

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

**Rodrigo Xavier de Almeida Leão**

**Análise de Histórico de Jogos de Poker  
para Aumento de Rendimento**

Belo Horizonte  
2022

**Rodrigo Xavier de Almeida Leão**

**ANÁLISE DE HISTÓRICO DE JOGOS DE POKER  
PARA AUMENTO DE RENDIMENTO**

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

## SUMÁRIO

<b>1. Introdução.....</b>	<b>4</b>
<b>1.1. Contextualização .....</b>	<b>4</b>
<b>1.2. O problema proposto .....</b>	<b>4</b>
<b>1.3. Fundamentos do Poker .....</b>	<b>4</b>
<b>1.4. Objetivos .....</b>	<b>6</b>
<b>2. Coleta de Dados.....</b>	<b>7</b>
<b>3. Processamento/Tratamento de Dados .....</b>	<b>8</b>
<b>4. Análise e Exploração dos Dados .....</b>	<b>10</b>
<b>5. Criação de Modelos de Machine Learning .....</b>	<b>12</b>
<i>Regressão Linear Simples .....</i>	<i>12</i>
<i>Regressão Linear com Atributos Polinomiais.....</i>	<i>13</i>
<i>Árvore de Decisão.....</i>	<i>14</i>
<b>6. Interpretação dos Resultados .....</b>	<b>17</b>
<b>7. Apresentação dos Resultados .....</b>	<b>18</b>
<b>8. Links .....</b>	<b>18</b>
<b>APÊNDICE.....</b>	<b>19</b>

## 1. Introdução

### 1.1. Contextualização

O *poker* é um dos jogos de carta mais populares do mundo e vem cada vez mais conquistando adeptos, principalmente através dos jogos online. Muitos entusiastas podem ficar frustrados com seus resultados iniciais ou mesmo culpar o azar, porém além das regras básicas o Poker possui algumas estratégias de jogo que são fundamentais para evitar grandes perdas.

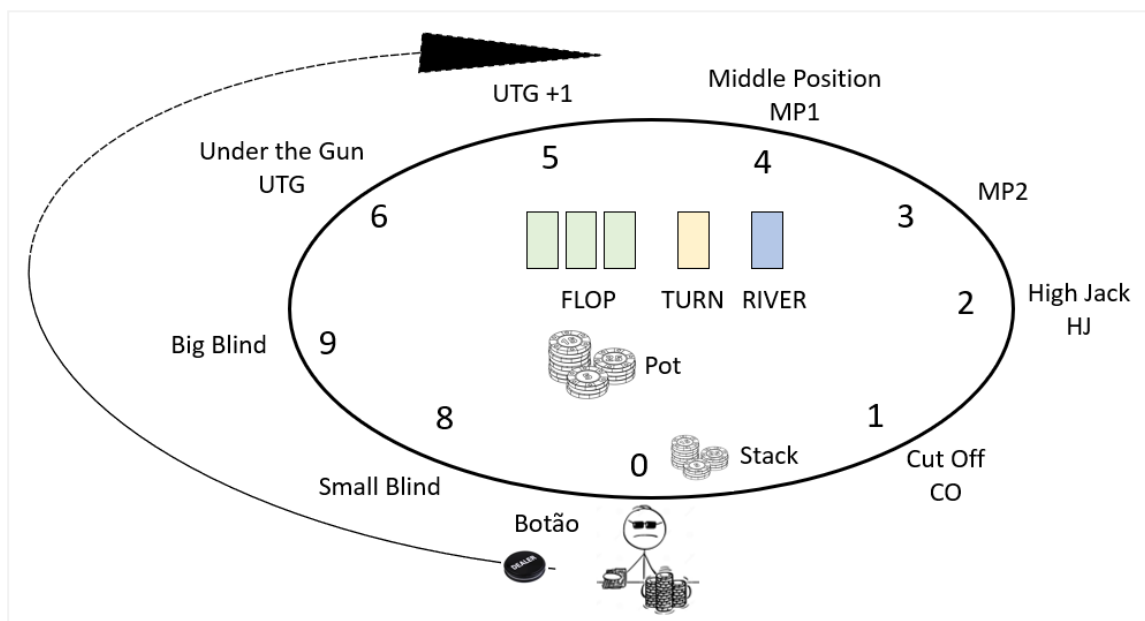
Na verdade, é possível eliminar consideravelmente o fator sorte tornando-o um jogo de probabilidade e estatísticas. É claro que a capacidade e habilidade individual faz total diferença na mesa, porém todos os grandes jogadores sabem como aplicar a probabilidade a seu favor fazendo com que no longo prazo seus ganhos superem as perdas.

### 1.2. O problema proposto

Neste contexto, este trabalho apresenta os esforços iniciais em desenvolver um método capaz de fornecer uma análise global do estilo de jogo do cliente a partir de seu histórico de jogos online, que pode ser obtido na plataforma de jogo. Desta forma é possível apontar falhas no estilo de jogo e sugerir alterações que resultem em maior lucratividade.

### 1.3. Fundamentos do Poker

Neste trabalho considera-se torneios do estilo Texas Hold'em de 9 jogadores como ilustrado abaixo.



Neste estudo, o jogador está sempre sentado na posição zero. O botão, assim como as posições giram no sentido horário a cada rodada. As regras básicas de cada rodada são:

1. O “blind” é o valor mínimo de aposta da mão e aumenta em períodos específicos de tempo;
2. Os jogadores nas posições SB e BB fazem uma aposta obrigatória cada, de 1/2 e 1 blind respectivamente;
3. Todos os jogadores recebem duas cartas cada;
4. Etapa 1: o pré flop, cada jogador decide se irá jogar com suas cartas e quanto irá apostar. Se mantêm no jogo todos aqueles que pagarem a maior aposta;
5. Etapa 2: o flop, são viradas três cartas sobre a mesa;
6. A partir de agora, a cada virada de carta, os jogadores anunciam suas apostas sempre no sentido horário iniciando do jogador à esquerda do botão (posição 8) até o botão (posição 0) que é sempre o último a falar. Se mantêm no jogo aqueles que pagarem a maior aposta;
7. Etapa 3: o turn, vira-se mais uma carta e repete-se o procedimento da etapa anterior.
8. Etapa 4: o river, vira-se mais uma carta e repete-se o procedimento da etapa anterior.
9. Etapa 5: o showdown, os jogadores que permaneceram no jogo mostram suas cartas e ganha aquele que obtiver a maior jogada considerando 5 cartas escolhidas entre as duas da mão e as cinco da mesa.

A tabela abaixo apresenta as jogadas do poker:

ROYAL STR. FLUSH	10 ♥	J ♥	Q ♥	K ♥	A ♥
STRAIGHT FLUSH	4 ♣	5 ♣	6 ♣	7 ♣	8 ♣
QUADRA	K ♠	K ♥	K ♣	K ♦	3 ♠
FULL HOUSE	10 ♥	10 ♠	10 ♦	A ♠	A ♣
FLUSH	10 ♠	K ♠	2 ♠	6 ♠	7 ♠
SEQUÊNCIA	7 ♣	8 ♠	9 ♦	10 ♠	J ♥
TRINCA	5 ♠	5 ♥	5 ♣	J ♦	A ♦
2 PARES	A ♠	A ♥	3 ♣	3 ♠	J ♣
1 PAR	Q ♦	Q ♥	2 ♥	8 ♠	9 ♣

É de conhecimento geral da comunidade de poker que quatro fatores afetam diretamente o resultado de uma mão. Em primeiro lugar a posição, sendo sempre melhor jogar no botão por ser o último a tomar decisões a cada mão; neste trabalho o parâmetro “BT\_pos” indica a posição em que o botão se encontra em relação ao jogador, de forma que quanto menor seu valor melhor para jogar.

Em segundo lugar a quantidade de fichas em jogo “Pot\_BB” e terceiro a quantidade de fichas que o jogador possui naquele momento “Stack\_BB”. Em quarto e mais óbvio a força da mão “Hand\_Power” que também depende se as cartas são do mesmo naipe ou não “Swited”. O Pot e o Stack são calculados em relação ao big blind daquela rodada.

#### 1.4. Objetivos

- Definir pesos e variáveis de relevância;
- Desenvolver um método de extração das variáveis a partir do histórico de jogos;
- Desenvolver uma forma de análise que permita relacionar as variáveis aos resultados obtidos pelo jogador;
- Aplicar métodos de aprendizado de máquina e avaliar a capacidade do modelo em tomar decisões consistentes com os fundamentos do Poker.

## 2. Coleta de Dados

Os dados são obtidos a partir do histórico de jogos que pode ser obtido na plataforma de jogo utilizada. No presente caso utilizou-se a plataforma *PokerStars* que fornece o arquivo do tipo “.txt”. A Figura 1 apresenta a estrutura dos dados para uma mão jogada, o arquivo completo é composto por todas as mãos de um torneio.

O histórico contém nas duas primeiras linhas as identificações e especificações do torneio, estas informações são utilizadas para agrupar as mãos entre os torneios. NA terceira linha são fornecidas as fichas iniciais de cada jogador e os níveis de aposta.

Finalmente o arquivo é dividido pelas etapas das mãos (*hole cards* = pré-flop, flop, turn e river) e a ação de cada jogador naquela etapa, além das cartas recebidas e colocadas sobre a mesa em cada uma.

```
PokerStars Hand #236216463916: Tournament #3413806416, $0.23+$0.02 USD Hold'em No Limit
Table '3413806416 1' 3-max Seat #1 is the button
Seat 1: zoriank (500 in chips)
Seat 2: Asecos (500 in chips)
Seat 3: rodxdal (500 in chips)
Asecos: posts small blind 10
rodxdal: posts big blind 20
*** HOLE CARDS ***
Dealt to rodxdal [4h 9s]
zoriank: folds
Asecos: calls 10
rodxdal: checks
*** FLOP *** [5h Td 5s]
Asecos: checks
rodxdal: checks
*** TURN *** [5h Td 5s] [Jc]
Asecos: checks
rodxdal: bets 29
Asecos: folds
Uncalled bet (29) returned to rodxdal
rodxdal collected 40 from pot
rodxdal: doesn't show hand
*** SUMMARY ***
Total pot 40 | Rake 0
Board [5h Td 5s Jc]
Seat 1: zoriank (button) folded before Flop (didn't bet)
Seat 2: Asecos (small blind) folded on the Turn
Seat 3: rodxdal (big blind) collected (40)
```

Figura1. Históricos de mãos

### 3. Processamento/Tratamento de Dados

Foram processados um total de 96 históricos que passam pelo fluxo KNIME apresentado na Figura 2.

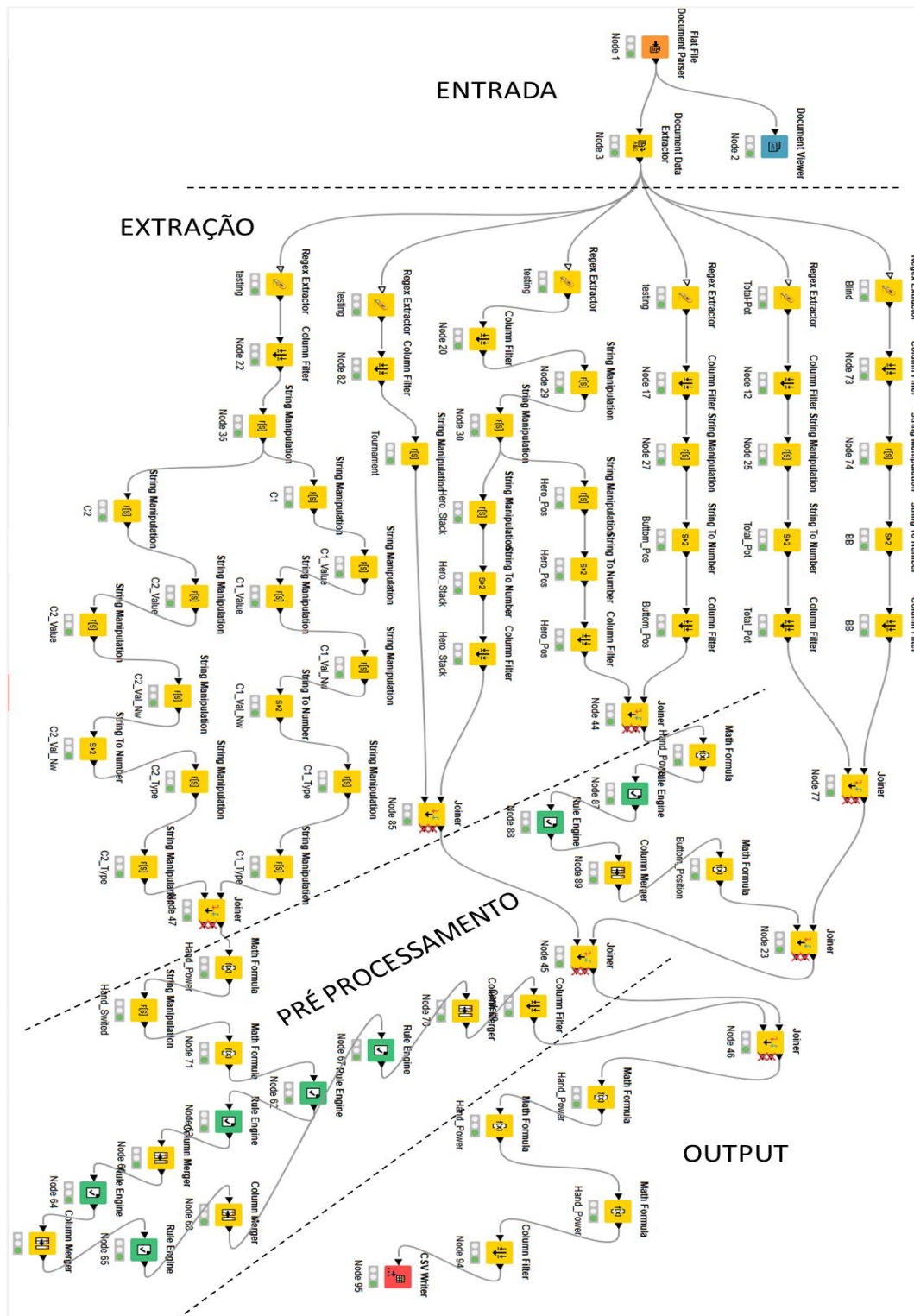


Figura 2. Fluxo KNIME



A partir do fluxo apresentado é extraída a Tabela 1 com um total de 1812 mão.

Table "default" - Rows: 1812 Spec - Columns: 6 Properties Flow Variables						
Row ID	D BT_Pos	S Tourna...	D Swited	D Stack_BB	D Pot_BB	D Hand_P...
Row0	5	3400213716	1	75	6.5	60
Row1	6	3400213716	1	75	21	224
Row2	7	3400213716	1	65.5	80.6	88
Row3	8	3400213716	1	58.5	48.15	21
Row4	0	3400213716	1	28.3	101.433	22
Row5	1	3400213716	1	28.3	15	24
Row6	2	3400213716	1	28.3	9.5	45
Row7	3	3400213716	1	28.3	107	60
Row8	4	3400213716	0	16.98	6	26
Row9	5	3400213716	0	16.98	18	8
Row10	6	3400213716	1	16.98	39.28	117
Row11	1	3399211094	1	75	157.55	39

Os parâmetros que constam na tabela são: BT\_Pos = Posição do botão; Tournament = identifica o torneio em que aquela mão foi jogada; Swited = Identifica se a mão é do mesmo naipe; Stack\_BB = número de fichas do jogador ao iniciar aquela mão, contado em blinds; Pot\_BB = quantidade de fichas no pote ao final da mão, contado em blinds; Hand\_Power = Força da mão, resultado da multiplicação do valor das cartas da mão, que vão de 2 a 14 (Ás), pela categoria da mão:

- 5. Cartas iguais (par de mão);
- 4. Ás e outra carta qualquer;
- 3. Duas figuras;
- 2. Figura e outra carta qualquer que não seja o Ás;
- 1. Dois números diferentes.

A tabela de dados é então transmitida por meio de arquivo “.csv” para o Colab no qual as análises são realizadas por meio de linguagem Python. A Tabela 1 aberta no Colab é apresentada a seguir.

<pre>#DataSet -&gt; Dataframe df=pd.read_csv('PokerData.csv') print('Dimensões do DataSet --&gt; ', df.shape) df.head()</pre>						
<p>Dimensões do DataSet --&gt; (1812, 6)</p>						
	BT_Pos	Tournamnet_Number	Swited	Stack_BB	Pot_BB	Hand_Power
0	5	3400213716	1	75.0	6.50	60
1	6	3400213716	1	75.0	21.00	224
2	7	3400213716	1	65.5	80.60	88
3	8	3400213716	1	58.5	48.15	21
4	0	3400213716	1	28.3	101.43	22

Figura 3. Tabela 1 no Colab

#### 4. Análise e Exploração dos Dados

Foram identificados 54 torneios diferentes. O parâmetro *Stack* é transformado em ganho por mãos “Gain” subtraindo o *Stack* anterior pelo *Stack* da próxima mão em cada torneio. A remoção dos valores vazios resultou em uma redução do banco de dados para 1758 mãos representando uma perda de 2.98%. A Figura 4 apresenta a distribuição do ganho parametrizado em função do maior valor.

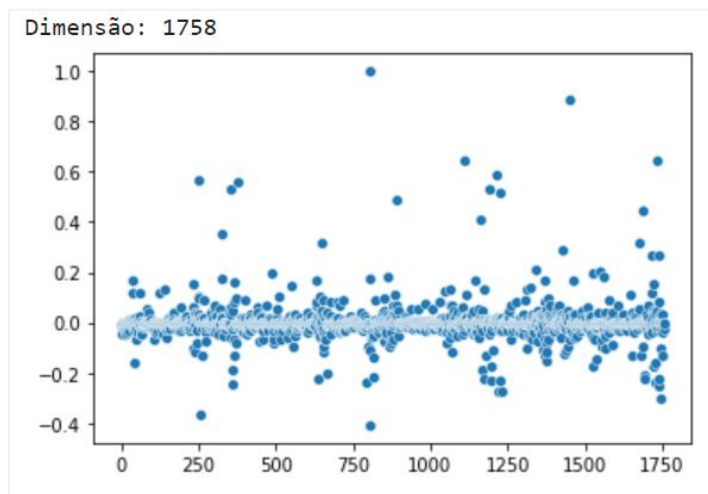


Figura 4. Ganho do jogador

São realizados dois cortes de *outliers* baseado nas análises primárias e dinâmica do jogo. As mãos envolvidas em pots maiores que 500 BB que normalmente representam os instantes finais do jogo em que os *blinds* já estão muito altos ou situações extraordinárias de início de jogo. E as mãos em que o jogador inicia com cartas que possuam “Hand\_Power” maior que JJ (par de valetes) que normalmente envolverão muitas fichas.

Os comportamentos acima são padrões e comuns em qualquer nível de jogo e mascaram as nuances referentes ao próprio jogador. O corte de *outliers* reduz o banco de dados para 1656 mãos, representando uma perda de 5.15 %.

A distribuição dos dados é harmonizada pela aplicação de escalas logarítmicas nos parâmetros Hand\_Power, Stack e Pot. A Figura 5 apresenta os histogramas dos parâmetros antes e depois da harmonização.

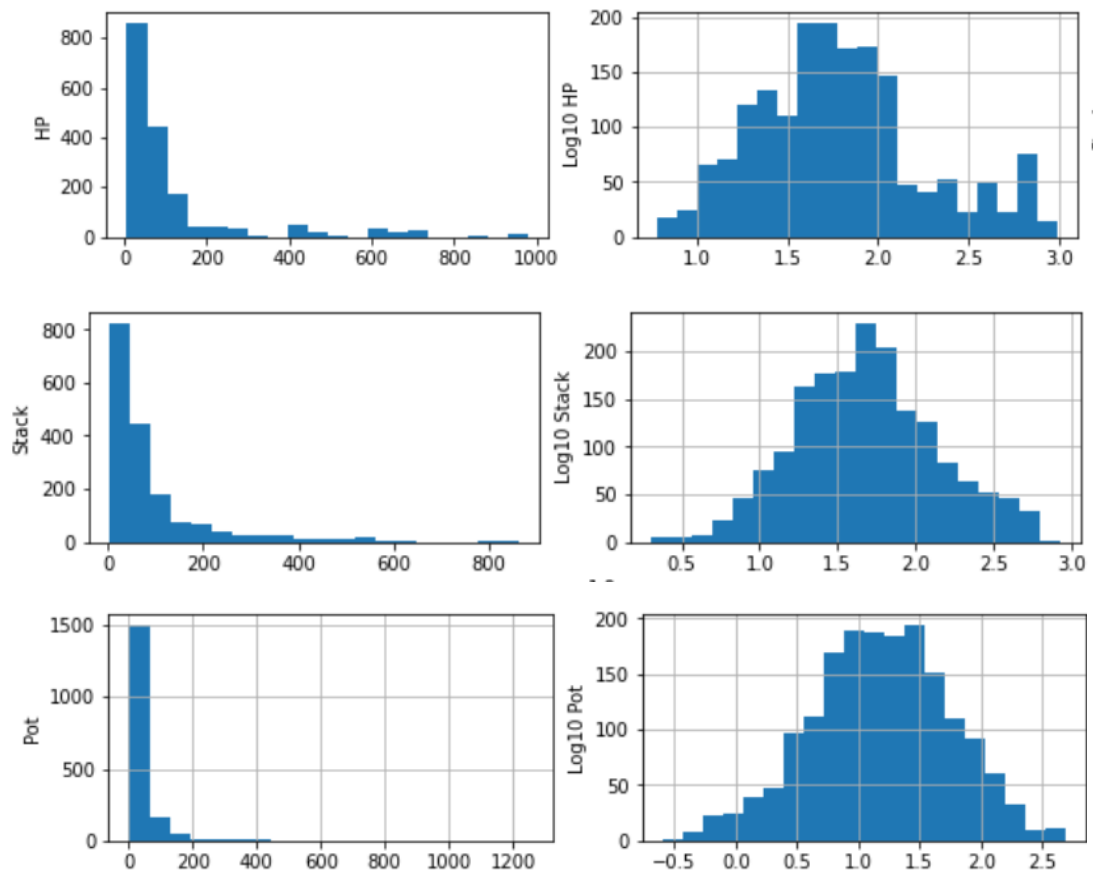


Figura 5. Harmozização de parâmetros

A Figura 6 apresetna a nova dsitribuição de parâmetros e o histograma de ganho.

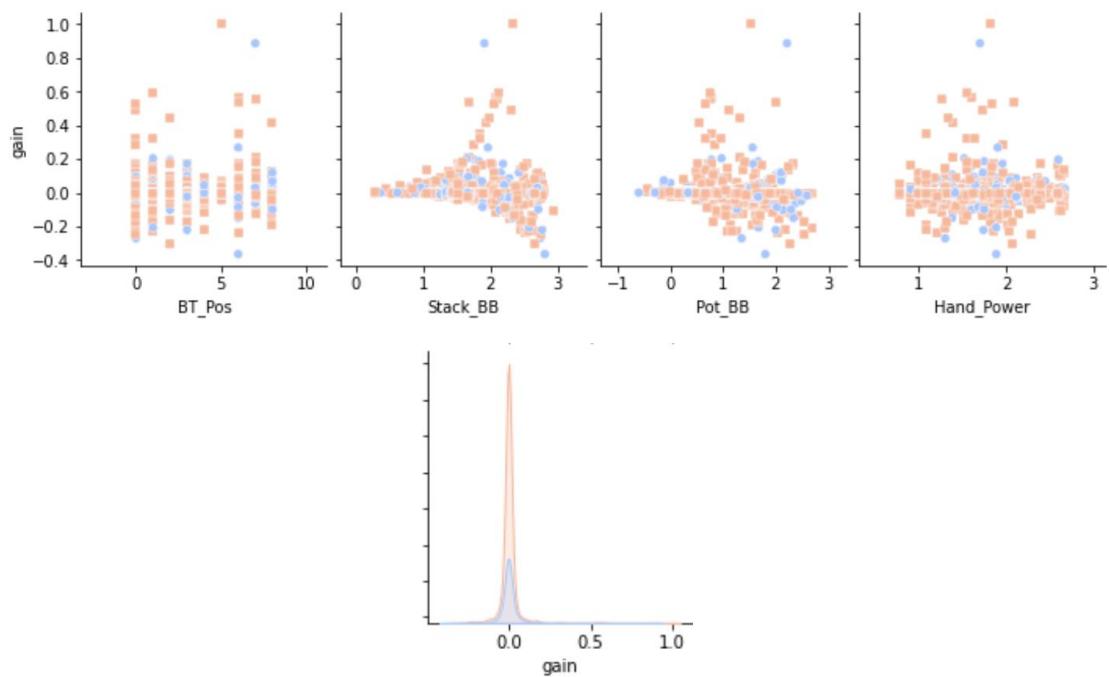


Figura 6. Distribuição de parâmetros em relação ao ganho.

## 5. Criação de Modelos de Machine Learning

### Regressão Linear Simples

O modelo linear é aplicado considerando o ganho escalonado como Target, apresentado na Figura 7.

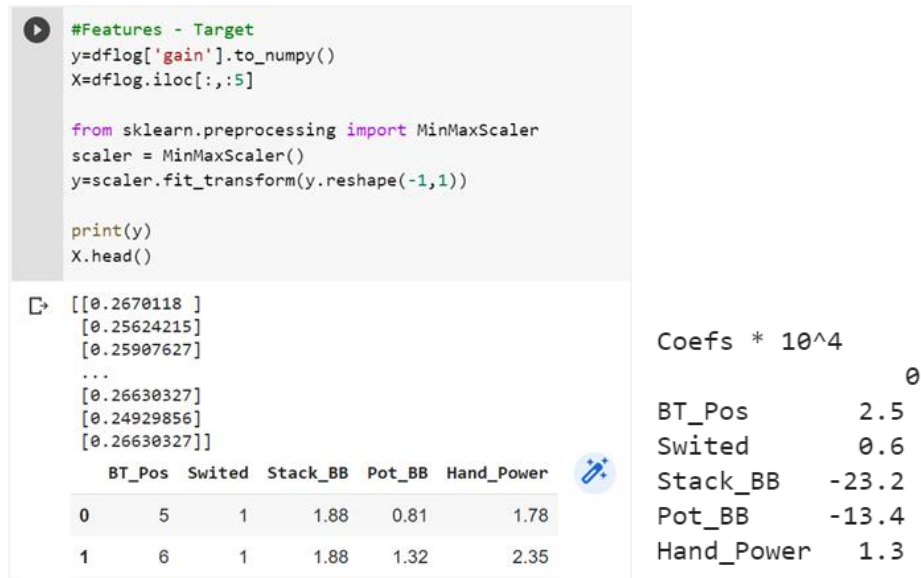


Figura 7. Features e Target.

O gráfico de erros é apresentado na Figura 8.

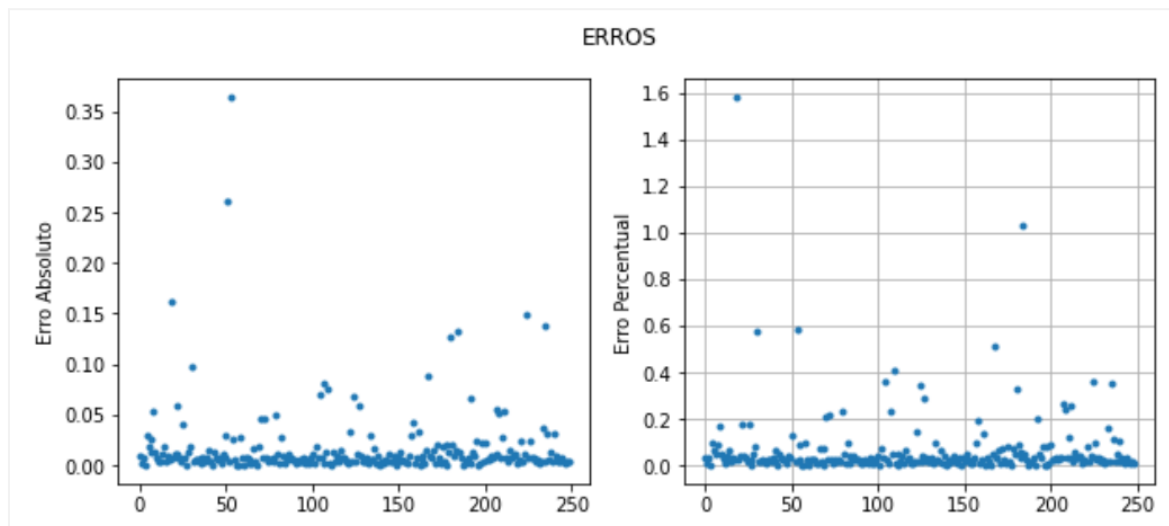


Figura 8. Erros

## Regressão Linear com Atributos Polinomiais

Os resultados dos modelos de regressão RIDGE são apresentados na Figura 9 no qual são avaliados 20 parâmetros e a variação de Alpha.

```
from sklearn.preprocessing import PolynomialFeatures

Xpoly=PolynomialFeatures(degree=2, include_bias=False).fit_transform(X)

('Xpoly SHAPE --> ', Xpoly.shape)
```

↳ ('Xpoly SHAPE --> ', (1746, 20))

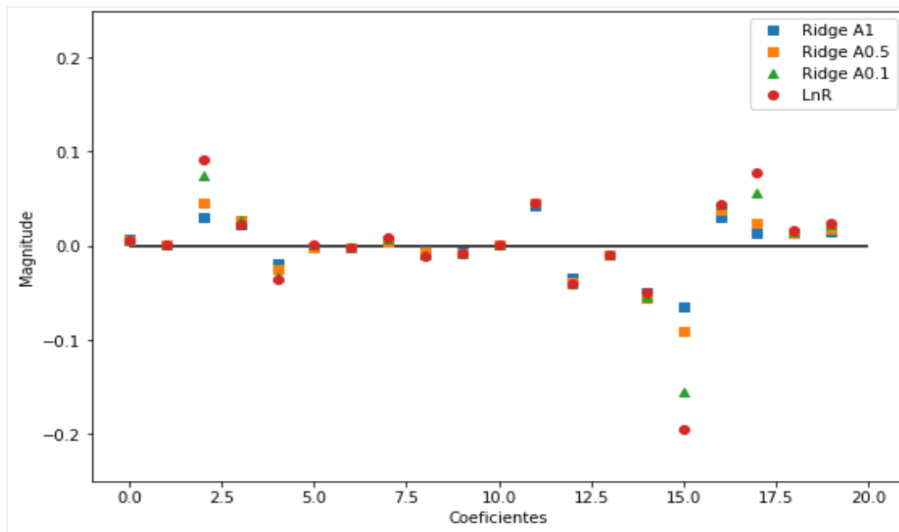


Figura 9. Modelos RIDGE

O melhor modelo RIDGE é comparado ao LASSO na Figura 10.

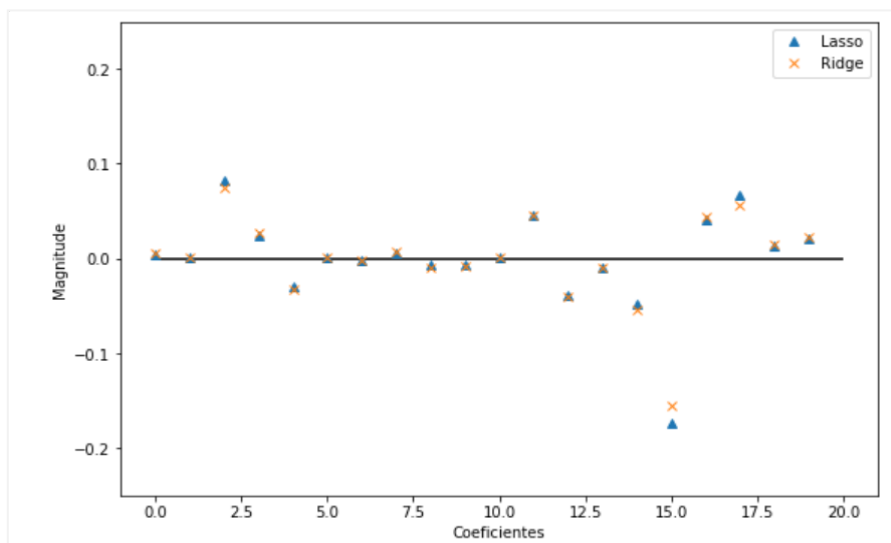


Figura 10. RIDGE x LASSO

## Árvore de Decisão

Os parâmetros são classificados em categorias como apresentado abaixo. O novo parâmetro POT é obtido pela divisão ( Pot\_BB / Stack\_BB ], ou seja, o número de fichas na mesa pelo número de fichas na mão do jogador. Desta forma o modelo possui quatro parâmetros categóricos de entrada que podem ser obtidos facilmente em cada mão em um jogo real.

HAND POWER CLASSES

```
[1.4313637641589874, 1.7323937598229686, 1.9590413923210936, 2.6989700043360187]
```

STACK CLASSES

```
[0.4783110046077968, 0.6868093788388092, 0.8936897164191739, 2.2223924213364477]
```

SHAPE -> (1656, 5)

	BT_Pos	HP	SWT	POT	LDW
0	5	C	ST	A	D
1	6	D	ST	C	L
2	7	C	ST	D	L
3	8	A	ST	D	L
4	0	A	ST	D	D

O resultado nesta análise é binário, vitória “W” ou derrota “L”, removendo os empates que são inexpressivos na amostra.

#ENCODING

```
dfENC=dfEncode(dfLDW)
dfENC.shape
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
['A', 'B', 'C', 'D']
['NS', 'ST']
['A', 'B', 'C', 'D']
['D', 'L', 'W']
```

ENCODED DF

```
BT_Pos HP SWT POT LDW
0      5  2  1  0  0
1      6  3  1  2  1
2      7  2  1  3  1
3      8  0  1  3  1
4      0  0  1  3  0
(1656, 5)
```

#REMOVENDO DRAWS

```
dfENC=dfENC[dfENC['LDW'] != 0]
dfENC.shape
```

(1635, 5)

A árvore de decisão é apresentada na Figura 11 seguido da tabela verdade e parâmetros da solução.

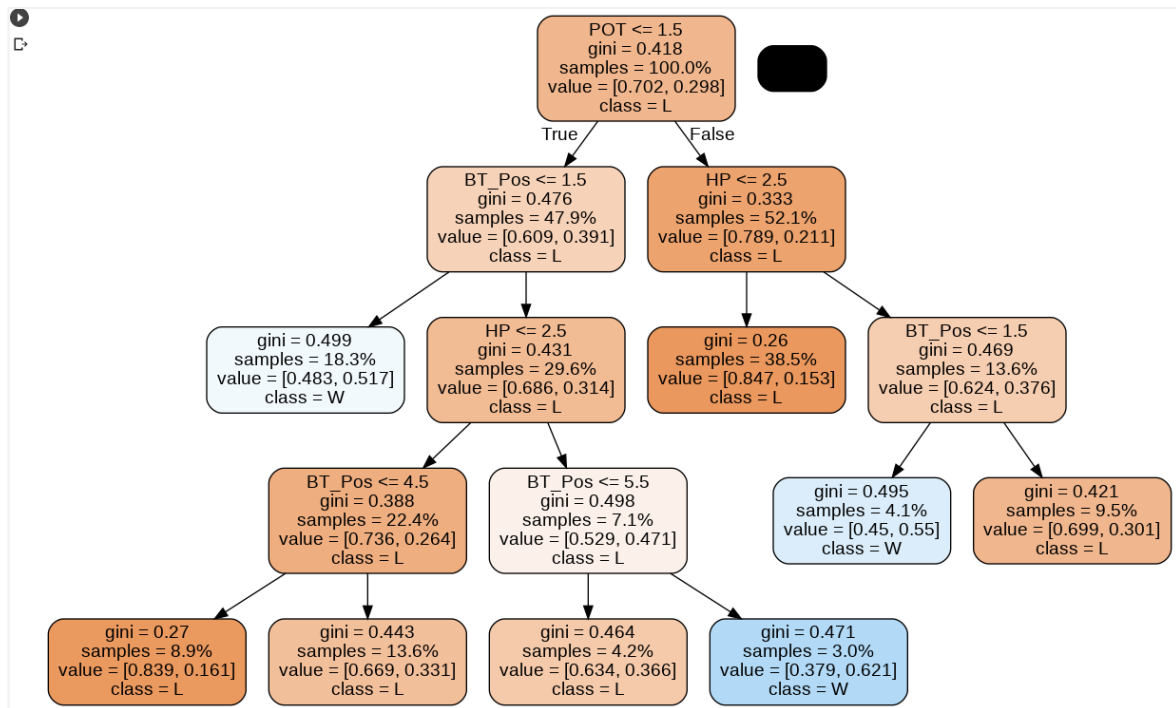


Figura 11. Árvore de Decisão

```
PARAMETROS -> {'criterion': 'gini', 'min_impurity_decrease': 0.002,
'min_weight_fraction_leaf': 0.017, 'splitter': 'best'}
```

	L	W
L	361	109
W	104	80

Acurácia do Treino: 72.0 %

Acurácia do Teste: 67.4 %

	precision	recall	f1-score	support
1	0.78	0.77	0.77	470
2	0.42	0.43	0.43	184
accuracy			0.67	654
macro avg	0.60	0.60	0.60	654
weighted avg	0.68	0.67	0.68	654

Resultado do Modelo Naive-Bayes

▶ Erros de Classificação: 23.9 %

✕ Acurácia do Teste: 76.1 %

	precision	recall	f1-score	support
L	0.76	0.97	0.86	239
W	0.73	0.18	0.29	88
accuracy			0.76	327
macro avg	0.75	0.58	0.57	327
weighted avg	0.75	0.76	0.70	327

	L	W
L	233	6
W	72	16



## 6. Interpretação dos Resultados

A partir do histórico de mão do jogador foi possível obter uma tabela que relaciona seu ganho em relação a alguns parâmetros fundamentais do poker e permite algumas análises em relação ao comportamento do jogador. O modelo desconsidera mãos inicialmente muito forte que obviamente tendem a um maior ganho.

Uma primeira análise do histórico de jogo indica que o jogador joga preferencialmente com mãos do mesmo naipe o que é vantajoso, porém tende a jogar com mão de naipes diferentes quando está com uma quantidade maior de fichas. Da mesma forma, as maiores perdas ocorrem quando o jogador se envolve em mão com muitas fichas ou quando seu *Stack* é muito superior ao *Pot*. O ganho do jogador praticamente nulo, indicando que seu padrão de jogo não tem refletido em lucro efetivo sendo esta a principal reclamação do cliente.



O modelo de regressão linear obteve coeficientes condizentes com a realidade, apresentado para o ganho uma relação positiva para cartas naipadas e força das mão e negativa para *Stack* e *Pot*. Este resultado indica que o modelo foi capaz de compreender os princípios básicos que levam à vitória no poker. O melhor modelo para regressão polinomial foi o RIDGE considerando  $\alpha$  de 0.1 e 20 parâmetros.

A árvore de decisão pode ajudar o jogador a tomar decisões a cada mão. Este modelo indica jogar preferencialmente com mão cuja razão *Pot/Stack* inferior a 1.5 e em posições menores que 2. Nos demais casos, a força da mão devem ser superior a 2.5 que corresponde a par de oitos “88”. Estes são os principais componentes que levam este jogador à vitória.

A matriz de confusão indica que o modelo tem uma boa capacidade de prever mão perdedora, porém precisa evoluir para avaliar melhor as mãos vencedora. Neste ponto vale ressaltar que a base de aprendizado do modelo é um histórico de um jogador amador que como demonstrado não obtém lucro com seus jogos. Espera-se um resultado muito melhor caso o modelo utilize como banco de dados o histórico de um jogador profissional lucrativo.

Ademais, diversos outros parâmetros podem ser obtidos a partir do histórico e considerados no modelo de forma a contribuir na tomada de decisão em cada etapa da mão e podendo posteriormente evoluir para um modelo autônomo.

## 7. Apresentação dos Resultados

<div> <div>NING CANVAS</div> <div> Designed for: TCC PUC Minas Designed by: <i>Rodrigo Xavier</i> Date: 14/08/2022 Iteration: 1 </div> </div>			
<b>DECISÃO</b>  <p>Os modelos ajudam a tomar decisões durante as mãos de um jogo de poker com base da análise de ações vitoriosas do jogador.</p>	<b>PÚBLICO ALVO</b>  <p>Jogadores de Poker que queiram entender melhor as estatísticas do jogo e analisar suas ações na mesa de forma a torná-las mais lucrativas</p>	<b>FONTE DE DADOS</b>  <p>Histórico de mãos obtido em Plataforma de jogos de poker.</p>	<b>ONDE CONSEGUIR</b>  <p>Poker Stars 888 Poker Full Tilt Poker Party Poker</p>

## 8. Links

<https://youtu.be/aArSSwsSamM>

[https://github.com/rodrigoxal/TCC\\_PUCMinas](https://github.com/rodrigoxal/TCC_PUCMinas)

## APÊNDICE

### Scripts

# -\*- coding: utf-8 -\*-

"""PSL-Regression.ipynb

Automatically generated by Colaboratory.

Original file is located at

<https://colab.research.google.com/drive/14Lpfe2ba3pKVYwU0fk2KxJUjSxbZaYoE>

MACHINE POKER LEARNING

"""

#Bibliotecas

import pandas as pd

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier as d3

from sklearn.preprocessing import LabelEncoder as le #Classes nominais para binaria

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

#Configuração de impressão

np.printoptions(threshold=None, precision=2)

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.max\_rows', 500)

pd.set\_option('precision',2)

```
"""-> FUNÇÕES"""
```

```
#Relativizar em relação ao maior valor
```

```
def rlta (lt_abs):
```

```
    lt_rlta = list()
```

```
    lt_max=abs(max(lt_abs))
```

```
    for i in lt_abs:
```

```
        lt_rlta.append(i/lt_max)
```

```
    return(lt_rlta)
```

```
#Classificação LDW
```

```
def ldw_cls (lt):
```

```
    lt_cls = list()
```

```
    for i in lt:
```

```
        if i<0:
```

```
            lt_cls.append('L')
```

```
        elif i>0:
```

```
            lt_cls.append('W')
```

```
        elif i==0.0:
```

```
            lt_cls.append('D')
```

```
        else:
```

```
            lt_cls.append('D')
```

```
    return (lt_cls)
```

```
#Classificação Swt
```

```
def swt_cls (lt):
```

```
    lt_cls = list()
```

```
    for i in lt:
```

```

if i==0:
    lt_cls.append('NS')
else:
    lt_cls.append('ST')

return (lt_cls)

```

#Classificador de Mãos

```

def HP_cls (HP):
    #Classes
    A=HP.describe().get('25%')
    B=HP.describe().get('50%')
    C=HP.describe().get('75%')
    D=HP.describe().get('max')

    classlt=[A,B,C,D]
    classltstr=['A','B','C','D']

    print(classlt)

```

#Classificacao

```

HPlt=list()

for i in HP:
    n=0
    for j in classlt:
        if (i<=j):
            HPlt.append(classltstr[n])
            break
    n+=1

```

```
return(HPlt)
```

```
#DATA FRAME ENCODING
```

```
def dfEncode (df):
```

```
    le = preprocessing.LabelEncoder()
```

```
    dfENC=df.copy()
```

```
    for i in df.keys():
```

```
        dfENC[i]=le.fit_transform(df[i])
```

```
    print(list(le.classes_))
```

```
    print('\nENCODED DF\n',dfENC.head())
```

```
    return(dfENC)
```

```
""""-> BEGIN
```

```
-> KNIME -> db -> python
```

```
""""
```

```
#Download do DataSet
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
for fn in uploaded.keys():
```

```
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))
```

```
#DataSet -> Dataframe
```

```
df=pd.read_csv('PokerData.csv')
```

```
print('Dimensões do DataSet --> ', df.shape)
```

```
df.head()
```

```
gain=list()
```

```

gain.clear()

#Ganho hand by hand
#Identificador: Numero do torneio
i=0
n=1
for x in df['Tournamnet_Number']:
    if i<df.shape[0]-1:
        if x==df['Tournamnet_Number'][i+1]:           #Calcula o ganho para mãos no mesmo
torneio
            gain.append(df['Stack_BB'][i+1]-df['Stack_BB'][i]) #Stack i+1 - Stack i
        else:
            gain.append(np.nan)
        n+=1
    i+=1

gain.append(np.nan)
print('Tamanho da lista --> ', len(gain))
print('\nNumero de Torneios --> ',n)

#DataFrame Completo com ganho das mãos
df['gain']=gain
df.head()

""""-> TRATAMENTO E ANALISE PRIMARIA""""

#Excluir col_torneios*
del df['Tournamnet_Number']

#Removendo Nan Values
#Referente a maos iniciais e finais dos torneios
dftr=df[~np.isnan(df['gain'])]

```

```

print('SHAPE --> ', dftr.shape)
print('Perda de dados: ',100*(1-(len(dftr)/df.shape[0])),'%')

gain_rlta=rlta(dftr['gain'])
sns.scatterplot(data=gain_rlta)
print('Dimensão:', len(gain_rlta))

#DataFrame com ganho Relativizado
dftr.loc[:,['gain']]=gain_rlta
dftr.head()

sns.pairplot(dftr, hue='Swited', corner=True, palette='coolwarm')
#hue: parametro de coloração
print('BY SWITED HANDS \n')

dftr['gain'].describe()

dftr.to_csv('PSLpdDFckp1.csv')
files.download('PSLpdDFckp1.csv')

"""*CKP 1
-> Fim do PreProcessamento
"""

#Download do DataSet
from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

```



```

#DataSet -> Dataframe
dftr=pd.read_csv('PSLpdDFckp1.csv', index_col=0)
print('Dimensões do DataSet --> ', dftr.shape)
dftr.head()

""""--> CORTE DE OUTLIERS""""

# Corte de Outliers

#Removendo Outliers
dftrH=dftr[(abs(dftr['Pot_BB'])<500)] # Corte de potes maiores que 500 BB
dftrH=dftrH[dftrH['Hand_Power']<605] # Top Hands - JJ
(605#dftr_out=dftr[(abs(dftr['gain'])<200)] # Ganhos ou perdas acima de 200 BB
#dftr_out1=dftr_out[dftr_out['Stack_BB']<250] # High Stacks

print('SHAPE --> ', dftrH.shape)
print('Perda de dados: ',100*(1-(len(dftrH)/dftr.shape[0])),'%')

""""-> HARMONIZAÇÃO log""""

#LogModel
from numpy.ma.core import log10

dflog=dftrH.copy()
#Ajustando dados com logaritmo
dflog['Hand_Power']=np.log10(dftr['Hand_Power'])
dflog['Stack_BB']=np.log10(dftr['Stack_BB'])
dflog['Pot_BB']=np.log10(dftr['Pot_BB'])

""""-> SAVE LOG DATA""""

```

```

dflog.to_csv('PSLdflog.csv')
files.download('PSLdflog.csv')

""""*CKP2 -> LOG DATA""""

#Download do DataSet
from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

#DataSet -> Dataframe
dflog=pd.read_csv('PSLdflog (1).csv', index_col=0)
print('Dimensões do DataSet --> ', dflog.shape)
dflog.head()

dflog.head()

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

S= dflog['Stack_BB'].to_numpy().reshape(-1,1)
print(scaler.fit(S))
print('Stack_BB')
print(scaler.data_max_)
print(scaler.data_min_)

S= dflog['Pot_BB'].to_numpy().reshape(-1,1)
print(scaler.fit(S))

```

```

print('Pot_BB')
print(scaler.data_max_)
print(scaler.data_min_)

S= dflog['Hand_Power'].to_numpy().reshape(-1,1)
print(scaler.fit(S))
print('Hand_Power')
print(scaler.data_max_)
print(scaler.data_min_)

#histograma
b=20
fig, ax = plt.subplots(2, 4 , figsize=(20,5))
plt.suptitle('TRANSFORMAÇÃO DE DADOS')

ax[0,0].hist(dftr['Hand_Power'], bins=b)
ax[0,0].set_ylabel('HP')

ax[0,1].hist(dflog['Hand_Power'], bins=b)
ax[0,1].set_ylabel('Log10 HP')
ax[0,1].grid()

ax[0,2].hist(dftr['Stack_BB'], bins=b)
ax[0,2].set_ylabel('Stack')

ax[0,3].hist(dflog['Stack_BB'], bins=b)
ax[0,3].set_ylabel('Log10 Stack')
ax[0,3].grid()

ax[1,0].hist(dftr['Pot_BB'], bins=b)
ax[1,0].set_ylabel('Pot')
ax[1,0].grid()

```

```
ax[1,1].hist(dflog['Pot_BB'], bins=b)
ax[1,1].set_ylabel('Log10 Pot')
ax[1,1].grid()
```

```
ax[1,2].hist(dflog['gain'], bins=b)
ax[1,2].set_ylabel('gain')
ax[1,2].grid()
```

```
sns.pairplot(dflog, hue='Swited', markers=['o','s'], corner=False, palette='coolwarm')
#hue: parametro de coloração
print('BY SWITED HANDS \n')
```

```
""""-> FEATURES / TARGET""""
```

```
#Features - Target
y=dflog['gain'].to_numpy()
X=dflog.iloc[:,5]
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
y=scaler.fit_transform(y.reshape(-1,1))
```

```
print(y)
X.head()
```

```
""""-> 1 MODELO LINEAR""""
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=5)
```

```
print('X train shape -> ', X_train.shape)
print('y train shape -> ', y_train.shape)
```

#Regressão Linear

```
Inr=LinearRegression()
```

```
Inr.fit(X_train, y_train) #Fitando
```

```
y_prev=Inr.predict(X_test) #previsão do teste
```

```
print('Acurácia do Treinamento: {:.2f}'.format(Inr.score(X_train,y_train)))
```

```
print('Acurácia do Teste: {:.2f}'.format(Inr.score(X_test,y_test))) #y_test é comparado com  
y_prev
```

```
""""-> COEFICIENTES PRIMARIOS""""
```

#Coeficientes

```
import math
```

```
atributos=list(dftr.iloc[:,0:dftr.shape[1]-1])
```

```
print(atributos)
```

```
print(Inr.coef_)
```

#Nota -> Coeficientes condizem com a realidade

#ERRO

```
errabs=np.abs(y_prev-y_test)
```

```
errperc=errabs/list(y_test)
```

```
#print("\nErro Percentual:\n\n Max: {0: .2f}\n Mean: {1: .2f}\n Min: {2:  
.2f}'.format(np.max(errperc), np.mean(errperc), np.min(errperc)))
```

#ERRO GRAFICO

```
plt.figure(figsize=(8,4))
```

```
plt.errorbar(np.arange(y_test.size),list(y_test),yerr=errabs,capsize=3,ecolor='r',fmt='.')
```

```
plt.title("Erros da previsão")
```

```
plt.grid()
```

```

#ERRO_GRAFICO
fig, ax = plt.subplots(1, 2 , figsize=(10,4))
plt.suptitle('ERROS')
ax[0].plot(errabs, '.')
ax[0].set_ylabel('Erro Absoluto')

ax[1].plot(errperc, '.')
ax[1].set_ylabel('Erro Percentual')
ax[1].grid()

""""-> REGRESSÃO LINEAR COM ATRIBUTOS POLINOMIAIS""""

from sklearn.preprocessing import PolynomialFeatures

Xpoly=PolynomialFeatures(degree=2, include_bias=False).fit_transform(X)

('Xpoly SHAPE --> ', Xpoly.shape)

X_train, X_test, y_train, y_test = train_test_split(Xpoly, y, test_size=0.25, random_state=369)

print('X train shape -> ', X_train.shape)
print('y train shape -> ', y_train.shape)

#Regressão Linear

lnr=LinearRegression()
lnr.fit(X_train, y_train) #Fitando
y_prev=lnr.predict(X_test) #previsão do teste

print('Acurácia do Treinamento: {:.2f}'.format(lnr.score(X_train,y_train)))

```

```
print('Acurácia do Teste: {:.2f}'.format(lnr.score(X_test,y_test))) #y_test é comparado com
y_prev
```

```
""""->AJUSTE DE PARÂMETROS""""
```

```
from sklearn.linear_model import Ridge, Lasso
```

```
#Ridge
```

```
ridge=Ridge().fit(X_train, y_train) #Fitando
```

```
print('Acurácia do Treinamento: {:.2f}'.format(ridge.score(X_train,y_train)))
```

```
print('Acurácia do Teste: {:.2f}'.format(ridge.score(X_test,y_test))) #y_test é comparado com
y_prev
```

```
y_prevRid=ridge.predict(X_test) #previsão do teste
```

```
#Ridge -> Ajuste de Parâmetros e Alpha
```

```
ridge_alpha=Ridge(alpha=0.1).fit(X_train, y_train) #Fitando
```

```
print('Acurácia do Treinamento: {:.2f}'.format(ridge_alpha.score(X_train,y_train)))
```

```
print('Acurácia do Teste: {:.2f}'.format(ridge_alpha.score(X_test,y_test))) #y_test é
comparado com y_prev
```

```
y_prevRidAlpha=ridge_alpha.predict(X_test) #previsão do teste
```

```
#Ridge -> Ajuste de Parâmetros e Alpha
```

```
ridge_alphaHalf=Ridge(alpha=0.5).fit(X_train, y_train) #Fitando
```

```
print('Acurácia do Treinamento: {:.2f}'.format(ridge_alphaHalf.score(X_train,y_train)))
```

```
print('Acurácia do Teste: {:.2f}'.format(ridge_alphaHalf.score(X_test,y_test))) #y_test é
comparado com y_prev
```

```
y_prevRidAlphaHalf=ridge_alphaHalf.predict(X_test) #previsão do teste
```

```
#Comparação
```

```
plt.figure(figsize=(9,6))
plt.plot(ridge.coef_, 's', label='Ridge A1')
plt.plot(ridge_alphaHalf.coef_, 's', label='Ridge A0.5')
plt.plot(ridge_alpha.coef_, '^', label='Ridge A0.1')
plt.plot(lnr.coef_, 'o', label='LnR')
```

```
plt.ylim(-0.25, 0.25)
plt.xlabel('Coeficientes')
plt.ylabel('Magnitude')
plt.legend()
plt.hlines(0,0,len(lnr.coef_))
```

```
#LASSO (Regressão Linear com Regularização L1)
```

```
#REduz e elimina coeficientes que não agregam ao modelo
```

```
#ALPHA -> default=1 e alpha =0 é uma lnr
```

```
lasso=Lasso(alpha=0.000005).fit(X_train, y_train) #Fitando
```

```
print('Acurácia do Treinamento: {:.2f}'.format(lasso.score(X_train,y_train)))
```

```
print('Acurácia do Teste: {:.2f}'.format(lasso.score(X_test,y_test))) #y_test é comparado com
y_prev
```

```
print('Numero de Coeficientes: ', np.sum(lasso.coef_ != 0))
```

```
y_prevLasso=lasso.predict(X_test) #previsão do teste
```



```
"""RIDGE X LASSO"""
```

```
#Comparação
```

```
plt.figure(figsize=(9,6))
```

```
plt.plot(lasso.coef_, '^', label='Lasso')
```

```
plt.plot(ridge_alpha.coef_, 'x', label='Ridge')
```

```
plt.ylim(-0.25, 0.25)
```

```
plt.xlabel('Coeficientes')
```

```
plt.ylabel('Magnitude')
```

```
plt.legend()
```

```
plt.hlines(0,0,len(lasso.coef_))
```

```
"""MODELO QUALITATIVO LDW
```

```
L -> -1 D -> 0 W ->1
```

```
"""
```

```
!pip install pydotplus
```

```
!pip install dtreeviz
```

```
from sklearn import tree, datasets
```

```
from sklearn.tree import DecisionTreeClassifier as d3
```

```
from sklearn import preprocessing
```

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
#Download do DataSet
```

```

from google.colab import files
uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

#DataSet -> Dataframe
dflog=pd.read_csv('PSLdflog (1).csv', index_col=0)
print('Dimensões do DataSet --> ', dflog.shape)

#Descrição de HandPower
sns.boxplot(dflog['Hand_Power'])

#CLASSIFICACAO
dflogLDW=dflog.copy()

#Classificação de Hand Power
print('HAND POWER CLASSES')
dflogLDW['HP']=HP_cls(dflogLDW['Hand_Power'])
del dflogLDW['Hand_Power']

#Classificação Swited
dflogLDW['SWT']=swt_cls(dflog['Swited'])
del dflogLDW['Swited']

# Stack/Pot Class
print('\nSTACK CLASSES')
dflogLDW['POTstk']=dflogLDW['Pot_BB']/dflogLDW['Stack_BB']
dflogLDW['POT']=HP_cls(dflogLDW['POTstk'])
del dflogLDW['POTstk']
del dflogLDW['Stack_BB']
del dflogLDW['Pot_BB']

#Classificação do Ganho

```

```

dflogLDW['LDW']=ldw_cls(dflog['gain'])
del dflogLDW['gain']
#
print('\nSHAPE -> ',dflogLDW.shape)
dfLDW=dflogLDW.copy()
dfLDW.head()

""""-> Árvore de Decisão""""

#ENCODING
dfENC=dfEncode(dfLDW)
dfENC.shape

#REMOVENDO DRAWS
dfENC=dfENC[dfENC['LDW'] != 0]
dfENC.shape

print('\n',100*dfENC.corr())
plt.figure()
ax=pd.plotting.parallel_coordinates(dfENC, 'LDW',color=('#156270', '#C7F464'))

#Features - Target

y=dfENC['LDW']
X=dfENC.iloc[:, :-1]

print(y.head())
X.head()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=6)

print('X train shape -> ', X_train.shape)

```

```
print('y train shape -> ', y_train.shape)
```

```
print('\nX test shape -> ', X_test.shape)
```

```
print('y test shape -> ', y_test.shape)
```

```
#D3 Model
```

```
#1. d3 -> GridSearch
```

```
dset3=d3(random_state=3)
```

```
parm = {'criterion':('gini','entropy'),'splitter':('best','random'),
'min_impurity_decrease':(0.002,0.0025), 'min_weight_fraction_leaf':(0.017,0.018, 0.020)}
```

```
dset3Grid=GridSearchCV(dset3,param_grid=parm)
```

```
#2. FitGrid
```

```
yPredictG=dset3Grid.fit(X_train, y_train).predict(X_test)
```

```
#Matriz de Confusao
```

```
cnf_matrix=confusion_matrix(y_test, yPredictG)
```

```
cnf_Df=pd.DataFrame(data=cnf_matrix, index=('L','W'), columns=('L','W'))
```

```
print('PARAMETROS -> ',dset3Grid.best_params_)
```

```
print('\n', cnf_Df)
```

```
print('\nAcurácia do Treino: {0:.1f} %'.format(100*dset3Grid.score(X_train, y_train)))
```

```
print('\nAcurácia do Teste: {0:.1f} %\n'.format(100*accuracy_score(y_test, yPredictG)))
```

```
print(classification_report(y_test, yPredictG))
```

```
#parametros do grid
```

```
dset3Grid.get_params().keys()
```

```
""""-> O CAMINHO DA VITÓRIA""""
```

```

#BEST GRIDSEARCH D3
dset3=d3(random_state=0, criterion=dset3Grid.best_params_['criterion'],
splitter=dset3Grid.best_params_['splitter'],
        min_impurity_decrease=dset3Grid.best_params_['min_impurity_decrease'],
min_weight_fraction_leaf=dset3Grid.best_params_['min_weight_fraction_leaf'])

yPredict=dset3.fit(X_train, y_train).predict(X_test)

import pydotplus
from IPython.display import Image

#Dot Data

dot_data=tree.export_graphviz(dset3, out_file=None, proportion=True, rounded= True,
filled=True,
                             feature_names=X.columns[:], class_names=('L','W'))

#Graph

graph=pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

import graphviz
from graphviz import Graph

with open ("graph3.dot",'w') as f:
    f=tree.export_graphviz(dset3, out_file=f)
with open ("graph3.dot") as f:
    dot_graph=f.read()
graphviz.Source(dot_graph)

"""->NAIVE-BAYES"""

```

```

from sklearn.naive_bayes import GaussianNB

#Modelo
gnb=GaussianNB()
yPred=gnb.fit(X_train, y_train).predict(X_test)

#Avaliação

print('Erros de Classificação: {0:.1f} %'.format(100*(yPred != y_test).sum()/len(yPred)))

print('\nAcurácia do Teste: {0:.1f} %'.format( 100*accuracy_score(y_test, yPred)))
print(classification_report(y_test, yPred, target_names=('L','W')))

cnf_matrix=confusion_matrix(y_test, yPred)
cnf_Df=pd.DataFrame(data=cnf_matrix, index=('L','W'), columns=('L','W'))
cnf_Df

#Notas:
#Avaliar posição mais frequente
#Hero joga com mãos fracas
#Hero tende a jogar mãos offswited ao aumentar seu stack
#HandPower_Classes --> HP > 900 XR, 900 > HP > 800 : MR, 800 < HP < 600 : R, 600 < HP <
400 : E, HP<400 : C

#Retorna lista contendo DataFrames agrupados por Tournament_Number
#dfgb = [x for _, x in df.groupby('Tournamnet_Number')]

```