

Growth Equestre

Plano de Machine Learning (Hackathon)

3 caminhos práticos para evoluir o MVP com o que já está funcionando

10 fev 2026 | https://github.com/brodyandre/growth_equestre_hackathon_2026

Objetivo

Este PDF descreve, em linguagem simples, **3 possibilidades de Machine Learning** para o projeto Growth Equestre (MVP já operacional) e lista os **passos e arquivos do repositório** onde o time pode trabalhar para implementar.

Para quem é

Equipe do hackathon (incluindo pessoas não técnicas). O documento evita jargões e foca em **o que fazer** e **onde mexer**.

Resumo das 3 ideias

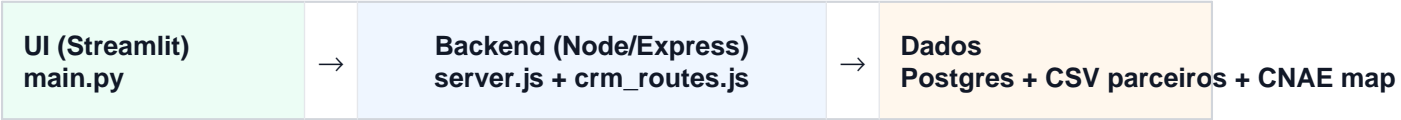
1) Propensão/Lead Scoring (ML)	Prever chance de qualificar e priorizar atendimento
2) Ranking de Parceiros	Ordenar os melhores parceiros para cada lead
3) Next Best Action (funil)	Recomendar a próxima ação/conteúdo para aumentar conversão

Importante: para o hackathon, o foco é entregar valor visível no produto. As implementações abaixo começam simples (baseline) e podem evoluir com dados reais.

1. O que já existe no MVP (visão rápida)

Hoje a aplicação já executa o ciclo básico: **captura de leads** (com eventos de funil), **consulta de parceiros** (com filtros e export), **matching** e um **painel administrativo** em Streamlit.

Arquitetura (sem complicar)



O Machine Learning entra como uma camada de inteligência: ele não muda a UI; ele **melhora as decisões** (score, ranking, próxima ação) usando os dados já coletados.

Exemplo de tela (Admin)

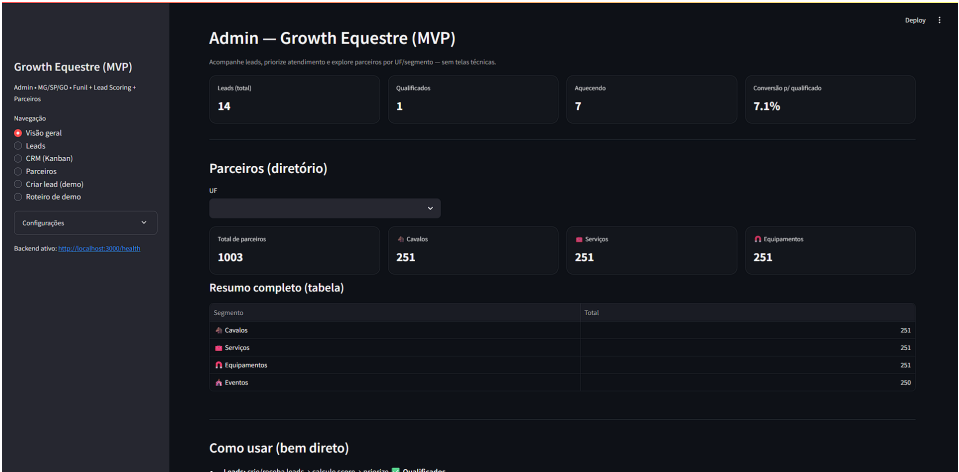


Figura: Painel Admin (visão geral).

2. Preparação: dados, eventos e mapa de arquivos

Antes de treinar qualquer modelo, precisamos garantir que os dados mínimos estão registrados. Boa notícia: boa parte disso já existe na aplicação.

2.1 Dados mínimos (o que precisamos ter)

- **Leads:** perfil (segmento, UF, cidade, orçamento, prazo).
- **Eventos do funil:** pelo menos 3 tipos (ex.: page_view, hook_complete, cta_click).
- **Resultado:** um “alvo” para treinar (ex.: status QUALIFICADO/ENVIADO ou score \geq X).
- **Parceiros:** diretório com segmento, UF e CNAE (já carregado via CSV/cache).

2.2 Onde mexer no repositório (arquivos principais)

Camada	Arquivos (caminhos típicos)	O que contém
UI (Streamlit)	main.py	Telas, filtros, botões e chamadas HTTP (safe_get/safe_post).
Backend (Node/Express)	src/server.js	Config do servidor e rotas gerais (/leads, /events, /handoff...).
Rotas CRM	src/crm_routes.js	Rotas /crm: parceiros (summary/list), matching (/crm/leads/:id/matches), next-action, notes.
Banco (Postgres)	init.sql crm_min.sql backend_crm_next_action_v2.sql	Tabelas e campos do CRM (leads, eventos, ações, notas).
CNAE (crawler/map)	tools/cnae/generate_cnae_map.py tools/cnae/cnae_keywords.csv data/cnae/cnae_map.csv	Gera mapeamento keyword → CNAE via busca online da CONCLA/IBGE (cache local).

2.3 Comandos úteis para localizar onde implementar

```
Windows (PowerShell):  
- Encontrar rota de score do lead:  
Select-String -Path .\src\*.js -Recurse -Pattern "leads.*score|/leads/.*/score"  
  
- Encontrar rota de matching:  
Select-String -Path .\src\*.js -Recurse -Pattern "leads/:id/matches|matches"  
  
- Encontrar rota de next-action:  
Select-String -Path .\src\*.js -Recurse -Pattern "next-action|/next-action"
```

3. Ideia #1: Propensão de conversão (Lead Scoring com ML)

Objetivo: prever a chance de um lead virar **QUALIFICADO** (ou **ENVIADO**) em 7/30 dias e usar isso para priorizar atendimento e automatizar o funil.

O que muda no produto (visível para quem usa)

- O botão “Calcular/Atualizar score” passa a usar um modelo (em vez de regra fixa).
- A UI mostra **Score**, **Status sugerido** e **motivos** (explicação simples).
- Lista de leads pode ser ordenada por “maior chance de qualificar”.

Passos práticos (do zero ao funcionando)

- **1) Montar dataset:** juntar tabela de leads + eventos (contagem e recência).
- **2) Definir o alvo:** label=1 se status final == QUALIFICADO ou ENVIADO.
- **3) Treinar baseline:** regressão logística ou árvore; medir AUC e precisão.
- **4) Explicação:** listar 3-5 fatores que mais puxaram o score (ex.: orçamento alto, CTA).
- **5) Integrar:** endpoint de score passa a chamar o modelo e salvar no Postgres.

Arquivos onde normalmente implementamos

- **Backend:** arquivo que define **POST /leads/:id/score** (procure em src/server.js ou rotas de leads).
- **Banco:** tabela de leads deve ter colunas para score/status/motivos (se não tiver, ajustar SQL).
- **DS:** criar pasta sugerida **tools/ml/** ou **scoring/** com script de treino e artefato do modelo (ex.: models/lead_propensity.joblib).
- **UI:** já está pronta - só consome o JSON retornado pelo endpoint (score, status, motivos).

Tela relacionada

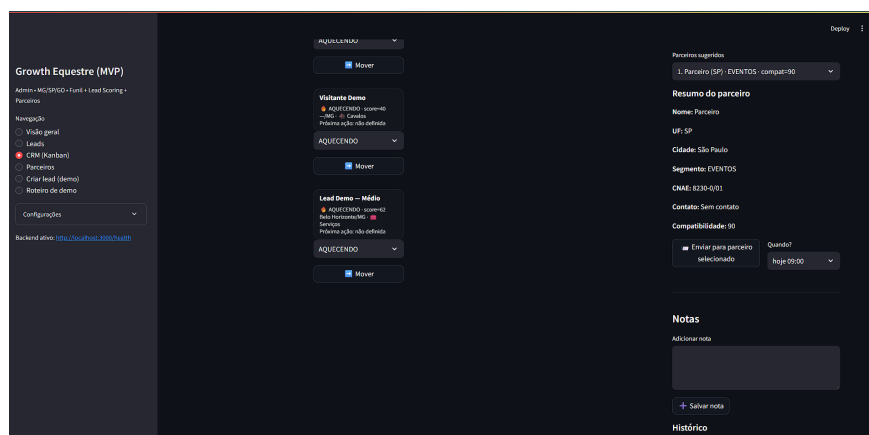


Figura: Página de Leads (onde o score aparece e é recalculado).

Dica de hackathon: comece com um modelo simples e com poucos sinais. O importante é o **score mudar com o comportamento do lead** (eventos do funil).

4. Ideia #2: Ranking de parceiros por lead (matching inteligente)

Objetivo: dado um lead, retornar uma lista **ordenada** de parceiros com maior compatibilidade (segmento, UF/cidade, CNAE e prioridade).

O que muda no produto

- A área de matching deixa de ser apenas “filtrar” e vira “recomendar Top N”.
- Cada parceiro recomendado vem com **motivos** simples (ex.: mesma UF, CNAE compatível).
- Permite medir qualidade: “o parceiro recomendado gerou handoff mais rápido?”.

Passos práticos

- **1) Definir um score de compatibilidade** (baseline): segmento igual + UF igual + CNAE compatível + prioridade.
- **2) (Opcional) Treinar um ranker** quando houver histórico (pares lead-parceiro aceitos/contatados).
- **3) Integrar no endpoint** já existente: **GET /crm/leads/:id/matches** (em src/crm_routes.js).
- **4) Ajustar a UI** (se necessário): ela já consome /crm/leads/:id/matches e exibe a lista.

Arquivos onde trabalhar

- **Backend:** src/crm_routes.js (rota /crm/leads/:id/matches).
- **Dados:** tabela/CSV de parceiros e o mapa CNAE (data/cnae/cnae_map.csv).
- **DS:** módulo de features de compatibilidade (ex.: tools/ml/partner_ranker.py).

Tela relacionada

nome	telefone	segmento	uf	cidade	prioridade	cnae	contato	nome_social	cnpj
Exemplo		Eventos	SP	Sao Paulo	1	0152-1102		Exemplo LTDA	12345678901234
Parceiro Demo 1		Serviços	MG	Uberlândia	1	9313-1100		Parceiro Demo 1 LTDA	136962359863977
Parceiro Demo 10		Eventos	MG	Belo Horizonte	1	8230-6001		Parceiro Demo 10 LTDA	136053627735169
Parceiro Demo 100		Eventos	MG	Belo Horizonte	1	0152-1102		Parceiro Demo 100 LTDA	16468704446203
Parceiro Demo 1000		Eventos	MG	Belo Horizonte	1	0152-1102		Parceiro Demo 1000 LTDA	10272714333738
Parceiro Demo 101		Serviços	GO	Anápolis	1	9313-1100		Parceiro Demo 101 LTDA	10449799398677
Parceiro Demo 102		Eventos	SP	Ribeirão Preto	1	8230-6001		Parceiro Demo 102 LTDA	104465851159515
Parceiro Demo 11		Equipamentos	GO	Anápolis	1	4647-8001		Parceiro Demo 11 LTDA	149119811940032
Parceiro Demo 110		Eventos	GO	Goiânia	1	8230-6001		Parceiro Demo 110 LTDA	16025883166039
Parceiro Demo 111		Equipamentos	SP	Campos	1	4647-8001		Parceiro Demo 111 LTDA	17363460113624

Figura: Página de Parceiros (diretório e filtros para prospecção).

Dica de hackathon: mesmo sem dados históricos, um ranking “inteligente” com CNAE + prioridade já conta como ML aplicado ao crescimento, especialmente se você explicar os sinais usados.

5. Ideia #3: Next Best Action (próxima ação do funil)

Objetivo: recomendar a próxima ação para aumentar a chance de conversão: qual conteúdo/gancho enviar, quando abordar, e por qual canal.

O que muda no produto

- Para cada lead, aparece um campo **“Próxima ação recomendada”**.
- A recomendação é baseada nos sinais do lead (perfil + eventos).
- Exemplos: “enviar calculadora”, “enviar catálogo”, “abordar no WhatsApp agora”.

Passos práticos

- 1) Definir ações possíveis (3-6 opções, simples).
- 2) Criar uma regra inicial (baseline) e depois treinar um classificador multiclases.
- 3) Usar a rota existente: **POST /crm/leads/:id/next-action** (já existe em src/crm_routes.js).
- 4) Registrar resultado: salvar recomendação e se houve clique/CTA depois.

Arquivos onde trabalhar

- **Backend:** src/crm_routes.js (rotas next-action).
- **Banco:** backend_crm_next_action_v2.sql (campos para guardar ação sugerida).
- **UI:** página de Leads pode exibir a ação recomendada (campo simples).

Tela relacionada



Figura: Roteiro de demo (bom para gerar dados e apresentar o funil).

6. Plano de execução com 5 pessoas (Data Science)

Pessoa	Responsabilidade principal	Entregas
DS1 (Dados)	Dataset + features	Script de extração + tabela final treino/val
DS2 (Lead Scoring)	Modelo de propensão	Modelo + métricas + explicação (top fatores)
DS3 (Ranking parceiros)	Score de compatibilidade / ranker	Ranking + explicação + teste com casos
DS4 (Next Action)	Recomendação de próxima ação	Ações + modelo/regra + validação offline
DS5 (Integração)	Contrato JSON + deploy	Modelo carregado no serviço + endpoint funcionando + logs

Checklist de entrega (Definition of Done)

- **Endpoint funciona:** UI chama e recebe JSON (score/ranking/ação) sem erro.
- **Explicação simples:** cada decisão tem 3-5 motivos legíveis (sem JSON bruto).
- **Teste de demo:** “Criar cenário de demo” gera leads e os modelos respondem.
- **Export:** CSV continua funcionando (para prospecção/uso no pitch).

Riscos comuns e como evitar

- **Pouco dado:** use demo + regras como baseline; foque em integração e explicação.
- **Modelos difíceis de deploy:** preferir modelo simples (joblib) e inferência rápida.
- **Equipe não técnica:** manter termos simples na UI e no pitch (probabilidade/compatibilidade).