Regressão Linear Múltipla

Nesse arquivo iremos analisar a base de dados de "stats" de vários players de CS.

Com base nas informações, construiremos um modelo de regressão linear múltipla que será capaz de prever a quantidade de "kills" com base na na quantidade "HS" e de "Damage".

In [29]:

```
# importando as bibliotecas
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Vamos realizar uma análise exploratória dos dados inicialmente para sabermos algumas informações descritivas dos dados que iremos trabalhar.

In [30]:

```
data= pd.read_csv("C:/Users/Vinicius/Desktop/Dados/tb_lobby_stats_player.csv")
display(data)
data.info()
data.describe()
```

	idLobbyGame	idPlayer	idRoom	qtKill	qtAssist	qtDeath	qtHs	qtBombeDefuse	qtBombePlant	qtTk	...	qtFlashAssist	qtHitHeadsl
0	1	1	1	5	1	16	2	0	0	0.0	...	0.0	
1	2	1	2	24	3	18	6	0	4	0.0	...	0.0	
2	3	2	3	6	4	23	2	0	1	0.0	...	0.0	
3	3	391	27508	10	5	20	4	1	0	0.0	...	0.0	
4	4	2	4	8	4	26	6	0	2	0.0	...	2.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
184147	172907	2716	178496	21	3	13	5	1	1	0.0	...	0.0	
184148	172908	2716	178497	15	1	22	5	0	1	0.0	...	0.0	
184149	172909	2716	178498	9	6	23	2	0	3	0.0	...	0.0	
184150	172910	2716	178499	15	5	20	6	0	2	0.0	...	1.0	
184151	172911	2716	178500	12	6	11	4	0	1	0.0	...	0.0	

184152 rows × 38 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 184152 entries, 0 to 184151
Data columns (total 38 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   idLobbyGame     184152 non-null  int64  
 1   idPlayer        184152 non-null  int64  
 2   idRoom          184152 non-null  int64  
 3   qtKill          184152 non-null  int64  
 4   qtAssist        184152 non-null  int64  
 5   qtDeath         184152 non-null  int64  
 6   qtHs            184152 non-null  int64  
 7   qtBombeDefuse   184152 non-null  int64  
 8   qtBombePlant    184152 non-null  int64  
 9   qtTk            184032 non-null  float64
 10  qtTkAssist      184032 non-null  float64
 11  qt1Kill         184152 non-null  int64  
 12  qt2Kill         184152 non-null  int64  
 13  qt3Kill         184152 non-null  int64  
 14  qt4Kill         184152 non-null  int64  
 15  qt5Kill         184152 non-null  int64  
 16  qtPlusKill      184152 non-null  int64  
 17  qtFirstKill     184152 non-null  int64  
 18  vlDamage        184152 non-null  int64  
 19  qtHits          184032 non-null  float64
 20  qtShots         184152 non-null  int64  
 21  qtLastAlive     184032 non-null  float64
 22  qtClutchWon     184152 non-null  int64  
 23  qtRoundsPlayed  184152 non-null  int64  
 24  descMapName     184152 non-null  object 
 25  vlLevel         184152 non-null  int64  
 26  qtSurvived      183447 non-null  float64
 27  qtTrade         183447 non-null  float64
 28  qtFlashAssist   183447 non-null  float64
 29  qtHitHeadshot   183447 non-null  float64
 30  qtHitChest      183447 non-null  float64
 31  qtHitStomach    183447 non-null  float64
 32  qtHitLeftAtm    183447 non-null  float64
 33  qtHitRightArm   183447 non-null  float64
 34  qtHitLeftLeg    183447 non-null  float64
 35  qtHitRightLeg   183447 non-null  float64
 36  flWinner        184152 non-null  int64  
 37  dtCreatedAt     184152 non-null  object 
dtypes: float64(14), int64(22), object(2)
memory usage: 53.4+ MB
```

Out[30]:

	idLobbyGame	idPlayer	idRoom	qtKill	qtAssist	qtDeath	qtHs	qtBombeDefuse	qtBombePla
count	184152.000000	184152.000000	184152.000000	184152.000000	184152.000000	184152.000000	184152.000000	184152.000000	184152.0000
mean	84720.886854	1361.148622	88343.226248	19.113531	3.756033	18.792459	7.640123	0.316054	1.3213
std	49931.048091	784.157397	51564.451107	7.481041	2.205265	5.211614	4.042324	0.575803	1.3366
min	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	41399.750000	676.000000	43742.750000	14.000000	2.000000	16.000000	5.000000	0.000000	0.0000
50%	83838.500000	1388.000000	87877.500000	19.000000	4.000000	19.000000	7.000000	0.000000	1.0000
75%	127911.250000	2061.000000	133000.250000	24.000000	5.000000	22.000000	10.000000	1.000000	2.0000
max	172911.000000	2716.000000	178500.000000	85.000000	24.000000	65.000000	41.000000	5.000000	12.0000

8 rows × 36 columns

Nos gráficos abaixo, podemos observar os coeficientes de correlação de Pearson, onde é possível identificar as variáveis que mais se correlacionam positivamente com a nossa variável dependente, no caso a quantidade "Kill". Observamos que as variáveis "vlDamage" e "qtHs" possuem os índices de correlação mais altos, e por isso vamos utiliza-lás para treinar o nosso modelo.

In [31]:

```
datafilter = data[['qtKill', 'qtAssist','qtDeath', 'qtHs', 'vlDamage']]
corr = datafilter.corr()
corr.style.background_gradient(cmap='coolwarm')
```

Out[31]:

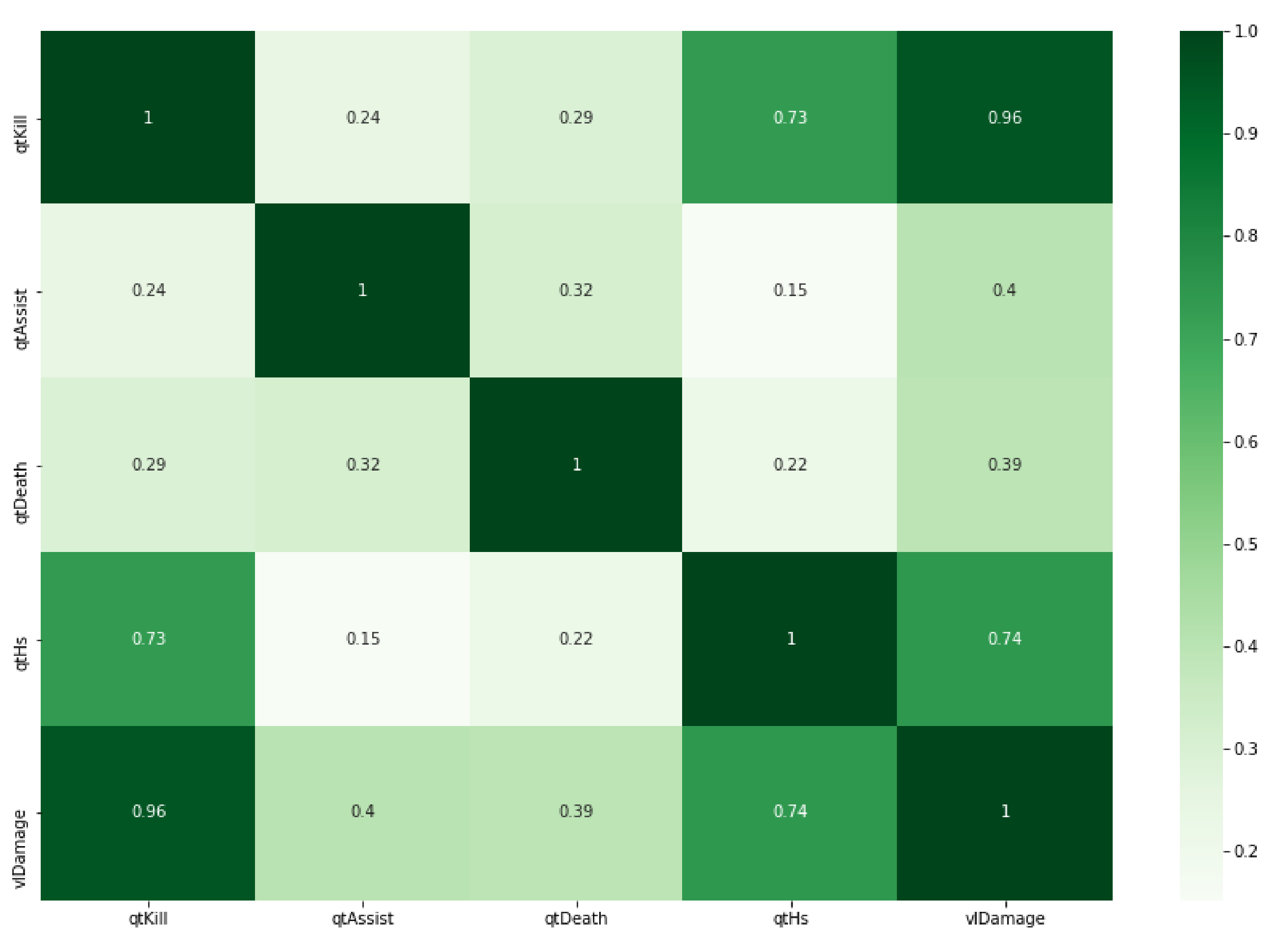
	qtKill	qtAssist	qtDeath	qtHs	vlDamage
qtKill	1.000000	0.243858	0.293997	0.734080	0.955880
qtAssist	0.243858	1.000000	0.316440	0.152456	0.403197
qtDeath	0.293997	0.316440	1.000000	0.217032	0.394373
qtHs	0.734080	0.152456	0.217032	1.000000	0.743856
vlDamage	0.955880	0.403197	0.394373	0.743856	1.000000

In [32]:

```
f, ax = plt.subplots(figsize=(15, 10))
sns.heatmap(datafilter.corr(), cmap="Greens",annot=True)
```

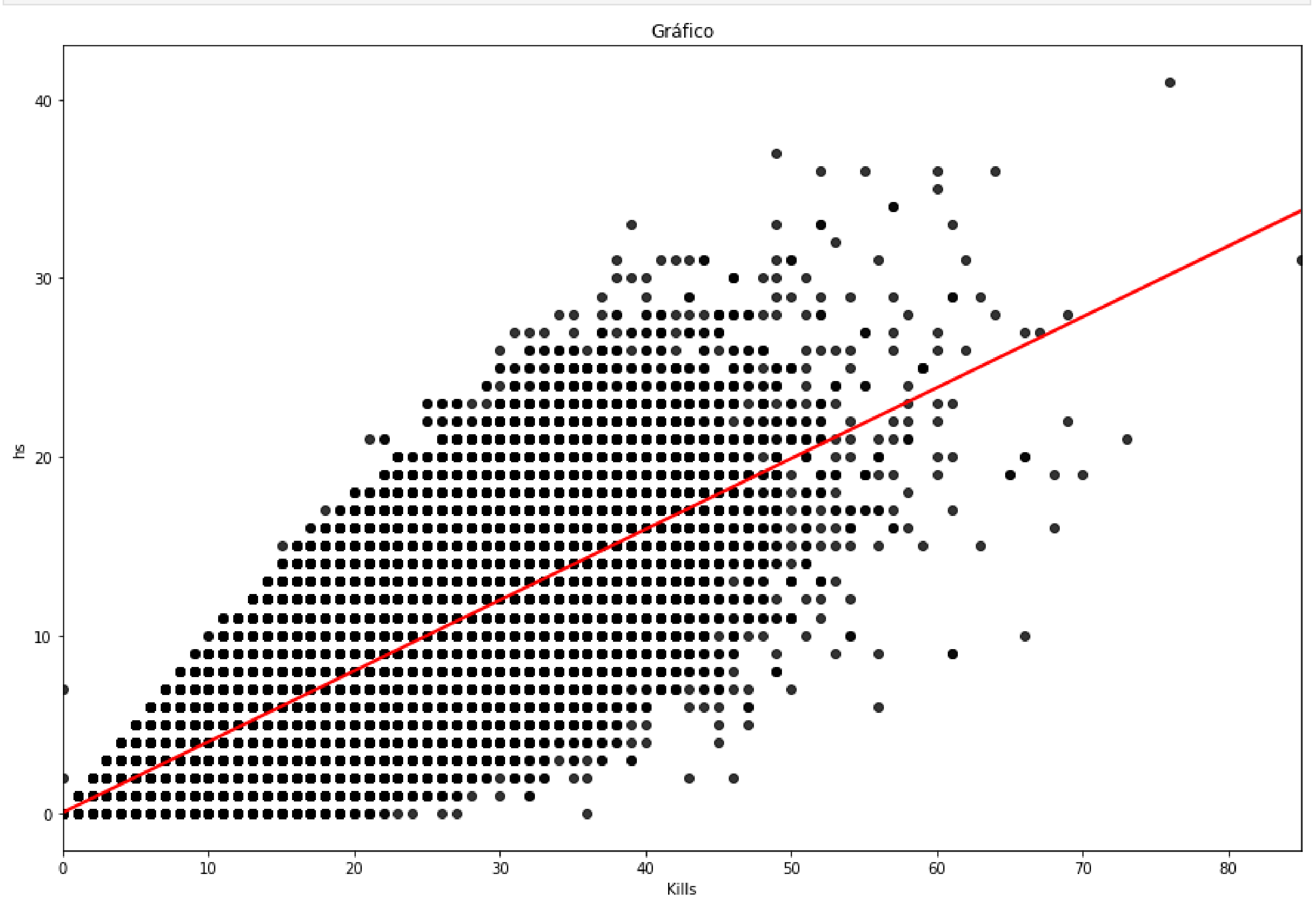
Out[32]:

```
<AxesSubplot:>
```



In [52]:

```
f, ax = plt.subplots(figsize=(15, 10))
sns.regplot(x=data['qtKill'], y=data['qtHs'], data=data, scatter_kws={"color": "black"}, line_kws={"color": "re
plt.title("Gráfico Kill - HS")
plt.xlabel('Kills' )
plt.ylabel("HS")
plt.show()
```

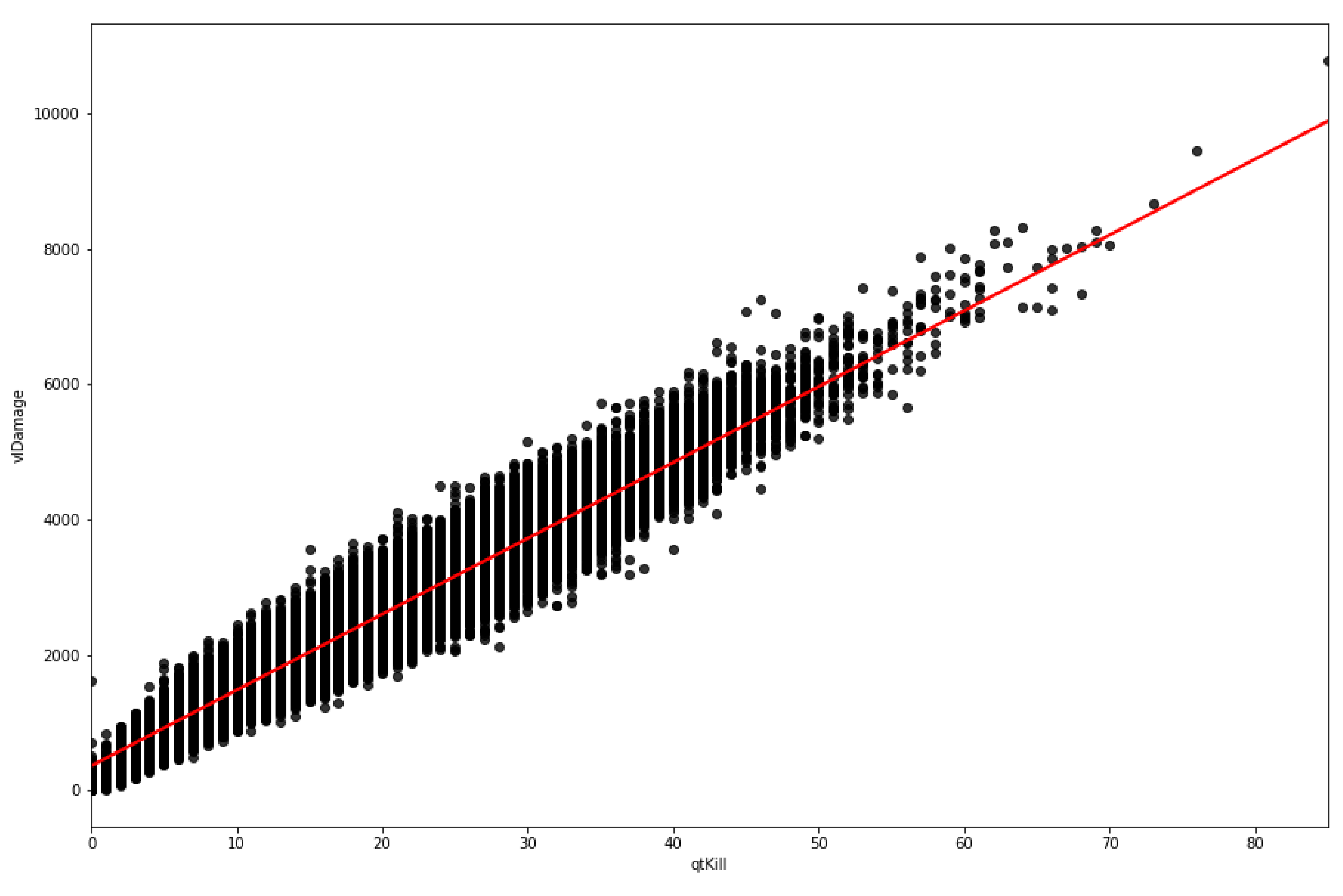


In [51]:

```
f, ax = plt.subplots(figsize=(15, 10))
sns.regplot(x=data['qtKill'], y=data['vlDamage'], data=data, scatter_kws={"color": "black"}, line_kws={"color":
plt.title("Gráfico Kill - Damage")
plt.xlabel('Kills' )
plt.ylabel("HS")
```

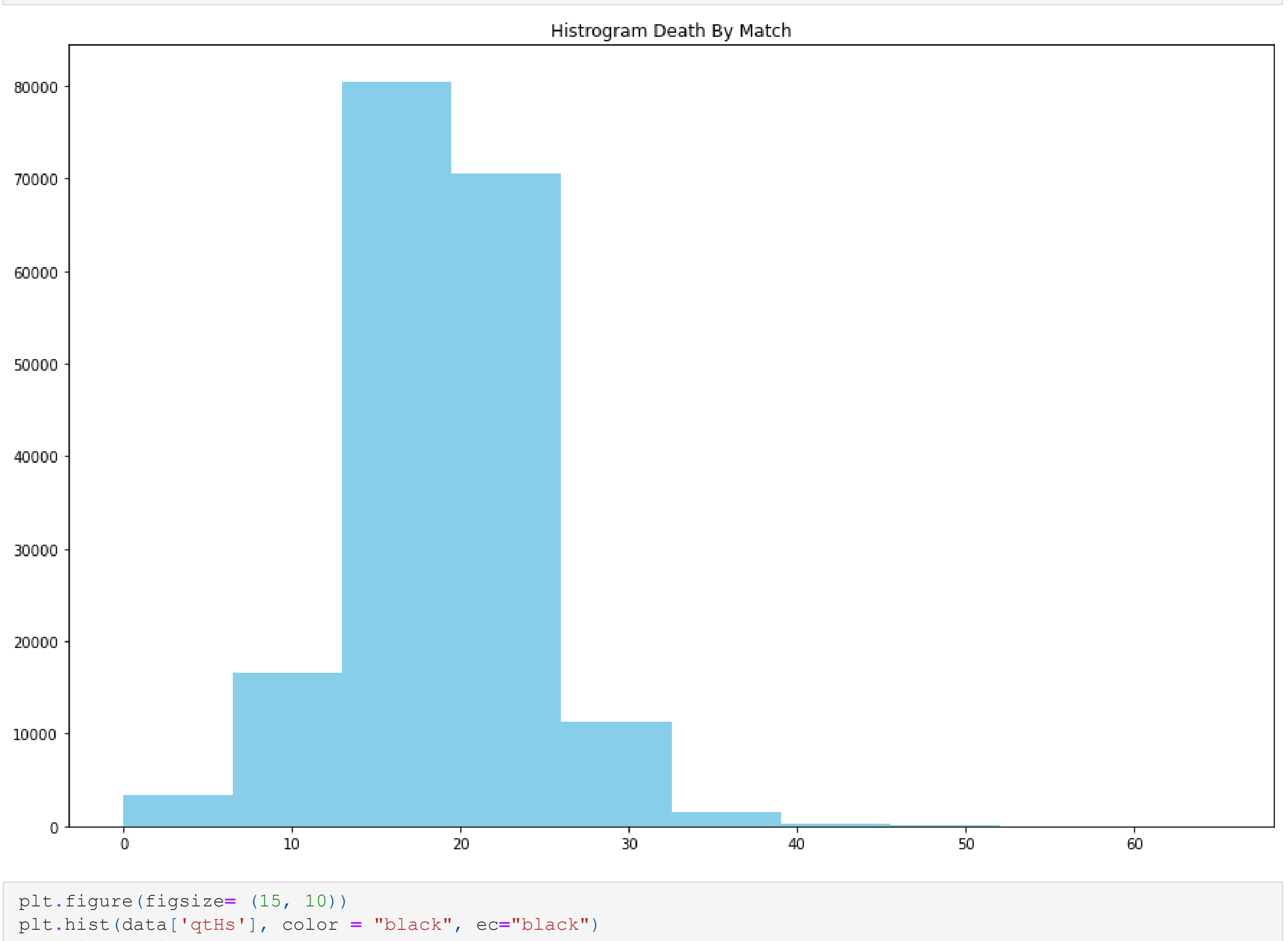
Out[51]:

```
<AxesSubplot:xlabel='qtKill', ylabel='vlDamage'>
```



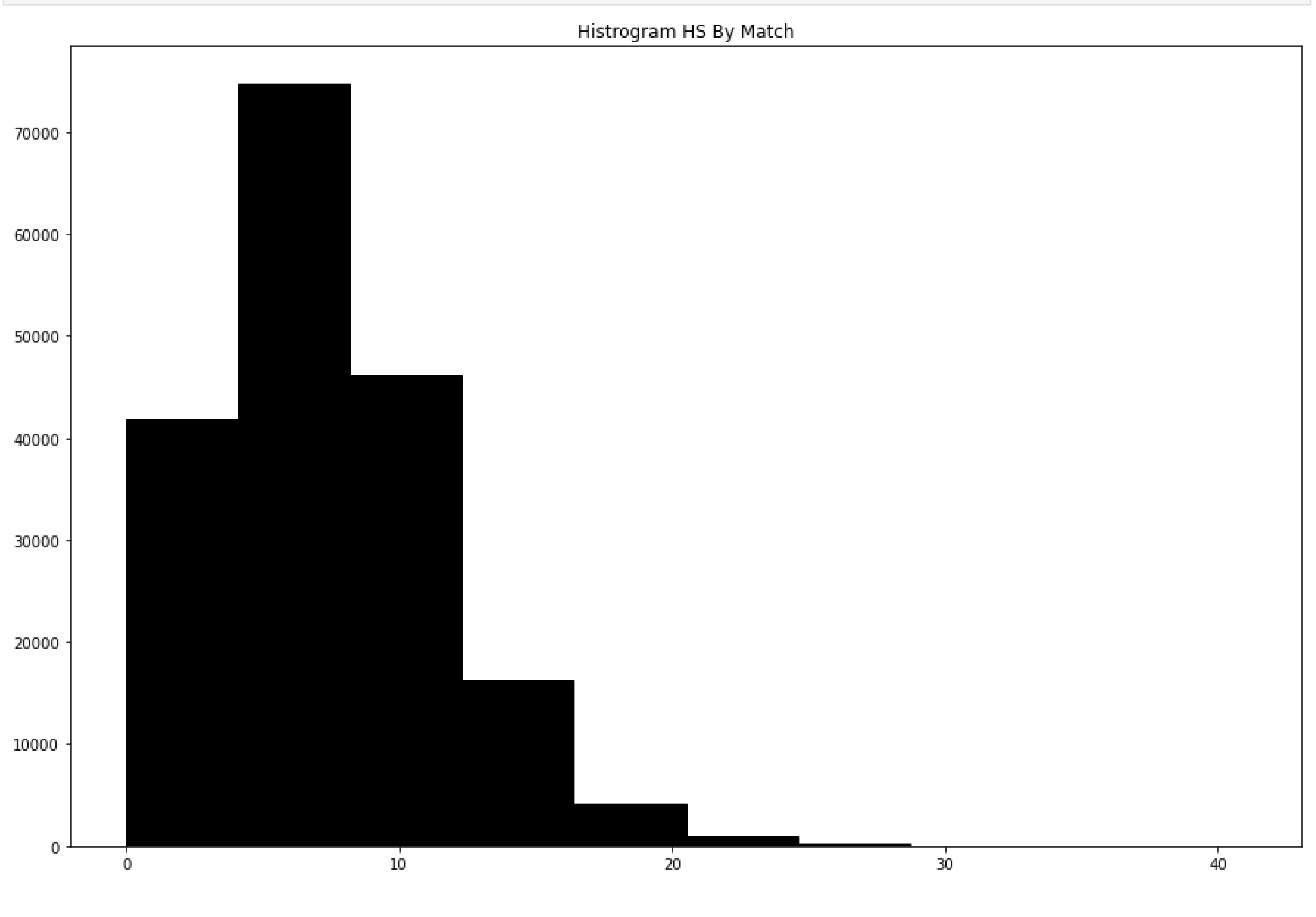
In [55]:

```
plt.figure(figsize= (15, 10))
plt.hist(data['qtDeath'], color = "skyblue")
plt.title("Histrogram Death By Match")
plt.show()
```



In [36]:

```
plt.figure(figsize= (15, 10))
plt.hist(data['qtHs'], color = "black", ec="black")
plt.title("Histrogram HS By Match")
plt.show()
```



Agora que sabamos as variáveis que melhor se correlacionam com a variável dependente, vamos separar a nossa base de dados em 80% para treino do modelo e 20% para realizarmos previsões e avaliarmos a precisão do modelo

In [37]:

```
xcs = data[['qtHs','vlDamage']]
ycs = data['qtKill']

x_traincs, x_testcs, y_traincs, y_testcs = train_test_split(xcs,ycs, test_size=0.2)
```

In [53]:

```
linear = LinearRegression().fit(x_traincs, y_traincs)
predictioncs = linear.predict(x_testcs)
```

Com o modelo treinado, podemos ver o coeficiente de determinação (ou R^2) para medirmos o grau de precisão que o modelo apresentou na base de dados de treino.

In [68]:

```
linear.score(x_traincs, y_traincs)
```

Out[68]:

```
0.914860113720179
```

Também podemos montar um dataframe com os valores verdadeiros(esperados) e os valores que foram preditos pelo modelo, e assim termos uma noção do grau de diferença que o modelo está apresentando com os números reais.

In [62]:

```
compcs = pd.DataFrame(y_testcs)
compcs.columns = ['Y_test']
compcs['Prediction'] = predictioncs
compcs['Diference Between Y_test and Prediction'] = compcs['Prediction'] - compcs['Y_test']
display(compcs)
```

	Y_test	Prediction	Diference Between Y_test and Prediction
46109	20	20.275408	0.275408
10223	19	19.769980	0.769980
100744	18	19.134949	1.134949
41522	5	6.894332	1.894332
94265	11	12.144446	1.144446
...	...	...	...
172323	24	22.486622	-1.513378
45905	18	16.817590	-1.182410
46968	39	42.792321	3.792321
79370	14	13.575388	-0.424612
105525	27	28.726963	1.726963

36831 rows × 3 columns

In [63]:

```
compcs.describe()
```

Out[63]:

	Y_test	Prediction	Diference Between Y_test and Prediction
count	36831.000000	36831.000000	36831.000000
mean	19.082946	19.081905	-0.001042
std	7.438654	7.133567	2.168296
min	0.000000	-1.172412	-11.713751
25%	14.000000	14.394757	-1.407173
50%	19.000000	18.690909	-0.008148
75%	24.000000	23.267390	1.439554
max	67.000000	64.156286	12.118511

In [42]:

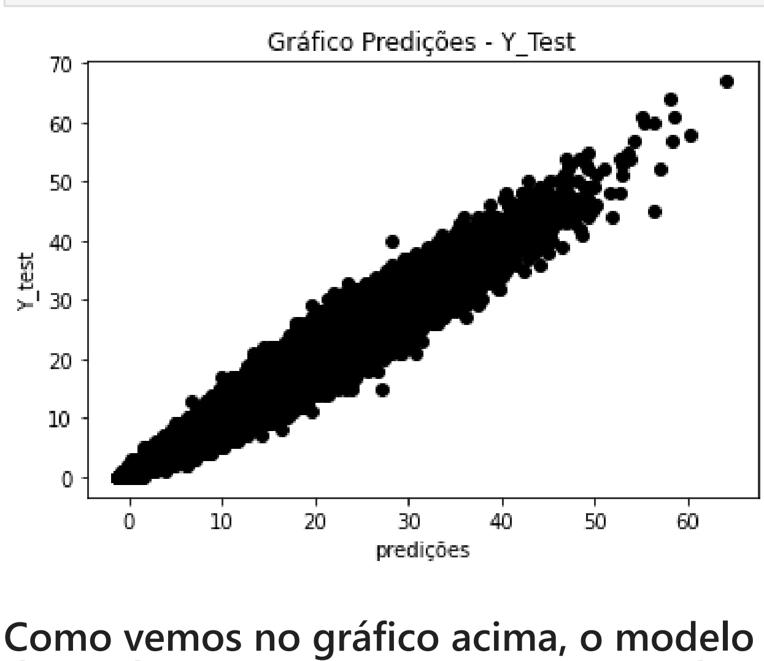
```
linear.score(x_testcs, y_testcs)
```

Out[42]:

```
0.9150333432680284
```

In [67]:

```
fig, ax = plt.subplots()
ax.scatter(compcs['Prediction'], compcs['Y_test'], color = "black")
plt.title("Gráfico Predições - Y_Test")
plt.xlabel("predições" )
plt.ylabel("Y_test")
plt.show()
```



Como vemos no gráfico acima, o modelo está conseguindo prever a variável dependente(resposta) com uma grande precisão, aproximadamnte 91,5%. Isso significa que o modelo linear explica 91,5% da variância da variável dependente a partir do regressores (variáveis independentes) incluídas no modelo linear.