**UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO**

Cintia Izumi Shinoda
Cristiano Gois
Fernando Miguel Escribano Martinez
Jordana Barcala
Juliana de Almeida Gonçalves
Pedro Henrique Faria Cruz
Rogério Gonçalves da Silva
Willy Paulino de Oliveira Gomes

# Análise da base de dados IoT-23: Origem, aplicações, aprendizado de máquina e plataforma de visualização

**Vídeo de apresentação do Projeto Integrador**

https://www.youtube.com/watch?v=fRZynukI0sU

**Repositório**

https://github.com/jobarcala/Projeto-Integrador-IV

São Paulo - SP

2025

**UNIVERSIDADE VIRTUAL DO ESTADO DE SÃO PAULO**

# Análise da base de dados IoT-23: Origem, aplicações, aprendizado de máquina e plataforma de visualização

Relatório Técnico-Científico apresentado na disciplina de Projeto Integrador para o curso de Bacharelado em Engenharia de Computação, Bacharelado em Ciência de Dados e Bacharelado em Tecnologia da Informação da Universidade Virtual do Estado de São Paulo (UNIVESP) como requisito para aprovação.

Orientadora: Iolanda Alves Roque da Fonseca

São Paulo - SP
2025

SHINODA, Cintia Izumi; GOIS, Cristiano; MARTINEZ, Fernando Miguel Escribano; BARCALA, Jordana; GONCALVES, Juliana de Almeida; CRUZ, Pedro Henrique Faria; SILVA, Rogério Gonçalves da; GOMES, Willy Paulino de Oliveira. **Análise da base de dados IoT-23: Origem, aplicações, aprendizado de máquina e plataforma de visualização.** Relatório Técnico-Científico. Engenharia da Computação, Ciência de Dados e Bacharelado em Tecnologia da Informação – **Universidade Virtual do Estado de São Paulo**. Tutor: Iolanda Alves Roque da Fonseca. Polo Jardim Paulistano, Polo Vila Rubi, Polo Casa Branca, Polo Jaçanã, Polo Sapopemba, Polo Carrão, 2025.

## RESUMO

O crescimento da Internet das Coisas (IoT) tem ampliado as oportunidades de inovação, mas também intensificado os riscos de segurança cibernética em dispositivos conectados. Este trabalho analisa a base de dados IoT-23, desenvolvida pela Czech Technical University, composta por cenários de tráfego benigno e malicioso, com o objetivo de aplicar técnicas de aprendizado de máquina para detecção de anomalias e ataques em redes IoT. O projeto envolveu a consolidação e higienização de mais de 740 mil registros, a realização de análises exploratórias (quantificação de rótulos, distribuição de ataques, estatísticas descritivas, protocolos e padrões temporais) e a implementação de dois modelos supervisionados: *Random Forest* e *Gradient Boosting*. Ambos apresentaram resultados robustos, com acurácia superior a 98%, evidenciando a viabilidade do uso de tais algoritmos para classificação de tráfego. Além disso, foi desenvolvida uma plataforma de visualização feita em HTML/CSS interativa, baseada em Flask e tecnologias de front-end, permitindo a visualização dinâmica das análises e a classificação em tempo real de novas conexões. Os resultados demonstram que a integração entre ciência de dados, aprendizado de máquina e interfaces de visualização pode fortalecer a segurança digital em ambientes de IoT, oferecendo subsídios para sistemas de detecção mais precisos e aplicáveis em cenários reais.

Palavras-chave: Internet das coisas, Segurança cibernética, Aprendizado de máquina, Visualização de dados.

**SUMÁRIO**

## 1. INTRODUÇÃO

Com o crescimento exponencial da Internet das Coisas (IoT), a segurança de dispositivos conectados tornou-se uma preocupação central em ambientes residenciais, corporativos e industriais. A diversidade de dispositivos — como câmeras, sensores, eletrodomésticos inteligentes e controladores — amplificou as vulnerabilidades da infraestrutura digital, exigindo novas abordagens para análise e detecção de ameaças. Nesse contexto, o IoT-23 Dataset surge como uma importante fonte de dados reais para estudo e desenvolvimento de soluções baseadas em aprendizado de máquina voltadas à segurança de redes IoT.

O IoT-23 Dataset foi desenvolvido pelo grupo de pesquisa Stratosphere Lab da Czech Technical University in Prague (disponível em: https://www.stratosphereips.org/datasets-iot23?), sendo disponibilizado publicamente como parte de um esforço para fomentar o desenvolvimento de sistemas de detecção de anomalias em redes IoT. O conjunto de dados foi publicado em 2020 e é amplamente citado em pesquisas acadêmicas por conter tráfego real de dispositivos IoT infectados e não infectados, com o diferencial de apresentar os dados em formato PCAP[1] (captura de pacotes de rede) e CSV[2] com metadados[3] e rotulagem para facilitar o uso por pesquisadores e desenvolvedores (GARCIA et al., 2020, p. 2–3).

O IoT-23 é composto por 23 arquivos de captura de tráfego de rede (scenarios), cada um representando diferentes situações de tráfego:

- 14 cenários com dispositivos infectados (ex.: infecção por *malware* como Mirai, Gafgyt, Tsunami, Okiru);
- 9 cenários com tráfego legítimo (benigno) gerado por dispositivos como câmeras IP, tomadas inteligentes, sensores etc.

---

[1] Arquivo com extensão .pcap (*Packet Capture*) é um formato padrão utilizado para armazenar capturas de tráfego de rede, contendo pacotes de dados transmitidos entre dispositivos durante um intervalo de tempo. Esses pacotes incluem informações como endereços IP de origem e destino, portas, protocolos, *timestamps* e dados do conteúdo transmitido. Arquivos .pcap são gerados por ferramentas como Wireshark e tcpdump, sendo amplamente utilizados em análises de rede, segurança cibernética e treinamentos com aprendizado de máquina.

[2] CSV, Comma-Separated Values (Valores Separados por Vírgula), um formato de arquivo de texto simples utilizado para armazenar dados estruturados em forma de tabela. Cada linha do arquivo representa um registro e cada valor é separado por um delimitador — geralmente a vírgula, mas podendo ser ponto e vírgula ou outro caractere.

[3] Apenas na versão completa do pacote.

Cada cenário possui:

- Arquivos .pcap com o tráfego de rede capturado.
- Arquivos .csv com meta-informações, incluindo marcação de cada fluxo como Malicioso ou Benigno, além de campos como protocolo, duração, número de pacotes, bytes transmitidos, portas utilizadas, entre outros.

Essa estrutura permite que o *dataset* seja usado tanto para análises brutas de tráfego quanto para a aplicação de técnicas automatizadas com aprendizado de máquina supervisionado.

A principal finalidade do IoT-23 é servir como base para o desenvolvimento e a avaliação de sistemas de detecção de anomalias ou intrusões em redes IoT. Ele pode ser usado para:

- Treinar e testar modelos de detecção de malware;
- Classificação de tráfego (normal x malicioso);
- Estudos de comportamento de *botnets*; e
- Análise de protocolos utilizados por dispositivos IoT em redes domésticas.

Além disso, a granularidade dos dados permite aplicações em sistemas de segurança cibernética baseados em Inteligência Artificial (IA), *firewalls*[4] inteligentes e plataformas de análise forense (GARCIA et al., 2020, p. 2).

[4] *Firewall* é um sistema de segurança de rede, implementado por *hardware, software* ou ambos, que monitora e controla o tráfego de dados com base em regras predefinidas. Sua função é permitir ou bloquear comunicações entre redes — como a interna e a *Internet* — para proteger sistemas contra acessos não autorizados, ataques ou tráfego malicioso, funcionando como uma barreira entre ambientes confiáveis e não confiáveis.

## 2. DESCRIÇÃO E TRATAMENTO DA BASE DE DADOS

A universidade de Praga dá a opção de *download* de dois arquivos: a base de dados completa e a "*lighter version*" com pouco mais de 47Gb de dados compactados. Todos os dados estão no formato *.labeled, que nada mais é do que o arquivo.pcap, mas com outra extensão para se diferenciar do PCAP original[5].

Cada arquivo .labeled contém 21 campos com as seguintes descrições[6]:

Significado dos principais campos

| Campo | Descrição |
|---|---|
| ts | Timestamp do início da conexão (Unix epoch). |
| uid | Identificador único da conexão. |
| id.orig_h | IP de origem. |
| id.orig_p | Porta de origem. |
| id.resp_h | IP de destino. |
| id.resp_p | Porta de destino. |
| proto | Protocolo utilizado (TCP, UDP, ICMP, etc.). |
| service | Serviço identificado (HTTP, DNS, SSH, etc.). |
| duration | Duração da conexão em segundos. |
| orig_bytes | *Bytes* enviados pelo host de origem. |
| resp_bytes | *Bytes* recebidos pelo host de origem. |
| conn_state | Estado da conexão (por exemplo, S0, SF, REJ). |
| orig_pkts | Número de pacotes enviados pelo host de origem. |
| resp_pkts | Número de pacotes enviados pelo host de destino. |
| label | Rótulo binário: Benigno ou Malicioso. |
| detailed-label | Categoria específica de ataque (ex.: Mirai-UDP, DDoS, etc.). |

As informações que podem ser obtidas com esse tipo de dado são:

A base de dados IoT-23 oferece um rico conjunto de informações extraídas de tráfego de rede gerado por dispositivos IoT em cenários diversos, incluindo situações

[5] O PCAP original está na opção "*full version*" do pacote IoT23.

[6] O código utilizado para a extração dessa informação está no Anexo 1.

com comportamento legítimo e simulações de ataques reais. Cada entrada nos arquivos .labeled representa uma conexão de rede registrada pelo monitoramento do Zeek (antigo Bro), contendo informações como *timestamp* (ts), IP de origem e destino, protocolo utilizado (proto), número de pacotes trocados (orig_pkts e resp_pkts), volume de dados transferidos (orig_bytes e resp_bytes), entre outros. Além desses dados técnicos, os arquivos contêm rótulos (label e detailed-label) que indicam se a conexão é considerada Benigna ou maliciosa e, em caso de ameaça, especificam o tipo de ataque, como Mirai[7], Gafgyt[8], DDoS[9], entre outros (Garcia et al., 2020, p. 4–7).

Com base nesses campos é possível realizar diferentes análises de rede com múltiplas finalidades. Uma das principais é a detecção de atividades maliciosas, por meio da classificação binária (Benigno ou malicioso) e da identificação do tipo específico de ataque. Isso permite, por exemplo, identificar padrões comuns em tráfego malicioso, como o uso repetido de portas específicas, protocolos predominantes (como UDP[10] em ataques de negação de serviço) e baixa duração de conexão acompanhada de grande número de pacotes, caracterizando comportamentos típicos de *botnets*.

Além disso, a base permite realizar uma análise descritiva do tráfego de rede, com o objetivo de mensurar a quantidade de conexões por IP[11], o volume de dados

---

[7] Mirai é um tipo de malware que infecta dispositivos conectados à *Internet* — especialmente equipamentos de Internet das Coisas (IoT), como câmeras IP e roteadores — explorando senhas fracas ou configurações padrão.

[8] Gafgyt é um tipo de *malware* voltado para infectar dispositivos de *Internet* das Coisas (IoT), como roteadores, câmeras IP e gravadores de vídeo digital (DVRs). Assim como o Mirai, ele explora credenciais fracas ou configurações padrão para assumir o controle do equipamento e integrá-lo a uma *botnet*.

[9] DDoS é a sigla para *Distributed Denial of Service* (Negação de Serviço Distribuída), um tipo de ataque cibernético que busca tornar um serviço, servidor ou rede indisponível, sobrecarregando-o com um grande volume de requisições simultâneas provenientes de múltiplos dispositivos comprometidos.

[10] UDP, *User Datagram Protocol* (Protocolo de Datagrama do Usuário), é um protocolo de comunicação da camada de transporte que permite a transmissão rápida de dados entre dispositivos em uma rede. Ele é considerado “não orientado à conexão” e “não confiável”, pois não estabelece uma sessão prévia nem realiza verificação de entrega ou ordem dos pacotes, sendo usado em aplicações que priorizam velocidade e baixa latência, como transmissões de áudio e vídeo em tempo real ou jogos *online*.

[11] IP, *Internet Protocol* (Protocolo de Internet), um conjunto de regras que define como os dados são endereçados e transmitidos entre dispositivos conectados a redes, como a *Internet*. Cada dispositivo possui um endereço IP único, que funciona como um identificador numérico, permitindo sua localização e comunicação dentro da rede.

transmitidos, o tempo médio das sessões e os serviços mais acessados (HTTP[12], DNS[13], SSH[14], etc.). Essas informações são fundamentais para entender o comportamento dos dispositivos na rede e também para comparar o tráfego legítimo ao malicioso. Como destacam LAUDON e LAUDON (2020, p. 215), o uso de sistemas informatizados para monitoramento de dados e geração de relatórios analíticos oferece às organizações a capacidade de otimizar processos e aumentar sua segurança operacional.

No aspecto temporal, a base IoT-23 também permite a realização de análises cronológicas, como a frequência de ataques ao longo do tempo, os períodos do dia com maior atividade suspeita ou a detecção de eventos coordenados. Essas análises podem ser associadas ao campo ts (*timestamp*), e são especialmente úteis para identificar padrões de comportamento automatizado, como os realizados por *botnets*[15] ou *scanners*[16].

Outra aplicação relevante é a construção de perfis comportamentais, tanto de tráfego benigno quanto de tráfego malicioso. Isso pode ser feito ao analisar o conjunto de atributos disponíveis para cada tipo de rótulo e identificar comportamentos típicos de dispositivos infectados. Essa abordagem é particularmente útil para a criação de modelos de detecção de anomalias que possam ser aplicados em ambientes reais, onde nem sempre há rótulos explícitos para o tráfego analisado.

Antes de começarmos a de fato trabalhar com os dados precisamos antes fazer a normalização deles, ou seja, verificar se há dados faltando e o que fazer

---

[12] HTTP, *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto), um protocolo de comunicação da camada de aplicação usado para transferir dados entre clientes e servidores na *World Wide Web*.

[13] DNS, *Domain Name System* (Sistema de Nomes de Domínio), um serviço essencial da Internet que traduz nomes de domínio legíveis por humanos (como www.exemplo.com) em endereços IP numéricos utilizados pelos computadores para localizar e se comunicar com servidores.

[14] SSH, *Secure Shell*, um protocolo de comunicação criptografado utilizado para estabelecer conexões seguras entre dois dispositivos em rede, geralmente para administração remota de servidores.

[15] *Botnet* é uma rede formada por dispositivos comprometidos por malware — como computadores, servidores ou equipamentos de *Internet* das Coisas (IoT) — que são controlados remotamente por um invasor.

[16] *Scanner*, no contexto de conexões e tráfego de *Internet*, é uma ferramenta ou programa utilizado para identificar e mapear dispositivos, serviços ou portas abertas em uma rede. Ele envia solicitações sistemáticas para diferentes endereços IP e portas, registrando as respostas para detectar vulnerabilidades, serviços ativos ou topologias de rede.

nessas situações; verificar se todos os dados são numéricos; se estão todos parametrizados da mesma forma etc.[17]

A análise inicial dos arquivos .labeled evidencia a necessidade crítica de pré-processamento antes da realização de qualquer atividade analítica ou modelagem preditiva. A estrutura desses arquivos segue uma convenção própria, na qual os nomes das colunas são definidos em uma linha iniciada por #fields, o que difere do formato CSV convencional. No entanto, ao realizar uma leitura direta sem considerar essa particularidade, os dados são interpretados incorretamente, levando a problemas como a perda dos nomes das colunas, a fusão indevida de campos múltiplos em uma única coluna e a detecção equivocada dos tipos de dados (por exemplo, valores numéricos interpretados como strings).

Essas distorções comprometem a análise estatística e inviabilizam a aplicação direta de algoritmos de aprendizado de máquina, que dependem da correta identificação das variáveis, de suas escalas e tipos. Como observam HAN, PEI e KAMBER (2022, p. 77), a etapa de pré-processamento de dados é essencial para remover ruídos, lidar com dados incompletos e inconsistentes, e garantir a confiabilidade dos resultados produzidos por modelos computacionais. A leitura apropriada desses arquivos requer, portanto, a implementação de um *parser* especializado, capaz de reconhecer dinamicamente a linha de definição dos campos (#fields) e utilizá-la como cabeçalho efetivo, assegurando a correta separação por tabulação e o mapeamento preciso dos dados.

Além disso, a ausência de separação adequada dos campos de rótulo (*label* e *detailed*-label) compromete sua utilidade como variável-alvo em tarefas de classificação supervisionada. De acordo com CIOS et al. (2007, p. 34), a integridade semântica dos rótulos é um requisito básico para qualquer análise baseada em aprendizado supervisionado. Portanto, o tratamento inicial dos dados da IoT-23 deve contemplar, além da redefinição do cabeçalho, a conversão de colunas para os tipos apropriados (inteiros, ponto flutuante, categóricos), a normalização de valores quando necessário, e a identificação de valores ausentes ou inconsistentes.

[17] O código utilizado para a verificação da necessidade de normalização está no Anexo 2.

Para tornar o trabalho mais fácil de ser acompanhado, optou-se por fazer a junção dos 23 arquivos .labeled em um único arquivo CSV limitado a 50.000 linhas[18] por arquivo dado o limite de hardware do computador, assim podemos fazer as modificações e acompanhar o que está sendo feito com o próprio Python ou RStudio. A consolidação dos 23 arquivos de captura da base IoT-23 em um único conjunto de dados representa uma etapa estratégica no fluxo de análise. Essa unificação permite aplicar técnicas de pré-processamento, modelagem preditiva e visualização de forma integrada, reduzindo redundâncias operacionais e maximizando a consistência do tratamento de dados. Conforme observam HAN, PEI e KAMBER (2022, p. 94), conjuntos amplos e bem organizados favorecem a extração de padrões mais robustos, além de facilitar a avaliação cruzada entre diferentes contextos. É importante, contudo, manter o controle da procedência de cada registro por meio de uma coluna que identifique o cenário original, garantindo rastreabilidade e integridade analítica. O arquivo final possui 746532 linhas, volume o suficiente para fazermos análise e aprendizado de máquina.

Como os relatórios já são feitos por um programa a necessidade de normalização (corrigir valores faltantes ou dados colocados de maneira errada) não existe, os dados estão prontos para serem analisados e utilizados para algum algoritmo de aprendizado de máquina.

[18] O código usado para fazer essa seleção está no Anexo 3.

## 3. OBJETIVOS DO PROJETO

Analisar e explorar a base de dados IoT-23 a fim de construir um sistema de detecção de tráfego malicioso em redes IoT utilizando algoritmos de aprendizado de máquina, culminando no desenvolvimento de uma plataforma para visualização e consulta interativa dos dados processados e dos resultados obtidos.

### 3.1. OBJETIVOS ESPECÍFICOS

- Realizar uma análise exploratória dos dados (EDA) da base IoT-23, investigando a distribuição das variáveis, correlações relevantes, padrões comportamentais e eventuais anomalias, com o intuito de subsidiar a etapa de modelagem e definição das variáveis preditoras mais relevantes.
- Construir e avaliar modelos de aprendizado de máquina, com foco em tarefas de classificação binária (tráfego benigno versus malicioso), classificação multiclasse (tipos de ataques) e detecção de anomalias, buscando identificar o tráfego malicioso de forma automatizada e precisa a partir de atributos extraídos do tráfego de rede.
- Desenvolver uma plataforma em HTML, com visualização de estatísticas, filtros interativos e gráficos, permitindo a apresentação clara dos resultados obtidos pelas análises e modelos aplicados, além de oferecer a possibilidade de consulta por usuários finais interessados na segurança de redes IoT.

## 4. METODOLOGIA DO PROJETO

Para embasar teoricamente o desenvolvimento deste projeto foram utilizadas fontes confiáveis como livros acadêmicos, artigos científicos e publicações especializadas em tecnologia da informação, segurança cibernética e análise de dados. A fundamentação teórica aborda a importância da adoção de tecnologias digitais no monitoramento e defesa de redes, os benefícios da automação na detecção de ameaças em dispositivos IoT, e as melhores práticas no desenvolvimento de sistemas orientados a dados.

A utilização de ferramentas tecnológicas robustas é essencial para garantir a segurança em ambientes cada vez mais interconectados. De acordo com LAUDON e LAUDON (2020, p. 215), a implementação de sistemas informatizados voltados ao controle e à análise de dados permite que organizações otimizem seus processos internos, reduzam riscos e tomem decisões com base em informações confiáveis. Em um cenário onde milhões de dispositivos IoT estão conectados a redes residenciais, industriais e públicas, esse tipo de gestão digital se torna indispensável.

No contexto da ciber-segurança, o comportamento de usuários e dispositivos revela uma crescente expectativa por agilidade na resposta a incidentes e por sistemas inteligentes capazes de identificar padrões maliciosos em tempo real. Como observam KOTLER et al. (2017, p. 62), a experiência do usuário moderno é

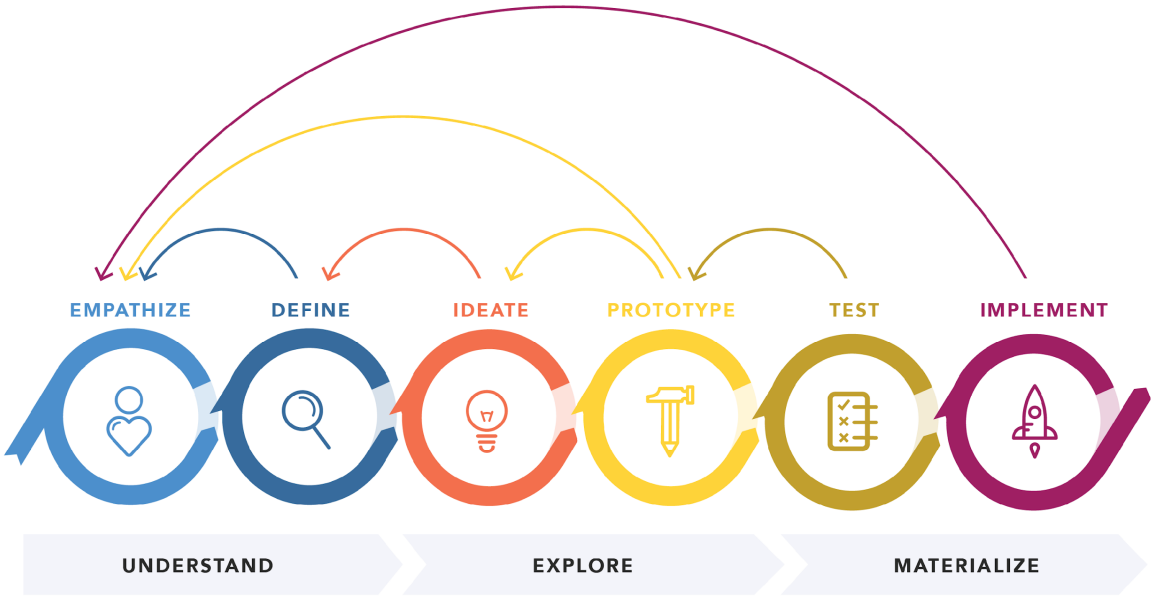




Figura 1: Diagrama do Design Thinking. Fonte: GIBBONS, 2016.

moldada pela percepção de confiança, praticidade e proteção, elementos fundamentais também em aplicações voltadas à segurança digital.

De acordo com Chaffey e Ellis-Chadwick (2019, p. 285), a automação de processos operacionais, como a detecção de tráfego anômalo, permite maior eficiência e redução de erros manuais. Sistemas automatizados com inteligência embarcada são capazes de reagir mais rapidamente a ameaças, além de fornecer alertas e relatórios úteis para os analistas. Técnicas como lembretes automáticos mencionadas no contexto de marketing digital (CHAFFEY & ELLIS-CHADWICK, 2019, p. 290) encontram paralelo funcional aqui nos mecanismos de alerta em tempo real e *dashboards* de monitoramento.

O desenvolvimento técnico do projeto segue os princípios da Engenharia de Software, buscando a criação de uma plataforma de visualização feita em HTML/ CSS para análise, visualização e classificação de tráfego de rede com base em dados reais. Serão utilizadas metodologias ágeis, como *Scrum*, que permitem entregas incrementais e testes contínuos dos modelos e da interface de visualização. Turban et al. (2018, p. 348) ressaltam que esse tipo de abordagem é ideal para projetos orientados por dados e requisitos variáveis. A arquitetura será modular e adaptada a padrões reconhecidos, garantindo escalabilidade e manutenção futura (TURBAN et al., 2018, p. 355).

A análise de dados é um dos pilares deste projeto. A base IoT-23 será explorada com ferramentas estatísticas e computacionais para organização, filtragem, seleção de atributos e visualização interativa. A proposta inclui, ainda, a criação de modelos supervisionados de aprendizado de máquina para classificação de tráfego malicioso e Benigno. Segundo Creswell (2014, p. 195), a análise de dados envolve processos sistemáticos de interpretação e organização, com o objetivo de identificar padrões e relações significativas. Isso se aplica diretamente à modelagem preditiva, onde algoritmos extraem conhecimento útil a partir de dados rotulados.

A aplicação dos modelos será acompanhada de métricas rigorosas, como acurácia[19], precisão[20], recall[21] e AUC-ROC[22]. A visualização dos resultados será feita por meio de uma plataforma de visualização feita em HTML/CSS interativa desenvolvida com HTML, CSS, JavaScript e ferramentas como Chart.js, permitindo que os usuários explorem os dados em gráficos, tabelas e filtros dinâmicos.

Essa metodologia, centrada na integração entre engenharia de software, análise de dados e aprendizado de máquina, fornece os subsídios necessários para alcançar o objetivo do projeto: criar um sistema capaz de analisar e visualizar tráfego IoT com acurácia e aplicabilidade prática.

---

[19] Acurácia é uma métrica de avaliação em aprendizado de máquina que indica a proporção de previsões corretas feitas por um modelo em relação ao total de previsões realizadas. É calculada dividindo-se o número de classificações corretas pelo número total de amostras avaliadas.

[20] Precisão (*precision*) é uma métrica de avaliação em aprendizado de máquina que mede a proporção de previsões positivas corretas feitas por um modelo em relação ao total de previsões positivas realizadas. Em outras palavras, indica quantos dos itens classificados como positivos são realmente positivos.

[21] Revocação (*recall*) é uma métrica de avaliação em aprendizado de máquina que mede a proporção de previsões positivas corretas em relação ao total de casos positivos existentes no conjunto de dados. Em outras palavras, indica quantos dos itens que realmente pertencem à classe positiva foram corretamente identificados pelo modelo.

[22] AUC-ROC, *Area Under the Curve – Receiver Operating Characteristic* (Área sob a Curva – Característica de Operação do Receptor), uma métrica utilizada para avaliar o desempenho de modelos de classificação binária. A curva ROC representa a relação entre a taxa de verdadeiros positivos (*True Positive Rate*) e a taxa de falsos positivos (*False Positive Rate*) em diferentes limiares de decisão. O valor da AUC indica a capacidade global do modelo de distinguir entre as classes, variando de 0 a 1 — quanto mais próximo de 1, melhor a capacidade discriminatória.

## 5. ANÁLISE DOS DADOS E ALGORITMOS DE APRENDIZADO DE MÁQUINA

A análise inicial dos dados provenientes da base IoT-23 tem como objetivo compreender o perfil geral do tráfego registrado e fornecer subsídios para as etapas subsequentes de pré-processamento e modelagem de aprendizado de máquina. Esse tipo de abordagem é fundamental para identificar padrões e características que possam auxiliar na detecção de atividades maliciosas e no aprimoramento de sistemas de segurança de redes baseados em dados (HAN et al., 2020, p. 3; MECHELLI et al., 2022, p. 6).

A primeira etapa consiste na quantificação das conexões por rótulo, ou seja, a separação entre tráfego benigno e malicioso a partir da variável label. Essa medida permite avaliar o balanceamento da base de dados e estimar a necessidade de técnicas específicas de tratamento de desbalanceamento, como *oversampling* ou *undersampling*, comuns em problemas de classificação de intrusões (CHAWLA et al., 2002, p. 328).

Em seguida, é realizada a distribuição das categorias de ataque contidas no atributo *detailed*-label. Essa variável descreve o tipo específico de ameaça, como ataques de negação de serviço distribuído (DDoS), *botnets* ou variações do *malware* Mirai, permitindo mapear o espectro de comportamentos maliciosos presentes na base. A identificação de quais ataques ocorrem com maior frequência fornece indícios sobre possíveis vetores predominantes e auxilia na priorização de esforços de detecção (ZHU et al., 2021, p. 112).

Outro ponto relevante da análise é o cálculo de estatísticas descritivas para variáveis numéricas, como *duration* (tempo de conexão), orig_bytes e resp_bytes (quantidade de bytes transmitidos e recebidos), bem como orig_pkts e resp_pkts (número de pacotes enviados e recebidos). Essas métricas permitem identificar padrões médios, valores extremos e dispersão, oferecendo um panorama das diferenças entre tráfego Benigno e malicioso em termos de volume e intensidade das conexões (SOMMER; PAXSON, 2010, p. 28).

A análise de frequências de protocolos (proto) e serviços (*service*) representa outra dimensão essencial, pois certos protocolos de comunicação, como TCP, UDP

ou ICMP[23], e serviços, como HTTP, DNS ou SSH, podem ser mais frequentemente explorados por ataques específicos. O mapeamento dessas ocorrências facilita a criação de *features* relevantes para os modelos e possibilita o desenvolvimento de políticas de segurança mais específicas.

Por fim, o uso do campo ts (*timestamp*) permite investigar padrões temporais, como a ocorrência de picos de tráfego malicioso em horários ou dias específicos. Essa análise pode revelar comportamentos automatizados ou campanhas coordenadas de ataque, que geralmente seguem ciclos temporais previsíveis (KORONIOS et al., 2022, p. 14).

Essa etapa de análise exploratória, portanto, não apenas caracteriza a base IoT-23, mas também fundamenta as escolhas metodológicas para o aprendizado de máquina, garantindo que o processo de modelagem esteja alinhado às características intrínsecas dos dados.

## 5.1. QUANTIFICAÇÃO DAS CONEXÕES POR RÓTULO

A quantificação das conexões por rótulo tem como objetivo identificar a proporção de tráfego benigno e malicioso no conjunto de dados, a partir da variável *label*. Essa informação é essencial para compreender o nível de desbalanceamento da base, que é um dos fatores críticos no desenvolvimento de modelos de aprendizado de máquina voltados para a detecção de intrusões. Bases altamente desbalanceadas — por exemplo, com predominância de registros benignos — podem levar a modelos enviesados, com baixa capacidade de identificar comportamentos maliciosos (CHAWLA et al., 2002, p. 328).

No contexto da base IoT-23, cada registro de conexão é previamente classificado como benigno (tráfego legítimo) ou malicioso (tráfego malicioso) e essa divisão inicial permite tanto avaliar a qualidade e diversidade dos dados quanto indicar a necessidade de aplicar técnicas de balanceamento artificial, como o

[23] ICMP, *Internet Control Message Protocol* (Protocolo de Mensagens de Controle da Internet), um protocolo da camada de rede usado para enviar mensagens de erro, diagnóstico e controle entre dispositivos. Ele é amplamente utilizado para verificar a conectividade e o estado da rede, como nos comandos *ping* e *traceroute*.

*Synthetic Minority Over-sampling Technique* (SMOTE)[24] ou o *Random Under-sampling*. A análise desse ponto fornece, portanto, a primeira visão quantitativa sobre a estrutura dos dados e fundamenta as decisões de pré-processamento necessárias para treinar algoritmos de detecção mais robustos.[25]

O resultado revelou um total de 637.485 conexões maliciosas e 109.047 conexões benignas, sendo que, dentro deste segundo grupo, observou-se uma inconsistência na rotulagem: parte dos registros benignos estava marcada como "Benign" e outra parte como "benign". Essa divergência exige uma etapa prévia de padronização dos valores da variável alvo (*target*), de forma a unificar a nomenclatura e evitar que sejam interpretadas como categorias distintas durante a análise.

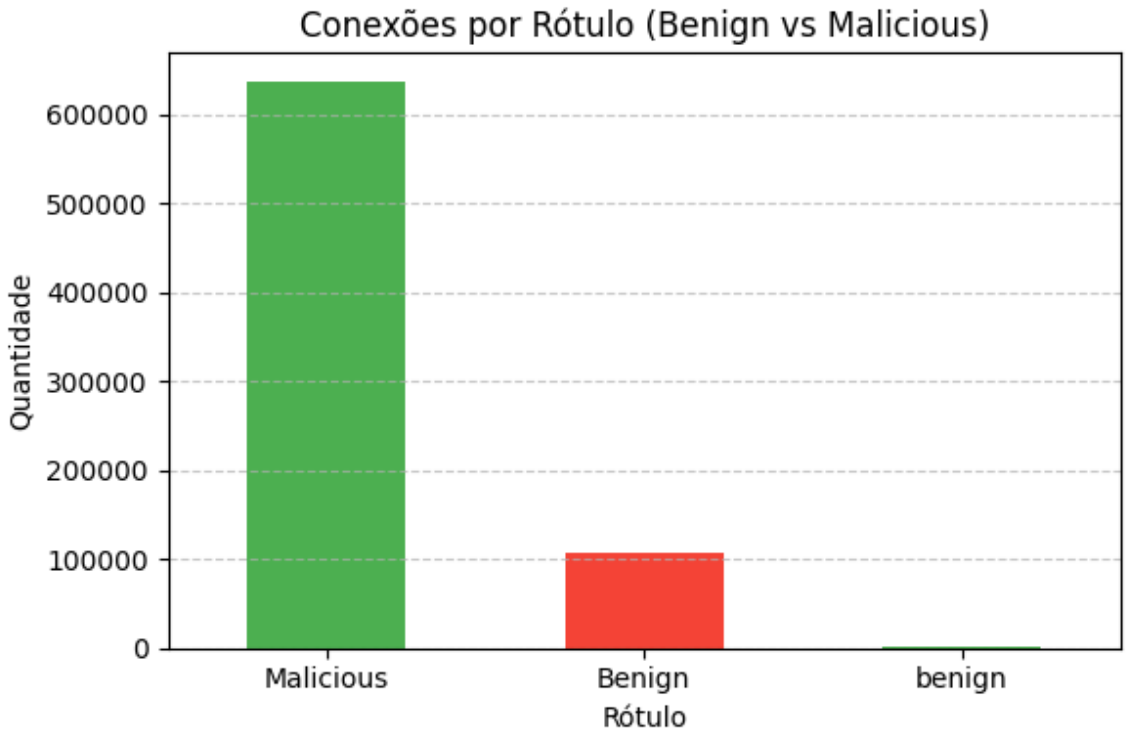


Figura 1: Gráfico de Benignos e malignos.

[24] SMOTE (*Synthetic Minority Over-sampling Technique*) é uma técnica utilizada no pré-processamento de dados para lidar com conjuntos desbalanceados em problemas de classificação. Ela cria amostras sintéticas da classe minoritária, em vez de simplesmente replicar registros existentes, gerando novos pontos de dados por interpolação entre amostras reais próximas. Isso ajuda a reduzir o viés do modelo em favor da classe majoritária e pode melhorar métricas como revocação (*recall*) e AUC-ROC.

[25] O algoritmo utilizado para coletar os dados dessa análise está no Anexo 4.

A distribuição encontrada indica um desbalanceamento de classes significativo, com aproximadamente 84% das conexões classificadas como maliciosas e 16% como benignas. Em problemas de aprendizado de máquina supervisionado o desbalanceamento de classes pode levar os modelos a privilegiar a predição da classe majoritária, obtendo alta acurácia aparente, mas baixo desempenho na detecção de instâncias da classe minoritária (CHAWLA et al., 2002, p. 330).

Para lidar com esse problema, será necessário adotar estratégias de balanceamento de dados, como *undersampling* da classe majoritária, *oversampling* da classe minoritária ou SMOTE. Além disso, a avaliação de desempenho dos modelos deverá priorizar métricas como F1-score, precisão (*precision*) e revocação (*recall*), que fornecem uma visão mais fiel do desempenho em cenários de classes desbalanceadas (HE; GARCIA, 2009, p. 1263).

A variável *label* será utilizada como variável dependente (variável-alvo) nos modelos supervisionados a serem desenvolvidos, permitindo treinar algoritmos capazes de classificar novas conexões de rede como benignas ou maliciosas com base nas demais características registradas no conjunto de dados. Essa etapa é essencial para que a base IoT-23 seja aplicada em soluções de detecção de intrusão, prevenção de ataques e análise comportamental de tráfego de rede no contexto de dispositivos IoT.

## 5.2. DISTRIBUIÇÃO DAS CATEGORIAS DE ATAQUES

Após a padronização da coluna *label*, procedeu-se à análise da variável *detailed*-label, que especifica a categoria de cada conexão maliciosa, como por exemplo DDoS, Mirai-UDP, Okiru[26], entre outras. Essa etapa é fundamental para compreender o perfil das ameaças presentes na base IoT-23 e identificar quais tipos de ataque são mais frequentes no contexto de dispositivos IoT.

A distribuição das categorias de ataques fornece indicadores importantes para a priorização de esforços de defesa, pois permite identificar vetores de ataque predominantes e direcionar a configuração de sistemas de detecção e prevenção de

[26] Okiru é uma variante do *malware* Mirai projetada para infectar dispositivos de Internet das Coisas (IoT), incluindo equipamentos com processadores baseados em arquitetura ARC, comuns em sistemas embarcados. Assim como outras versões do Mirai, o Okiru explora credenciais fracas ou vulnerabilidades conhecidas para comprometer dispositivos e incorporá-los a uma *botnet*, que pode ser usada para executar ataques distribuídos de negação de serviço (DDoS).

intrusões (*Intrusion Detection and Prevention Systems* – IDPS). Além disso, essa análise serve como subsídio para a seleção de características (*feature selection*), já que alguns tipos de ataques podem ser melhor identificados por variáveis específicas, como portas de destino, protocolos utilizados ou padrões de volume de tráfego (MOUSTAFA; SLADEK; TURNBULL, 2019, p. 22).

A identificação das categorias também contribui para modelos de aprendizado supervisionado multiclasses, onde cada instância maliciosa é classificada em uma categoria específica de ataque. Alternativamente, pode-se adotar uma abordagem hierárquica, treinando um primeiro modelo binário para distinguir conexões benignas de maliciosas e, em seguida, um modelo secundário para classificar o tipo específico de ataque entre as conexões maliciosas detectadas.

Compreender a distribuição das categorias não apenas melhora a eficiência operacional de soluções de segurança, mas também aumenta a interpretabilidade dos modelos de aprendizado de máquina, permitindo explicar quais ameaças estão sendo identificadas e por quais características elas são detectadas (SOMMER; PAXSON, 2010, p. 9).

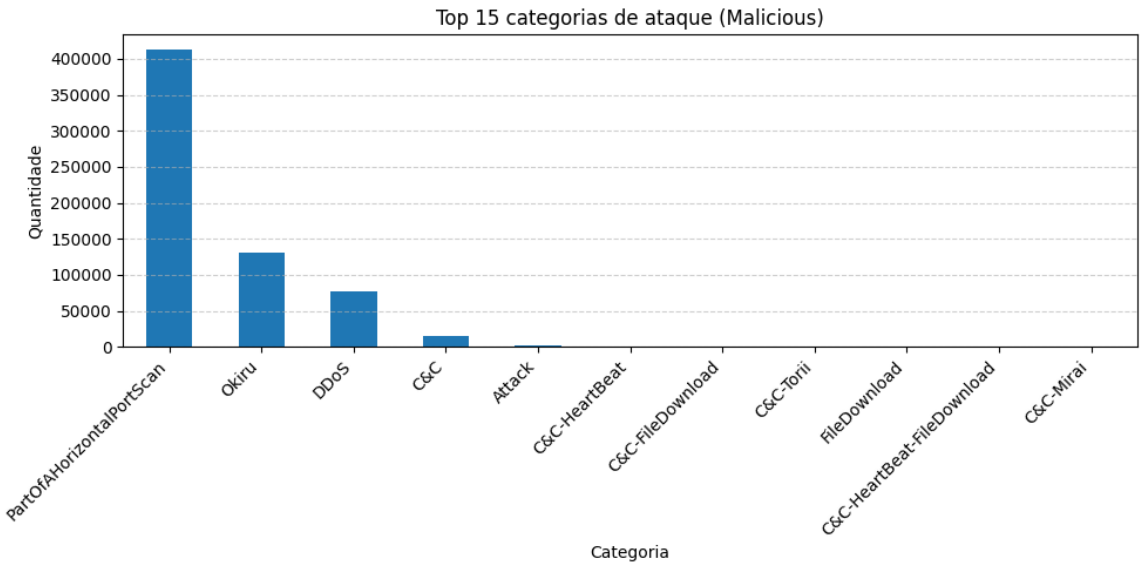


Figura 2 - Gráfico das frequências dos ataques por categorias.

A análise da coluna *detailed-label* para conexões classificadas como maliciosas mostrou que a maior parte das ocorrências concentra-se em poucas categorias, indicando padrões claros de comportamento malicioso na base IoT-23. A categoria *PartOfAHorizontalPortScan* foi predominante, representando 64,79% de

todas as conexões maliciosas. Esse tipo de ataque caracteriza-se por tentativas sistemáticas de varredura horizontal de portas em múltiplos hosts, possivelmente visando identificar serviços vulneráveis em dispositivos IoT expostos.

A segunda categoria mais frequente foi Okiru (20,60%). Em seguida, observou-se a categoria DDoS (11,98%), que consiste em ataques distribuídos de negação de serviço, potencialmente orquestrados a partir de dispositivos comprometidos. Outras categorias, como C&C (2,36%), Attack (0,22%) e diversas subcategorias de comunicação com servidores de comando e controle, apresentaram ocorrências significativamente menores.

Essa concentração em poucas categorias é relevante, pois indica que um conjunto restrito de padrões de tráfego pode responder pela maioria das ameaças detectadas, facilitando a criação de modelos de aprendizado de máquina focados em identificar assinaturas e comportamentos associados a esses ataques mais comuns, aumentando assim a eficácia de sistemas de detecção baseados em dados de rede (Sommer & Paxson, 2010; Ring et al., 2019).

### 5.3. ESTATÍSTICAS DESCRITIVAS DAS VARIÁVEIS QUANTITATIVAS

Nesta etapa, o objetivo é caracterizar numericamente o comportamento das conexões presentes no conjunto de dados IoT-23, utilizando variáveis que representam atributos de tráfego de rede. A análise será conduzida sobre as seguintes métricas: *duration* (tempo de duração da conexão, em segundos), *orig_bytes* (quantidade de bytes enviados pelo host de origem), *resp_bytes* (quantidade de bytes enviados pelo host de destino), *orig_pkts* (número de pacotes enviados pelo host de origem) e *resp_pkts* (número de pacotes enviados pelo host de destino).

Para cada variável, serão calculadas estatísticas descritivas como média, mediana, desvio-padrão, mínimo e máximo, possibilitando identificar a dispersão e a variação dos valores. Essa abordagem fornece uma visão inicial sobre a distribuição dos dados e permite detectar possíveis discrepâncias, como valores anômalos ou extremos que podem indicar comportamentos atípicos — sejam eles fruto de ataques, mau funcionamento de dispositivos ou características legítimas do tráfego de rede (Chaffey & Ellis-Chadwick, 2019; Kotler et al., 2017).

A análise também servirá como base para decisões posteriores sobre tratamento e normalização dos dados, garantindo que modelos de aprendizado de máquina possam trabalhar de forma mais eficiente. Ao compreender o perfil geral das conexões, será possível avaliar padrões médios de comunicação, diferenças no volume de dados transmitidos e a duração típica das interações, além de identificar se determinadas métricas apresentam variações significativas entre conexões benignas e maliciosas.[27]

Tráfego geral:

| | Contagem | Média | Mediana | Desvio Padrão | min | max | Perda de pacotes |
|---|---|---|---|---|---|---|---|
| **orig_pkts** | 746532 | 343,208 | 1,0 | 100582,78 | 0,0 | 66027354 | 0,0 |
| **resp_pkts** | 746532 | 0,503 | 0,0 | 278,16 | 0,0 | 239484 | 0,0 |

Tráfego por rótulo:

| Rótulo | Contagem | Média | Mediana | Desvio Padrão | min | max | Perda de pacotes |
|---|---|---|---|---|---|---|---|
| **Benigno** | 109047 | 147,10 | 1,0 | 43480,10 | 0,0 | 14319118 | 0,0 |
| **Benigno** | 109047 | 2,65 | 0,0 | 726,05 | 0,0 | 239484 | 0,0 |
| **Malicioso** | 637485 | 376,75 | 1,0 | 107350,28 | 0,0 | 66027354 | 0,0 |
| **Malicioso** | 637485 | 0,13 | 0,0 | 20,90 | 0,0 | 9307 | 0,0 |

Os resultados mostram um contraste muito claro entre o tráfego benigno e o malicioso, tanto em número de pacotes enviados (orig_pkts) quanto recebidos (resp_pkts):

No tráfego geral

- O valor médio de pacotes enviados por conexão (orig_pkts) é de 343, mas a mediana é apenas 1. Isso indica uma distribuição altamente assimétrica, com muitos casos de tráfego mínimo e alguns poucos casos extremos que elevam muito a média (picos de até 66 milhões de pacotes).
- O tráfego recebido (resp_pkts) é, em média, muito baixo (0,5 pacotes), também com mediana zero e presença de valores extremos (até 239 mil pacotes). Isso sugere que muitas conexões não receberam resposta ou foram unidirecionais.

[27] O código utilizado para esse cálculo estatístico está no Anexo 6.

Por rótulo (*label*)

- Benignos: A média de pacotes enviados é de 147, com mediana de 1 e grande dispersão (desvio padrão altíssimo). O tráfego recebido tem média de 2,65 pacotes, mas também com mediana zero.
- Maliciosos: A média de pacotes enviados é muito maior (376 contra 147 nos benignos), o que indica que ataques tendem a gerar mais tráfego de saída. Curiosamente, a média de pacotes recebidos é menor nos ataques (0,13 contra 2,65), o que pode estar relacionado a ataques como *port scans* e DoS, onde não há resposta do destino.

A presença de valores extremos sugere que será necessário aplicar normalização ou transformação logarítmica para atenuar o impacto de *outliers* nos modelos de aprendizado de máquina.

A diferença nos padrões de tráfego enviado e recebido pode servir como variável discriminante importante para a classificação entre tráfego benigno e malicioso.

O fato de a mediana ser muito menor que a média indica que muitos ataques e conexões benignas têm comportamento “mínimo” (poucos pacotes), enquanto uma minoria gera volumes massivos.

## 5.4. FREQUÊNCIA DOS PROTOCOLOS E SERVIÇOS

A análise da frequência de protocolos e serviços é fundamental para compreender como o tráfego de rede se distribui entre diferentes camadas e aplicações. Protocolos como TCP, UDP e ICMP compõem a base da comunicação na internet e, no contexto de segurança, determinados protocolos podem ser mais explorados em ataques devido às suas características técnicas ou vulnerabilidades conhecidas.

Da mesma forma, a inspeção dos serviços detectados (por exemplo, HTTP, DNS, SSH) permite identificar quais aplicações estão mais presentes nas comunicações e se estas apresentam padrões suspeitos. Em cenários de tráfego malicioso, é comum observar concentrações de tentativas de conexão em serviços

específicos, o que pode indicar varreduras (*scans*) ou tentativas de exploração direcionadas.[28]

Essa análise possibilita:

- Identificar protocolos predominantes no tráfego Benigno e malicioso;
- Mapear serviços mais visados por atacantes;
- Detectar comportamentos atípicos de uso de protocolo/serviço em determinadas categorias de ataque; e
- Fornecer subsídios para filtragem e mitigação de tráfego malicioso.

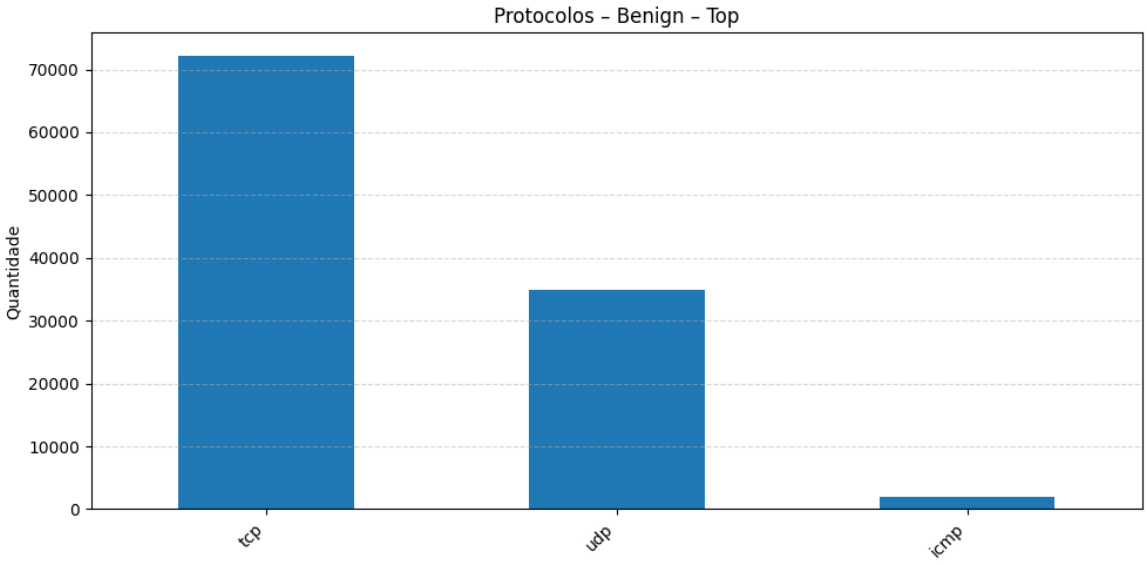


Figura 3: Gráfico da frequência dos protocolos nos acessos Benignos.

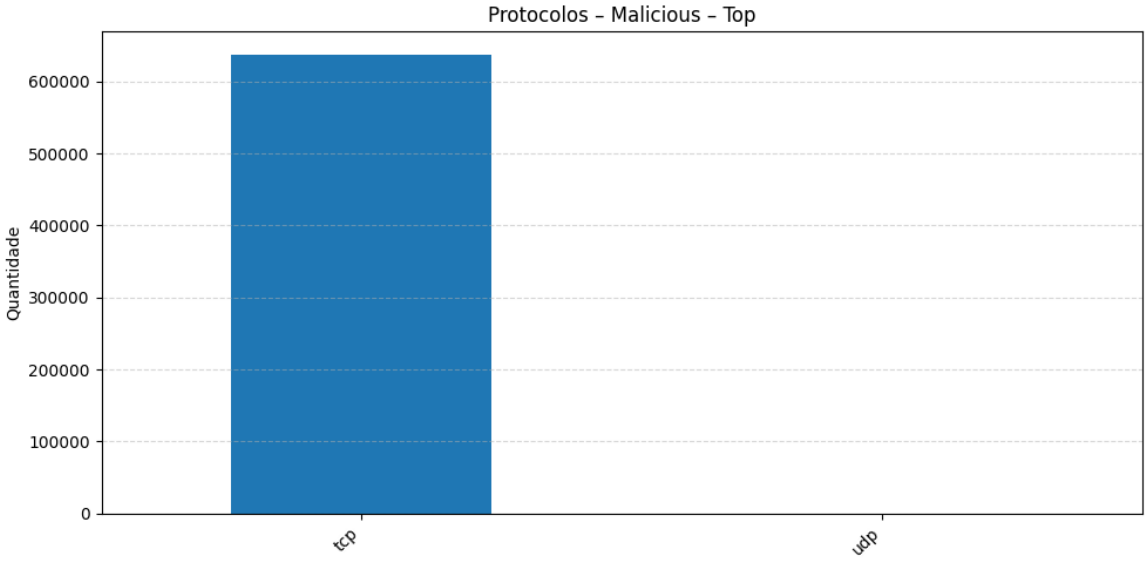


Figura 4: Gráfico da frequência dos protocolos nos acessos maliciosos.

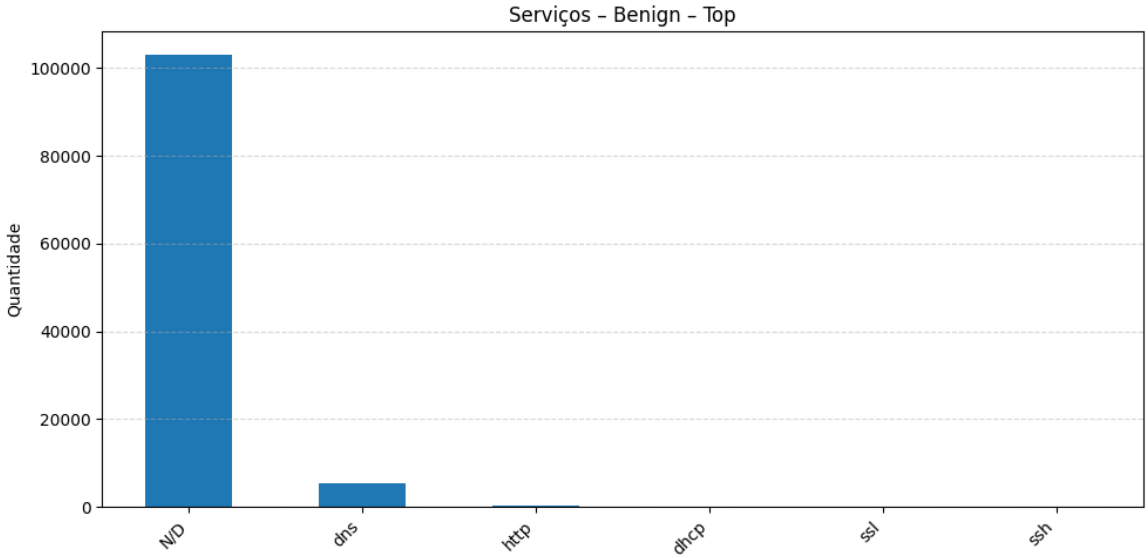


Figura 5: Gráfico da frequência dos serviços nos acessos Benignos.

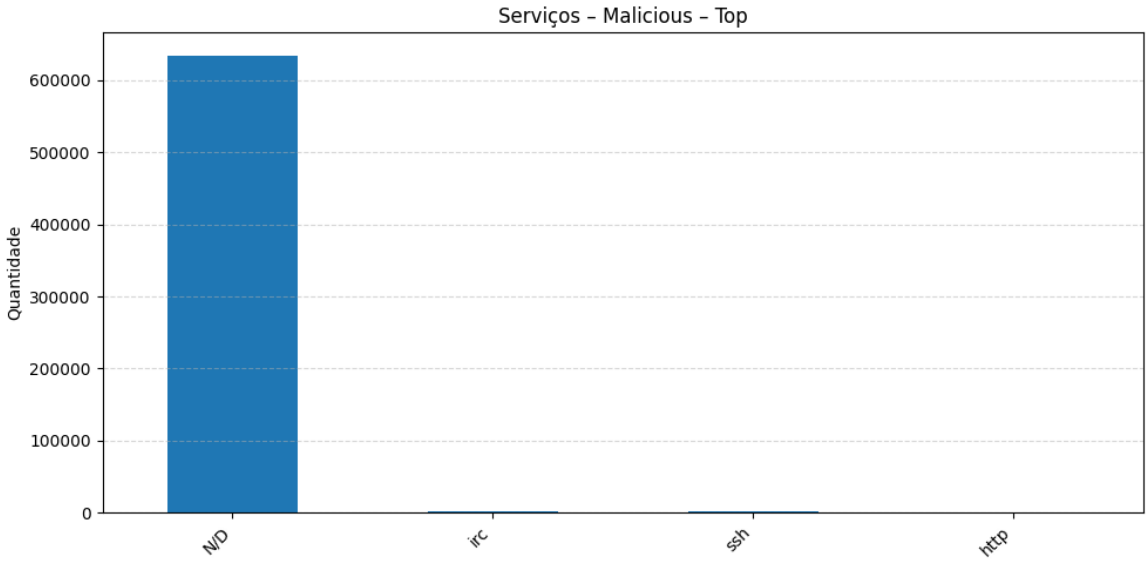


Figura 6: Gráfico da frequência dos serviços nos acessos maliciosos.

A análise de frequência de protocolos no tráfego geral indica um predomínio do TCP (95,06%), seguido pelo UDP(4,67%) e ICMP (0,26%). Essa distribuição é consistente com a natureza da maior parte das comunicações em redes IP, nas quais o TCP é amplamente utilizado para transmissões confiáveis, enquanto o UDP

é empregado em serviços de menor *overhead*, como consultas DNS ou *streaming*, e o ICMP é voltado para mensagens de controle e diagnóstico.

Em relação aos serviços, a grande maioria das conexões foi classificada como "N/D" (98,78%), indicando ausência ou não detecção de identificação do serviço pela ferramenta de captura. Dentre os serviços identificados, destacam-se DNS(0,72%), IRC (0,22%), SSH (0,18%) e HTTP (0,06%). Isso sugere que boa parte das conexões não se refere a serviços de aplicação mapeáveis ou não foi possível determinar tal informação nos pacotes analisados.

Ao segmentar a análise por rótulo, observou-se que, no tráfego Benigno, o TCP representa 66,24% e o UDP 31,96% das conexões, com um percentual mais equilibrado entre os dois protocolos. Já no tráfego Malicioso, há praticamente um monopólio do TCP (100%), com apenas 25 conexões UDP, o que pode refletir o uso predominante desse protocolo em ataques orientados a sessão ou em comunicação com servidores de comando e controle (C2).

Nos serviços, o tráfego Benigno apresentou maior diversidade: além do grande volume de conexões N/D (94,57%), foram identificados DNS (4,91%), HTTP, DHCP, SSL e SSH em menores proporções. Já no tráfego Malicioso, o cenário é menos diversificado: N/D (99,50%), com pequenas ocorrências de IRC (0,26%), SSH (0,20%) e HTTP (0,03%). O destaque para o IRC no tráfego malicioso é relevante, uma vez que este protocolo tem histórico de utilização para controle de *botnets*, sugerindo que parte dos eventos analisados pode estar associada a esse tipo de ameaça.

Em suma, essa análise revela que:

- O TCP é dominante no tráfego malicioso, enquanto o tráfego Benigno apresenta maior equilíbrio com o UDP.
- O IRC e o SSH aparecem mais significativamente no tráfego malicioso, reforçando seu potencial uso para atividades suspeitas.
- A elevada proporção de serviços N/D pode indicar necessidade de aprofundar a inspeção para melhor classificação, possivelmente com técnicas de inspeção profunda de pacotes (Deep Packet Inspection – DPI).

## 5.5. PADRÃO TEMPORAL DE ACESSOS MALICIOSOS

Para verificar a existência de padrões temporais nos acessos classificados como maliciosos, analisamos o campo ts, (*timestamp*) do início da conexão no formato *Unix Epoch* (segundos desde 1º de janeiro de 1970). A conversão desse valor para um formato de data e hora legível permite agrupar e visualizar os eventos por diferentes escalas temporais — horas, dias, semanas ou meses — e, assim, identificar concentrações ou ciclos de atividade.

Essa abordagem é útil para:

- Detectar picos de atividade maliciosa: picos concentrados em horários ou dias específicos podem indicar ataques coordenados.
- Identificar padrões sazonais: ataques podem ocorrer mais frequentemente em determinados dias da semana ou horários do dia.
- Relacionar eventos com outros fatores: como atualizações de sistemas, campanhas de *malware* ou eventos específicos na rede.

Se padrões claros forem identificados, isso pode auxiliar no planejamento de defesas preventivas, reforçando a segurança nos períodos de maior risco.[29]

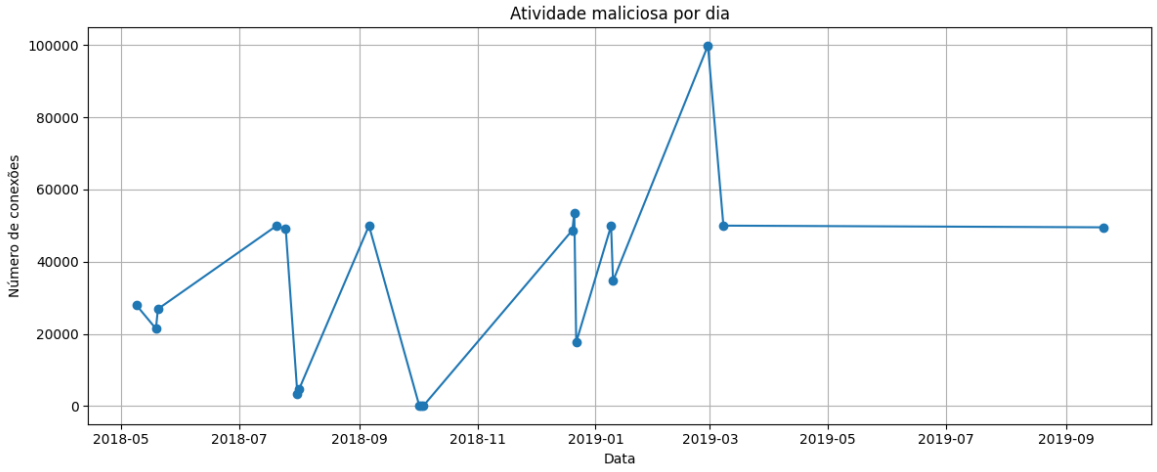


Figura 7: Gráfico do histórico de acessos maliciosos ao longo de 05/2018 até 09/2019.

A análise temporal do tráfego rotulado como malicioso foi conduzida em duas perspectivas complementares: distribuição diária e distribuição horária das conexões detectadas.

A evolução da atividade maliciosa ao longo do tempo revelou um comportamento intermitente, com variações significativas na quantidade de

[29] O código utilizado para essa análise está no Anexo 8.

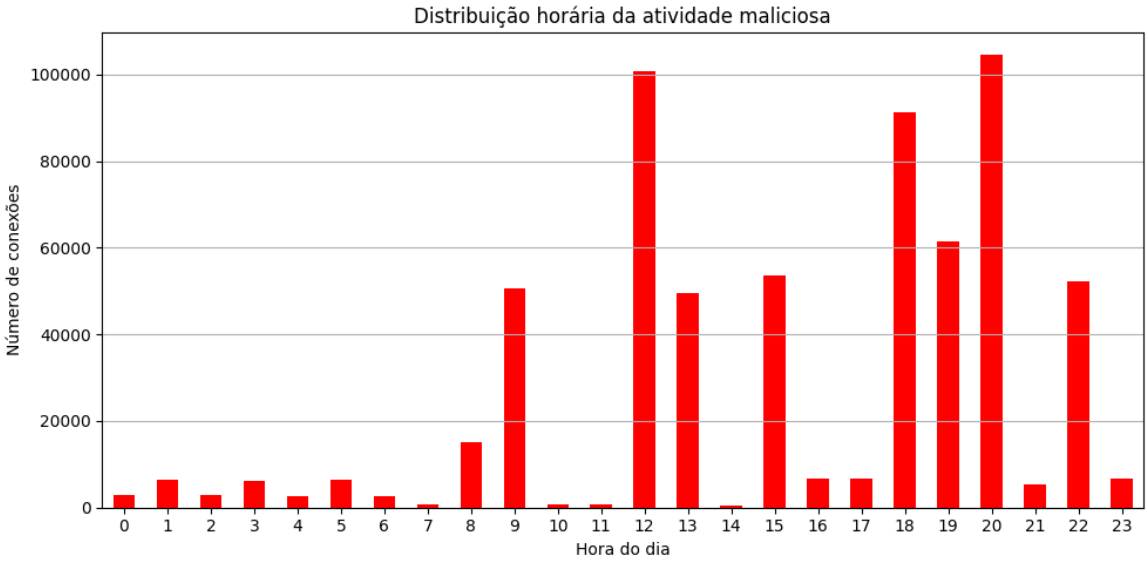


Figura 8: Gráfico do volume de acessos maliciosos categorizados por hora.

conexões por dia. Observou-se que os ataques não se distribuem uniformemente, mas ocorrem em campanhas concentradas, alternando entre períodos de alta intensidade e períodos com pouca ou nenhuma atividade. O maior pico registrado ultrapassou 100 mil conexões maliciosas em um único dia, ocorrendo por volta de fevereiro/março de 2019. Também foram identificados intervalos com atividade praticamente nula, como em setembro e outubro de 2018, possivelmente associados à ausência de tráfego coletado ou a períodos sem ocorrência de ataques. Esses resultados sugerem que as campanhas de ataque têm janelas temporais bem definidas, podendo estar associadas a eventos específicos ou ao uso de infraestrutura de ataque por tempo limitado.

A análise horária demonstrou concentração marcante da atividade maliciosa em determinados horários, com picos às 09h, 12h, 18h e 20h. Esse padrão indica que parte significativa dos ataques é executada em horários programados, possivelmente automatizados, coincidindo com momentos estratégicos em que o tráfego legítimo das redes-alvo pode ser mais intenso. Embora haja registros de atividade em outros horários, especialmente durante a madrugada, esses volumes são substancialmente menores, reforçando a hipótese de que a distribuição temporal não é aleatória.

Os padrões observados oferecem subsídios relevantes para a defesa cibernética, uma vez que permitem antecipar períodos de maior risco e otimizar a

alocação de recursos de monitoramento. Além disso, a correlação entre os horários de pico e a tipologia dos ataques pode contribuir para a identificação de campanhas específicas, melhorando a eficácia das medidas preventivas.

## 6. ALGORITMO DE APRENDIZADO DE MÁQUINA

A detecção de tráfego malicioso é um dos pilares da segurança de redes, desempenhando papel fundamental na mitigação de ataques e na preservação da integridade dos sistemas. A análise exploratória do conjunto de dados IoT-23 revelou a presença de conexões de rede rotuladas como benignas ou maliciosas, bem como categorias específicas de ataques. Este cenário caracteriza um problema de classificação supervisionada, cujo objetivo consiste em treinar modelos preditivos capazes de identificar, a partir de variáveis observadas no tráfego, se uma conexão deve ser classificada como benigna ou maliciosa.

Para este estudo, serão aplicados dois algoritmos amplamente utilizados na detecção de intrusões: *Random Forest* e *Gradient Boosting*.

O *Random Forest* (RF) é um método de aprendizado de máquina baseado em conjuntos de árvores de decisão, proposto por BREIMAN (2001). Sua robustez advém da utilização de múltiplas árvores treinadas com subconjuntos aleatórios de dados e variáveis, combinando seus resultados por votação majoritária. Essa abordagem permite lidar eficientemente com variáveis heterogêneas e dados ruidosos, além de fornecer métricas de importância das variáveis, facilitando a interpretação do modelo (BREIMAN, 2001; HASTIE, TIBSHIRANI & FRIEDMAN, 2017).

O *Gradient Boosting*, por sua vez, é um método de boosting sequencial que constrói modelos fracos (geralmente árvores de decisão) de forma iterativa, corrigindo os erros cometidos por modelos anteriores (FRIEDMAN, 2001). Implementações modernas como *XGBoost* (Chen & Guestrin, 2016), *LightGBM* (Ke et al., 2017) e CatBoost (PROKHORENKOVA et al., 2018) têm demonstrado desempenho superior na detecção de intrusões, especialmente quando há necessidade de capturar interações complexas entre as variáveis e lidar com desequilíbrios de classes.

O processo metodológico proposto para aplicação dos modelos contempla as seguintes etapas:

1. Pré-processamento dos dados

    1.1.Padronização dos nomes das colunas e verificação de valores ausentes.

1.2. Tratamento de variáveis categóricas, como proto (protocolo) e *service* (serviço), por meio de codificação apropriada (*one-hot encoding* ou *label encoding*).

1.3. Definição da variável-alvo (*label*) como indicadora binária de tráfego benigno ou malicioso.

1.4. Balanceamento por conta da superamostragem dos acessos maliciosos. Como o número de acessos maliciosos que temos é muito maior do que os benignos, precisamos fazer esse balancemaneto para que o algoritmo não fique viciado.

2. Divisão do conjunto de dados

    2.1. Separação em conjunto de treino (70%) e conjunto de teste (30%), preservando a proporção das classes (*stratified split*).

3. Treinamento e avaliação dos modelos

    3.1. Treinamento inicial com *Random Forest* como modelo de referência (*baseline*).

    3.2. Treinamento subsequente com *Gradient Boosting*, com ajuste de hiperparâmetros (*hyperparameter tuning*) para maximização do desempenho.

    3.3. Avaliação de ambos os modelos por meio das métricas Acurácia, Precisão, Revocação (*Recall*), F1-Score e Matriz de Confusão (SOLOKOVA & LAPALME, 2009).

4. Validação cruzada

    4.1. Aplicação de *k-fold cross-validation* para verificação da estabilidade dos modelos e redução do risco de sobreajuste (overfitting).

5. Interpretação dos resultados

    5.1. Análise das variáveis mais relevantes para a classificação, permitindo identificar os atributos de tráfego que mais influenciam a detecção de comportamentos maliciosos.

    5.2. Comparação de desempenho entre os modelos, determinando a abordagem mais adequada ao contexto de redes IoT.

A execução deste processo permitirá não apenas a classificação automatizada de conexões de rede com base em padrões previamente aprendidos, mas também a extração de *insights* sobre os fatores que mais contribuem para a ocorrência de tráfego malicioso, apoiando a tomada de decisões em segurança cibernética.[30]

Antes do treinamento, realizou-se o balanceamento da base de dados para mitigar o problema de desbalanceamento de classes, visto que a quantidade de conexões maliciosas superava amplamente a de conexões benignas. O balanceamento foi obtido via *undersampling* da classe majoritária, resultando em 76.333 instâncias para cada classe no conjunto de treinamento.

A avaliação dos modelos foi conduzida utilizando métricas amplamente reconhecidas na literatura (KOTLER et al., 2017; LAUDON & LAUDON, 2020), incluindo acurácia, precisão, revocação, F1-Score e Área sob a Curva ROC (ROC-AUC), calculadas sobre o conjunto de teste.

### 6.1. RESULTADOS DO *RANDOM FOREST*

O modelo RF apresentou desempenho extremamente satisfatório, com acurácia de 98,79%, precisão de 99,84%, revocação de 98,74% e F1-Score de 99,29%. A métrica ROC-AUC atingiu 0,9952, indicando elevada capacidade de discriminação entre conexões benignas e maliciosas.

A análise da importância das variáveis revelou que as portas de origem (id.orig_p) e destino (id.resp_p) foram os atributos mais relevantes para a classificação, seguidas pelo tamanho total de bytes enviados pelo IP de origem (orig_ip_bytes), duração da conexão (*duration*) e número de pacotes enviados pelo host de origem (orig_pkts). Este resultado sugere que o comportamento de tráfego em termos de portas e volume de dados é um forte indicativo para a detecção de ataques.

### 6.2. RESULTADOS DO *GRADIENT BOOSTING*

O modelo GB também apresentou desempenho robusto, com acurácia de 98,53%, precisão de 99,97%, revocação de 98,31% e F1-Score de 99,13%. A métrica ROC-AUC foi ligeiramente superior à do Random Forest, atingindo 0,9981, o que reforça a elevada capacidade preditiva.

[30] O código utilizado para esse treinamento de aprendizado de máquina está no Anexo 9.

Embora ambos os modelos tenham apresentado métricas próximas, o RF obteve ligeiramente maior equilíbrio entre precisão e revocação, enquanto o GB apresentou melhor desempenho no ROC-AUC. A diferença marginal entre os resultados indica que ambos os modelos são adequados para a tarefa, e a escolha final poderá considerar fatores como tempo de treinamento e interpretabilidade.

### 6.3. CONSIDERAÇÕES

Os resultados obtidos demonstram que, mesmo com um processo de balanceamento relativamente simples, é possível atingir elevadas taxas de acerto na detecção de tráfego malicioso. A relevância das portas de origem e destino e dos atributos relacionados ao volume de dados corrobora estudos prévios (TURBAN et al., 2018) que destacam tais parâmetros como indicadores-chave em sistemas de detecção de intrusão.

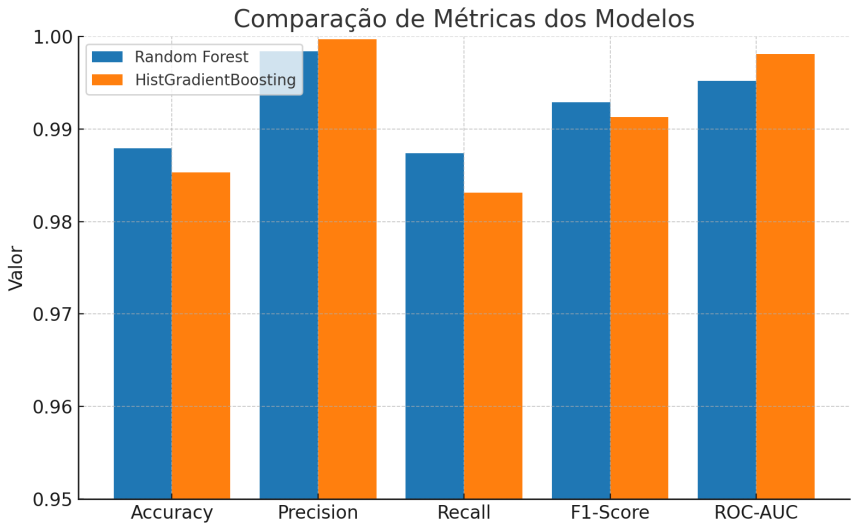


Figura 9: Gráfico de comparação entre Random Forest e Gradient Boosting.

## 7. CRIAÇÃO DA PLATAFORMA PARA APRESENTAÇÃO DOS DADOS

Antes de propriamente criarmos a plataforma para apresentação dos dados, precisamos tratá-los de maneira definitiva e precisamos também de um modelo de aprendizado de máquina treinado para utilização no site.

A base utilizada foi consolidada a partir dos arquivos originais do IoT-23 Dataset, sendo unificada em um único arquivo CSV (merged.csv). Em seguida, aplicou-se um processo de limpeza e padronização:[31]

- Normalização de rótulos: uniformização da variável-alvo (label) para as classes “Benign” e “Malicious”.
- Tratamento de valores numéricos: conversão para tipo numérico com substituição de valores ausentes por zero.
- Padronização de valores categóricos: substituição de campos vazios, traços ou valores nulos por “N/D”.
- Seleção de atributos relevantes: definição de colunas numéricas e categóricas a serem utilizadas pelo modelo.
- Exportação final: armazenamento dos dados tratados no formato Parquet (merged_clean.parquet) para uso direto no sistema, evitando a repetição do processamento.

Antes da modelagem, foi realizada análise exploratória com foco em três eixos:

1. Distribuição das classes — observou-se forte desbalanceamento entre amostras benignas e maliciosas, sendo a classe maliciosa predominante.
2. Frequência de protocolos e serviços — identificação dos principais protocolos (TCP, UDP, ICMP) e serviços (HTTP, DNS, SSH, entre outros) utilizados nos acessos.
3. Padrões temporais — verificação de variações diárias e horárias no volume de acessos maliciosos, permitindo avaliar sazonalidades e horários críticos.

[31] O código utilizado para essa higienização está no Anexo 10.

Essa etapa foi essencial para compreender a estrutura dos dados, identificar potenciais vieses e definir a estratégia de balanceamento.

Como vimos anteriormente o RF teve desempenho levemente melhor com esse caso do que o GB, por conta disso essa foi a técnica escolhida para ser colocada em produção.[32]

Por fim podemos fazer toda a estrutura para construir nossa plataforma de dados. Para isso temos a seguinte estrutura de pastas:

- projeto_iot/
    - app.py [backend[33] em python/flask][34]
    - modelo_rf.joblib [arquivo com o modelo treinado]
    - merged_clean.parquet [dados higienizados]
    - templates/
        - index.html [arquivo HTML principal][35]
        - visualizar_dados.html [Página de visualização de dados][36]
        - prever.html [Página que faz a avaliação do ataque com base no modelo de aprendizado de máquina][37]
    - static/
        - css/
            - style.css [folha de estilos[38]][39]

[32] O algoritmo utilizado para treinar o modelo está no Anexo 11.

[33] *Backend* é um intermediário entre os arquivos HTML e a base de dados configurada. Ele faz a chamada de dados ou métodos para serem utilizados e apresentados no navegador.

[34] O algoritmo que descreve o *backend* está no Anexo 12.

[35] O HTML dessa página está no Anexo 13.

[36] O HTML dessa página está no Anexo 14.

[37] O HTML dessa página está no Anexo 15.

[38] A folha de estilos é o arquivo que determina todas as questões de design da página como fontes, tamanhos, cores, barras de menu, rodapés etc.

[39] O arquivo que descreve o css está no Anexo 16.

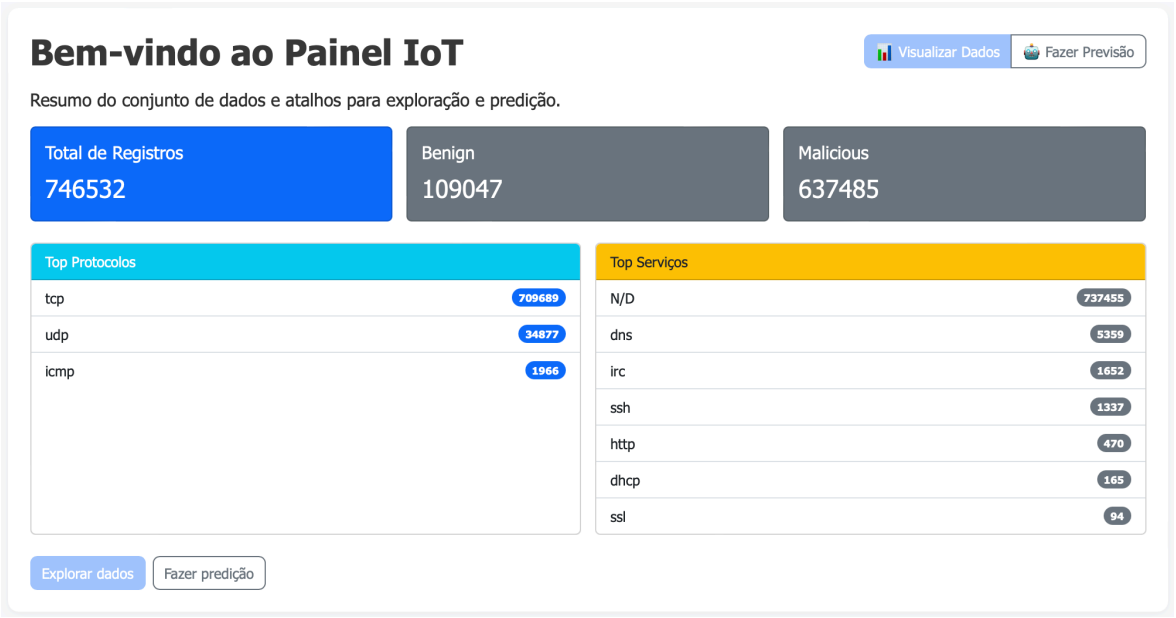


Figura 10: Tela inicial da plataforma.

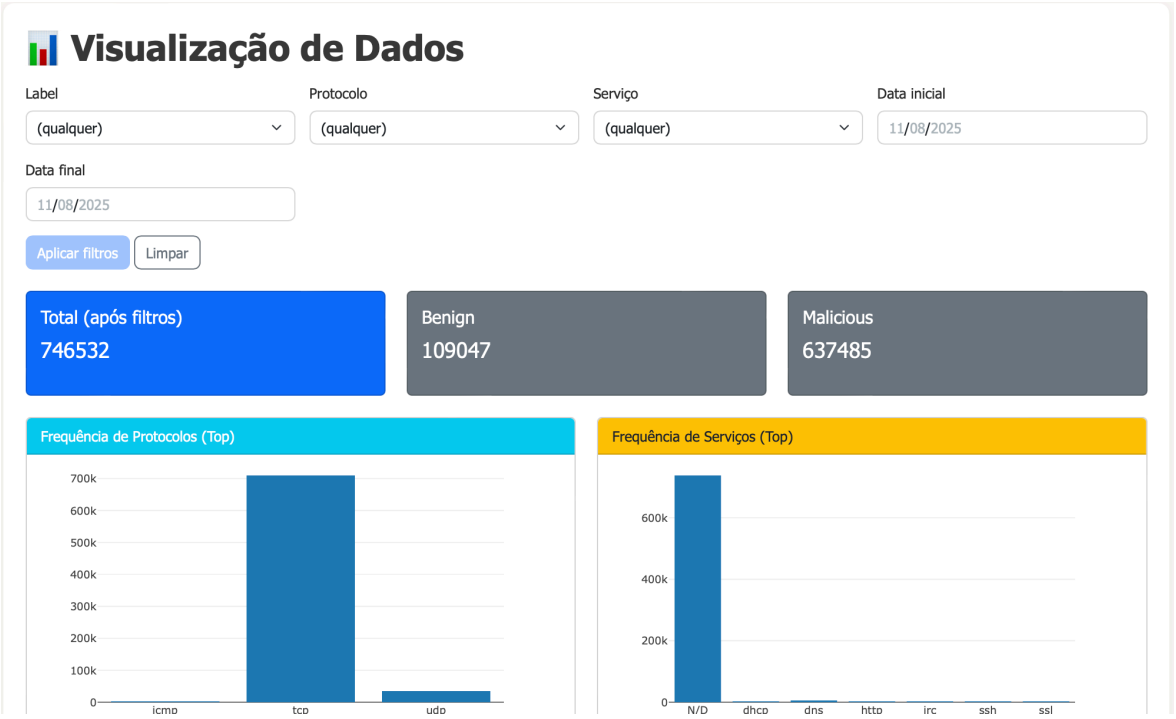


FIgura 11: Tela de exploração de dados.

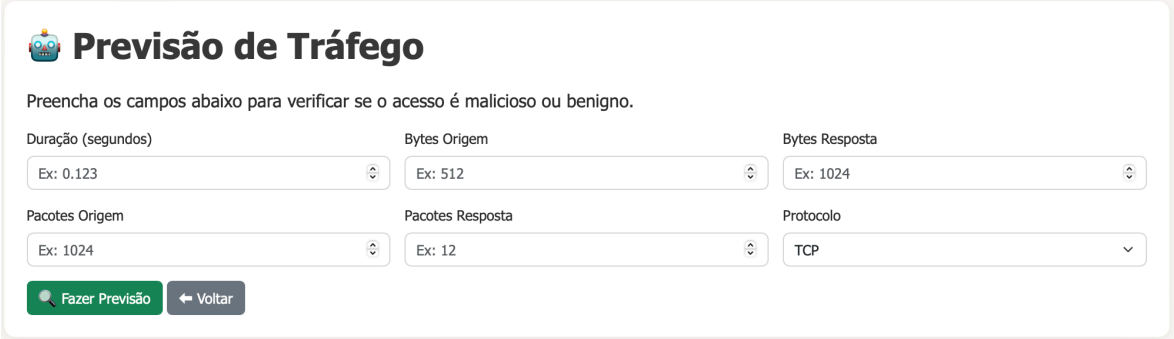


Figura 12: Campos para preenchimento para avaliação de acesso malicioso.

## CONCLUSÃO

O presente estudo demonstrou que a análise sistemática da base de dados IoT-23, aliada a técnicas de aprendizado de máquina, pode oferecer soluções eficientes para a detecção de tráfego malicioso em redes IoT. A aplicação de métodos de pré-processamento e balanceamento de dados foi fundamental para garantir a integridade e a representatividade do conjunto de treinamento, reduzindo vieses e potencializando o desempenho dos modelos.

Os algoritmos *Random Forest* e *Gradient Boosting* apresentaram resultados robustos, com métricas de acurácia, precisão, revocação e F1-Score superiores a 98%, evidenciando a viabilidade do uso de tais técnicas no contexto de segurança cibernética. Destaca-se que, mesmo com um conjunto de dados altamente desbalanceado, foi possível atingir um elevado grau de discriminação entre tráfego benigno e malicioso, reforçando a importância do tratamento adequado das classes e da seleção de atributos relevantes.

A criação da plataforma de visualização feita em HTML/CSS para visualização e consulta interativa dos dados consolida o caráter prático do projeto, permitindo não apenas a exploração visual das estatísticas e padrões identificados, mas também a utilização do modelo em tempo real para classificação de novas conexões. Essa integração entre análise de dados, modelagem preditiva e interface de uso amplia o potencial de aplicação da solução em ambientes reais, contribuindo para o fortalecimento da defesa contra ameaças em redes IoT.

Por fim, este trabalho evidencia que abordagens baseadas em ciência de dados e aprendizado de máquina, quando integradas a ferramentas de visualização acessíveis, podem não apenas identificar ameaças com alta precisão, mas também fornecer subsídios estratégicos para a tomada de decisão em segurança digital. Como perspectivas futuras, recomenda-se a ampliação do escopo para análise de séries temporais, experimentação com técnicas avançadas de balanceamento e adoção de estratégias de *ensemble* para potencializar ainda mais a eficácia do sistema desenvolvido.

## REFERENCIAS BIBLIOGRÁFICAS


BREIMAN, L. (2001). Random forests. Machine Learning, 45(1), 5–32.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794.

CIOS, Krzysztof J.; PEDRYCZ, Witold; SWINIARSKI, Roman W.; KURTZ, Lisa A. Data Mining: A Knowledge Discovery Approach. New York: Springer, 2007.

CHAFFEY, D., & ELLIS-CHADWICK, F. (2019). Digital marketing: Strategy, implementation, and practice (7th ed.). Pearson Education Limited.

CHAWLA, N. V. et al. SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, v. 16, p. 321-357, 2002.

CRESWELL, J. W. (2014). Investigação qualitativa e projeto de pesquisa: escolhendo entre cinco abordagens (3. ed.). Porto Alegre: Penso.

FRIEDMAN, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189–1232.

GARCIA, S. et al. An empirical evaluation of botnet behavior and detection. Stratosphere Laboratory, Czech Technical University in Prague, 2020. p. 2–7. Disponível em: https://www.stratosphereips.org/datasets-iot23. Acesso em: 26 jul. 2025.

GIBBONS, S. (2016). Design Thinking 101. Disponível em: https://www.nngroup.com/articles/design-thinking/?platform=hootsuite. Acessado em 26 de setembro de 2024.

HAN, J.; PEI, Jian; KAMBER, Micheline. Data Mining: Concepts and Techniques. 4. ed. Cambridge, MA: Morgan Kaufmann, 2022.

HAN, X. et al. IoT Network Intrusion Detection Based on Supervised Machine Learning. IEEE Access, v. 8, p. 217027-217037, 2020. DOI: 10.1109/ACCESS.2020.3042462.

HASTIE, T., TIBSHIRANI, R., & FRIEDMAN, J. (2017). The elements of statistical learning: Data mining, inference, and prediction. Springer.

HE, H.; GARCIA, E. A. Learning from imbalanced data. IEEE Transactions on

Knowledge and Data Engineering, v. 21, n. 9, p. 1263-1284, 2009.

KE, G., MENG, Q., FINLEY, T., WANG, T., CHEN, W., MA, W., … & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 30.

KOTLER, P., KARTAJAYA, H., & SETIAWAN, I. (2017). Marketing 4.0: Moving from traditional to digital. Wiley.

KORONIOS, N. et al. Machine Learning Techniques for Cybersecurity: A Comprehensive Review. Future Internet, v. 14, n. 3, p. 1-23, 2022. DOI: 10.3390/fi14030100.

KUMAR, A.; SINGH, P.; TYAGI, H. Malicioso traffic classification in IoT using machine learning. Procedia Computer Science, v. 198, p. 800–807, 2022. DOI: 10.1016/j.procs.2022.06.105.

LAUDON, K. C., & LAUDON, J. P. (2020). Management information systems: Managing the digital firm (16th ed.). Pearson.

MECHELLI, A. et al. A Comprehensive IoT Traffic Analysis Dataset for Machine Learning. Data in Brief, v. 42, p. 1-10, 2022. DOI: 10.1016/j.dib.2022.108353.

MOUSTAFA, Nour; SLADEK, Ross; TURNBULL, Ben. An evaluation of IoT intrusion detection systems and their impact on network traffic. Journal of Network and Computer Applications, v. 144, p. 19-31, 2019.

PROKHORENKOVA, L., GUSEV, G., VOROBEV, A., DOROGUSH, A. V., & GULIN, A. (2018). CatBoost: unbiased boosting with categorical features. Advances in Neural Information Processing Systems, 31.

SOLOKOVA, M., & LAPALME, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427–437.

SOMMER, R.; PAXSON, V. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. IEEE Symposium on Security and Privacy, p. 26-40, 2010. DOI: 10.1109/SP.2010.11.

TURBAN, E., Pollard, C., Wood, G., & Gill, P. (2018). Information technology for management: On-demand strategies for performance, growth, and sustainability (11th ed.). Wiley.

YAVUZ, T.; CELIK, A.; KAYA, H. A comparative study on anomaly detection for IoT networks based on machine learning algorithms. Computer Networks, v. 197, p. 1–11, 2021. DOI: 10.1016/j.comnet.2021.108281.

ZHU, Y. et al. Network Traffic Classification for IoT Devices Using Machine Learning. Sensors, v. 21, n. 1, p. 110-125, 2021. DOI: 10.3390/s21010110.

**ANEXO 1 - CÓDIGO UTILIZADO PARA A LEITURA DOS CABEÇALHOS DOS ARQUIVOS.**

```
import os


def identificar_campos_labeled(base_dir):
    resultados = {}

    # percorre cada subpasta (cenário) dentro do diretório base
    for nome_cenario in os.listdir(base_dir):
        caminho_cenario = os.path.join(base_dir, nome_cenario, 'bro')
        if not os.path.isdir(caminho_cenario):
            continue

        # procura por um arquivo .labeled dentro da pasta 'bro'
        arquivos = [f for f in os.listdir(caminho_cenario) if f.endswith('.labeled')]
        if not arquivos:
            continue

        caminho_arquivo = os.path.join(caminho_cenario, arquivos[0])

        # abre e lê o cabeçalho do arquivo
        with open(caminho_arquivo, 'r', encoding='utf-8', errors='ignore') as f:
            for linha in f:
                if linha.startswith('#fields'):
                    campos = linha.strip().split('\t')[1:]
                    resultados[nome_cenario] = campos
                    break
```

```
    # Exibe os resultados
    for cenario, campos in resultados.items():
        print(f'\n Cenário: {cenario}')
        print(f' Campos encontrados ({len(campos)}):')
        for campo in campos:
            print(f'  - {campo}')

# Substitua pelo caminho onde está o diretório com as 23 pastas
base_de_dados = '.'
identificar_campos_labeled(base_de_dados)
```

**ANEXO 2 - CÓDIGO PARA A VERIFICAÇÃO DE NORMALIZAÇÃO.**

```
import os
import pandas as pd

# Caminho onde estão as 23 pastas
base_dir = '.'  # ou coloque o caminho absoluto se necessário

# Dicionário para guardar resultados por cenário
resumos = []

for pasta in sorted(os.listdir(base_dir)):
    caminho_bro = os.path.join(base_dir, pasta, 'bro')
    if not os.path.isdir(caminho_bro):
        continue

    # Tenta encontrar um arquivo .labeled na pasta 'bro'
    arquivos = [f for f in os.listdir(caminho_bro) if f.endswith('.labeled')]
    if not arquivos:
        continue

    caminho_arquivo = os.path.join(caminho_bro, arquivos[0])

    try:
            df = pd.read_csv(caminho_arquivo, sep='\t', comment='#', nrows=1000,
engine='python')

        resumo = {
```

```
            'cenário': pasta,
            'n_linhas_amostra': len(df),
            'colunas': list(df.columns),
            'tipos': df.dtypes.to_dict(),
            'valores_nulos': df.isnull().sum().to_dict(),
            'rótulos_label': df['label'].value_counts().to_dict() if 'label' in df else 'N/D',
             'rótulos_detailed': df['detailed-label'].value_counts().to_dict() if 'detailed-label' in df else 'N/D'
        }

        resumos.append(resumo)

    except Exception as e:
        print(f'Erro ao ler {caminho_arquivo}: {e}')
        continue

# Exibe resultados
for r in resumos:
    print(f"\nCenário: {r['cenário']}")
    print(f"Linhas lidas: {r['n_linhas_amostra']}")
    print(f"Tipos de dados:")
    for col, tipo in r['tipos'].items():
        print(f"  - {col}: {tipo}")
    print(f"Valores ausentes por coluna:")
    for col, faltas in r['valores_nulos'].items():
        if faltas > 0:
            print(f"  - {col}: {faltas}")
```

```
print(f"Rótulos 'label': {r['rótulos_label']}")
print(f"Rótulos 'detailed-label': {r['rótulos_detailed']}")
```

**ANEXO 3 - CÓDIGO PARA SELEÇÃO DE DADOS DENTRE OS 23 ARQUIVOS DISPONÍVEIS.**

```
import os
import pandas as pd
from tqdm import tqdm


def processar_arquivo_labeled_streaming(caminho_arquivo, cenario, max_linhas=50000):
    with open(caminho_arquivo, 'r', encoding='utf-8', errors='ignore') as f:
        campos = []
        dados = []
        for linha in f:
            if linha.startswith('#fields'):
                campos = linha.strip().split('\t')[1:]
                continue
            if linha.startswith('#'):
                continue
            partes = linha.strip().split('\t')
            if len(partes) == len(campos):
                dados.append(partes)
            if len(dados) >= max_linhas:
                break

    if not campos or not dados:
        print(f"Nenhum dado válido encontrado em {cenario}")
        return None
```

```
    df = pd.DataFrame(dados, columns=campos)
    df['cenario'] = cenario
    print(f"{cenario} carregado com {len(df)} linhas.")
    return df

# Diretório base
base_dir = '.'

# Lista de DataFrames válidos
dataframes = []

# Iteração
pastas = sorted(os.listdir(base_dir))
for pasta in tqdm(pastas, desc="Processando cenários"):
    caminho_bro = os.path.join(base_dir, pasta, 'bro')
    if not os.path.isdir(caminho_bro):
        continue

    arquivos = [f for f in os.listdir(caminho_bro) if f.endswith('.labeled')]
    if not arquivos:
        print(f"Nenhum arquivo .labeled em {pasta}")
        continue

    caminho_arquivo = os.path.join(caminho_bro, arquivos[0])
    try:
                df = processar_arquivo_labeled_streaming(caminho_arquivo, pasta,
max_linhas=50000)
```

```
        if df is not None:
            dataframes.append(df)
    except Exception as e:
        print(f'Erro ao processar {pasta}: {e}')

# Junta os DataFrames carregados
if dataframes:
    df_final = pd.concat(dataframes, ignore_index=True)
    df_final.to_csv('iot23_amostra_merged.csv', index=False)
        print(f"\nArquivo salvo como 'iot23_amostra_merged.csv' com {len(df_final)} linhas.")
else:
    print("\nNenhum dado foi processado com sucesso.")
```

**ANEXO 4 - COLETA DE DADOS DE TRÁFEGO MALICIOSO**

```
import pandas as pd
import matplotlib.pyplot as plt

# Nome do arquivo CSV
arquivo_csv = "merged.csv"

# Ler apenas as colunas necessárias para otimizar a memória
df = pd.read_csv(arquivo_csv, usecols=["label"])

# Contagem por rótulo
contagem = df["label"].value_counts()

print("Quantidade de conexões por rótulo:")
print(contagem)

# Gráfico de barras
plt.figure(figsize=(6,4))
contagem.plot(kind="bar", color=["#4CAF50", "#F44336"])
plt.title("Conexões por Rótulo (Benigno vs Malicioso)")
plt.xlabel("Rótulo")
plt.ylabel("Quantidade")
plt.xticks(rotation=0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.tight_layout()
plt.show()
```

**ANEXO 5 - DISTRIBUIÇÃO POR CATEGORIAS DE ATAQUE**

```
import re
import pandas as pd
import matplotlib.pyplot as plt

# ======= CONFIG =======
ARQUIVO = "merged.csv"
TOP_N = 15                    # top categorias para o gráfico
# ======================

# Função de normalização de nomes de colunas (para achar "detailed-label" mesmo com variações)
def norm(col):
    s = re.sub(r'[^a-z0-9]+', '_', col.strip().lower())
    s = re.sub(r'_+', '_', s).strip('_')
    return s

# 1) Ler só o cabeçalho para localizar colunas
hdr = pd.read_csv(ARQUIVO, nrows=0)
map_norm2orig = {norm(c): c for c in hdr.columns}

# Detectar nomes reais das colunas de interesse
label_col = map_norm2orig.get("label")
detailed_col = (map_norm2orig.get("detailed_label") or
                map_norm2orig.get("detailed_label_") or
                map_norm2orig.get("detailedlabel"))
```

```
if label_col is None:
    # fallback: qualquer coisa que contenha "label" e não seja detailed-label
        candidatos = [map_norm2orig[k] for k in map_norm2orig if "label" in k and "detailed" not in k]
    label_col = candidatos[0] if candidatos else None

if detailed_col is None:
    # fallback: procurar por algo que contenha "detailed" e "label"
       candidatos = [map_norm2orig[k] for k in map_norm2orig if "detailed" in k and "label" in k]
    detailed_col = candidatos[0] if candidatos else None

if not label_col or not detailed_col:
     raise ValueError(f"Não encontrei colunas de rótulo. Detectado - label: {label_col}, detailed-label: {detailed_col}")

# 2) Ler somente as colunas necessárias
df = pd.read_csv(ARQUIVO, usecols=[label_col, detailed_col])

# 3) Padronizar o label (Benigno/Malicioso)
df[label_col] = df[label_col].astype(str).str.strip().str.lower()
df[label_col] = df[label_col].replace({"benigno": "Benigno", "malicioso": "Malicioso"})

# 4) Filtrar apenas conexões maliciosas
mal = df[df[label_col] == "Malicioso"].copy()

# 5) Contagem por categoria (detailed-label)
```

```
contagem = mal[detailed_col].astype(str).str.strip().replace({"": "N/
D"}).value_counts(dropna=False)
total_mal = int(mal.shape[0])
percentual = (contagem / total_mal * 100).round(2)

# 6) Montar tabela final e salvar
tabela = pd.DataFrame({
    "categoria": contagem.index,
    "quantidade": contagem.values,
    "percentual_%": percentual.values
})
tabela.to_csv("dist_categorias_ataque.csv", index=False)

print("Distribuição das categorias (maliciosas):")
print(tabela.head(30))  # mostra as 30 primeiras no console
print(f"\nTotal de conexões maliciosas consideradas: {total_mal}")
print("Resultado completo salvo em: dist_categorias_ataque.csv")

# 7) Gráfico (Top N)
top = tabela.head(TOP_N).set_index("categoria")
plt.figure(figsize=(10, 5))
top["quantidade"].plot(kind="bar")
plt.title(f"Top {TOP_N} categorias de ataque (Malicioso)")
plt.xlabel("Categoria")
plt.ylabel("Quantidade")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y", linestyle="--", alpha=0.6)
```

```
plt.tight_layout()
plt.savefig("dist_categorias_ataque.png", dpi=120)
plt.show()

print("Gráfico salvo como: dist_categorias_ataque.png")
```

**ANEXO 6 - ESTATÍSTICAS GERAIS E POR LABEL**

```
import re
import pandas as pd

# ================== CONFIG ==================
ARQUIVO = "merged.csv"
METRICAS = ["duration", "orig_bytes", "resp_bytes", "orig_pkts", "resp_pkts"]
# ============================================

def norm(s: str) -> str:
    s = re.sub(r'[^a-z0-9]+', '_', s.strip().lower())
    return re.sub(r'_+', '_', s).strip('_')

# 1) Detectar nomes reais das colunas de interesse
hdr = pd.read_csv(ARQUIVO, nrows=0)
norm2orig = {norm(c): c for c in hdr.columns}

# localizar coluna de rótulo
label_col = (
    norm2orig.get("label")
    or next((norm2orig[k] for k in norm2orig if "label" in k and "detailed" not in k), None)
)

if label_col is None:
        raise ValueError("Não encontrei a coluna de rótulo (label) no CSV. Verifique o cabeçalho.")
```

```
# garantir que métricas existem no arquivo
metricas_presentes = [c for c in METRICAS if c in hdr.columns]
if not metricas_presentes:
    raise ValueError(f"Nenhuma das métricas esperadas {METRICAS} foi encontrada no CSV.")

usecols = [label_col] + metricas_presentes

print(" Lendo colunas:", usecols)
df = pd.read_csv(ARQUIVO, usecols=usecols, low_memory=False)

# 2) Padronizar rótulos (Benigno/Malicioso)
df[label_col] = (
    df[label_col]
    .astype(str)
    .str.strip()
    .str.lower()
    .replace({"Benigno": "Benigno", "Malicioso": "Malicioso"})
)

# 3) Estatísticas descritivas gerais
def stats_basicas(frame: pd.DataFrame, cols: list[str]) -> pd.DataFrame:
    base = frame[cols]
    desc = base.describe().T  # count, mean, std, min, 25%, 50%, 75%, max
    desc = desc.rename(columns={"50%": "median"})
    # % ausentes
    pct_nan = 100 * (1 - (desc["count"] / len(frame))) if len(frame) else 0
```

```
    out = desc[["count", "mean", "median", "std", "min", "max"]].copy()
    out["pct_missing_%"] = pct_nan.round(2)
    return out

estat_geral = stats_basicas(df, metricas_presentes)
estat_geral.to_csv("estatisticas_trafego_geral.csv")
print("\n Estatísticas gerais (amostra):")
print(estat_geral.head())

# 4) Estatísticas por rótulo
estat_por_rotulo = (
    df.groupby(label_col)[metricas_presentes]
      .apply(lambda x: stats_basicas(x, metricas_presentes))
)

# ajeitar o índice para salvar bonito
estat_por_rotulo = estat_por_rotulo.reset_index(level=0).rename(columns={label_col:
"label"})
estat_por_rotulo.to_csv("estatisticas_trafego_por_label.csv", index=False)

print("\n Estatísticas por rótulo (amostra):")
print(estat_por_rotulo.head(10))

print("\n Arquivos salvos:")
print(" - estatisticas_trafego_geral.csv")
print(" - estatisticas_trafego_por_label.csv")
```

**ANEXO 7 - CÓDIGO PARA ANÁLISE DA FREQUÊNCIA DOS PROTOCOLOS E SERVIÇOS**

```
import re
import pandas as pd
import matplotlib.pyplot as plt

# ============= CONFIG =============
ARQUIVO = "merged.csv"
TOP_N = 12              # quantos itens mostrar no gráfico
SALVAR_GRAFICOS = True
# ==================================

def norm(s: str) -> str:
    s = re.sub(r'[^a-z0-9]+', '_', s.strip().lower())
    return re.sub(r'_+', '_', s).strip('_')

# 1) Detectar colunas reais pelo cabeçalho
hdr = pd.read_csv(ARQUIVO, nrows=0)
norm2orig = {norm(c): c for c in hdr.columns}

label_col   = (norm2orig.get("label")
               or next((norm2orig[k] for k in norm2orig if "label" in k and "detailed" not in k),
None))
proto_col   = norm2orig.get("proto")
service_col = norm2orig.get("service")

if not proto_col:
```

```
    raise ValueError("Coluna 'proto' não encontrada no CSV.")
if not service_col:
    raise ValueError("Coluna 'service' não encontrada no CSV.")
if not label_col:
    # Se realmente não houver rótulo, ainda dá para fazer a frequência geral
     print("  Coluna de rótulo não encontrada; executando apenas frequências gerais (sem separação Benigno/Malicioso).")

usecols = [c for c in [label_col, proto_col, service_col] if c is not None]
print(" Lendo colunas:", usecols)
df = pd.read_csv(ARQUIVO, usecols=usecols, low_memory=False)

# 2) Padronizar rótulo (se existir)
if label_col:
    df[label_col] = (df[label_col]
                     .astype(str)
                     .str.strip()
                     .str.lower()
                     .replace({"benigno": "Benigno", "malicioso": "Malicioso"}))

# 3) Higiene nos campos categóricos
df[proto_col]   = df[proto_col].astype(str).str.strip().replace({"": "N/D", "nan": "N/D"})
df[service_col] = df[service_col].astype(str).str.strip().replace({"": "N/D", "-": "N/D", "nan": "N/D"})

# --------- FUNÇÕES AUXILIARES ---------
def salvar_tabela_e_plot(series_counts: pd.Series, titulo: str, nome_base: str):
```

```
    """Salva CSV com contagem e % e gera gráfico de barras com Top N."""
    total = int(series_counts.sum())
    tabela = pd.DataFrame({
        "valor": series_counts.index,
        "quantidade": series_counts.values,
        "percentual_%": (series_counts / total * 100).round(2)
    })
    csv_path = f"{nome_base}.csv"
    tabela.to_csv(csv_path, index=False)
    print(f" Tabela salva: {csv_path}")
    print(tabela.head(20))

    # Gráfico Top N
    top = tabela.head(TOP_N).set_index("valor")
    plt.figure(figsize=(10, 5))
    top["quantidade"].plot(kind="bar")
    plt.title(titulo)
    plt.xlabel("")
    plt.ylabel("Quantidade")
    plt.xticks(rotation=45, ha="right")
    plt.grid(axis="y", linestyle="--", alpha=0.5)
    plt.tight_layout()
    if SALVAR_GRAFICOS:
        img_path = f"{nome_base}.png"
        plt.savefig(img_path, dpi=120)
        print(f" Gráfico salvo: {img_path}")
    plt.show()
```

```
# --------- 4) FREQUÊNCIAS GERAIS ---------
print("\n=== FREQUÊNCIA DE PROTOCOLOS (GERAL) ===")
proto_counts = df[proto_col].value_counts(dropna=False)
salvar_tabela_e_plot(proto_counts, "Protocolos (Geral) – Top", "freq_proto_geral")

print("\n=== FREQUÊNCIA DE SERVIÇOS (GERAL) ===")
service_counts = df[service_col].value_counts(dropna=False)
salvar_tabela_e_plot(service_counts, "Serviços (Geral) – Top", "freq_service_geral")

# --------- 5) FREQUÊNCIAS POR RÓTULO ---------
if label_col:
    for rotulo, grupo in df.groupby(label_col):
        tag = norm(str(rotulo)) or "nd"
        print(f"\n=== PROTOCOLOS – {rotulo} ===")
        proto_lbl = grupo[proto_col].value_counts(dropna=False)
                        salvar_tabela_e_plot(proto_lbl,  f"Protocolos  –  {rotulo}  –  Top",
f"freq_proto_{tag}")

        print(f"\n=== SERVIÇOS – {rotulo} ===")
        serv_lbl = grupo[service_col].value_counts(dropna=False)
        salvar_tabela_e_plot(serv_lbl, f"Serviços – {rotulo} – Top", f"freq_service_{tag}")
```

**ANEXO 8 - CÓDIGO PARA ANÁLISE DA HORA DE ACESSOS MALICIOSOS**

```
import pandas as pd
import matplotlib.pyplot as plt

# === Configurações ===
ARQUIVO = "merged.csv"

# 1. Carregar dados
df = pd.read_csv(ARQUIVO)

# 2. Garantir que a coluna 'label' existe e filtrar apenas maliciosos
if 'label' not in df.columns:
    raise ValueError("A coluna 'label' não foi encontrada no CSV.")

df_mal = df[df['label'].str.lower() == 'malicious'].copy()

# 3. Converter 'ts' (timestamp) para datetime
df_mal['datetime'] = pd.to_datetime(df_mal['ts'], unit='s')

# 4. Criar colunas auxiliares
df_mal['date'] = df_mal['datetime'].dt.date
df_mal['hour'] = df_mal['datetime'].dt.hour

# === Análise por dia ===
por_dia = df_mal.groupby('date').size()

plt.figure(figsize=(12, 5))
```

```
por_dia.plot(kind='line', marker='o')
plt.title("Atividade maliciosa por dia")
plt.xlabel("Data")
plt.ylabel("Número de conexões")
plt.grid(True)
plt.tight_layout()
plt.savefig("atividade_maliciosa_por_dia.png")
plt.close()

# === Análise por hora ===
por_hora = df_mal.groupby('hour').size()

plt.figure(figsize=(10, 5))
por_hora.plot(kind='bar', color='red')
plt.title("Distribuição horária da atividade maliciosa")
plt.xlabel("Hora do dia")
plt.ylabel("Número de conexões")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.savefig("atividade_maliciosa_por_hora.png")
plt.close()

print("Gráficos salvos: 'atividade_maliciosa_por_dia.png' e 'atividade_maliciosa_por_hora.png'")
```

**ANEXO 8 - DATA E HORA DOS ATAQUES**

```
import pandas as pd
import matplotlib.pyplot as plt

# === Configurações ===
ARQUIVO = "merged.csv"

# 1. Carregar dados
df = pd.read_csv(ARQUIVO)

# 2. Garantir que a coluna 'label' existe e filtrar apenas maliciosos
if 'label' not in df.columns:
    raise ValueError("A coluna 'label' não foi encontrada no CSV.")

df_mal = df[df['label'].str.lower() == 'malicious'].copy()

# 3. Converter 'ts' (timestamp) para datetime
df_mal['datetime'] = pd.to_datetime(df_mal['ts'], unit='s')

# 4. Criar colunas auxiliares
df_mal['date'] = df_mal['datetime'].dt.date
df_mal['hour'] = df_mal['datetime'].dt.hour

# === Análise por dia ===
por_dia = df_mal.groupby('date').size()

plt.figure(figsize=(12, 5))
```

```
por_dia.plot(kind='line', marker='o')
plt.title("Atividade maliciosa por dia")
plt.xlabel("Data")
plt.ylabel("Número de conexões")
plt.grid(True)
plt.tight_layout()
plt.savefig("atividade_maliciosa_por_dia.png")
plt.close()

# === Análise por hora ===
por_hora = df_mal.groupby('hour').size()

plt.figure(figsize=(10, 5))
por_hora.plot(kind='bar', color='red')
plt.title("Distribuição horária da atividade maliciosa")
plt.xlabel("Hora do dia")
plt.ylabel("Número de conexões")
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.tight_layout()
plt.savefig("atividade_maliciosa_por_hora.png")
plt.close()

print("Gráficos salvos: 'atividade_maliciosa_por_dia.png' e 'atividade_maliciosa_por_hora.png'")
```

## ANEXO 9 - ALGORITMO DE APRENDIZADO DE MÁQUINA

```
# -*- coding: utf-8 -*-
"""
Pipeline IoT-23 (Benign vs Malicious)
- Leitura robusta do CSV (auto-deteção de colunas)
- Higienização de numéricas e categóricas
- Split estratificado
- Undersampling (apenas no treino)
- Modelos: RandomForest e HistGradientBoosting
- Métricas e gráficos
"""

import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, classification_report, confusion_matrix,
    ConfusionMatrixDisplay
)
```

```
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier

# ====================== CONFIG ======================
ARQUIVO = "merged.csv"
TEST_SIZE = 0.30
RANDOM_STATE = 42
UNDERSAMPLE_RATIO = 1.0          # 1.0 => Malicious ~ Benign no treino; 2.0 => Malicious ~ 2x Benign
SALVAR_PREDICOES = True
MOSTRAR_GRAFICOS = True
# ====================================================

def norm(s: str) -> str:
    s = re.sub(r'[^a-z0-9]+', '_', str(s).strip().lower())
    return re.sub(r'_+', '_', s).strip('_')

# 1) Ler cabeçalho e mapear nomes
hdr = pd.read_csv(ARQUIVO, nrows=0, low_memory=False)
norm2orig = {norm(c): c for c in hdr.columns}

# localizar colunas de interesse
label_col = (
    norm2orig.get("label")
    or next((norm2orig[k] for k in norm2orig if "label" in k and "detailed" not in k), None)
)
if label_col is None:
```

```
    raise ValueError("Não encontrei a coluna de rótulo (label). Corrija o CSV e tente novamente.")

# candidatos
num_cands = [
    "duration","orig_bytes","resp_bytes","orig_pkts","resp_pkts",
    "id.orig_p","id.resp_p","missed_bytes","orig_ip_bytes","resp_ip_bytes"
]
cat_cands = ["proto","service","conn_state","local_orig","local_resp","history"]

present_nums = [c for c in num_cands if c in hdr.columns]
present_cats = [c for c in cat_cands if c in hdr.columns]

usecols = [label_col] + present_nums + present_cats
print("Lendo colunas:", usecols)
df = pd.read_csv(ARQUIVO, usecols=usecols, low_memory=False)

# 2) Padronizar rótulos
df[label_col] = (
    df[label_col].astype(str).str.strip().str.lower()
    .replace({"benign":"Benign","malicious":"Malicious"})
)
df = df[df[label_col].isin(["Benign","Malicious"])].copy()
print(f"Amostras após limpeza de rótulo: {len(df)}")

# 3) Higiene: numéricas -> float; categóricas -> strings limpas
if present_nums:
```

```
    for c in present_nums:
        df[c] = pd.to_numeric(df[c], errors="coerce")
    df[present_nums] = df[present_nums].fillna(0)

if present_cats:
    for c in present_cats:
        df[c] = (
            df[c].astype(str).str.strip()
            .replace({"": "N/D", "-": "N/D", "nan": "N/D", "None": "N/D"})
        )

# 4) Definir X e y
X = df[present_nums + present_cats].copy()
y = df[label_col].copy()

# 5) Split estratificado
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=TEST_SIZE, random_state=RANDOM_STATE, stratify=y
)

# 6) Undersampling da majoritária (Malicious) APENAS no treino
def undersample_train(X_tr: pd.DataFrame, y_tr: pd.Series, ratio=1.0,
random_state=RANDOM_STATE):
    tr = X_tr.copy()
    tr["_label_tmp_"] = y_tr.values  # índice alinhado
    benign = tr[tr["_label_tmp_"] == "Benign"]
    mal    = tr[tr["_label_tmp_"] == "Malicious"]
```

```
    if len(benign) == 0 or len(mal) == 0:
        print("Uma das classes está vazia no treino; pulando undersampling.")
        return X_tr, y_tr
    alvo_mal = min(int(len(benign) * ratio), len(mal))
    mal_sampled = mal.sample(n=alvo_mal, random_state=random_state)
    bal = pd.concat([benign, mal_sampled], axis=0).sample(frac=1, random_state=random_state)
    y_bal = bal["_label_tmp_"].copy()
    X_bal = bal.drop(columns=["_label_tmp_"]).copy()
    print(f"⚖️ Undersampling no treino: Benign={len(benign)}, "
          f"Malicious(orig)={len(mal)}, Malicious(amostrado)={alvo_mal}, Total={len(bal)}")
    return X_bal, y_bal

X_train_bal, y_train_bal = undersample_train(X_train, y_train, UNDERSAMPLE_RATIO)

# 7) Pré-processador (One-Hot nas categóricas)
num_transform = "passthrough"  # árvores/boosting não exigem scaling
cat_transform = OneHotEncoder(handle_unknown="ignore", sparse_output=False)

preprocess = ColumnTransformer(
    transformers=[
        ("num", num_transform, present_nums),
        ("cat", cat_transform, present_cats)
    ],
    remainder="drop"
```

```
)

# 8) Modelos
rf_pipe = Pipeline(steps=[
    ("prep", preprocess),
    ("clf", RandomForestClassifier(
        n_estimators=200, n_jobs=-1, random_state=RANDOM_STATE
    ))
])

hgb_pipe = Pipeline(steps=[
    ("prep", preprocess),
    ("clf", HistGradientBoostingClassifier(
        random_state=RANDOM_STATE, max_iter=300, learning_rate=0.1
    ))
])

# 9) Avaliação comum
def avaliar(model, nome:str):
    model.fit(X_train_bal, y_train_bal)
    y_pred = model.predict(X_test)

    # probabilidades para ROC-AUC (se disponíveis)
    y_proba = None
    try:
        y_proba = model.predict_proba(X_test)[:, 1]
    except Exception:
```

```
    try:
        y_proba = model.decision_function(X_test)
    except Exception:
        pass

pos = "Malicious"
acc  = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, pos_label=pos)
rec  = recall_score(y_test, y_pred, pos_label=pos)
f1   = f1_score(y_test, y_pred, pos_label=pos)
if y_proba is not None:
    roc = roc_auc_score((y_test==pos).astype(int), y_proba)
else:
    roc = roc_auc_score((y_test==pos).astype(int), (y_pred==pos).astype(int))

print("\n"+("="*70))
print(f"Modelo: {nome}")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1-Score : {f1:.4f}")
print(f"ROC-AUC  : {roc:.4f}")
print("\nClassification report:")
print(classification_report(y_test, y_pred, digits=4))

# Matriz de confusão
cm = confusion_matrix(y_test, y_pred, labels=["Benign","Malicious"])
```

```
    disp = ConfusionMatrixDisplay(cm, display_labels=["Benign","Malicious"])
    if MOSTRAR_GRAFICOS:
        plt.figure(figsize=(4.5,4))
        disp.plot(values_format="d", cmap="Blues", colorbar=False)
        plt.title(f"Matriz de Confusão – {nome}")
        plt.tight_layout()
        plt.show()

    # Importância de features (RF)
    if nome.startswith("RandomForest"):
        prep = model.named_steps["prep"]
        clf  = model.named_steps["clf"]
        # nomes após One-Hot
        cat_names = []
        if present_cats:
            cat_encoder = prep.named_transformers_["cat"]
            cat_names = list(cat_encoder.get_feature_names_out(present_cats))
        feat_names = present_nums + cat_names
                        importancias = pd.Series(clf.feature_importances_,
index=feat_names).sort_values(ascending=False)
        print("\n Top 20 importâncias de features (RF):")
        print(importancias.head(20))
        if MOSTRAR_GRAFICOS:
            importancias.head(25).plot(kind="bar", figsize=(10,4))
            plt.title("Top importâncias de features – RandomForest")
            plt.tight_layout()
            plt.show()
```

```
    # Exportar predições
    if SALVAR_PREDICOES:
        out = pd.DataFrame({"y_true": y_test, "y_pred": y_pred})
        if y_proba is not None:
            out["score_malicious"] = y_proba
        fname = f"predicoes_{nome.replace(' ','_').lower()}.csv"
        out.to_csv(fname, index=False)
        print(f" Predições salvas em: {fname}")

# 10) Rodar avaliações
avaliar(rf_pipe,  "RandomForest")
avaliar(hgb_pipe, "HistGradientBoosting")
```

**ANEXO 10 - ALGORITMO PARA HIGIENIZAÇÃO DOS DADOS**

```
import pandas as pd
import json

CSV_IN   = "merged.csv"
CSV_OUT  = "merged_clean.csv"
PAR_OUT  = "merged_clean.parquet"
DD_JSON  = "dropdowns.json"

COLS_NUM = [
    "duration","orig_bytes","resp_bytes","orig_pkts","resp_pkts",
    "id.orig_p","id.resp_p","missed_bytes","orig_ip_bytes","resp_ip_bytes"
]
COLS_CAT = ["proto","service","conn_state","local_orig","local_resp","history"]

print("Lendo CSV bruto...")
df = pd.read_csv(CSV_IN, low_memory=False)

# Padroniza rótulo, se existir
if "label" in df.columns:
    df["label"] = (
        df["label"].astype(str).str.strip().str.lower()
        .replace({"benign":"Benign","malicious":"Malicious"})
    )

# Coage numéricas → float (ou int quando fizer sentido), preenche NaN com 0
for c in COLS_NUM:
```

```
    if c in df.columns:
        df[c] = pd.to_numeric(df[c], errors="coerce").fillna(0)

# Higiene categóricas
for c in COLS_CAT:
    if c in df.columns:
        df[c] = (
            df[c].astype(str).str.strip()
            .replace({"": "N/D", "-": "N/D", "nan": "N/D", "None": "N/D"})
        )

# (Opcional) downcast para reduzir memória
for c in ["id.orig_p","id.resp_p","orig_pkts","resp_pkts","orig_bytes","resp_bytes",
          "orig_ip_bytes","resp_ip_bytes","missed_bytes"]:
    if c in df.columns:
        try:
            df[c] = pd.to_numeric(df[c], downcast="integer")
        except Exception:
            pass
if "duration" in df.columns:
    df["duration"] = pd.to_numeric(df["duration"], downcast="float")

# Gera listas “top” para combos do front
def top_list(col, n=20, fallback=None):
    if col in df.columns:
        vals = (df[col].dropna().astype(str)
                .replace({"": "N/D","-":"N/D"})
```

```
            .value_counts().head(n).index.tolist())
        vals = ["N/D"] + [v for v in vals if v != "N/D"]
        return vals
    return fallback or ["N/D"]

dropdowns = {
    "proto": top_list("proto", 10, ["tcp","udp","icmp","N/D"]),
    "service": top_list("service", 15, ["N/D","dns","http","ssh"]),
    "conn_state": top_list("conn_state", 15, ["S0","SF","REJ","OTH","N/D"])
}
with open(DD_JSON, "w", encoding="utf-8") as f:
    json.dump(dropdowns, f, ensure_ascii=False, indent=2)

# Salva arquivos limpos
print(" Salvando CSV limpo...")
df.to_csv(CSV_OUT, index=False)
print(" Salvando Parquet limpo...")
df.to_parquet(PAR_OUT, index=False)
print(f" Pronto: {CSV_OUT}, {PAR_OUT}, {DD_JSON}")
```

**ANEXO 11 - ALGORITMO UTILIZADO PARA TREINAR O MODELO**

```
# treinar_modelo.py (compatível com app.py e dados limpos)
import json, os, platform
import numpy as np
import pandas as pd
import joblib

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
import sklearn, pandas, numpy

# ---------- Config ----------
PARQUET = "merged_clean.parquet"   # preferencial (rápido)
CSV     = "merged_clean.csv"       # fallback
MODEL   = "modelo_rf.joblib"
META    = "modelo_meta.json"

COLS_NUM = [
    "duration","orig_bytes","resp_bytes","orig_pkts","resp_pkts",
    "id.orig_p","id.resp_p","missed_bytes","orig_ip_bytes","resp_ip_bytes"
]
COLS_CAT = ["proto","service","conn_state","local_orig","local_resp","history"]
FEATURE_ORDER = COLS_NUM + COLS_CAT
TARGET = "label"
```

```
UNDERSAMPLE_RATIO = 1.0  # 1:1
RANDOM_STATE = 42
# ----------------------------

# 1) Carregar dados limpos
if os.path.exists(PARQUET):
    df = pd.read_parquet(PARQUET)
elif os.path.exists(CSV):
    df = pd.read_csv(CSV, low_memory=False)
else:
        raise FileNotFoundError("Nenhum arquivo limpo encontrado (merged_clean.parquet ou merged_clean.csv).")

# 2) Verificações básicas
if TARGET not in df.columns:
    raise ValueError(f"Coluna de rótulo '{TARGET}' não encontrada nos dados limpos.")
faltantes = [c for c in FEATURE_ORDER if c not in df.columns]
if faltantes:
    raise ValueError(f"As seguintes features esperadas não existem nos dados limpos: {faltantes}")

# 3) Garantir tipos (de novo) — só por segurança
for c in COLS_NUM:
    df[c] = pd.to_numeric(df[c], errors="coerce").fillna(0)
for c in COLS_CAT:
    df[c] = (df[c].astype(str).str.strip()
          .replace({"": "N/D", "-": "N/D", "nan": "N/D", "None": "N/D"}))
```

```
df[TARGET] = (df[TARGET].astype(str).str.strip().str.lower()
              .replace({"benign":"Benign","malicious":"Malicious"}))
df = df[df[TARGET].isin(["Benign","Malicious"])].copy()

# 4) Split estratificado
X = df[FEATURE_ORDER].copy()
y = df[TARGET].copy()
X_tr, X_te, y_tr, y_te = train_test_split(
    X, y, test_size=0.30, random_state=RANDOM_STATE, stratify=y
)

# 5) Undersampling (classe majoritária: Malicious) — apenas no treino
ben = y_tr[y_tr=="Benign"].index
mal = y_tr[y_tr=="Malicious"].index
n_mal = min(len(mal), int(len(ben) * UNDERSAMPLE_RATIO))
rng = np.random.RandomState(RANDOM_STATE)
mal_s = rng.choice(mal, size=n_mal, replace=False)
idx_bal = ben.union(mal_s)

X_trb = X_tr.loc[idx_bal]
y_trb = y_tr.loc[idx_bal]
print(f"Undersampling no treino: Benign={sum(y_trb=='Benign')},
Malicious={sum(y_trb=='Malicious')}")

# 6) Pipeline (prep + RF)
pre = ColumnTransformer(
    transformers=[
```

```
        ("num", "passthrough", COLS_NUM),
                ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False),
COLS_CAT),
    ],
    remainder="drop"
)
rf = RandomForestClassifier(n_estimators=200, n_jobs=-1,
random_state=RANDOM_STATE)
pipe = Pipeline([("prep", pre), ("clf", rf)])

print("Treinando modelo...")
pipe.fit(X_trb, y_trb)

# 7) Auto-teste: tentar prever 1 amostra do conjunto de teste
print("Testando predição em 1 amostra...")
sample = X_te.iloc[[0]].copy()
pred = pipe.predict(sample)[0]
print("OK. Predição de exemplo:", pred)

# 8) Salvar
joblib.dump(pipe, MODEL, compress=3)
print(f"Modelo salvo em: {MODEL}")

meta = {
    "python": platform.python_version(),
    "sklearn": sklearn.__version__,
    "pandas": pandas.__version__,
```

```
    "numpy": numpy.__version__,
    "features": FEATURE_ORDER
}
with open(META, "w", encoding="utf-8") as f:
    json.dump(meta, f, ensure_ascii=False, indent=2)
print(f"Metadados salvos em: {META}")
```

**ANEXO 12 - BACKEND UTILIZADO**

```
# app.py
# Flask 3.x – Dashboard + Predição usando dados limpos (Parquet/CSV) e dropdowns.json
from flask import Flask, render_template, request, jsonify
import os, json
import pandas as pd
import joblib
from datetime import datetime
from flask import request

# ================== CONFIG ==================
PARQUET_PATH = "merged_clean.parquet"   # preferencial (rápido)
CSV_PATH     = "merged_clean.csv"       # fallback
DROPDOWNS    = "dropdowns.json"
MODEL_PATH   = "modelo_rf.joblib"       # pipeline salvo (opcional)
# Features usadas no treino (mantenha em sincronia com o pipeline salvo)
COLS_NUM = [
    "duration","orig_bytes","resp_bytes","orig_pkts","resp_pkts",
    "id.orig_p","id.resp_p","missed_bytes","orig_ip_bytes","resp_ip_bytes"
]
COLS_CAT = ["proto","service","conn_state","local_orig","local_resp","history"]
FEATURE_ORDER = COLS_NUM + COLS_CAT
# ============================================

app = Flask(__name__)
```

```
# ================== CARGA INICIAL ==================
# Dados
if os.path.exists(PARQUET_PATH):
    df = pd.read_parquet(PARQUET_PATH)
elif os.path.exists(CSV_PATH):
    df = pd.read_csv(CSV_PATH, low_memory=False)
else:
        raise FileNotFoundError("Nenhum arquivo de dados encontrado (merged_clean.parquet ou merged_clean.csv).")

# Dropdowns
if os.path.exists(DROPDOWNS):
    with open(DROPDOWNS, "r", encoding="utf-8") as f:
        dd = json.load(f)
else:
    # fallback mínimo caso JSON não exista
    dd = {
        "proto": ["N/D","tcp","udp","icmp"],
        "service": ["N/D","dns","http","ssh"],
        "conn_state": ["N/D","S0","SF","REJ","OTH"]
    }

# Modelo (opcional)
modelo = None
if os.path.exists(MODEL_PATH):
    try:
        modelo = joblib.load(MODEL_PATH)
```

```
        print("Modelo carregado:", MODEL_PATH)
    except Exception as e:
        print(f"Falha ao carregar {MODEL_PATH}: {e}")
else:
    print("Modelo não encontrado; predição ficará desabilitada até incluir o .joblib.")

# ================== ROTAS ==================
@app.get("/saude")
def saude():
    return jsonify(status="ok")

@app.get("/")
def index():
    total = len(df)

    por_label = {}
    if "label" in df.columns:
        por_label = df["label"].value_counts(dropna=False).to_dict()

    top_proto = {}
    if "proto" in df.columns:
        top_proto = df["proto"].value_counts().head(10).to_dict()

    top_service = {}
    if "service" in df.columns:
        top_service = df["service"].value_counts().head(10).to_dict()
```

```
    return render_template(
        "index.html",
        total=total,
        por_label=por_label,
        top_proto=top_proto,
        top_service=top_service,
        modelo_pronto=(modelo is not None)
    )

@app.route("/prever", methods=["GET", "POST"])
def prever():
    """
    GET  -> renderiza formulário com combos de dropdowns.json
    POST -> recebe form, monta vetor de features e aplica o modelo (se houver)
    """
    protos = dd.get("proto", ["N/D"])
    servs  = dd.get("service", ["N/D"])
    states = dd.get("conn_state", ["N/D"])

    resultado = None
    score = None

    if request.method == "POST":
        # 1) coletar dados do formulário
        form = request.form

        # 2) montar dicionário com TODAS as features esperadas
```

```
        x = {}
        # numéricos
        for c in COLS_NUM:
            val = form.get(c, 0)
            try:
                x[c] = float(val)
            except Exception:
                x[c] = 0.0
        # categóricos
        for c in COLS_CAT:
            val = str(form.get(c, "N/D")).strip()
            x[c] = "N/D" if val in ("", "-", "nan", "None") else val

        X = pd.DataFrame([x], columns=FEATURE_ORDER)

        # 3) predição (se houver modelo)
        if modelo is None:
                resultado = "Modelo não carregado. Adicione 'modelo_rf.joblib' e reinicie o
servidor."
        else:
            try:
                y_pred = modelo.predict(X)[0]
                resultado = str(y_pred)
                # probabilidade "Malicious", se disponível
                try:
                    proba = modelo.predict_proba(X)[0]
                    classes_ = None
```

```
                if hasattr(modelo, "classes_"):
                    classes_ = modelo.classes_
                elif hasattr(modelo, "named_steps") and "clf" in modelo.named_steps:
                    classes_ = modelo.named_steps["clf"].classes_
                if classes_ is not None and "Malicious" in list(classes_):
                    idx = list(classes_).index("Malicious")
                    score = float(proba[idx])
            except Exception:
                score = None
        except Exception as e:
            resultado = f"Erro ao prever: {e}"

    return render_template(
        "prever.html",
        modelo_pronto=(modelo is not None),
        protos=protos, servs=servs, states=states,
        resultado=(f"{resultado} (score Malicious: {score:.4f})" if resultado and score is
not None else resultado)
    )

def _options(col, top=30):
    if col in df.columns:
        vals = (df[col].dropna().astype(str)
                .replace({"": "N/D", "-": "N/D"})
                .value_counts().head(top).index.tolist())
        return ["(qualquer)"] + [v for v in vals]
    return ["(qualquer)"]
```

```
@app.get("/visualizar_dados")
def visualizar_dados():
    """
    Página de visualização com filtros e gráficos interativos.
    As opções dos combos são obtidas do próprio df limpo já carregado.
    """
    labels = _options("label", top=5)
    protos = _options("proto", top=15)
    servs  = _options("service", top=20)
    has_ts = "ts" in df.columns  # se houver timestamp, habilitamos filtro por data

    return render_template(
        "visualizar_dados.html",
        labels=labels, protos=protos, servs=servs, has_ts=has_ts
    )

@app.get("/api/dados")
def api_dados():
    """
    Endpoint JSON para retornar agregados conforme filtros:
      - label: "(qualquer)" | "Benign" | "Malicious"
      - proto: "(qualquer)" | ...
      - service: "(qualquer)" | ...
      - data_ini, data_fim: "YYYY-MM-DD" (se houver ts)
      Retorna: totais, counts de label, proto, service e série temporal diária (se ts existir).
```

```
"""
f_label  = request.args.get("label", "(qualquer)")
f_proto  = request.args.get("proto", "(qualquer)")
f_serv   = request.args.get("service", "(qualquer)")
data_ini = request.args.get("data_ini")
data_fim = request.args.get("data_fim")

dff = df.copy()

# filtros categóricos
if f_label and f_label != "(qualquer)" and "label" in dff.columns:
    dff = dff[dff["label"] == f_label]
if f_proto and f_proto != "(qualquer)" and "proto" in dff.columns:
    dff = dff[dff["proto"] == f_proto]
if f_serv and f_serv != "(qualquer)" and "service" in dff.columns:
    dff = dff[dff["service"] == f_serv]

# filtro temporal (se ts existir)
ts_ok = "ts" in dff.columns
if ts_ok:
    # cria coluna de data só se necessário
    if "date" not in dff.columns:
        try:
            dff["date"] = pd.to_datetime(dff["ts"], unit="s").dt.date
        except Exception:
            ts_ok = False  # se não der, desabilita série temporal
```

```
    def _parse(d):
        try:
            return datetime.strptime(d, "%Y-%m-%d").date()
        except Exception:
            return None

    di = _parse(data_ini) if data_ini else None
    dfim = _parse(data_fim) if data_fim else None
    if ts_ok and (di or dfim):
        if di:
            dff = dff[dff["date"] >= di]
        if dfim:
            dff = dff[dff["date"] <= dfim]

# agregados
total = int(len(dff))

label_counts = {}
if "label" in dff.columns:
    label_counts = dff["label"].value_counts(dropna=False).to_dict()

proto_counts = {}
if "proto" in dff.columns:
    vc = dff["proto"].value_counts(dropna=False).head(12)
    proto_counts = {str(k): int(v) for k, v in vc.items()}

service_counts = {}
```

```
    if "service" in dff.columns:
        vc = dff["service"].value_counts(dropna=False).head(12)
        service_counts = {str(k): int(v) for k, v in vc.items()}

    serie_diaria = []
    if ts_ok and "date" in dff.columns:
        s = dff.groupby("date").size().sort_index()
        serie_diaria = [{"data": str(k), "qtd": int(v)} for k, v in s.items()]

    return jsonify({
        "total": total,
        "label_counts": label_counts,
        "proto_counts": proto_counts,
        "service_counts": service_counts,
        "serie_diaria": serie_diaria
    })

# ================== MAIN ==================
if __name__ == "__main__":
    # Rode:  python app.py
    app.run(host="127.0.0.1", port=5000, debug=True)
```

**ANEXO 13 - INDEX.HTML**

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Painel IoT</title>
  <!-- Bootstrap -->
   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet"/>
  <!-- CSS personalizado -->
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body class="bg-light">

  <!-- NAVBAR -->
  <nav class="navbar navbar-expand-lg">
    <div class="container">
      <a class="navbar-brand" href="{{ url_for('index') }}">Painel IoT</a>
      <div class="ms-auto">
        {% if modelo_pronto %}
          <span class="badge text-bg-success">Modelo carregado</span>
        {% else %}
          <span class="badge text-bg-secondary">Modelo indisponível</span>
        {% endif %}
      </div>
    </div>
```

```
</nav>

<!-- CONTEÚDO PRINCIPAL -->
<div class="container mt-4">
  <div class="d-flex justify-content-between align-items-center mb-3">
    <h1 class="mb-0">Bem-vindo ao Painel IoT</h1>
    <div class="btn-group">
      <a href="{{ url_for('visualizar_dados') }}" class="btn btn-primary">📊 Visualizar Dados</a>
      <a href="{{ url_for('prever') }}" class="btn btn-outline-secondary">🤖 Fazer Previsão</a>
    </div>
  </div>
    <p class="lead">Resumo do conjunto de dados e atalhos para exploração e predição.</p>

  <!-- CARDS -->
  <div class="row g-3 mb-4">
    <div class="col-md-4">
      <div class="card text-bg-primary">
        <div class="card-body">
          <h5 class="card-title">Total de Registros</h5>
          <p class="card-text fs-3">{{ total | default(0) }}</p>
        </div>
      </div>
    </div>
```

```
  {% set benign = por_label.get('Benign') if por_label else 0 %}
  {% set malicious = por_label.get('Malicious') if por_label else 0 %}

  <div class="col-md-4">
    <div class="card text-bg-secondary">
      <div class="card-body">
        <h5 class="card-title">Benign</h5>
        <p class="card-text fs-3">{{ benign | default(0) }}</p>
      </div>
    </div>
  </div>
  <div class="col-md-4">
    <div class="card text-bg-secondary">
      <div class="card-body">
        <h5 class="card-title">Malicious</h5>
        <p class="card-text fs-3">{{ malicious | default(0) }}</p>
      </div>
    </div>
  </div>
</div>

<!-- LISTAS TOPS -->
<div class="row g-3">
  <div class="col-md-6">
    <div class="card h-100">
      <div class="card-header bg-info text-white">Top Protocolos</div>
      <ul class="list-group list-group-flush">
```

```
      {% if top_proto %}
        {% for proto, qtd in top_proto.items() %}
        <li class="list-group-item d-flex justify-content-between align-items-center">
          {{ proto }}
          <span class="badge bg-primary rounded-pill">{{ qtd }}</span>
        </li>
        {% endfor %}
      {% else %}
        <li class="list-group-item">Sem dados de protocolo.</li>
      {% endif %}
    </ul>
  </div>
</div>

<div class="col-md-6">
  <div class="card h-100">
    <div class="card-header bg-warning text-dark">Top Serviços</div>
    <ul class="list-group list-group-flush">
      {% if top_service %}
        {% for serv, qtd in top_service.items() %}
        <li class="list-group-item d-flex justify-content-between align-items-center">
          {{ serv }}
          <span class="badge bg-secondary rounded-pill">{{ qtd }}</span>
        </li>
        {% endfor %}
      {% else %}
        <li class="list-group-item">Sem dados de serviço.</li>
```

```
        {% endif %}
      </ul>
    </div>
  </div>
</div>

<!-- AÇÕES -->
<div class="mt-4 d-flex gap-2">
    <a href="{{ url_for('visualizar_dados') }}" class="btn btn-primary">Explorar dados</a>
    <a href="{{ url_for('prever') }}" class="btn btn-outline-secondary">Fazer predição</a>
</div>
</div>

<!-- FOOTER -->
<footer class="text-center py-3 mt-4">
  <small>&copy; 2025 - Projeto IoT</small>
</footer>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

**ANEXO 14 - VISUALIZAR_DADOS.HTML**

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Visualização de Dados – IoT-23</title>
  <!-- Bootstrap + seu CSS pastel -->
   <link  href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet"/>
  <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
  <!-- Plotly -->
  <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
</head>
<body>
  <!-- NAVBAR -->
  <nav class="navbar navbar-expand-lg">
    <div class="container">
      <a class="navbar-brand" href="{{ url_for('index') }}">Painel IoT</a>
    </div>
  </nav>

  <div class="container mt-4">
    <h1 class="mb-3">📊 Visualização de Dados</h1>

    <!-- FILTROS -->
    <form id="filtros" class="row g-3 mb-4">
```

```
<div class="col-md-3">
  <label class="form-label">Label</label>
  <select id="f_label" class="form-select">
    {% for v in labels %}
      <option value="{{ v }}">{{ v }}</option>
    {% endfor %}
  </select>
</div>
<div class="col-md-3">
  <label class="form-label">Protocolo</label>
  <select id="f_proto" class="form-select">
    {% for v in protos %}
      <option value="{{ v }}">{{ v }}</option>
    {% endfor %}
  </select>
</div>
<div class="col-md-3">
  <label class="form-label">Serviço</label>
  <select id="f_service" class="form-select">
    {% for v in servs %}
      <option value="{{ v }}">{{ v }}</option>
    {% endfor %}
  </select>
</div>

{% if has_ts %}
<div class="col-md-3">
```

```
      <label class="form-label">Data inicial</label>
      <input id="f_ini" type="date" class="form-control">
    </div>
    <div class="col-md-3">
      <label class="form-label">Data final</label>
      <input id="f_fim" type="date" class="form-control">
    </div>
    {% endif %}

    <div class="col-12">
        <button id="aplicar" class="btn btn-primary" type="submit">Aplicar filtros</button>
        <a class="btn btn-outline-secondary" href="{{ url_for('visualizar_dados') }}">Limpar</a>
    </div>
  </form>

  <!-- CARDS RESUMO -->
  <div class="row mb-4">
    <div class="col">
      <div class="card text-bg-primary">
        <div class="card-body">
          <h5 class="card-title">Total (após filtros)</h5>
          <p id="card-total" class="fs-4">–</p>
        </div>
      </div>
    </div>
```

```
      <div class="col">
        <div class="card text-bg-secondary">
          <div class="card-body">
            <h5 class="card-title">Benign</h5>
            <p id="card-benign" class="fs-4">–</p>
          </div>
        </div>
      </div>
      <div class="col">
        <div class="card text-bg-secondary">
          <div class="card-body">
            <h5 class="card-title">Malicious</h5>
            <p id="card-malicious" class="fs-4">–</p>
          </div>
        </div>
      </div>
    </div>

    <!-- GRÁFICOS -->
    <div class="row">
      <div class="col-md-6 mb-4">
        <div class="card">
          <div class="card-header bg-info text-white">Frequência de Protocolos (Top)</
div>
          <div id="chart_proto" style="height:360px;"></div>
        </div>
      </div>
```

```
      <div class="col-md-6 mb-4">
        <div class="card">
          <div class="card-header bg-warning text-dark">Frequência de Serviços (Top)</
div>
          <div id="chart_service" style="height:360px;"></div>
        </div>
      </div>
    </div>

    <div class="row">
      <div class="col mb-4">
        <div class="card">
          <div class="card-header bg-secondary text-white">Série Temporal Diária</div>
          <div id="chart_time" style="height:380px;"></div>
        </div>
      </div>
    </div>

    <div class="mb-4">
          <a href="{{ url_for('index') }}" class="btn btn-outline-secondary">⬅ Voltar ao
início</a>
    </div>
  </div>

  <footer class="text-center py-3 mt-4">
    <small>&copy; 2025 - Projeto IoT</small>
```

```
  </footer>

  <script>
    async function carregarDados() {
      const params = new URLSearchParams();
      const label  = document.getElementById('f_label').value;
      const proto  = document.getElementById('f_proto').value;
      const serv   = document.getElementById('f_service').value;
      const iniEl  = document.getElementById('f_ini');
      const fimEl  = document.getElementById('f_fim');

      if (label && label !== "(qualquer)")  params.append('label', label);
      if (proto && proto !== "(qualquer)")  params.append('proto', proto);
      if (serv  && serv  !== "(qualquer)")  params.append('service', serv);
      if (iniEl) { const v = iniEl.value; if (v) params.append('data_ini', v); }
      if (fimEl) { const v = fimEl.value; if (v) params.append('data_fim', v); }

      const resp = await fetch('/api/dados?' + params.toString());
      return await resp.json();
    }

    function desenharGraficos(d) {
      // cards
      document.getElementById('card-total').textContent = d.total ?? 0;
                  document.getElementById('card-benign').textContent =
d.label_counts?.Benign ?? 0;
```

```
                document.getElementById('card-malicious').textContent =
d.label_counts?.Malicious ?? 0;

    // protocolos
    const p_keys = Object.keys(d.proto_counts || {});
    const p_vals = p_keys.map(k => d.proto_counts[k]);
    Plotly.newPlot('chart_proto', [{
      x: p_keys, y: p_vals, type: 'bar'
    }], {margin:{t:10}, xaxis:{automargin:true}});

    // serviços
    const s_keys = Object.keys(d.service_counts || {});
    const s_vals = s_keys.map(k => d.service_counts[k]);
    Plotly.newPlot('chart_service', [{
      x: s_keys, y: s_vals, type: 'bar'
    }], {margin:{t:10}, xaxis:{automargin:true}});

    // série temporal
    const serie = d.serie_diaria || [];
    Plotly.newPlot('chart_time', [{
      x: serie.map(o => o.data),
      y: serie.map(o => o.qtd),
      type: 'scatter', mode: 'lines+markers'
    }], {margin:{t:10}, xaxis:{title:'Data'}});
  }

  // primeiro carregamento
```

```
    (async () => {
      const dados = await carregarDados();
      desenharGraficos(dados);
    })();

    // aplicar filtros
    document.getElementById('filtros').addEventListener('submit', async (e) => {
      e.preventDefault();
      const dados = await carregarDados();
      desenharGraficos(dados);
    });
  </script>
</body>
</html>
```

**ANEXO 15 - PREVER.HTML**

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Previsão - Painel IoT</title>
    <!-- Bootstrap -->
        <link   href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">
    <!-- CSS personalizado -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>

    <!-- NAVBAR -->
    <nav class="navbar navbar-expand-lg">
        <div class="container">
            <a class="navbar-brand" href="{{ url_for('index') }}">Painel IoT</a>
        </div>
    </nav>

    <!-- CONTEÚDO PRINCIPAL -->
    <div class="container mt-4">
        <h1 class="mb-4">🤖 Previsão de Tráfego</h1>
            <p class="lead">Preencha os campos abaixo para verificar se o acesso é <strong>malicioso</strong> ou <strong>benigno</strong>.</p>
```

```
<form action="{{ url_for('prever') }}" method="POST" class="row g-3">

    <!-- Exemplo de campos de entrada -->
    <div class="col-md-4">
      <label for="duration" class="form-label">Duração (segundos)</label>
          <input type="number" step="any" class="form-control" id="duration"
name="duration" required placeholder="Ex: 0.123">
    </div>

    <div class="col-md-4">
      <label for="orig_bytes" class="form-label">Bytes Origem</label>
          <input type="number" step="any" class="form-control" id="orig_bytes"
name="orig_bytes" required placeholder="Ex: 512">
    </div>

    <div class="col-md-4">
      <label for="resp_bytes" class="form-label">Bytes Resposta</label>
          <input type="number" step="any" class="form-control" id="resp_bytes"
name="resp_bytes" required placeholder="Ex: 1024">
    </div>

    <div class="col-md-4">
      <label for="orig_pkts" class="form-label">Pacotes Origem</label>
          <input type="number" step="any" class="form-control" id="orig_pkts"
name="orig_pkts" required placeholder="Ex: 1024">
    </div>
```

```
        <div class="col-md-4">
          <label for="resp_pkts" class="form-label">Pacotes Resposta</label>
                <input type="number" step="any" class="form-control" id="resp_pkts"
name="resp_pkts" required placeholder="Ex: 12">
        </div>

        <div class="col-md-4">
          <label for="proto" class="form-label">Protocolo</label>
          <select id="proto" name="proto" class="form-select" required>
            <option value="tcp">TCP</option>
            <option value="udp">UDP</option>
            <option value="icmp">ICMP</option>
          </select>
        </div>

        <div class="col-12">
                <button type="submit" class="btn btn-success">🔍 Fazer Previsão</
button>
          <a href="{{ url_for('index') }}" class="btn btn-secondary">⬅ Voltar</a>
        </div>
      </form>

      {% if resultado %}
      <div class="alert alert-info mt-4">
        <h5>Resultado da previsão:</h5>
        <p>{{ resultado }}</p>
      </div>
```

```
        {% endif %}

    </div>

    <!-- FOOTER -->
    <footer class="text-center py-3 mt-4">
        <small>&copy; 2025 - Projeto IoT</small>
    </footer>

</body>
</html>
```

**ANEXO 16 - FOLHA DE ESTILOS CSS**

```
/* ======== ESTILO GERAL ======== */
body {
    background-color: #f8f6f3; /* bege pastel claro */
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    color: #2d2d2d;
}

/* ======== NAVBAR ======== */
.navbar {
     background: linear-gradient(90deg, #a0c4ff, #bdb2ff); /* azul pastel → lilás pastel
*/
}

.navbar-brand {
    font-weight: bold;
    color: #fff !important;
    letter-spacing: 0.5px;
}

/* ======== CONTAINER PRINCIPAL ======== */
.container {
    background-color: #ffffff;
    padding: 25px;
    border-radius: 12px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.05);
}
```

```
/* ======== TÍTULOS ======== */
h1, h5 {
    color: #3a3a3a;
}

h1 {
    font-weight: 600;
}

/* ======== FORMULÁRIOS ======== */
label {
    font-weight: 500;
    margin-bottom: 4px;
}

input.form-control,
select.form-select {
    border-radius: 8px;
    border: 1px solid #d8d8d8;
    transition: border-color 0.3s, box-shadow 0.3s;
}

input.form-control:focus,
select.form-select:focus {
    border-color: #bdb2ff;
    box-shadow: 0 0 5px rgba(189,178,255,0.5);
```

```
}

/* ======== BOTÕES ======== */
.btn-primary {
    background-color: #a0c4ff;
    border-color: #a0c4ff;
    font-weight: 500;
    border-radius: 8px;
    transition: all 0.3s ease-in-out;
}

.btn-primary:hover {
    background-color: #8ab9ff;
    border-color: #8ab9ff;
}

.btn-outline-secondary {
    border-radius: 8px;
    font-weight: 500;
}

/* ======== RESULTADO ======== */
#resultado {
    font-family: monospace;
    background-color: #fdfdfd;
    border-left: 5px solid #bdb2ff;
}
```

```
/* ======== FOOTER ======== */
footer {
    background-color: #f1f0ee;
    color: #555;
}
```