PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

Carolina Costa Souza Alves

POKÉMON! Temos Que Pegar!

Belo Horizonte
2022

Carolina Costa Souza Alves

POKÉMON! Temos Que Pegar!

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2022

AGRADECIMENTOS

Primeiramente a Deus, base de conforto e confiança em todas etapas da vida.

Aos meus pais por mostrarem que o estudo nunca é demais, que é de extrema importância em nossa vida e carreira profissional.

Ao meu esposo Leandro, meus filhos Evy e Bry (que ficaram fascinados que a mamãe estava fazendo um trabalho para a escola sobre Pokémon), pela ajuda nas pesquisas, paciência e apoio necessário durante os dias de estudo, passando por minhas crises de ansiedade, pânico e incerteza. Passamos por poucas e boas nos últimos anos, situações nunca imaginadas, vencemos um câncer e covid, mas independente da situação sempre juntos, nunca me abandonaram, sempre me dando a certeza que eu era forte e conseguiria.

Ao meu professor Marco Antônio Lopes que me aguentou bastante, e coloca bastante nisso, durante os meus estudos na faculdade, mostrando que a profissão que escolhi realmente é a minha vocação.

Ao meu professor Sergio Avila, que no ensino médio, viu meu potencial e me mostrou através da matemática que tudo existe uma lógica, foi ali que você conseguiu destravar minha mente e mudou um conceito formado de que eu não seria nada no futuro. Hoje sou uma profissional bem reconhecida graças a você.

A todos os professores e tutores da PUC MINAS, o curso foi simplesmente inovador e desafiador, muito obrigada por todo o aprendizado.

SUMÁRIO

1. Introdução

1.1. Contextualização

Minha paixão pelo mundo Pokémon veio da febre mundial com as aventuras do Ash e Pikachu em série animada passada em canal aberto nos anos 90. Desde lá Pikachu, Charmander, Squirtle e Bulbasaur são meus queridinhos e me acompanham até a vida adulta (na verdade estão sobre o meu monitor nesse exato momento). Estava quebrando tanto a cabeça de que tema usar para o TCC quando olhei para eles e pensei: Porque não? Porque não comentar de uma fase da minha infância que lembro com tanta felicidade e carinho. Dessa forma resolvi utilizar os dados primários e informações desse mundo para montar esse trabalho.

A ideia dos Pokémon veio quando criança Satoshi Tajiri colecionava pequenos insetos, que ele chamava de monstros de bolso (poket monsters). Em 1996, junto com Ken Sugimori e a Nintendo, criaram os jogos Pokémon Red e Pokémon Green, onde existiam 151 monstrinhos que deveriam ser capturados em aventuras no Game Boy. Devido ao sucesso dos jogos, foi criada uma versão anime que hoje possui mais de 18 temporadas, além de filmes e especiais. Atualmente os jogos estão na 8ª geração, totalizando 905 monstrinhos de bolso. Esse número está para aumentar com a chegada da 9ª geração (Pokémon Scarlet & Violet) no desde ano (2022).

O objetivo desse mundo é derrotar os líderes e se tornar um grande mestre Pokémon, para isso os Pokémons lutam uns contra os outros com ordem dos seus treinadores, este por sua vez precisa confiar em sua estratégia para utilizar de forma correta a habilidade e força de cada Pokémon. Cada treinador Pokémon, possui uma Pokédex, conhecida também como Poké-Agenda, esse equipamento serve como enciclopédia virtual portátil, pois contem a informação de todos os Pokémons conhecidos no mundo Pokémon. A enciclopédia do Pokédex se divide em cores: amarelo, azul, branco, marrom, verde, rosa, cinza, vermelho, roxo, amarelo e preto.

Os Pokémons podem possuir dois tipos, que são: água, pedra, dragão, gelo, inseto, normal, terra, sombrio, veneno, voador, lutador, fantasma, elétrico, psíquico, fogo, fada, grama, ferro. Normalmente seus tipos estão ligados aos seus estilos de vida.

Eles vêm de alguns grupos de ovos: monstro, inseto, voador, campo, desconhecido, fada, grama, água 1, água 2, água 3, bípedes, inorgânico, indeterminado, dito e dragão. O ovo é relacionado ao Pokémon e um fator determinante na criação desde. Cada Pokémon pode ter até dois grupos de ovos.

Ao longo da vida um Pokémon pode adquirir o conhecimento de vários ataques, conhecido aqui como habilidades. Porém o Pokémon já nasce com uma habilidade definida para sua segurança e ataque. Alguns Pokémon possuem ataques ocultos que são descobertos pelos seus treinadores durante o processo de treino ou desafiando outros Pokémons.

Existem Pokémons lendários e místicos, existe uma grande discussão que os dois são iguais, porém não são. Os lendários são aqueles que as pessoas sabem que existem, possuem habilidades diferentes e extraordinários. Os místicos são lendas, as pessoas contam sobre a sua existência sem saberem se realmente existem. No jogo os místicos são mais fracos que os lendários, em contrapartida eles são muito mais raros e difíceis de obter. Existe uma teoria que os Pokémons lendários não são diferentes em poder quanto os outros Pokémons, sendo assim, não vale muita a pena a dificuldade de procurá-los, mas esse é um ponto que desejo me aprofundar.

Como nível de análise estática usaremos os dados dos Card Games, lançado em 1996, pois neles temos as informações (variáveis) como: sexo, ataques, habilidades, fraquezas, resistência, entre outras informações. Tentando encontrar relações e agrupamentos por critérios.

Vai ser grande a emoção

Pokémon!

Temos que pegar!

1.2. O problema proposto

Sem uma contextualização dos dados não é possível fazer uma análise de forma efetiva, por isso descrever o contexto e desenvolve-lo é tão importante.

Podemos considerar que o ciclo de um projeto de análise de dados é como um ciclo de resolução utilizado por um combatente durante uma batalha de Card Games:

(Demanda) Para poder ganhar a batalha preciso: resgatar todas as minhas cartas prêmio, ou, derrotar todos os Pokémons do meu adversário, ou, fazê-lo gastar todo seu monte antes da partida acabar.

(Entendendo) Meu deck de 60 cartas possui boas cartas? Está completo e com cartas atuais? Quais as melhores cartas para ganhar um jogo?

(Coleta de dados) Das cartas que estão na mão preciso de Pokémons básicos, se não tiver precisarei embaralhar as cartas novamente, dando a oportunidade do meu oponente pegar mais uma carta e sair a frente no jogo.

(Análise dos dados) Baseado nos valores de batalha identificar qual será o Pokémon ativo e os que estarão no banco de ataque.

(Comunicação de resultados) Retirar da mão e montar uma boa jogada na mesa mantendo as cartas viradas para baixo.

(Feedback) Iniciar o ataque virando o Pokémon ativo e os que estão no banco.

A técnica que mais me chamou a atenção foi a dos 5-Ws, pois é uma técnica que podemos utilizar para todos os seguimentos, ou seja, independente dos dados que estaremos analisando é possível entender e desenvolver o que vai ser feito baseado nessa técnica. Por isso, e para confirmar o que estou falando (utilizar em todos os seguimentos) é que resolvi utiliza-la aqui.

(Why?) As estratégias para uma batalha são baseadas em valores de estatísticas dos Pokémons que cada treinador possui, aquele que conseguir Pokémons com melhores estatísticas podem e usa-los ao seu favor para ganha o jogo. Então pretendo usar técnicas de analise de dados para identificar Pokémons que tenham maiores ganhos aos seus técnicos.

(Who?) Os dados como: número, nome, cor, entre outras informações, são da marca registrada Pokémon, que hoje é de propriedade da Nintendo. Mesmo os níveis de ataque, defesa, entre outros, utilizados nos Card Games são baseados nos níveis de cada Pokémon nos jogos da franquia.

(What?) Analisar estatisticamente as informações usadas de cada Pokémon, usar técnicas de análise de dados e machine learning, afim de encontrar relações e agrupamento entre os valores para identificar melhores Pokémons a serem pegos em jogos ou comprados pelos jogadores de Card Games.

(Where?) Os resultados dessas analises poderão ser utilizados por qualquer jogador do mundo Pokemon, seja Card Games, jogos de console, entre outros, como por exemplo: Pokémon Go. Em âmbito nacional como internacional, pois os dados numéricos não interferem/mudam de acordo com a nacionalidade.

(When?) Será analisado todas as gerações dos Pokémons desde outubro de 1995 até novembro de 2019.

Para um jogador Pokémon é muito importante entender e saber sobre o universo Pokémon, pois com isso ele conseguirá saber quantos Pokémons ele ainda precisa obter/comprar e se a gama dele está diversificada de uma forma melhor para atacar os seus oponentes, por isso a importância de algumas estáticas, como por exemplo: qual o tipo de Pokémon que possui o maior poder.

Ponto importante a destacar: sou muito apaixonada pelos jogos eletrônicos desde a primeira versão (red and blue), porém nunca joguei Card Games, conheço as cartas, já acompanhei partidas e estudei vários tutoriais para montar esse TCC, mas não posso me declarar uma conhecedora assídua de Card Games, verdadeiros jogadores de Card Games Pokémon podem argumentar que esse TCC não é a forma correta de fazer a análise para o ganho efetivo de uma batalha, pois vários fatores podem influenciar no jogo, como cartas especiais, que podem por exemplo: confundir, adormecer e paralisar o Pokémon adversário. Além de cartas de poções que podem aumentar poderes ou devolver vida aos Pokémons machucados.

2. Coleta de Dados

Iniciei a pesquisa procurando pela palavra POKEMON no Google Dataset Search (https://datasetsearch.research.google.com/). Foram encontrados vários datasets, com as informações distribuídas. Alguns dados, por serem novos, não existiam em nenhum dataset, por isso busquei as informações no site da própria marca (pokemon.com/br/pokedex/) e em um site de enciclopédia do universo Pokémon (bulbagarden.net/).

Ao total para o processo de tratamento serão utilizados 11 datasets, com os seguintes layouts:

Arquivos	pokemons_criacao.csv	
Coluna	**Descrição**	**Tipo**
Geracao	Número da geração do Pokémon	Numérico
DataCriacao	Data que a geração foi lançada	Texto

Arquivos	pokemons_1.csv pokemons_2.csv pokemons_3.csv pokemons_4.csv pokemons_5a8.csv	
Coluna	**Descrição**	**Tipo**
Numero	Número do Pokémon na Pokédex	Numérico
Nome	Nome do Pokémon	Texto
Geracao	Número da geração do Pokémon	Texto

Arquivos	pokemons_batalha.csv	
Coluna	**Descrição**	**Tipo**
Numero	Número do Pokémon na Pokédex	Numérico
HP	Pontos de vida do Pokémon, quanto maior for, mais tempo o Pokémon é capaz de aguentar os ataques antes de desmaiar e deixar o combate	Texto
Ataque	Valor de ataque, físico, do Pokémon, quanto maior	Texto

	for mais danos físicos irá causar no oponente	
Defesa	Valor de defesa do Pokémon, para ataques físicos, quanto maior mais o Pokémon conseguirá se defender dos ataques do oponente	Texto
AtaqueEspecial	Valor de ataque especial, sem contato físico, do Pokémon, quanto maior for mais danos físicos irá causar no oponente	Texto
DefesaEspecial	Valor de defesa especial, usado para ataques especiais, quanto maior mais o Pokémon conseguirá se defender dos ataques do oponente	Texto
Rapidez	Valor de rapidez do Pokémon, quanto maior mais chances ele tem de fugir e atacar o adversário	Texto

Arquivos	pokemons_habilidades.csv	
Coluna	**Descrição**	**Tipo**
Numero	Número do Pokémon na Pokédex	Numérico
Habilidade1	Primeiro ataque do Pokémon	Texto
Habilidade2	Segundo ataque do Pokémon	Texto
HabilidadeOculta	Habilidade oculta do Pokémon	Texto

Arquivos	pokemons_informacoes_1a6.csv pokemons_informacoes_7a8.csv	
Coluna	**Descrição**	**Tipo**
Numero	Número do Pokémon na Pokédex	Numérico
Nome	Nome do Pokémon	Texto
Cor	Cor do Pokémon na Pokédex	Texto
OvoTipo1	Primeiro tipo de ovo do Pokémon	Texto
OvoTipo2	Segundo tipo do ovo do Pokémon	Texto
Tipo1	Primeiro tipo do Pokémon	Texto
Tipo2	Segundo tipo do Pokémon	Texto
Tamanho	Tamanho do Pokémon em metros	Texto
Peso	Peso do Pokémon em quilos (kg)	Texto
Lendario	Verdadeiro para Pokémons lendários	Texto

Mistico	Verdadeiro para Pokémons místicos	Texto

Depois de todo tratamento, o Dataset final, que será utilizado para analise desse TCC, ficou da seguinte maneira:

Coluna	Descrição	Tipo
Numero	Número do Pokémon na Pokédex, esse é o identificador único do Pokémon, em todas as bases, revistas, enciclopédias esse valor sempre corresponderá ao mesmo Pokémon.	Index
Nome	Nome do Pokémon, dado único para todos os Pokémons, só é possível existir um único nome para cada identificador único, dessa forma entendemos que a coluna NUMERO é 1=1 da coluna NOME	Object
Geracao	Número da geração do Pokémon, cada identificador único (Numero), possui somente uma geração	Int64
Cor	Cor do Pokémon na Pokédex, possuindo também somente uma única cor de classificação para cada Pokémon	Object
OvoTipo1	Primeiro tipo de ovo do Pokémon	Object
OvoTipo2	Segundo tipo do ovo do Pokémon	Object
Tipo1	Primeiro tipo do Pokémon	Object
Tipo2	Segundo tipo do Pokémon	Object
Tamanho	Tamanho do Pokémon em metros	Float64
Peso	Peso do Pokémon em quilos (kg)	Float64
Lendario	Verdadeiro para Pokémons lendários	Bool
Mistico	Verdadeiro para Pokémons místicos	Bool
Habilidade1	Primeiro ataque do Pokémon	Object
Habilidade2	Segundo ataque do Pokémon	Object
HabilidadeOculta	Habilidade oculta do Pokémon	Object
Sexo	Que pode ser definido em F (feminino), M (masculino) e I (indefinido)	Object

Level	Número correspondente ao nível atual do Pokémon, esse valor é corresponde as estáticas abaixo.	Float64
HP	Pontos de vida do Pokémon, quanto maior for, mais tempo o Pokémon é capaz de aguentar os ataques antes de desmaiar e deixar o combate	Float64
Ataque	Valor de ataque, físico, do Pokémon, quanto maior for mais danos físicos irá causar no oponente	Float64
Defesa	Valor de defesa do Pokémon, para ataques físicos, quanto maior mais o Pokémon conseguirá se defender dos ataques do oponente	Float64
AtaqueEspecial	Valor de ataque especial, sem contato físico, do Pokémon, quanto maior for mais danos físicos irá causar no oponente	Float64
DefesaEspecial	Valor de defesa especial, usado para ataques especiais, quanto maior mais o Pokémon conseguirá se defender dos ataques do oponente	Float64
Rapidez	Valor de rapidez do Pokémon, quanto maior mais chances ele tem de fugir e atacar o adversário	Float64
Total	Somatória de todas as estatísticas do Pokemon (HP, Ataque, Defesa, AtaqueEspecial, DefesaEspecial e Rapidez)	Float64
Total Ataque	Somatória dos ataques do Pokémon (Ataque, AtaqueEspecial)	Float64
TotalDefesa	Somatória das defesas do Pokémon (Defesa, DefesaEspecial)	Float64

3. Processamento/Tratamento de Dados

Para tal utilizei três etapas muito conhecidas na análise de dados: data cleaning (limpeza dos dados), data fusion (mescla de dados) e feature engineering (novas informações), utilizando a linguagem Python com a biblioteca Pandas. Segue abaixo código ipynb que foi exportado via ferramenta Pandoc através da linha de comando: *pandoc analise.ipynb -s -o export.docx*.

IMPORT DOS CSVS

pokemons_criacao.csv

```
pokemons_criacao = pd.read_csv('pokemons_criacao.csv',encoding='utf-8',sep=';')
len(pokemons_criacao)
```

```
8
```

pokemons_1.csv

```
pokemons_1 = pd.read_csv('pokemons_1.csv',encoding='utf-8',sep=';')
len(pokemons_1)
```

```
151
```

pokemons_2.csv

```
pokemons_2 = pd.read_csv('pokemons_2.csv',encoding='utf-8',sep=';')
len(pokemons_2)
```

```
100
```

pokemons_3.csv

```
pokemons_3 = pd.read_csv('pokemons_3.csv',encoding='utf-8',sep=';')
len(pokemons_3)
```

```
135
```

pokemons_4.csv

```
pokemons_4 = pd.read_csv('pokemons_4.csv',encoding='utf-8',sep=';')
len(pokemons_4)
```

```
107
```

pokemons_5a8.csv

```
pokemons_5a8 = pd.read_csv('pokemons_5a8.csv',encoding='utf-8',sep=';')
len(pokemons_5a8)
```

```
412
```

pokemons_batalha.csv

```
pokemons_batalha = pd.read_csv('pokemons_batalha.csv',encoding='utf-8',sep=';')
len(pokemons_batalha)
```

```
4887
```

pokemons_habilidades.csv

```
pokemons_habilidades = pd.read_csv('pokemons_habilidades.csv',encoding='utf-8',sep=';')
len(pokemons_habilidades)
```

```
905
```

pokemons_informacoes_1a6.csv

```
pokemons_informacoes_1a6 = pd.read_csv('pokemons_informacoes_1a6.csv',encoding='utf-8',sep=';')
len(pokemons_informacoes_1a6)
```

```
721
```

pokemons_informacoes_7a8.csv

```
pokemons_informacoes_7a8 = pd.read_csv('pokemons_informacoes_7a8.csv',encoding='utf-8',sep=';')
len(pokemons_informacoes_7a8)
```

```
184
```

Todos os arquivos foram carregados e a quantidade de linhas validadas.

DATA CLEANING

pokemons_criacao

```
pokemons_criacao[pokemons_criacao.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Geracao, DataCriacao]
Index: []
```

```
print(pokemons_criacao.dtypes)
```

```
Geracao         int64
DataCriacao    object
dtype: object
```

```
pokemons_criacao.DataCriacao = pd.to_datetime(pokemons_criacao.DataCriacao, format='%d/%m/%Y')
print(pokemons_criacao.dtypes)
```

```
Geracao                int64
DataCriacao    datetime64[ns]
dtype: object
```

A data foi convertida para datatime, pois estava com o tipo object.

pokemons_1

```
pokemons_1[pokemons_1.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Numero, Nome, Geracao]
Index: []
```

```
pokemons_1.groupby(['Geracao']).count()
```

```
         Numero  Nome
Geracao
1           151   151
```

```
print(pokemons_1.dtypes)
```

```
Numero      int64
Nome       object
Geracao     int64
dtype: object
```

```
pokemons_1 = pokemons_1.set_index('Numero')
pokemons_1.head()
```

```
                Nome  Geracao
Numero
1        Bulbasaur        1
2          Ivysaur        1
3         Venusaur        1
4       Charmander        1
5       Charmeleon        1
```

Não existem valores nulos, os tipos estão conforme necessidade e os valores estão corretos.

pokemons_2

```
pokemons_2[pokemons_2.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Numero, Nome, Geracao]
Index: []
```

```
pokemons_2.loc[pokemons_2.Geracao == '2ª','Geracao'] = '2'
pokemons_2.groupby(['Geracao']).count()
```

```
         Numero  Nome
Geracao
2           100   100
```

```
print(pokemons_2.dtypes)
```

```
Numero      int64
Nome       object
Geracao    object
dtype: object
```

```
pokemons_2['Geracao'] = pd.to_numeric(pokemons_2['Geracao'])
print(pokemons_2.dtypes)
```

```
Numero      int64
Nome       object
Geracao     int64
dtype: object
```

```
pokemons_2 = pokemons_2.set_index('Numero')
pokemons_2.head()
```

Numero	Nome	Geracao
152	Chikorita	2
153	Bayleef	2
154	Meganium	2
155	Cyndaquil	2
156	Quilava	2

A GERACAO estava com o valor errado, foi modificado de 2ª para 2 e seu tipo alterado para inteiro.

pokemons_3

```
pokemons_3[pokemons_3.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Numero, Nome, Geracao]
Index: []
```

```
pokemons_3.groupby(['Geracao']).count()
```

Geracao	Numero	Nome
3	135	135

```
print(pokemons_3.dtypes)
```

```
Numero      int64
Nome       object
Geracao     int64
dtype: object
```

```
pokemons_3 = pokemons_3.set_index('Numero')
pokemons_3.head()
```

Numero	Nome	Geracao
252	Treecko	3
253	Grovyle	3
254	Sceptile	3
255	Torchic	3
256	Combusken	3

Não existem valores nulos, os tipos estão conforme necessidade e os valores estão corretos.

pokemons_4

```
pokemons_4[pokemons_4.isnull().T.any()]
```

```
        Empty DataFrame
        Columns: [Numero, Nome, Geracao]
        Index: []
```

```
pokemons_4.groupby(['Geracao']).count()
```

```
                 Numero  Nome
        Geracao
        4th         107   107
```

```
pokemons_4.loc[pokemons_4.Geracao == '4th','Geracao'] = '4'
pokemons_4.groupby(['Geracao']).count()
```

```
                 Numero  Nome
        Geracao
        4           107   107
```

```
print(pokemons_4.dtypes)
```

```
        Numero      int64
        Nome       object
        Geracao    object
        dtype: object
```

```
pokemons_4['Geracao'] = pd.to_numeric(pokemons_4['Geracao'])
print(pokemons_4.dtypes)
```

```
        Numero      int64
        Nome       object
        Geracao     int64
        dtype: object
```

```
pokemons_4 = pokemons_4.set_index('Numero')
pokemons_4.head()
```

```
                    Nome  Geracao
        Numero
        387      Turtwig        4
        388       Grotle        4
        389     Torterra        4
        390     Chimchar        4
        391     Monferno        4
```

A GERACAO estava com o valor errado, foi modificado de 4th para 4 e seu tipo alterado para inteiro.

pokemons_5a8

```
pokemons_5a8[pokemons_5a8.isnull().T.any()]
```

```
        Empty DataFrame
        Columns: [Numero, Nome, Geracao]
        Index: []
```

```
pokemons_5a8.groupby(['Geracao']).count()
```

```
                 Numero  Nome
        Geracao
        5           156   156
        6            72    72
        7            88    88
        8            96    96
```

```
print(pokemons_5a8.dtypes)
```

```
        Numero      int64
        Nome       object
```

```
Geracao     int64
dtype: object
```

```
pokemons_5a8 = pokemons_5a8.set_index('Numero')
pokemons_5a8.head()
```

```
                Nome  Geracao
Numero
494       Victini        5
495         Snivy        5
496       Servine        5
497     Serperior        5
498         Tepig        5
```

Não existem valores nulos, os tipos estão conforme necessidade e os valores estão corretos.

pokemons_batalha

```
pokemons_batalha[pokemons_batalha.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Numero, Sexo, Level, HP, Ataque, Defesa, AtaqueEspecial, DefesaEspecial, Rapidez]
Index: []
```

```
print(pokemons_batalha.dtypes)
```

```
Numero             int64
Sexo              object
Level              int64
HP                 int64
Ataque             int64
Defesa             int64
AtaqueEspecial     int64
DefesaEspecial     int64
Rapidez            int64
dtype: object
```

```
pokemons_batalha = pokemons_batalha.set_index('Numero')
pokemons_batalha.head()
```

```
       Sexo  Level   HP  Ataque  Defesa  AtaqueEspecial  DefesaEspecial  \
Numero
1         M      1   45      49      49              65              65
1         M     50  152     111     111             128             128
1         M    100  294     216     216             251             251
1         F      1   45      49      49              65              65
1         F     50  152     111     111             128             128



        Rapidez
Numero
1            45
1           106
1           207
1            45
1           106
```

Não existem valores nulos, os tipos estão conforme necessidade e os valores estão corretos.

pokemons_habilidades

```
pokemons_habilidades[pokemons_habilidades.isnull().T.any()]
```

```
     Numero     Habilidade1 Habilidade2 HabilidadeOculta
0         1        overgrow         NaN      chlorophyll
1         2        overgrow         NaN      chlorophyll
2         3        overgrow         NaN      chlorophyll
3         4           blaze         NaN      solar-power
4         5           blaze         NaN      solar-power
..      ...             ...         ...              ...
894     895     dragons-maw         NaN              NaN
895     896  chilling-neigh         NaN              NaN
896     897      grim-neigh         NaN              NaN
897     898         unnerve         NaN              NaN
902     903        pressure         NaN     poison-touch

[418 rows x 4 columns]
```

```
pokemons_habilidades.loc[pokemons_habilidades.Habilidade2.isnull(),'Habilidade2'] = ''
pokemons_habilidades.loc[pokemons_habilidades.HabilidadeOculta.isnull(),'HabilidadeOculta'] = ''
pokemons_habilidades.isnull().sum()
```

```
Numero             0
Habilidade1        0
Habilidade2        0
HabilidadeOculta   0
dtype: int64
```

```
print(pokemons_habilidades.dtypes)
```

```
Numero              int64
Habilidade1        object
Habilidade2        object
HabilidadeOculta   object
dtype: object
```

```
pokemons_habilidades['Habilidade1'] = pokemons_habilidades['Habilidade1'].str.capitalize()
pokemons_habilidades['Habilidade2'] = pokemons_habilidades['Habilidade2'].str.capitalize()
pokemons_habilidades['HabilidadeOculta'] = pokemons_habilidades['HabilidadeOculta'].str.capitalize()

pokemons_habilidades = pokemons_habilidades.set_index('Numero')
pokemons_habilidades.head()
```

```
       Habilidade1 Habilidade2 HabilidadeOculta
Numero
1         Overgrow                  Chlorophyll
2         Overgrow                  Chlorophyll
3         Overgrow                  Chlorophyll
4            Blaze                  Solar-power
5            Blaze                  Solar-power
```

Existiam valores nulos nas colunas HABILIDADE2 e HABILIDADEOCULTA, para não haver problemas posteriores na análise dos dados, esses valores foram alterados para '' (vazio), pois não são informações obrigatórias. Foi colocado letra maiúscula na primeira letra de todas as habilidades.

pokemons_informacoes_1a6

```
pokemons_informacoes_1a6[pokemons_informacoes_1a6.isnull().T.any()]
```

```
     Numero        Nome       Cor      OvoTipo1 OvoTipo2     Tipo1     Tipo2  \
3         4  Charmander  vermelho       monstro   dragao      fogo       NaN
4         5  Charmeleon  vermelho       monstro   dragao      fogo       NaN
6         7    Squirtle      azul       monstro   agua_1      agua       NaN
7         8   Wartortle      azul       monstro   agua_1      agua       NaN
8         9   Blastoise      azul       monstro   agua_1      agua       NaN
```

```
..      ...         ...       ...           ...      ...       ...       ...
716     717     Yveltal  vermelho  desconhecido      NaN   sombrio    voador
717     718     Zygarde     verde  desconhecido      NaN    dragao  fantasma
718     719     Diancie      rosa  desconhecido      NaN     pedra      fada
719     720       Hoopa      roxo  desconhecido      NaN  psiquico  fantasma
720     721   Volcanion    marrom  desconhecido      NaN      fogo      agua

     Tamanho   Peso  Lendario  Mistico
3        0.6    8.5         0        0
4        1.1   19.0         0        0
6        0.5    9.0         0        0
7        1.0   22.5         0        0
8        1.6   85.5         0        0
..       ...    ...       ...      ...
716      5.8  203.0         1        0
717      5.0  305.0         1        0
718      0.7    8.8         0        1
719      0.5    9.0         0        1
720      1.7  195.0         0        1

[618 rows x 11 columns]
```

```
pokemons_informacoes_1a6.loc[pokemons_informacoes_1a6.OvoTipo2.isnull(),'OvoTipo2'] = ''
pokemons_informacoes_1a6.loc[pokemons_informacoes_1a6.Tipo2.isnull(),'Tipo2'] = ''
pokemons_informacoes_1a6.isnull().sum()
```

```
Numero      0
Nome        0
Cor         0
OvoTipo1    0
OvoTipo2    0
Tipo1       0
Tipo2       0
Tamanho     0
Peso        0
Lendario    0
Mistico     0
dtype: int64
```

```
print(pokemons_informacoes_1a6.dtypes)
```

```
Numero        int64
Nome         object
Cor          object
OvoTipo1     object
OvoTipo2     object
Tipo1        object
Tipo2        object
Tamanho     float64
Peso        float64
Lendario      int64
Mistico       int64
dtype: object
```

```
pokemons_informacoes_1a6.groupby(['Lendario']).count().Numero
```

```
Lendario
0    683
1     38
Name: Numero, dtype: int64
```

```
pokemons_informacoes_1a6.groupby(['Mistico']).count().Numero
```

```
Mistico
0    705
1     16
Name: Numero, dtype: int64
```

```
pokemons_informacoes_1a6.iloc[:, 9:] = pokemons_informacoes_1a6.iloc[:, 9:].astype(bool)
print(pokemons_informacoes_1a6.dtypes)
```

```
Numero        int64
Nome         object
```

```
Cor          object
OvoTipo1     object
OvoTipo2     object
Tipo1        object
Tipo2        object
Tamanho     float64
Peso        float64
Lendario       bool
Mistico        bool
dtype: object
```

```
pokemons_informacoes_1a6['Cor'] = pokemons_informacoes_1a6['Cor'].str.capitalize()
pokemons_informacoes_1a6['OvoTipo1'] = pokemons_informacoes_1a6['OvoTipo1'].str.capitalize()
pokemons_informacoes_1a6['OvoTipo2'] = pokemons_informacoes_1a6['OvoTipo2'].str.capitalize()
pokemons_informacoes_1a6['Tipo1'] = pokemons_informacoes_1a6['Tipo1'].str.capitalize()
pokemons_informacoes_1a6['Tipo2'] = pokemons_informacoes_1a6['Tipo2'].str.capitalize()

pokemons_informacoes_1a6 = pokemons_informacoes_1a6.set_index('Numero')
pokemons_informacoes_1a6.head()
```

Numero	Nome	Cor	OvoTipo1	OvoTipo2	Tipo1	Tipo2	Tamanho	Peso	Lendario	Mistico
1	Bulbasaur	Verde	Monstro	Grama	Grama	Veneno	0.7	6.9	False	False
2	Ivysaur	Verde	Monstro	Grama	Grama	Veneno	1.0	13.0	False	False
3	Venusaur	Verde	Monstro	Grama	Grama	Veneno	2.0	100.0	False	False
4	Charmander	Vermelho	Monstro	Dragao	Fogo		0.6	8.5	False	False
5	Charmeleon	Vermelho	Monstro	Dragao	Fogo		1.1	19.0	False	False

Existiam valores nulos nas colunas OVOTIPO2 e TIPO2, para não haver problemas posteriores na análise dos dados, esses valores foram alterados para '' (vazio), pois não são informações obrigatórias. As informações das colunas LEGENDARIO e MISTICO são booleanas, porém vieram com os valores 1 e 0 (int), dessa forma foi efetuada a conversão das informações para TRUE e FALSE. Foi colocado letra maiúscula na primeira letra dos objects.

pokemons_informacoes_7a8

```
pokemons_informacoes_7a8[pokemons_informacoes_7a8.isnull().T.any()]
```

	Numero	Nome	Cor	OvoTipo1	OvoTipo2	Tipo1	Tipo2	Tamanho	Peso	Lendario	Mistico
0	722	Rowlet	marrom	voador	NaN	grama	voador	0,3	1,5	F	F
1	723	Dartrix	marrom	voador	NaN	grama	voador	0,7	16	F	F
2	724	Decidueye	marrom	voador	NaN	grama	fantasma	1,6	36,6	F	F
3	725	Litten	vermelho	campo	NaN	fogo	NaN				
4	726	Torracat	vermelho	campo	NaN	fogo	NaN				
..	...	...	...	...	...	...	...				
179	901	Ursaluna	marrom	campo	NaN	terra	normal				
180	902	Basculegion	verde	agua_2	NaN	agua	fantasma				
181	903	Sneasler	azul	campo	NaN	lutador	veneno				
182	904	Overqwil	preto	agua_2	NaN	sombrio	veneno				
183	905	Enamorus	rosa	desconhecido	NaN	fada	voador				

```
3       0,4   4,3        F       F
4       0,7    25        F       F
..      ...   ...      ...     ...
179     2,4   290        F       F
180       3   110        F       F
181     1,3    43        F       F
182     2,5  60,5        F       F
183     1,6    48        T       F

[158 rows x 11 columns]
```

```
pokemons_informacoes_7a8.loc[pokemons_informacoes_7a8.OvoTipo2.isnull(),'OvoTipo2'] = ''
pokemons_informacoes_7a8.loc[pokemons_informacoes_7a8.Tipo2.isnull(),'Tipo2'] = ''
pokemons_informacoes_7a8.isnull().sum()
```

```
Numero      0
Nome        0
Cor         0
OvoTipo1    0
OvoTipo2    0
Tipo1       0
Tipo2       0
Tamanho     0
Peso        0
Lendario    0
Mistico     0
dtype: int64
```

```
print(pokemons_informacoes_7a8.dtypes)
```

```
Numero       int64
Nome        object
Cor         object
OvoTipo1    object
OvoTipo2    object
Tipo1       object
Tipo2       object
Tamanho     object
Peso        object
Lendario    object
Mistico     object
dtype: object
```

```
pokemons_informacoes_7a8['Tamanho'] = pd.to_numeric(pokemons_informacoes_7a8['Tamanho'].str.replace(',','.'),
downcast="float")
pokemons_informacoes_7a8['Peso'] = pd.to_numeric(pokemons_informacoes_7a8['Peso'].str.replace(',','.'),
downcast="float")
pokemons_informacoes_7a8['Lendario'] =
pd.to_numeric(pokemons_informacoes_7a8['Lendario'].replace(['F','T'],['0','1'])).astype(bool)
pokemons_informacoes_7a8['Mistico'] =
pd.to_numeric(pokemons_informacoes_7a8['Mistico'].replace(['F','T'],['0','1'])).astype(bool)
print(pokemons_informacoes_7a8.dtypes)
```

```
Numero       int64
Nome        object
Cor         object
OvoTipo1    object
OvoTipo2    object
Tipo1       object
Tipo2       object
Tamanho    float32
Peso       float32
Lendario      bool
Mistico       bool
dtype: object
```

```
pokemons_informacoes_7a8['Cor'] = pokemons_informacoes_7a8['Cor'].str.capitalize()
pokemons_informacoes_7a8['OvoTipo1'] = pokemons_informacoes_7a8['OvoTipo1'].str.capitalize()
pokemons_informacoes_7a8['OvoTipo2'] = pokemons_informacoes_7a8['OvoTipo2'].str.capitalize()
pokemons_informacoes_7a8['Tipo1'] = pokemons_informacoes_7a8['Tipo1'].str.capitalize()
pokemons_informacoes_7a8['Tipo2'] = pokemons_informacoes_7a8['Tipo2'].str.capitalize()
pokemons_informacoes_7a8 = pokemons_informacoes_7a8.set_index('Numero')
pokemons_informacoes_7a8.head()
```

```
             Nome       Cor OvoTipo1 OvoTipo2  Tipo1     Tipo2  Tamanho  \
Numero
722        Rowlet    Marrom   Voador           Grama    Voador      0.3
723       Dartrix    Marrom   Voador           Grama    Voador      0.7
724     Decidueye    Marrom   Voador           Grama  Fantasma      1.6
725        Litten  Vermelho    Campo            Fogo                0.4
726      Torracat  Vermelho    Campo            Fogo                0.7

              Peso  Lendario  Mistico
Numero
722       1.500000     False    False
723      16.000000     False    False
724      36.599998     False    False
725       4.300000     False    False
726      25.000000     False    False
```

Existiam valores nulos nas colunas OVOTIPO2 e TIPO2, para não haver problemas posteriores na análise dos dados, esses valores foram alterados para '' (vazio), pois não são informações obrigatórias. As informações das colunas TAMANHO e PESO vieram com , (virgula) ao invés de . (ponto), foi efetuada o replace de virgula para ponto e posteriormente a conversão do datatype para float. As informações das colunas LEGENDARIO e MISTICO são booleanas, porém vieram com os valores T e F (object), dessa forma foi efetuada a conversão das informações para TRUE e FALSE. Foi colocado letra maiúscula na primeira letra dos objects.

DATA FUSION

pokemons_geracao

```
pokemons_geracao =
pd.concat([pokemons_1,pokemons_2,pokemons_3,pokemons_4,pokemons_5a8]).sort_values(by='Numero')
pokemons_geracao.head()
```

```
                Nome  Geracao
Numero
1          Bulbasaur        1
2            Ivysaur        1
3           Venusaur        1
4         Charmander        1
5         Charmeleon        1
```

Os dataframes POKEMONS_1, POKEMONS_2, POKEMONS_3, POKEMONS_4 e POKEMONS_5a8, possuem os mesmos dados, porém estão em arquivos diferentes, pois seus dados vieram de 5 fontes diferentes.

pokemons_informacoes

```
pokemons_informacoes = pd.concat([pokemons_informacoes_1a6,pokemons_informacoes_7a8]).sort_values(by='Numero')
pokemons_informacoes.head()
```

```
                Nome       Cor OvoTipo1 OvoTipo2  Tipo1   Tipo2  Tamanho   Peso  \
Numero
1          Bulbasaur     Verde  Monstro    Grama  Grama  Veneno      0.7    6.9
2            Ivysaur     Verde  Monstro    Grama  Grama  Veneno      1.0   13.0
3           Venusaur     Verde  Monstro    Grama  Grama  Veneno      2.0  100.0
4         Charmander  Vermelho  Monstro   Dragao   Fogo              0.6    8.5
5         Charmeleon  Vermelho  Monstro   Dragao   Fogo              1.1   19.0
```

```
        Lendario  Mistico
Numero
1          False    False
2          False    False
3          False    False
4          False    False
5          False    False
```

Os dataframes POKEMONS_INFORMACOES_1a6 e POKEMONS_INFORMACOES_7a8, possuem os mesmos dados, porém estão em arquivos diferentes, pois seus dados vieram de 2 fontes diferentes.

pokemons_dados

```
pokemons_dados = pd.merge(pokemons_geracao, pokemons_informacoes[['Cor','OvoTipo1','OvoTipo2','Tipo1','Tipo2','Tamanho','Peso','Lendario','Mistico']], on=["Numero"], how="inner")
pokemons_dados.head()
```

```
                Nome  Geracao       Cor OvoTipo1 OvoTipo2  Tipo1   Tipo2  \
Numero
1          Bulbasaur        1     Verde  Monstro    Grama  Grama  Veneno
2            Ivysaur        1     Verde  Monstro    Grama  Grama  Veneno
3           Venusaur        1     Verde  Monstro    Grama  Grama  Veneno
4         Charmander        1  Vermelho  Monstro   Dragao   Fogo
5         Charmeleon        1  Vermelho  Monstro   Dragao   Fogo

        Tamanho   Peso  Lendario  Mistico
Numero
1           0.7    6.9     False    False
2           1.0   13.0     False    False
3           2.0  100.0     False    False
4           0.6    8.5     False    False
5           1.1   19.0     False    False
```

```
pokemons_dados = pokemons_dados.join(pokemons_habilidades,on='Numero')
pokemons_dados[pokemons_dados.isnull().T.any()]
```

```
Empty DataFrame
Columns: [Nome, Geracao, Cor, OvoTipo1, OvoTipo2, Tipo1, Tipo2, Tamanho, Peso, Lendario, Mistico, Habilidade1, Habilidade2, HabilidadeOculta]
Index: []
```

```
pokemons_dados.head()
```

```
                Nome  Geracao       Cor OvoTipo1 OvoTipo2  Tipo1   Tipo2  \
Numero
1          Bulbasaur        1     Verde  Monstro    Grama  Grama  Veneno
2            Ivysaur        1     Verde  Monstro    Grama  Grama  Veneno
3           Venusaur        1     Verde  Monstro    Grama  Grama  Veneno
4         Charmander        1  Vermelho  Monstro   Dragao   Fogo
5         Charmeleon        1  Vermelho  Monstro   Dragao   Fogo

        Tamanho   Peso  Lendario  Mistico Habilidade1 Habilidade2  \
Numero
1           0.7    6.9     False    False    Overgrow
2           1.0   13.0     False    False    Overgrow
3           2.0  100.0     False    False    Overgrow
4           0.6    8.5     False    False       Blaze
5           1.1   19.0     False    False       Blaze

       HabilidadeOculta
Numero
1          Chlorophyll
2          Chlorophyll
3          Chlorophyll
```

```
4           Solar-power
5           Solar-power
```

Sabemos que temos 905 Pokémons dessa forma podemos concluir, que após análise de nulos e quantidade de dados, o dataframe POKEMONS está com os dados de todos os Pokémons existentes até o momento.

pokemons_stats

```
pokemons_stats = pd.concat([pokemons_batalha]).sort_values(by='Numero')
pokemons_stats.head()
```

```
       Sexo  Level   HP  Ataque  Defesa  AtaqueEspecial  DefesaEspecial  \
Numero
1         M      1   45      49      49              65              65
1         M     50  152     111     111             128             128
1         M    100  294     216     216             251             251
1         F      1   45      49      49              65              65
1         F     50  152     111     111             128             128

        Rapidez
Numero
1            45
1           106
1           207
1            45
1           106
```

Com a analise acima, conseguimos identificar que existem sexos de Pokémons inexistentes na base de batalhas, isso porque existem Pokémons que só são do sexo feminino ou masculino, por isso, foi montada a tabela POKEMON_STATS com somente os dados existentes/verdadeiros.

pokemons

```
pokemons =
(pd.merge(pokemons_dados,pokemons_criacao,on='Geracao').set_index(pokemons_dados.index)).join(pokemons_s
tats,on='Numero')
pokemons.head()
```

```
              Nome  Geracao    Cor OvoTipo1 OvoTipo2  Tipo1   Tipo2  Tamanho  \
Numero
1       Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1       Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1       Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1       Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1       Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7

        Peso  Lendario  ...  HabilidadeOculta DataCriacao Sexo Level   HP  \
Numero                  ...
1        6.9     False  ...       Chlorophyll  1996-02-27    M     1   45
1        6.9     False  ...       Chlorophyll  1996-02-27    M    50  152
1        6.9     False  ...       Chlorophyll  1996-02-27    M   100  294
1        6.9     False  ...       Chlorophyll  1996-02-27    F     1   45
1        6.9     False  ...       Chlorophyll  1996-02-27    F    50  152

       Ataque  Defesa  AtaqueEspecial  DefesaEspecial  Rapidez
Numero
1          49      49              65              65       45
1         111     111             128             128      106
1         216     216             251             251      207
```

```
1          49      49             65             65       45
1         111     111            128            128      106

[5 rows x 23 columns]
```

O dataframe pokemons contém todas as correções e as informações que serão necessárias para desenvolver a análise de dados.

FEATURE ENGINEERING

Total

```
columns_sum = ['HP','Ataque', 'Defesa', 'AtaqueEspecial', 'DefesaEspecial', 'Rapidez']
pokemons.loc[:,'Total'] = pokemons[columns_sum].sum(numeric_only=True, axis=1)
```

O Total é uma combinação linear, sendo a soma de todas as outras estasticas de combate.

TotalAtaque

```
columns_sum = ['Ataque','AtaqueEspecial']
pokemons.loc[:,'TotalAtaque'] = pokemons[columns_sum].sum(numeric_only=True, axis=1)
```

O TotalAtaque é uma combinação linear da soma dos ataques que podem ser usados no combate.

TotalDefesa

```
columns_sum = ['Defesa','DefesaEspecial']
pokemons.loc[:,'TotalDefesa'] = pokemons[columns_sum].sum(numeric_only=True, axis=1)
```

O TotalDefesa é uma combinação linear da soma das defesas que podem ser usados no combate.

```
pokemons.head()
```

```
                Nome  Geracao    Cor OvoTipo1 OvoTipo2  Tipo1   Tipo2  Tamanho  \
Numero
1          Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1          Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1          Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1          Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7
1          Bulbasaur        1  Verde  Monstro    Grama  Grama  Veneno      0.7

           Peso  Lendario  ...  Level   HP Ataque Defesa AtaqueEspecial  \
Numero                     ...
1           6.9     False  ...      1   45     49     49             65
1           6.9     False  ...     50  152    111    111            128
1           6.9     False  ...    100  294    216    216            251
```

```
1        6.9     False  ...      1   45     49     49            65
1        6.9     False  ...     50  152    111    111           128

       DefesaEspecial  Rapidez  Total  TotalAtaque  TotalDefesa
Numero
1                  65       45    318          114          114
1                 128      106    736          239          239
1                 251      207   1435          467          467
1                  65       45    318          114          114
1                 128      106    736          239          239

[5 rows x 26 columns]
```

4. Análise e Exploração dos Dados

A melhor forma de apresentar uma análise de dados é através de gráficos, pois podemos possuir um volume de dados tão grande que olhar diretamente em uma tabela não é muito intuitivo.

A primeira etapa de uma análise de dados é entender a necessidade do solicitante e identificar/definir com ele quais são as informações/respostas que ele deseja obter com os dados recebidos. Pensando nessa linha de raciocínio, levantei algumas perguntas que um líder/treinador pode fazer para identificar possíveis pontos de atenção no momento de adquirir um Pokémon para seu jogo ou Card Game.

ANALISE DE DADOS

GRAFICOS DE BARRAS

- 1 – Qual geração possui a maior parte dos Pokémons?

```
pokemons_a1 = pokemons[['Nome','Geracao']].drop_duplicates()
graf_geracao = pokemons_a1.groupby('Geracao')['Nome'].count()
```

- 2 – Qual a cor predominante da Pokédex?

```
pokemons_a2 = pokemons[['Nome','Cor']].drop_duplicates()
graf_cor = pokemons_a2.groupby('Cor')['Nome'].count()
```

- 3 – Qual o tipo predominante de Pokémons?

```
pokemons_a3_1 = pokemons[['Nome','Tipo1']].rename(columns={'Tipo1': 'Tipo'}).drop_duplicates()
pokemons_a3_2 = pokemons[['Nome','Tipo2']].rename(columns={'Tipo2': 'Tipo'}).drop_duplicates()
graf_tipo = pd.concat([pokemons_a3_1,pokemons_a3_2]).sort_values(by='Numero').groupby('Tipo')['Nome'].count()
graf_tipo.drop([''], inplace=True)
```

- 4 – Qual o tipo de Pokémon que possuí o maior poder?

```
pokemons_a4_1 = pokemons[['Total','Tipo1']].rename(columns={'Tipo1': 'Tipo'}).drop_duplicates()
pokemons_a4_2 = pokemons[['Total','Tipo2']].rename(columns={'Tipo2': 'Tipo'}).drop_duplicates()
graf_poder_tipo = pd.concat([pokemons_a4_1,pokemons_a4_2]).sort_values(by='Numero').groupby('Tipo')['Total'].count()
graf_poder_tipo.drop([''], inplace=True)
```

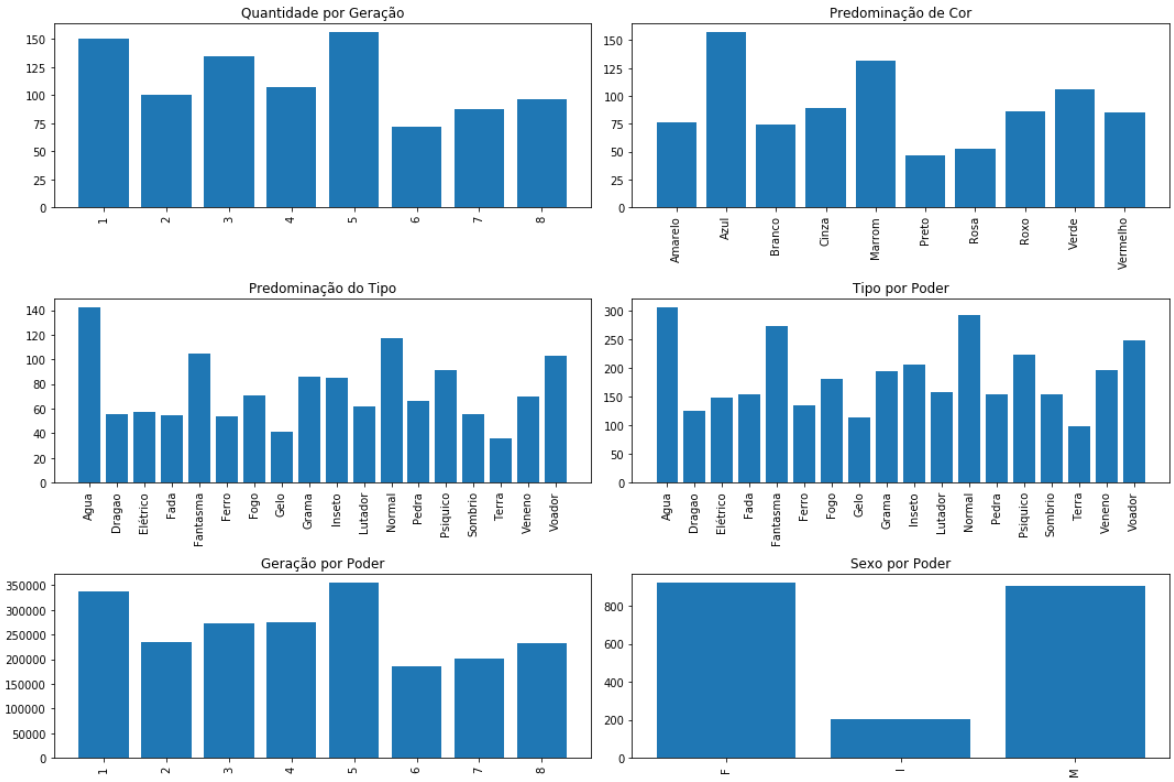
- 5 - Qual geração de Pokemon possuí o maior poder?

```
pokemons_a5 = pokemons[['Total','Geracao']].drop_duplicates()
graf_poder_geracao = pokemons_a5.groupby('Geracao')['Total'].sum()
```

- 6 - Qual o Pokemon mais forte pelo sexo?

```
pokemons_a6 = pokemons[['Total','Sexo']].drop_duplicates()
graf_sexo = pokemons_a6.groupby('Sexo')['Total'].count()
```

```
plt.figure(figsize=(15,10))

plt.subplot(3,2,1)
plt.bar(graf_geracao.index,graf_geracao.values)
plt.title('Quantidade por Geração')
plt.xticks(rotation=90)

plt.subplot(3,2,2)
plt.bar(graf_cor.index,graf_cor.values)
plt.title('Predominação de Cor')
plt.xticks(rotation=90)

plt.subplot(3,2,3)
plt.bar(graf_tipo.index,graf_tipo.values)
plt.title('Predominação do Tipo')
plt.xticks(rotation=90)

plt.subplot(3,2,4)
plt.bar(graf_poder_tipo.index,graf_poder_tipo.values)
plt.title('Tipo por Poder')
plt.xticks(rotation=90)

plt.subplot(3,2,5)
plt.bar(graf_poder_geracao.index,graf_poder_geracao.values)
plt.title('Geração por Poder')
plt.xticks(rotation=90)

plt.subplot(3,2,6)
plt.bar(graf_sexo.index,graf_sexo.values)
plt.title('Sexo por Poder')
plt.xticks(rotation=90)

plt.tight_layout()
```



Com a análise dos gráficos acima, podemos indicar que se o treinador procurar por Pokémons da 5º geração do tipo água do sexo masculino ou feminino, ele terá mais chances de encontrar Pokémons com maiores poderes. Para analisar melhor esse achado, vamos continuar explorando com perguntas a 5ª geração.

GRAFICO DE PIZZA

- 7 – Qual a porcentagem de Pokémons do tipo água que possuem duas habilidades?

```
total_hab_1 = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua'))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
total_hab_2 = pokemons[['Nome']].loc[(pokemons['Habilidade1'].str.len() > 1) & (pokemons['Habilidade2'].str.len() < 1) & (pokemons['Geracao'] == 5)].rename(columns={'Nome': 'Hab'}).drop_duplicates().count()
total_hab_3 = pokemons[['Nome']].loc[(pokemons['Habilidade1'].str.len() > 1) & (pokemons['Habilidade2'].str.len() > 1) & (pokemons['Geracao'] == 5)].rename(columns={'Nome': 'Hab'}).drop_duplicates().count()
total_hab = pd.concat([total_hab_2,total_hab_3])
graf_hab = (total_hab / total_hab_1.values) * 100
```

- 8 – Qual a porcentagem de Pokémons do tipo água que possuem habilidade oculta?

```
total_ocu_1 = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua'))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
total_ocu_2 = pokemons[['Nome']].loc[(pokemons['HabilidadeOculta'].str.len() < 1) & (pokemons['Geracao'] == 5)].rename(columns={'Nome': 'Oculta'}).drop_duplicates().count()
total_ocu = pd.concat([total_ocu_1,total_ocu_2])
graf_ocu = (total_ocu / total_ocu_1.values) * 100
```

- 9 - Qual a porcentagem de Pokémons por sexo?

```
total = pokemons[['Sexo']].loc[(pokemons['Geracao'] == 5)].count()
total_sexo = pd.value_counts((pokemons[['Sexo']].loc[(pokemons['Geracao'] == 5)]).values.flatten())
frequencia = (total_sexo / total.values) * 100
```

- 10 – Temos mais Pokémons místicos ou lendários?

```
total = pokemons[['Nome']].loc[(pokemons['Geracao'] == 5)].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
lendario = pd.value_counts((pokemons[['Lendario']].loc[(pokemons['Geracao'] == 5 & pokemons['Lendario'])].rename(columns={'Lendario': 'L'})).values.flatten())
mistico = pd.value_counts((pokemons[['Mistico']].loc[(pokemons['Geracao'] == 5 & pokemons['Mistico'])].rename(columns={'Mistico': 'M'})).values.flatten())
pokemons_lm = pd.concat([total,lendario,mistico])
frequencia_lm = (pokemons_lm / total.values) * 100
```

- 11 – Quantos Pokémons do tipo água são lendários ou místicos?

```
total = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua'))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
lendario_5 = pd.value_counts((pokemons[['Lendario']].loc[((pokemons['Geracao'] == 5) & (pokemons['Lendario']) & (pokemons['Tipo1'] == 'Agua'))].rename(columns={'Lendario': 'L'})).values.flatten())
if (lendario_5.count() == 0) : lendario_5.loc['True'] = 0
mistico_5 = pd.value_counts((pokemons[['Mistico']].loc[((pokemons['Geracao'] == 5) & (pokemons['Mistico']) & (pokemons['Tipo1'] == 'Agua'))].rename(columns={'Mistico': 'M'})).values.flatten())
if (mistico_5.count() == 0) : mistico_5.loc['True'] = 0
pokemons_lm_5 = pd.concat([total,lendario_5,mistico_5])
frequencia_lm_5 = (pokemons_lm_5 / total.values) * 100
```

- 12 – Quantos Pokémons do tipo água possuímos na 5ª geração?

```
total = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
agua = pd.value_counts((pokemons[['Tipo1']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua'))].values.flatten()))
pokemons_agua = pd.concat([total,agua])
frequencia_agua = (pokemons_agua / total.values) * 100
```

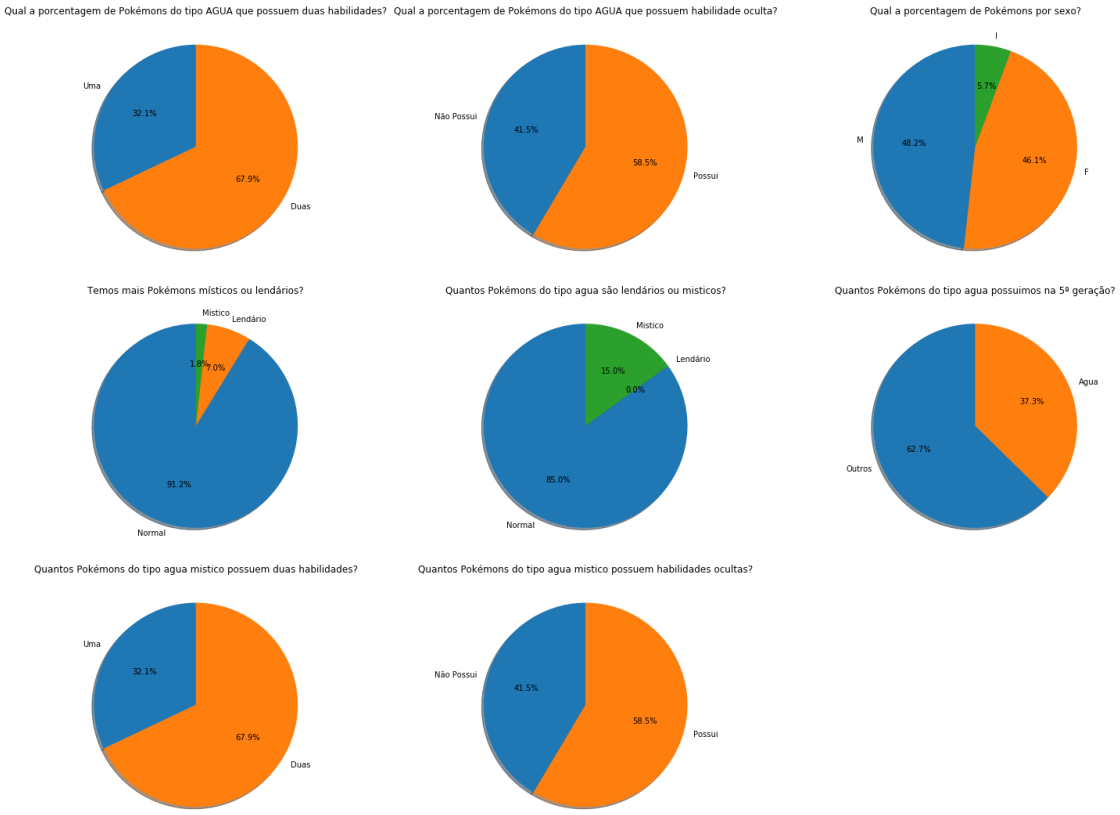
- 13 – Quantos Pokémons do tipo água mistico possuem duas habilidades?

```
m_total_hab_1 = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua') &
(pokemons['Mistico']))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
m_total_hab_2 = pokemons[['Nome']].loc[(pokemons['Habilidade1'].str.len() > 1) & (poke-
mons['Habilidade2'].str.len() < 1) & (pokemons['Geracao'] == 5)].rename(columns={'Nome':
'Hab'}).drop_duplicates().count()
m_total_hab_3 = pokemons[['Nome']].loc[(pokemons['Habilidade1'].str.len() > 1) & (poke-
mons['Habilidade2'].str.len() > 1) & (pokemons['Geracao'] == 5)].rename(columns={'Nome':
'Hab'}).drop_duplicates().count()
m_total_hab = pd.concat([total_hab_2,total_hab_3])
m_graf_hab = (total_hab / total_hab_1.values) * 100
```

- 14 – Quantos Pokémons do tipo água místico possuem habilidades ocultas?

```
m_total_ocu_1 = pokemons[['Nome']].loc[((pokemons['Geracao'] == 5) & (pokemons['Tipo1'] == 'Agua') &
(pokemons['Mistico']))].rename(columns={'Nome': 'Todos'}).drop_duplicates().count()
m_total_ocu_2 = pokemons[['Nome']].loc[(pokemons['HabilidadeOculta'].str.len() < 1) & (poke-
mons['Geracao'] == 5)].rename(columns={'Nome': 'Oculta'}).drop_duplicates().count()
m_total_ocu = pd.concat([total_ocu_1,total_ocu_2])
m_graf_ocu = (total_ocu / total_ocu_1.values) * 100
```

```
plt.figure(figsize=(20,15))

plt.subplot(3,3,1)
plt.pie(graf_hab.values, labels = ['Uma', 'Duas'], autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Qual a porcentagem de Pokémons do tipo AGUA que possuem duas habilidades?')

plt.subplot(3,3,2)
plt.pie(graf_ocu.values, labels = ['Não Possui', 'Possui'], autopct='%1.1f%%', shadow=True, startan-
gle=90)
plt.title('Qual a porcentagem de Pokémons do tipo AGUA que possuem habilidade oculta?')

plt.subplot(3,3,3)
plt.pie(frequencia.values, labels = frequencia.index, autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Qual a porcentagem de Pokémons por sexo?')

plt.subplot(3,3,4)
plt.pie(frequencia_lm.values, labels = ['Normal','Lendário', 'Mistico'], autopct='%1.1f%%', shadow=True,
startangle=90)
plt.title('Temos mais Pokémons místicos ou lendários?')

plt.subplot(3,3,5)
plt.pie(frequencia_lm_5.values, labels = ['Normal','Lendário', 'Mistico'], autopct='%1.1f%%', sha-
dow=True, startangle=90)
plt.title('Quantos Pokémons do tipo agua são lendários ou misticos?')

plt.subplot(3,3,6)
plt.pie(frequencia_agua.values, labels = ['Outros','Agua'], autopct='%1.1f%%', shadow=True, startan-
gle=90)
plt.title('Quantos Pokémons do tipo agua possuimos na 5ª geração?')

plt.subplot(3,3,7)
plt.pie(m_graf_hab.values, labels = ['Uma', 'Duas'], autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Quantos Pokémons do tipo agua mistico possuem duas habilidades?')

plt.subplot(3,3,8)
plt.pie(m_graf_ocu.values, labels = ['Não Possui', 'Possui'], autopct='%1.1f%%', shadow=True, startan-
gle=90)
plt.title('Quantos Pokémons do tipo agua mistico possuem habilidades ocultas?')

plt.tight_layout()
```



A chance de um treinador pegar um Pokémon da 5ª Geração do sexo indeterminado é de menos de 6%. Dentro dessa geração não temos muitos Pokémons místicos ou lendários, pois essa porcentagem é menor que 10%. Possuímos quase 70% dos Pokémons com duas habilidades. Também podemos encontrar nessa geração Pokémon do tipo água místico, em uma porcentagem de 15%, bem desafiadora, mas que seria um excelente achado.

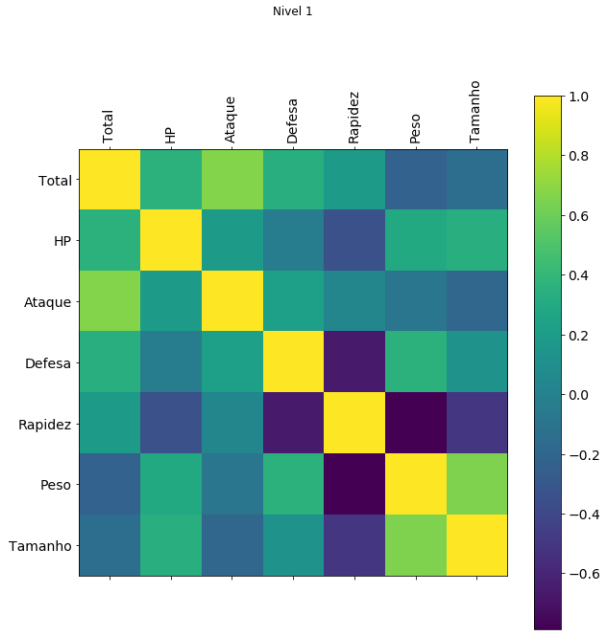
CORRELAÇÃO DE PEARSON

```
pokemon_l1 = (pokemons[['Total','HP','Ataque','Defesa','Rapidez','Peso','Tamanho']].loc[(pokemons['Level'] == 1)]).corr(method='pearson')
pokemon_l1.style.background_gradient(cmap='coolwarm')
```

```
<pandas.io.formats.style.Styler at 0x1ddb6eabb48>
```

```
plt.matshow(pokemon_l1.corr(method='pearson'),fignum=(plt.figure(figsize=(10, 10))).number)
plt.xticks(range(pokemon_l1.select_dtypes(['number']).shape[1]), pokemon_l1.select_dtypes(['number']).columns, fontsize=14, rotation=45)
plt.yticks(range(pokemon_l1.select_dtypes(['number']).shape[1]), pokemon_l1.select_dtypes(['number']).columns, fontsize=14)
(plt.colorbar()).ax.tick_params(labelsize=14)
plt.title('Nivel 1', y=1.3)
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text xticklabel objects>)
```

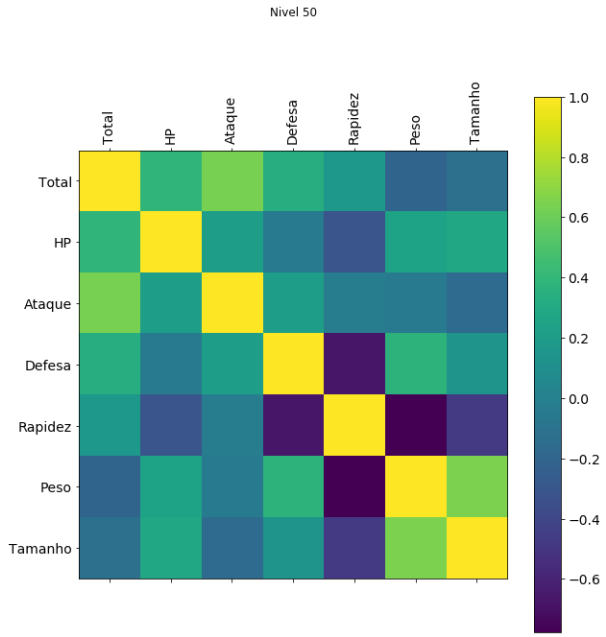



```
pokemon_l50 = (poke-
mons[['Total','HP','Ataque','Defesa','Rapidez','Peso','Tamanho']].loc[(pokemons['Level'] ==
50)]).corr(method='pearson')
pokemon_l50.style.background_gradient(cmap='coolwarm')
```

```
        <pandas.io.formats.style.Styler at 0x1ddb77022c8>
```

```
plt.matshow(pokemon_l50.corr(method='pearson'),fignum=(plt.figure(figsize=(10, 10))).number)
plt.xticks(range(pokemon_l50.select_dtypes(['number']).shape[1]), poke-
mon_l50.select_dtypes(['number']).columns, fontsize=14, rotation=45)
plt.yticks(range(pokemon_l50.select_dtypes(['number']).shape[1]), poke-
mon_l50.select_dtypes(['number']).columns, fontsize=14)
(plt.colorbar()).ax.tick_params(labelsize=14)
plt.title('Nivel 50', y=1.3)
plt.xticks(rotation=90)
```

```
        (array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text xticklabel objects>)
```

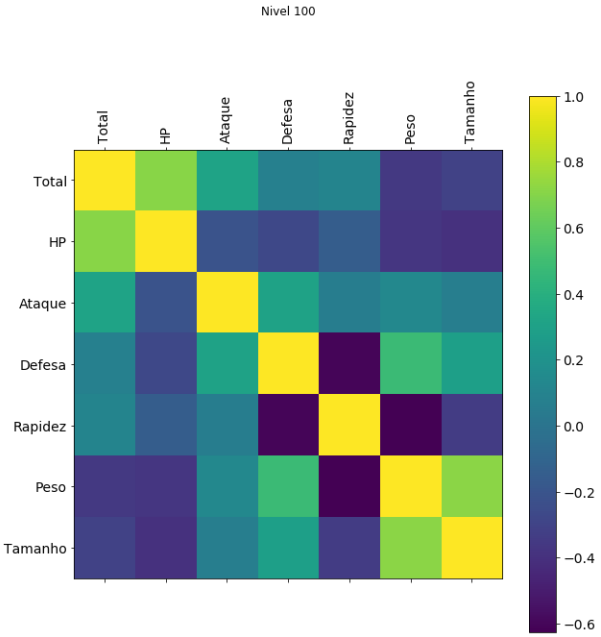


```
pokemon_l100 = (pokemons[['Total','HP','Ataque','Defesa','Rapidez','Peso','Tamanho']].loc[(pokemons['Level'] == 100)]).corr(method='pearson')
pokemon_l100.style.background_gradient(cmap='coolwarm')
```

```
<pandas.io.formats.style.Styler at 0x1ddb76fadc8>
```

```
plt.matshow(pokemon_l100.corr(method='pearson'),fignum=(plt.figure(figsize=(10, 10))).number)
plt.xticks(range(pokemon_l100.select_dtypes(['number']).shape[1]), pokemon_l100.select_dtypes(['number']).columns, fontsize=14, rotation=45)
plt.yticks(range(pokemon_l100.select_dtypes(['number']).shape[1]), pokemon_l100.select_dtypes(['number']).columns, fontsize=14)
(plt.colorbar()).ax.tick_params(labelsize=14)
plt.title('Nivel 100', y=1.3)
plt.xticks(rotation=90)
```

```
(array([0, 1, 2, 3, 4, 5, 6]), <a list of 7 Text xticklabel objects>)
```



Foi calculado o coeficiente de Pearson entre as colunas TOTAL, HP, ATAQUE, DEFESA, RAPIDEZ, PESO e TAMANHO. Com isso foi possível chegar a algumas conclusões. A correlação entre as estatísticas de combate não modifica muito entre os níveis do Pokémon (1, 50 ou 100), dessa forma, podemos entender que quanto maior o nível do Pokémon maior será sua força geral. O TOTAL possui uma correlação maior e positiva entre HP, ATAQUE, DEFESA e RAPIDEZ, isso acontece, pois esse valor é a soma das estatísticas de combate. Outras relações positivas, porém, com menor correlação, aparecem entre ATAQUE e DEFESA, DEFESA e PESO, RAPIDEZ e ATAQUE, dessa forma podemos concluir, por exemplo, que um Pokémon que possua uma RAPIDEZ mais elevada possuirá um ATAQUE mais preciso. Podemos notar também uma correlação forte entre PESO e TAMANHO, porém temos uma correlação muito fraca entre o TAMANHO e PESO do Pokémon com a sua RAPIDEZ, dessa forma, não podemos concluir que essas variáveis possuem um impacto direto na velocidade do Pokémon dentro da batalha.

NUVEM DE PALAVRAS

```
pokemons_palavras = pd.concat([pokemons[['Nome']].rename(columns={'Nome':
'P'}),pokemons[['Cor']].rename(columns={'Cor': 'P'}),pokemons[['OvoTipo1']].rename(columns={'OvoTipo1':
'P'}),pokemons[['OvoTipo2']].rename(columns={'OvoTipo2':
'P'}),pokemons[['Tipo1']].rename(columns={'Tipo1': 'P'}),pokemons[['Tipo2']].rename(columns={'Tipo2':
'P'}),pokemons[['Habilidade1']].rename(columns={'Habilidade1':
'P'}),pokemons[['Habilidade2']].rename(columns={'Habilidade2':
'P'}),pokemons[['HabilidadeOculta']].rename(columns={'HabilidadeOculta': 'P'})])
summary = pokemons_palavras.dropna(subset=['P'], axis=0)['P']
all_summary = " ".join(s for s in summary)
stopwords = set(STOPWORDS)
pikachu_mask = np.array(Image.open("pikachu.png"))
wordcloud = WordCloud(stopwords=stopwords,background_color="black",width=200, height=200,
max_words=200,mask=pikachu_mask,
max_font_size=400,min_font_size=1,collocations=False).generate(all_summary)
fig, ax = plt.subplots(figsize=(10,10))
ax.imshow(wordcloud, interpolation='bilinear')
ax.set_axis_off()
plt.imshow(wordcloud)
```

```
<matplotlib.image.AxesImage at 0x1ddb687c808>
```

Vemos que a palavra que mais se destaca em todo o dataframe é CAMPO, dessa forma, é mais fácil concluirmos que achar um Pokémon do tipo campo (considerando todas as gerações) é mais fácil.

```
pokemons_stopwords5 = pokemons.loc[(pokemons['Geracao'] == 5)]
pokemons_palavras = pd.concat([pokemons_stopwords5[['Nome']].rename(columns={'Nome':
'P'}),pokemons_stopwords5[['Cor']].rename(columns={'Cor':
'P'}),pokemons_stopwords5[['OvoTipo1']].rename(columns={'OvoTipo1':
'P'}),pokemons_stopwords5[['OvoTipo2']].rename(columns={'OvoTipo2':
'P'}),pokemons_stopwords5[['Tipo1']].rename(columns={'Tipo1':
'P'}),pokemons_stopwords5[['Tipo2']].rename(columns={'Tipo2':
'P'}),pokemons_stopwords5[['Habilidade1']].rename(columns={'Habilidade1':
'P'}),pokemons_stopwords5[['Habilidade2']].rename(columns={'Habilidade2':
'P'}),pokemons_stopwords5[['HabilidadeOculta']].rename(columns={'HabilidadeOculta': 'P'})])
summary = pokemons_palavras.dropna(subset=['P'], axis=0)['P']
all_summary = " ".join(s for s in summary)
```

```
stopwords = set(STOPWORDS)
pikachu_mask = np.array(Image.open("charmander.png"))
wordcloud = WordCloud(stopwords=stopwords,background_color="black",width=200, height=200,
max_words=200,mask=pikachu_mask,
max_font_size=400,min_font_size=1,collocations=False).generate(all_summary)
fig, ax = plt.subplots(figsize=(9,9))
ax.imshow(wordcloud, interpolation='bilinear')
ax.set_axis_off()
plt.imshow(wordcloud)
```

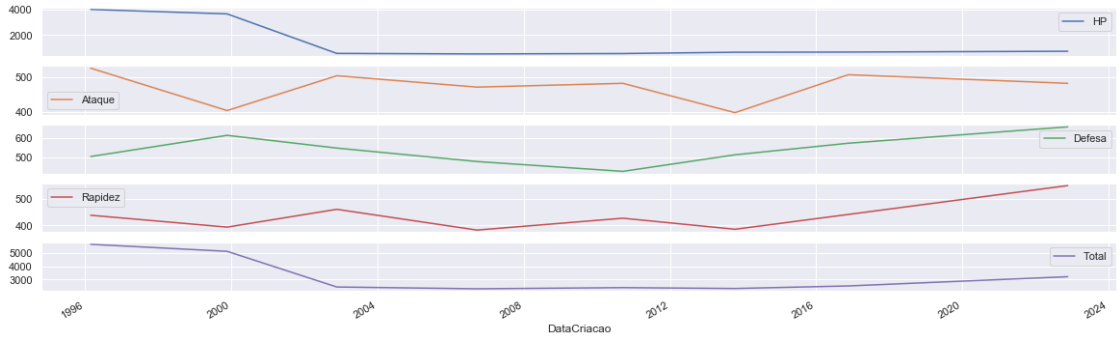
```
<matplotlib.image.AxesImage at 0x1ddb6a8f048>
```

Conforme vimos acima, encontrar Pokémons do tipo água na 5ª Geração teremos chances de conseguir os Pokémons com maiores poderes, porém analisando a nuvem de palavras dessa mesma geração, podemos identifica que não será tão fácil encontrar desse tipo, visto que nessa geração ainda temos como predominância o tipo CAMPO e posteriormente INSETO.

SÉRIE TEMPORAL

```
pokemon_temporal = pokemons[['DataCriacao','HP','Ataque','Defesa','Rapidez','Total']].loc[(pokemons['Level']
== 100)].groupby('DataCriacao').max()
pokemon_temporal
sns.set()
pokemon_temporal.plot(subplots = True, figsize = (20, 6))
```

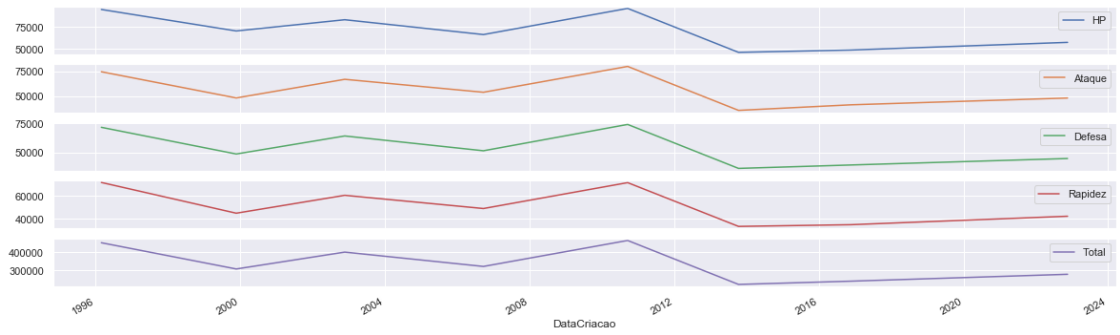
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB6BF1E88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB6BED808>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB69B9A88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB6FC6748>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB6FFC348>],
      dtype=object)
```



Através da analise temporal, dos valores máximos das estatísticas de batalha, conseguimos identificar quanto mais atual o Pokémon mais suas estatísticas de batalha aumentam, porém temos uma exceção que é o HP, onde as primeiras gerações possuíam um valor maior.

```
pokemon_temporal = pokemons[['DataCriacao','HP','Ataque','Defesa','Rapidez','Total']].loc[(pokemons['Level'] == 100)].groupby('DataCriacao').sum()
pokemon_temporal
sns.set()
pokemon_temporal.plot(subplots = True, figsize = (20, 6))
```

```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB692F048>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB71E3208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB720E208>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB76B3A88>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x000001DDB7A1E348>],
      dtype=object)
```



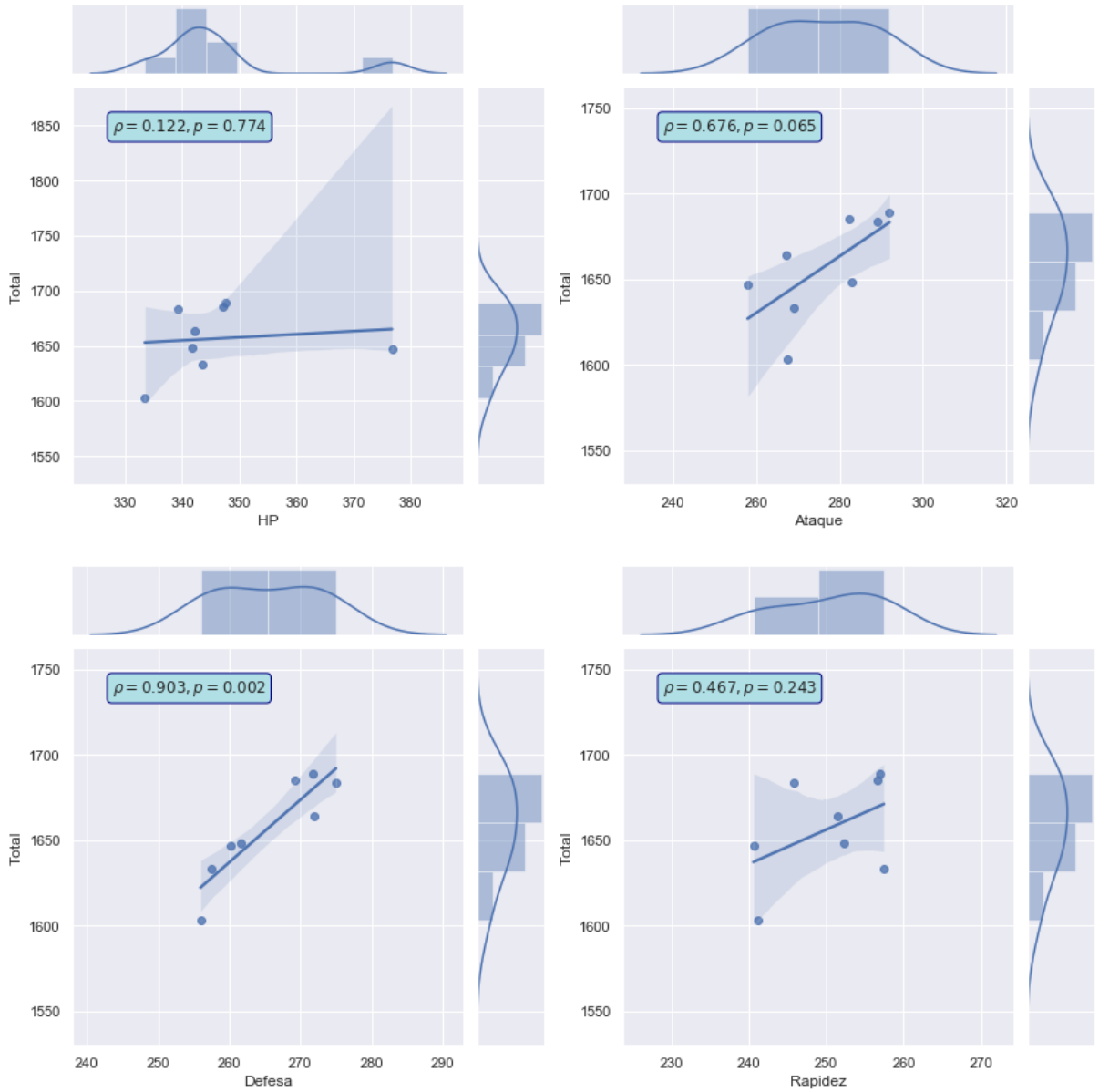
Através da soma dos valores das estáticas, podemos afirmar o que foi falado anteriormente, que os Pokémons da 5º geração possuem estatísticas mais atenuadas que as outras gerações. Veja que o pico de todas as estatísticas é em meados de 2010, ano de criação da 5ª geração.

GRÁFICO DE DISPERSÃO

```
pokemon_temporal = pokemons[['DataCriacao','HP','Ataque','Defesa','Rapidez','Total']].loc[(pokemons['Level'] == 100)].groupby('DataCriacao').mean()
pokemon_temporal

nvars=['HP','Ataque','Defesa','Rapidez']
```

```
rt=['Total']

fig = plt.figure(figsize=(13,8))
gs = gridspec.GridSpec(len(rt),len(nvars))

for i, idxs in enumerate(itertools.product(rt, nvars)):
    nrt, nvar = idxs
    g=sns.jointplot(data=pokemon_temporal, y=nrt, x=nvar,kind = 'reg')#, space=0,ax=axs[idx_rt,idx_var])
    r, p = stats.pearsonr(pokemon_temporal[nrt], pokemon_temporal[nvar])
    g.ax_joint.annotate(f'$\\rho = {r:.3f}, p = {p:.3f}$',xy=(0.1, 0.9), xycoords='axes frac-
tion',ha='left', va='center',bbox={'boxstyle': 'round', 'fc': 'powderblue', 'ec': 'navy'})

plt.show()
```

```
<Figure size 936x576 with 0 Axes>
```



O gráfico de dispersão, através da média, baseado no TOTAL, que é a soma de todas as estáticas, confirma a correção, onde tivemos um aumento considerável na seguinte ordem: Defesa, Ataque e Rapidez, tivemos aumento também no HP, porem de uma forma menos acentuada. Dessa forma, podemos concluir para um treina-

dor Pokémon que quanto maior a geração do Pokémon mais forte ele é, porém eles possuem menor resistência para a batalha (HP) comparado com as primeiras gerações.

LENDÁRIO

Existe uma teoria que os Pokémons lendários não são diferentes em poder quanto os outros Pokémons, sendo assim, não vale muita a pena a dificuldade de procurá-los, mas esse é um ponto que vamos nos aprofundar através da análise dos dados.

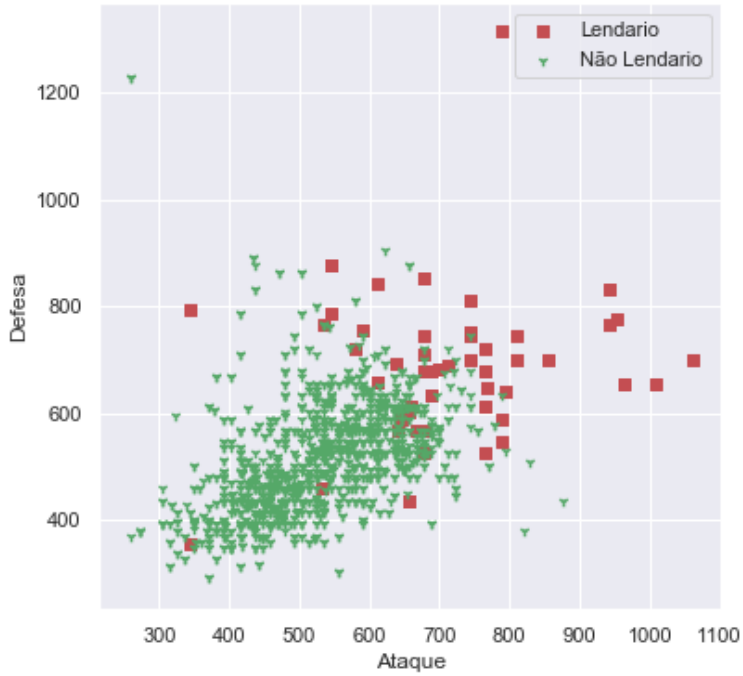
pokemons_lendario

```
pokemons_lendario = pokemons[['TotalAtaque','TotalDefesa','Lendario']].loc[(pokemons['Level'] == 100)]
pokemons_lendario['Lendario'] = (pokemons_lendario['Lendario']).astype(int)
pokemons_lendario.head()
```

Numero	TotalAtaque	TotalDefesa	Lendario
1	467	467	0
1	467	467	0
2	529	531	0
2	529	531	0
3	617	619	0

```
positivo = pokemons_lendario.loc[(pokemons_lendario['Lendario'] == 1)]
negativo = pokemons_lendario.loc[(pokemons_lendario['Lendario'] == 0)]
fig,graf = plt.subplots(figsize=(6,6))
graf.scatter(positivo['TotalAtaque'],positivo['TotalDefesa'],s=30,c='r',marker='s',label='Lendario')
graf.scatter(negativo['TotalAtaque'],negativo['TotalDefesa'],s=30,c='g',marker='1',label='Não Lendario')
graf.legend()
graf.set_xlabel('Ataque')
graf.set_ylabel('Defesa')
```

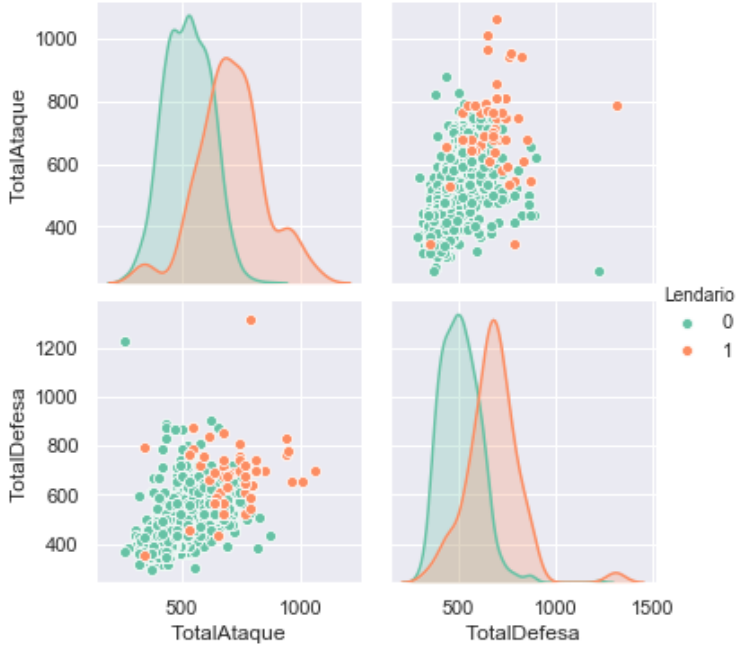
```
Text(0, 0.5, 'Defesa')
```



Podemos ver, salvo poucas exceções, que os Pokémons lendários possuem maior defesa e ataque dos que os Pokémons não lendários. Não conseguimos traçar

uma linha 100% reta, mas chega bem próximo disso. Porém com esse gráfico conseguimos desmistificar que a teoria que Pokémons lendários não possuem poderes diferenciados, sendo assim, podemos falar a um treinador de Pokémons, que se a sua procura for por poder e defesa, ele pode buscar isso entre os Pokémons lendários.

```
sns.pairplot(pokemons_lendario.dropna(), vars=['TotalAtaque','TotalDefesa'], hue='Lendario', palette="Set2");
```



Através desse gráfico conseguimos provar o que o gráfico de dispersão acima nos confirmou: a teoria de que Pokémons lendários não são diferentes em poder não é verdadeira, pois em sua grande maioria, os Pokémons lendários possuem ataques e defesas superiores.

5. Criação de Modelos de Machine Learning

Para os modelos de Machine Learning utilizei: REGRESSÃO LOGISTICA, ARVORE DE DECISÃO e REDES NEURAIS que foram utilizados para o mesmo dataframe, com as variáveis: TIPO, GERACAO, SEXO, TAMANHO, PESO, LEVEL, TOTAL, HABILIDADE e OVOTIPO. Onde o binário, classificatório, é a variável TIPO.

MACHINE LEARNING

ALVO

```
pokemons_ml = pokemons[['Tipo1','Geracao','Sexo','Tamanho','Peso','Level','Total','OvoTipo1','Habilidade1']].rename(columns={'Tipo1':'Tipo'})
pokemons_ml.head()
```

Numero	Tipo	Geracao	Sexo	Tamanho	Peso	Level	Total	OvoTipo1	Habilidade1
1	Grama	1	M	0.7	6.9	1	318	Monstro	Overgrow
1	Grama	1	M	0.7	6.9	50	736	Monstro	Overgrow
1	Grama	1	M	0.7	6.9	100	1435	Monstro	Overgrow
1	Grama	1	F	0.7	6.9	1	318	Monstro	Overgrow
1	Grama	1	F	0.7	6.9	50	736	Monstro	Overgrow

Para os modelos de Machine Learning darei continuidade a análise que identifiquei acima: se o treinador procurar por Pokémons da 5º geração do tipo água do sexo masculino ou feminino, ele terá mais chances de encontrar Pokémons com maiores poderes. Através dos dados GERACAO, SEXO, TAMANHO, PESO, LEVEL, TOTAL, OVOTIPO e HABILIDADE montarei os modelos de Machine Learning para prever se o Pokémon é do tipo: ÁGUA ou OUTROS, esse é o nosso ALVO. OBS.: Minha ideia era utilizar somente a 5ª geração como dados, porém como temos poucas informações, os modelos não ficaram tão assertivos, por isso a geração foi colocada como uma variável do modelo e todos os Pokémons foram considerados.

CLASSIFICAÇÃO

Tipo

```
pokemons_ml.loc[pokemons_ml['Tipo'] == 'Agua','Tipo'] = 1
pokemons_ml.loc[pokemons_ml['Tipo'] == 1].head()
```

Numero	Tipo	Geracao	Sexo	Tamanho	Peso	Level	Total	OvoTipo1	Habilidade1
7	1	1	F	0.5	9.0	50	732	Monstro	Torrent
7	1	1	F	0.5	9.0	100	1427	Monstro	Torrent
7	1	1	F	0.5	9.0	1	314	Monstro	Torrent
7	1	1	M	0.5	9.0	1	314	Monstro	Torrent
7	1	1	M	0.5	9.0	50	732	Monstro	Torrent

```
pokemons_ml.loc[pokemons_ml['Tipo'] != 1,'Tipo'] = 0
pokemons_ml.loc[pokemons_ml['Tipo'] == 0].head()
```

Numero	Tipo	Geracao	Sexo	Tamanho	Peso	Level	Total	OvoTipo1	Habilidade1
1	0	1	M	0.7	6.9	1	318	Monstro	Overgrow
1	0	1	M	0.7	6.9	50	736	Monstro	Overgrow
1	0	1	M	0.7	6.9	100	1435	Monstro	Overgrow

```
1         0        1    F      0.7   6.9      1    318  Monstro    Overgrow
1         0        1    F      0.7   6.9     50    736  Monstro    Overgrow
```

```
pokemons_ml['Tipo'] = pokemons_ml['Tipo'].astype(int)
pokemons_ml.head()
```

```
         Tipo  Geracao Sexo  Tamanho  Peso  Level  Total OvoTipo1 Habilidade1
Numero
1           0        1    M      0.7   6.9      1    318  Monstro    Overgrow
1           0        1    M      0.7   6.9     50    736  Monstro    Overgrow
1           0        1    M      0.7   6.9    100   1435  Monstro    Overgrow
1           0        1    F      0.7   6.9      1    318  Monstro    Overgrow
1           0        1    F      0.7   6.9     50    736  Monstro    Overgrow
```

Esse será o tipo binário, pois quero separar os Pokémons entre AGUA e OUTROS, dessa forma, todos os tipos de Pokémon diferente de AGUA recebem 0 e os do tipo AGUA recebem 1.

Sexo

```
pokemons_ml['Sexo'] = pokemons_ml['Sexo'].replace('M',1)
pokemons_ml['Sexo'] = pokemons_ml['Sexo'].replace('F',2)
pokemons_ml['Sexo'] = pokemons_ml['Sexo'].replace('I',3)
pokemons_ml.head()
```

```
         Tipo  Geracao  Sexo  Tamanho  Peso  Level  Total OvoTipo1 Habilidade1
Numero
1           0        1     1      0.7   6.9      1    318  Monstro    Overgrow
1           0        1     1      0.7   6.9     50    736  Monstro    Overgrow
1           0        1     1      0.7   6.9    100   1435  Monstro    Overgrow
1           0        1     2      0.7   6.9      1    318  Monstro    Overgrow
1           0        1     2      0.7   6.9     50    736  Monstro    Overgrow
```

OvoTipo

```
pokemons_ovotipo = pokemons_ml[['OvoTipo1']].drop_duplicates()
pokemons_ovotipo.insert(0,'OvoTipo', range(1, 1 + len(pokemons_ovotipo)))

pokemons_ml = pd.merge(pokemons_ml,pokemons_ovotipo,on='OvoTipo1').set_index(pokemons_ml.index)
pokemons_ml = pokemons_ml.drop('OvoTipo1', axis=1)

pokemons_ml.head()
```

```
         Tipo  Geracao  Sexo  Tamanho  Peso  Level  Total Habilidade1  OvoTipo
Numero
1           0        1     1      0.7   6.9      1    318    Overgrow        1
1           0        1     1      0.7   6.9     50    736    Overgrow        1
1           0        1     1      0.7   6.9    100   1435    Overgrow        1
1           0        1     2      0.7   6.9      1    318    Overgrow        1
1           0        1     2      0.7   6.9     50    736    Overgrow        1
```

```
del pokemons_ovotipo
```

Habilidade

```
pokemons_habilidade = pokemons_ml[['Habilidade1']].drop_duplicates()
pokemons_habilidade.insert(0,'Habilidade', range(1, 1 + len(pokemons_habilidade)))

pokemons_ml = pd.merge(pokemons_ml,pokemons_habilidade,on='Habilidade1').set_index(pokemons_ml.index)
pokemons_ml = pokemons_ml.drop('Habilidade1', axis=1)

pokemons_ml.head()
```

Numero	Tipo	Geracao	Sexo	Tamanho	Peso	Level	Total	OvoTipo	Habilidade
1	0	1	1	0.7	6.9	1	318	1	1
1	0	1	1	0.7	6.9	50	736	1	1
1	0	1	1	0.7	6.9	100	1435	1	1
1	0	1	2	0.7	6.9	1	318	1	1
1	0	1	2	0.7	6.9	50	736	1	1

```
del pokemons_habilidade
```

```
pokemons_ml.dtypes
```

```
Tipo            int32
Geracao         int64
Sexo            int64
Tamanho       float64
Peso          float64
Level           int64
Total           int64
OvoTipo         int64
Habilidade      int64
dtype: object
```

Para facilitar o trabalho e a análise dos modelos de Machine Learning estou transformando os textos em números. Além disso modelo de regressão logística só trabalha com valores numéricos.

GRÁFICO PAIRPLOT

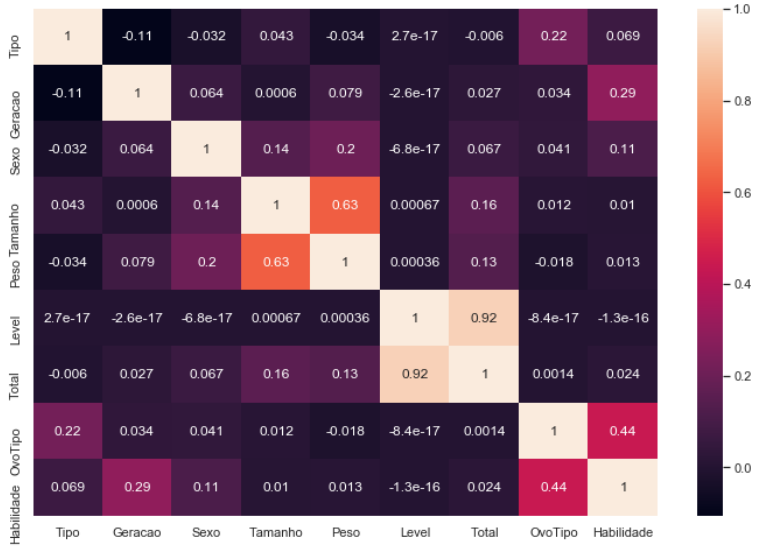
```
sns.pairplot(pokemons_ml.dropna(),
vars=['Geracao','Sexo','Tamanho','Peso','Total','OvoTipo','Habilidade'], hue='Tipo', palette='Set2',
diag_kws={'bw': 0.2});
```

Utilizei a biblioteca seaborn por ser uma das mais conhecidas e poderosas ferramentas de plotagem de gráficos no Python. Através dessa plotagem conseguimos visualizar melhor a correlação entre os Pokémons do tipo ÁGUA e OUTROS. Veja que para o dado SEXO, OVOTIPO e HABILIDADE a correlação é praticamente linear, pois só possuímos poucos tipos dentro dessas variáveis, já nas outras os dados são bem embaralhados mostrando que não possuímos uma correção linear forte entre eles.

MAPA DE CALOR

```
plt.figure(figsize=(12,8))
sns.heatmap(pokemons_ml.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ddb90b2dc8>
```

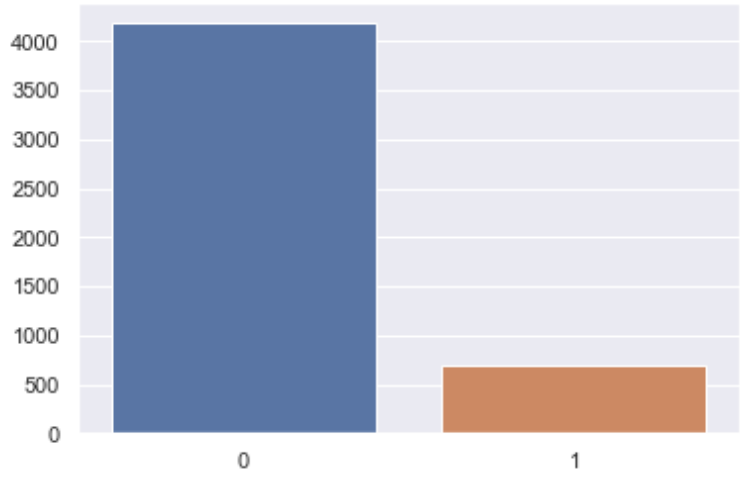


Quanto mais claro maior a correção positiva, com esse gráfico conseguimos provar o que falei acima, as correlações lineares não são muito fortes nesse dataframe. Para um modelo Machine Learning quanto mais as variáveis forem correlacionadas, menor sua performance, por isso o gráfico de calor é importante, pois conseguimos definir que não existe muitas correlações positivas.

GRÁFICO DE BARRAS

```
sns.barplot(x=pokemons_ml['Tipo'].value_counts().index,y=pokemons_ml['Tipo'].value_counts().values)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1ddbcbdd188>
```




```
pokemons_ml['Tipo'].value_counts()
```

```
0    4185
1     702
Name: Tipo, dtype: int64
```

O gráfico de barras aqui é útil para verificarmos se o nosso dado binário está realmente dividido de forma correta. Com ele podemos concluir que possuímos a menor quantidade de Pokémons do tipo AGUA X OUTROS.

ESCALONAMENTO DE DADOS

```
escalonamento = StandardScaler()

y_ml = pokemons_ml['Tipo']
x_ml = pokemons_ml.drop('Tipo', axis=1)

x_ml_esc = escalonamento.fit_transform(x_ml.values)
x_ml_esc = pd.DataFrame(x_ml_esc, index=x_ml.index, columns=x_ml.columns)
x_ml = x_ml_esc.copy()

x_ml.head()
```

Numero	Geracao	Sexo	Tamanho	Peso	Level	Total	OvoTipo	Habilidade
1	-1.386193	-0.986663	-0.390745	-0.48604	-1.220600	-1.191339	-1.336754	-1.162484
1	-1.386193	-0.986663	-0.390745	-0.48604	-0.008247	-0.427165	-1.336754	-1.162484
1	-1.386193	-0.986663	-0.390745	-0.48604	1.228848	0.850724	-1.336754	-1.162484
1	-1.386193	0.625447	-0.390745	-0.48604	-1.220600	-1.191339	-1.336754	-1.162484
1	-1.386193	0.625447	-0.390745	-0.48604	-0.008247	-0.427165	-1.336754	-1.162484

```
del x_ml_esc
```

Para o escalonamento dos dados estou utilizando biblioteca StandardScaler do sklearn, pois ela segue a distribuição normal padrão (SND). O escalonamento é importante para evitar distorções de escalas onde possuímos determinados valores muitos altos, muito baixos e decimais, essas variações prejudica muito a análise dos modelos. Podemos perceber que após o escalonamento os valores teremos escalas muito menores e muito mais parecidos, fazendo com que o algoritmo não de preferência entre escalas maiores ou menores.

BASES PARA TREINO E TESTE

```
x_ml_treino, x_ml_teste, y_ml_treino, y_ml_teste = train_test_split(x_ml, y_ml, test_size=0.3)
```

Para separação dos dados utilizei a biblioteca train_test_split do sklearn. Primeiro por possuir uma sintaxe bem simples e intuitiva, além disso é a biblioteca mais popular e utilizada para montagem de bases de teste e treino. Nela estou classificando que 30% da minha base será de teste e 70% será utilizada para treinar meus modelos de machine learning.

CURVA ROC

```
def plot_curva_roc(fper, tper, roc):
    plt.plot(fper, tper, color='red', label='Curva ROC (%0.2f)' % roc)
    plt.plot([0, 1], [0, 1], color='green', linestyle='--')
    plt.xlabel('FALSO POSITIVO')
    plt.ylabel('VERDADEIRO POSITIVO')
    plt.legend()
    plt.show()
```

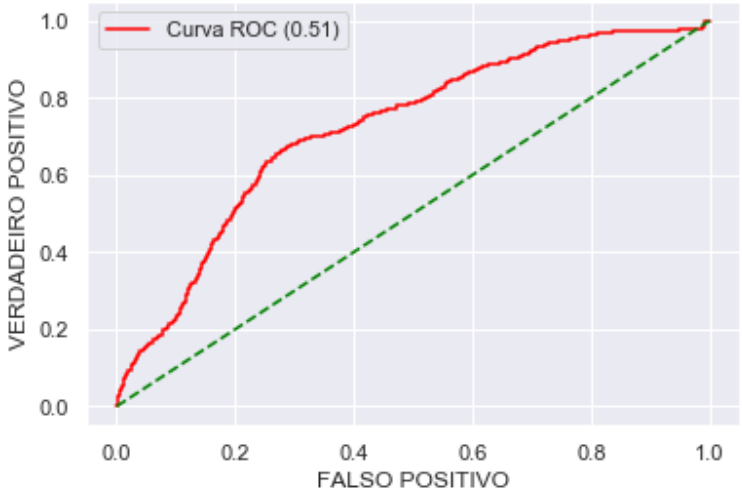
REGRESSÃO LOGÍSTICA

```
modelo_rl = LogisticRegression(random_state=0)
```

Estou usando o modelo LogisticRegression da biblioteca Sklearn, que usa técnica de classificação com saída de probabilidades de valores baseadas em uma variável categórica, que nesse caso é TIPO.

TREINO

```
modelo_rl_treino = modelo_rl.fit(x_ml_treino,y_ml_treino)

proba_rl_treino = modelo_rl.predict_proba(x_ml_treino)
proba_rl_treino = proba_rl_treino[:,1]

pred_rl_treino = modelo_rl_treino.predict(x_ml_treino)

fper, tper, thresholds = roc_curve(y_ml_treino, proba_rl_treino)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_treino, pred_rl_treino))
```




```
print(confusion_matrix(y_ml_treino, pred_rl_treino))
```

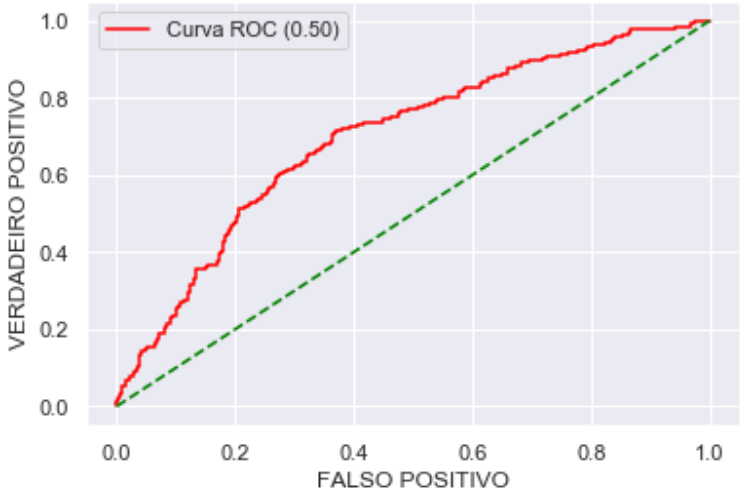
```
[[2920    7]
 [ 483   10]]
```

```
print(classification_report(y_ml_treino,pred_rl_treino,zero_division=0))
```

```
                precision    recall  f1-score   support

             0       0.86      1.00      0.92      2927
             1       0.59      0.02      0.04       493

      accuracy                           0.86      3420
     macro avg       0.72      0.51      0.48      3420
  weighted avg       0.82      0.86      0.80      3420
```

TESTE

```
modelo_rl_teste = modelo_rl.fit(x_ml_teste,y_ml_teste)

proba_rl_teste = modelo_rl.predict_proba(x_ml_teste)
proba_rl_teste = proba_rl_teste[:,1]

pred_rl_teste = modelo_rl_teste.predict(x_ml_teste)

fper, tper, thresholds = roc_curve(y_ml_teste, proba_rl_teste)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_teste, pred_rl_teste))
```



```
print(confusion_matrix(y_ml_teste, pred_rl_teste))
```

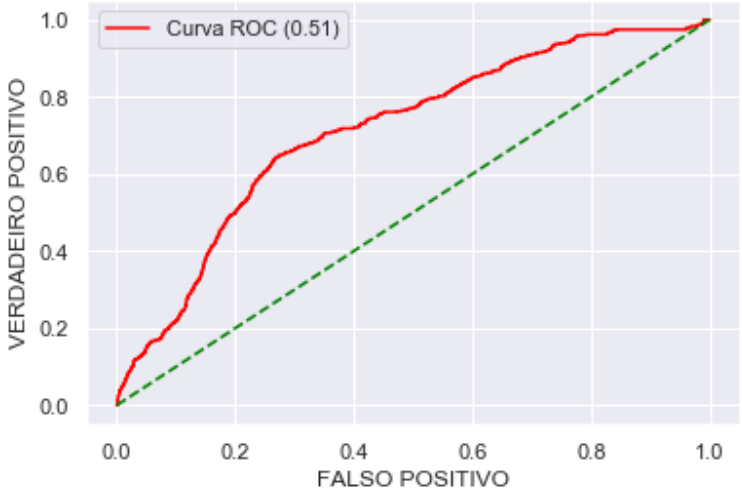
```
[[1258    0]
 [ 209    0]]
```

```
print(classification_report(y_ml_teste,pred_rl_teste,zero_division=0))
```

```
                    precision    recall  f1-score   support

                 0       0.86      1.00      0.92      1258
                 1       0.00      0.00      0.00       209

          accuracy                           0.86      1467
         macro avg       0.43      0.50      0.46      1467
      weighted avg       0.74      0.86      0.79      1467
```

COMPLETA

```
modelo_rl_full = modelo_rl.fit(x_ml,y_ml)

proba_rl_full = modelo_rl.predict_proba(x_ml)
proba_rl_full = proba_rl_full[:,1]

pred_rl_full = modelo_rl_full.predict(x_ml)

fper, tper, thresholds = roc_curve(y_ml, proba_rl_full)
plot_curva_roc(fper, tper, roc_auc_score(y_ml, pred_rl_full))
```




```
print(confusion_matrix(y_ml, pred_rl_full))
```

```
[[4176    9]
 [ 693    9]]
```

```
print(classification_report(y_ml, pred_rl_full,zero_division=0))
```

```
                    precision    recall  f1-score   support

                 0       0.86      1.00      0.92      4185
                 1       0.50      0.01      0.03       702

          accuracy                           0.86      4887
         macro avg       0.68      0.51      0.47      4887
      weighted avg       0.81      0.86      0.79      4887
```

Com esse modelo apesar de termos uma boa precisão para os Pokémons que não são do tipo ÁGUA, não obtivemos o mesmo ganho para o tipo ÁGUA, fazendo com que a curva ROC ficasse com score pouco acima de 50%. Estou usando a curva ROC para análise do modelo, pois é um gráfico muito bom para visualizarmos o quando o nosso modelo conseguiu distinguir entre duas coisas (classificação), no caso desse modelo entre os Pokémons tipo ÁGUA x OUTROS.

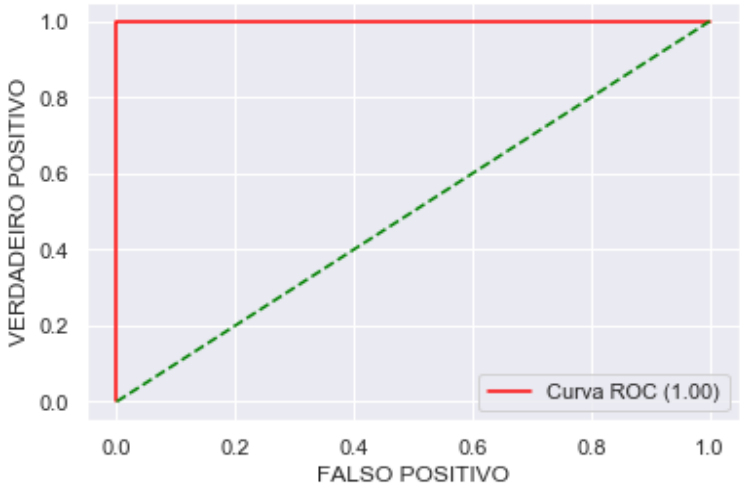
ÁRVORE DE DECISÃO

```
modelo_arvore = ExtraTreesClassifier()
```

Para esse modelo utilizarei a classe ExtraTreesClassifier da biblioteca Sklearn, pois ela cria um conjunto de dados aleatório para a arvore de decisão.

TREINO

```
modelo_ad_treino = modelo_arvore.fit(x_ml_treino, y_ml_treino)

proba_ad_treino = modelo_ad_treino.predict_proba(x_ml_treino)
proba_ad_treino = proba_ad_treino[:,1]

pred_ad_treino = modelo_ad_treino.predict(x_ml_treino)

fper, tper, thresholds = roc_curve(y_ml_treino, proba_ad_treino)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_treino, pred_ad_treino))
```




```
print(confusion_matrix(y_ml_treino, pred_ad_treino))
```

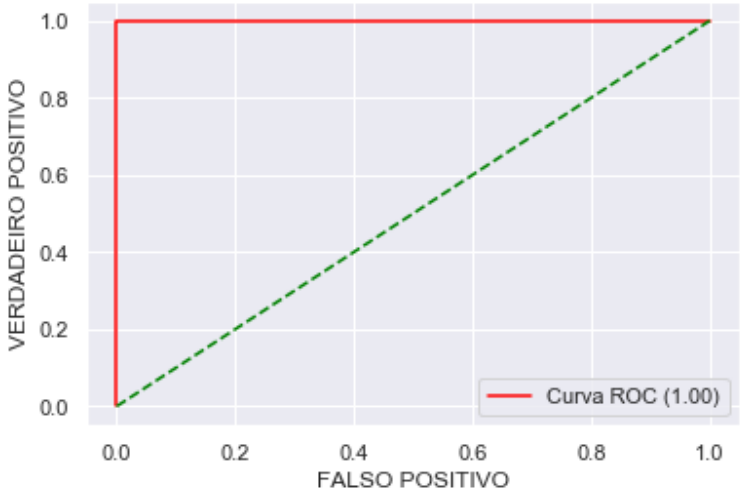
```
[[2927    0]
 [   0  493]]
```

```
print(classification_report(y_ml_treino, pred_ad_treino,zero_division=0))
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00      2927
               1       1.00      1.00      1.00       493

        accuracy                           1.00      3420
       macro avg       1.00      1.00      1.00      3420
    weighted avg       1.00      1.00      1.00      3420
```

TESTE

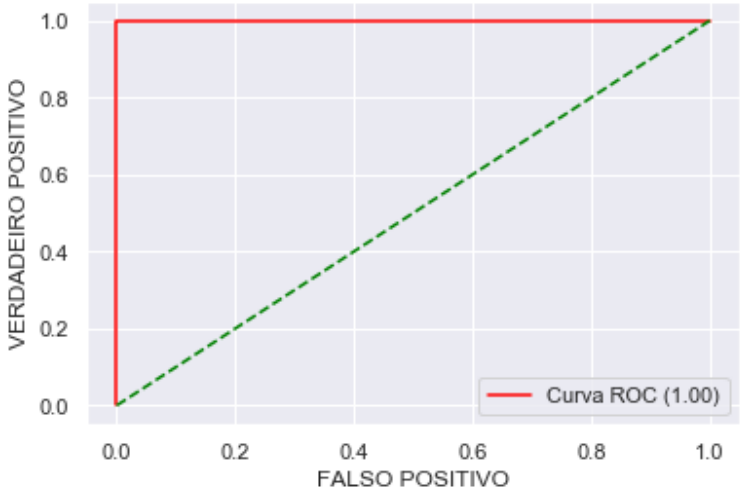
```
modelo_ad_teste = modelo_arvore.fit(x_ml_teste, y_ml_teste)

proba_ad_teste = modelo_ad_teste.predict_proba(x_ml_teste)
proba_ad_teste = proba_ad_teste[:,1]

pred_ad_teste = modelo_ad_teste.predict(x_ml_teste)

fper, tper, thresholds = roc_curve(y_ml_teste, proba_ad_teste)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_teste, pred_ad_teste))
```




```
print(confusion_matrix(y_ml_teste, pred_ad_teste))

        [[1258    0]
         [   0  209]]

print(classification_report(y_ml_teste, pred_ad_teste,zero_division=0))

                      precision    recall  f1-score   support

                   0       1.00      1.00      1.00      1258
                   1       1.00      1.00      1.00       209

            accuracy                           1.00      1467
           macro avg       1.00      1.00      1.00      1467
        weighted avg       1.00      1.00      1.00      1467
```

COMPLETA

```
modelo_ad_full = modelo_arvore.fit(x_ml, y_ml)

proba_ad_full = modelo_ad_full.predict_proba(x_ml)
proba_ad_full = proba_ad_full[:,1]

pred_ad_full = modelo_ad_full.predict(x_ml)

fper, tper, thresholds = roc_curve(y_ml, proba_ad_full)
plot_curva_roc(fper, tper, roc_auc_score(y_ml, pred_ad_full))
```




```
print(confusion_matrix(y_ml, pred_ad_full))
```

```
[[4185    0]
 [   0  702]]
```

```
print(classification_report(y_ml, pred_ad_full,zero_division=0))
```

```
                   precision    recall  f1-score   support

               0       1.00      1.00      1.00      4185
               1       1.00      1.00      1.00       702

        accuracy                           1.00      4887
       macro avg       1.00      1.00      1.00      4887
    weighted avg       1.00      1.00      1.00      4887
```

Com esse modelo conseguimos uma precisão excelente, onde foi possível precisar 100% se o Pokémon é do tipo ÁGUA ou OUTRO. Através da curva ROC, podemos comprovar essa eficiência, que ficou com o máximo de AUC.

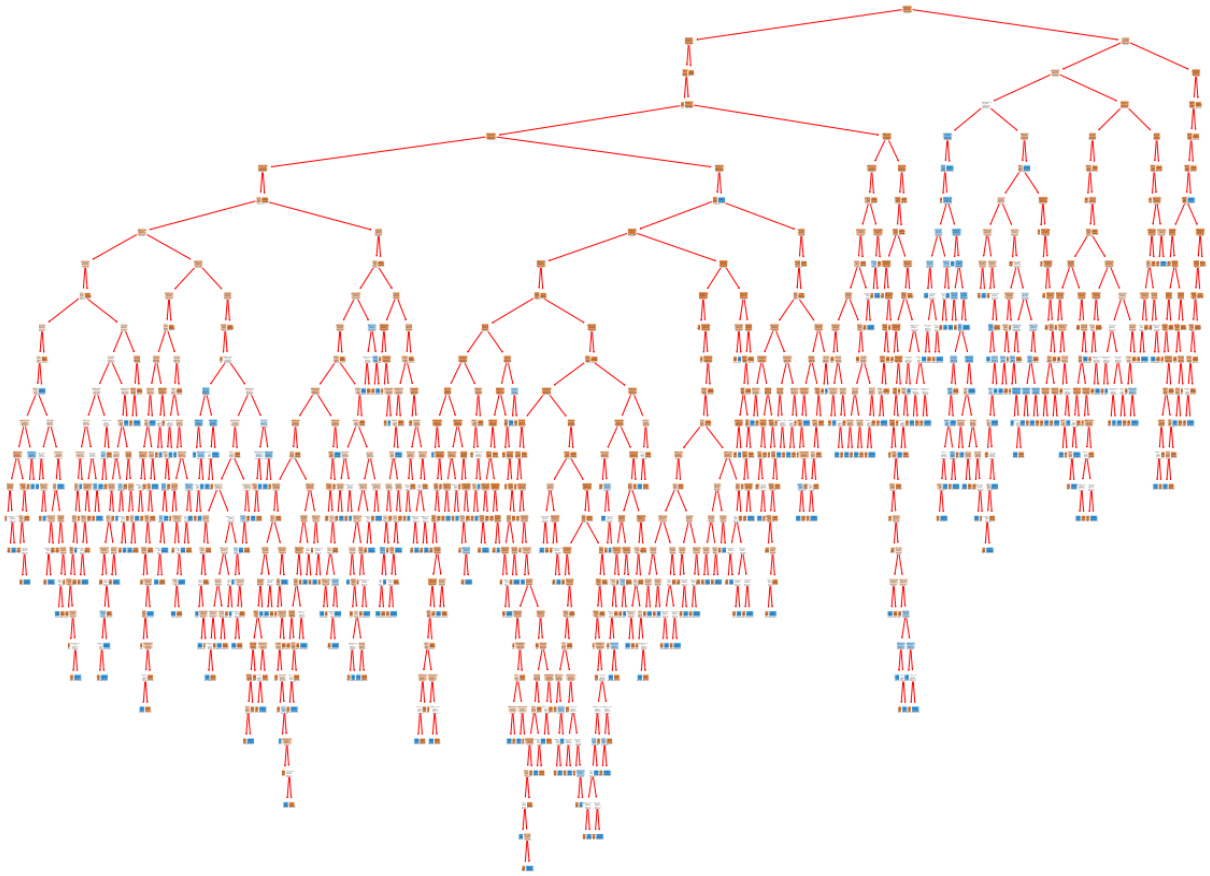
```
pd.DataFrame({'Importancia': modelo_arvore.feature_importances_}, in-
dex=x_ml.columns).sort_values(by='Importancia', ascending=False)
```

```
             Importancia
OvoTipo         0.316378
Habilidade      0.270201
Peso            0.146555
Tamanho         0.115108
Geracao         0.089153
Total           0.049321
Sexo            0.011053
Level           0.002232
```

Acima foi determinado a importância de cada recurso (dado numérico) utilizado na árvore de decisão.

```
arvore_fig = plt.figure(figsize=(20,15))
arvore_fig = tree.plot_tree(modelo_arvore.estimators_[0], feature_names=x_ml.columns,
class_names=pokemons_ml.columns, filled=True, impurity=True, rounded=True)
for arvore_arrow in arvore_fig:
```

```
    arrow = arvore_arrow.arrow_patch
    if arrow is not None:
        arrow.set_edgecolor('red')
        arrow.set_linewidth(1)
plt.show()
```



Com essa arvore de decisão conseguimos definir a partir de alguns aspectos do Pokémon se ele é do tipo ÁGUA ou OUTROS.

REDES NEURAIS

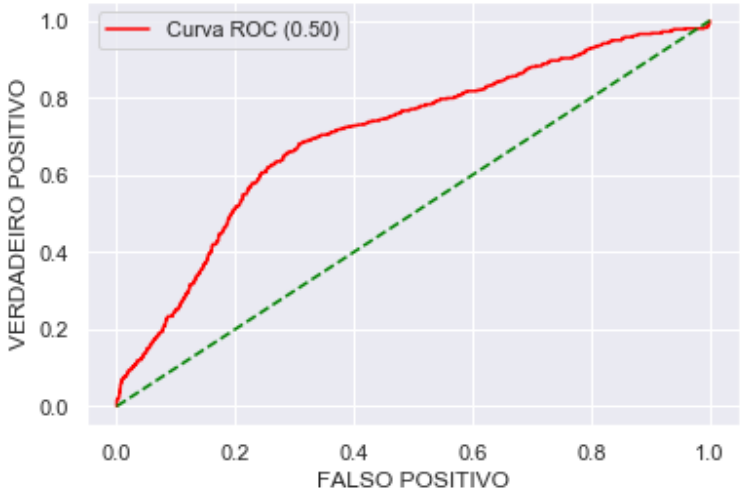
Para esse modelo estou usando a biblioteca Keras com TensorFlow, pois permite treinar modelos com várias camadas com poucas linhas de código e de maneira fácil.

TREINO

```
modelo_rn_treino = Sequential();
modelo_rn_treino.add(Dense(15,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_treino.columns)))
modelo_rn_treino.add(Dense(7,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_treino.columns)))
modelo_rn_treino.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_trei
```

```
no.columns)))
mode-
lo_rn_treino.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_trei
no.columns)))
mode-
lo_rn_treino.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_trei
no.columns)))
modelo_rn_treino.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
modelo_rn_treino.fit(x_ml_treino,y_ml_treino,epochs=5, callbacks =
tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3))

pred_rn_treino = modelo_rn_treino.predict(x_ml_treino)
pred_rn_treino_one = pred_rn_treino.argmax(axis=1)

fper, tper, thresholds = roc_curve(y_ml_treino, pred_rn_treino)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_treino, pred_rn_treino_one))
```

```
Epoch 1/5
107/107 [==============================] - 3s 5ms/step - loss: 0.6658 - accuracy: 0.7137
Epoch 2/5
107/107 [==============================] - 1s 6ms/step - loss: 0.5797 - accuracy: 0.8558
Epoch 3/5
107/107 [==============================] - 1s 9ms/step - loss: 0.5017 - accuracy: 0.8558
Epoch 4/5
107/107 [==============================] - 1s 8ms/step - loss: 0.4537 - accuracy: 0.8558
Epoch 5/5
107/107 [==============================] - 1s 11ms/step - loss: 0.4303 - accuracy: 0.8558
107/107 [==============================] - 1s 8ms/step
```




```
print(confusion_matrix(y_ml_treino, pred_rn_treino_one))
```

```
[[2927    0]
 [ 493    0]]
```

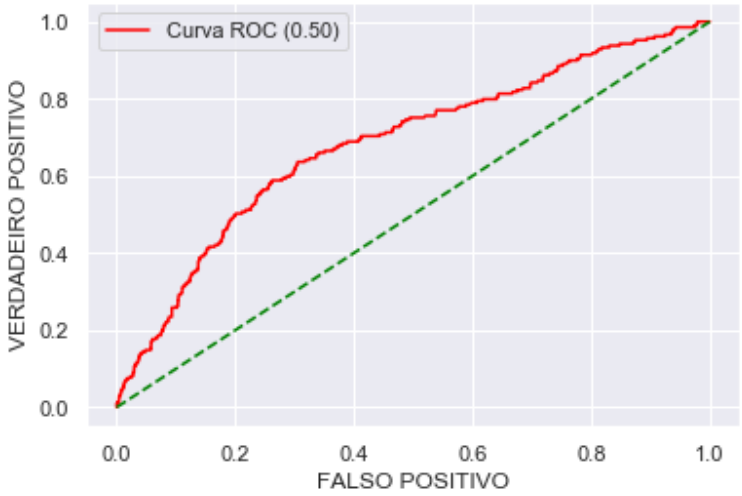
```
print(classification_report(y_ml_treino, pred_rn_treino_one,zero_division=0))
```

```
                   precision    recall  f1-score   support

               0       0.86      1.00      0.92      2927
               1       0.00      0.00      0.00       493

        accuracy                           0.86      3420
       macro avg       0.43      0.50      0.46      3420
    weighted avg       0.73      0.86      0.79      3420
```

TESTE

```
modelo_rn_teste = Sequential();
modelo_rn_teste.add(Dense(15,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_teste.columns)))
modelo_rn_teste.add(Dense(7,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_teste.columns)))
modelo_rn_teste.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_teste.columns)))
modelo_rn_teste.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_teste.columns)))
modelo_rn_teste.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml_teste.columns)))
modelo_rn_teste.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
modelo_rn_teste.fit(x_ml_teste,y_ml_teste,epochs=5, callbacks =
tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3))

pred_rn_teste = modelo_rn_teste.predict(x_ml_teste)
pred_rn_teste_one = pred_rn_teste.argmax(axis=1)

fper, tper, thresholds = roc_curve(y_ml_teste, pred_rn_teste)
plot_curva_roc(fper, tper, roc_auc_score(y_ml_teste, pred_rn_teste_one))
```

```
Epoch 1/5
46/46 [==============================] - 3s 8ms/step - loss: 0.6858 - accuracy: 0.6230
Epoch 2/5
46/46 [==============================] - 0s 9ms/step - loss: 0.6470 - accuracy: 0.8575
Epoch 3/5
46/46 [==============================] - 0s 9ms/step - loss: 0.6088 - accuracy: 0.8575
Epoch 4/5
46/46 [==============================] - 1s 13ms/step - loss: 0.5700 - accuracy: 0.8575
Epoch 5/5
46/46 [==============================] - 1s 12ms/step - loss: 0.5329 - accuracy: 0.8575
46/46 [==============================] - 1s 13ms/step
```




```
print(confusion_matrix(y_ml_teste, pred_rn_teste_one))
```

```
[[1258    0]
 [ 209    0]]
```

```
print(classification_report(y_ml_teste, pred_rn_teste_one,zero_division=0))
```

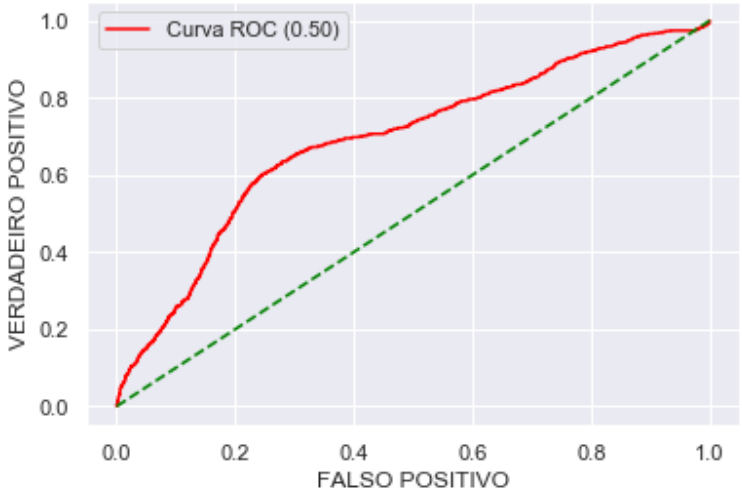
```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      1258
           1       0.00      0.00      0.00       209

    accuracy                           0.86      1467
```

```
           macro avg       0.43      0.50      0.46      1467
        weighted avg       0.74      0.86      0.79      1467
```

COMPLETA

```
modelo_rn_full = Sequential();
mode-
lo_rn_full.add(Dense(15,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml.colum
ns)))
mode-
lo_rn_full.add(Dense(7,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml.column
s)))
mode-
lo_rn_full.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml.column
s)))
mode-
lo_rn_full.add(Dense(3,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml.column
s)))
mode-
lo_rn_full.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal',input_dim=len(x_ml.column
s)))
modelo_rn_full.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
modelo_rn_full.fit(x_ml,y_ml,epochs=5, callbacks = tf.keras.callbacks.EarlyStopping(monitor='loss',
patience=3))

pred_rn_full = modelo_rn_full.predict(x_ml)
pred_rn_full_one = pred_rn_full.argmax(axis=1)

fper, tper, thresholds = roc_curve(y_ml, pred_rn_full)
plot_curva_roc(fper, tper, roc_auc_score(y_ml, pred_rn_full_one))
```

```
Epoch 1/5
153/153 [==============================] - 3s 7ms/step - loss: 0.6296 - accuracy: 0.8510
Epoch 2/5
153/153 [==============================] - 1s 5ms/step - loss: 0.5126 - accuracy: 0.8564
Epoch 3/5
153/153 [==============================] - 1s 5ms/step - loss: 0.4463 - accuracy: 0.8564
Epoch 4/5
153/153 [==============================] - 1s 5ms/step - loss: 0.4221 - accuracy: 0.8564
Epoch 5/5
153/153 [==============================] - 1s 5ms/step - loss: 0.4145 - accuracy: 0.8564
153/153 [==============================] - 1s 3ms/step
```




```
print(confusion_matrix(y_ml, pred_rn_full_one))
```

```
[[4185    0]
 [ 702    0]]
```

```
print(classification_report(y_ml, pred_rn_full_one,zero_division=0))
```

	precision	recall	f1-score	support
0	0.86	1.00	0.92	4185
1	0.00	0.00	0.00	702
accuracy			0.86	4887
macro avg	0.43	0.50	0.46	4887
weighted avg	0.73	0.86	0.79	4887

Com esse modelo apesar de termos uma boa precisão para os Pokémons que não são do tipo ÁGUA, não obtivemos o mesmo ganho para o tipo ÁGUA, fazendo com que a curva ROC ficasse com score em 50%.

No modelo de REGRESSÃO LOGISTICA, a curva ROC ficou de 51% para as bases de treino e completa, e 50% para a base de teste. Houveram muitos falsos positivos, o que mostra que em vários casos a previsão foi incorreta. A acurácia do score desse modelo não ficou tão ruim, 86%, mas se levarmos em consideração que o modelo não conseguiu prever Pokémons do tipo água na base de teste, e nas de treino e completa, ficou pouco acima de 50%, podemos concluir que não é um modelo bom para o tipo de classificação desejada.

No modelo ARVORE DE DECISÃO, a curva ROC ficou em 100% nas bases de treino, teste e completa, o que é um excelente resultado, pois indica que a precisão do modelo foi de 100%. Para provar 100% a performance/predição desse modelo tivemos a acurácia do score em 100%, assim como não houveram falsos positivos e falsos negativos em nenhuma das bases, dessa forma, podemos concluir que esse modelo atingiu 100% a necessidade da classificação desejada.

No modelo REDES NEURAIS, a curva ROC ficou em 50% nas bases de treino, teste e completa, esse não é um bom valor, pois indica que somente 50% do nosso modelo foi efetivo. Houveram muitos falsos positivos, indicando que muitos casos o modelo errou durante a analise da massa de dados. A acurácia do score desse modelo não ficou tão ruim, 86%, mas se levarmos em consideração que em nenhuma dos modelos ele conseguiu prever Pokémons do tipo água, podemos concluir que não é um modelo que atenda a necessidade.

Baseado nas analises detalhadas acima, é obvio que o melhor modelo de Machine Learning para essa massa de dados é a: ARVORE DE DECISÃO.

6. Apresentação dos Resultados

Quando comecei a trabalhar com esse TCC uma dúvida surgiu: como apresentar um resultado, que trará ganhos reais, através do mundo Pokémon. Juro que por várias vezes pensei em mudar de tema, usar datasets mais padrões, como: doenças, comercio, gastos, entre outros, em que eu pudesse me basear em estudos já feitos, mas acredito que o TCC nada mais é do que um desafio, por isso aceitei esse e aqui estou, apresentando resultados que por várias horas achei impossível.

A principal função dessa analise era mostrar a um treinador Pokémon, seja ele jogador de games ou card, o que ele deve procurar para ter em seu time os Pokémons que trarão mais ganhos nas batalhas e consequentemente faça ele ganhar batalhas cards ou ser o campeão Pokémon nos games.

Para isso foi gerado vários gráficos das variáveis obtidas através do datasets, onde encontramos uma informação muito relevante: Pokémons da 5ª geração, do tipo água, do sexo feminino ou masculino, possuem maiores chances de possuírem maiores estatísticas de batalha. Ao responder algumas perguntas, conseguimos identificar que a chance de um treinador pegar um tipo de Pokémons de sexo indeterminado é de menos de 6%, sendo assim ele possui 94% de chance de encontrar Pokémons do sexo feminino ou masculino dentro da 5ª geração. Ainda nessa geração possuímos Pokémons do tipo lendário ou místico, porém essa porcentagem é inferior a 10%, levando em consideração que estamos focados nos Pokémons do tipo água, temos a chance de 15% em pegar um místico, porém 0% de chance de pegar um lendário, pois nessa geração não existe Pokémon lendário do tipo água. As chances de um treinador encontrar um Pokémon do tipo água nessa geração é de menos de 38%. Em relação as habilidades, menos de 68% dos Pokémons do tipo água possuem duas habilidades e menos de 14% possuem habilidade oculta. Pokémons do tipo água místico, possuem a chance de terem mais de 60% duas habilidades e mais de 50% uma habilidade oculta, o que faz desse tipo de Pokémon um bom achado.

Com essa análise, podemos passar para um treinador Pokémon que ele deve focar na 5ª geração e procurar por Pokémons do tipo água, podendo encontrar místicos, dessa forma ele terá excelentes Pokémons para seu time.

Através do gráfico de calor conseguimos mostrar para um treinador Pokémon, o quanto é importante treinar seus Pokémons, ou procurar por cards onde os Pokémons estejam com níveis mais avançados, pois maior será sua força geral.

Também conseguimos a informação que Pokémons com rapidez mais elevada, possui um ataque mais preciso, sendo assim o treinador deverá ficar antenado com essa informação.

Pela nuvem de palavras conseguimos mostrar que os Pokémons que mais fáceis de encontrar são do tipo campo, inseto e voador. E na quinta geração seguimos o padrão, mostrando que existe uma dificuldade maior de encontrar Pokémon do tipo água, que é o foco.

Na análise temporal, baseado na data de criação de cada geração Pokémon, identificamos que quanto mais atual mais suas estatísticas de batalha aumentam, com exceção do HP, que é mais alto nas primeiras versões. Dessa forma, se um treinador está procurando um Pokémon que tenha mais resistência na batalha deverá procurar nas primeiras gerações, porém se está procurando por força deverá procurar em gerações mais atuais. Também conseguimos identificar mais uma vez, através da soma das estatísticas, que na 5ª geração é onde podemos encontrar os possíveis Pokémons com melhores estatísticas de batalha.

Pelo gráfico de dispersão, temos mais uma confirmação, que o treinador procura por resistência deverá procurar Pokémons nas primeiras gerações, porém se procura por força deverá procurar em gerações mais atuais.

No início desse trabalho, levantei uma teoria muito difundida no mundo Pokémon: que os Pokémons lendários não são diferentes em poder quanto os outros Pokémons. Mas através da análise dos dados pudermos ver, que salvo poucas exceções, os Pokémons lendários possuem maior defesa e ataque dos que os Pokémons não lendários. Sendo assim, podemos falar a um treinador de Pokémons, que se a sua procura for por poder e defesa, ele pode buscar isso entre os Pokémons lendários descartando essa teoria.

Para Machine Learning utilizei três modelos: Regressão Logística, Árvore de Decisão e Rede Neural. Considerei dar continuidade a análise que identifiquei durante a minha análise de dados: Pokémons da 5ª geração, do tipo água, do sexo feminino ou masculino, possuem maiores chances de possuírem maiores estatísticas de batalha. Então através das variáveis geração, sexo, tamanho, peso, level, total, tipo de ovo e habilidade treinei os modelos de Machine Learning para prever se o Pokémon é do tipo: água ou outros. Ao final escolhi o modelo de ÁRVORE DE DECISÃO, pois conforme testes apurados com esse modelo conseguimos uma precisão de 100% para os dois tipos (água ou outros) e também 100% na Curva

ROC, comprovando assim a eficiência desse modelo. Sendo assim, através desse modelo, um treinador Pokémon poderá colocar informações iniciais que ele considera relevante (como por exemplo: geração, estatísticas de batalha), que ele conseguirá ter uma precisão se esse Pokémon será do tipo água ou outro.

7. Links

https://github.com/KarolMellody/PucMinasTCC_2022

https://youtu.be/HS3brOpvosg