# 🤖 Integração com Code LLM

Este guia detalha como integrar a Automação de Prospecção com Code LLMs para análise inteligente, geração de conteúdo e otimização de campanhas.

## 🎯 Visão Geral

### Casos de Uso com Code LLM

1. **Análise de Dados**: Insights sobre negócios coletados
2. **Geração de Mensagens**: Mensagens personalizadas por segmento
3. **Otimização de Campanhas**: Análise de performance e sugestões
4. **Classificação Automática**: Categorização inteligente de negócios
5. **Detecção de Padrões**: Identificação de oportunidades de mercado

### Arquitetura de Integração

```
Automação de Prospecção → API/Database → Code LLM → Insights/Content
                                              ↓
                                    Feedback Loop para Otimização
```

## 🔧 Configuração Básica

### 1. Instalar Dependências Adicionais

```
# Ativar ambiente virtual
source venv/bin/activate

# Instalar bibliotecas para LLM
pip install openai anthropic langchain tiktoken
```

### 2. Configurar Variáveis de Ambiente

```
# Adicionar ao arquivo .env
nano .env
```

```
# Configurações de LLM
OPENAI_API_KEY=sua_chave_openai_aqui
ANTHROPIC_API_KEY=sua_chave_anthropic_aqui
LLM_PROVIDER=openai  # ou anthropic, local, etc.
LLM_MODEL=gpt-4  # ou claude-3, llama-2, etc.
LLM_MAX_TOKENS=2000
LLM_TEMPERATURE=0.7

# Configurações específicas
ENABLE_LLM_ANALYSIS=true
ENABLE_AUTO_MESSAGE_GENERATION=true
LLM_ANALYSIS_INTERVAL=3600  # segundos
```

### 3. Criar Módulo de Integração LLM

```
# Criar arquivo para integração
nano llm_integration.py
```

```python
# llm_integration.py
import os
import json
import logging
from typing import List, Dict, Optional
from datetime import datetime
import openai
from anthropic import Anthropic
from models import Business, MessageLog, SessionLocal

logger = logging.getLogger(__name__)

class LLMIntegration:
    def __init__(self):
        self.provider = os.getenv('LLM_PROVIDER', 'openai')
        self.model = os.getenv('LLM_MODEL', 'gpt-4')
        self.max_tokens = int(os.getenv('LLM_MAX_TOKENS', 2000))
        self.temperature = float(os.getenv('LLM_TEMPERATURE', 0.7))

        if self.provider == 'openai':
            openai.api_key = os.getenv('OPENAI_API_KEY')
        elif self.provider == 'anthropic':
            self.anthropic = Anthropic(api_key=os.getenv('ANTHROPIC_API_KEY'))

    def generate_completion(self, prompt: str) -> str:
        """Gera resposta usando o LLM configurado"""
        try:
            if self.provider == 'openai':
                response = openai.ChatCompletion.create(
                    model=self.model,
                    messages=[{"role": "user", "content": prompt}],
                    max_tokens=self.max_tokens,
                    temperature=self.temperature
                )
                return response.choices[0].message.content

            elif self.provider == 'anthropic':
                response = self.anthropic.messages.create(
                    model=self.model,
                    max_tokens=self.max_tokens,
                    temperature=self.temperature,
                    messages=[{"role": "user", "content": prompt}]
                )
                return response.content[0].text

        except Exception as e:
            logger.error(f"Erro ao gerar completion: {e}")
            return None

    def analyze_business_data(self, businesses: List[Business]) -> Dict:
        """Analisa dados de negócios usando LLM"""

        # Preparar dados para análise
        business_data = []
        for business in businesses:
            business_data.append({
                'name': business.name,
                'category': business.category,
                'rating': business.rating,
                'reviews_count': business.reviews_count,
                'address': business.address,
                'phone': business.phone
```

```
            })

        prompt = f"""
        Analise os seguintes dados de {len(businesses)} negócios coletados em Curitiba:

        {json.dumps(business_data, indent=2, ensure_ascii=False)}

        Forneça uma análise detalhada incluindo:

        1. **Distribuição por Categorias**: Quais categorias são mais comuns?
        2. **Análise de Qualidade**: Padrões nas avaliações e número de reviews
        3. **Oportunidades de Mercado**: Segmentos com potencial
        4. **Recomendações de Abordagem**: Como abordar cada categoria
        5. **Insights Geográficos**: Padrões por região/bairro
        6. **Timing Recomendado**: Melhores horários para contato por categoria

        Responda em formato JSON estruturado.
        """

        response = self.generate_completion(prompt)

        try:
            return json.loads(response)
        except:
            return {"analysis": response, "structured": False}

    def generate_personalized_message(self, business: Business, campaign_type: str = "p
rospeccao") -> str:
        """Gera mensagem personalizada para um negócio específico"""

        prompt = f"""
        Crie uma mensagem de {campaign_type} profissional e personalizada para:

        **Negócio**: {business.name}
        **Categoria**: {business.category}
        **Localização**: {business.address}
        **Avaliação**: {business.rating} estrelas ({business.reviews_count} avaliações)

        **Diretrizes**:
        - Máximo 160 caracteres (SMS/WhatsApp)
        - Tom profissional mas amigável
        - Mencionar algo específico sobre o negócio
        - Incluir proposta de valor clara
        - Personalizar para a categoria do negócio
        - Incluir call-to-action

        **Contexto**: Somos uma empresa de marketing digital em Curitiba oferecendo
serviços de automação e prospecção.

        Retorne apenas a mensagem, sem explicações adicionais.
        """

        return self.generate_completion(prompt)

    def optimize_campaign_strategy(self, campaign_results: Dict) -> Dict:
        """Analisa resultados de campanha e sugere otimizações"""

        prompt = f"""
        Analise os resultados da campanha de prospecção:

        {json.dumps(campaign_results, indent=2, ensure_ascii=False)}

        Forneça recomendações de otimização para:
```

```
        1. **Taxa de Resposta**: Como melhorar o engajamento
        2. **Segmentação**: Ajustes na segmentação de público
        3. **Timing**: Otimização de horários de envio
        4. **Mensagens**: Melhorias no conteúdo das mensagens
        5. **Frequência**: Ajustes na frequência de contato
        6. **Próximos Passos**: Estratégias para follow-up

        Responda em formato JSON com recomendações específicas e métricas esperadas.
        """

        response = self.generate_completion(prompt)

        try:
            return json.loads(response)
        except:
            return {"recommendations": response, "structured": False}

    def classify_business_potential(self, business: Business) -> Dict:
        """Classifica o potencial de conversão de um negócio"""

        prompt = f"""
        Avalie o potencial de conversão deste negócio para serviços de marketing digit-
al:

        **Nome**: {business.name}
        **Categoria**: {business.category}
        **Avaliação**: {business.rating}/5 ({business.reviews_count} reviews)
        **Localização**: {business.address}

        Classifique em:
        - **Potencial**: Alto/Médio/Baixo
        - **Prioridade**: 1-10
        - **Abordagem Recomendada**: Estratégia específica
        - **Proposta de Valor**: O que destacar
        - **Objeções Prováveis**: Possíveis resistências
        - **Timing Ideal**: Melhor momento para contato

        Responda em formato JSON.
        """

        response = self.generate_completion(prompt)

        try:
            return json.loads(response)
        except:
            return {"classification": response, "structured": False}

# Instância global
llm = LLMIntegration()
```

# 📊 Implementação de Análises

## 1. Análise Automática de Dados Coletados

```python
# Adicionar ao app.py
from llm_integration import llm

@app.route('/api/llm_analysis')
def get_llm_analysis():
    """Endpoint para análise LLM dos dados"""
    if not os.getenv('ENABLE_LLM_ANALYSIS', 'false').lower() == 'true':
        return jsonify({'error': 'Análise LLM não habilitada'})

    db = SessionLocal()
    try:
        # Buscar negócios recentes
        businesses = db.query(Business).limit(100).all()

        if not businesses:
            return jsonify({'error': 'Nenhum dado para análise'})

        # Gerar análise
        analysis = llm.analyze_business_data(businesses)

        return jsonify({
            'success': True,
            'analysis': analysis,
            'businesses_analyzed': len(businesses),
            'generated_at': datetime.now().isoformat()
        })

    except Exception as e:
        logger.error(f"Erro na análise LLM: {e}")
        return jsonify({'error': str(e)})
    finally:
        db.close()
```

## 2. Geração Automática de Mensagens

```python
@app.route('/api/generate_messages', methods=['POST'])
def generate_messages():
    """Gera mensagens personalizadas usando LLM"""
    if not os.getenv('ENABLE_AUTO_MESSAGE_GENERATION', 'false').lower() == 'true':
        return jsonify({'error': 'Geração automática não habilitada'})

    data = request.json
    category_filter = data.get('category_filter')
    max_messages = int(data.get('max_messages', 10))

    db = SessionLocal()
    try:
        # Buscar negócios para gerar mensagens
        query = db.query(Business).filter(Business.phone.isnot(None))

        if category_filter:
            query = query.filter(Business.category.contains(category_filter))

        businesses = query.limit(max_messages).all()

        generated_messages = []

        for business in businesses:
            # Gerar mensagem personalizada
            message = llm.generate_personalized_message(business)

            # Classificar potencial
            potential = llm.classify_business_potential(business)

            generated_messages.append({
                'business_id': business.id,
                'business_name': business.name,
                'category': business.category,
                'message': message,
                'potential': potential,
                'phone': business.phone
            })

        return jsonify({
            'success': True,
            'messages': generated_messages,
            'total_generated': len(generated_messages)
        })

    except Exception as e:
        logger.error(f"Erro na geração de mensagens: {e}")
        return jsonify({'error': str(e)})
    finally:
        db.close()
```

# 🎯 Casos de Uso Avançados

## 1. Análise de Sentimento de Respostas

```
def analyze_response_sentiment(response_text: str) -> Dict:
    """Analisa sentimento das respostas recebidas"""

    prompt = f"""
    Analise o sentimento e intenção da seguinte resposta de um prospect:

    "{response_text}"

    Classifique:
    - **Sentimento**: Positivo/Neutro/Negativo
    - **Interesse**: Alto/Médio/Baixo/Nenhum
    - **Próxima Ação**: Recomendação específica
    - **Urgência**: Imediata/Média/Baixa
    - **Objeções**: Identificar resistências

    Responda em formato JSON.
    """

    return llm.generate_completion(prompt)
```

## 2. Otimização de Horários de Envio

```
def optimize_sending_schedule(historical_data: List[Dict]) -> Dict:
    """Otimiza horários de envio baseado em dados históricos"""

    prompt = f"""
    Baseado nos dados históricos de campanhas:

    {json.dumps(historical_data, indent=2)}

    Recomende:
    - **Melhores Horários**: Por dia da semana
    - **Horários por Categoria**: Segmentação por tipo de negócio
    - **Frequência Ideal**: Intervalo entre mensagens
    - **Sazonalidade**: Padrões mensais/sazonais

    Forneça cronograma otimizado em formato JSON.
    """

    return llm.generate_completion(prompt)
```

### 3. Geração de Follow-up Inteligente

```python
def generate_followup_sequence(business: Business, interaction_history: List[Dict]) ->
List[str]:
    """Gera sequência de follow-up personalizada"""

    prompt = f"""
    Crie uma sequência de 3 mensagens de follow-up para:

    **Negócio**: {business.name} ({business.category})
    **Histórico de Interações**: {json.dumps(interaction_history)}

    Cada mensagem deve:
    - Ser progressivamente mais específica
    - Abordar possíveis objeções
    - Incluir social proof relevante
    - Ter call-to-action claro

    Retorne array JSON com as 3 mensagens.
    """

    response = llm.generate_completion(prompt)

    try:
        return json.loads(response)
    except:
        return [response]
```

# 🔄 Automação com LLM

## 1. Análise Automática Periódica

```
# Criar arquivo de tarefa automática
# nano llm_tasks.py

import schedule
import time
from llm_integration import llm
from models import Business, SessionLocal

def automated_analysis():
    """Executa análise automática dos dados"""
    logger.info("Iniciando análise automática com LLM")

    db = SessionLocal()
    try:
        # Buscar dados recentes
        businesses = db.query(Business).limit(200).all()

        if businesses:
            # Gerar análise
            analysis = llm.analyze_business_data(businesses)

            # Salvar análise em arquivo
            with open(f'logs/llm_analysis_{datetime.now().strftime("%Y%m%d_%H%M")}.json
', 'w') as f:
                json.dump(analysis, f, indent=2, ensure_ascii=False)

            logger.info(f"Análise concluída para {len(businesses)} negócios")

    except Exception as e:
        logger.error(f"Erro na análise automática: {e}")
    finally:
        db.close()

# Agendar análise a cada 6 horas
schedule.every(6).hours.do(automated_analysis)

def run_scheduler():
    while True:
        schedule.run_pending()
        time.sleep(60)

if __name__ == "__main__":
    run_scheduler()
```

## 2. Integração com Webhook para Respostas

```
@app.route('/webhook/whatsapp_response', methods=['POST'])
def handle_whatsapp_response():
    """Processa respostas do WhatsApp com análise LLM"""

    data = request.json
    phone = data.get('phone')
    message = data.get('message')

    if not phone or not message:
        return jsonify({'error': 'Dados incompletos'})

    # Analisar sentimento da resposta
    sentiment_analysis = llm.analyze_response_sentiment(message)

    # Buscar negócio correspondente
    db = SessionLocal()
    business = db.query(Business).filter(Business.phone == phone).first()

    if business:
        # Gerar resposta automática baseada no sentimento
        if sentiment_analysis.get('interest') == 'Alto':
            # Agendar follow-up personalizado
            followup = llm.generate_followup_sequence(business, [{'message': message, '
timestamp': datetime.now()}])

            # Salvar para processamento posterior
            # ... implementar lógica de agendamento

    db.close()

    return jsonify({'success': True, 'analysis': sentiment_analysis})
```

# 📈 Dashboard com Insights LLM

## 1. Endpoint para Dashboard

```python
@app.route('/api/llm_insights')
def get_llm_insights():
    """Retorna insights gerados por LLM para dashboard"""

    db = SessionLocal()
    try:
        # Estatísticas básicas
        total_businesses = db.query(Business).count()
        categories = db.query(Business.category).distinct().all()

        # Gerar insights rápidos
        quick_insights = llm.generate_completion(f"""
        Baseado em {total_businesses} negócios coletados em {len(categories)} categori-
as diferentes,
        forneça 5 insights rápidos sobre oportunidades de mercado em Curitiba.

        Responda em formato de lista JSON com insights concisos.
        """)

        return jsonify({
            'total_businesses': total_businesses,
            'categories_count': len(categories),
            'quick_insights': json.loads(quick_insights) if quick_insights else [],
            'generated_at': datetime.now().isoformat()
        })

    except Exception as e:
        return jsonify({'error': str(e)})
    finally:
        db.close()
```

### 2. Template HTML para Insights

```
<!-- Adicionar ao template base.html -->
<div class="llm-insights-section" id="llm-insights">
    <h3>🤖 Insights Inteligentes</h3>
    <div id="insights-content">
        <p>Carregando insights...</p>
    </div>
    <button onclick="refreshInsights()" class="btn btn-primary">
        Atualizar Insights
    </button>
</div>

<script>
function loadLLMInsights() {
    fetch('/api/llm_insights')
        .then(response => response.json())
        .then(data => {
            const content = document.getElementById('insights-content');

            if (data.quick_insights) {
                let html = '<ul class="insights-list">';
                data.quick_insights.forEach(insight => {
                    html += `<li class="insight-item">${insight}</li>`;
                });
                html += '</ul>';
                content.innerHTML = html;
            }
        })
        .catch(error => {
            console.error('Erro ao carregar insights:', error);
        });
}

function refreshInsights() {
    document.getElementById('insights-content').innerHTML = '<p>Gerando novos in-
sights...</p>';
    loadLLMInsights();
}

// Carregar insights ao carregar a página
document.addEventListener('DOMContentLoaded', loadLLMInsights);
</script>
```

## 🔧 Configuração para Diferentes Provedores

### 1. OpenAI GPT

```
# Configuração específica para OpenAI
OPENAI_CONFIG = {
    'api_key': os.getenv('OPENAI_API_KEY'),
    'model': 'gpt-4',
    'max_tokens': 2000,
    'temperature': 0.7,
    'top_p': 1,
    'frequency_penalty': 0,
    'presence_penalty': 0
}
```

### 2. Anthropic Claude

```
# Configuração específica para Claude
ANTHROPIC_CONFIG = {
    'api_key': os.getenv('ANTHROPIC_API_KEY'),
    'model': 'claude-3-sonnet-20240229',
    'max_tokens': 2000,
    'temperature': 0.7
}
```

### 3. LLM Local (Ollama)

```
# Configuração para LLM local
import requests

class LocalLLM:
    def __init__(self, base_url="http://localhost:11434"):
        self.base_url = base_url
        self.model = os.getenv('LOCAL_LLM_MODEL', 'llama2')

    def generate_completion(self, prompt: str) -> str:
        response = requests.post(
            f"{self.base_url}/api/generate",
            json={
                'model': self.model,
                'prompt': prompt,
                'stream': False
            }
        )

        if response.status_code == 200:
            return response.json()['response']
        return None
```

# 📊 Métricas e Monitoramento

## 1. Tracking de Performance LLM

```
class LLMMetrics:
    def __init__(self):
        self.metrics = {
            'total_requests': 0,
            'successful_requests': 0,
            'failed_requests': 0,
            'average_response_time': 0,
            'token_usage': 0,
            'cost_tracking': 0
        }

    def track_request(self, success: bool, response_time: float, tokens: int, cost: flo
at):
        self.metrics['total_requests'] += 1

        if success:
            self.metrics['successful_requests'] += 1
        else:
            self.metrics['failed_requests'] += 1

        # Calcular média de tempo de resposta
        current_avg = self.metrics['average_response_time']
        total_requests = self.metrics['total_requests']
        self.metrics['average_response_time'] = (current_avg * (total_requests - 1) +
response_time) / total_requests

        self.metrics['token_usage'] += tokens
        self.metrics['cost_tracking'] += cost

    def get_metrics(self) -> Dict:
        return self.metrics.copy()

# Instância global de métricas
llm_metrics = LLMMetrics()
```

## 2. Dashboard de Métricas LLM

```
@app.route('/api/llm_metrics')
def get_llm_metrics():
    """Retorna métricas de uso do LLM"""
    return jsonify(llm_metrics.get_metrics())
```

# 🚀 Exemplos Práticos

## 1. Script de Análise Completa

```python
# analyze_campaign.py
from llm_integration import llm
from models import Business, MessageLog, SessionLocal

def run_complete_analysis():
    """Executa análise completa com LLM"""

    db = SessionLocal()

    # 1. Analisar dados de negócios
    businesses = db.query(Business).all()
    business_analysis = llm.analyze_business_data(businesses)

    # 2. Analisar performance de campanhas
    messages = db.query(MessageLog).all()
    campaign_data = [
        {
            'business_name': msg.business_name,
            'sent_at': msg.sent_at.isoformat() if msg.sent_at else None,
            'success': msg.message_sent,
            'error': msg.error_message
        }
        for msg in messages
    ]

    campaign_analysis = llm.optimize_campaign_strategy({'messages': campaign_data})

    # 3. Gerar relatório consolidado
    report = {
        'business_analysis': business_analysis,
        'campaign_analysis': campaign_analysis,
        'generated_at': datetime.now().isoformat(),
        'total_businesses': len(businesses),
        'total_messages': len(messages)
    }

    # Salvar relatório
    with open(f'reports/llm_analysis_{datetime.now().strftime("%Y%m%d")}.json', 'w') as
f:
        json.dump(report, f, indent=2, ensure_ascii=False)

    print("Análise completa concluída!")
    return report

if __name__ == "__main__":
    run_complete_analysis()
```

## 2. Geração de Campanha Inteligente

```python
# smart_campaign.py
def create_smart_campaign(target_category: str = None):
    """Cria campanha inteligente usando LLM"""

    db = SessionLocal()

    # Buscar negócios alvo
    query = db.query(Business).filter(Business.phone.isnot(None))
    if target_category:
        query = query.filter(Business.category.contains(target_category))

    businesses = query.limit(50).all()

    campaign_messages = []

    for business in businesses:
        # Classificar potencial
        potential = llm.classify_business_potential(business)

        # Gerar mensagem apenas para alto potencial
        if potential.get('potential') == 'Alto':
            message = llm.generate_personalized_message(business)

            campaign_messages.append({
                'business': business,
                'message': message,
                'potential': potential,
                'priority': potential.get('priority', 5)
            })

    # Ordenar por prioridade
    campaign_messages.sort(key=lambda x: x['priority'], reverse=True)

    print(f"Campanha criada com {len(campaign_messages)} mensagens de alta prioridade")

    return campaign_messages

if __name__ == "__main__":
    campaign = create_smart_campaign("restaurante")

    # Exibir primeiras 5 mensagens
    for i, item in enumerate(campaign[:5]):
        print(f"\n{i+1}. {item['business'].name}")
        print(f"Mensagem: {item['message']}")
        print(f"Prioridade: {item['priority']}")
```

---

**Integração com Code LLM configurada! 🤖**

Agora você pode:
- ✅ Analisar dados de negócios automaticamente
- ✅ Gerar mensagens personalizadas
- ✅ Otimizar campanhas com IA
- ✅ Classificar potencial de conversão
- ✅ Automatizar follow-ups inteligentes
- ✅ Monitorar performance com métricas

**Próximos passos**: Guia de Uso (USAGE.md) | Troubleshooting (../README.md#troubleshooting)