Table of Contents

# 🔍 CÓRTEX BI - Guia Completo de Detecção Proativa de Anomalias

**Versão:** 2.0  
**Data:** Outubro 2025  
**Desenvolvido em parceria com:** Manus AI

## 📋 Índice

1. [Visão Geral](#visão-geral)
2. [Como Funciona](#como-funciona)
3. [Configuração Inicial](#configuração-inicial)
4. [Treinamento do Modelo](#treinamento-do-modelo)
5. [Ativação da Detecção Proativa](#ativação-da-detecção-proativa)
6. [Configuração de Alertas](#configuração-de-alertas)
7. [Monitoramento Contínuo](#monitoramento-contínuo)
8. [Casos de Uso Práticos](#casos-de-uso-práticos)
9. [Ajuste Fino e Otimização](#ajuste-fino-e-otimização)
10. [Troubleshooting](#troubleshooting)

## 🎯 Visão Geral

O sistema de **Detecção Proativa de Anomalias** do CÓRTEX BI utiliza algoritmos de Machine Learning (especificamente **Isolation Forest**) para identificar automaticamente padrões anormais em seus dados de negócio.

### O que o sistema detecta:

✅ **Quedas súbitas em métricas** - Ex: vendas caíram 15% sem motivo aparente  
✅ **Picos inesperados** - Ex: aumento anormal de custos  
✅ **Padrões atípicos** - Ex: comportamento de compra incomum  
✅ **Desvios de tendência** - Ex: KPI fora da faixa histórica esperada  
✅ **Outliers em dados** - Ex: valores extremos que fogem do padrão

### Benefícios:

🚀 **Detecção automática** - Não precisa monitorar manualmente  
⚡ **Alertas em tempo real** - Notificação imediata quando anomalia é detectada  
🎯 **Redução de riscos** - Identifica problemas antes que se tornem críticos  
📊 **Insights acionáveis** - Recomendações de ações corretivas  
🔄 **Aprendizado contínuo** - Sistema melhora com o tempo

## 🔧 Como Funciona

### Arquitetura do Sistema

O sistema de detecção de anomalias é composto por três componentes principais:

┌─────────────────────────────────────────────────────────┐  
│ CÓRTEX BI │  
├─────────────────────────────────────────────────────────┤  
│ │  
│ ┌──────────────┐ ┌──────────────┐ ┌───────────┐ │  
│ │ ML Engine │───▶│ Anomaly │──▶│ Alert │ │  
│ │ │ │ Detector │ │ System │ │  
│ └──────────────┘ └──────────────┘ └───────────┘ │  
│ │ │ │ │  
│ ▼ ▼ ▼ │  
│ ┌──────────────┐ ┌──────────────┐ ┌───────────┐ │  
│ │ Training │ │ Real-time │ │ Notifi- │ │  
│ │ Data │ │ Monitoring │ │ cation │ │  
│ └──────────────┘ └──────────────┘ └───────────┘ │  
│ │  
└─────────────────────────────────────────────────────────┘

### Algoritmo Isolation Forest

O CÓRTEX BI utiliza o algoritmo **Isolation Forest**, que funciona assim:

1. **Treinamento**: Aprende o comportamento “normal” dos seus dados históricos
2. **Isolamento**: Identifica pontos que são facilmente “isolados” (anômalos)
3. **Score**: Atribui um score de anomalia (-1 = anômalo, 1 = normal)
4. **Threshold**: Define limiar de sensibilidade para alertas

**Vantagens do Isolation Forest:** - ✅ Não requer dados rotulados (unsupervised learning) - ✅ Eficiente com grandes volumes de dados - ✅ Detecta anomalias multidimensionais - ✅ Baixo custo computacional

## ⚙️ Configuração Inicial

### Passo 1: Verificar Pré-requisitos

Certifique-se de que o CÓRTEX BI está instalado e rodando:

# Verificar status do sistema  
curl http://localhost:5000/health  
  
# Resposta esperada deve incluir:  
{  
 "services": {  
 "ml\_engine": "active",  
 ...  
 }  
}

### Passo 2: Configurar Variáveis de Ambiente

Edite o arquivo .env na raiz do projeto:

# ===== MACHINE LEARNING =====  
ML\_ENABLED=True  
ML\_MODEL\_PATH=data/models/  
  
# ===== DETECÇÃO DE ANOMALIAS =====  
ANOMALY\_DETECTION\_ENABLED=True  
ANOMALY\_CONTAMINATION=0.1 # 10% dos dados são considerados potenciais anomalias  
ANOMALY\_THRESHOLD=-0.5 # Score abaixo deste valor é considerado anomalia  
ANOMALY\_MIN\_SAMPLES=20 # Mínimo de amostras para treinar o modelo  
ANOMALY\_RETRAIN\_INTERVAL=86400 # Retreinar a cada 24 horas (em segundos)  
  
# ===== MONITORAMENTO PROATIVO =====  
PROACTIVE\_MONITORING\_ENABLED=True  
MONITORING\_INTERVAL=300 # Verificar a cada 5 minutos (em segundos)  
MONITORING\_METRICS=vendas,custos,receita,lucro # Métricas para monitorar  
  
# ===== ALERTAS =====  
ALERT\_ENABLED=True  
ALERT\_CHANNELS=email,teams,webhook # Canais de notificação  
ALERT\_SEVERITY\_THRESHOLD=medium # low, medium, high, critical  
  
# ===== EMAIL (para alertas) =====  
ALERT\_EMAIL\_TO=gerente@empresa.com,diretor@empresa.com  
ALERT\_EMAIL\_FROM=cortexbi-alerts@empresa.com  
SMTP\_HOST=smtp.office365.com  
SMTP\_PORT=587  
SMTP\_USER=cortexbi@empresa.com  
SMTP\_PASSWORD=sua-senha  
  
# ===== MICROSOFT TEAMS (para alertas) =====  
TEAMS\_WEBHOOK\_URL=https://outlook.office.com/webhook/seu-webhook-url  
  
# ===== WEBHOOK CUSTOMIZADO =====  
CUSTOM\_WEBHOOK\_URL=https://seu-sistema.com/api/alerts  
CUSTOM\_WEBHOOK\_AUTH\_TOKEN=seu-token-de-autenticacao

### Passo 3: Criar Estrutura de Diretórios

# Criar diretórios necessários  
mkdir -p data/models  
mkdir -p data/anomalies  
mkdir -p logs/anomalies  
mkdir -p config/anomaly\_rules  
  
# Configurar permissões  
chmod 755 data/models data/anomalies logs/anomalies config/anomaly\_rules

### Passo 4: Reiniciar o Sistema

# Parar o servidor  
./scripts/stop\_ai.sh # Linux/macOS  
# ou  
.\scripts\stop\_ai.bat # Windows  
  
# Iniciar o servidor  
./scripts/start\_ai.sh # Linux/macOS  
# ou  
.\scripts\start\_ai.bat # Windows

## 🎓 Treinamento do Modelo

### Método 1: Treinamento Automático via API

O método mais simples é usar a API para treinar o modelo:

# Treinar o detector de anomalias  
curl -X POST "http://localhost:5000/ml/train/anomaly\_detector" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{  
 "retrain": true,  
 "contamination": 0.1  
 }'

**Resposta esperada:**

{  
 "status": "success",  
 "model": "anomaly\_detector",  
 "metrics": {  
 "anomaly\_rate": 0.095,  
 "training\_samples": 1250,  
 "features": 5,  
 "trained\_at": "2025-10-14T18:30:00.000000"  
 },  
 "message": "Modelo treinado com sucesso"  
}

### Método 2: Treinamento via Python Script

Crie um script personalizado para treinar o modelo:

# train\_anomaly\_detector.py  
  
from src.agents.ml\_engine import MLEngine  
import json  
  
# Inicializar ML Engine  
ml\_engine = MLEngine(models\_dir="data/models")  
  
# Treinar detector de anomalias  
print("Iniciando treinamento do detector de anomalias...")  
metrics = ml\_engine.train\_anomaly\_detector(retrain=True)  
  
print("\nResultados do treinamento:")  
print(json.dumps(metrics, indent=2))  
  
if "error" not in metrics:  
 print("\n✅ Modelo treinado com sucesso!")  
 print(f"📊 Amostras de treinamento: {metrics['training\_samples']}")  
 print(f"🎯 Taxa de anomalia: {metrics['anomaly\_rate']:.2%}")  
else:  
 print(f"\n❌ Erro no treinamento: {metrics['error']}")

Execute o script:

python train\_anomaly\_detector.py

### Método 3: Treinamento Agendado (Automático)

Configure o treinamento automático periódico:

**Linux/macOS (usando cron):**

# Editar crontab  
crontab -e  
  
# Adicionar linha para retreinar diariamente às 2h da manhã  
0 2 \* \* \* cd /caminho/para/cortex-bi && python train\_anomaly\_detector.py >> logs/training.log 2>&1

**Windows (usando Task Scheduler):**

# Criar tarefa agendada  
$action = New-ScheduledTaskAction -Execute "python" -Argument "C:\caminho\para\cortex-bi\train\_anomaly\_detector.py"  
$trigger = New-ScheduledTaskTrigger -Daily -At 2am  
Register-ScheduledTask -Action $action -Trigger $trigger -TaskName "CORTEX\_BI\_Train\_Anomaly" -Description "Retreinar detector de anomalias diariamente"

### Verificar Status do Modelo

# Verificar se o modelo foi treinado  
curl -X GET "http://localhost:5000/ml/models/status" \  
 -H "X-API-Key: sua-api-key"

**Resposta:**

{  
 "models": {  
 "anomaly\_detector": {  
 "trained": true,  
 "last\_training": "2025-10-14T02:00:00.000000",  
 "samples": 1250,  
 "accuracy": 0.95  
 }  
 }  
}

## 🚀 Ativação da Detecção Proativa

### Configurar Métricas para Monitoramento

Crie um arquivo de configuração para definir quais métricas monitorar:

# Criar arquivo de configuração  
nano config/anomaly\_rules/metrics\_config.json

**Conteúdo do arquivo:**

{  
 "metrics": [  
 {  
 "name": "vendas\_totais",  
 "data\_source": "sql\_server",  
 "query": "SELECT SUM(valor\_venda) as total FROM vendas WHERE data = CAST(GETDATE() AS DATE)",  
 "threshold\_type": "percentage",  
 "threshold\_value": 15,  
 "severity": "high",  
 "check\_interval": 300,  
 "enabled": true  
 },  
 {  
 "name": "custos\_operacionais",  
 "data\_source": "csv",  
 "file\_path": "data/custos\_diarios.csv",  
 "column": "custo\_total",  
 "threshold\_type": "absolute",  
 "threshold\_value": 50000,  
 "severity": "medium",  
 "check\_interval": 600,  
 "enabled": true  
 },  
 {  
 "name": "taxa\_conversao",  
 "data\_source": "api",  
 "api\_endpoint": "https://api.empresa.com/metrics/conversao",  
 "threshold\_type": "percentage",  
 "threshold\_value": 10,  
 "severity": "high",  
 "check\_interval": 300,  
 "enabled": true  
 },  
 {  
 "name": "tempo\_resposta\_sistema",  
 "data\_source": "internal",  
 "metric\_key": "avg\_response\_time",  
 "threshold\_type": "absolute",  
 "threshold\_value": 5000,  
 "severity": "critical",  
 "check\_interval": 60,  
 "enabled": true  
 }  
 ],  
 "global\_settings": {  
 "min\_historical\_days": 30,  
 "confidence\_threshold": 0.8,  
 "alert\_cooldown": 3600  
 }  
}

### Iniciar Monitoramento Proativo

**Método 1: Via API**

# Iniciar monitoramento  
curl -X POST "http://localhost:5000/anomaly/monitoring/start" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{  
 "config\_file": "config/anomaly\_rules/metrics\_config.json",  
 "interval": 300  
 }'

**Método 2: Via Script Python**

Crie um script de monitoramento contínuo:

# start\_proactive\_monitoring.py  
  
import time  
import requests  
import json  
from datetime import datetime  
from src.agents.ml\_engine import MLEngine  
from src.agents.analytics\_engine import AnalyticsEngine  
  
class ProactiveAnomalyMonitor:  
 def \_\_init\_\_(self, config\_file: str):  
 self.ml\_engine = MLEngine()  
 self.analytics\_engine = AnalyticsEngine()  
   
 # Carregar configuração  
 with open(config\_file, 'r') as f:  
 self.config = json.load(f)  
   
 self.metrics = self.config['metrics']  
 self.global\_settings = self.config['global\_settings']  
   
 print(f"✅ Monitor inicializado com {len(self.metrics)} métricas")  
   
 def check\_metric(self, metric\_config: dict):  
 """Verifica uma métrica específica"""  
 try:  
 # Obter valor atual da métrica  
 current\_value = self.\_get\_metric\_value(metric\_config)  
   
 # Obter histórico  
 historical\_data = self.\_get\_historical\_data(metric\_config)  
   
 # Detectar anomalia usando ML Engine  
 result = self.ml\_engine.detect\_anomaly({  
 "metric\_name": metric\_config['name'],  
 "current\_value": current\_value,  
 "historical\_data": historical\_data,  
 "execution\_time": 0,  
 "request\_data": json.dumps(metric\_config),  
 "response\_data": json.dumps({"value": current\_value}),  
 "user\_id": "system"  
 })  
   
 if result.get("is\_anomaly"):  
 self.\_trigger\_alert(metric\_config, current\_value, result)  
   
 return result  
   
 except Exception as e:  
 print(f"❌ Erro ao verificar métrica {metric\_config['name']}: {e}")  
 return {"error": str(e)}  
   
 def \_get\_metric\_value(self, metric\_config: dict):  
 """Obtém valor atual da métrica"""  
 data\_source = metric\_config['data\_source']  
   
 if data\_source == "sql\_server":  
 # Executar query SQL  
 from src.agents.data\_loader import DataLoader  
 loader = DataLoader()  
 result = loader.query\_sql\_server(metric\_config['query'])  
 return result[0]['total'] if result else 0  
   
 elif data\_source == "csv":  
 # Ler arquivo CSV  
 import pandas as pd  
 df = pd.read\_csv(metric\_config['file\_path'])  
 return df[metric\_config['column']].iloc[-1]  
   
 elif data\_source == "api":  
 # Chamar API externa  
 response = requests.get(metric\_config['api\_endpoint'])  
 return response.json().get('value', 0)  
   
 elif data\_source == "internal":  
 # Métrica interna do sistema  
 return self.\_get\_internal\_metric(metric\_config['metric\_key'])  
   
 return 0  
   
 def \_get\_historical\_data(self, metric\_config: dict, days: int = 30):  
 """Obtém dados históricos da métrica"""  
 # Implementar lógica para obter histórico  
 # Por enquanto, retorna lista vazia  
 return []  
   
 def \_get\_internal\_metric(self, metric\_key: str):  
 """Obtém métrica interna do sistema"""  
 # Implementar lógica para métricas internas  
 return 0  
   
 def \_trigger\_alert(self, metric\_config: dict, current\_value: float, anomaly\_result: dict):  
 """Dispara alerta de anomalia"""  
 alert\_data = {  
 "timestamp": datetime.now().isoformat(),  
 "metric\_name": metric\_config['name'],  
 "current\_value": current\_value,  
 "severity": metric\_config['severity'],  
 "anomaly\_score": anomaly\_result.get('anomaly\_score'),  
 "confidence": anomaly\_result.get('confidence'),  
 "message": f"Anomalia detectada em {metric\_config['name']}: valor atual {current\_value}"  
 }  
   
 print(f"\n🚨 ALERTA DE ANOMALIA:")  
 print(json.dumps(alert\_data, indent=2))  
   
 # Enviar alertas  
 self.\_send\_email\_alert(alert\_data)  
 self.\_send\_teams\_alert(alert\_data)  
 self.\_send\_webhook\_alert(alert\_data)  
   
 def \_send\_email\_alert(self, alert\_data: dict):  
 """Envia alerta por email"""  
 # Implementar envio de email  
 pass  
   
 def \_send\_teams\_alert(self, alert\_data: dict):  
 """Envia alerta para Microsoft Teams"""  
 # Implementar envio para Teams  
 pass  
   
 def \_send\_webhook\_alert(self, alert\_data: dict):  
 """Envia alerta para webhook customizado"""  
 # Implementar envio para webhook  
 pass  
   
 def run(self):  
 """Executa monitoramento contínuo"""  
 print("\n🚀 Iniciando monitoramento proativo de anomalias...")  
 print(f"📊 Monitorando {len(self.metrics)} métricas")  
   
 while True:  
 for metric in self.metrics:  
 if not metric.get('enabled', True):  
 continue  
   
 print(f"\n🔍 Verificando: {metric['name']}")  
 result = self.check\_metric(metric)  
   
 if result.get("is\_anomaly"):  
 print(f"⚠️ Anomalia detectada!")  
 else:  
 print(f"✅ Normal")  
   
 # Aguardar intervalo específico da métrica  
 time.sleep(metric.get('check\_interval', 300))  
  
# Executar monitor  
if \_\_name\_\_ == "\_\_main\_\_":  
 monitor = ProactiveAnomalyMonitor("config/anomaly\_rules/metrics\_config.json")  
 monitor.run()

Execute o script:

# Executar em background  
nohup python start\_proactive\_monitoring.py > logs/anomalies/monitor.log 2>&1 &  
  
# Ver logs em tempo real  
tail -f logs/anomalies/monitor.log

## 📧 Configuração de Alertas

### Alertas por Email

Configure o envio de emails no arquivo .env:

ALERT\_EMAIL\_ENABLED=True  
ALERT\_EMAIL\_TO=gerente@empresa.com,diretor@empresa.com  
ALERT\_EMAIL\_CC=ti@empresa.com  
ALERT\_EMAIL\_FROM=cortexbi-alerts@empresa.com  
ALERT\_EMAIL\_SUBJECT\_PREFIX=[CÓRTEX BI - ALERTA]  
  
SMTP\_HOST=smtp.office365.com  
SMTP\_PORT=587  
SMTP\_USE\_TLS=True  
SMTP\_USER=cortexbi@empresa.com  
SMTP\_PASSWORD=sua-senha

**Template de email personalizado:**

Crie um arquivo config/anomaly\_rules/email\_template.html:

<!DOCTYPE html>  
<html>  
<head>  
 <style>  
 body { font-family: Arial, sans-serif; }  
 .alert-box {   
 border: 2px solid #dc3545;   
 padding: 20px;   
 margin: 20px 0;  
 background-color: #f8d7da;  
 }  
 .severity-high { border-color: #dc3545; background-color: #f8d7da; }  
 .severity-medium { border-color: #ffc107; background-color: #fff3cd; }  
 .severity-low { border-color: #17a2b8; background-color: #d1ecf1; }  
 .metric-value { font-size: 24px; font-weight: bold; color: #dc3545; }  
 </style>  
</head>  
<body>  
 <h2>🚨 Alerta de Anomalia Detectada</h2>  
   
 <div class="alert-box severity-{{severity}}">  
 <h3>{{metric\_name}}</h3>  
 <p><strong>Valor Atual:</strong> <span class="metric-value">{{current\_value}}</span></p>  
 <p><strong>Severidade:</strong> {{severity}}</p>  
 <p><strong>Confiança:</strong> {{confidence}}%</p>  
 <p><strong>Data/Hora:</strong> {{timestamp}}</p>  
 </div>  
   
 <h3>📊 Detalhes da Anomalia</h3>  
 <ul>  
 <li><strong>Score de Anomalia:</strong> {{anomaly\_score}}</li>  
 <li><strong>Desvio do Padrão:</strong> {{deviation}}%</li>  
 <li><strong>Valor Esperado:</strong> {{expected\_value}}</li>  
 </ul>  
   
 <h3>💡 Recomendações</h3>  
 <ul>  
 {{#each recommendations}}  
 <li>{{this}}</li>  
 {{/each}}  
 </ul>  
   
 <hr>  
 <p><small>Este é um alerta automático do CÓRTEX BI. Para mais informações, acesse o dashboard administrativo.</small></p>  
</body>  
</html>

### Alertas no Microsoft Teams

Configure o webhook do Teams:

# No arquivo .env  
ALERT\_TEAMS\_ENABLED=True  
TEAMS\_WEBHOOK\_URL=https://outlook.office.com/webhook/seu-webhook-url

**Criar webhook no Teams:**

1. Abra o canal do Teams onde deseja receber alertas
2. Clique nos três pontos (⋯) → “Connectors”
3. Procure por “Incoming Webhook” e clique em “Configure”
4. Dê um nome (ex: “CÓRTEX BI Alertas”) e clique em “Create”
5. Copie a URL do webhook gerada
6. Cole no arquivo .env

**Template de mensagem Teams:**

{  
 "@type": "MessageCard",  
 "@context": "https://schema.org/extensions",  
 "summary": "Anomalia Detectada",  
 "themeColor": "dc3545",  
 "title": "🚨 CÓRTEX BI - Alerta de Anomalia",  
 "sections": [  
 {  
 "activityTitle": "{{metric\_name}}",  
 "activitySubtitle": "Severidade: {{severity}}",  
 "facts": [  
 {  
 "name": "Valor Atual:",  
 "value": "{{current\_value}}"  
 },  
 {  
 "name": "Confiança:",  
 "value": "{{confidence}}%"  
 },  
 {  
 "name": "Data/Hora:",  
 "value": "{{timestamp}}"  
 }  
 ]  
 }  
 ],  
 "potentialAction": [  
 {  
 "@type": "OpenUri",  
 "name": "Ver Dashboard",  
 "targets": [  
 {  
 "os": "default",  
 "uri": "http://localhost:5000/admin/admin\_dashboard.html"  
 }  
 ]  
 }  
 ]  
}

### Alertas via Webhook Customizado

Para integrar com sistemas próprios:

# No arquivo .env  
ALERT\_WEBHOOK\_ENABLED=True  
CUSTOM\_WEBHOOK\_URL=https://seu-sistema.com/api/alerts  
CUSTOM\_WEBHOOK\_METHOD=POST  
CUSTOM\_WEBHOOK\_AUTH\_TYPE=bearer # ou 'basic', 'api\_key'  
CUSTOM\_WEBHOOK\_AUTH\_TOKEN=seu-token

**Payload enviado:**

{  
 "source": "cortex\_bi",  
 "alert\_type": "anomaly\_detection",  
 "timestamp": "2025-10-14T18:30:00.000000",  
 "severity": "high",  
 "metric": {  
 "name": "vendas\_totais",  
 "current\_value": 85000,  
 "expected\_value": 100000,  
 "deviation\_percentage": -15  
 },  
 "anomaly": {  
 "score": -0.75,  
 "confidence": 0.92,  
 "is\_anomaly": true  
 },  
 "recommendations": [  
 "Verificar campanhas de marketing ativas",  
 "Analisar concorrência",  
 "Revisar estratégia de preços"  
 ]  
}

## 📊 Monitoramento Contínuo

### Dashboard de Anomalias

Acesse o dashboard específico de anomalias:

http://localhost:5000/admin/anomalies\_dashboard.html

**Funcionalidades do Dashboard:**

* 📈 Gráfico de anomalias detectadas ao longo do tempo
* 🎯 Métricas mais afetadas
* 📊 Distribuição de severidade
* 🔔 Histórico de alertas
* ⚙️ Configuração de regras
* 📥 Exportar relatórios

### API de Monitoramento

**Listar anomalias detectadas:**

curl -X GET "http://localhost:5000/anomaly/list?days=7" \  
 -H "X-API-Key: sua-api-key"

**Verificar métrica específica:**

curl -X POST "http://localhost:5000/anomaly/check" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{  
 "metric\_name": "vendas\_totais",  
 "current\_value": 85000,  
 "historical\_data": [100000, 98000, 102000, 99000, 101000]  
 }'

**Status do monitoramento:**

curl -X GET "http://localhost:5000/anomaly/monitoring/status" \  
 -H "X-API-Key: sua-api-key"

## 💼 Casos de Uso Práticos

### Caso 1: Monitorar Vendas Diárias

**Objetivo:** Detectar quedas inesperadas nas vendas

**Configuração:**

{  
 "name": "vendas\_diarias",  
 "data\_source": "sql\_server",  
 "query": "SELECT SUM(valor\_venda) as total FROM vendas WHERE data = CAST(GETDATE() AS DATE)",  
 "threshold\_type": "percentage",  
 "threshold\_value": 10,  
 "severity": "high",  
 "check\_interval": 3600,  
 "enabled": true,  
 "alert\_channels": ["email", "teams"],  
 "recipients": ["gerente.vendas@empresa.com"]  
}

**Resultado:** - Sistema verifica vendas a cada hora - Se vendas caírem mais de 10%, dispara alerta - Gerente de vendas recebe email e notificação no Teams

### Caso 2: Monitorar Custos Operacionais

**Objetivo:** Identificar picos anormais de custos

**Configuração:**

{  
 "name": "custos\_operacionais",  
 "data\_source": "csv",  
 "file\_path": "data/custos\_diarios.csv",  
 "column": "custo\_total",  
 "threshold\_type": "absolute",  
 "threshold\_value": 50000,  
 "severity": "medium",  
 "check\_interval": 7200,  
 "enabled": true,  
 "alert\_channels": ["email"],  
 "recipients": ["controller@empresa.com"]  
}

**Resultado:** - Sistema verifica custos a cada 2 horas - Se custos ultrapassarem R$ 50.000, dispara alerta - Controller recebe email com detalhes

### Caso 3: Monitorar Performance do Sistema

**Objetivo:** Detectar degradação de performance

**Configuração:**

{  
 "name": "tempo\_resposta\_api",  
 "data\_source": "internal",  
 "metric\_key": "avg\_response\_time",  
 "threshold\_type": "absolute",  
 "threshold\_value": 5000,  
 "severity": "critical",  
 "check\_interval": 60,  
 "enabled": true,  
 "alert\_channels": ["teams", "webhook"],  
 "recipients": ["ti@empresa.com"]  
}

**Resultado:** - Sistema verifica performance a cada minuto - Se tempo de resposta > 5 segundos, dispara alerta crítico - TI recebe notificação imediata no Teams

## 🎛️ Ajuste Fino e Otimização

### Ajustar Sensibilidade

A sensibilidade do detector pode ser ajustada através do parâmetro contamination:

# Mais sensível (detecta mais anomalias)  
contamination = 0.05 # 5% dos dados são anomalias  
  
# Sensibilidade padrão  
contamination = 0.1 # 10% dos dados são anomalias  
  
# Menos sensível (detecta apenas anomalias muito evidentes)  
contamination = 0.2 # 20% dos dados são anomalias

**Via API:**

curl -X POST "http://localhost:5000/ml/train/anomaly\_detector" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{  
 "retrain": true,  
 "contamination": 0.05  
 }'

### Ajustar Threshold de Alerta

# No arquivo .env  
ANOMALY\_THRESHOLD=-0.5 # Padrão  
ANOMALY\_THRESHOLD=-0.3 # Mais sensível (mais alertas)  
ANOMALY\_THRESHOLD=-0.7 # Menos sensível (menos alertas)

### Filtrar Falsos Positivos

Crie regras para ignorar falsos positivos conhecidos:

{  
 "false\_positive\_rules": [  
 {  
 "metric\_name": "vendas\_totais",  
 "condition": "day\_of\_week == 0",  
 "reason": "Domingos sempre têm vendas baixas"  
 },  
 {  
 "metric\_name": "acessos\_sistema",  
 "condition": "hour >= 22 or hour <= 6",  
 "reason": "Baixo acesso durante madrugada é normal"  
 }  
 ]  
}

### Retreinamento Automático

Configure retreinamento periódico para manter o modelo atualizado:

# No arquivo .env  
ANOMALY\_AUTO\_RETRAIN=True  
ANOMALY\_RETRAIN\_INTERVAL=86400 # 24 horas  
ANOMALY\_RETRAIN\_MIN\_NEW\_SAMPLES=100 # Mínimo de novas amostras para retreinar

## 🔍 Troubleshooting

### Problema: Modelo não detecta anomalias óbvias

**Possíveis causas:** - Threshold muito alto - Contamination muito alta - Dados de treinamento insuficientes

**Soluções:**

# 1. Ajustar threshold  
ANOMALY\_THRESHOLD=-0.3 # Mais sensível  
  
# 2. Ajustar contamination  
curl -X POST "http://localhost:5000/ml/train/anomaly\_detector" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{"retrain": true, "contamination": 0.05}'  
  
# 3. Verificar quantidade de dados de treinamento  
curl -X GET "http://localhost:5000/ml/models/status" \  
 -H "X-API-Key: sua-api-key"

### Problema: Muitos falsos positivos

**Possíveis causas:** - Threshold muito baixo - Contamination muito baixa - Dados de treinamento não representativos

**Soluções:**

# 1. Ajustar threshold  
ANOMALY\_THRESHOLD=-0.7 # Menos sensível  
  
# 2. Ajustar contamination  
curl -X POST "http://localhost:5000/ml/train/anomaly\_detector" \  
 -H "Content-Type: application/json" \  
 -H "X-API-Key: sua-api-key" \  
 -d '{"retrain": true, "contamination": 0.15}'  
  
# 3. Adicionar regras de falsos positivos  
# Editar config/anomaly\_rules/false\_positive\_rules.json

### Problema: Alertas não estão sendo enviados

**Verificações:**

# 1. Verificar se alertas estão habilitados  
grep ALERT\_ENABLED .env  
  
# 2. Verificar logs de alertas  
tail -f logs/anomalies/alerts.log  
  
# 3. Testar envio de email  
curl -X POST "http://localhost:5000/admin/test/email" \  
 -H "X-API-Key: sua-api-key"  
  
# 4. Testar webhook do Teams  
curl -X POST "http://localhost:5000/admin/test/teams" \  
 -H "X-API-Key: sua-api-key"

### Problema: Alto uso de CPU/memória

**Soluções:**

# 1. Aumentar intervalo de verificação  
MONITORING\_INTERVAL=600 # 10 minutos em vez de 5  
  
# 2. Reduzir número de métricas monitoradas  
# Editar config/anomaly\_rules/metrics\_config.json  
# Desabilitar métricas menos críticas  
  
# 3. Limitar histórico de dados  
ANOMALY\_HISTORICAL\_DAYS=30 # Em vez de 90

## 📞 Suporte e Recursos

### Documentação Adicional

* **README Principal**: /docs/README.md
* **API Reference**: http://localhost:5000/docs
* **Guia de Instalação**: /docs/guia\_instalacao.md

### Scripts Úteis

* train\_anomaly\_detector.py - Treinar modelo
* start\_proactive\_monitoring.py - Iniciar monitoramento
* test\_anomaly\_detection.py - Testar detecção
* export\_anomalies\_report.py - Exportar relatório

### Contato

* **GitHub Issues**: https://github.com/Rimkus85/cortex-bi/issues
* **Email**: suporte@cortexbi.com

**CÓRTEX BI v2.0** - *Cognitive Operations & Real-Time EXpert Business Intelligence*  
Desenvolvido em parceria com **Manus AI** | Outubro 2025

🚨 **Detecção Proativa de Anomalias**  
Identifique problemas antes que se tornem críticos!