**Machine Learning-Based Phishing Detection: An Approach for Cyber Attack Prevention**

**(Detecção de *phishing* baseada em aprendizado de máquina:**

**uma abordagem para prevenção de ataques cibernéticos)**

 Flávia de Oliveira Silva, Gabriel Lima de Lana Pedrosa, Lucas Henrique de Paula Adelino

Orientador: Americo Talarico Neto

**Faculdade São Paulo Tech School - São Paulo, SP, Brasil**

**ABSTRACT**

This article explores the importance of balance in competitive League of Legends matches, a popular MOBA game where skill and player synergy are essential for a fair experience. The developed project uses Machine Learning to assess match balance based on real player data, such as KDA (Kill, Death, Assist), gold per minute, and global vision. Using Python and ML libraries, the system is trained to predict each team's chance of victory, aiming to improve the matchmaking process and benefit players, developers, and eSports platforms. The research highlights the impact of MMR (Matchmaking Rating) and League Points (LP) on balance, explaining how these metrics influence competitiveness in ranked matches. Future challenges include developing algorithms that account for the game’s complexity and diverse player profiles, as well as adapting the system to changes in the meta and game strategies.

1 ***Student's Bachelor's degree in Computer Science***

1. ***Professor of Computer Science at the São Paulo Tech School, Brazil***

**RESUMO**

O presente artigo explora a importância do balanceamento em partidas competitivas de League of Legends, um jogo popular do gênero MOBA, onde a habilidade e sinergia entre os jogadores são fundamentais para uma experiência justa. O projeto desenvolvido utiliza Machine Learning para avaliar o equilíbrio das partidas, baseado em dados reais dos jogadores, como KDA (Kill, Death, Assist), ouro por minuto e visão global. Utilizando Python e bibliotecas de ML, o sistema é treinado para prever a chance de vitória de cada equipe, buscando aprimorar o processo de matchmaking e beneficiar jogadores, desenvolvedores e plataformas de eSports. A pesquisa destaca o impacto do MMR (Matchmaking Rating) e dos pontos de liga (PdL) no balanceamento, explicando como essas métricas influenciam a competitividade nas ranqueadas. Segundo análise, possui desafios futuros no desenvolvimento de algoritmos que considerem a complexidade do jogo e a diversidade dos perfis dos jogadores, além da adaptação do sistema a mudanças no meta e nas estratégias de jogo.

1. INTRODUÇÃO

Com o avanço acelerado da tecnologia e o crescente uso de plataformas de jogos online, a demanda por soluções que garantam uma experiência equilibrada e justa para os jogadores também aumentou, em jogos competitivos como *League of Legends*, onde a habilidade e a sinergia entre jogadores podem determinar o desfecho de uma partida, a questão do balanceamento se tornou um tema relevante e complexo. *League of Legends* é um dos jogos mais populares do gênero multiplayer online battle arena (*MOBA*) e atrai milhões de jogadores ao redor do mundo, o que destaca a importância de proporcionar um ambiente competitivo e equilibrado.

Este artigo propõe uma abordagem inovadora para avaliar o balanceamento de partidas de *League of Legends* por meio de *Machine Learning*, usando dados dos jogadores e características específicas das partidas para prever o nível de equilíbrio entre os times. O desenvolvimento foi realizado com Python, explorando bibliotecas de machine learning, e a análise dos dados busca determinar a probabilidade de vitória para cada equipe, o que pode ser útil para desenvolvedores de jogos, plataformas de apostas e jogadores interessados em uma visão mais técnica do jogo.

Pela presente pesquisa, pretende-se contribuir para um ambiente de jogo mais justo e engajador, fornecendo uma base sólida para futuras melhorias no balanceamento de partidas. Ao entender as variáveis que influenciam o resultado de uma partida, este projeto não apenas busca aprimorar a experiência dos jogadores, mas também se posiciona na vanguarda das soluções baseadas em *Machine Learning* aplicadas a balanceamento de jogos eletrônicos.

**2. ENTENDENDO O BALANCEAMENTO E O MMR EM *LEAGUE OF LEGENDS***

*League of Legends* é um jogo do gênero multiplayer online battle arena (*MOBA*), onde duas equipes de cinco jogadores cada se enfrenta para destruir a base adversária. Cada jogador escolhe um personagem, chamado de "campeão", com habilidades únicas, e trabalha em equipe para derrotar os oponentes. O jogo é altamente competitivo, e o sistema de balanceamento é essencial para garantir que partidas ranqueadas sejam justas e ofereçam uma experiência desafiadora e equilibrada para todos os jogadores.

O *MMR* (Matchmaking Rating) é um sistema usado para calcular o nível de habilidade dos jogadores e criar partidas com equipes de habilidade semelhante. Esse sistema é fundamental para determinar se uma partida será balanceada, ajustando os times para que ambos tenham cerca de 50% de chance de vitória.

**2.1. Ranques, PdL e o Papel do MMR**

Embora o MMR seja um valor oculto, ele está diretamente ligado aos ranques dos jogadores e à quantidade de PdL (*Pontos de Liga*) que eles ganham ou perdem. O MMR mede o nível real de habilidade do jogador, enquanto os ranques e divisões servem como uma representação visível de sua posição competitiva. Conforme o jogador vence partidas, o sistema de MMR ajusta sua posição e oferece uma quantidade maior de PdL, facilitando a escalada de ranque se o desempenho continuar superior ao esperado.

Os PdL, ou Pontos de Liga, são essenciais para o sistema de ranqueamento. Eles aumentam com as vitórias e diminuem com as derrotas, aproximando o jogador de um ranque que reflita mais adequadamente seu MMR. Esse sistema permite que o MMR e o ranque eventualmente se alinhem, resultando em partidas onde os jogadores têm habilidades equivalentes.

**2.2. Como o MMR Impacta o Balanceamento das Partidas**

Em *League of Legends*, o balanceamento ideal é quando ambos os times têm uma chance similar de vencer. Quando o MMR de um jogador está alinhado com seu ranque, as vitórias e derrotas produzem mudanças de PdL mais equilibradas, indicando que ele está em uma posição competitiva adequada. No entanto, quando o MMR de um jogador é significativamente superior ao seu ranque, o sistema atribui mais PdL nas vitórias para ajustá-lo mais rapidamente ao nível de habilidade adequado.

Se o MMR está abaixo do ranque atual do jogador, as derrotas resultam em uma perda maior de PdL para refletir um ajuste no sistema. Esse alinhamento constante entre o MMR e os ranques dos jogadores é crucial para criar partidas competitivas, especialmente em partidas ranqueadas, onde cada ponto é importante.

**2.3. Estratégia para Análise de Balanceamento no Projeto**

O projeto utiliza Machine Learning para explorar esses dados de KDA(Kill, Deaths, Assistanci Ou Abates, Mortes, Assistências), ouro por minuto, visão global, entre outros fatores, e determinar se uma partida é equilibrada. As características avaliadas incluem a média de KDA, ouro, placar de visão e o histórico de desempenho individual dos jogadores em suas respectivas funções em partidas anteriores.

* 1. Estatísticas sobre o Gerenciamento de Partidas no *League of Legends*

De acordo com a Riot Games, em média, as equipes em *League of Legends* têm uma taxa de vitória esperada de aproximadamente 50%, com uma margem de erro de apenas +/-1%. Isso significa que a maioria das partidas é planejada para ter um equilíbrio ideal, com chances iguais para ambos os lados, embora existam fatores que podem afetar a percepção de equilíbrio.

Em sistemas de ranqueadas, os dados da Riot indicam que jogadores nas divisões mais altas, como Mestre e Desafiante, experimentam uma flutuação significativa nos Pontos de Liga (PdL) devido ao aumento constante de competitividade. No entanto, mesmo que esses jogadores façam pausas curtas, seus PdLs tendem a diminuir em relação ao nível médio de habilidade, que continua subindo enquanto eles estão inativos.

Ainda segundo a Riot, cerca de 70% dos jogadores estão concentrados nas divisões Prata e Bronze, onde os novos jogadores começam seu percurso nas ranqueadas. Essa concentração permite que o sistema de gerenciamento de partidas ajuste o nível dos novatos de forma mais precisa ao colocá-los em partidas com jogadores que já possuem dados confiáveis sobre habilidade e desempenho.

* 1. Desafios futuros

Com a desigualdade das partidas, um dos principais será trazer manter uma base de público ativa. Isso pode prejudicar a experiência dos jogadores e afetar a competitividade do cenário, especialmente em jogos de alto nível. Modelos de aprendizado de máquina avançados, capazes de analisar as características dos jogadores e prever os resultados das partidas, estão sendo desenvolvidos para identificar e mitigar esses desequilíbrios. No entanto, com a complexidade do jogo e a diversidade de perfis dos jogadores, o desafio permanece em criar algoritmos que sejam precisos e justos para diferentes contextos e níveis de habilidade. Além disso, a aplicação de técnicas avançadas, como processamento de linguagem natural para interpretação de comunicações entre jogadores e verificar o nível de satisfação dos jogadores durante as partidas.

1. INTERVENÇÃO

O objetivo deste projeto é desenvolver um sistema em Python de previsão de balanceamento em partidas de *League of Legends*, baseado em Machine Learning, capaz de identificar de forma precisa se uma partida é justa e balanceada. Utilizando dados reais de partidas, incluindo estatísticas de jogadores, equipes, e métricas como KDA, Ouro, Visão, o sistema será treinado para detectar padrões e características que indicam se um jogo terá chances equilibradas para ambos os times.

A intervenção proposta, ao desenvolver um sistema preditivo para partidas de *League of Legends*, busca fortalecer o processo de matchmaking, oferecendo uma análise aprofundada do balanceamento. Esse projeto não apenas busca aprimorar a experiência dos jogadores ao evitar partidas desiguais, mas também fornece uma base sólida para desenvolvedores de jogos e plataformas de eSports ao analisar partidas com maior precisão.

Além disso, a utilização de dados reais e métricas de balanceamento durante o treinamento do sistema permite a identificação de padrões e variáveis que impactam diretamente o resultado das partidas. O sistema também pode ser atualizado com novas informações sobre o meta do jogo, permitindo que ele se adapte a novas estratégias e alterações nas regras do jogo. Dessa forma, o projeto reforça a importância de um ambiente competitivo e balanceado, promovendo uma experiência mais satisfatória para todos os jogadores e contribuindo para o avanço das análises preditivas em jogos eletrônicos.

1. CATÁLOGO DE DADOS E ANÁLISE EXPLORATÓRIA

Este catálogo descreve as métricas extraídas e processadas da API pública da Riot Games para o League of Legends, com foco em dados de partidas e desempenho dos jogadores. Esses dados são posteriormente estruturados na camada Silver e utilizados para modelos preditivos e análise de performance.

* 1. KDA (Kills, Deaths, Assists).

Descrição: é uma métrica combinada que mede o desempenho geral do jogador em termos de abates, mortes e assistências durante a partida. É um dos indicadores mais tradicionais de performance.

Cálculo:

Kills (K): Número de abates realizados pelo jogador.

Deaths (D): Número de vezes que o jogador morreu durante a partida.

Assists (A): Número de assistências fornecidas pelo jogador, ou seja, quando ele ajudou a eliminar um inimigo sem ser o responsável pelo abate.

Fórmula:

KDA=K+ADKDA=DK+A

Objetivo: O KDA é utilizado para avaliar a eficiência do jogador. Um KDA alto indica um bom desempenho, enquanto um KDA baixo pode indicar um jogador com muitas mortes ou pouco envolvimento nas eliminações.

Aplicação: Usado em modelos preditivos para avaliar a performance de jogadores e prever os fatores que influenciam as vitórias ou derrotas.

* 1. Ouro por Minuto (Gold Per Minute - GPM)

Descrição: O GPM mede a quantidade de ouro que o jogador consegue acumular por minuto durante a partida. Essa métrica é crucial para entender a eficiência econômica do jogador, pois o ouro é essencial para comprar itens que aumentam o poder de combate.

Cálculo:

Ouro Total: A quantidade total de ouro acumulado pelo jogador

Tempo de Jogo: A duração total da partida.

Fórmula:

GPM=Ouro TotalTempo de JogoGPM=Tempo de JogoOuro Total

Objetivo: Avaliar a capacidade do jogador de acumular ouro durante a partida, um indicador importante da habilidade do jogador em farmar (realizar o last-hit nas minions) e participar de objetivos.

Aplicação: Usado para prever se o controle econômico de um time impacta a vitória e analisar a correlação entre GPM e outras métricas de sucesso, como o KDA.

* 1. Dano por Minuto (Damage Per Minute - DPM)

Descrição: é a média de dano que o jogador causa por minuto na partida. Esse dado é útil para medir o impacto que o jogador tem durante o combate, especialmente em relação aos inimigos.

Cálculo:

Dano Total: A quantidade total de dano causado pelo jogador a outros campeões, estruturas e objetivos.

Tempo de Jogo: A duração total da partida.

Fórmula:

DPM=Dano TotalTempo de JogoDPM=Tempo de JogoDano Total

Objetivo: Acompanhar a agressividade e a contribuição ofensiva de um jogador. Jogadores com alto DPM normalmente têm um papel significativo em combates de equipe e nas lutas por objetivos.

Aplicação: Usado para treinar modelos preditivos que avaliem o impacto do dano no desempenho do time e em eventos chave da partida.

* 1. Score de Visão.

Descrição: refere-se ao controle de visão de um jogador durante a partida, como a colocação e a destruição de wards (sentinelas), o que afeta diretamente a capacidade de um time de se manter seguro ou surpreender o adversário.

Cálculo:

Wards Colocadas: Número de wards colocadas pelo jogador.

Wards Destruídas: Número de wards destruídas por inimigos, ou as wards inimigas que o jogador elimina.

Objetivo: Medir o controle de visão de um jogador no mapa, uma parte essencial da estratégia e do controle de objetivos no jogo.

Aplicação: Usado para modelos que avaliam como a visão do mapa afeta as vitórias ou derrotas e a capacidade de um time de conquistar objetivos de forma estratégica.

* 1. Tempo de Jogo (Game Duration)

Descrição: é a duração total da partida, geralmente expressa em minutos. Essa métrica é importante para entender a dinâmica do jogo, como o ritmo de progressão do jogo e a relação entre o tempo e o sucesso de diferentes estratégias.

Objetivo: Permite medir e categorizar as partidas de acordo com a sua duração, possibilitando análises baseadas no tempo.

Aplicação: Usado para categorizar partidas em modelos preditivos, como jogos rápidos versus jogos mais longos, ajudando a entender como a duração da partida pode influenciar o desempenho de jogadores e times.

* 1. Objetivos do Jogo

Descrição: Os objetivos do jogo, como Baron, Dragão, Torres e Inibidores, são metas estratégicas que os times devem conquistar ao longo da partida. A captura desses objetivos tem um grande impacto no progresso da partida, oferecendo vantagens importantes como ouro extra, buffs, ou poder de destruição.

Subcategorias:

Baron: Quantidade de vezes que o Baron Nashor foi capturado.

Dragão: Quantidade de dragões capturados.

Torres: Quantidade de torres destruídas.

Inibidores: Quantidade de inibidores destruídos.

Objetivo: Avaliar o controle e a execução de objetivos chave, que muitas vezes determinam o desfecho da partida. A conquista desses objetivos pode proporcionar uma vantagem decisiva no jogo, especialmente quando combinados com boas estratégias de equipe.

Aplicação: Usado para prever como a captura de objetivos influencia as vitórias, a estratégia do time e a performance geral, ajudando a identificar o impacto desses objetivos nos resultados das partidas.

* 1. ID da Partida (Match ID)

Descrição: O ID da Partida é um identificador único atribuído a cada partida jogada. Esse ID é importante para rastrear e associar todas as métricas e dados específicos de uma partida.

Objetivo: Garantir a unicidade e rastreabilidade dos dados, permitindo a correlação entre os diferentes registros e facilitando a consulta e a análise.

Aplicação: Essencial para a estruturação dos dados, servindo como chave primária no banco de dados ou sistema de arquivos para identificar unicamente cada jogo.

* 1. Tentativa 1 - Árvore de Decisão

Árvore de decisão é um modelo de aprendizado de máquina supervisionado que organiza seus resultados em forma de uma árvore (nós) para classificar ou prever valores baseada nos atributos. É útil devido à sua eficácia em lidar com dados complexos e capacidade de requerer pouco pré-processamento dos dados.

* + 1. BIBLIOTECAS UTILIZADAS – modelo
* pandas: manipulação e análise de dados
* Sklearn: aprendizado de máquina
* Graphviz: manipulação e processamento de grafos

**Figura 3** – Bibliotecas e módulos utilizados na árvore de decisão

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

* + 1. ENTENDENDO O CÓDIGO – modelo

As primeiras condições foram a importação do *dataset* e exclusão da coluna “url” da tabela pois não seria utilizada nesse momento do tratamento dos dados (figura 4).

**Figura 4** – Lendo *dataset* e excluindo coluna “url”

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

Após esses requisitos, os dados começaram a ser trabalhados.  
O primeiro passo foi converter o tipo da coluna “*status*” de *string* (texto) para *in*t (números inteiros), de forma que os campos com o valor “*legitimate*” fossem substituídos por “1” e os que possuíssem o valor “*phishing*” por “0” (figura 5).

**Figura 5** – Convertendo a coluna “*status*” de texto para numérico

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

**Figura 6** – Coluna *status* antes da conversão para numéirico

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Criação do autor

**Figura 7** – Coluna *status*após conversão para numéricos

Interface gráfica do usuário, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Fonte: Criação do autor

Com isso realizado, as variáveis independentes e dependentes foram separadas e o treinamento delas foi feito utilizando o módulo DecisionTreeClassifier do Sklearn (figura 8).

**Figura 8 –** Trecho de código para a separação de variáveis dependentes e independentes e o treinamentoTexto

Descrição gerada automaticamenteFonte: Criação do autor

Com esse treinamento foi encontrada uma acurácia (nível de exatidão do resultado obtido a partir dos processos realizados) de: 0.93%.

Após isso foi plotada a árvore de decisão, que é uma representação de uma tabela de decisão sob a forma de árvore.

**Figura 9** – Visualização da Árvore de Decisão obtida Gráfico, Gráfico de dispersão

Descrição gerada automaticamenteFonte: Criação do autor

Com a visualização da imagem da árvore nota-se que não era o modelo ideal para o conjunto de dados, mesmo obtendo uma alta acurácia, pois está repleta de ramificações (possibilidades), o que torna um problema para a busca de apenas um caminho de maior veracidade.

Com isso, a pesquisa foi para a sua segunda tentativa, a normalização dos dados.

* 1. Tentativa 2 – Classificação – modelo

O modelo de classificação, assim como a Árvore de Decisão, é algoritmo que aprende a rotular dados com base em padrões identificando durante o treinamento. É usado para prever a categoria ou classe de novas observações. Esses modelos são aplicados em várias áreas para reconhecimento de padrões.

* + 1. BIBLIOTECAS UTILIZADAS – modelo
* Sklearn: aprendizado de máquina
* Numpy: processamento de grandes, multi-dimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes

**Figura 10** – *Import* da biblioteca Normalizer



Fonte: Criação do autor

* + 1. ENTENDENDO O CÓDIGO – modelo

E novamente o mesmo processo de treinamento das variáveis para obter o resultado da acurácia, porém agora utilizando o módulo KNeighborsClassifier (figura 11).

**Figura 11 –** Trecho de código para a separação de variáveis dependentes e independentes e classificação do modelo com KNeighborsClassifier

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

O resultado de saída da acurácia foi de: 85%. Um resultado menor do que o do primeiro modelo, porém, ainda um bom resultado quando se olha para uma escala global em utilizações de projetos, onde acima de 80% já é classificado como um bom treinamento dos dados.

Após isso, se obtém os métodos para classificar a importância de cada coluna (através de um *array)* do modelo de dados e fazendo um *plot* (exibição) do gráfico para uma melhor visualização (figura 12 e 13).

**Figura 12 -** Indice de importância

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamenteFonte: Criação do autor

**igura 13 -** Gráfico do índice de importância

Uma imagem contendo Linha do tempo

Descrição gerada automaticamente

Fonte: Criação do autor

A partir desse ponto, nota-se que a trajetória estava equivocada. Como citado no começo da tese, a base de dados é um conjunto de requisitos que leva a um valor único no final. Com esses modelos foi testado os campos da tabela de forma individual, onde o desfecho não conduziria a nenhum destino significativo.

Descartando mais uma vez esses modelos, se faz necessário partir para a terceira tentativa.

* 1. Tentativa 3 – Análise dos componentes principais (PCA) – modelo

Agora tendo em mente que o principal problema é a grande quantidade de colunas, é aplicada a técnica de PCA (análise de componentes principais) para reduzir as dimensionalidades da base de dados.

* + 1. BIBLIOTECAS UTILIZADAS – modelo
* Numpy: processamento de grandes, multi-dimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes;
* Sklearn: aprendizado de máquina,
* Pandas: manipulação e análise de dados;
* Seaborn: visualização de dados.
  + 1. ENTENDENDO O CÓDIGO– modelo

Fazendo uma abordagem diferente dessa vez, é criada uma lógica nesse primeiro momento para remover as colunas que estão com seus campos apenas em binários (0 ou 1) para não dificultar a análise (figura 14).

**Figura 14 -** Remoção de colunas binárias

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

Além de transformar o valor NaN (valor ou símbolo usado para representar um valor indefinido ou irrepresentável) para zero em todas as colunas da tabela (figura 15).

**Figura 15 –** Transformando campos NaN em 0

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Fonte: Criação do autor

Se iniciam os ajustes de escala dos dados. Calculando a média de cada variável, o desvio padrão e escalando os dados com o seguinte cálculo (figura 16):

**Figura 16 –** Média e desvio padrão

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Criação do autor

Após isso, é calculada a sua matriz de correlação e é plotado um *Heatmap* (matriz de correlação) para uma melhor visualização e entendimento dos dados. Nesse momento utilizam-se apenas 15 campos para conseguir visualizar de uma forma mais limpa o gráfico (figura 17).

**Figura 17 –** Matriz de correlação

Uma imagem contendo Gráfico

Descrição gerada automaticamente

Fonte: Criação do autor

Nesse *Heatmap* está em escala de tons de azul, onde quanto mais forte sua tonalidade, mais forte é a correlação de cada campo, um a um, do *dataset*.

Isso ocasiona um maior embasamento do que foi descoberto nos testes dos modelos de dados anteriores. Onde é impossível (nessa base de dados) comparar apenas um componente com outro. Pode-se observar que suas tonalidades estão em tons claros, praticamente em branco, exibindo assim que suas relações são muito fracas, fortalecendo a ideia que se deve considerar todas as colunas em suas totalidades, não apenas em comparação de dois campos.

Nisso, é notável que está sendo enfrentado nesses testes a maldição da dimensionalidade, que acontece, em maioria, quando se trabalha com grandes volumes de dados. Em um trecho do artigo da ODSC - Ciência de Dados Abertos, diz que “à medida que aumenta o número de recursos para ajustar um modelo, a qualidade do modelo ajustado aumentará proporcionalmente, mas esse não é necessariamente o caso”, que é o que aconteceu de acordo com os testes de modelos de *machine learning* abordados, fazendo com que existisse uma falsa sensação de assertividade, mas com riscos de *overfitting*.

Com isso, é utilizado a técnica que trata dessa redução de dimensionalidade, o PCA (análise dos componentes principais), que fornece uma maneira de resumir esses dados em pequenos números de variáveis que representam de uma forma coletiva o conjunto original.

Com isso segue-se para a terceira tentativa, a análise dos componentes principais.

Se começa com a decomposição da matriz de variância e covariância e criação dos componentes (figura 18 e 19).

**Figura 18 –** Matriz de variância e covariância

Texto

Descrição gerada automaticamente

Fonte: Criação do autor

**Figura 19 –** Criação dos componentes

Logotipo

Descrição gerada automaticamente

Fonte: Criação do autor

Feitos os testes, foram se reduzindo as dimensionalidades até chegar em 40 dimensões (figura 18) e apenas 20 componentes (figura 19), onde validados que não iria interferir em grande impacto no modelo inicial.

Após isso é possível visualizar que 80% do modelo de dados é explicado a partir do décimo oitavo componente (figura 20).

**Figura 20 –** Visualização do *Array*

Texto

Descrição gerada automaticamenteFonte: Criação do autor

Com isso, são criados 18 PCA’s para serem trabalhados (figura 21).

**Figura 21 –** Criação dos PCA's

Uma imagem contendo Texto

Descrição gerada automaticamente

Fonte: Criação do autor

**Figura 22 –** Adicionando os PCA’s

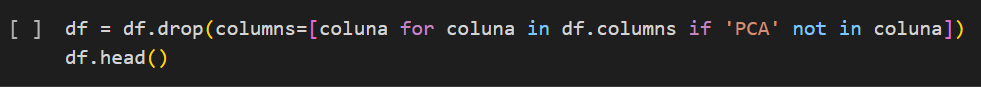
Calendário

Descrição gerada automaticamente

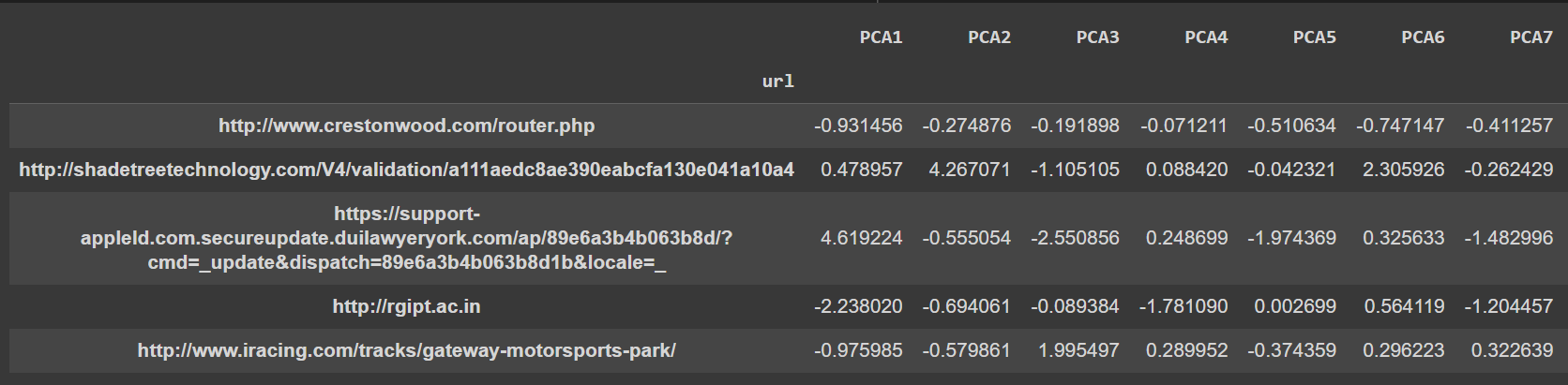
Fonte: Criação do autor

Sabendo que todas as informações necessárias já estão incluidas dentro desses PCA’s, todos os outros campos da base de dados que já não se fazem mais necessários são deletados, mantendo apenas o índice que são as URL’s (figura 23 e 24).

**Figura 23 –** Excluindo demais colunas do modelo

Fonte: Criação do autor

**Figura 24 –** Visualização da novo *dataset*

Fonte: Criação do autor

1. *MACHINE LEARNING* – DECISÃO FINAL – modelo

Com a proposta de utilizar os PCA gerados no pré-processamento para as previsões de uma URL, é iniciado o código de *machine learning* dentro do Jupyter(ambiente de computação interativa) que está alocado em uma máquina EC2.

* 1. Bibliotecas Utilizadas – modelo
* ApacheSpark: *framework* para processamento de dados
* Urllib: pacote que coleta vários módulos para trabalhar com URLs

**Figura 25** – Importando bibliotecas da *machine learning* final

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Fonte: Criação do autor

**Figura 26** – Importando módulos do pyspark e urllib

Texto

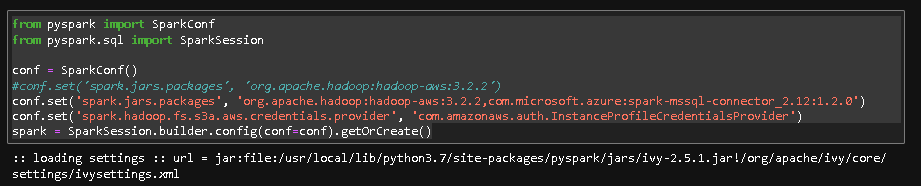
Descrição gerada automaticamente

Fonte: Criação do autor

* 1. Entendendo o Código – modelo

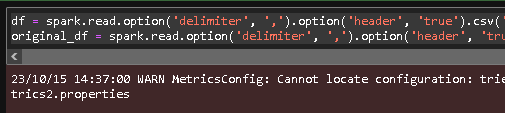
O código é iniciado de forma a configurar uma sessão do *Apache Spark* com o módulo SparkSession (figura 27).

**Figura 27 -** Criação de sessão

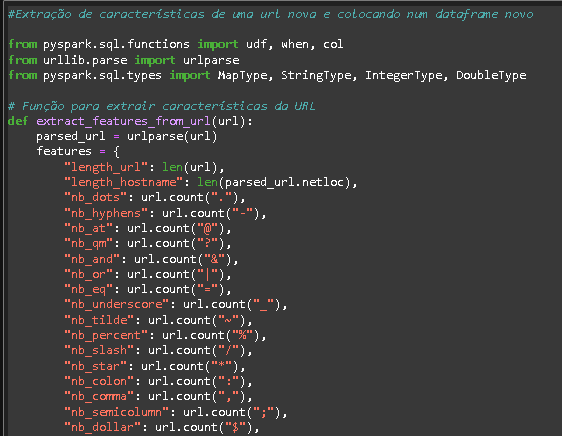
Fonte: Criação do autor

Em seguida, é utilizado o comando “*read*” para acessar os arquivos dentro da S3, sendo as variáveis “df” e “original\_df” (figura 28) os *dataframes* que contém os *dataset`s* com os 18 PCA’s gerados e o *dataset* original, respectivamente.

**Figura 28 –** Acesso aos documentos

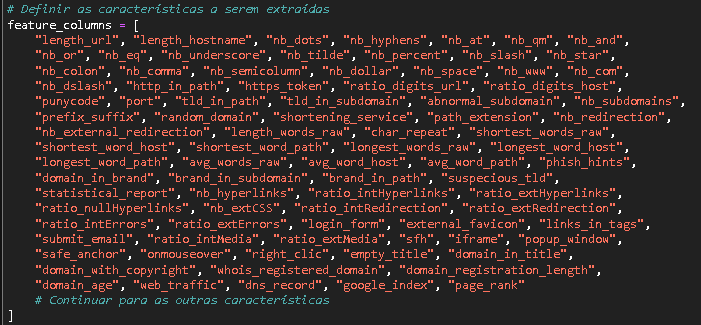
Fonte: Criação do autor

Posteriormente são realizadas as extrações de características provindas de uma url nova (figura 29), a qual seria disponibilizada pelo usuário para a análise de possibilidade de *phishing* ou legitimo.

**Figura 29 -** Extração de característicasFonte: Criação do autor

É feita listagem de todas as características que serão extraídas por meio de um objeto chamado feature\_columns (figura 30).

**Figura 30 –** Listagem de características

Fonte: Criação do autor

Aqui é definida uma URL que será trabalhada para a previsão, essa seria requisição de exemplo por parte do cliente (figura 31).

**Figura 31 –** URL especificada para uso



Fonte: Criação do autor

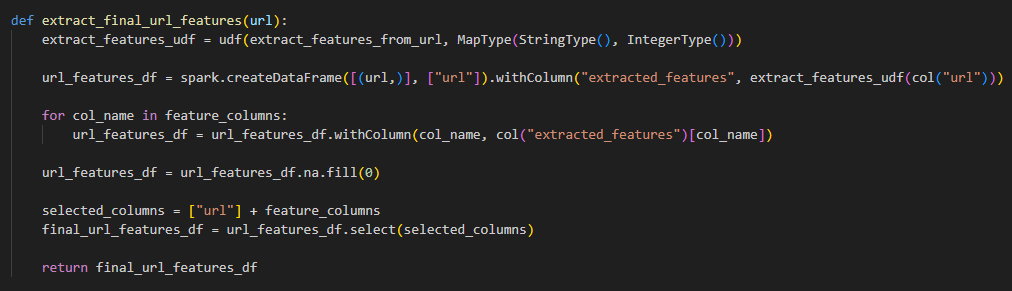
Aqui é iniciado o processo de criação de uma UDF (User Define Function) que irá criar toda a lógica para definição das características extraídas e colocá-las em um novo dataframe próprio (figura 32).

**Figura 32 -** Criação de UDF para extração

Fonte: Criação do autor

Extraindo *features* da URL a ser analisada e chamando as funções anteriormente criadas (figura 33).

**Figura 33 –** Extraindo e guardando novas características

Fonte: Criação do autor

No processo de extração das *features* da URL analisada, inicialmente é criada uma UDF para extrair suas características. Informação que é armazenada na função definida “extract\_features\_url”.

Em seguida é aplicada a UDF para extrair características da URL e criar um *dataframe* com uma única linha, o “url\_features\_df”. Nele os dados são representados como dicionários de características (mapas) a partir da coluna “extracted\_features” e o código está convertendo essas características em colunas individuais desse novo *dataframe*. E, posteriormente, trata as colunas com valores nulas, substituindo seu valor nulo por zero.

Logo em seguida é dada origem ao *dataframe* das características novas chamado de ”final\_url\_features\_df”.

* + 1. INICIANDO TREINAMENTO DO MODELO *RANDOM FOREST* – modelo

Aqui começa a parte direcionada ao treinamento do modelo. Nesse ponto foram enfrentados desafios associados à implementação dos Componentes Principais de Análise (PCA) que foram criados e a fim de prosseguir o desenvolvimento, foram armazenados como informações valiosas caso fossem necessárias análises mais profundas em versões futuras do projeto. Optou-se por seguir por um desenvolvimento de um modelo de classificação *random forest* (atualmente sem a utilização dos PCA),considerando suas vantagens em relação a uma base de informações muito densas.

Com o *random forest* foi possível evitar problemas de *overfitting* em relação ao *decision tree* pois as árvores individuais são muito mais propensas a isso e a média de “voto” de várias árvores do *random forest* ajuda em um modelo mais equilibrado.

Além disse, esse modelo também se mostra superior no quesito de generalização considerando sua natureza aleatória nas amostras de treinamentos realizada, fazendo com que dados não vistos no treino sejam melhor analisados e sinergizando com o escopo do projeto de receber URL’s novas para análise.

Relacionado aos hipêrparametros, o *random forest* tem uma eficiência maior em questão de amortecimento a sensibilidade, ou seja, pequenas alterações nos mesmos não seriam tão impactantes no modelo, o que é um viés imprescindível para uma base de dados tão robusta.

Finalmente, dado sua natureza de realizar o processo de classificação por árvores em processos paralelos, tem-se uma eficiência computacional maior considerando outros modelos, como por exemplo o de *decision tree* que realiza os processos com as árvores de forma dependente uma da outra.

Em resumo, r*andom forest* é visto em consenso geral como a melhor opção para modelo de classificação considerando seus aspectos de funcionamento em relação a bases de dados grandes, proporcionando uma melhor generalização e melhor desempenho.

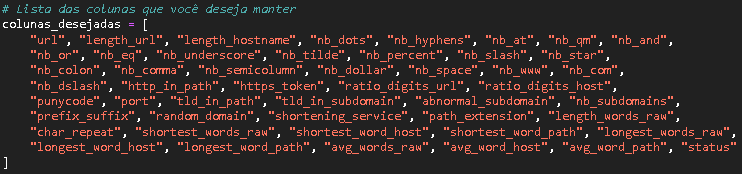
Inicia-se o processo instanciando o dataframe original assim como convertendo a sua coluna “*status*” (ao qual possui os rótulos de “*phishing*” ou “l*egitimate*”) para 0 e 1.

**Figura 34 –** Carregando antigo *dataframe* de forma bináriaFonte: Criação do autor

E aqui é onde filtra-se o que vai ser trabalhado pro treinamento do modelo, foi decidido seguir por uma premissa de filtrar as características que são possivelmente captáveis a partir de uma ”leitura” de uma *string* (figura 35). Por exemplo no texto ”www.exemplo.com” em pode-se perceber dois pontos nessa url, porém somente com a leitura não é possível localizar informações como ”formulários” provindas da mesma, por exemplo.

Dada essa premissa, foi necessário descartar a utilização dos PCA’s.

**Figura 35 -** Filtrando informações captáveis a partir da *string*

Fonte: Criação do autor

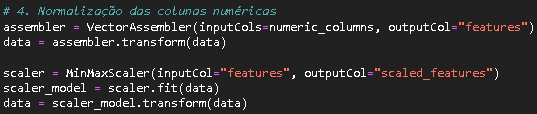
Foram convertidos todos os valores capturados das características filtradas em valores numéricos para serem usados no treinamento (figura 36).

**Figura 36 -** Conversão de texto em número

Fonte: Criação do autor

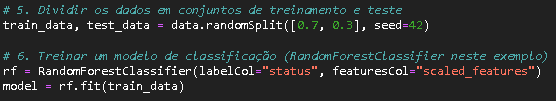
Em seguida com os valores já convertidos aplica-se a normalização dos valores para uma escala de 0 pra 1 utilizando o MinMaxScaler (figura 37).

**Figura 37 –** Normalizando dados

Fonte: Criação do autor

Em seguida é feita a divisão do conjunto de dados entre treino e teste (assim como é treinado um modelo do tipo r*andom forest)* com base nos dados processados, sendo adotada divisão da base em 70 para 30 para treinamento e teste, respectivamente (figura 38).

**Figura 38 -** Divisão de teste e treino

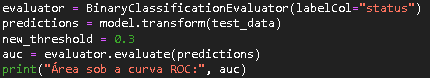


Fonte: Criação do autor

* + 1. AVALIANDO MODELO – modelo

Para a avaliação de funcionalidade do modelo que foi treinado, utiliza-se especificamente a métrica AUC (Área sob a curva) da curva ROC que calcula com base em previsões no conjunto de teste (figura 39 e 40).

**Figura 39 -** Avaliação do modelo



Fonte: Criação do autor

**Figura 40 –** Valor ROC calculado



Fonte: Criação do autor

A métrica de curva ROC (Receiving Operating Characteristics) é amplamente utilizada para a avaliação de modelos, considerando o valor obtido da “Área sob a curva ROC:” onde os modelos que se aproximam do 1 são cada vez mais eficientes, pode-se dizer que o modelo está tendo um desempenho considerável em relação a previsão do *phishing* e em fornecer verdadeiros positivos conforme indicado na figura 41 onde o modelo desenvolvido está se aproximando mais da taxa de definir verdadeiros positivos eficientemente.

**Figura 41** **-** Gráfico de métrica ROC

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

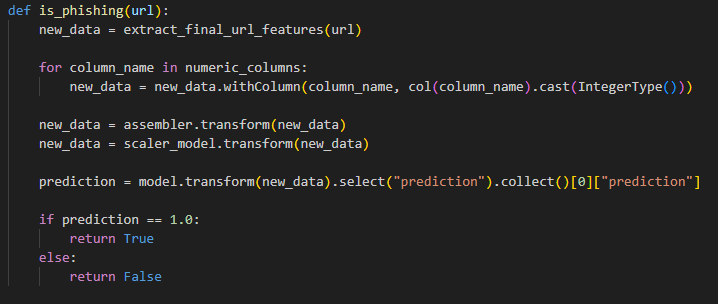
Fonte: Criação do autor

* + 1. CLASSIFICAÇÃO DE URL – modelo

Com a avaliação e aprovação do modelo, cria-se a função que vai realizar processos focados no *dataframe* da url a ser analisada. Que é feita convertendo-se seus valores para numéricos, aplicando a transformação das características pelo assembler e normalizando-se os dados (figura 42).

Assim, quando o resultado da previsão for 1 devolverá que “*True*”, caso contrário “*False*”.

**Figura 42 –** Definidor de resultado

Fonte: Criação do autor

**6.** **API *BACKEND***

Para conectar a *machine learning* desenvolvida ao *plug-in*, foi construída uma API que possui a funcionalidade de receber a URL a ser analisada, liberar o CORS para que o *frontend* tenha acesso e devolver o resultado da análise da ML.

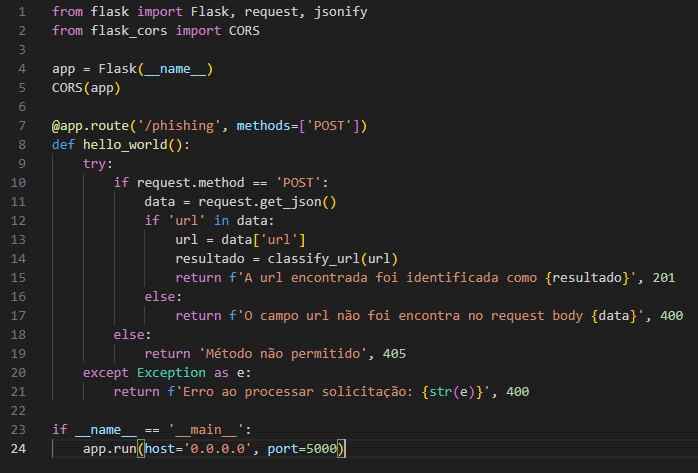
1. 1. Bibliotecas utilizadas – modelo

* *flask:* comumente utilizada para construção de APIs em pythondevida à sua fácil usabilidade.
  1. Funcionamento da API – modelo

1. Validar o verbo utilizado na requisição (permitido somente POST);
2. Extrair o campo URL no *request body* da requisição (e lançar erro caso não exista);
3. A partir da URL extraída chamar a função *is\_phishing()* criada na ML para identificar *phishing* e devolver o resultado de sua classificação na resposta da API;
4. Para demais erros encontrar ao executar chamada, tratar de forma genérica retornando o erro com HTTP *status* 400.

Todos esses passos são realizados conforme é mostrado na figura 43 a seguir.

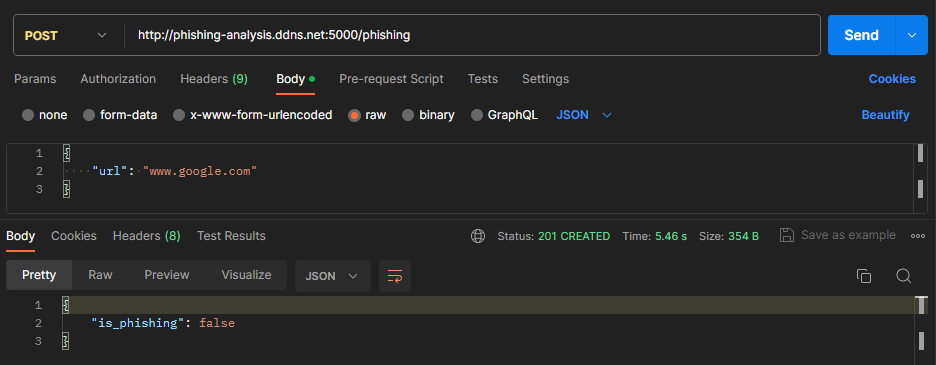
**Figura 43 –** Criação corpo da API



Fonte: Criação do autor

* 1. Resultado da Chamada da API no Postman

**Figura 44 –** Chamada da API no Postman

Fonte: Criação do autor

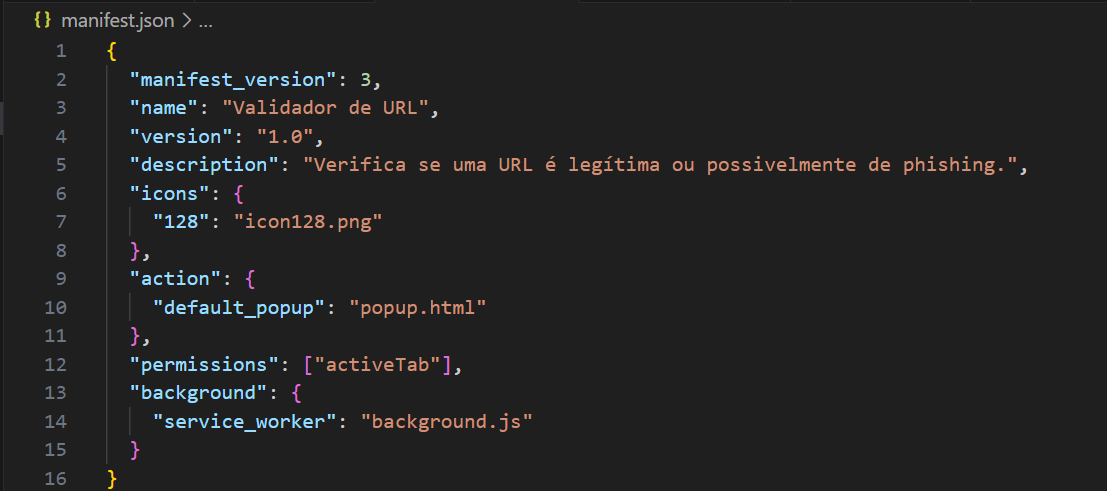
1. *PLUG-IN* – modelo

Foi feita a criação do *plug-in* para navegadores Chrome, com a finalidade de atuar como validador de URLs. Isso permite que os usuários verifiquem a legitimidade ou potencial caráter de *phishing* de um *link* antes de clicarem, especialmente se houver suspeitas sobre sua confiabilidade.

Para isso, ele foi feito utilizando Visual Studio Code como interface, com o uso de ferramentas puras do *Front-end*, como: HTML, CSS e *JavaScript*.

A primeira etapa do processo de criação envolveu a elaboração do arquivo manifest.json. Este arquivo é fundamental para a extensão do Chrome, pois detalha informações como título, versão, descrição, ícone, permissões e a conexão com outros arquivos da extensão.

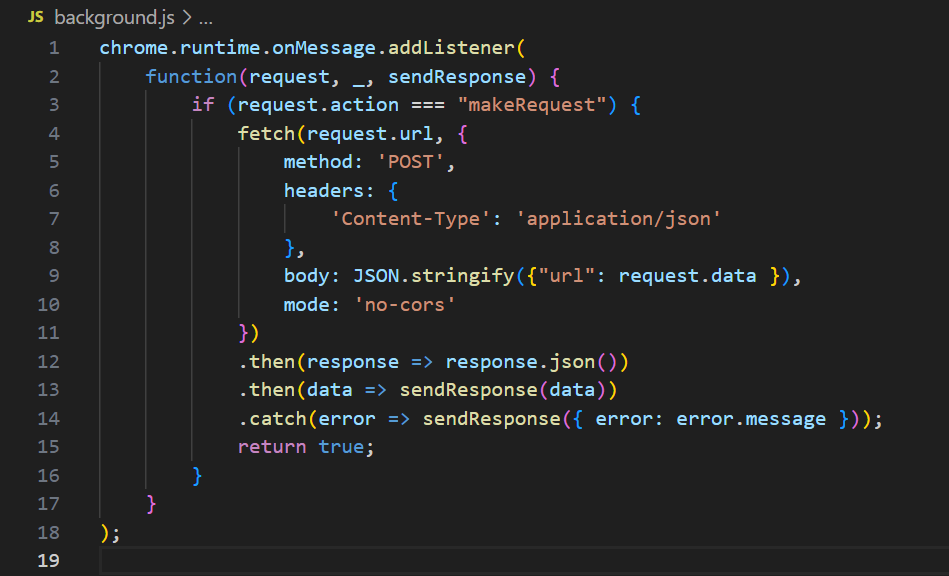
**Figura 45 –** Configurações do *plug-in*

Fonte: Criação do Autor

No arquivo de fundo ‘background.js’ está configurado para ouvir mensagens recebidas. Quando uma mensagem com a ação "makeRequest" é recebida, o código utiliza a API Fetch para realizar uma solicitação POST para uma URL especificada nos dados da mensagem. O corpo da solicitação contém uma URL a ser validada. A resposta JSON resultante é então enviada de volta ao remetente da mensagem original.

O código trata erros durante a solicitação, enviando uma mensagem de erro de volta ao remetente se ocorrer algum problema.

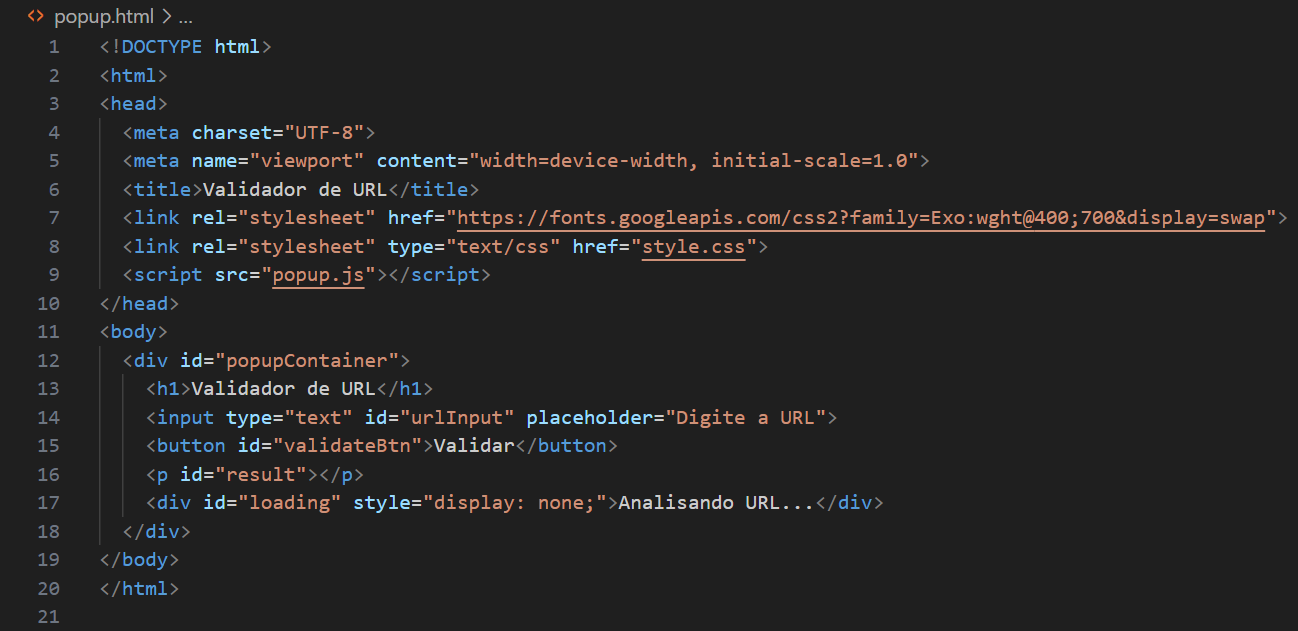
**Figura 46 –** Configurações do *background*



Fonte: Criação do Autor

O arquivo ‘popup.html’ abriga a estrutura fundamental do *pop-up* criado e está interligado aos demais arquivos. Geralmente, esse *pop-up* é visualizado quando o usuário aciona o botão/ícone da extensão localizado na barra de ferramentas do navegador Chrome.

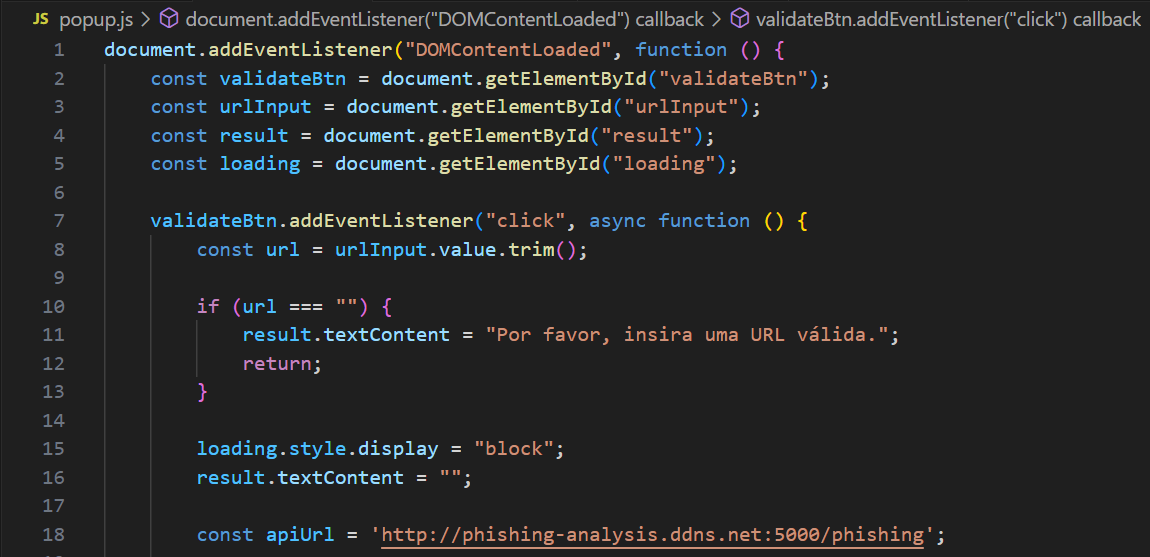
**Figura 47 –** Estrutura base do *plug-in*



Fonte: Criação do Autor

Dentro do arquivo ‘popup.js’, está o código em JavaScriptresponsável por interagir com o *pop-up* e conduzir a validação da URL utilizando a API. Este arquivo reflete a lógica elaborada para apresentar o resultado da validação da URL, informando ao usuário se o *link* pesquisado é legítimo ou se apresenta características de *phishing*. Adicionalmente, inclui verificações do *status* da API e mensagens para fornecer informações de erro de maneira compreensível ao usuário.

**Figura 48 –** Validações e chamada da API (parte 1)



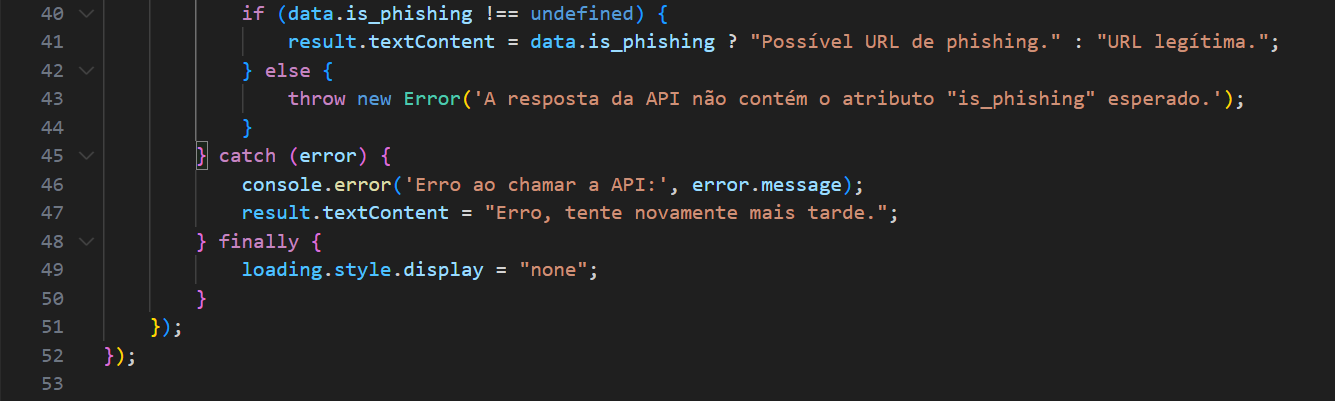
Fonte: Criação do Autor

**Figura 49 –** Validações e chamada da API (parte 2)



Fonte: Criação do Autor

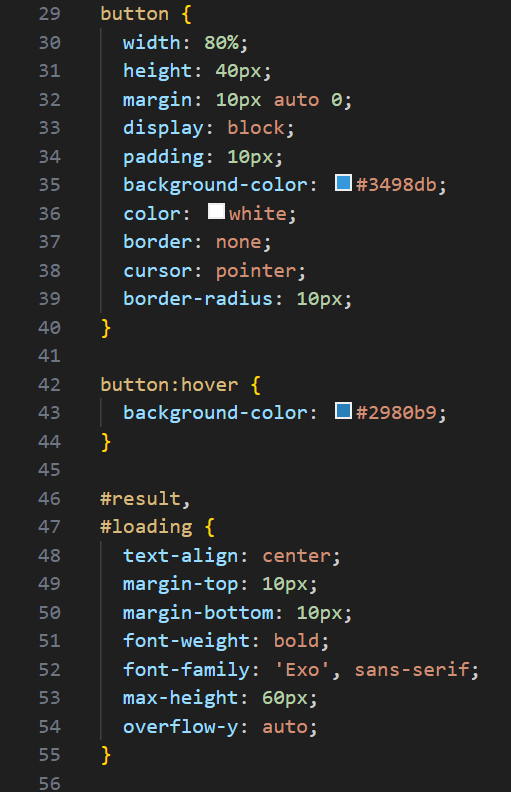
**Figura 50 –** Validações e chamada da API (parte 3)



Fonte: Criação do Autor

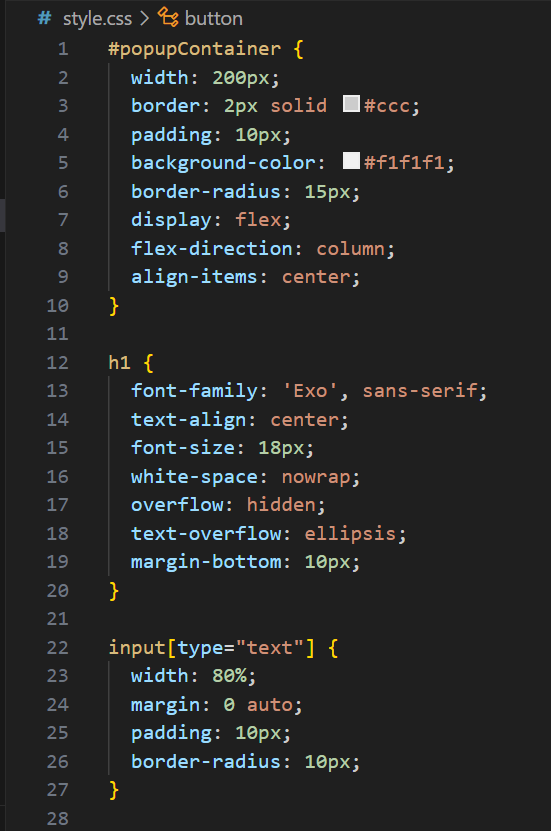
O arquivo ‘style.css’ concentra toda a formatação visual destinada ao *pop-up.* Ele determina como o *pop-up* será estilizado, incluindo propriedades como cores, dimensões, posicionamento e outros aspectos visuais, garantindo uma experiência visual coerente e agradável para o usuário da extensão.

**Figura 51 –** Estilização do *plug-in* (parte 1)



Fonte: Criação do Autor

**Figura 52 –** Estilização do *plug-in* (parte 2)



Fonte: Criação do Autor

* 1. Configuração do Chrome – modelo

Agora, com todos os arquivos devidamente configurados e armazenados no mesmo diretório, você está pronto para conectar a extensão ao seu navegador Chrome. Para isso, basta seguir os seguintes passos:

1. Abra seu navegador e localize a opção de extensões e depois em gerenciar extensões, geralmente localizada no canto superior direito:

**Figura 53 –** Localização da configuração no Google Chrome (parte 1)

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Criação do Autor

**Figura 54 –** Localização da configuração no Google Chrome (parte 2)

Texto

Descrição gerada automaticamente

Fonte: Criação do Autor

1. Ative o “Modo do Desenvolvedor” no canto superior direito da página

**Figura 55 –** Ativação do modo desenvolvedor

Texto

Descrição gerada automaticamente

Fonte: Criação do Autor

1. Acione o botão “Carregar sem compactação” e selecione o diretório do seu projeto.

**Figura 56 –** Upload do diretório do *plug-in*

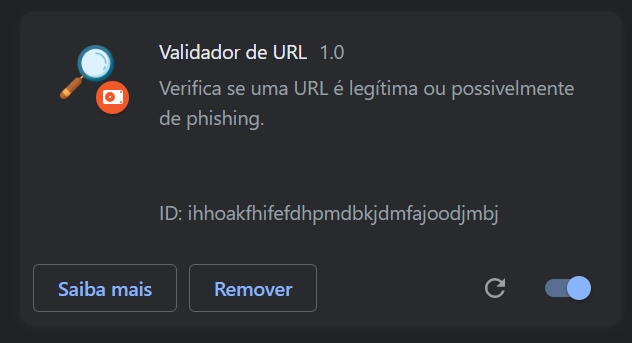
Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Fonte: Criação do Autor

1. Pronto. Sua extensão será adicionada ao Chrome e você já poderá observar na tela e o ícone que você definiu na barra de ferramentas.

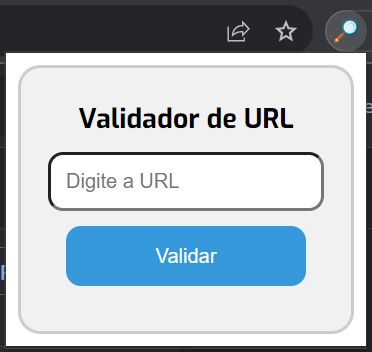
**Figura 57 –** Visualização da extensão



Fonte: Criação do Autor

A partir de agora, ao clicar no ícone da extensão, o *pop-up* será exibido, apresentando um campo para inserir a URL desejada e um botão para realizar a validação. Este processo oferece aos usuários a capacidade de verificar a legitimidade ou a possível presença de *phishing* em URLs antes de interagirem com elas. Essa funcionalidade adiciona uma camada de segurança, auxiliando na tomada de decisões informadas ao navegar na web.

**Figura 58 –** Visualização do *plug-in* no Google Chrome



Fonte: Criação do Autor

Casos de uso

1. URL inválida ou campo vazio

**Figura 59 –** Caso *plug-in* com campo vazio



Fonte: Criação do Autor

1. Em *loading*

**Figura 60 –** Caso *plug-in* analisando URL



Fonte: Criação do Autor

1. URL legítima

**Figura 61 –** Caso *plug-in* análise legitima



Fonte: Criação do Autor

1. Possível URL de *phishing*

**Figura 62 –** Caso *plug-in* análise *phishing*



Fonte: Criação do Autor

1. ARQUITETURA, PROCESSAMENTO e ARMAZENAMENTO

A arquitetura de processamento de dados adotada neste trabalho é baseada em um fluxo de ingestão e transformação que utiliza a AWS para realizar a extração e processamento dos dados de partidas do jogo League of Legends a partir da API pública disponibilizada pela Riot Games. O fluxo de dados é dividido em três camadas principais: Raw, Silver e Gold, com cada uma dessas camadas desempenhando um papel crucial na preparação dos dados para análise e modelagem preditiva.

1. Camada Raw: Ingestão de Dados no Bucket S3

A primeira etapa da arquitetura envolve a ingestão de dados da API pública da Riot Games para um bucket S3. A API da Riot retorna os dados no formato JSON, contendo uma ampla gama de informações sobre partidas de League of Legends. Cada registro JSON é detalhado, incluindo dados como o tempo de duração da partida, jogadores, estatísticas de desempenho, itens comprados, entre outros. Em média, cada resposta JSON pode conter informações sobre múltiplos aspectos de uma partida, como:

Informações Gerais da Partida: Identificadores da partida, tipo de partida, duração, data e hora de início e término, entre outros.

Jogadores: Dados sobre os jogadores, como identificador de summoner, campeões jogados, roles, status de morte, entre outros.

Estatísticas de Desempenho: Quantidade de kills, deaths, assists, farm (CS), dano causado, ouro ganho, entre outras métricas relevantes.

Itens: Informações sobre os itens adquiridos por cada jogador durante a partida, incluindo o tempo de aquisição e os efeitos dos itens.

Dado que a API da Riot gera grandes volumes de dados, cada resposta JSON pode variar de centenas a milhares de kilobytes, dependendo do número de jogadores e detalhes da partida.

Esses arquivos JSON são armazenados de maneira bruta no bucket S3, representando a camada Raw. A principal função dessa camada é servir como repositório de dados originais, sem qualquer transformação ou limpeza, para garantir que todos os dados estejam acessíveis para futuras necessidades de análise ou reprocessamento.

2. Camada Silver: Processamento e Armazenamento em CSV

Quando um novo arquivo JSON é adicionado ao bucket S3 na camada Raw, uma função AWS Lambda é acionada para processar o arquivo e transformá-lo em um formato mais adequado para análise, armazenando-o em uma estrutura de dados tabular na camada Silver.

A função Lambda extrai os dados relevantes do JSON e realiza as seguintes transformações para gerar o arquivo CSV:

Seleção de Campos: Dado o volume e a complexidade dos dados JSON, a Lambda filtra apenas as informações necessárias para análise de partidas e treinamento de modelos de Machine Learning. Os principais campos selecionados para a camada Silver são:

- Identificador da Partida: Utilizado para fazer referência a cada jogo.

- Jogadores e Identificadores de Summoner: Informações sobre os jogadores, permitindo mapear as estatísticas de cada um.

- Dados de Desempenho: Estatísticas de cada jogador durante a partida, como kills, deaths, assists, farm (CS), dano causado, ouro ganho e outros indicadores de performance.

- Itens Comprados: Informações sobre os itens adquiridos, especialmente aqueles que influenciam diretamente no desempenho do jogador.

- Resultado da Partida: Vencedor, duração da partida e outros indicadores de vitória ou derrota.

Transformação de Dados: Além da seleção de campos, a função Lambda realiza algumas transformações importantes, como:

- Normalização de Dados: Ajustes nos valores para garantir que estejam dentro de uma escala consistente, como normalizar o tempo de partida ou a quantidade de ouro.

- Conversão para Formato CSV: A transformação dos dados em um formato tabular (.csv) facilita a análise posterior e a integração com ferramentas de BI e modelos de Machine Learning.

A camada Silver contém agora um arquivo CSV que representa um conjunto de dados estruturados, prontos para serem usados em análises e modelos preditivos. Esse arquivo pode ser acessado diretamente ou transferido para outras ferramentas de processamento de dados, como pipelines de treinamento de Machine Learning ou sistemas de BI.

3. Preparação para Machine Learning e BI/Analytics

Após a transformação para a camada Silver, os dados são organizados de forma que possam ser facilmente consumidos por modelos de Machine Learning para análise preditiva, como classificação ou regressão, e por ferramentas de BI para análise de performance de jogadores ou times.

Modelos de Machine Learning: A partir da camada Silver, pode-se treinar modelos preditivos para prever o desempenho dos jogadores, identificar padrões de vitória ou derrota com base nas estatísticas da partida, ou até mesmo prever quais itens têm mais impacto no desempenho de jogadores em diferentes cenários.

BI/Analytics: A camada Silver também serve como base para análise exploratória dos dados e para a geração de relatórios de BI, como dashboards interativos que mostram tendências de performance, itens mais utilizados, taxas de vitória, entre outras métricas relevantes.

Essa arquitetura de ingestão e processamento de dados, utilizando AWS Lambda, S3 e CSV, proporciona uma estrutura escalável e eficiente para lidar com grandes volumes de dados, extraindo, processando e armazenando informações de partidas de League of Legends de maneira organizada e acessível. A camada Raw serve como repositório seguro e imutável, enquanto a camada Silver estrutura os dados em um formato pronto para análise e aprendizado de máquina. Esse fluxo facilita a criação de modelos preditivos e análises de BI/Analytics, com o objetivo de entender melhor os fatores que influenciam o desempenho no jogo e proporcionar insights valiosos para os stakeholders.

**Figura 63** – Arquitetura POC

Diagrama

Descrição gerada automaticamente

Fonte: Criação do Autor

**Figura 64** – Arquitetura para coleta de dados

Diagrama

Descrição gerada automaticamente

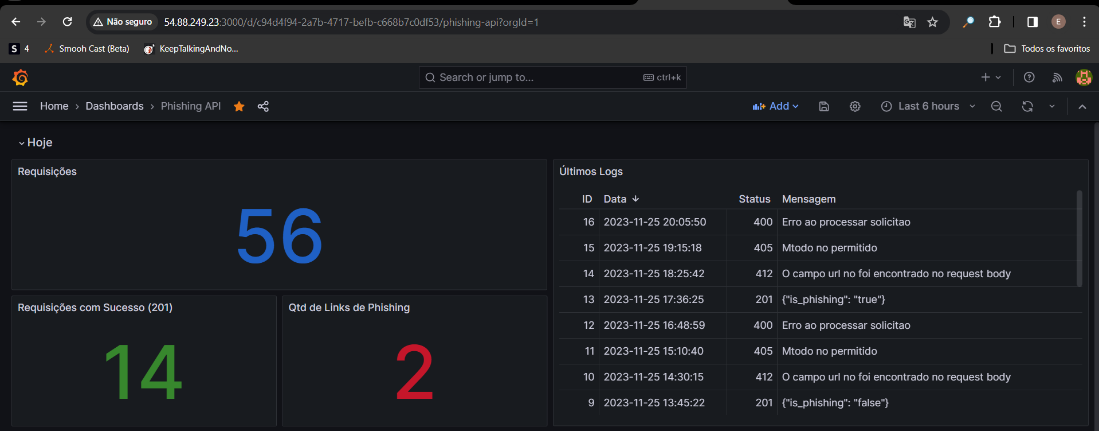
Fonte: Criação do Autor

Nessa base é armazenado o ID da consulta, Data e hora que foi realizada, *Status* HTTP retornado e Mensagem de resposta da API.

Estes *logs* armazenados, são visualizados no Grafana, ferramenta de visualização de dados, *logs* e métricas, muito utilizada no ramo de observabilidade, pois é *opensource*.

No *dashboard* criado, foi dividido em 2 partes, a primeira contendo os *Logs* do dia atual, mostrando o número de requisições, além de detalhar os últimos *logs* recebidos (figura 65).

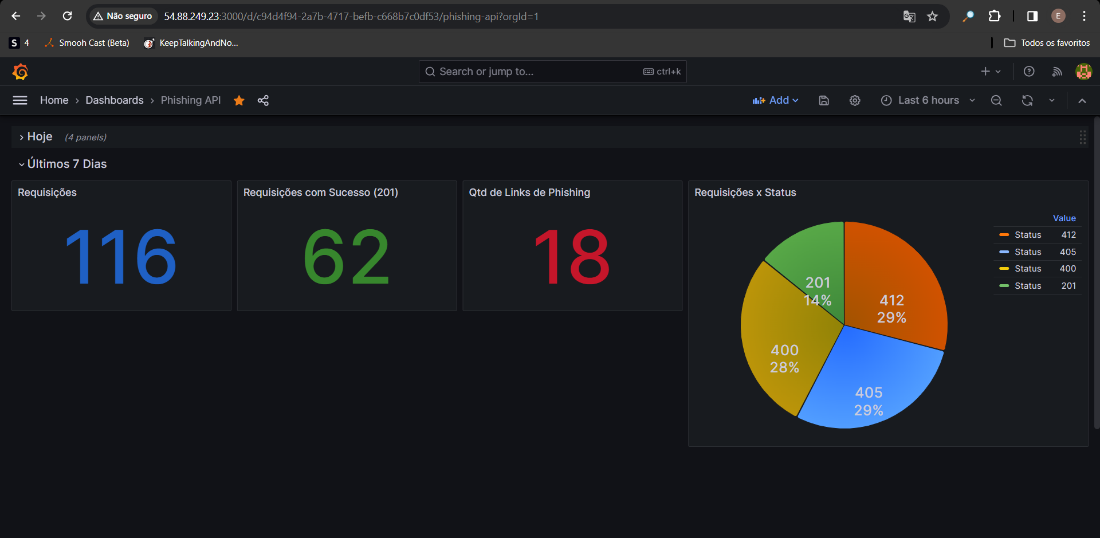
**Figura 65** – *Dashboard* do dia atual no Grafana



Fonte: Criação do Autor

Na segunda parte, há as requisições recebidas nos últimos 7 dias divididas em *big numbers*, além de dividir pelos s*tatus* obtidos (figura 66).

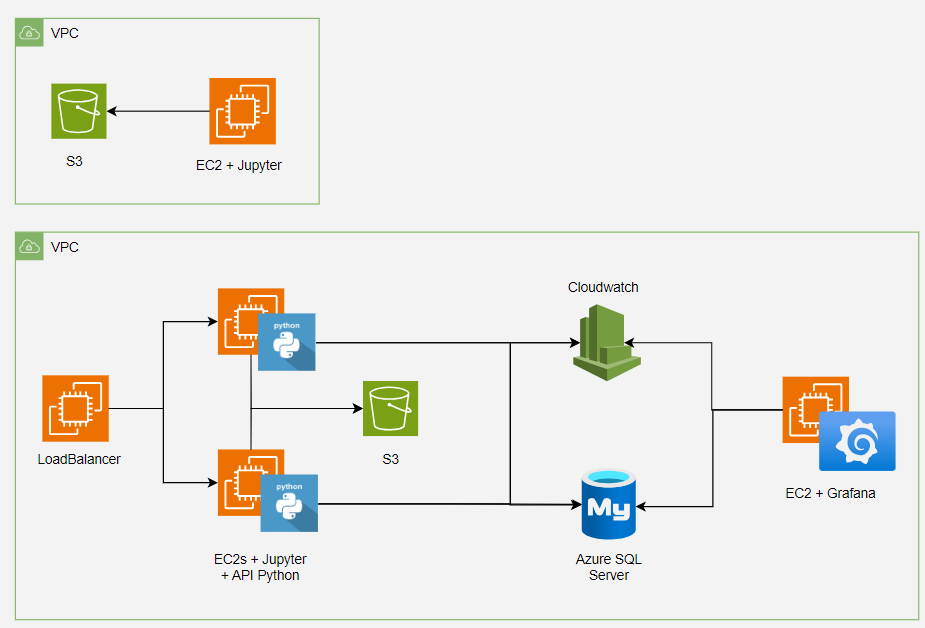
**Figura 66** – *Dashboard* da semana no Grafana



Fonte: Criação do Autor

Após os testes iniciais, seria possível implementar uma arquitetura maior e mais complexa, que seria separada em dois ambientes, um para Homologação e outro para Produção, conforme a imagem seguinte (figura 67).

**Figura 67** – Arquitetura Final



Fonte: Criação do Autor

**Ambiente de Homologação:**

* 1 Máquina Virtual (AmazonEC2);
* 1 *Bucket* de Armazenamento de Objetos (Amazon S3).

Composta por uma máquina virtual onde seria alocada a nossa aplicação para desenvolvimento e testes.

Todo os dados utilizados pelo *machine learning* da aplicação precisariam estar disponíveis em um ambiente online, acessível e seguro, por isso seria utilizado um *bucket* de armazenamento de objetos.

**Ambiente de Produção:**

* 4 Máquinas Virtuais (Amazon EC2);
* 1 *Bucket* de Armazenamento de Objetos (Amazon S3);
* 1 Banco de Dados SQL Server *(*Azure SQL Server*)*.

Para este segundo ambiente, seria utilizado mais alguns recursos disponíveis para novas funcionalidades do ambiente.

2 máquinas virtuais alocariam a aplicação, assim estaria disponível 2 serviços da aplicação, aumentando a disponibilidade. Para garantir a distribuição e acesso às 2 aplicações em produção, seria configurada uma terceira máquina virtual como balanceadora de carga.

A última máquina virtual do ambiente seria utilizada para observabilidade da aplicação, pois alocaria o Grafana. Pensando na praticidade na gestão da aplicação, o Grafana seria integrado com o serviço CloudWatchda AWS, responsável pela monitoração da infraestrutura (recursos do ambiente) e com o banco de dados SQL Server que seria criado no ambiente *cloud* do Azure da Microsoft, para consulta dos *logs* da nossa aplicação.

1. **CONCLUSÃO** – modelo

O projeto abordou de forma abrangente e proativa a crescente ameaça representada pelos golpes virtuais, especialmente o *phishing*, que tem impactado significativamente empresas e indivíduos. O avanço contínuo da tecnologia ampliou a sofisticação dessas práticas maliciosas, exigindo uma resposta inovadora e adaptável para proteger a segurança cibernética.

A implementação de um sistema de detecção de *phishing* baseado em *machine learning*, desenvolvido em Python, demonstra um compromisso com a defesa proativa contra ameaças cibernéticas. A escolha cuidadosa do modelo de classificação *random forest* agregada com métrica de curva ROC, reflete uma abordagem estratégica para lidar com a abrangência dos dados que serão trabalhados garantindo uma generalização mais eficiente, diminuição de *overfitting*, paralelização, avaliação do modelo e outros diversos benefícios relacionados a eficiência em devolver valor da solução.

Apesar do desenvolvimento inicial com uma análise de PCA para redução de dimensionalidade, optou-se por não a incorporar atualmente, reconhecendo sua desnecessidade para os objetivos específicos do projeto. Essa decisão reflete uma abordagem cuidadosa na escolha das técnicas, garantindo eficácia e relevância, mas mantendo a possibilidade de explorar o PCA em trabalhos futuros para análises mais aprofundadas.

Além disso, a pesquisa não se limitou à abordagem técnica, incorporando uma iniciativa prática com a criação de um *plug-in* para navegadores que atua como um validador de URLs. Essa ferramenta oferece aos usuários a capacidade de verificar a legitimidade dos links antes de acioná-los, proporcionando uma camada adicional de proteção.

A utilização de ambientes em nuvem, como a AWS, demonstra uma preocupação com a escalabilidade e disponibilidade da solução.

Em última análise, o projeto não apenas aborda as ameaças atuais, mas também antecipa desafios futuros, como evidenciado pela consideração das tendências em aprendizado de máquina, *deep fakes* e mineração de dados. A contribuição vai além da proteção contra práticas enganosas, aspirando a promover um ambiente digital mais seguro e confiável para todos os usuários.

REFERÊNCIAS

League of Legends. **Balanceamento de partidas**. Disponível em: < <https://nexus.leagueoflegends.com/pt-br/2018/02/dev-matchmaking-real-talk/>>. Acessado em fevereiro de 2018.

League of Legends. **Partidas ranqueadas, MMR, PDL**. Disponível em: < [https://support-leagueoflegends.riotgames.com/hc/pt-br/articles/4405781372051-MMR-ranque-e-PdL#:~:text=O%20MMR%20(Matchmaking%20Rating)%20é,%2C%20se%20perder%2C%20perde%20posição](https://support-leagueoflegends.riotgames.com/hc/pt-br/articles/4405781372051-MMR-ranque-e-PdL#:~:text=O%20MMR%20(Matchmaking%20Rating)%20%C3%A9,%2C%20se%20perder%2C%20perde%20posi%C3%A7%C3%A3o).>. Acessado em junho de 2024.