

# Smart Soccer – Modelo de predição de resultados de futebol com machine learning

Aluno: Allan Ferreira

Orientador: Bruno Rafael

Curso: MBA em Machine Learning

Linha de Especialização: Machine Learning aplicado à Sistemas de Recomendação

Ano: 2020

# Setor de mercado e a justificativa de tal seleção

- Grande crescente do mercado de trading esportivo.(Trading= compra/venda de um ativo “probabilidade” para valorização, ou “ODDs”)
- Alto volume de jogos(cobertura)
- Oportunidade de aplicar a solução para iniciantes e profissionais da área.



# Características e restrições de escopo do desafio

- Auxílio aos apostadores fornecendo recomendações de vencedores de partida de futebol.
- Restrição dos campeonatos que possuem datasets disponíveis
- Somente partidas de futebol
- A restrição do projeto será recomendar somente os vencedores, apesar de o trading esportivo possuir outras modalidades.



# Oportunidade vislumbrada que motivou o desenvolvimento do desafio

- Minha atuação no mercado de bolsa esportiva.
- Percepção dos métodos empíricos mesmo dos apostadores profissionais.

O que eles veem?	O que eles escutam?	O que eles falam?	O que eles fazem?	O que eles pensam?	O que eles sentem?
Falta de informações científicas	Que só começarão a ter lucro depois de muito tempo	É muito difícil "advinhar" resultados	Compram cursos e se frustram	Melhorar a margem de lucro nas apostas realizadas	Medo ao realizar os investimentos na carreira de trader
Falta de ferramentas para o auxílio da tomada de decisão durante a aposta	Que os traders profissionais tem um dom especial	Cada trader tem seu próprio método de prever	Desistem do mercado, a médio prazo	Diminuir o prejuízo de apostas equivocadas	Medo de que uma hora ou outra perderão toda a banca

Tabela 1-Mapa de empatia

# Fase de preparação

Os dados foram obtidos através de datasets em formato CSV no site datahub.io

Durante a fase de preparação dos dados foram utilizados os recursos disponíveis na biblioteca do PANDAS que é uma biblioteca open source voltada para análise de dados que torna fácil a manipulação de conjuntos de dados.

Também nos testes iniciais foram utilizados algoritmos da biblioteca SKLEARN como KNeighborsClassifier, GaussianNB e SVM



# Fase de preparação

```
[336] dsLaLiga1617 = pd.read_csv('laliga-1617_csv.csv')
      dsLaLiga1718 = pd.read_csv('laliga-1718_csv.csv')
      dsLaLiga1819 = pd.read_csv('laliga-1819_csv.csv')
      dsTodos = pd.concat([ dsLaLiga1819,dsLaLiga1617,dsLaLiga1718])
      dsTodos.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1140 entries, 0 to 379
Columns: 64 entries, AC to WHH
dtypes: float64(39), int64(19), object(6)
memory usage: 578.9+ KB
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarning: Sorting be
of pandas will change to not sort by default.
```

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

after removing the cwd from sys.path.

```
[338] dsTodos.sample(5)
```

	AC	AF	AR	AS	AST	AY	AwayTeam	B365A	B365D	B365H	BWA	BWD	BWH	Bb1X2	B
58	6	13	1	13	4	7	Valencia	3.10	3.40	2.30	3.10	3.60	2.20	38	
4	6	10	0	8	4	3	Sociedad	3.80	3.40	2.04	3.90	3.30	2.05	40	
330	4	9	0	17	6	1	Barcelona	1.40	5.25	7.00	1.42	5.00	7.00	34	
48	7	0	0	10	4	0	Elche	15.00	10.00	1.11	17.00	10.00	1.10	01	

# Fase de preparação

O resultado esperado para o modelo é uma eficiência de 60% de acertos em recomendações que possam auxiliar os traders esportivos nas apostas de partidas de futebol com 3 alternativas para vencedor: TIME DA CASA, VISITANTE ou EMPATE Durante os testes iniciais obtivemos os seguintes resultados:

```
[31] 7 print('previsoes entrando com dados no segundo tempo')
8
9 print("\nAcurácia Gaussian primeiro tempo:",metrics.accuracy_score(y_test_it, previsoesGaussianNB))
10 print("Recall Gaussian primeiro tempo:",metrics.precision_score(y_test_it, previsoesGaussianNB,average='macro'))
11 print("Precisão Gaussian primeiro tempo:",metrics.precision_score(y_test_it, previsoesGaussianNB,average='macro'))
12
13 print("Acurácia SVM primeiro tempo:",metrics.accuracy_score(y_test_it, previsoesSVM))
14 print("Recall SVM primeiro tempo:",metrics.precision_score(y_test_it, previsoesSVM,average='macro'))
15 print("Precisão SVM primeiro tempo:",metrics.precision_score(y_test_it, previsoesSVM,average='macro'))
16
17 print("Acurácia KNeighborsClassifier primeiro tempo:",metrics.accuracy_score(y_test_it, previsoesKNeighborsClassifier))
18 print("Recall KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_it, previsoesKNeighborsClassifier,average
19 print("Precisão KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_it, previsoesKNeighborsClassifier,av
20
```

previsoes entrando com dados no segundo tempo

```
Acurácia Gaussian primeiro tempo: 0.47619047619047616
Recall Gaussian primeiro tempo: 0.31355311355311355
Precisão Gaussian primeiro tempo: 0.31355311355311355
Acurácia SVM primeiro tempo: 0.48412698412698413
Recall SVM primeiro tempo: 0.46666666666666666
Precisão SVM primeiro tempo: 0.3199062233589088
Acurácia KNeighborsClassifier primeiro tempo: 0.44444444444444444
Recall KNeighborsClassifier primeiro tempo: 0.42287878787878785
Precisão KNeighborsClassifier primeiro tempo: 0.4201369436663554
```

# Fase de preparação

Na tabela acima vemos que o algoritmo que teve melhor resultado em termos de acurácia e precisão foi o SVM, com 48% de acurácia e 31% de precisão.

Para resolver o problema da baixa precisão, foi criada uma feature nomeada SUPERTIME(MANUAL)

```
1 def GetGrandeza(time):
2     supertimes= "Paris SG,Lyon,Roma,Milan,Inter,Lazio,Barcelona,Real Madrid,Ath Madrid,Leverkusen,Bayern Munich"
3     timesbons = "FiorentinaCelta de Vigo,Villarreal,Sevilla,Ath Bilbao,Werder Bremen,Wolfsburg,RB Leipzig,Schalke 04"
4     retorno = 3
5     if supertimes.find(time)>-1:
6         retorno = 1
7     if timesbons.find(time)>-1:
8         retorno = 2
9
10    return retorno
11
```



# Fase de preparação

Resultado com o critério de supertime:

```
[36] 2  previsoosSVM = modeloSVM.predict(X_test_1t)
3  previsoosKNeighborsClassifier = modeloKNeighborsClassifier.predict(X_test_1t)
4
5  from sklearn import metrics
6
7  print('previsoos entrando com dados no segundo tempo')
8
9  print("\nAcurácia Gaussian primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoosGaussianNB))
10 print("Recall Gaussian primeiro tempo:",metrics.precision_score(y_test_1t, previsoosGaussianNB,average='macro'))
11 print("Precisão Gaussian primeiro tempo:",metrics.precision_score(y_test_1t, previsoosGaussianNB,average='macro'))
12
13 print("Acurácia SVM primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoosSVM))
14 print("Recall SVM primeiro tempo:",metrics.precision_score(y_test_1t, previsoosSVM,average='macro'))
15 print("Precisão SVM primeiro tempo:",metrics.precision_score(y_test_1t, previsoosSVM,average='macro'))
16
17 print("Acurácia KNeighborsClassifier primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoosKNeighborsClassifier))
18 print("Recall KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_1t, previsoosKNeighborsClassifier,average='macro'))
19 print("Precisão KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_1t, previsoosKNeighborsClassifier,average='macro'))
20
```



previsoos entrando com dados no segundo tempo

Acurácia Gaussian primeiro tempo: 0.5317460317460317  
Recall Gaussian primeiro tempo: 0.5356125356125356  
Precisão Gaussian primeiro tempo: 0.5356125356125356  
Acurácia SVM primeiro tempo: 0.5158730158730159  
Recall SVM primeiro tempo: 0.5008333333333334  
Precisão SVM primeiro tempo: 0.3381921247089787  
Acurácia KNeighborsClassifier primeiro tempo: 0.42063492063492064  
Recall KNeighborsClassifier primeiro tempo: 0.39731060606060603  
Precisão KNeighborsClassifier primeiro tempo: 0.3713170163170163

# Fase de preparação

A acurácia continuava baixa, então depois do encontro com o orientador, houve a orientação para fazer este calculo de peso de SUPERTIMES automaticamente.

O dataframe foi dividido em 2(visto que tinha que colocar todos na mesma coluna e no dataframe, isto ocorre alternado, pois hora o time está em casa e hora é visitante).

Realizando um agrupamento sobre o montante de VITÓRIAS do time, foi observado o seguinte resultado, conforme Figura:



# Fase de preparação

```
[82] dsTimeHome = dsTodos.copy()
     dsTimeHome['NOME'] = dsTimeHome['HomeTeam']

     dsTimeHome['VITORIA'] = dsTimeHome.apply(lambda row: [0, 1][row['FTR'] == 0], axis=1)
     dsTimeHome['DERROTA'] = dsTimeHome.apply(lambda row: [0, 1][row['FTR'] == 1], axis=1)
     dsTimeHome['EMPATE'] = dsTimeHome.apply(lambda row: [0, 1][row['FTR'] == 2], axis=1)

     dsTimeFora = dsTodos.copy()
     dsTimeFora['NOME'] = dsTimeFora['AwayTeam']

     dsTimeFora['VITORIA'] = dsTimeFora.apply(lambda row: [0, 1][row['FTR'] == 1], axis=1)
     dsTimeFora['DERROTA'] = dsTimeFora.apply(lambda row: [0, 1][row['FTR'] == 0], axis=1)
     dsTimeFora['EMPATE'] = dsTimeFora.apply(lambda row: [0, 1][row['FTR'] == 2], axis=1)

     dsTimeUnificado = pd.concat([dsTimeHome, dsTimeFora])

     dsTimeUnificado.groupby('NOME')['VITORIA'].sum()
     grupo=dsTimeUnificado.groupby('NOME', as_index=False).agg({"VITORIA": "sum"}).sort_values('VITORIA', ascending=False)
     grupo
```



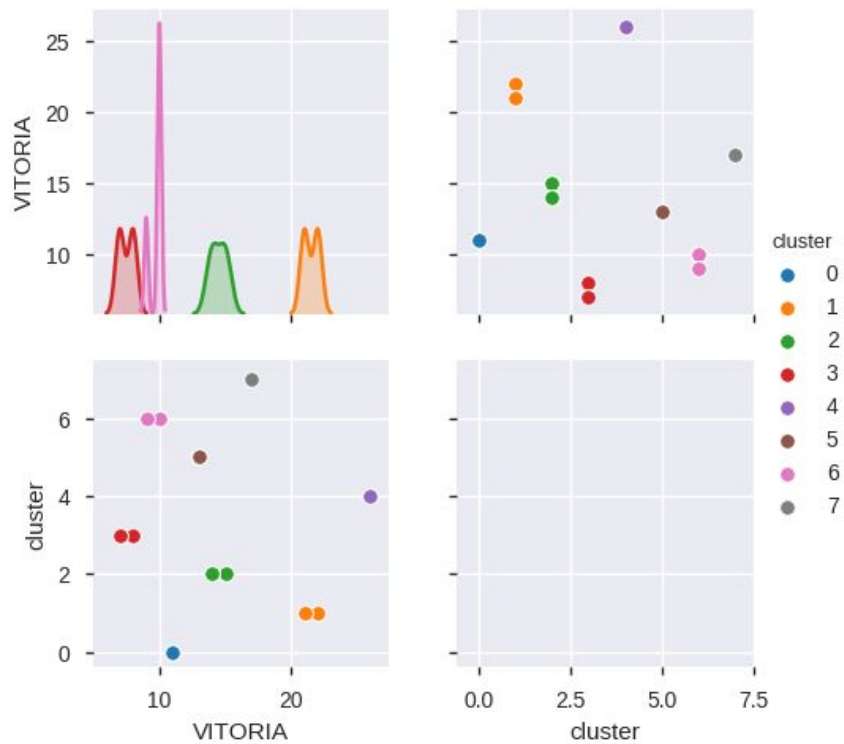
	NOME	VITORIA
3	Barcelona	82
18	Real Madrid	72
2	Ath Madrid	68
19	Sevilla	55
22	Valencia	50

# Fase de preparação

## Clusterizando para tabela SUPERTIMES

```
1 dfCluster = grupo[['ID', 'VITORIA']]
2 X = np.array(dfCluster)
3 from sklearn.cluster import KMeans
4 kmeans = KMeans( random_state=0)
5 kmeans.fit(X)
6 dfCluster['cluster'] = kmeans.labels_
7 sb.pairplot(dfCluster, hue='cluster')
8 grupo['cluster'] = kmeans.labels_
9
10 #SuperTimes Manual
11 #dsTratado['GH'] = dsTratado.apply(lambda row: GetGrandeza(row.HomeTeam), axis = 1)
12 #dsTratado['GA'] = dsTratado.apply(lambda row: GetGrandeza(row.AwayTeam), axis = 1)
13
14 #SuperTimes Cluster
15 dsTratado['GH'] = dsTratado.apply(lambda row: GetGrandezaCluster(row.HomeTeam), axis = 1)
16 dsTratado['GA'] = dsTratado.apply(lambda row: GetGrandezaCluster(row.AwayTeam), axis = 1)
17
```

# Fase de preparação



# Fase de preparação

## Escolha do Naive Bayes:

```
[55] 7 print('previsoes entrando com dados no segundo tempo')
8
9 print("\nAcurácia Gaussian primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoesGaussianNB))
10 print("Recall Gaussian primeiro tempo:",metrics.precision_score(y_test_1t, previsoesGaussianNB,average='macro'))
11 print("Precisão Gaussian primeiro tempo:",metrics.precision_score(y_test_1t, previsoesGaussianNB,average='macro'))
12
13 print("Acurácia SVM primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoesSVM))
14 print("Recall SVM primeiro tempo:",metrics.precision_score(y_test_1t, previsoesSVM,average='macro'))
15 print("Precisão SVM primeiro tempo:",metrics.precision_score(y_test_1t, previsoesSVM,average='macro'))
16
17 print("Acurácia KNeighborsClassifier primeiro tempo:",metrics.accuracy_score(y_test_1t, previsoesKNeighborsClassifier))
18 print("Recall KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_1t, previsoesKNeighborsClassifier,average='macro'))
19 print("Precisão KNeighborsClassifier primeiro tempo:",metrics.precision_score(y_test_1t, previsoesKNeighborsClassifier,average='macro'))
20
```



previsoes entrando com dados no segundo tempo

```
Acurácia Gaussian primeiro tempo: 0.49206349206349204
Recall Gaussian primeiro tempo: 0.6544578853046595
Precisão Gaussian primeiro tempo: 0.6544578853046595
Acurácia SVM primeiro tempo: 0.5079365079365079
Recall SVM primeiro tempo: 0.49041666666666667
Precisão SVM primeiro tempo: 0.34375
Acurácia KNeighborsClassifier primeiro tempo: 0.42063492063492064
Recall KNeighborsClassifier primeiro tempo: 0.3926515151515151
Precisão KNeighborsClassifier primeiro tempo: 0.3599163679808841
```

# Fase de processamento

Para a fase de processamento o conjunto de dados foi dividido em 70% para treinamento e 30% para testes, conforme mostra figura abaixo, separando em dados para treino com informações do segundo tempo e outro para início de partida baseado em histórico.



[illegible]



# Fase de armazenamento

Os dados foram obtidos através de datasets em formato CSV no site datahub.io (que é um site parceiro oficial de sites governamentais como data.gov.uk e provê também datasets gerais para muitos outros fins, como saúde, educação e também esportes, como futebol).

Os arquivos estão no formato de CSV e possuem 64 variáveis e 380 registros cada um.

Para a utilização no projeto, o escopo das variáveis, no tratamento inicial foi reduzido à 8 features, como mostra figura :

```
[73] 1 dsTratado = dsTodos[['HomeTeam', 'AwayTeam', 'FTHG', 'FTAG', 'HTHG', 'HTAG', 'HS', 'HTR', 'AS', 'FTR']].copy()  
    2 dsTratado
```

	HomeTeam	AwayTeam	FTHG	FTAG	HTHG	HTAG	HS	HTR	AS	FTR
0	Betis	Levante	0	3	0	1	22	A	6	A
1	Girona	Valladolid	0	0	0	0	13	D	2	D
2	Barcelona	Alaves	3	0	0	0	25	D	3	H
3	Celta	Espanol	1	1	0	1	12	A	14	D
4	Villarreal	Sociedad	1	2	1	1	16	D	8	A
...	...	...	...	...	...	...	...	...	...	...
375	Levante	Ath Madrid	2	2	2	0	17	H	17	D
376	Sevilla	Ath Bilbao	2	0	1	0	9	H	9	H
377	Valladolid	Valencia	0	2	0	1	19	A	9	A
378	Eibar	Barcelona	2	2	2	2	15	D	6	D
379	Real Madrid	Betis	0	2	0	0	9	D	9	A

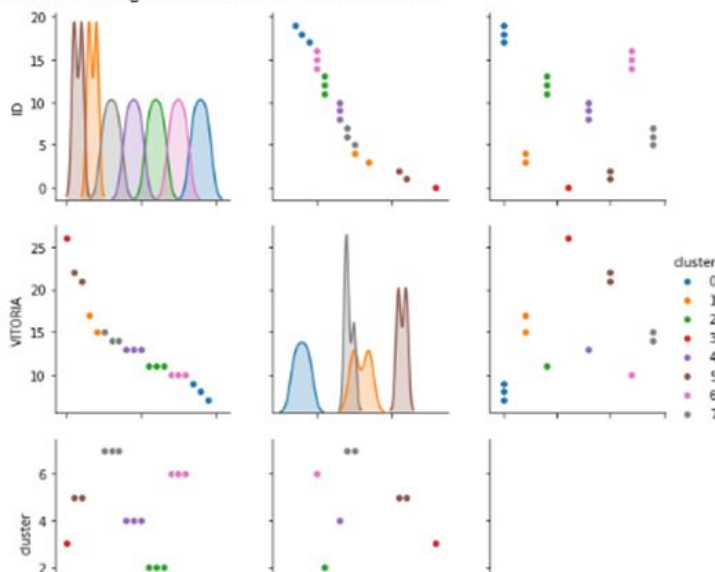
380 rows x 10 columns

# Fase de visualização

Foram utilizadas as bibliotecas matplotlib(que é uma biblioteca para a visualização de dados em Python com diversos tipos de gráficos, como em barra, em linha, em pizza, histogramas entre outras opções) e seaborn(que é uma biblioteca que atua em cima do matplotlib e ajuda a melhorar o visual dos gráficos, dando uma aparência mais bem acabada) para a geração de gráficos

```
dfCluster['cluster'] = kmeans.labels_  
sb.pairplot(dfCluster, hue='cluster')
```

```
↳ /usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:217: RuntimeWarning: Degrees of freedom  
  keepdims=keepdims)  
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:209: RuntimeWarning: invalid value encou  
  ret = ret.dtype.type(ret / rcount)  
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487: RuntimeWarning: invalid v  
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)  
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: RuntimeWarning: inval  
  FAC1 = 2*(np.pi*bw/RANGE)**2  
<seaborn.axisgrid.PairGrid at 0x7f30a8c797b8>
```

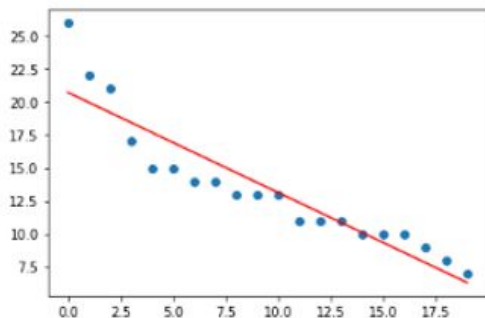


# Fase de visualização dos dados

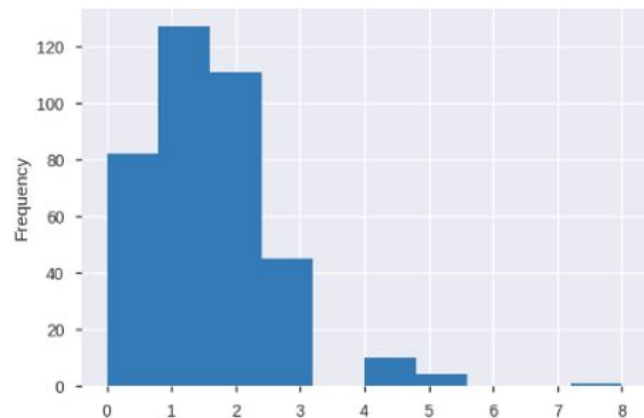
```
%matplotlib inline
# passando os valores de x e y como Dataframes
dfRegressaoPlot = grupo
X = dfRegressaoPlot[['ID']]
Y = dfRegressaoPlot[['VITORIA']]
# criando e treinando o modelo
model = LinearRegression()
model.fit(X, Y)
Y_pred = model.predict(X)
pl.scatter(X, Y)

pl.plot(X, Y_pred, color='red')
pl.show()
```

Plotagem das diferenças de performance dentre os times

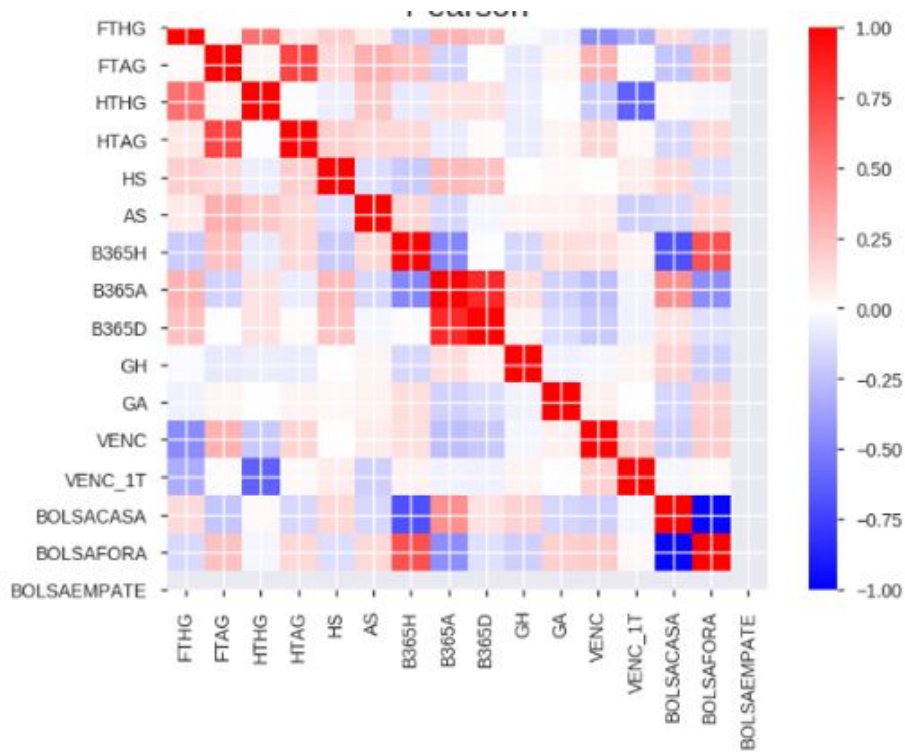


# Fase de Visualização -Profiling



Value Count Frequency (%)

1	127	33.4%
2	111	29.2%
0	82	21.6%
3	45	11.8%
4	10	2.6%
5	4	1.1%
8	1	0.3%



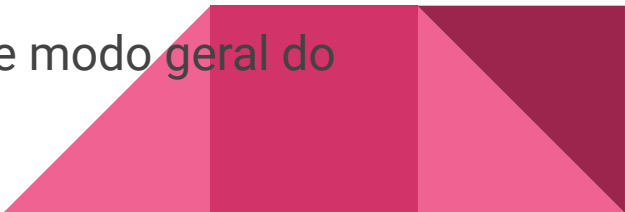
# Fase de Visualização

## • Testando

```
[69] 1  modelo = modeloGaussianNB
      2
      3  timeHome='Barcelona'
      4  timeAway='Girona'
      5
      6  chutesAGolHome=5
      7  chutesAGolAway=5
      8
      9  golsHome=0
     10  golsAway=0
     11
     12  jogoAtual = [[golsHome,\
     13                golsAway,\
     14                chutesAGolHome,\
     15                chutesAGolAway,\
     16                GetGrandezaCluster(timeHome),\
     17                GetGrandezaCluster(timeAway)]]
     18  previsoes = modelo.predict(jogoAtual)
     19
     20  print('A previsão é',DesConverter(previsoes))
```

☞ A previsão é Home

# Conclusões

- Um dos principais pontos positivos encontrados no trabalho foi o teste exaustivo para encontrar a melhor forma para se obter uma boa precisão, visto que teve, ao longo do processo, mudanças de mindset e até mesmo mudanças de algoritmos.
  - A originalidade do trabalho também foi um ponto positivo, tanto com relação ao contexto do negócio, onde foi constatado que não existe nenhum produto parecido ainda, e também com relação à busca contínua por melhorias, onde foram criadas novas features como solução que impulsionaram muito a qualidade do algoritmo.
  - O principal resultado negativo foi a baixa acurácia, de modo geral do algoritmo.
- 

# Conclusões

- A maior contribuição é mostrar aos envolvidos que é possível realizar previsões de resultados de partida de futebol utilizando técnicas de machine learning, com uma qualidade aceitável aos que é conhecido no meio como "ODD de valor", que precisa ser maior que 50%

