def predict_hours(user: UserFormInput):  # você define esse modelo Pydantic
    input_data = pd.DataFrame([{
        "gender": 0 if user.gender == "Male" else 1,
        "age": user.age,
```

```
            "numberofkids": user.numberofkids
        }])
        prediction = model.predict(input_data)[0]
        return {"predicted_hours": round(prediction, 2)}
```

📈 7. **Melhorias Futuras (após MVP)**

- Incorporar variáveis temporais (dia da semana, horário, sazonalidade).
- Criar clusters de usuários (ex: KMeans) e fazer modelos especializados por cluster.
- Considerar séries temporais se quiser refinar por padrão individual.