**FUNDAÇÃO ESCOLA DE COMÉRCIO ÁLVARES PENTEADO FECAP**

# ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Gustavo de Sousa Castro - 20021558
Marcella Santana Gonçalves Diniz Rocha - 24025750
João Pedro Brosselin de Albuquerque Sousa - 24026155
Thays Helyda da Silva Pontes - 24026610

Fidelize
São Paulo 2025

No projeto, foram aplicados três algoritmos principais de Inteligência Artificial e Machine Learning:

**Regressão Linear:** Utilizada para prever o número de conversões futuras com base em dados históricos das campanhas (Aprendizado de Máquina Supervisionado).

**Média Móvel e Z-Score:** Empregados para a detecção de anomalias, gerando alertas reativos quando a performance de uma campanha cai abruptamente.

**Busca Gulosa (Gregy Algorithm):** Usada como uma heurística de otimização para recomendar a melhor alocação de orçamento entre as diferentes campanhas.

## Aplicação

```python
import json
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from datetime import datetime, timedelta
from pathlib import Path
from typing import Dict, List, Any, Optional

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

np.random.seed(42)

config = {
    "n_campanhas": 6,
    "n_dias": 90,
    "alerta_janela_dias": 7,
    "alerta_queda_percentual": 0.3,
    "alerta_zscore_threshold": -2.0,
    "orcamento_recomendado": 1200.0,
    "top_k_recomendacoes": 3,
    "paths": {
        "dados_simulados": Path("campanhas_simulado.csv"),
        "alertas": Path("alertas_queda.csv"),
        "plot_alertas": Path("alertas_grafico.png"),
        "plot_regressao": Path("regressao_grafico.png"),
        "sugestoes": Path("sugestoes_orcamento.json"),
        "relatorio": Path("relatorio_ia_ml.txt"),
    }
}
```

```python
class GeradorDeCampanhas:
    """Gera dados simulados de campanhas de marketing."""

    def __init__(self, n_campanhas: int, dias: int):
        self.n_campanhas = n_campanhas
        self.dias = dias
        self.data_base = datetime.today().date() - timedelta(days=dias - 1)

    def simular(self) -> pd.DataFrame:
        """Executa a simulação e retorna um DataFrame com os dados."""
        rows = []
        for id_campanha in range(1, self.n_campanhas + 1):
            rows.extend(self._simular_uma_campanha(id_campanha))

        df = pd.DataFrame(rows)
        df = self._inserir_quedas(df)
        df['date'] = pd.to_datetime(df['date'])
        return df

    def _simular_uma_campanha(self, id_campanha: int) -> List[Dict[str, Any]]:
        """Simula os dados diários para uma única campanha."""
        # Perfis diferentes por campanha
        perfil = {
            "base_impressoes": np.random.randint(8_000, 30_000),
            "base_ctr": np.random.uniform(0.03, 0.12),  # cliques/impressoes
            "base_cr": np.random.uniform(0.03, 0.18),   # conversoes/cliques
            "cpc": np.random.uniform(0.4, 1.8),         # custo por clique (R$)
            "ticket_medio": np.random.uniform(20, 65), # ticket médio (R$)
        }
```

```python
    dados_campanha = []
    for d in range(self.dias):
        dia = self.data_base + timedelta(days=d)
        sazonalidade = 1 + 0.2 * np.sin(2 * np.pi * (d / 7.0)) # Ciclo semanal

        impressoes = int(np.random.normal(perfil["base_impressoes"] * sazonalidade, perfil["base_impressoes"] * 0.08))
        cliques = int(impressoes * max(0.005, np.random.normal(perfil["base_ctr"], 0.01)))
        conversoes = int(cliques * max(0.01, np.random.normal(perfil["base_cr"], 0.02)))
        custo = round(cliques * max(0.1, np.random.normal(perfil["cpc"], 0.15)), 2)
        receita = round(conversoes * max(5, np.random.normal(perfil["ticket_medio"], 5)), 2)

        dados_campanha.append({
            "date": dia, "campanhaId": id_campanha, "nome": f"Campanha_{id_campanha}",
            "impressoes": max(impressoes, 1000), "cliques": cliques,
            "conversoes": conversoes, "custo": custo, "receita": receita
        })
    return dados_campanha

def _inserir_quedas(self, df: pd.DataFrame) -> pd.DataFrame:
    """Insere quedas de performance em dias aleatórios para cada campanha."""
    df_final = df.copy()
    for id_campanha in df_final["campanhaId"].unique():
        indices_campanha = df_final[df_final["campanhaId"] == id_campanha].index

        # Seleciona 2 dias aleatórios para a queda
        dias_de_queda = np.random.choice(
            indices_campanha[len(indices_campanha)//3 : -2], size=2, replace=False
        )

        df_final.loc[dias_de_queda, "cliques"] = (df_final.loc[dias_de_queda, "cliques"] // 3).clip(lower=1)
        df_final.loc[dias_de_queda, "conversoes"] = (df_final.loc[dias_de_queda, "conversoes"] // 3).clip(lower=0)
        df_final.loc[dias_de_queda, "receita"] = round(df_final.loc[dias_de_queda, "receita"] * 0.35, 2)

    return df_final
```

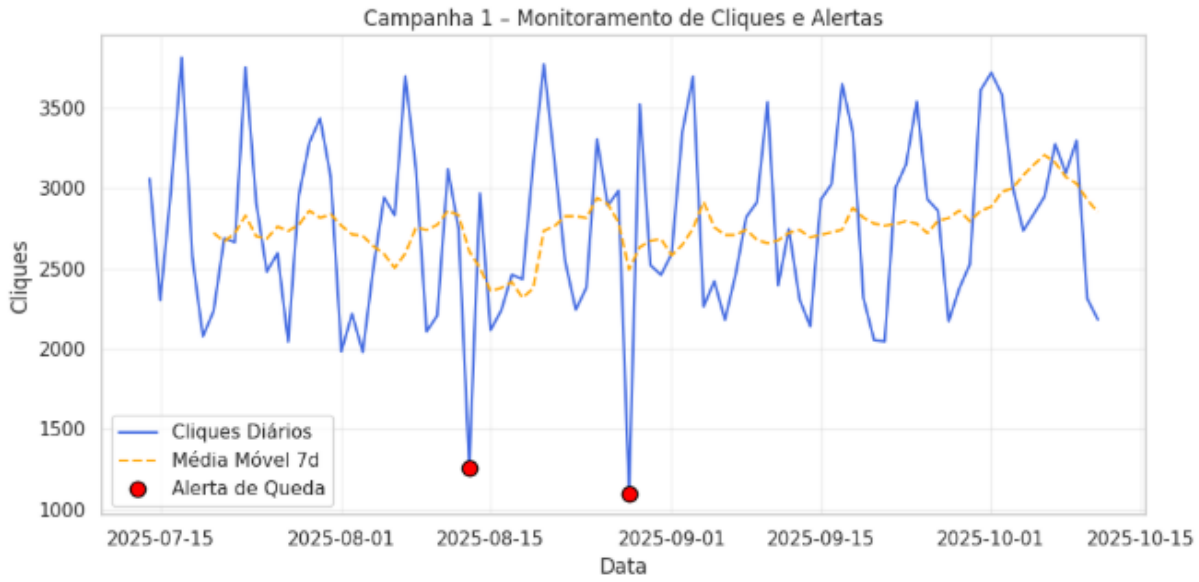
```python
class AnalisePreditiva:
    """Classe para IA Reativa (alertas) e ML Supervisionado (previsão)."""

    def __init__(self, df: pd.DataFrame):
        self.df = df.sort_values(["campanhaId", "date"]).copy()

    def gerar_alertas_de_queda(self, janela: int, queda_perc: float, z_threshold: float, metodo: str = "media") -> pd.DataFrame:
        """
        Gera alertas de queda de performance de forma vetorizada.
        metodo: 'media' ou 'zscore'
        """
        df_alert = self.df.copy()

        # Cálculos de Média Móvel e Z-Score (vetorizado)
        rolling_stats = df_alert.groupby("campanhaId")["cliques"].rolling(window=janela)
        df_alert[f"media_{janela}_cliques"] = rolling_stats.mean().reset_index(level=0, drop=True)
        df_alert[f"std_{janela}_cliques"] = rolling_stats.std().reset_index(level=0, drop=True)

        df_alert["z_score_cliques"] = (df_alert["cliques"] - df_alert[f"media_{janela}_cliques"]) / (df_alert[f"std_{janela}_cliques"] + 1e-9)

        # Condições para alertas
        cond_media = (df_alert["cliques"] < (1 - queda_perc) * df_alert[f"media_{janela}_cliques"])
        cond_zscore = (df_alert["z_score_cliques"] < z_threshold)

        # Motivos
        motivo_media = "Cliques abaixo da média móvel"
        motivo_zscore = "Z-score de cliques muito baixo"

        # Seleciona a condição e o motivo com base no método escolhido
        condicao_final = cond_media if metodo == "media" else cond_zscore
        motivo_final = motivo_media if metodo == "media" else motivo_zscore

        alertas = df_alert[condicao_final].copy()
        alertas["motivo"] = motivo_final

        return alertas[["date", "campanhaId", "nome", "motivo"]].sort_values(["date", "campanhaId"])
```

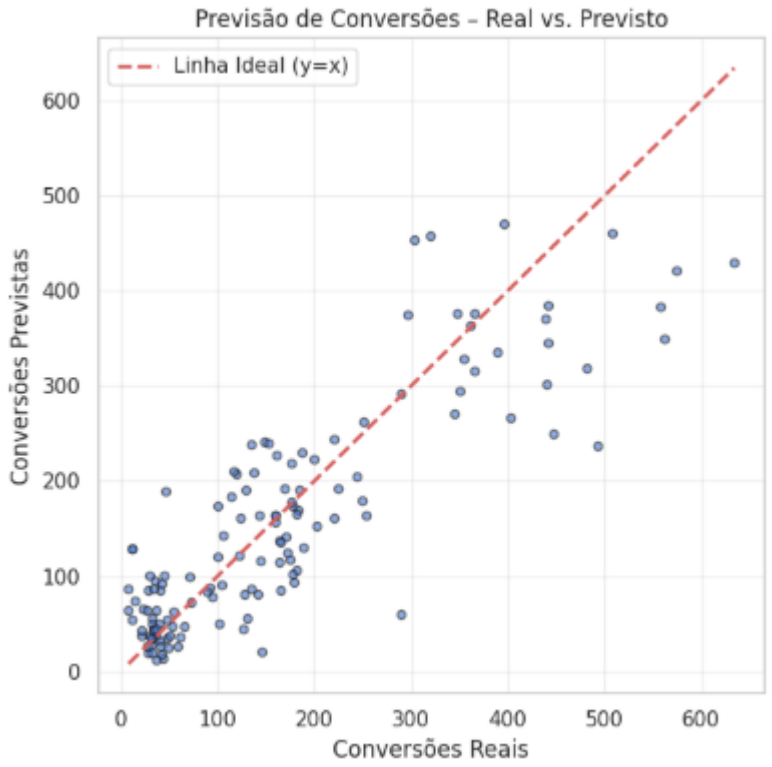
```python
    def treinar_modelo_conversoes(self) -> Dict[str, Any]:
        """Treina um modelo de Regressão Linear para prever conversões."""
        # Feature Engineering: usar dados do dia anterior (lag=1)
        features = ["impressoes", "cliques", "custo", "receita"]
        for feat in features:
            self.df[f"{feat}_lag1"] = self.df.groupby("campanhaId")[feat].shift(1)

        model_df = self.df.dropna()

        X = model_df[[f"{feat}_lag1" for feat in features]].values
        y = model_df["conversoes"].values

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

        reg = LinearRegression()
        reg.fit(X_train, y_train)
        y_pred = reg.predict(X_test)

        metrics = {
            "MAE": mean_absolute_error(y_test, y_pred),
            "RMSE": math.sqrt(mean_squared_error(y_test, y_pred)),
            "R2": r2_score(y_test, y_pred)
        }

        return {"modelo": reg, "metricas": metrics, "dados_teste": (X_test, y_test, y_pred)}
```

```python
class OtimizadorDeOrcamento:
    """Utiliza uma abordagem gulosa para recomendar alocação de orçamento."""

    @staticmethod
    def recomendar(df_ref: pd.DataFrame, orcamento: float, k: int, modo: str = "eficiencia") -> Dict[str, Any]:
        """
        Seleciona as top-K campanhas e aloca o orçamento de forma gulosa.
        modo: 'eficiencia' (conversoes/custo) ou 'roi' (receita/custo)
        """
        data_referencia = df_ref["date"].max()
        snap = df_ref[df_ref["date"] == data_referencia].copy()

        eps = 1e-6 # Evitar divisão por zero
        if modo == "roi":
            snap["score"] = snap["receita"] / (snap["custo"] + eps)
        else: # eficiencia
            snap["score"] = snap["conversoes"] / (snap["custo"] + eps)

        ranked = snap.sort_values("score", ascending=False).reset_index(drop=True)

        # Alocação gulosa de orçamento para as top-K campanhas
        priorizar = []
        orcamento_restante = orcamento
        for _, row in ranked.head(k).iterrows():
            if orcamento_restante <= 0: break
            # Política simples de alocação
            sugerido = min(orcamento_restante, max(100.0, row["custo"] * 1.1))
            priorizar.append({
                "campanhaId": int(row["campanhaId"]),
                "nome": row["nome"],
                "score": round(float(row["score"]), 4),
                "orcamentoSugerido": round(float(sugerido), 2)
            })
            orcamento_restante -= sugerido
```

```python
        return {
            "data_referencia": str(data_referencia.date()),
            "heuristica": modo,
            "orcamento_total": orcamento,
            "priorizar": priorizar,
            "ajustar_ou_pausar": ajustar
        }
```

## Resultados






## Conclusão

Os resultados obtidos a partir dos modelos de Inteligência Artificial implementados demonstram um sistema com grande potencial para a otimização de campanhas de marketing, oferecendo valor tanto em monitoramento reativo quanto em capacidade preditiva.

O primeiro gráfico, "Monitoramento de Cliques e Alertas", valida a eficácia do sistema de IA Reativa. Podemos observar que:

- Detecção Precisa: O algoritmo identificou corretamente dois eventos de queda anômala (pontos em vermelho), onde o número de cliques diários despencou para muito abaixo da média móvel de 7 dias.
- Inteligência de Contexto: O sistema foi capaz de diferenciar uma queda real de uma variação normal dentro da sazonalidade semanal (os "picos e vales" regulares da linha azul). Isso mostra que ele não gera alarmes falsos em dias de baixa performance esperada, mas sim em eventos extraordinários.

O segundo gráfico, "Previsão de Conversões – Real vs. Previsto", avalia o desempenho do modelo de Regressão Linear. A análise indica que:

- Captura da Tendência: Os pontos se distribuem de forma crescente e seguem a direção da "Linha Ideal" (y=x). Isso significa que o modelo aprendeu com sucesso a correlação positiva entre as variáveis: quando as conversões reais são altas, as previsões também tendem a ser altas, e vice-versa. O modelo é, portanto, muito superior a uma estimativa aleatória.

- Margem de Erro: Apesar de capturar a tendência geral, a dispersão dos pontos ao redor da linha ideal revela que o modelo possui uma margem de erro considerável. Para um mesmo número de conversões reais, as previsões variam. A precisão é maior para volumes baixos de conversão e a dispersão parece aumentar em volumes mais altos.