UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

ADA MARIA CAYRES FERNANDEZ
LEONARDO COPPOLA BIAZUCCI
MICHELLY RODRIGUES DA SILVA
THIAGO DE OLIVEIRA DEODATO
RAFAEL MARTINS CAMPOS

T4 - TÉCNICA DE DEPENDÊNCIA: REGRESSÃO LOGÍSTICA

SÃO PAULO - SP
2021

SUMÁRIO

1. INTRODUÇÃO

Será feito neste trabalho o uso da técnica de dependência conhecida como regressão logística sobre um recorte de banco de dados que diz respeito a partidas jogadas em *League of Legends*, jogo online famoso mundialmente. O uso da *RLog* nos permitirá descobrir ao final da análise a probabilidade de uma equipe vencer o jogo. Esta informação poderá nos dar uma vantagem significativa no ambiente competitivo, além de ajudar os estatísticos, analistas e apostadores envolvidos com o jogo.

A análise será feita utilizando códigos na linguagem *Python* com auxílio das bibliotecas *pandas*, *numpy*, *matplotlib*, *pylab*, *scipy, statsmodels, seaborn, sklearn*. A execução dos códigos foi realizada no *Google Colaboratory* (ou simplesmente *Colab*).

2. OBJETIVOS

Este documento tem como objetivo principal o estudo da relação entre as variáveis do banco de dados dos primeiros 10 minutos das partidas do jogo online conhecido mundialmente, *League of Legends*. Com o uso da técnica regressão logística, espera-se descobrir ao final da análise se é possível obter a probabilidade de uma equipe vencer o jogo. Esta análise é de profunda importância para competidores profissionais, estatísticos, analistas e apostadores envolvidos com o jogo.

Podemos dizer que a obtenção desse conhecimento é ainda mais eficaz, pois não é uma informação tão simples de ser obtida pelo jogador comum, pois necessita de profunda análise, uso de técnicas como a *RLog* e interpretação dos resultados .

Neste documento, temos como foco descobrir se é possível obter probabilidade da equipe azul vencer o jogo (variável binária: vitória ou derrota) utilizando *RLog* (regressão logística) dado a composição e o desempenho do time azul nos primeiros 10 minutos da partida no ranque diamante. Assim, poderemos ter uma noção melhor das chances do time azul após esse tempo de partida.

3. PRÉ-PROCESSAMENTO

Antes de iniciar a análise, criamos o *data frame* (recorte do banco de dados) com vinte e uma colunas de interesse.

Note que todas variáveis selecionadas são do time azul.

```
df = raw_data.filter(['gameId', 'blueWins', 'blueWardsPlaced', 'blueWardsDestroyed',
       'blueFirstBlood', 'blueKills', 'blueDeaths', 'blueAssists',
       'blueEliteMonsters', 'blueDragons', 'blueHeralds',
       'blueTowersDestroyed', 'blueTotalGold', 'blueAvgLevel',
       'blueTotalExperience', 'blueTotalMinionsKilled',
       'blueTotalJungleMinionsKilled', 'blueGoldDiff', 'blueExperienceDiff',
       'blueCSPerMin', 'blueGoldPerMin'])
df.info
```

```
<bound method DataFrame.info of          gameId  blueWins  ...  blueCSPerMin  blueGoldPerMin
0     4519157822         0  ...          19.5          1721.0
1     4523371949         0  ...          17.4          1471.2
2     4521474530         0  ...          18.6          1611.3
3     4524384067         0  ...          20.1          1515.7
4     4436033771         0  ...          21.0          1640.0
...          ...       ...  ...           ...             ...
9874  4527873286         1  ...          21.1          1776.5
9875  4527797466         1  ...          23.3          1623.8
9876  4527713716         0  ...          21.0          1590.3
9877  4527628313         0  ...          22.4          1445.9
9878  4523772935         1  ...          20.7          1626.6

[9879 rows x 21 columns]>
```

Figura 1: linhas de código do pré-processamento do banco de dados.

```
[ ]  df.isna().sum()
```

```
gameId                        0
blueWins                      0
blueWardsPlaced               0
blueWardsDestroyed            0
blueFirstBlood                0
blueKills                     0
blueDeaths                    0
blueAssists                   0
blueEliteMonsters             0
blueDragons                   0
blueHeralds                   0
blueTowersDestroyed           0
blueTotalGold                 0
blueAvgLevel                  0
blueTotalExperience           0
blueTotalMinionsKilled        0
blueTotalJungleMinionsKilled  0
blueGoldDiff                  0
blueExperienceDiff            0
blueCSPerMin                  0
blueGoldPerMin                0
dtype: int64
```

Figura 2: mostrando que não há NaN (valores indefinidos) no *data frame*.

4. PRÉ-REQUISITOS DO MODELO LOGÍSTICO

Foi determinado 5 suposições:

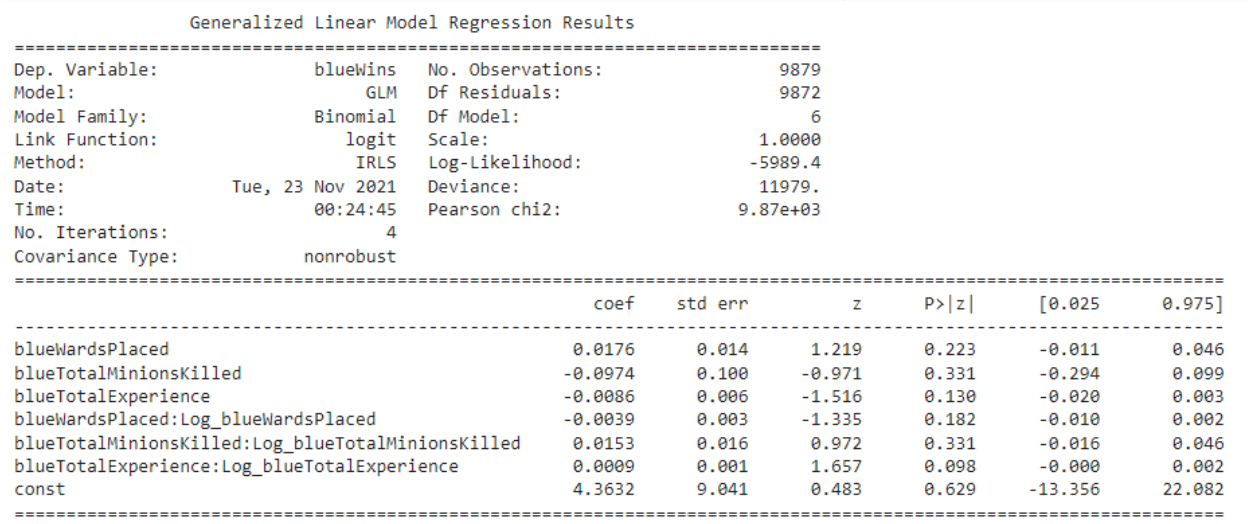
- Resposta binária;
- Relação linear entre o vetor das variáveis explicativas X e o *logit* da variável resposta Y;
- Sem *outliers* fortemente influentes;
- Ausência de multicolinearidade perfeita entre as variáveis independentes;
- Independência das observações.

Suposição 1 - Resposta binária: Como queremos prever uma resposta vitória ou derrota, que é uma resposta binária, a primeira suposição está satisfeita.

Suposição 2 - Linearidade das variáveis independentes e o *logit* da variável de resposta Y: Para checar a segunda suposição devemos selecionar as variáveis independentes do *data frame*, que são as variáveis de composição do time e aplicar o Teste de Box-Tidwell.

```
independent_vars = df.filter(['blueWardsPlaced', 'blueTotalMinionsKilled', 'blueTotalExperience'])

independent_vars_columns = ['blueWardsPlaced', 'blueTotalMinionsKilled', 'blueTotalExperience']

for var in independent_vars_columns:
  df[f'{var}:Log_{var}'] = df[var].apply(lambda x: x * np.log(x))

cols_to_keep = independent_vars_columns + df.columns.tolist()[-len(independent_vars_columns):]

X, y = df.loc[:, df.columns!='blueWins'], df.filter(['blueWins'], axis=1)

X_lt = df[cols_to_keep]

X_lt = sm.add_constant(X_lt, prepend=False)

logit_results = sm.GLM(y, X_lt, family=sm.families.Binomial()).fit()

print(logit_results.summary())
```

Figura 3: linhas de código para aplicação do Teste de Box-Tidwell.



Generalized Linear Model Regression Results

Dep. Variable:	blueWins	No. Observations:	9879
Model:	GLM	Df Residuals:	9872
Model Family:	Binomial	Df Model:	6
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-5989.4
Date:	Tue, 23 Nov 2021	Deviance:	11979.
Time:	00:24:45	Pearson chi2:	9.87e+03
No. Iterations:	4		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
blueWardsPlaced	0.0176	0.014	1.219	0.223	-0.011	0.046
blueTotalMinionsKilled	-0.0974	0.100	-0.971	0.331	-0.294	0.099
blueTotalExperience	-0.0086	0.006	-1.516	0.130	-0.020	0.003
blueWardsPlaced:Log_blueWardsPlaced	-0.0039	0.003	-1.335	0.182	-0.010	0.002
blueTotalMinionsKilled:Log_blueTotalMinionsKilled	0.0153	0.016	0.972	0.331	-0.016	0.046
blueTotalExperience:Log_blueTotalExperience	0.0009	0.001	1.657	0.098	-0.000	0.002
const	4.3632	9.041	0.483	0.629	-13.356	22.082

Figura 4: resultado do Teste de Box-Tidwell.

A significância estatística das seguintes variáveis baseadas no *p-value* (P > |z|):

- ***blueWardsPlaced:Log_blueWardsPlaced***
- ***blueTotalMinionsKilled:Log_blueTotalMinionsKilled***
- ***blueTotalExperience:Log_blueTotalExperience***

Vemos que as variáveis ***blueWardsPlaced:Log_blueWardsPlaced***, ***blueTotalMinionsKilled:Log_blueTotalMinionsKilled*** e ***blueTotalExperience:Log_blueTotalExperience*** tem seu *p-value* maior que 0.05, logo, são linearmente relatadas ao seu *logit*.

Ainda sim, pode-se verificar visualmente o teste da linearidade relatada ao *logit* das variáveis:

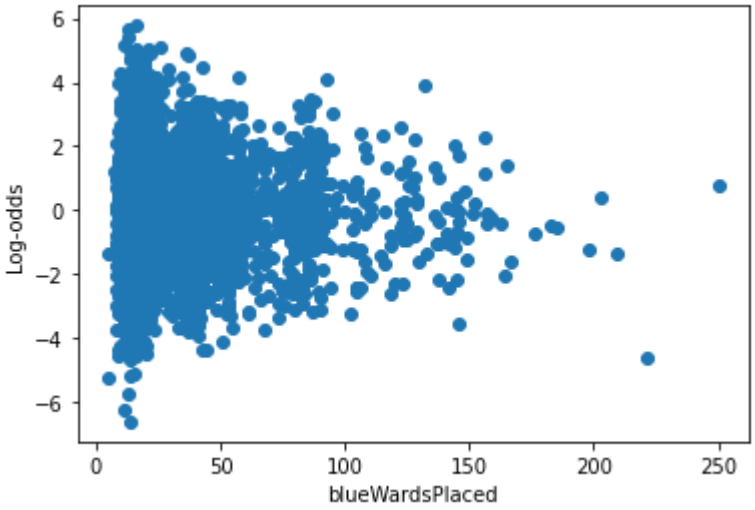


Figura 5: gráfico com valores de *blueWardsPlaced* x *log-odds* (ou *logit*)

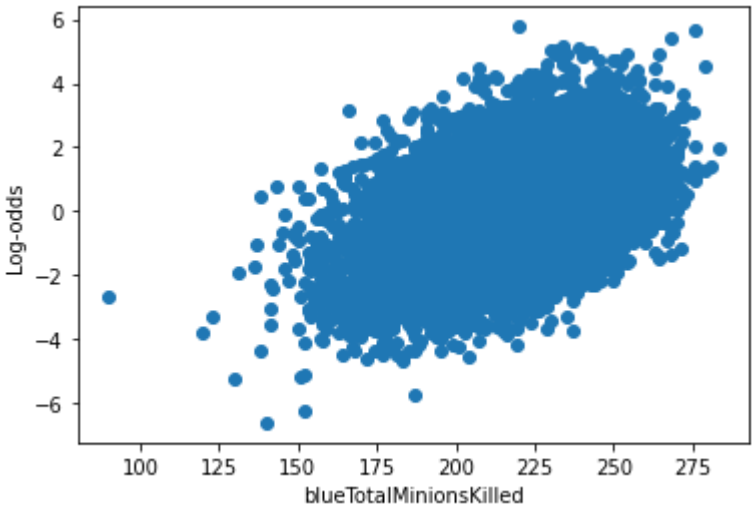


Figura 6: gráfico com valores de *blueTotalMinionsKilled* x *log-odds* (ou *logit*)

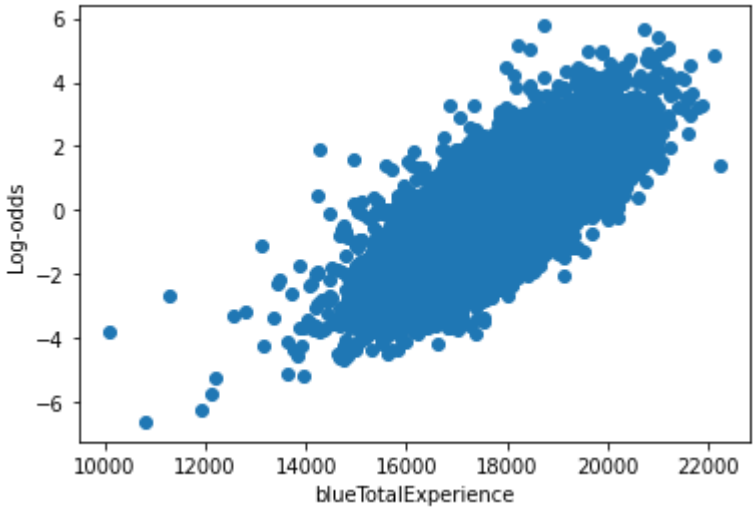


Figura 7: gráfico com valores de *blueTotalExperience* x *log-odds* (ou *logit*)

Desta forma, podemos concluir que a variável ***blueWardsPlaced:Log_blueWardsPlaced*** não é linearmente ao seu logit dada sua distribuição na figura 5.

Suposição 3 - Sem *outliers* fortemente influentes: Utilizou-se a Distância de Cook para medir a influência de uma observação.

	cooks_d	std_resid
1194	0.001559	4.051948
7286	0.001316	3.417986
3359	0.001197	4.771672
9608	0.001037	6.739349
8554	0.001030	5.661852

Figura 8: Os 5 *outliers* com maior influência do data frame.

Na tabela acima, identificamos todos os *outliers* mais influentes do *data frame*, ou seja, dados com valores residuais padronizados absolutos maiores que 3 (exibindo os cinco primeiros). Com isso, bastou excluí-los do *data frame* original.

```
df = df.drop(diagnosis_df[(diagnosis_df['cooks_d'] > cook_threshold) &
                          (diagnosis_df['std_resid'] > 3)].index)
```

Figura 9: linha de código da exclusão de *outliers* mais influentes.

Suposição 4 - ausência de multicolinearidade perfeita entre as variáveis independentes: Para realizar tal teste, usaremos o *Variance Inflation Factor* (VIF).

	variables	VIF
0	blueWardsPlaced	2.531855
1	blueTotalMinionsKilled	146.181893
2	blueTotalExperience	149.421702

Figura 10: resultado do VIF sobre as variáveis independentes.

Valores de VIF que excedem 5 ou 10 indicam alto grau de multicolinearidade. Logo, pelo teste, vemos que existe multicolinearidade entre as variáveis ***blueTotalMinionsKilled*** e ***blueTotalExperience***.

Outra forma de verificar a multicolinearidade é por meio da matriz de correlação. Como não existe multicolinearidade perfeita (igual a 1), seguiremos com a aplicação do modelo utilizando as variáveis ***blueTotalMinionsKilled*** e ***blueTotalExperience***.

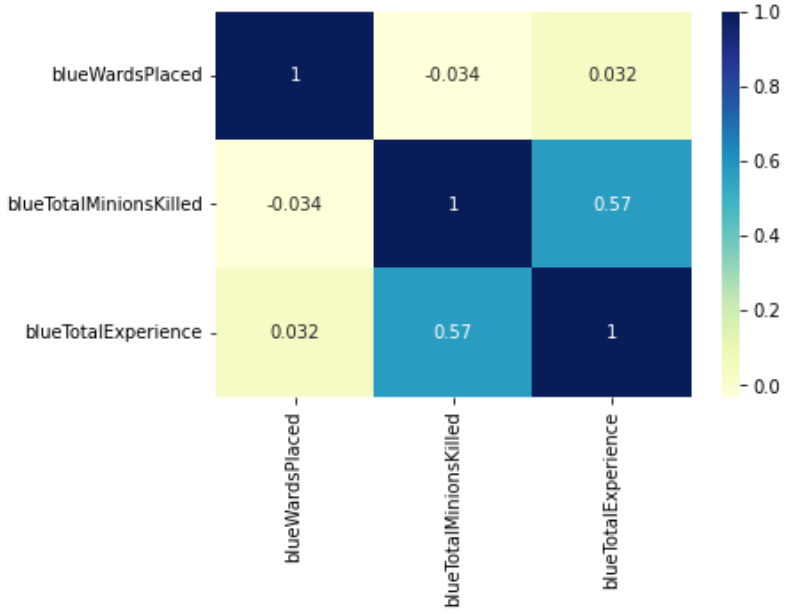


Figura 11: matriz de correlação das variáveis independentes.

Suposição 5 - independência das observações: As observações devem ser independentes entre si. Dado que cada linha do *data frame* consiste de uma partida diferente (***gameID*** único), podemos concluir que as observações são independentes.

Ainda assim, para testar tal condição, podemos plotar uma série residual com os resíduos de desvio do modelo *logit* versus o número de linhas do *data frame*.

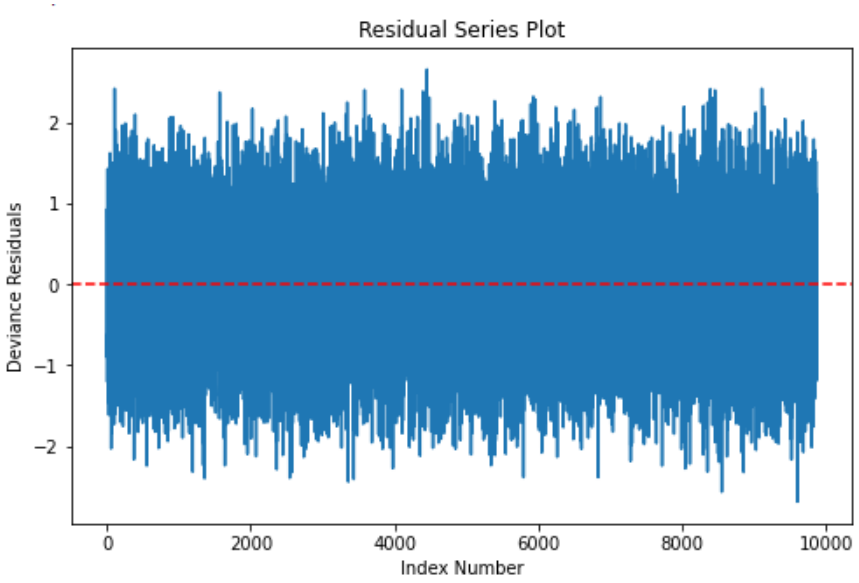


Figura 12: gráfico série residual: resíduos de desvio do modelo *logit* x número de linhas do *data frame*.

Como os resíduos no gráfico parecem estar espalhados aleatoriamente em torno da linha central, podemos inferir que a suposição é satisfeita.

5. APLICANDO A REGRESSÃO LOGÍSTICA

Foi feito um pré-processamento do *data frame* para aplicação do modelo logístico, dividindo 70% das linhas para treino e 30% das linhas para teste no *sklearn*

```
from sklearn.model_selection import train_test_split

df = df.drop(diagnosis_df[(diagnosis_df['cooks_d'] > cook_threshold) &
                          (diagnosis_df['std_resid'] > 3)].index)

normalized_df = (df - df.min())/(df.max()-df.min())

X, y = normalized_df.loc[:, normalized_df.columns!='blueWins'], normalized_df.filter(['blueWins'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Figura 13: linhas de código do pré-processamento para aplicação do modelo logístico.

O modelo foi criado utilizando o método de regressão logística do *sklearn*.

```
from sklearn.linear_model import LogisticRegression
reg =  LogisticRegression().fit(X_train,y_train)
y_pred = reg.predict(X_test)

print(y_pred)
```

```
[0. 1. 1. ... 1. 0. 0.]
```

Figura 14: linhas de código da criação do modelo de regressão logística através do *sklearn.*

6. AVALIAÇÃO DA ADEQUAÇÃO DO MODELO

A acurácia do modelo foi medida e obteve-se o valor 73,507% (cerca de 73% dos vencedores das partidas foram previstos pelo modelo com sucesso). Um valor interessante levando em conta que a análise está sendo feita apenas em cima dos 10 primeiros minutos das partidas.

```
count_correct = 0

for x in range(y_test.size):
  if(y_test.iloc[x]['blueWins'] == y_pred_bin.item(x)):
    count_correct = count_correct + 1
accuracy = count_correct / y_test.size * 100
print("A acurácia do modelo é de ", end = '')
print("%.3f" % accuracy, end = '')
print("%", end = '')
```

```
A acurácia do modelo é de 73.507%
```

Figura 15: linhas de código para se obter a acurácia do modelo criado.

```
print("Coeficientes da equação do modelo:\n{}\n".format(reg.coef_))

print("Bias adicionadas à função de decisão:\n{}".format(reg.intercept_))
```

```
Coeficientes da equação do modelo:
[[ 0.11995714 -0.60760718 -0.14276772  0.12018092  0.5987327  -2.32750414
  -0.28981735  0.21543567  0.40790924  0.02296211 -0.5092463   1.52250711
  -0.44277886  0.09355183 -0.25579373  0.49291541  4.58480046  4.38569405
  -0.25579373  1.52250711]]

Bias adicionadas à função de decisão:
[-5.32458669]
```

Figura 16: coeficientes da equação do modelo e *bias* adicionadas a função de decisão.

7. INTERPRETAÇÃO DOS RESULTADOS

No resultado apresentado pelos cálculos utilizando regressão logística foi de uma predição com acurácia de 73,507%. Este valor representa a quantidade de partidas que o algoritmo foi capaz de realizar uma predição correta, utilizando apenas os dados dos primeiros 10 minutos de partida.

Antes de tirarmos conclusões sobre o quão efetiva essa acurácia obtida pelo algoritmo é, temos que ressaltar pontos importantes sobre como o jogo funciona. O primeiro ponto é de que nós estamos avaliando apenas os dados do ranque diamante (alto nível de competitividade), ou seja, nós não podemos supor que essa acurácia se manteria para jogadores de outros níveis, pois o peso dos fatores se alteraria, já que há diversas maneiras de se ganhar uma partida e jogadores de níveis diferentes de habilidade podem dar prioridades distintas a certas mecânicas do jogo. O segundo ponto é o de que as partidas do ranque diamante duram, em média, 26 minutos, entretanto, existem os casos mais extremos de partidas que duram 15, 40 e até 50 minutos. Nessa análise, nós não avaliamos se o algoritmo perde ou ganha efetividade nos extremos, pois não há a informação da duração das partidas no *data frame*, embora logicamente há a hipótese de que o algoritmo perderia efetividade caso as partidas se estendessem em duração, visto que o jogo tem mecanicas que permitem que o time que está perdendo se recupere na partida conforme o passar do tempo.

8. CONCLUSÃO

Tendo feito as ressalvas, nós podemos concluir que o algoritmo obteve uma acurácia alta. Esta conclusão se deve ao fato de que o jogo tem muitos fatores de imprevisibilidade, como por exemplo, mecânicas que dependem de frações de segundo para serem executadas e que podem resultar na derrota dos 5 jogadores da equipe. Além disso, existe um cenário em que a aplicação de um algoritmo desse tipo é muito relevante: apostas online, por exemplo. Caso realizássemos apostas nos vencedores das partidas, nós lucraríamos muito caso seguíssemos a conclusão do algoritmo.

9. REFERÊNCIAS

1. GOOGLE. **Google Colaboratory** - Disponível em: https://colab.research.google.com. Acesso em: 21 nov. 2021.

2. WIKIPÉDIA, A enciclopédia livre. **Regressão logística** - Disponível em: https://pt.wikipedia.org/wiki/Regress%C3%A3o_log%C3%ADstica. Acesso em: 17 nov. 2021.

3. STATICS SOLUTIONS. **Assumptions of Logistic Regression** - Disponível em: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/assumptions-of-logistic-regression/. Acesso em: 18 nov. 2021.

4. LEUNG, Kenneth. **Towards Data Science - Assumptions of Logistic Regression, Clearly Explained** - Disponível em: https://towardsdatascience.com/assumptions-of-logistic-regression-clearly-explained-44d85a22b290. Acesso em: 18 nov. 2021.