Homework 2: Trees and Calibration

Instructions

Please push the .ipynb, .py, and .pdf to Github Classroom prior to the deadline. Please include your UNI as well.

Make sure to use the dataset that we provide in CourseWorks/Classroom. DO NOT download it from the link provided (It may be different).

Due Date : 03/02 (2nd March), 11:59 PM EST

Name: Vipul H Harihar

UNI: vhh2105

The Dataset

Credit (Link | License)

The goal is to predict wins based on in match performace of multiple players. Please use this dataset and this task for all parts of the assignment.

Features

idLobbyGame - Categorical (The Lobby ID for the game)

idPlayer - Categorical (The ID of the player)

idRooom - Categorical (The ID of the room)

qtKill - Numerical (Number of kills)

qtAssist - Numerical (Number of Assists)

qtDeath - Numerical (Number of Deaths)

qtHs - Numerical (Number of kills by head shot)

qtBombeDefuse - Numerical (Number of Bombs Defuses)

qtBombePlant - Numerical (Number of Bomb plants)

qtTk - Numerical (Number of Team kills)

qtTkAssist - Numerical Number of team kills assists)

qt1Kill - Numerical (Number of rounds with one kill)

qt2Kill - Numerical (Number of rounds with two kill)

qt3Kill - Numerical (Number of rounds with three kill)

qt4Kill - Numerical (Number of rounds with four kill)

qt5Kill - Numerical (Number of rounds with five kill)

qtPlusKill - Numerical (Number of rounds with more than one kill)

qtFirstKill - Numerical (Number of rounds with first kill)

vlDamage - Numerical (Total match Damage)

qtHits - Numerical (Total match hits)

qtShots - Numerical (Total match shots)

qtLastAlive - Numerical (Number of rounds being last alive)

qtClutchWon - Numerical (Number of total clutchs wons)

qtRoundsPlayed - Numerical (Number of total Rounds Played)

descMapName - Categorical (Map Name - de_mirage, de_inferno, de_dust2, de_vertigo, de_overpass, de_nuke, de_train, de_ancient)

vlLevel - Numerical (GC Level)

qtSurvived - Numerical (Number of rounds survived)

qtTrade - Numerical (Number of trade kills)

qtFlashAssist - Numerical (Number of flashbang assists)

qtHitHeadshot - Numerical (Number of times the player hit headshot

qtHitChest - Numerical (Number of times the player hit chest)

qtHitStomach - Numerical (Number of times the player hit stomach)

qtHitLeftAtm - Numerical (Number of times the player hit left arm)

qtHitRightArm - Numerical (Number of times the player hit right arm)

qtHitLeftLeg - Numerical (Number of times the player hit left leg)

qtHitRightLeg - Numerical (Number of times the player hit right leg)

flWinner - Winner Flag (**Target Variable**).

dtCreatedAt - Date at which this current row was added. (Date)

Question 1: Decision Trees

1.1: Load the provided dataset

Answer

I will load the data here, though I might have used zip file package but on Google colab I used the csv file(unzipped) directly

In [116…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.decomposition import PCA
```

```
from sklearn.manifold import TSNE
import zipfile
sns.set_style('darkgrid')

df  = pd.read_csv('tb_lobby_stats_player.csv')
df.head(5)
```

Out[116…

	idLobbyGame	idPlayer	idRoom	qtKill	qtAssist	qtDeath	qtHs	qtBombeDefuse	qtBombePlant	qtTk	...	qtFlashAssist	qtHitHeac
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	

5 rows × 38 columns

1.2: Plot % of missing values in each column. Would you consider dropping any columns? Assuming we want to train a decision tree, would you consider imputing the missing values? If not, why? (Remove the columns that you consider dropping - you must remove the dtCreatedAt column)

In [117…

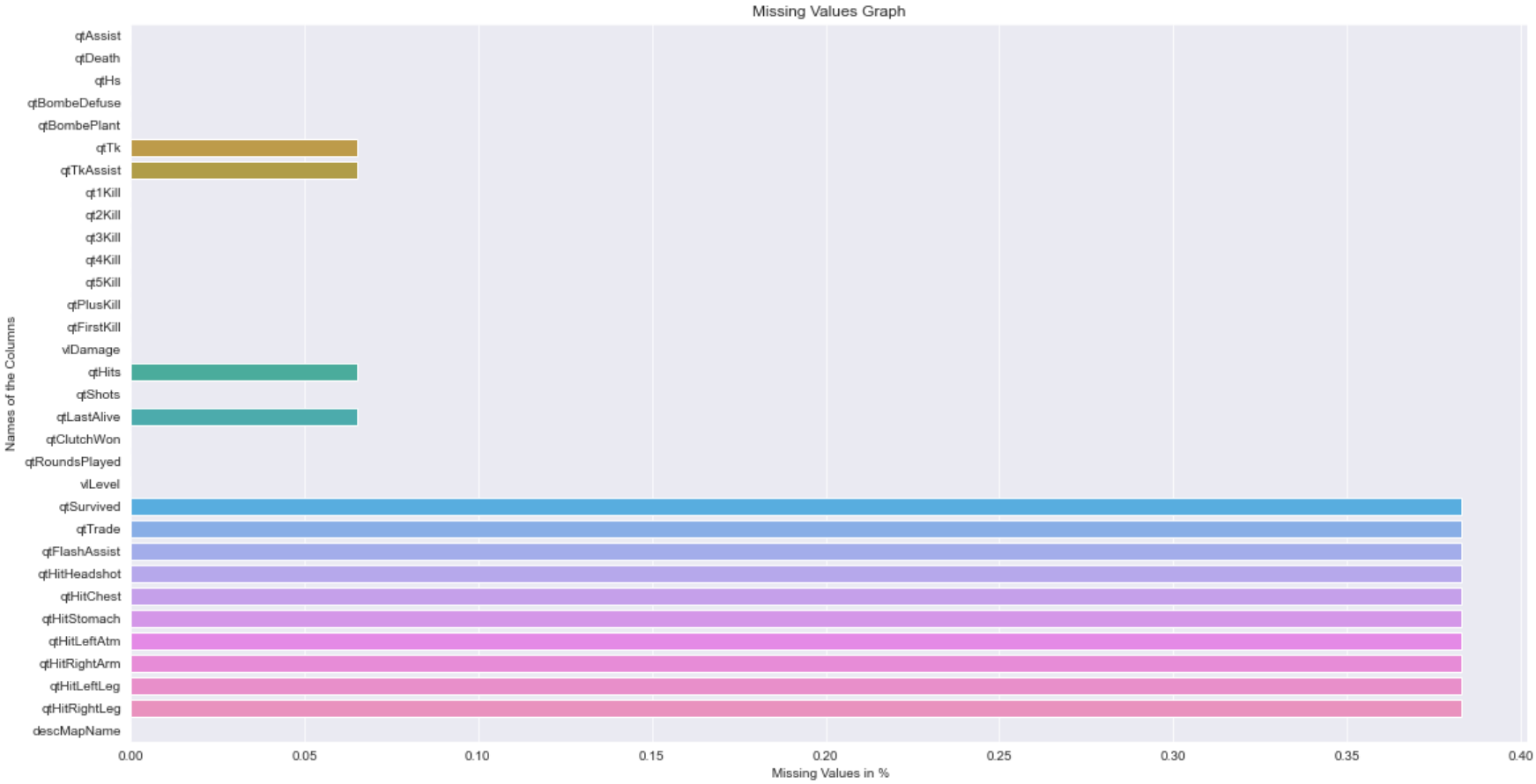
```
import missingno as msno
```

In [118…

```
x=df.columns.array
y=df.isna().sum().array

df = df.replace(' ?', np.nan)
numericals = ['qtAssist', 'qtDeath', 'qtHs', 'qtBombeDefuse', 'qtBombePlant', 'qtTk', 'qtTkAssist','qt1Kill',
              'qt2Kill', 'qt3Kill', 'qt4Kill', 'qt5Kill', 'qtPlusKill', 'qtFirstKill', 'vlDamage', 'qtHits',
              'qtShots', 'qtLastAlive', 'qtClutchWon', 'qtRoundsPlayed','vlLevel', 'qtSurvived', 'qtTrade',
              'qtFlashAssist', 'qtHitHeadshot', 'qtHitChest','qtHitStomach', 'qtHitLeftAtm', 'qtHitRightArm',
              'qtHitLeftLeg', 'qtHitRightLeg']
categoricals = ['descMapName']
```

```
missing_numericals = (df[numericals].isnull().sum()/len(df))*100
missing_categories = (df[categoricals].isna().sum()/len(df))*100
missing_values = missing_numericals.append(missing_categories)

plt.figure(figsize = (19, 10))
sns.barplot(y = missing_values.index, x =  missing_values.values)
plt.ylabel('Names of the Columns')
plt.xlabel(' Missing Values in %')
plt.title('Missing Values Graph')
```

Out[118… `Text(0.5, 1.0, 'Missing Values Graph')`



Would you consider dropping any columns? Assuming we want to train a decision tree, would you consider imputing the missing values? If not, why? (Remove the columns that you consider dropping - you must remove the dtCreatedAt column)

a) Yes I would consider dropping the missing values as they do not contribute that much to the dataset.

b) I might impute the missing values with standard mean but for now for learning purposes I am just dropping them also we know decision trees uses Gini and Information Gain for decising importances of the features so imputing would not be that helpful in the cases of decision trees but it might be done according to problem statement. Right now, I don't think it might be needed for my learning as I did practiced it in last assignment.

c) I have dropped the dtCreatedAt column. So its done

1.3: Plot side-by-siide bars of class distribtuion for each category for the categorical feature and the target categories.

Answer

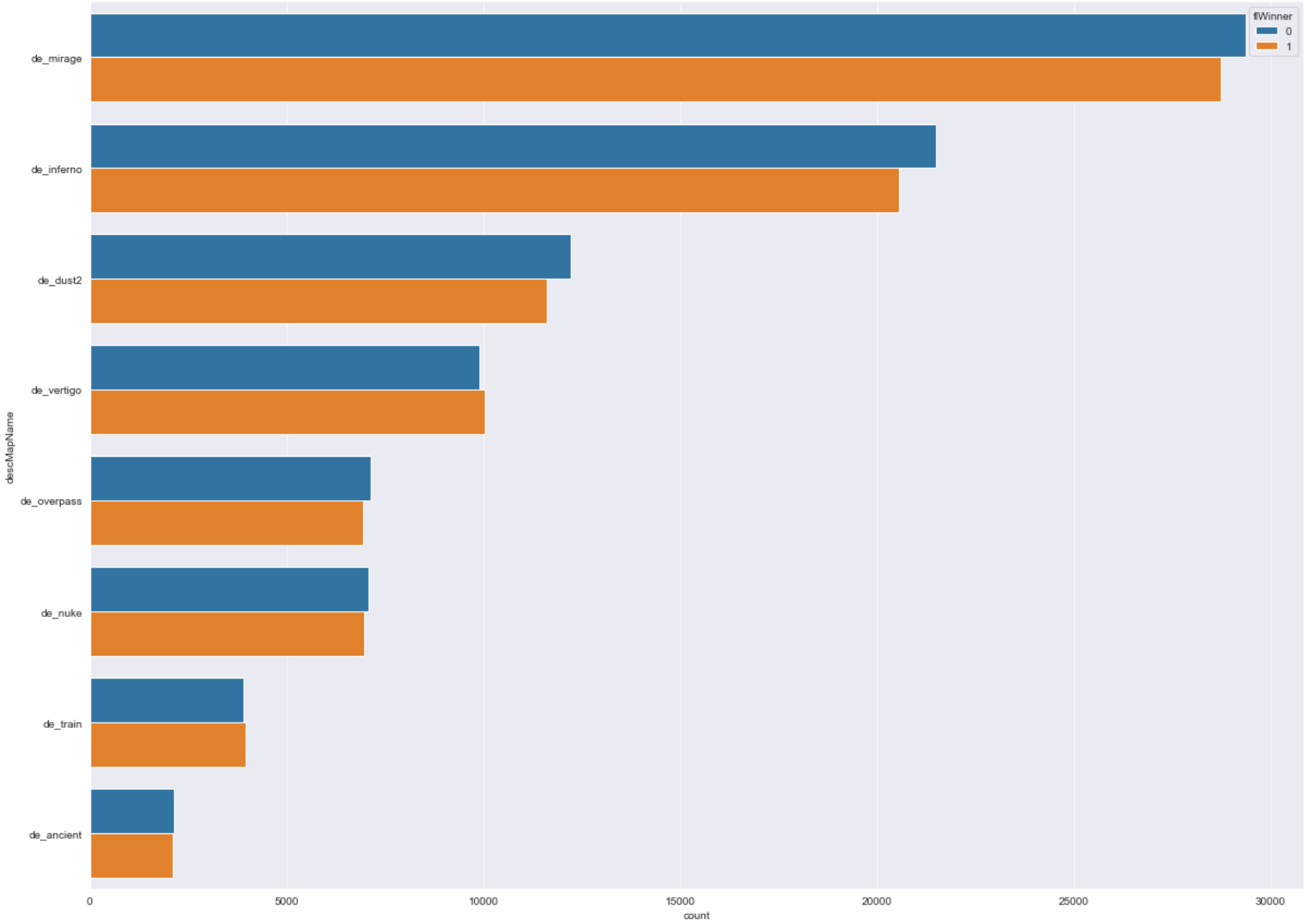
We can plot the graph as below:-using seaborn library for the display

In [119…

```
fig, axs = plt.subplots(1, 1, figsize = (20, 15))

sns.countplot(data = df, y = 'descMapName', hue = 'flWinner', order = df['descMapName'].value_counts().index)
```

Out[119…

```
<AxesSubplot:xlabel='count', ylabel='descMapName'>
```



1.4: Split the data into development and test datasets. Which splitting methodology did you choose and why?

ANSWER 1.4

Development-test split

● Typically we are aware that the dataset is split into development dataset and test dataset in the ratio of 4:1 (also called 80/20 split) or 3:1

● The purpose of test dataset is to evaluate the performance of the final optimal model

● Model evaluation is supposed to give a pulse on how the model would perform in the wild.

Splitting strategies:

○ Random splitting

○ Stratified splitting

○ Structured splitting

My Splitting Stratergy would be:- the default one Random splitting

● development/train-test split - 3:1

● The random splitting is best as the ratio of indices (classes) in development and test datasets equals that of the original dataset.

● We also know that it is generally employed when performing classification balanced datasets, likea swe saw the above plot , it seems to me a balanced dataset

● It serves my purpose!

In [120…

```
from sklearn.model_selection import train_test_split

df = df.dropna(axis = 0)

ohe_features = ['descMapName']
num_features = ['qtAssist', 'qtDeath', 'qtHs', 'qtBombeDefuse', 'qtBombePlant', 'qtTk', 'qtTkAssist', 'qt1Kill'
                'qt2Kill', 'qt3Kill', 'qt4Kill', 'qt5Kill', 'qtPlusKill', 'qtFirstKill', 'vlDamage', 'qtHits',
                'qtShots', 'qtLastAlive', 'qtClutchWon', 'qtRoundsPlayed','vlLevel', 'qtSurvived', 'qtTrade',
                'qtFlashAssist', 'qtHitHeadshot', 'qtHitChest', 'qtHitStomach', 'qtHitLeftAtm', 'qtHitRightArm'
                'qtHitLeftLeg', 'qtHitRightLeg']
mixed_df = df[ohe_features + num_features]
target = df['flWinner']

X_dev, X_test, y_dev, y_test = train_test_split(mixed_df, target, test_size = .2, random_state = 42)
```

```
print("Checking the Balance measure in the dataset:")
print(len(target)/2)
print(sum(target))
```

```
Checking the Balance measure in the dataset:
91723.5
90861
```

From the above measures we might say that it is not perfectly balanced but it is balanced (in some proportions) and we can go ahead with the random splitting as we have kept, thus the random splitting is a good option of splitting for our game dataset

1.5: Preprocess the data (Handle the Categorical Variable). Do we need to apply scaling? Briefly Justify

Preprocessing of Data

I need to handle the DescMapName column as this is a categorical feature and contributes to the dataset. I would like to perform One hot Encoding using the get_dummies method to preprocess the data.

In [121…

```
dummies=pd.get_dummies(X_dev['descMapName'])
X_dev=X_dev.join(dummies)
t_dummies=pd.get_dummies(X_test['descMapName'])
X_test=X_test.join(t_dummies)
```

In [122…

```
X_dev=X_dev.drop('descMapName',1)
X_test=X_test.drop('descMapName',1)
```

Do we need to apply scaling? Briefly Justify

I feel that applying scaling in our decision trees won't be a good approach as it doesn't require any normalization and neither any procedures related to data to be normalized in any sort thus I would have used Standard Scaler but it was not helping me in improving the accuracy of learning rate so I thought I won't apply it at all. Indstead the data used here in the dataset was observed was not as sensitive to the the variance in the data, thus Standardscaler operation to preprocess numerical features won't hurt but as it wasn't useful for me so I commented it out

1.6: Fit a Decision Tree on the development data until all leaves are pure. What is the performance of the tree on the development set and test set? Provide metrics you believe are relevant and briefly justify.

In [128…

```
from sklearn.pipeline import make_pipeline
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
clf = DecisionTreeClassifier(random_state = 42)
pipe = make_pipeline(clf)
pipe.fit(X_dev, y_dev)
print("Score (Development Data):", pipe.score(X_dev, y_dev))
print("Score (Test Data):", pipe.score(X_test, y_test))
```

```
Score (Development Data): 1.0
Score (Test Data): 0.7231398201144726
```

What is the performance of the tree on the development set and test set? Provide metrics you believe are relevant and briefly justify.

Answer

As we see above the true leaves are pure as the Development Data is scoring out to be 1.0 and score on the test data is 0.7231 which is quite justifiable as the data been inspected in the forst portion of the question shows the features importances which shows that we have few measures of features on which we can make the decision tree to learn but it was as expected for the decsion tree to act like that on test data

1.7: Visualize the trained tree until the max_depth 8

Answer

I will plot the decision tree as below for the max_depth=8

In [130…

```
#Code for plotting the decision tree upto level max_depth=8
fig = plt.figure(figsize=(15,15))
plot_tree(clf, filled = True, max_depth = 8, fontsize = 8, feature_names = X_train.columns)
```

Out[130…

```
[Text(0.48645513803680984, 0.95, 'qtSurvived <= 6.5\ngini = 0.5\nsamples = 146757\nvalue = [74114, 72643]'),
 Text(0.2420053680981595, 0.85, 'qtSurvived <= 4.5\ngini = 0.337\nsamples = 63655\nvalue = [49996, 13659]'),
 Text(0.11717791411042945, 0.75, 'qtSurvived <= 3.5\ngini = 0.163\nsamples = 34107\nvalue = [31044, 3063]'),
 Text(0.06311349693251533, 0.65, 'qtRoundsPlayed <= 26.5\ngini = 0.098\nsamples = 22867\nvalue = [21687, 118
0]'),
 Text(0.0343558282208589, 0.55, 'qtDeath <= 14.5\ngini = 0.072\nsamples = 20346\nvalue = [19588, 758]'),
 Text(0.015644171779141104, 0.45, 'qtSurvived <= 1.5\ngini = 0.195\nsamples = 2948\nvalue = [2625, 323]'),
 Text(0.006748466257668712, 0.35, 'qtDeath <= 5.5\ngini = 0.075\nsamples = 1860\nvalue = [1787, 73]'),
 Text(0.0036809815950920245, 0.25, 'qtHitChest <= 14.0\ngini = 0.133\nsamples = 756\nvalue = [702, 54]'),
 Text(0.00245398773006135, 0.15, 'qtSurvived <= 0.5\ngini = 0.126\nsamples = 753\nvalue = [702, 51]'),
 Text(0.001226993865030675, 0.05, '\n  (...)  \n'),
 Text(0.0036809815950920245, 0.05, '\n  (...)  \n'),
 Text(0.0049079754601227, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.0098159509202454, 0.25, 'qtAssist <= 4.5\ngini = 0.034\nsamples = 1104\nvalue = [1085, 19]'),
```

```
Text(0.007361963190184049, 0.15, 'qtHitRightLeg <= 4.5\ngini = 0.031\nsamples = 1094\nvalue = [1077, 17]'),
Text(0.006134969325153374, 0.05, '\n  (...)  \n'),
Text(0.008588957055214725, 0.05, '\n  (...)  \n'),
Text(0.012269938650306749, 0.15, 'qtClutchWon <= 0.5\ngini = 0.32\nsamples = 10\nvalue = [8, 2]'),
Text(0.011042944785276074, 0.05, '\n  (...)  \n'),
Text(0.013496932515337423, 0.05, '\n  (...)  \n'),
Text(0.024539877300613498, 0.35, 'qtRoundsPlayed <= 10.5\ngini = 0.354\nsamples = 1088\nvalue = [838, 250]'),
Text(0.0196319018404908, 0.25, 'qtShots <= 100.5\ngini = 0.444\nsamples = 456\nvalue = [304, 152]'),
Text(0.01717791411042945, 0.15, 'vlDamage <= 1092.0\ngini = 0.32\nsamples = 180\nvalue = [144, 36]'),
Text(0.015950920245398775, 0.05, '\n  (...)  \n'),
Text(0.018404907975460124, 0.05, '\n  (...)  \n'),
Text(0.022085889570552148, 0.15, 'vlDamage <= 709.5\ngini = 0.487\nsamples = 276\nvalue = [160, 116]'),
Text(0.020858895705521473, 0.05, '\n  (...)  \n'),
Text(0.023312883435582823, 0.05, '\n  (...)  \n'),
Text(0.029447852760736196, 0.25, 'qtAssist <= 3.5\ngini = 0.262\nsamples = 632\nvalue = [534, 98]'),
Text(0.026993865030674847, 0.15, 'qtHits <= 67.0\ngini = 0.206\nsamples = 550\nvalue = [486, 64]'),
Text(0.025766871165644172, 0.05, '\n  (...)  \n'),
Text(0.02822085889570552, 0.05, '\n  (...)  \n'),
Text(0.03190184049079755, 0.15, 'vlDamage <= 1237.5\ngini = 0.485\nsamples = 82\nvalue = [48, 34]'),
Text(0.03067484662576687, 0.05, '\n  (...)  \n'),
Text(0.033128834355828224, 0.05, '\n  (...)  \n'),
Text(0.053067484662576686, 0.45, 'qtAssist <= 5.5\ngini = 0.049\nsamples = 17398\nvalue = [16963, 435]'),
Text(0.044171779141104296, 0.35, 'qtFirstKill <= 4.5\ngini = 0.035\nsamples = 16106\nvalue = [15817, 289]'),
Text(0.0392638036809816, 0.25, 'qtTrade <= 5.5\ngini = 0.027\nsamples = 15281\nvalue = [15072, 209]'),
Text(0.03680981595092025, 0.15, 'qtSurvived <= 2.5\ngini = 0.021\nsamples = 14548\nvalue = [14392, 156]'),
Text(0.03558282208588957, 0.05, '\n  (...)  \n'),
Text(0.03803680981595092, 0.05, '\n  (...)  \n'),
Text(0.04171779141104295, 0.15, 'vlDamage <= 2545.0\ngini = 0.134\nsamples = 733\nvalue = [680, 53]'),
Text(0.04049079754601227, 0.05, '\n  (...)  \n'),
Text(0.04294478527607362, 0.05, '\n  (...)  \n'),
Text(0.049079754601226995, 0.25, 'qtTrade <= 4.5\ngini = 0.175\nsamples = 825\nvalue = [745, 80]'),
Text(0.046625766871165646, 0.15, 'qtBombePlant <= 6.5\ngini = 0.118\nsamples = 683\nvalue = [640, 43]'),
Text(0.04539877300613497, 0.05, '\n  (...)  \n'),
Text(0.04785276073619632, 0.05, '\n  (...)  \n'),
Text(0.051533742331288344, 0.15, 'qtClutchWon <= 0.5\ngini = 0.385\nsamples = 142\nvalue = [105, 37]'),
Text(0.05030674846625767, 0.05, '\n  (...)  \n'),
Text(0.05276073619631902, 0.05, '\n  (...)  \n'),
Text(0.06196319018404908, 0.35, 'qtFirstKill <= 6.5\ngini = 0.2\nsamples = 1292\nvalue = [1146, 146]'),
Text(0.05889570552147239, 0.25, 'qtSurvived <= 2.5\ngini = 0.189\nsamples = 1276\nvalue = [1141, 135]'),
Text(0.05644171779141104, 0.15, 'qtTrade <= 9.5\ngini = 0.091\nsamples = 668\nvalue = [636, 32]'),
Text(0.05521472392638037, 0.05, '\n  (...)  \n'),
Text(0.05766871165644172, 0.05, '\n  (...)  \n'),
Text(0.06134969325153374, 0.15, 'qtDeath <= 16.5\ngini = 0.281\nsamples = 608\nvalue = [505, 103]'),
Text(0.060122699386503067, 0.05, '\n  (...)  \n'),
Text(0.06257668711656442, 0.05, '\n  (...)  \n'),
Text(0.06503067484662577, 0.25, 'qt2Kill <= 1.5\ngini = 0.43\nsamples = 16\nvalue = [5, 11]'),
Text(0.0638036809815951, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.06625766871165645, 0.15, 'qtHitChest <= 53.5\ngini = 0.26\nsamples = 13\nvalue = [2, 11]'),
Text(0.06503067484662577, 0.05, '\n  (...)  \n'),
```

```
Text(0.06748466257668712, 0.05, '\n  (...)  \n'),
Text(0.09187116564417178, 0.55, 'qtRoundsPlayed <= 28.5\ngini = 0.279\nsamples = 2521\nvalue = [2099, 422]'),
Text(0.0785276073619632, 0.45, 'qtFirstKill <= 1.5\ngini = 0.211\nsamples = 1532\nvalue = [1348, 184]'),
Text(0.0736196319018405, 0.35, 'qtTrade <= 11.5\ngini = 0.082\nsamples = 397\nvalue = [380, 17]'),
Text(0.07239263803680981, 0.25, 'qtHitLeftLeg <= 12.0\ngini = 0.073\nsamples = 395\nvalue = [380, 15]'),
Text(0.07116564417177915, 0.15, 'qtHitLeftAtm <= 3.5\ngini = 0.069\nsamples = 394\nvalue = [380, 14]'),
Text(0.06993865030674846, 0.05, '\n  (...)  \n'),
Text(0.07239263803680981, 0.05, '\n  (...)  \n'),
Text(0.0736196319018405, 0.15, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.07484662576687116, 0.25, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.0834355828220859, 0.35, 'qtTrade <= 7.5\ngini = 0.251\nsamples = 1135\nvalue = [968, 167]'),
Text(0.0785276073619632, 0.25, 'qtHitChest <= 38.5\ngini = 0.237\nsamples = 1078\nvalue = [930, 148]'),
Text(0.07607361963190185, 0.15, 'qt2Kill <= 7.5\ngini = 0.209\nsamples = 886\nvalue = [781, 105]'),
Text(0.07484662576687116, 0.05, '\n  (...)  \n'),
Text(0.07730061349693251, 0.05, '\n  (...)  \n'),
Text(0.08098159509202454, 0.15, 'qtHitRightLeg <= 0.5\ngini = 0.348\nsamples = 192\nvalue = [149, 43]'),
Text(0.07975460122699386, 0.05, '\n  (...)  \n'),
Text(0.08220858895705521, 0.05, '\n  (...)  \n'),
Text(0.08834355828220859, 0.25, 'qtHitRightArm <= 7.5\ngini = 0.444\nsamples = 57\nvalue = [38, 19]'),
Text(0.08588957055214724, 0.15, 'vlDamage <= 1977.0\ngini = 0.364\nsamples = 46\nvalue = [35, 11]'),
Text(0.08466257668711656, 0.05, '\n  (...)  \n'),
Text(0.08711656441717791, 0.05, '\n  (...)  \n'),
Text(0.09079754601226994, 0.15, 'vlLevel <= 10.5\ngini = 0.397\nsamples = 11\nvalue = [3, 8]'),
Text(0.08957055214723926, 0.05, '\n  (...)  \n'),
Text(0.09202453987730061, 0.05, '\n  (...)  \n'),
Text(0.10521472392638037, 0.45, 'qtTrade <= 4.5\ngini = 0.365\nsamples = 989\nvalue = [751, 238]'),
Text(0.09938650306748466, 0.35, 'qtFlashAssist <= 3.5\ngini = 0.278\nsamples = 456\nvalue = [380, 76]'),
Text(0.09815950920245399, 0.25, 'qt2Kill <= 5.5\ngini = 0.27\nsamples = 453\nvalue = [380, 73]'),
Text(0.09570552147239264, 0.15, 'qtRoundsPlayed <= 29.5\ngini = 0.241\nsamples = 406\nvalue = [349, 57]'),
Text(0.09447852760736196, 0.05, '\n  (...)  \n'),
Text(0.09693251533742331, 0.05, '\n  (...)  \n'),
Text(0.10061349693251534, 0.15, 'qtTk <= 0.5\ngini = 0.449\nsamples = 47\nvalue = [31, 16]'),
Text(0.09938650306748466, 0.05, '\n  (...)  \n'),
Text(0.10184049079754601, 0.05, '\n  (...)  \n'),
Text(0.10061349693251534, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.11104294478527607, 0.35, 'vlLevel <= 7.5\ngini = 0.423\nsamples = 533\nvalue = [371, 162]'),
Text(0.10797546012269939, 0.25, 'qt2Kill <= 1.5\ngini = 0.498\nsamples = 73\nvalue = [39, 34]'),
Text(0.10552147239263804, 0.15, 'de_train <= 0.5\ngini = 0.375\nsamples = 28\nvalue = [21, 7]'),
Text(0.10429447852760736, 0.05, '\n  (...)  \n'),
Text(0.1067484662576687, 0.05, '\n  (...)  \n'),
Text(0.11042944785276074, 0.15, 'qtFirstKill <= 5.5\ngini = 0.48\nsamples = 45\nvalue = [18, 27]'),
Text(0.10920245398773006, 0.05, '\n  (...)  \n'),
Text(0.1116564417177914, 0.05, '\n  (...)  \n'),
Text(0.11411042944785275, 0.25, 'qtHitHeadshot <= 1.5\ngini = 0.402\nsamples = 460\nvalue = [332, 128]'),
Text(0.11288343558282209, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.11533742331288344, 0.15, 'vlDamage <= 3910.5\ngini = 0.399\nsamples = 458\nvalue = [332, 126]'),
Text(0.11411042944785275, 0.05, '\n  (...)  \n'),
Text(0.1165644171779141, 0.05, '\n  (...)  \n'),
Text(0.17124233128834357, 0.65, 'qtDeath <= 15.5\ngini = 0.279\nsamples = 11240\nvalue = [9357, 1883]'),
```

```
Text(0.14539877300613496, 0.55, 'qtTrade <= 3.5\ngini = 0.499\nsamples = 629\nvalue = [330, 299]'),
Text(0.13312883435582823, 0.45, 'qtRoundsPlayed <= 12.5\ngini = 0.487\nsamples = 547\nvalue = [318, 229]'),
Text(0.1245398773006135, 0.35, 'qtDeath <= 1.5\ngini = 0.46\nsamples = 176\nvalue = [63, 113]'),
Text(0.12147239263803682, 0.25, 'qtHitHeadshot <= 1.5\ngini = 0.36\nsamples = 17\nvalue = [13, 4]'),
Text(0.12024539877300613, 0.15, 'vlLevel <= 12.5\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
Text(0.11901840490797547, 0.05, '\n  (...)  \n'),
Text(0.12147239263803682, 0.05, '\n  (...)  \n'),
Text(0.12269938650306748, 0.15, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
Text(0.1276073619631902, 0.25, 'qtRoundsPlayed <= 9.5\ngini = 0.431\nsamples = 159\nvalue = [50, 109]'),
Text(0.12515337423312883, 0.15, 'vlDamage <= 654.5\ngini = 0.26\nsamples = 78\nvalue = [12, 66]'),
Text(0.12392638036809817, 0.05, '\n  (...)  \n'),
Text(0.1263803680981595, 0.05, '\n  (...)  \n'),
Text(0.13006134969325153, 0.15, 'qtFirstKill <= 1.5\ngini = 0.498\nsamples = 81\nvalue = [38, 43]'),
Text(0.12883435582822086, 0.05, '\n  (...)  \n'),
Text(0.1312883435582822, 0.05, '\n  (...)  \n'),
Text(0.14171779141104293, 0.35, 'qtAssist <= 5.5\ngini = 0.43\nsamples = 371\nvalue = [255, 116]'),
Text(0.1374233128834356, 0.25, 'qtTrade <= 1.5\ngini = 0.406\nsamples = 353\nvalue = [253, 100]'),
Text(0.13496932515337423, 0.15, 'qtFirstKill <= 2.5\ngini = 0.27\nsamples = 174\nvalue = [146, 28]'),
Text(0.13374233128834356, 0.05, '\n  (...)  \n'),
Text(0.1361963190184049, 0.05, '\n  (...)  \n'),
Text(0.13987730061349693, 0.15, 'de_vertigo <= 0.5\ngini = 0.481\nsamples = 179\nvalue = [107, 72]'),
Text(0.13865030674846626, 0.05, '\n  (...)  \n'),
Text(0.1411042944785276, 0.05, '\n  (...)  \n'),
Text(0.1460122699386503, 0.25, 'qtShots <= 240.0\ngini = 0.198\nsamples = 18\nvalue = [2, 16]'),
Text(0.14478527607361963, 0.15, 'qtHits <= 23.0\ngini = 0.444\nsamples = 3\nvalue = [2, 1]'),
Text(0.14355828220858896, 0.05, '\n  (...)  \n'),
Text(0.1460122699386503, 0.05, '\n  (...)  \n'),
Text(0.147239263803681, 0.15, 'gini = 0.0\nsamples = 15\nvalue = [0, 15]'),
Text(0.15766871165644172, 0.45, 'qtRoundsPlayed <= 22.5\ngini = 0.25\nsamples = 82\nvalue = [12, 70]'),
Text(0.15644171779141106, 0.35, 'vlLevel <= 9.5\ngini = 0.219\nsamples = 80\nvalue = [10, 70]'),
Text(0.1521472392638037, 0.25, 'qtFirstKill <= 1.5\ngini = 0.42\nsamples = 20\nvalue = [6, 14]'),
Text(0.14969325153374233, 0.15, 'qtShots <= 318.0\ngini = 0.469\nsamples = 8\nvalue = [5, 3]'),
Text(0.14846625766871166, 0.05, '\n  (...)  \n'),
Text(0.150920245398773, 0.05, '\n  (...)  \n'),
Text(0.15460122699386503, 0.15, 'qtHitLeftLeg <= 7.5\ngini = 0.153\nsamples = 12\nvalue = [1, 11]'),
Text(0.15337423312883436, 0.05, '\n  (...)  \n'),
Text(0.1558282208588957, 0.05, '\n  (...)  \n'),
Text(0.1607361963190184, 0.25, 'qtAssist <= 2.5\ngini = 0.124\nsamples = 60\nvalue = [4, 56]'),
Text(0.15950920245398773, 0.15, 'vlDamage <= 2441.0\ngini = 0.36\nsamples = 17\nvalue = [4, 13]'),
Text(0.15828220858895706, 0.05, '\n  (...)  \n'),
Text(0.1607361963190184, 0.05, '\n  (...)  \n'),
Text(0.1619631901840491, 0.15, 'gini = 0.0\nsamples = 43\nvalue = [0, 43]'),
Text(0.1588957055214724, 0.35, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.19708588957055215, 0.55, 'qtTrade <= 3.5\ngini = 0.254\nsamples = 10611\nvalue = [9027, 1584]'),
Text(0.17822085889570552, 0.45, 'qtFirstKill <= 3.5\ngini = 0.159\nsamples = 6208\nvalue = [5668, 540]'),
Text(0.16993865030674846, 0.35, 'qtAssist <= 6.5\ngini = 0.112\nsamples = 4845\nvalue = [4556, 289]'),
Text(0.1668711656441718, 0.25, 'qtRoundsPlayed <= 27.5\ngini = 0.097\nsamples = 4601\nvalue = [4366, 235]'),
Text(0.16441717791411042, 0.15, 'qtClutchWon <= 3.5\ngini = 0.074\nsamples = 4071\nvalue = [3914, 157]'),
Text(0.16319018404907976, 0.05, '\n  (...)  \n'),
```

```
Text(0.1656441717791411, 0.05, '\n  (...)  \n'),
Text(0.16932515337423312, 0.15, 'qtRoundsPlayed <= 29.5\ngini = 0.251\nsamples = 530\nvalue = [452, 78]'),
Text(0.16809815950920245, 0.05, '\n  (...)  \n'),
Text(0.1705521472392638, 0.05, '\n  (...)  \n'),
Text(0.17300613496932515, 0.25, 'qtDeath <= 16.5\ngini = 0.345\nsamples = 244\nvalue = [190, 54]'),
Text(0.17177914110429449, 0.15, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(0.17423312883435582, 0.15, 'qtRoundsPlayed <= 24.5\ngini = 0.326\nsamples = 239\nvalue = [190, 49]'),
Text(0.17300613496932515, 0.05, '\n  (...)  \n'),
Text(0.1754601226993865, 0.05, '\n  (...)  \n'),
Text(0.18650306748466258, 0.35, 'qtAssist <= 4.5\ngini = 0.3\nsamples = 1363\nvalue = [1112, 251]'),
Text(0.18159509202453988, 0.25, 'qtFirstKill <= 5.5\ngini = 0.231\nsamples = 929\nvalue = [805, 124]'),
Text(0.17914110429447852, 0.15, 'qtClutchWon <= 1.5\ngini = 0.176\nsamples = 738\nvalue = [666, 72]'),
Text(0.17791411042944785, 0.05, '\n  (...)  \n'),
Text(0.18036809815950922, 0.05, '\n  (...)  \n'),
Text(0.18404907975460122, 0.15, 'qtClutchWon <= 0.5\ngini = 0.396\nsamples = 191\nvalue = [139, 52]'),
Text(0.18282208588957055, 0.05, '\n  (...)  \n'),
Text(0.18527607361963191, 0.05, '\n  (...)  \n'),
Text(0.19141104294478528, 0.25, 'qtDeath <= 18.5\ngini = 0.414\nsamples = 434\nvalue = [307, 127]'),
Text(0.18895705521472392, 0.15, 'qtShots <= 290.5\ngini = 0.469\nsamples = 32\nvalue = [12, 20]'),
Text(0.18773006134969325, 0.05, '\n  (...)  \n'),
Text(0.1901840490797546, 0.05, '\n  (...)  \n'),
Text(0.19386503067484662, 0.15, 'qtHits <= 70.5\ngini = 0.391\nsamples = 402\nvalue = [295, 107]'),
Text(0.19263803680981595, 0.05, '\n  (...)  \n'),
Text(0.1950920245398773, 0.05, '\n  (...)  \n'),
Text(0.21595092024539878, 0.45, 'qtAssist <= 3.5\ngini = 0.362\nsamples = 4403\nvalue = [3359, 1044]'),
Text(0.20613496932515338, 0.35, 'qtTrade <= 6.5\ngini = 0.256\nsamples = 1780\nvalue = [1512, 268]'),
Text(0.20122699386503068, 0.25, 'qtDeath <= 16.5\ngini = 0.229\nsamples = 1602\nvalue = [1391, 211]'),
Text(0.19877300613496932, 0.15, 'qtHitHeadshot <= 4.5\ngini = 0.486\nsamples = 24\nvalue = [10, 14]'),
Text(0.19754601226993865, 0.05, '\n  (...)  \n'),
Text(0.2, 0.05, '\n  (...)  \n'),
Text(0.20368098159509201, 0.15, 'qtRoundsPlayed <= 28.5\ngini = 0.219\nsamples = 1578\nvalue = [1381, 197]'),
Text(0.20245398773006135, 0.05, '\n  (...)  \n'),
Text(0.2049079754601227, 0.05, '\n  (...)  \n'),
Text(0.21104294478527608, 0.25, 'qtFirstKill <= 4.5\ngini = 0.435\nsamples = 178\nvalue = [121, 57]'),
Text(0.2085889570552147, 0.15, 'qtHitLeftAtm <= 6.5\ngini = 0.382\nsamples = 144\nvalue = [107, 37]'),
Text(0.20736196319018405, 0.05, '\n  (...)  \n'),
Text(0.2098159509202454, 0.05, '\n  (...)  \n'),
Text(0.2134969325153374, 0.15, 'qtDeath <= 21.5\ngini = 0.484\nsamples = 34\nvalue = [14, 20]'),
Text(0.21226993865030674, 0.05, '\n  (...)  \n'),
Text(0.2147239263803681, 0.05, '\n  (...)  \n'),
Text(0.22576687116564417, 0.35, 'qtFirstKill <= 4.5\ngini = 0.417\nsamples = 2623\nvalue = [1847, 776]'),
Text(0.22085889570552147, 0.25, 'qtTrade <= 5.5\ngini = 0.392\nsamples = 2162\nvalue = [1583, 579]'),
Text(0.2184049079754601, 0.15, 'qtDeath <= 17.5\ngini = 0.352\nsamples = 1444\nvalue = [1115, 329]'),
Text(0.21717791411042944, 0.05, '\n  (...)  \n'),
Text(0.2196319018404908, 0.05, '\n  (...)  \n'),
Text(0.2233128834355828, 0.15, 'vlLevel <= 12.5\ngini = 0.454\nsamples = 718\nvalue = [468, 250]'),
Text(0.22208588957055214, 0.05, '\n  (...)  \n'),
Text(0.2245398773006135, 0.05, '\n  (...)  \n'),
Text(0.23067484662576687, 0.25, 'qtDeath <= 19.5\ngini = 0.489\nsamples = 461\nvalue = [264, 197]'),
```

```
Text(0.2282208588957055, 0.15, 'qt2Kill <= 1.5\ngini = 0.308\nsamples = 42\nvalue = [8, 34]'),
Text(0.22699386503067484, 0.05, '\n  (...)  \n'),
Text(0.2294478527607362, 0.05, '\n  (...)  \n'),
Text(0.2331288343558282, 0.15, 'qtDeath <= 23.5\ngini = 0.475\nsamples = 419\nvalue = [256, 163]'),
Text(0.23190184049079754, 0.05, '\n  (...)  \n'),
Text(0.2343558282208589, 0.05, '\n  (...)  \n'),
Text(0.3668328220858896, 0.75, 'qtDeath <= 16.5\ngini = 0.46\nsamples = 29548\nvalue = [18952, 10596]'),
Text(0.2979294478527607, 0.65, 'qtTrade <= 2.5\ngini = 0.412\nsamples = 3517\nvalue = [1021, 2496]'),
Text(0.2659509202453988, 0.55, 'qtAssist <= 3.5\ngini = 0.495\nsamples = 1755\nvalue = [793, 962]'),
Text(0.25245398773006134, 0.45, 'qtDeath <= 14.5\ngini = 0.492\nsamples = 1190\nvalue = [668, 522]'),
Text(0.24478527607361963, 0.35, 'qtFirstKill <= 2.5\ngini = 0.478\nsamples = 598\nvalue = [236, 362]'),
Text(0.24049079754601227, 0.25, 'qtLastAlive <= 2.5\ngini = 0.499\nsamples = 404\nvalue = [195, 209]'),
Text(0.23803680981595093, 0.15, 'qtTrade <= 1.5\ngini = 0.493\nsamples = 364\nvalue = [160, 204]'),
Text(0.23680981595092024, 0.05, '\n  (...)  \n'),
Text(0.2392638036809816, 0.05, '\n  (...)  \n'),
Text(0.24294478527607363, 0.15, 'qtClutchWon <= 2.5\ngini = 0.219\nsamples = 40\nvalue = [35, 5]'),
Text(0.24171779141104294, 0.05, '\n  (...)  \n'),
Text(0.2441717791411043, 0.05, '\n  (...)  \n'),
Text(0.249079754601227, 0.25, 'qtRoundsPlayed <= 23.5\ngini = 0.333\nsamples = 194\nvalue = [41, 153]'),
Text(0.24785276073619633, 0.15, 'qtTkAssist <= 1.5\ngini = 0.319\nsamples = 191\nvalue = [38, 153]'),
Text(0.24662576687116564, 0.05, '\n  (...)  \n'),
Text(0.249079754601227, 0.05, '\n  (...)  \n'),
Text(0.25030674846625767, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.26012269938650306, 0.35, 'qtFirstKill <= 3.5\ngini = 0.394\nsamples = 592\nvalue = [432, 160]'),
Text(0.2552147239263804, 0.25, 'qtAssist <= 2.5\ngini = 0.343\nsamples = 487\nvalue = [380, 107]'),
Text(0.252760736196319, 0.15, 'qtClutchWon <= 0.5\ngini = 0.264\nsamples = 352\nvalue = [297, 55]'),
Text(0.25153374233128833, 0.05, '\n  (...)  \n'),
Text(0.25398773006134967, 0.05, '\n  (...)  \n'),
Text(0.25766871165644173, 0.15, 'qtDeath <= 15.5\ngini = 0.474\nsamples = 135\nvalue = [83, 52]'),
Text(0.25644171779141106, 0.05, '\n  (...)  \n'),
Text(0.2588957055214724, 0.05, '\n  (...)  \n'),
Text(0.2650306748466258, 0.25, 'qtShots <= 301.0\ngini = 0.5\nsamples = 105\nvalue = [52, 53]'),
Text(0.2625766871165644, 0.15, 'qtHitStomach <= 13.5\ngini = 0.308\nsamples = 21\nvalue = [17, 4]'),
Text(0.26134969325153373, 0.05, '\n  (...)  \n'),
Text(0.26380368098159507, 0.05, '\n  (...)  \n'),
Text(0.2674846625766871, 0.15, 'qtHitChest <= 34.5\ngini = 0.486\nsamples = 84\nvalue = [35, 49]'),
Text(0.26625766871165646, 0.05, '\n  (...)  \n'),
Text(0.2687116564417178, 0.05, '\n  (...)  \n'),
Text(0.2794478527607362, 0.45, 'qtDeath <= 15.5\ngini = 0.345\nsamples = 565\nvalue = [125, 440]'),
Text(0.27361963190184047, 0.35, 'qtLastAlive <= 3.5\ngini = 0.235\nsamples = 405\nvalue = [55, 350]'),
Text(0.27116564417177913, 0.25, 'vlDamage <= 994.5\ngini = 0.214\nsamples = 395\nvalue = [48, 347]'),
Text(0.26993865030674846, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.2723926380368098, 0.15, 'qtHitRightArm <= 3.5\ngini = 0.203\nsamples = 392\nvalue = [45, 347]'),
Text(0.27116564417177913, 0.05, '\n  (...)  \n'),
Text(0.27361963190184047, 0.05, '\n  (...)  \n'),
Text(0.27607361963190186, 0.25, 'qtClutchWon <= 2.5\ngini = 0.42\nsamples = 10\nvalue = [7, 3]'),
Text(0.2748466257668712, 0.15, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(0.2773006134969325, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.2852760736196319, 0.35, 'qtHits <= 66.5\ngini = 0.492\nsamples = 160\nvalue = [70, 90]'),
```

```
Text(0.2822085889570552, 0.25, 'qtHitStomach <= 10.5\ngini = 0.487\nsamples = 100\nvalue = [58, 42]'),
Text(0.27975460122699386, 0.15, 'vlLevel <= 5.0\ngini = 0.332\nsamples = 38\nvalue = [30, 8]'),
Text(0.2785276073619632, 0.05, '\n  (...)  \n'),
Text(0.2809815950920245, 0.05, '\n  (...)  \n'),
Text(0.2846625766871166, 0.15, 'qtLastAlive <= 0.5\ngini = 0.495\nsamples = 62\nvalue = [28, 34]'),
Text(0.28343558282208586, 0.05, '\n  (...)  \n'),
Text(0.28588957055214725, 0.05, '\n  (...)  \n'),
Text(0.2883435582822086, 0.25, 'qtFirstKill <= 0.5\ngini = 0.32\nsamples = 60\nvalue = [12, 48]'),
Text(0.2871165644171779, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.28957055214723926, 0.15, 'qtFlashAssist <= 0.5\ngini = 0.266\nsamples = 57\nvalue = [9, 48]'),
Text(0.2883435582822086, 0.05, '\n  (...)  \n'),
Text(0.2907975460122699, 0.05, '\n  (...)  \n'),
Text(0.3299079754601227, 0.55, 'qtAssist <= 2.5\ngini = 0.225\nsamples = 1762\nvalue = [228, 1534]'),
Text(0.3116564417177914, 0.45, 'qtDeath <= 15.5\ngini = 0.381\nsamples = 495\nvalue = [127, 368]'),
Text(0.301840490797546, 0.35, 'vlDamage <= 918.5\ngini = 0.272\nsamples = 338\nvalue = [55, 283]'),
Text(0.2969325153374233, 0.25, 'qtBombePlant <= 1.5\ngini = 0.465\nsamples = 19\nvalue = [12, 7]'),
Text(0.294478527607362, 0.15, 'qtRoundsPlayed <= 13.5\ngini = 0.153\nsamples = 12\nvalue = [11, 1]'),
Text(0.29325153374233126, 0.05, '\n  (...)  \n'),
Text(0.29570552147239265, 0.05, '\n  (...)  \n'),
Text(0.29938650306748466, 0.15, 'vlLevel <= 12.5\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
Text(0.298159509202454, 0.05, '\n  (...)  \n'),
Text(0.3006134969325153, 0.05, '\n  (...)  \n'),
Text(0.3067484662576687, 0.25, 'qtTrade <= 3.5\ngini = 0.233\nsamples = 319\nvalue = [43, 276]'),
Text(0.3042944785276074, 0.15, 'qtSurvived <= 5.5\ngini = 0.37\nsamples = 143\nvalue = [35, 108]'),
Text(0.3030674846625767, 0.05, '\n  (...)  \n'),
Text(0.30552147239263805, 0.05, '\n  (...)  \n'),
Text(0.30920245398773005, 0.15, 'qtTk <= 0.5\ngini = 0.087\nsamples = 176\nvalue = [8, 168]'),
Text(0.3079754601226994, 0.05, '\n  (...)  \n'),
Text(0.3104294478527607, 0.05, '\n  (...)  \n'),
Text(0.3214723926380368, 0.35, 'qtSurvived <= 5.5\ngini = 0.497\nsamples = 157\nvalue = [72, 85]'),
Text(0.3165644171779141, 0.25, 'qt1Kill <= 7.5\ngini = 0.454\nsamples = 69\nvalue = [45, 24]'),
Text(0.3141104294478528, 0.15, 'qtHitLeftLeg <= 0.5\ngini = 0.32\nsamples = 40\nvalue = [32, 8]'),
Text(0.3128834355828221, 0.05, '\n  (...)  \n'),
Text(0.31533742331288345, 0.05, '\n  (...)  \n'),
Text(0.31901840490797545, 0.15, 'qtAssist <= 1.5\ngini = 0.495\nsamples = 29\nvalue = [13, 16]'),
Text(0.3177914110429448, 0.05, '\n  (...)  \n'),
Text(0.3202453987730061, 0.05, '\n  (...)  \n'),
Text(0.3263803680981595, 0.25, 'qtHitHeadshot <= 10.5\ngini = 0.425\nsamples = 88\nvalue = [27, 61]'),
Text(0.3239263803680982, 0.15, 'qtHits <= 57.5\ngini = 0.486\nsamples = 60\nvalue = [25, 35]'),
Text(0.3226993865030675, 0.05, '\n  (...)  \n'),
Text(0.32515337423312884, 0.05, '\n  (...)  \n'),
Text(0.32883435582822085, 0.15, 'qtShots <= 221.0\ngini = 0.133\nsamples = 28\nvalue = [2, 26]'),
Text(0.3276073619631902, 0.05, '\n  (...)  \n'),
Text(0.3300613496932515, 0.05, '\n  (...)  \n'),
Text(0.348159509202454, 0.45, 'qtRoundsPlayed <= 23.5\ngini = 0.147\nsamples = 1267\nvalue = [101, 1166]'),
Text(0.3411042944785276, 0.35, 'qtSurvived <= 5.5\ngini = 0.123\nsamples = 1168\nvalue = [77, 1091]'),
Text(0.3361963190184049, 0.25, 'qtHits <= 40.5\ngini = 0.219\nsamples = 391\nvalue = [49, 342]'),
Text(0.3337423312883436, 0.15, 'qtTrade <= 3.5\ngini = 0.441\nsamples = 61\nvalue = [20, 41]'),
Text(0.3325153374233129, 0.05, '\n  (...)  \n'),
```

```
Text(0.33496932515337424, 0.05, '\n  (...)  \n'),
Text(0.33865030674846625, 0.15, 'qtRoundsPlayed <= 21.5\ngini = 0.16\nsamples = 330\nvalue = [29, 301]'),
Text(0.3374233128834356, 0.05, '\n  (...)  \n'),
Text(0.3398773006134969, 0.05, '\n  (...)  \n'),
Text(0.3460122699386503, 0.25, 'qtFirstKill <= 0.5\ngini = 0.069\nsamples = 777\nvalue = [28, 749]'),
Text(0.34355828220858897, 0.15, 'qtHitStomach <= 20.5\ngini = 0.216\nsamples = 73\nvalue = [9, 64]'),
Text(0.3423312883435583, 0.05, '\n  (...)  \n'),
Text(0.34478527607361964, 0.05, '\n  (...)  \n'),
Text(0.34846625766871164, 0.15, 'qtClutchWon <= 0.5\ngini = 0.053\nsamples = 704\nvalue = [19, 685]'),
Text(0.347239263803681, 0.05, '\n  (...)  \n'),
Text(0.3496932515337423, 0.05, '\n  (...)  \n'),
Text(0.35521472392638037, 0.35, 'qtShots <= 330.5\ngini = 0.367\nsamples = 99\nvalue = [24, 75]'),
Text(0.3521472392638037, 0.25, 'vlDamage <= 2833.5\ngini = 0.32\nsamples = 10\nvalue = [8, 2]'),
Text(0.350920245398773, 0.15, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]'),
Text(0.35337423312883437, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.35828220858895704, 0.25, 'qtFirstKill <= 0.5\ngini = 0.295\nsamples = 89\nvalue = [16, 73]'),
Text(0.3558282208588957, 0.15, 'qtAssist <= 4.5\ngini = 0.48\nsamples = 10\nvalue = [6, 4]'),
Text(0.35460122699386504, 0.05, '\n  (...)  \n'),
Text(0.3570552147239264, 0.05, '\n  (...)  \n'),
Text(0.36073619631901843, 0.15, 'qtHitRightLeg <= 4.5\ngini = 0.221\nsamples = 79\nvalue = [10, 69]'),
Text(0.3595092024539877, 0.05, '\n  (...)  \n'),
Text(0.3619631901840491, 0.05, '\n  (...)  \n'),
Text(0.4357361963190184, 0.65, 'qtTrade <= 3.5\ngini = 0.429\nsamples = 26031\nvalue = [17931, 8100]'),
Text(0.3983128834355828, 0.55, 'qtAssist <= 3.5\ngini = 0.348\nsamples = 14008\nvalue = [10870, 3138]'),
Text(0.37944785276073617, 0.45, 'qtFirstKill <= 3.5\ngini = 0.259\nsamples = 7378\nvalue = [6248, 1130]'),
Text(0.37116564417177916, 0.35, 'qtRoundsPlayed <= 28.5\ngini = 0.217\nsamples = 5533\nvalue = [4848, 685]'),
Text(0.36809815950920244, 0.25, 'qtSurvived <= 5.5\ngini = 0.189\nsamples = 4729\nvalue = [4229, 500]'),
Text(0.3656441717791411, 0.15, 'qtRoundsPlayed <= 26.5\ngini = 0.133\nsamples = 2557\nvalue = [2374, 183]'),
Text(0.36441717791411044, 0.05, '\n  (...)  \n'),
Text(0.36687116564417177, 0.05, '\n  (...)  \n'),
Text(0.37055214723926383, 0.15, 'qtDeath <= 17.5\ngini = 0.249\nsamples = 2172\nvalue = [1855, 317]'),
Text(0.3693251533742331, 0.05, '\n  (...)  \n'),
Text(0.3717791411042945, 0.05, '\n  (...)  \n'),
Text(0.37423312883435583, 0.25, 'qtAssist <= 0.5\ngini = 0.354\nsamples = 804\nvalue = [619, 185]'),
Text(0.37300613496932516, 0.15, 'gini = 0.0\nsamples = 33\nvalue = [33, 0]'),
Text(0.3754601226993865, 0.15, 'qtHitHeadshot <= 22.5\ngini = 0.365\nsamples = 771\nvalue = [586, 185]'),
Text(0.37423312883435583, 0.05, '\n  (...)  \n'),
Text(0.37668711656441717, 0.05, '\n  (...)  \n'),
Text(0.38773006134969323, 0.35, 'qtDeath <= 18.5\ngini = 0.366\nsamples = 1845\nvalue = [1400, 445]'),
Text(0.38282208588957056, 0.25, 'qtTrade <= 1.5\ngini = 0.443\nsamples = 377\nvalue = [252, 125]'),
Text(0.3803680981595092, 0.15, 'qtFirstKill <= 7.5\ngini = 0.323\nsamples = 158\nvalue = [126, 32]'),
Text(0.3791411042944785, 0.05, '\n  (...)  \n'),
Text(0.3815950920245399, 0.05, '\n  (...)  \n'),
Text(0.3852760736196319, 0.15, 'vlLevel <= 10.5\ngini = 0.489\nsamples = 219\nvalue = [126, 93]'),
Text(0.38404907975460123, 0.05, '\n  (...)  \n'),
Text(0.38650306748466257, 0.05, '\n  (...)  \n'),
Text(0.39263803680981596, 0.25, 'vlDamage <= 2522.5\ngini = 0.341\nsamples = 1468\nvalue = [1148, 320]'),
Text(0.3901840490797546, 0.15, 'qtHitLeftLeg <= 6.5\ngini = 0.27\nsamples = 529\nvalue = [444, 85]'),
Text(0.3889570552147239, 0.05, '\n  (...)  \n'),
```

```
Text(0.3914110429447853, 0.05, '\n  (...)  \n'),
Text(0.3950920245398773, 0.15, 'qtShots <= 573.5\ngini = 0.375\nsamples = 939\nvalue = [704, 235]'),
Text(0.39386503067484663, 0.05, '\n  (...)  \n'),
Text(0.39631901840490796, 0.05, '\n  (...)  \n'),
Text(0.4171779141104294, 0.45, 'qtSurvived <= 5.5\ngini = 0.422\nsamples = 6630\nvalue = [4622, 2008]'),
Text(0.40736196319018403, 0.35, 'qtFirstKill <= 4.5\ngini = 0.35\nsamples = 2988\nvalue = [2313, 675]'),
Text(0.40245398773006136, 0.25, 'qtClutchWon <= 0.5\ngini = 0.322\nsamples = 2523\nvalue = [2014, 509]'),
Text(0.4, 0.15, 'qtAssist <= 6.5\ngini = 0.275\nsamples = 1604\nvalue = [1340, 264]'),
Text(0.3987730061349693, 0.05, '\n  (...)  \n'),
Text(0.4012269938650307, 0.05, '\n  (...)  \n'),
Text(0.4049079754601227, 0.15, 'qtShots <= 922.0\ngini = 0.391\nsamples = 919\nvalue = [674, 245]'),
Text(0.403680981595092, 0.05, '\n  (...)  \n'),
Text(0.40613496932515336, 0.05, '\n  (...)  \n'),
Text(0.41226993865030676, 0.25, 'qt2Kill <= 5.5\ngini = 0.459\nsamples = 465\nvalue = [299, 166]'),
Text(0.4098159509202454, 0.15, 'qtHitRightLeg <= 7.5\ngini = 0.43\nsamples = 367\nvalue = [252, 115]'),
Text(0.4085889570552147, 0.05, '\n  (...)  \n'),
Text(0.4110429447852761, 0.05, '\n  (...)  \n'),
Text(0.4147239263803681, 0.15, 'qtHs <= 13.5\ngini = 0.499\nsamples = 98\nvalue = [47, 51]'),
Text(0.4134969325153374, 0.05, '\n  (...)  \n'),
Text(0.41595092024539876, 0.05, '\n  (...)  \n'),
Text(0.4269938650306748, 0.35, 'qtDeath <= 18.5\ngini = 0.464\nsamples = 3642\nvalue = [2309, 1333]'),
Text(0.42208588957055215, 0.25, 'qtFirstKill <= 3.5\ngini = 0.499\nsamples = 663\nvalue = [321, 342]'),
Text(0.4196319018404908, 0.15, 'qtTrade <= 2.5\ngini = 0.491\nsamples = 484\nvalue = [274, 210]'),
Text(0.41840490797546015, 0.05, '\n  (...)  \n'),
Text(0.4208588957055215, 0.05, '\n  (...)  \n'),
Text(0.4245398773006135, 0.15, 'vlDamage <= 2560.0\ngini = 0.387\nsamples = 179\nvalue = [47, 132]'),
Text(0.4233128834355828, 0.05, '\n  (...)  \n'),
Text(0.42576687116564416, 0.05, '\n  (...)  \n'),
Text(0.43190184049079755, 0.25, 'qtFirstKill <= 3.5\ngini = 0.444\nsamples = 2979\nvalue = [1988, 991]'),
Text(0.4294478527607362, 0.15, 'qtTrade <= 1.5\ngini = 0.412\nsamples = 1955\nvalue = [1387, 568]'),
Text(0.42822085889570555, 0.05, '\n  (...)  \n'),
Text(0.4306748466257669, 0.05, '\n  (...)  \n'),
Text(0.4343558282208589, 0.15, 'qtRoundsPlayed <= 27.5\ngini = 0.485\nsamples = 1024\nvalue = [601, 423]'),
Text(0.4331288343558282, 0.05, '\n  (...)  \n'),
Text(0.43558282208588955, 0.05, '\n  (...)  \n'),
Text(0.473159509202454, 0.55, 'qtDeath <= 18.5\ngini = 0.485\nsamples = 12023\nvalue = [7061, 4962]'),
Text(0.455521472392638, 0.45, 'qtFirstKill <= 2.5\ngini = 0.454\nsamples = 1312\nvalue = [457, 855]'),
Text(0.4466257668711656, 0.35, 'qtTrade <= 5.5\ngini = 0.495\nsamples = 734\nvalue = [330, 404]'),
Text(0.44171779141104295, 0.25, 'qtAssist <= 3.5\ngini = 0.499\nsamples = 567\nvalue = [297, 270]'),
Text(0.4392638036809816, 0.15, 'de_inferno <= 0.5\ngini = 0.451\nsamples = 274\nvalue = [180, 94]'),
Text(0.43803680981595094, 0.05, '\n  (...)  \n'),
Text(0.4404907975460123, 0.05, '\n  (...)  \n'),
Text(0.4441717791411043, 0.15, 'vlDamage <= 2370.5\ngini = 0.48\nsamples = 293\nvalue = [117, 176]'),
Text(0.4429447852760736, 0.05, '\n  (...)  \n'),
Text(0.44539877300613495, 0.05, '\n  (...)  \n'),
Text(0.45153374233128835, 0.25, 'qtShots <= 315.5\ngini = 0.317\nsamples = 167\nvalue = [33, 134]'),
Text(0.449079754601227, 0.15, 'qt3Kill <= 0.5\ngini = 0.497\nsamples = 28\nvalue = [13, 15]'),
Text(0.44785276073619634, 0.05, '\n  (...)  \n'),
Text(0.4503067484662577, 0.05, '\n  (...)  \n'),
```

```
Text(0.4539877300613497, 0.15, 'qtHitStomach <= 26.0\ngini = 0.246\nsamples = 139\nvalue = [20, 119]'),
Text(0.452760736196319, 0.05, '\n  (...)  \n'),
Text(0.45521472392638035, 0.05, '\n  (...)  \n'),
Text(0.4644171779141104, 0.35, 'qtAssist <= 4.5\ngini = 0.343\nsamples = 578\nvalue = [127, 451]'),
Text(0.46134969325153374, 0.25, 'qtDeath <= 17.5\ngini = 0.415\nsamples = 341\nvalue = [100, 241]'),
Text(0.4588957055214724, 0.15, 'qtHitStomach <= 2.5\ngini = 0.298\nsamples = 148\nvalue = [27, 121]'),
Text(0.45766871165644174, 0.05, '\n  (...)  \n'),
Text(0.4601226993865031, 0.05, '\n  (...)  \n'),
Text(0.4638036809815951, 0.15, 'qtHits <= 67.5\ngini = 0.47\nsamples = 193\nvalue = [73, 120]'),
Text(0.4625766871165644, 0.05, '\n  (...)  \n'),
Text(0.46503067484662575, 0.05, '\n  (...)  \n'),
Text(0.46748466257668714, 0.25, 'qtShots <= 229.0\ngini = 0.202\nsamples = 237\nvalue = [27, 210]'),
Text(0.4662576687116564, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.4687116564417178, 0.15, 'qt5Kill <= 0.5\ngini = 0.19\nsamples = 235\nvalue = [25, 210]'),
Text(0.46748466257668714, 0.05, '\n  (...)  \n'),
Text(0.4699386503067485, 0.05, '\n  (...)  \n'),
Text(0.49079754601226994, 0.45, 'qtAssist <= 4.5\ngini = 0.473\nsamples = 10711\nvalue = [6604, 4107]'),
Text(0.48098159509202454, 0.35, 'qtTrade <= 5.5\ngini = 0.44\nsamples = 5741\nvalue = [3867, 1874]'),
Text(0.47607361963190187, 0.25, 'qtFirstKill <= 2.5\ngini = 0.41\nsamples = 4106\nvalue = [2926, 1180]'),
Text(0.4736196319018405, 0.15, 'qtClutchWon <= 0.5\ngini = 0.36\nsamples = 2045\nvalue = [1563, 482]'),
Text(0.4723926380368098, 0.05, '\n  (...)  \n'),
Text(0.47484662576687114, 0.05, '\n  (...)  \n'),
Text(0.4785276073619632, 0.15, 'qtDeath <= 19.5\ngini = 0.448\nsamples = 2061\nvalue = [1363, 698]'),
Text(0.47730061349693254, 0.05, '\n  (...)  \n'),
Text(0.47975460122699387, 0.05, '\n  (...)  \n'),
Text(0.48588957055214727, 0.25, 'qtDeath <= 20.5\ngini = 0.489\nsamples = 1635\nvalue = [941, 694]'),
Text(0.4834355828220859, 0.15, 'qtFirstKill <= 3.5\ngini = 0.491\nsamples = 322\nvalue = [139, 183]'),
Text(0.4822085889570552, 0.05, '\n  (...)  \n'),
Text(0.48466257668711654, 0.05, '\n  (...)  \n'),
Text(0.4883435582822086, 0.15, 'qtTrade <= 7.5\ngini = 0.475\nsamples = 1313\nvalue = [802, 511]'),
Text(0.48711656441717793, 0.05, '\n  (...)  \n'),
Text(0.48957055214723927, 0.05, '\n  (...)  \n'),
Text(0.5006134969325153, 0.35, 'qtDeath <= 20.5\ngini = 0.495\nsamples = 4970\nvalue = [2737, 2233]'),
Text(0.49570552147239266, 0.25, 'qtFirstKill <= 2.5\ngini = 0.483\nsamples = 984\nvalue = [401, 583]'),
Text(0.49325153374233127, 0.15, 'qtAssist <= 6.5\ngini = 0.499\nsamples = 472\nvalue = [244, 228]'),
Text(0.4920245398773006, 0.05, '\n  (...)  \n'),
Text(0.49447852760736194, 0.05, '\n  (...)  \n'),
Text(0.498159509202454, 0.15, 'qtFirstKill <= 4.5\ngini = 0.425\nsamples = 512\nvalue = [157, 355]'),
Text(0.49693251533742333, 0.05, '\n  (...)  \n'),
Text(0.49938650306748467, 0.05, '\n  (...)  \n'),
Text(0.505521472392638, 0.25, 'qtAssist <= 6.5\ngini = 0.485\nsamples = 3986\nvalue = [2336, 1650]'),
Text(0.5030674846625767, 0.15, 'qtTrade <= 4.5\ngini = 0.471\nsamples = 2398\nvalue = [1486, 912]'),
Text(0.501840490797546, 0.05, '\n  (...)  \n'),
Text(0.5042944785276073, 0.05, '\n  (...)  \n'),
Text(0.5079754601226993, 0.15, 'qtDeath <= 22.5\ngini = 0.498\nsamples = 1588\nvalue = [850, 738]'),
Text(0.5067484662576687, 0.05, '\n  (...)  \n'),
Text(0.50920245398773, 0.05, '\n  (...)  \n'),
Text(0.7309049079754601, 0.85, 'qtDeath <= 17.5\ngini = 0.412\nsamples = 83102\nvalue = [24118, 58984]'),
Text(0.5995015337423313, 0.75, 'qtDeath <= 15.5\ngini = 0.145\nsamples = 36968\nvalue = [2901, 34067]'),
```

```
Text(0.5440184049079755, 0.65, 'qtLastAlive <= 5.5\ngini = 0.061\nsamples = 24879\nvalue = [780, 24099]'),
Text(0.5214723926380368, 0.55, 'qtShots <= 89.5\ngini = 0.048\nsamples = 24585\nvalue = [606, 23979]'),
Text(0.5128834355828221, 0.45, 'qt1Kill <= 4.5\ngini = 0.278\nsamples = 24\nvalue = [20, 4]'),
Text(0.5116564417177915, 0.35, 'qtRoundsPlayed <= 11.0\ngini = 0.091\nsamples = 21\nvalue = [20, 1]'),
Text(0.5104294478527608, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5128834355828221, 0.25, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
Text(0.5141104294478528, 0.35, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.5300613496932516, 0.45, 'qtLastAlive <= 3.5\ngini = 0.047\nsamples = 24561\nvalue = [586, 23975]'),
Text(0.5202453987730061, 0.35, 'qtDeath <= 14.5\ngini = 0.042\nsamples = 24223\nvalue = [521, 23702]'),
Text(0.5153374233128835, 0.25, 'qtShots <= 129.5\ngini = 0.029\nsamples = 19514\nvalue = [285, 19229]'),
Text(0.5128834355828221, 0.15, 'vlDamage <= 938.0\ngini = 0.432\nsamples = 38\nvalue = [12, 26]'),
Text(0.5116564417177915, 0.05, '\n  (...)  \n'),
Text(0.5141104294478528, 0.05, '\n  (...)  \n'),
Text(0.5177914110429448, 0.15, 'qtRoundsPlayed <= 28.5\ngini = 0.028\nsamples = 19476\nvalue = [273, 19203]'),
Text(0.5165644171779141, 0.05, '\n  (...)  \n'),
Text(0.5190184049079755, 0.05, '\n  (...)  \n'),
Text(0.5251533742331288, 0.25, 'qtTrade <= 0.5\ngini = 0.095\nsamples = 4709\nvalue = [236, 4473]'),
Text(0.5226993865030675, 0.15, 'qtAssist <= 2.5\ngini = 0.284\nsamples = 268\nvalue = [46, 222]'),
Text(0.5214723926380368, 0.05, '\n  (...)  \n'),
Text(0.5239263803680981, 0.05, '\n  (...)  \n'),
Text(0.5276073619631901, 0.15, 'qtRoundsPlayed <= 20.5\ngini = 0.082\nsamples = 4441\nvalue = [190, 4251]'),
Text(0.5263803680981595, 0.05, '\n  (...)  \n'),
Text(0.5288343558282208, 0.05, '\n  (...)  \n'),
Text(0.5398773006134969, 0.35, 'qtHits <= 39.5\ngini = 0.311\nsamples = 338\nvalue = [65, 273]'),
Text(0.5349693251533743, 0.25, 'qtRoundsPlayed <= 25.5\ngini = 0.457\nsamples = 34\nvalue = [22, 12]'),
Text(0.5325153374233129, 0.15, 'qtDeath <= 8.5\ngini = 0.459\nsamples = 14\nvalue = [5, 9]'),
Text(0.5312883435582823, 0.05, '\n  (...)  \n'),
Text(0.5337423312883436, 0.05, '\n  (...)  \n'),
Text(0.5374233128834356, 0.15, 'vlDamage <= 2471.0\ngini = 0.255\nsamples = 20\nvalue = [17, 3]'),
Text(0.5361963190184049, 0.05, '\n  (...)  \n'),
Text(0.5386503067484663, 0.05, '\n  (...)  \n'),
Text(0.5447852760736196, 0.25, 'qtClutchWon <= 2.5\ngini = 0.243\nsamples = 304\nvalue = [43, 261]'),
Text(0.5423312883435583, 0.15, 'qtSurvived <= 9.5\ngini = 0.358\nsamples = 167\nvalue = [39, 128]'),
Text(0.5411042944785276, 0.05, '\n  (...)  \n'),
Text(0.5435582822085889, 0.05, '\n  (...)  \n'),
Text(0.5472392638036809, 0.15, 'qtHitLeftAtm <= 7.5\ngini = 0.057\nsamples = 137\nvalue = [4, 133]'),
Text(0.5460122699386503, 0.05, '\n  (...)  \n'),
Text(0.5484662576687117, 0.05, '\n  (...)  \n'),
Text(0.5665644171779141, 0.55, 'vlDamage <= 1787.5\ngini = 0.483\nsamples = 294\nvalue = [174, 120]'),
Text(0.5576687116564417, 0.45, 'qtTrade <= 2.5\ngini = 0.172\nsamples = 168\nvalue = [152, 16]'),
Text(0.554601226993865, 0.35, 'qtShots <= 807.0\ngini = 0.067\nsamples = 144\nvalue = [139, 5]'),
Text(0.5533742331288344, 0.25, 'qt1Kill <= 8.5\ngini = 0.054\nsamples = 143\nvalue = [139, 4]'),
Text(0.5521472392638037, 0.15, 'qtFlashAssist <= 1.5\ngini = 0.041\nsamples = 142\nvalue = [139, 3]'),
Text(0.550920245398773, 0.05, '\n  (...)  \n'),
Text(0.5533742331288344, 0.05, '\n  (...)  \n'),
Text(0.554601226993865, 0.15, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5558282208588957, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.5607361963190184, 0.35, 'qtRoundsPlayed <= 25.0\ngini = 0.497\nsamples = 24\nvalue = [13, 11]'),
Text(0.558282208588957, 0.25, 'qtSurvived <= 10.5\ngini = 0.298\nsamples = 11\nvalue = [2, 9]'),
```

```
Text(0.5570552147239264, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.5595092024539877, 0.15, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
Text(0.5631901840490797, 0.25, 'qt3Kill <= 0.5\ngini = 0.26\nsamples = 13\nvalue = [11, 2]'),
Text(0.561963190184049, 0.15, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]'),
Text(0.5644171779141104, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
Text(0.5754601226993865, 0.45, 'qtRoundsPlayed <= 29.5\ngini = 0.288\nsamples = 126\nvalue = [22, 104]'),
Text(0.5717791411042945, 0.35, 'qtSurvived <= 10.5\ngini = 0.222\nsamples = 118\nvalue = [15, 103]'),
Text(0.5680981595092025, 0.25, 'qtClutchWon <= 0.5\ngini = 0.485\nsamples = 29\nvalue = [12, 17]'),
Text(0.5668711656441717, 0.15, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
Text(0.5693251533742332, 0.15, 'qt2Kill <= 4.5\ngini = 0.386\nsamples = 23\nvalue = [6, 17]'),
Text(0.5680981595092025, 0.05, '\n  (...)  \n'),
Text(0.5705521472392638, 0.05, '\n  (...)  \n'),
Text(0.5754601226993865, 0.25, 'qtLastAlive <= 12.5\ngini = 0.065\nsamples = 89\nvalue = [3, 86]'),
Text(0.5742331288343558, 0.15, 'qtHits <= 100.0\ngini = 0.044\nsamples = 88\nvalue = [2, 86]'),
Text(0.5730061349693252, 0.05, '\n  (...)  \n'),
Text(0.5754601226993865, 0.05, '\n  (...)  \n'),
Text(0.5766871165644172, 0.15, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
Text(0.5791411042944785, 0.35, 'qtRoundsPlayed <= 35.5\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
Text(0.5779141104294478, 0.25, 'gini = 0.0\nsamples = 7\nvalue = [7, 0]'),
Text(0.5803680981595092, 0.25, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
Text(0.6549846625766871, 0.65, 'qtTrade <= 2.5\ngini = 0.289\nsamples = 12089\nvalue = [2121, 9968]'),
Text(0.6171779141104294, 0.55, 'qtSurvived <= 8.5\ngini = 0.386\nsamples = 5518\nvalue = [1443, 4075]'),
Text(0.5975460122699386, 0.45, 'qtAssist <= 3.5\ngini = 0.47\nsamples = 1982\nvalue = [749, 1233]'),
Text(0.5877300613496933, 0.35, 'qtFirstKill <= 1.5\ngini = 0.5\nsamples = 1102\nvalue = [555, 547]'),
Text(0.5828220858895705, 0.25, 'qtSurvived <= 7.5\ngini = 0.458\nsamples = 352\nvalue = [227, 125]'),
Text(0.5803680981595092, 0.15, 'qtHits <= 55.5\ngini = 0.388\nsamples = 186\nvalue = [137, 49]'),
Text(0.5791411042944785, 0.05, '\n  (...)  \n'),
Text(0.5815950920245399, 0.05, '\n  (...)  \n'),
Text(0.5852760736196319, 0.15, 'qtTrade <= 1.5\ngini = 0.496\nsamples = 166\nvalue = [90, 76]'),
Text(0.5840490797546012, 0.05, '\n  (...)  \n'),
Text(0.5865030674846625, 0.05, '\n  (...)  \n'),
Text(0.592638036809816, 0.25, 'qtTrade <= 0.5\ngini = 0.492\nsamples = 750\nvalue = [328, 422]'),
Text(0.5901840490797546, 0.15, 'qtAssist <= 1.5\ngini = 0.44\nsamples = 95\nvalue = [64, 31]'),
Text(0.588957055214724, 0.05, '\n  (...)  \n'),
Text(0.5914110429447853, 0.05, '\n  (...)  \n'),
Text(0.5950920245398773, 0.15, 'qtClutchWon <= 0.5\ngini = 0.481\nsamples = 655\nvalue = [264, 391]'),
Text(0.5938650306748466, 0.05, '\n  (...)  \n'),
Text(0.596319018404908, 0.05, '\n  (...)  \n'),
Text(0.6073619631901841, 0.35, 'qtFirstKill <= 2.5\ngini = 0.344\nsamples = 880\nvalue = [194, 686]'),
Text(0.6024539877300613, 0.25, 'qtClutchWon <= 0.5\ngini = 0.421\nsamples = 408\nvalue = [123, 285]'),
Text(0.6, 0.15, 'qtLastAlive <= 0.5\ngini = 0.477\nsamples = 214\nvalue = [84, 130]'),
Text(0.5987730061349693, 0.05, '\n  (...)  \n'),
Text(0.6012269938650306, 0.05, '\n  (...)  \n'),
Text(0.6049079754601226, 0.15, 'qtAssist <= 4.5\ngini = 0.321\nsamples = 194\nvalue = [39, 155]'),
Text(0.603680981595092, 0.05, '\n  (...)  \n'),
Text(0.6061349693251534, 0.05, '\n  (...)  \n'),
Text(0.6122699386503068, 0.25, 'qtDeath <= 16.5\ngini = 0.256\nsamples = 472\nvalue = [71, 401]'),
Text(0.6098159509202454, 0.15, 'qtShots <= 363.0\ngini = 0.135\nsamples = 193\nvalue = [14, 179]'),
Text(0.6085889570552148, 0.05, '\n  (...)  \n'),
```

```
Text(0.6110429447852761, 0.05, '\n  (...)  \n'),
Text(0.6147239263803681, 0.15, 'qtShots <= 798.0\ngini = 0.325\nsamples = 279\nvalue = [57, 222]'),
Text(0.6134969325153374, 0.05, '\n  (...)  \n'),
Text(0.6159509202453988, 0.05, '\n  (...)  \n'),
Text(0.6368098159509202, 0.45, 'qtLastAlive <= 1.5\ngini = 0.315\nsamples = 3536\nvalue = [694, 2842]'),
Text(0.6269938650306749, 0.35, 'qtDeath <= 16.5\ngini = 0.281\nsamples = 2867\nvalue = [484, 2383]'),
Text(0.6220858895705521, 0.25, 'vlDamage <= 1634.0\ngini = 0.202\nsamples = 1375\nvalue = [157, 1218]'),
Text(0.6196319018404908, 0.15, 'qtFirstKill <= 0.5\ngini = 0.46\nsamples = 53\nvalue = [19, 34]'),
Text(0.6184049079754601, 0.05, '\n  (...)  \n'),
Text(0.6208588957055214, 0.05, '\n  (...)  \n'),
Text(0.6245398773006134, 0.15, 'qtAssist <= 2.5\ngini = 0.187\nsamples = 1322\nvalue = [138, 1184]'),
Text(0.6233128834355828, 0.05, '\n  (...)  \n'),
Text(0.6257668711656442, 0.05, '\n  (...)  \n'),
Text(0.6319018404907976, 0.25, 'qtFirstKill <= 2.5\ngini = 0.342\nsamples = 1492\nvalue = [327, 1165]'),
Text(0.6294478527607362, 0.15, 'qtTrade <= 1.5\ngini = 0.402\nsamples = 707\nvalue = [197, 510]'),
Text(0.6282208588957056, 0.05, '\n  (...)  \n'),
Text(0.6306748466257669, 0.05, '\n  (...)  \n'),
Text(0.6343558282208589, 0.15, 'qtAssist <= 1.5\ngini = 0.276\nsamples = 785\nvalue = [130, 655]'),
Text(0.6331288343558282, 0.05, '\n  (...)  \n'),
Text(0.6355828220858896, 0.05, '\n  (...)  \n'),
Text(0.6466257668711657, 0.35, 'vlDamage <= 2225.5\ngini = 0.431\nsamples = 669\nvalue = [210, 459]'),
Text(0.6417177914110429, 0.25, 'qtHitRightLeg <= 0.5\ngini = 0.499\nsamples = 103\nvalue = [49, 54]'),
Text(0.6392638036809816, 0.15, 'qtHitRightArm <= 7.0\ngini = 0.291\nsamples = 17\nvalue = [14, 3]'),
Text(0.6380368098159509, 0.05, '\n  (...)  \n'),
Text(0.6404907975460122, 0.05, '\n  (...)  \n'),
Text(0.6441717791411042, 0.15, 'qtHitChest <= 27.5\ngini = 0.483\nsamples = 86\nvalue = [35, 51]'),
Text(0.6429447852760736, 0.05, '\n  (...)  \n'),
Text(0.645398773006135, 0.05, '\n  (...)  \n'),
Text(0.6515337423312884, 0.25, 'qtLastAlive <= 2.5\ngini = 0.407\nsamples = 566\nvalue = [161, 405]'),
Text(0.649079754601227, 0.15, 'qtClutchWon <= 0.5\ngini = 0.356\nsamples = 371\nvalue = [86, 285]'),
Text(0.6478527607361964, 0.05, '\n  (...)  \n'),
Text(0.6503067484662577, 0.05, '\n  (...)  \n'),
Text(0.6539877300613497, 0.15, 'qtSurvived <= 9.5\ngini = 0.473\nsamples = 195\nvalue = [75, 120]'),
Text(0.652760736196319, 0.05, '\n  (...)  \n'),
Text(0.6552147239263804, 0.05, '\n  (...)  \n'),
Text(0.6927914110429448, 0.55, 'qtLastAlive <= 1.5\ngini = 0.185\nsamples = 6571\nvalue = [678, 5893]'),
Text(0.6760736196319018, 0.45, 'qtDeath <= 16.5\ngini = 0.156\nsamples = 5484\nvalue = [467, 5017]'),
Text(0.6662576687116565, 0.35, 'qtAssist <= 1.5\ngini = 0.091\nsamples = 2474\nvalue = [118, 2356]'),
Text(0.6613496932515337, 0.25, 'qtTrade <= 3.5\ngini = 0.223\nsamples = 211\nvalue = [27, 184]'),
Text(0.6588957055214724, 0.15, 'de_vertigo <= 0.5\ngini = 0.327\nsamples = 97\nvalue = [20, 77]'),
Text(0.6576687116564417, 0.05, '\n  (...)  \n'),
Text(0.660122699386503, 0.05, '\n  (...)  \n'),
Text(0.6638036809815951, 0.15, 'qtRoundsPlayed <= 28.5\ngini = 0.115\nsamples = 114\nvalue = [7, 107]'),
Text(0.6625766871165644, 0.05, '\n  (...)  \n'),
Text(0.6650306748466258, 0.05, '\n  (...)  \n'),
Text(0.6711656441717792, 0.25, 'qtHitChest <= 17.5\ngini = 0.077\nsamples = 2263\nvalue = [91, 2172]'),
Text(0.6687116564417178, 0.15, 'vlDamage <= 2664.5\ngini = 0.2\nsamples = 177\nvalue = [20, 157]'),
Text(0.6674846625766871, 0.05, '\n  (...)  \n'),
Text(0.6699386503067485, 0.05, '\n  (...)  \n'),
```

```
Text(0.6736196319018405, 0.15, 'qtBombeDefuse <= 4.5\ngini = 0.066\nsamples = 2086\nvalue = [71, 2015]'),
Text(0.6723926380368098, 0.05, '\n  (...)  \n'),
Text(0.6748466257668712, 0.05, '\n  (...)  \n'),
Text(0.6858895705521473, 0.35, 'qtAssist <= 3.5\ngini = 0.205\nsamples = 3010\nvalue = [349, 2661]'),
Text(0.6809815950920245, 0.25, 'qtSurvived <= 7.5\ngini = 0.274\nsamples = 1318\nvalue = [216, 1102]'),
Text(0.6785276073619632, 0.15, 'qtTrade <= 3.5\ngini = 0.392\nsamples = 239\nvalue = [64, 175]'),
Text(0.6773006134969325, 0.05, '\n  (...)  \n'),
Text(0.6797546012269938, 0.05, '\n  (...)  \n'),
Text(0.6834355828220859, 0.15, 'de_inferno <= 0.5\ngini = 0.242\nsamples = 1079\nvalue = [152, 927]'),
Text(0.6822085889570552, 0.05, '\n  (...)  \n'),
Text(0.6846625766871166, 0.05, '\n  (...)  \n'),
Text(0.69079754601227, 0.25, 'qtFirstKill <= 1.5\ngini = 0.145\nsamples = 1692\nvalue = [133, 1559]'),
Text(0.6883435582822086, 0.15, 'qtTrade <= 4.5\ngini = 0.23\nsamples = 407\nvalue = [54, 353]'),
Text(0.6871165644171779, 0.05, '\n  (...)  \n'),
Text(0.6895705521472393, 0.05, '\n  (...)  \n'),
Text(0.6932515337423313, 0.15, 'qtTrade <= 3.5\ngini = 0.115\nsamples = 1285\nvalue = [79, 1206]'),
Text(0.6920245398773006, 0.05, '\n  (...)  \n'),
Text(0.694478527607362, 0.05, '\n  (...)  \n'),
Text(0.7095092024539877, 0.45, 'qtClutchWon <= 0.5\ngini = 0.313\nsamples = 1087\nvalue = [211, 876]'),
Text(0.7024539877300614, 0.35, 'qtSurvived <= 8.5\ngini = 0.446\nsamples = 164\nvalue = [55, 109]'),
Text(0.6993865030674846, 0.25, 'de_vertigo <= 0.5\ngini = 0.468\nsamples = 51\nvalue = [32, 19]'),
Text(0.698159509202454, 0.15, 'vlDamage <= 3266.0\ngini = 0.423\nsamples = 46\nvalue = [32, 14]'),
Text(0.6969325153374233, 0.05, '\n  (...)  \n'),
Text(0.6993865030674846, 0.05, '\n  (...)  \n'),
Text(0.7006134969325153, 0.15, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
Text(0.7055214723926381, 0.25, 'vlDamage <= 1616.5\ngini = 0.324\nsamples = 113\nvalue = [23, 90]'),
Text(0.7030674846625767, 0.15, 'qtHitLeftAtm <= 1.5\ngini = 0.32\nsamples = 5\nvalue = [4, 1]'),
Text(0.701840490797546, 0.05, '\n  (...)  \n'),
Text(0.7042944785276074, 0.05, '\n  (...)  \n'),
Text(0.7079754601226994, 0.15, 'qtHitLeftLeg <= 8.5\ngini = 0.29\nsamples = 108\nvalue = [19, 89]'),
Text(0.7067484662576687, 0.05, '\n  (...)  \n'),
Text(0.7092024539877301, 0.05, '\n  (...)  \n'),
Text(0.7165644171779141, 0.35, 'qtLastAlive <= 14.0\ngini = 0.281\nsamples = 923\nvalue = [156, 767]'),
Text(0.7153374233128834, 0.25, 'qtSurvived <= 7.5\ngini = 0.277\nsamples = 920\nvalue = [153, 767]'),
Text(0.7128834355828221, 0.15, 'qtAssist <= 4.5\ngini = 0.369\nsamples = 213\nvalue = [52, 161]'),
Text(0.7116564417177914, 0.05, '\n  (...)  \n'),
Text(0.7141104294478527, 0.05, '\n  (...)  \n'),
Text(0.7177914110429447, 0.15, 'qtDeath <= 16.5\ngini = 0.245\nsamples = 707\nvalue = [101, 606]'),
Text(0.7165644171779141, 0.05, '\n  (...)  \n'),
Text(0.7190184049079754, 0.05, '\n  (...)  \n'),
Text(0.7177914110429447, 0.25, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
Text(0.8623082822085889, 0.75, 'qtDeath <= 19.5\ngini = 0.497\nsamples = 46134\nvalue = [21217, 24917]'),
Text(0.7923312883435583, 0.65, 'qtTrade <= 2.5\ngini = 0.465\nsamples = 15302\nvalue = [5641, 9661]'),
Text(0.7585889570552147, 0.55, 'qtAssist <= 2.5\ngini = 0.499\nsamples = 6274\nvalue = [2973, 3301]'),
Text(0.7398773006134969, 0.45, 'qtSurvived <= 8.5\ngini = 0.477\nsamples = 2083\nvalue = [1263, 820]'),
Text(0.7300613496932515, 0.35, 'qtFirstKill <= 6.5\ngini = 0.406\nsamples = 1042\nvalue = [747, 295]'),
Text(0.7251533742331289, 0.25, 'qtRoundsPlayed <= 26.5\ngini = 0.395\nsamples = 1003\nvalue = [731, 272]'),
Text(0.7226993865030675, 0.15, 'vlLevel <= 6.5\ngini = 0.32\nsamples = 430\nvalue = [344, 86]'),
Text(0.7214723926380369, 0.05, '\n  (...)  \n'),
```

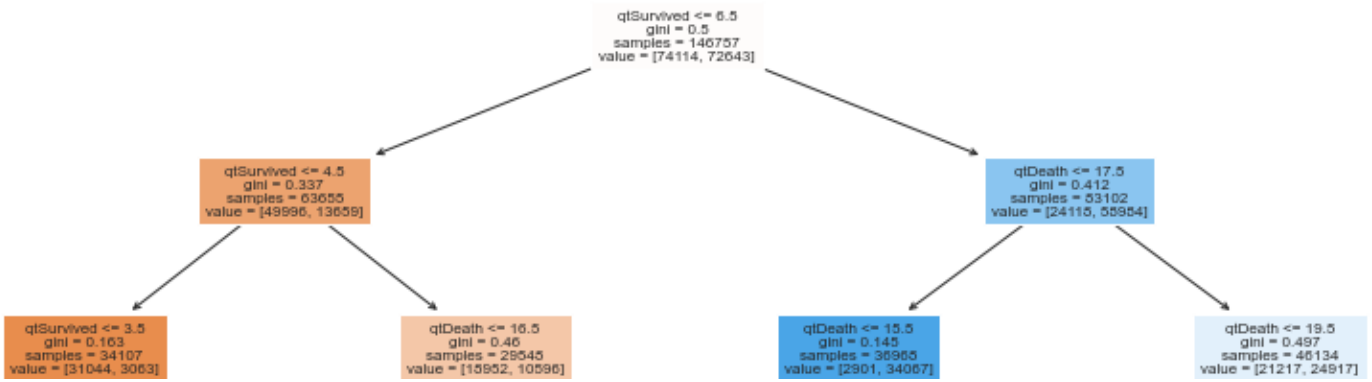
```
Text(0.7239263803680982, 0.05, '\n  (...)  \n'),
Text(0.7276073619631902, 0.15, 'qtClutchWon <= 1.5\ngini = 0.438\nsamples = 573\nvalue = [387, 186]'),
Text(0.7263803680981595, 0.05, '\n  (...)  \n'),
Text(0.7288343558282209, 0.05, '\n  (...)  \n'),
Text(0.7349693251533742, 0.25, 'qtLastAlive <= 1.5\ngini = 0.484\nsamples = 39\nvalue = [16, 23]'),
Text(0.7325153374233129, 0.15, 'qtHitRightArm <= 3.5\ngini = 0.412\nsamples = 31\nvalue = [9, 22]'),
Text(0.7312883435582822, 0.05, '\n  (...)  \n'),
Text(0.7337423312883435, 0.05, '\n  (...)  \n'),
Text(0.7374233128834355, 0.15, 'qtHitChest <= 53.0\ngini = 0.219\nsamples = 8\nvalue = [7, 1]'),
Text(0.7361963190184049, 0.05, '\n  (...)  \n'),
Text(0.7386503067484662, 0.05, '\n  (...)  \n'),
Text(0.7496932515337423, 0.35, 'qtFirstKill <= 2.5\ngini = 0.5\nsamples = 1041\nvalue = [516, 525]'),
Text(0.7447852760736197, 0.25, 'qtShots <= 302.5\ngini = 0.491\nsamples = 523\nvalue = [296, 227]'),
Text(0.7423312883435583, 0.15, 'qtHitStomach <= 20.5\ngini = 0.315\nsamples = 46\nvalue = [37, 9]'),
Text(0.7411042944785277, 0.05, '\n  (...)  \n'),
Text(0.743558282208589, 0.05, '\n  (...)  \n'),
Text(0.747239263803681, 0.15, 'qtHitLeftAtm <= 5.5\ngini = 0.496\nsamples = 477\nvalue = [259, 218]'),
Text(0.7460122699386503, 0.05, '\n  (...)  \n'),
Text(0.7484662576687117, 0.05, '\n  (...)  \n'),
Text(0.754601226993865, 0.25, 'qtRoundsPlayed <= 32.0\ngini = 0.489\nsamples = 518\nvalue = [220, 298]'),
Text(0.7521472392638037, 0.15, 'qtShots <= 387.5\ngini = 0.492\nsamples = 499\nvalue = [218, 281]'),
Text(0.750920245398773, 0.05, '\n  (...)  \n'),
Text(0.7533742331288343, 0.05, '\n  (...)  \n'),
Text(0.7570552147239263, 0.15, 'qtHitLeftAtm <= 4.5\ngini = 0.188\nsamples = 19\nvalue = [2, 17]'),
Text(0.7558282208588957, 0.05, '\n  (...)  \n'),
Text(0.758282208588957, 0.05, '\n  (...)  \n'),
Text(0.7773006134969325, 0.45, 'qtSurvived <= 8.5\ngini = 0.483\nsamples = 4191\nvalue = [1710, 2481]'),
Text(0.7693251533742331, 0.35, 'qtFirstKill <= 3.5\ngini = 0.5\nsamples = 1859\nvalue = [907, 952]'),
Text(0.7644171779141105, 0.25, 'qtAssist <= 5.5\ngini = 0.496\nsamples = 1271\nvalue = [691, 580]'),
Text(0.7619631901840491, 0.15, 'qtTrade <= 1.5\ngini = 0.488\nsamples = 1016\nvalue = [587, 429]'),
Text(0.7607361963190185, 0.05, '\n  (...)  \n'),
Text(0.7631901840490798, 0.05, '\n  (...)  \n'),
Text(0.7668711656441718, 0.15, 'qtHitRightLeg <= 0.5\ngini = 0.483\nsamples = 255\nvalue = [104, 151]'),
Text(0.7656441717791411, 0.05, '\n  (...)  \n'),
Text(0.7680981595092025, 0.05, '\n  (...)  \n'),
Text(0.7742331288343558, 0.25, 'qtAssist <= 3.5\ngini = 0.465\nsamples = 588\nvalue = [216, 372]'),
Text(0.7717791411042945, 0.15, 'qtHitChest <= 55.5\ngini = 0.5\nsamples = 165\nvalue = [80, 85]'),
Text(0.7705521472392638, 0.05, '\n  (...)  \n'),
Text(0.7730061349693251, 0.05, '\n  (...)  \n'),
Text(0.7766871165644171, 0.15, 'qt1Kill <= 8.5\ngini = 0.436\nsamples = 423\nvalue = [136, 287]'),
Text(0.7754601226993865, 0.05, '\n  (...)  \n'),
Text(0.7779141104294478, 0.05, '\n  (...)  \n'),
Text(0.7852760736196319, 0.35, 'qtLastAlive <= 11.5\ngini = 0.452\nsamples = 2332\nvalue = [803, 1529]'),
Text(0.7840490797546013, 0.25, 'qtTrade <= 1.5\ngini = 0.448\nsamples = 2315\nvalue = [786, 1529]'),
Text(0.7815950920245399, 0.15, 'qtDeath <= 18.5\ngini = 0.479\nsamples = 967\nvalue = [385, 582]'),
Text(0.7803680981595092, 0.05, '\n  (...)  \n'),
Text(0.7828220858895706, 0.05, '\n  (...)  \n'),
Text(0.7865030674846626, 0.15, 'qtFirstKill <= 0.5\ngini = 0.418\nsamples = 1348\nvalue = [401, 947]'),
Text(0.7852760736196319, 0.05, '\n  (...)  \n'),
```

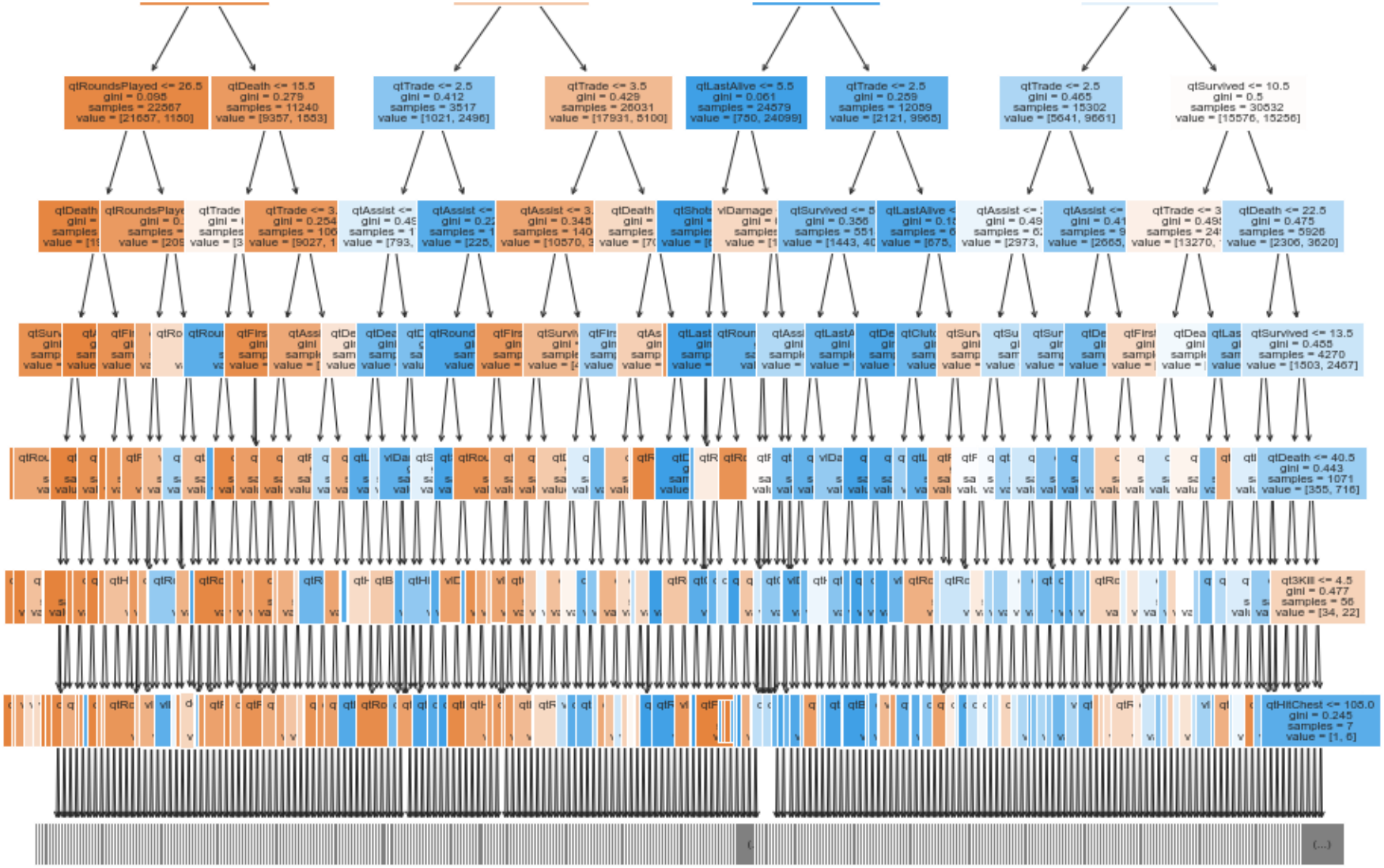
```
Text(0.7877300613496933, 0.05, '\n  (...)  \n'),
Text(0.7865030674846626, 0.25, 'gini = 0.0\nsamples = 17\nvalue = [17, 0]'),
Text(0.8260736196319018, 0.55, 'qtAssist <= 3.5\ngini = 0.416\nsamples = 9028\nvalue = [2668, 6360]'),
Text(0.8085889570552147, 0.45, 'qtSurvived <= 8.5\ngini = 0.461\nsamples = 3908\nvalue = [1410, 2498]'),
Text(0.7987730061349694, 0.35, 'qtTrade <= 4.5\ngini = 0.49\nsamples = 1946\nvalue = [837, 1109]'),
Text(0.7938650306748466, 0.25, 'qtFirstKill <= 1.5\ngini = 0.499\nsamples = 1395\nvalue = [672, 723]'),
Text(0.7914110429447853, 0.15, 'qtClutchWon <= 1.5\ngini = 0.479\nsamples = 387\nvalue = [233, 154]'),
Text(0.7901840490797546, 0.05, '\n  (...)  \n'),
Text(0.7926380368098159, 0.05, '\n  (...)  \n'),
Text(0.7963190184049079, 0.15, 'qtTrade <= 3.5\ngini = 0.492\nsamples = 1008\nvalue = [439, 569]'),
Text(0.7950920245398773, 0.05, '\n  (...)  \n'),
Text(0.7975460122699386, 0.05, '\n  (...)  \n'),
Text(0.803680981595092, 0.25, 'qtFirstKill <= 3.5\ngini = 0.42\nsamples = 551\nvalue = [165, 386]'),
Text(0.8012269938650307, 0.15, 'qtDeath <= 18.5\ngini = 0.467\nsamples = 377\nvalue = [140, 237]'),
Text(0.8, 0.05, '\n  (...)  \n'),
Text(0.8024539877300614, 0.05, '\n  (...)  \n'),
Text(0.8061349693251534, 0.15, 'qtHits <= 60.5\ngini = 0.246\nsamples = 174\nvalue = [25, 149]'),
Text(0.8049079754601227, 0.05, '\n  (...)  \n'),
Text(0.807361963190184, 0.05, '\n  (...)  \n'),
Text(0.8184049079754602, 0.35, 'qtTrade <= 4.5\ngini = 0.414\nsamples = 1962\nvalue = [573, 1389]'),
Text(0.8134969325153374, 0.25, 'qtLastAlive <= 0.5\ngini = 0.434\nsamples = 1419\nvalue = [452, 967]'),
Text(0.811042944785276, 0.15, 'qtDeath <= 18.5\ngini = 0.401\nsamples = 760\nvalue = [211, 549]'),
Text(0.8098159509202454, 0.05, '\n  (...)  \n'),
Text(0.8122699386503067, 0.05, '\n  (...)  \n'),
Text(0.8159509202453987, 0.15, 'qt1Kill <= 6.5\ngini = 0.464\nsamples = 659\nvalue = [241, 418]'),
Text(0.8147239263803681, 0.05, '\n  (...)  \n'),
Text(0.8171779141104294, 0.05, '\n  (...)  \n'),
Text(0.8233128834355828, 0.25, 'qtHitHeadshot <= 7.5\ngini = 0.346\nsamples = 543\nvalue = [121, 422]'),
Text(0.8208588957055215, 0.15, 'qtDeath <= 18.5\ngini = 0.432\nsamples = 187\nvalue = [59, 128]'),
Text(0.8196319018404908, 0.05, '\n  (...)  \n'),
Text(0.8220858895705522, 0.05, '\n  (...)  \n'),
Text(0.8257668711656442, 0.15, 'qt1Kill <= 9.5\ngini = 0.288\nsamples = 356\nvalue = [62, 294]'),
Text(0.8245398773006135, 0.05, '\n  (...)  \n'),
Text(0.8269938650306748, 0.05, '\n  (...)  \n'),
Text(0.843558282208589, 0.45, 'qtDeath <= 18.5\ngini = 0.371\nsamples = 5120\nvalue = [1258, 3862]'),
Text(0.8349693251533742, 0.35, 'qtFirstKill <= 3.5\ngini = 0.301\nsamples = 2432\nvalue = [449, 1983]'),
Text(0.8319018404907975, 0.25, 'qtLastAlive <= 12.5\ngini = 0.336\nsamples = 1619\nvalue = [346, 1273]'),
Text(0.8306748466257668, 0.15, 'qtTrade <= 4.5\ngini = 0.333\nsamples = 1614\nvalue = [341, 1273]'),
Text(0.8294478527607362, 0.05, '\n  (...)  \n'),
Text(0.8319018404907975, 0.05, '\n  (...)  \n'),
Text(0.8331288343558282, 0.15, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
Text(0.838036809815951, 0.25, 'qtHits <= 50.5\ngini = 0.221\nsamples = 813\nvalue = [103, 710]'),
Text(0.8355828220858895, 0.15, 'qtAssist <= 6.5\ngini = 0.463\nsamples = 33\nvalue = [12, 21]'),
Text(0.8343558282208589, 0.05, '\n  (...)  \n'),
Text(0.8368098159509203, 0.05, '\n  (...)  \n'),
Text(0.8404907975460123, 0.15, 'qtLastAlive <= 0.5\ngini = 0.206\nsamples = 780\nvalue = [91, 689]'),
Text(0.8392638036809816, 0.05, '\n  (...)  \n'),
Text(0.841717791411043, 0.05, '\n  (...)  \n'),
Text(0.8521472392638036, 0.35, 'qtAssist <= 6.5\ngini = 0.421\nsamples = 2688\nvalue = [809, 1879]'),
```

```
Text(0.8478527607361963, 0.25, 'qtTrade <= 4.5\ngini = 0.442\nsamples = 2110\nvalue = [696, 1414]'),
Text(0.845398773006135, 0.15, 'vlDamage <= 2271.5\ngini = 0.466\nsamples = 1410\nvalue = [521, 889]'),
Text(0.8441717791411043, 0.05, '\n  (...)  \n'),
Text(0.8466257668711656, 0.05, '\n  (...)  \n'),
Text(0.8503067484662576, 0.15, 'qtLastAlive <= 1.5\ngini = 0.375\nsamples = 700\nvalue = [175, 525]'),
Text(0.849079754601227, 0.05, '\n  (...)  \n'),
Text(0.8515337423312883, 0.05, '\n  (...)  \n'),
Text(0.8564417177914111, 0.25, 'qtLastAlive <= 11.5\ngini = 0.315\nsamples = 578\nvalue = [113, 465]'),
Text(0.8552147239263803, 0.15, 'qtHitRightArm <= 6.5\ngini = 0.311\nsamples = 576\nvalue = [111, 465]'),
Text(0.8539877300613496, 0.05, '\n  (...)  \n'),
Text(0.8564417177914111, 0.05, '\n  (...)  \n'),
Text(0.8576687116564418, 0.15, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
Text(0.9322852760736197, 0.65, 'qtSurvived <= 10.5\ngini = 0.5\nsamples = 30832\nvalue = [15576, 15256]'),
Text(0.8969325153374234, 0.55, 'qtTrade <= 3.5\ngini = 0.498\nsamples = 24906\nvalue = [13270, 11636]'),
Text(0.8773006134969326, 0.45, 'qtFirstKill <= 3.5\ngini = 0.484\nsamples = 11694\nvalue = [6880, 4814]'),
Text(0.8674846625766871, 0.35, 'qtAssist <= 2.5\ngini = 0.469\nsamples = 7205\nvalue = [4503, 2702]'),
Text(0.8625766871165644, 0.25, 'qtSurvived <= 8.5\ngini = 0.418\nsamples = 1751\nvalue = [1231, 520]'),
Text(0.8601226993865031, 0.15, 'qtHs <= 12.5\ngini = 0.385\nsamples = 1258\nvalue = [931, 327]'),
Text(0.8588957055214724, 0.05, '\n  (...)  \n'),
Text(0.8613496932515338, 0.05, '\n  (...)  \n'),
Text(0.8650306748466258, 0.15, 'qtLastAlive <= 4.5\ngini = 0.476\nsamples = 493\nvalue = [300, 193]'),
Text(0.8638036809815951, 0.05, '\n  (...)  \n'),
Text(0.8662576687116564, 0.05, '\n  (...)  \n'),
Text(0.8723926380368098, 0.25, 'qtRoundsPlayed <= 28.5\ngini = 0.48\nsamples = 5454\nvalue = [3272, 2182]'),
Text(0.8699386503067484, 0.15, 'qtDeath <= 20.5\ngini = 0.442\nsamples = 1048\nvalue = [703, 345]'),
Text(0.8687116564417178, 0.05, '\n  (...)  \n'),
Text(0.8711656441717791, 0.05, '\n  (...)  \n'),
Text(0.8748466257668711, 0.15, 'qtDeath <= 21.5\ngini = 0.486\nsamples = 4406\nvalue = [2569, 1837]'),
Text(0.8736196319018404, 0.05, '\n  (...)  \n'),
Text(0.8760736196319019, 0.05, '\n  (...)  \n'),
Text(0.8871165644171779, 0.35, 'qtDeath <= 20.5\ngini = 0.498\nsamples = 4489\nvalue = [2377, 2112]'),
Text(0.8822085889570552, 0.25, 'qtAssist <= 4.5\ngini = 0.496\nsamples = 1417\nvalue = [648, 769]'),
Text(0.8797546012269939, 0.15, 'qtTrade <= 1.5\ngini = 0.499\nsamples = 923\nvalue = [478, 445]'),
Text(0.8785276073619632, 0.05, '\n  (...)  \n'),
Text(0.8809815950920246, 0.05, '\n  (...)  \n'),
Text(0.8846625766871166, 0.15, 'qtHitChest <= 18.5\ngini = 0.451\nsamples = 494\nvalue = [170, 324]'),
Text(0.8834355828220859, 0.05, '\n  (...)  \n'),
Text(0.8858895705521472, 0.05, '\n  (...)  \n'),
Text(0.8920245398773006, 0.25, 'qtAssist <= 5.5\ngini = 0.492\nsamples = 3072\nvalue = [1729, 1343]'),
Text(0.8895705521472392, 0.15, 'qtRoundsPlayed <= 28.5\ngini = 0.484\nsamples = 2165\nvalue = [1279, 886]'),
Text(0.8883435582822086, 0.05, '\n  (...)  \n'),
Text(0.8907975460122699, 0.05, '\n  (...)  \n'),
Text(0.894478527607362, 0.15, 'qtDeath <= 38.5\ngini = 0.5\nsamples = 907\nvalue = [450, 457]'),
Text(0.8932515337423312, 0.05, '\n  (...)  \n'),
Text(0.8957055214723927, 0.05, '\n  (...)  \n'),
Text(0.9165644171779141, 0.45, 'qtDeath <= 21.5\ngini = 0.499\nsamples = 13212\nvalue = [6390, 6822]'),
Text(0.9067484662576687, 0.35, 'qtAssist <= 5.5\ngini = 0.489\nsamples = 5470\nvalue = [2336, 3134]'),
Text(0.901840490797546, 0.25, 'qtFirstKill <= 1.5\ngini = 0.497\nsamples = 4009\nvalue = [1839, 2170]'),
Text(0.8993865030674847, 0.15, 'qtHitRightArm <= 7.5\ngini = 0.496\nsamples = 1048\nvalue = [572, 476]'),
```

```
Text(0.898159509202454, 0.05, '\n  (...)  \n'),
Text(0.9006134969325154, 0.05, '\n  (...)  \n'),
Text(0.9042944785276074, 0.15, 'qtTrade <= 6.5\ngini = 0.49\nsamples = 2961\nvalue = [1267, 1694]'),
Text(0.9030674846625767, 0.05, '\n  (...)  \n'),
Text(0.905521472392638, 0.05, '\n  (...)  \n'),
Text(0.9116564417177914, 0.25, 'qtDeath <= 20.5\ngini = 0.449\nsamples = 1461\nvalue = [497, 964]'),
Text(0.90920245398773, 0.15, 'qtShots <= 356.0\ngini = 0.417\nsamples = 792\nvalue = [235, 557]'),
Text(0.9079754601226994, 0.05, '\n  (...)  \n'),
Text(0.9104294478527607, 0.05, '\n  (...)  \n'),
Text(0.9141104294478528, 0.15, 'qtTrade <= 6.5\ngini = 0.477\nsamples = 669\nvalue = [262, 407]'),
Text(0.912883435582822, 0.05, '\n  (...)  \n'),
Text(0.9153374233128835, 0.05, '\n  (...)  \n'),
Text(0.9263803680981595, 0.35, 'qtAssist <= 3.5\ngini = 0.499\nsamples = 7742\nvalue = [4054, 3688]'),
Text(0.9214723926380368, 0.25, 'qtTrade <= 5.5\ngini = 0.482\nsamples = 1950\nvalue = [1160, 790]'),
Text(0.9190184049079755, 0.15, 'qtDeath <= 25.5\ngini = 0.47\nsamples = 1303\nvalue = [812, 491]'),
Text(0.9177914110429448, 0.05, '\n  (...)  \n'),
Text(0.9202453987730062, 0.05, '\n  (...)  \n'),
Text(0.9239263803680982, 0.15, 'qtShots <= 311.0\ngini = 0.497\nsamples = 647\nvalue = [348, 299]'),
Text(0.9226993865030675, 0.05, '\n  (...)  \n'),
Text(0.9251533742331288, 0.05, '\n  (...)  \n'),
Text(0.9312883435582822, 0.25, 'qtDeath <= 26.5\ngini = 0.5\nsamples = 5792\nvalue = [2894, 2898]'),
Text(0.9288343558282208, 0.15, 'qtTrade <= 7.5\ngini = 0.497\nsamples = 3462\nvalue = [1588, 1874]'),
Text(0.9276073619631902, 0.05, '\n  (...)  \n'),
Text(0.9300613496932515, 0.05, '\n  (...)  \n'),
Text(0.9337423312883436, 0.15, 'qtTrade <= 6.5\ngini = 0.493\nsamples = 2330\nvalue = [1306, 1024]'),
Text(0.9325153374233128, 0.05, '\n  (...)  \n'),
Text(0.9349693251533743, 0.05, '\n  (...)  \n'),
Text(0.967638036809816, 0.55, 'qtDeath <= 22.5\ngini = 0.475\nsamples = 5926\nvalue = [2306, 3620]'),
Text(0.9527607361963191, 0.45, 'qtLastAlive <= 11.5\ngini = 0.423\nsamples = 1656\nvalue = [503, 1153]'),
Text(0.9460122699386503, 0.35, 'qtTrade <= 2.5\ngini = 0.403\nsamples = 1566\nvalue = [438, 1128]'),
Text(0.9411042944785276, 0.25, 'qtSurvived <= 13.5\ngini = 0.461\nsamples = 526\nvalue = [190, 336]'),
Text(0.9386503067484663, 0.15, 'qt2Kill <= 3.5\ngini = 0.484\nsamples = 386\nvalue = [158, 228]'),
Text(0.9374233128834356, 0.05, '\n  (...)  \n'),
Text(0.939877300613497, 0.05, '\n  (...)  \n'),
Text(0.943558282208589, 0.15, 'qtFirstKill <= 5.5\ngini = 0.353\nsamples = 140\nvalue = [32, 108]'),
Text(0.9423312883435583, 0.05, '\n  (...)  \n'),
Text(0.9447852760736196, 0.05, '\n  (...)  \n'),
Text(0.950920245398773, 0.25, 'qtHitHeadshot <= 8.5\ngini = 0.363\nsamples = 1040\nvalue = [248, 792]'),
Text(0.9484662576687116, 0.15, 'vlDamage <= 3974.0\ngini = 0.455\nsamples = 206\nvalue = [72, 134]'),
Text(0.947239263803681, 0.05, '\n  (...)  \n'),
Text(0.9496932515337423, 0.05, '\n  (...)  \n'),
Text(0.9533742331288344, 0.15, 'qtHits <= 67.5\ngini = 0.333\nsamples = 834\nvalue = [176, 658]'),
Text(0.9521472392638037, 0.05, '\n  (...)  \n'),
Text(0.9546012269938651, 0.05, '\n  (...)  \n'),
Text(0.9595092024539877, 0.35, 'qtSurvived <= 13.5\ngini = 0.401\nsamples = 90\nvalue = [65, 25]'),
Text(0.9570552147239264, 0.25, 'qtSurvived <= 12.5\ngini = 0.105\nsamples = 54\nvalue = [51, 3]'),
Text(0.9558282208588957, 0.15, 'gini = 0.0\nsamples = 43\nvalue = [43, 0]'),
Text(0.9582822085889571, 0.15, 'qtLastAlive <= 13.5\ngini = 0.397\nsamples = 11\nvalue = [8, 3]'),
Text(0.9570552147239264, 0.05, '\n  (...)  \n'),
```

```
Text(0.9595092024539877, 0.05, '\n  (...)  \n'),
Text(0.9619631901840491, 0.25, 'qtHitRightArm <= 2.5\ngini = 0.475\nsamples = 36\nvalue = [14, 22]'),
Text(0.9607361963190184, 0.15, 'gini = 0.0\nsamples = 8\nvalue = [0, 8]'),
Text(0.9631901840490797, 0.15, 'qtHits <= 86.5\ngini = 0.5\nsamples = 28\nvalue = [14, 14]'),
Text(0.9619631901840491, 0.05, '\n  (...)  \n'),
Text(0.9644171779141104, 0.05, '\n  (...)  \n'),
Text(0.9825153374233129, 0.45, 'qtSurvived <= 13.5\ngini = 0.488\nsamples = 4270\nvalue = [1803, 2467]'),
Text(0.9748466257668712, 0.35, 'qtLastAlive <= 8.5\ngini = 0.496\nsamples = 3199\nvalue = [1448, 1751]'),
Text(0.9705521472392638, 0.25, 'qtDeath <= 24.5\ngini = 0.495\nsamples = 3160\nvalue = [1415, 1745]'),
Text(0.9680981595092024, 0.15, 'qtTrade <= 4.5\ngini = 0.48\nsamples = 1242\nvalue = [498, 744]'),
Text(0.9668711656441717, 0.05, '\n  (...)  \n'),
Text(0.9693251533742331, 0.05, '\n  (...)  \n'),
Text(0.9730061349693252, 0.15, 'qtAssist <= 4.5\ngini = 0.499\nsamples = 1918\nvalue = [917, 1001]'),
Text(0.9717791411042945, 0.05, '\n  (...)  \n'),
Text(0.9742331288343559, 0.05, '\n  (...)  \n'),
Text(0.9791411042944785, 0.25, 'qt2Kill <= 7.5\ngini = 0.26\nsamples = 39\nvalue = [33, 6]'),
Text(0.9779141104294479, 0.15, 'qtShots <= 671.5\ngini = 0.153\nsamples = 36\nvalue = [33, 3]'),
Text(0.9766871165644172, 0.05, '\n  (...)  \n'),
Text(0.9791411042944785, 0.05, '\n  (...)  \n'),
Text(0.9803680981595092, 0.15, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
Text(0.9901840490797545, 0.35, 'qtDeath <= 40.5\ngini = 0.443\nsamples = 1071\nvalue = [355, 716]'),
Text(0.9852760736196319, 0.25, 'qtHits <= 78.5\ngini = 0.432\nsamples = 1015\nvalue = [321, 694]'),
Text(0.9828220858895705, 0.15, 'qtDeath <= 27.5\ngini = 0.491\nsamples = 173\nvalue = [75, 98]'),
Text(0.9815950920245399, 0.05, '\n  (...)  \n'),
Text(0.9840490797546012, 0.05, '\n  (...)  \n'),
Text(0.9877300613496932, 0.15, 'qtDeath <= 26.5\ngini = 0.414\nsamples = 842\nvalue = [246, 596]'),
Text(0.9865030674846625, 0.05, '\n  (...)  \n'),
Text(0.9889570552147239, 0.05, '\n  (...)  \n'),
Text(0.9950920245398773, 0.25, 'qt3Kill <= 4.5\ngini = 0.477\nsamples = 56\nvalue = [34, 22]'),
Text(0.992638036809816, 0.15, 'de_nuke <= 0.5\ngini = 0.44\nsamples = 49\nvalue = [33, 16]'),
Text(0.9914110429447853, 0.05, '\n  (...)  \n'),
Text(0.9938650306748467, 0.05, '\n  (...)  \n'),
Text(0.9975460122699387, 0.15, 'qtHitChest <= 108.0\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
Text(0.996319018404908, 0.05, '\n  (...)  \n'),
Text(0.9987730061349693, 0.05, '\n  (...)  \n')]
```





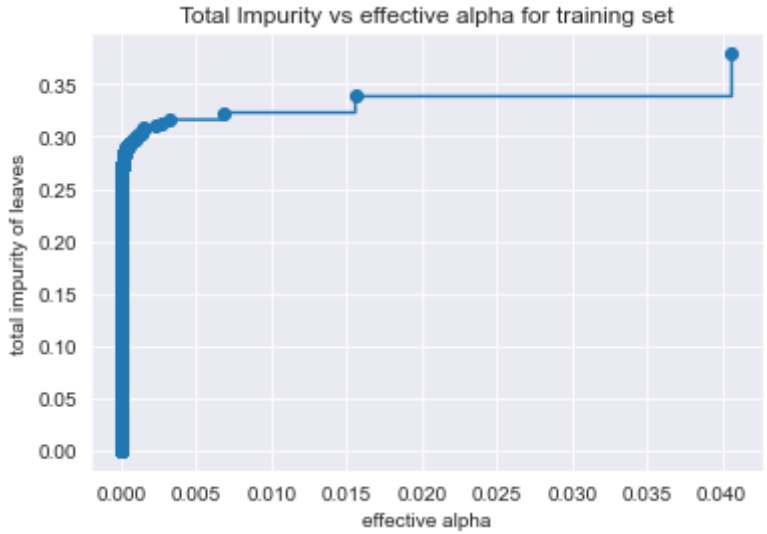
1.8: Prune the tree using one of the techniques discussed in class and evaluate the performance

There are many tree pruning techniques as discussed in the class I am gonna use the The DecisionTreeClassifier provides parameters such as min_samples_leaf and max_depth to prevent a tree from overfiting. Cost complexity pruning provides another option to control the size of a tree. In DecisionTreeClassifier, this pruning technique is parameterized by the cost complexity parameter, ccp_alpha. Greater values of ccp_alpha increase the number of nodes pruned. Here we only show the effect of ccp_alpha on regularizing the trees and how to choose a ccp_alpha based on validation scores.

Total impurity of leaves vs effective alphas of pruned tree will be discussed from here to get my alpha value for thge pruning

In [136…
```
path = clf.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
fig, ax = plt.subplots()
ax.plot(ccp_alphas[:-1], impurities[:-1], marker="o", drawstyle="steps-post")
ax.set_xlabel("effective alpha")
ax.set_ylabel("total impurity of leaves")
ax.set_title("Total Impurity vs effective alpha for training set")
```

Out[136… `Text(0.5, 1.0, 'Total Impurity vs effective alpha for training set')`



Note

To save time jsut taking last 10 values as seen on Ed Discussion

In [154…
```
ccp_alphas=ccp_alphas.tolist()[-12:-1]
```

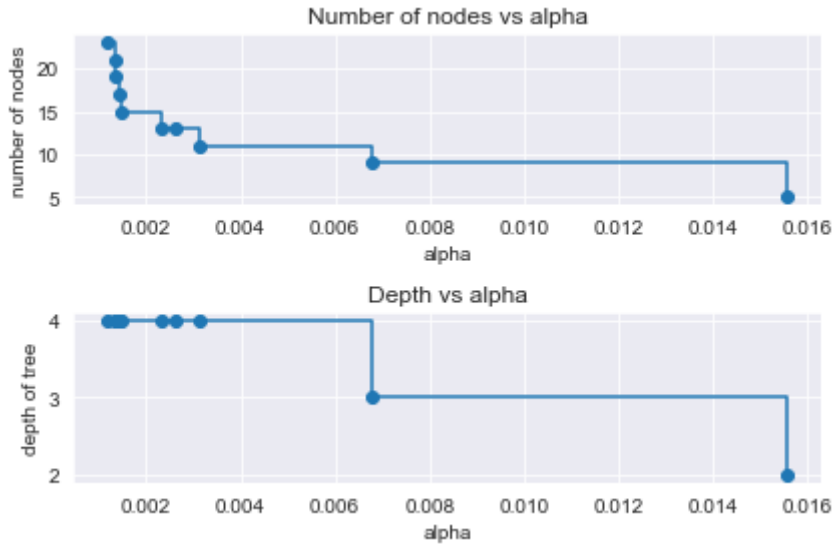
In [155…
```
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
print(
```

```
    "Number of nodes in the last tree is: {} with ccp_alpha: {}".format(
        clfs[-1].tree_.node_count, ccp_alphas[-1]
    )
)
```

```
Number of nodes in the last tree is: 3 with ccp_alpha: 0.04063466278924577
```

In [156…

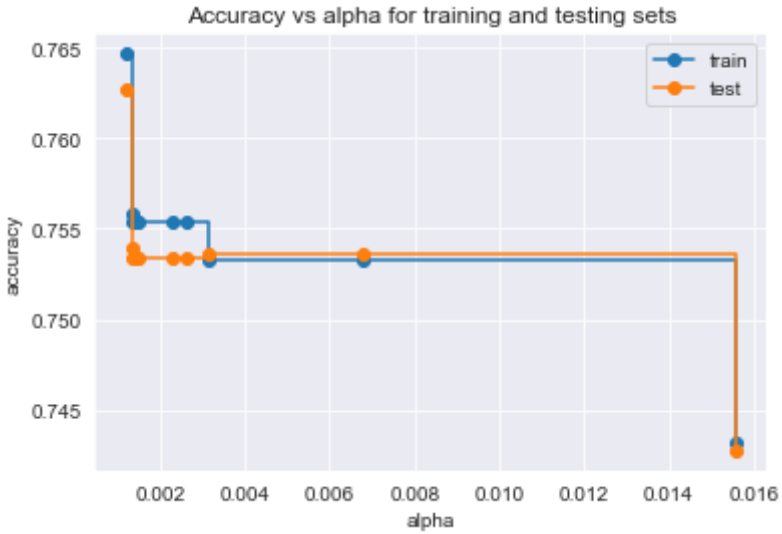
```
clfs = clfs[:-1]
ccp_alphas = ccp_alphas[:-1]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]
fig, ax = plt.subplots(2, 1)
ax[0].plot(ccp_alphas, node_counts, marker="o", drawstyle="steps-post")
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("number of nodes")
ax[0].set_title("Number of nodes vs alpha")
ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
fig.tight_layout()
```



In [157…

```
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]
```

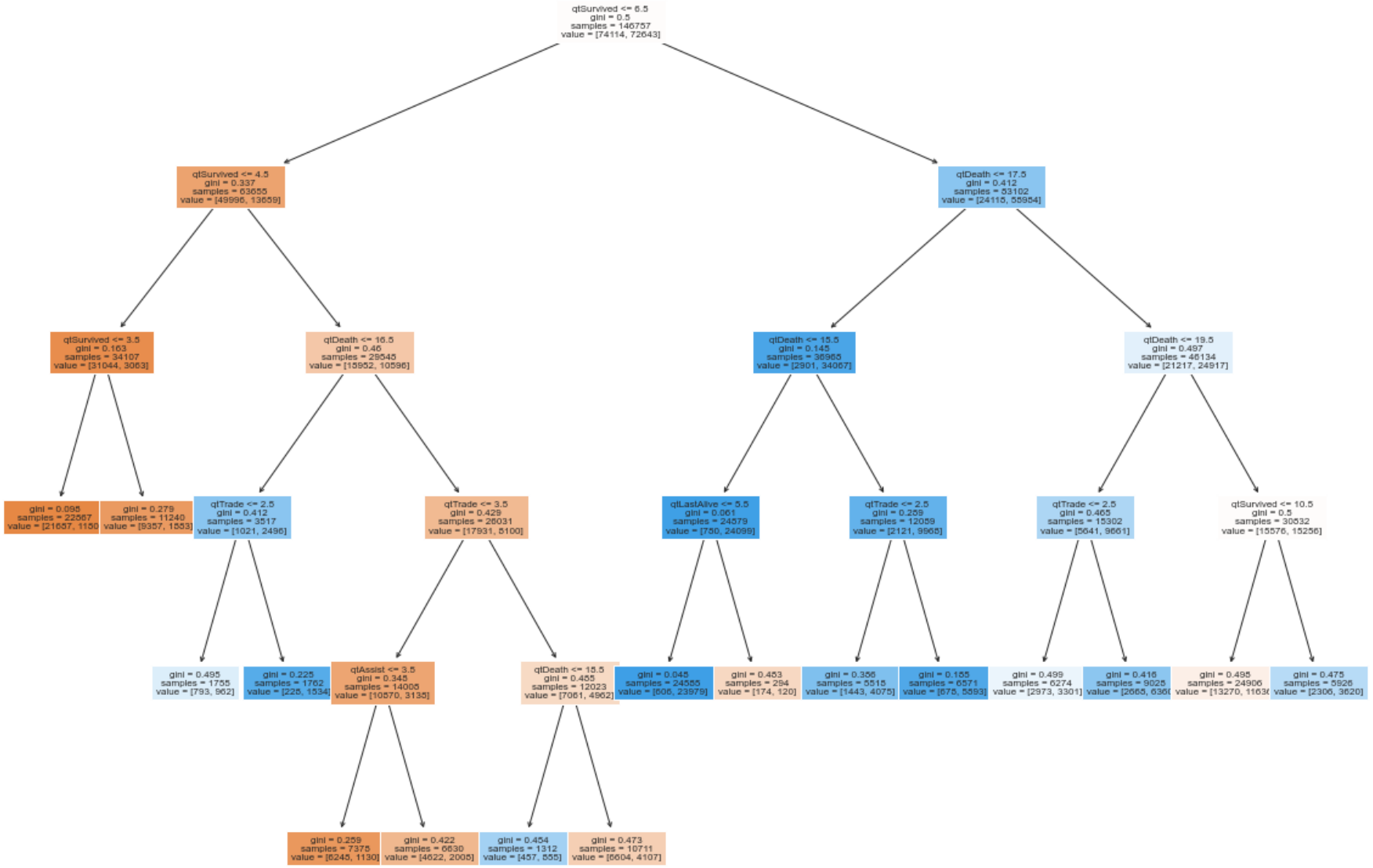
```
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker="o", label="train", drawstyle="steps-post")
ax.plot(ccp_alphas, test_scores, marker="o", label="test", drawstyle="steps-post")
ax.legend()
plt.show()
```



In [158…

```
clf_pruned = DecisionTreeClassifier(random_state = 42, ccp_alpha = 0.001)
pipe_pruned = make_pipeline(clf_pruned)
pipe_pruned.fit(X_dev, y_dev)
print("Score (Development data):", pipe_pruned.score(X_dev, y_dev))

plt.figure(figsize = (20, 15))
_ = plot_tree(clf_pruned, filled = True,fontsize = 8, feature_names = X_train.columns
              )
```

```
Score (Development data): 0.7668526884577908
```



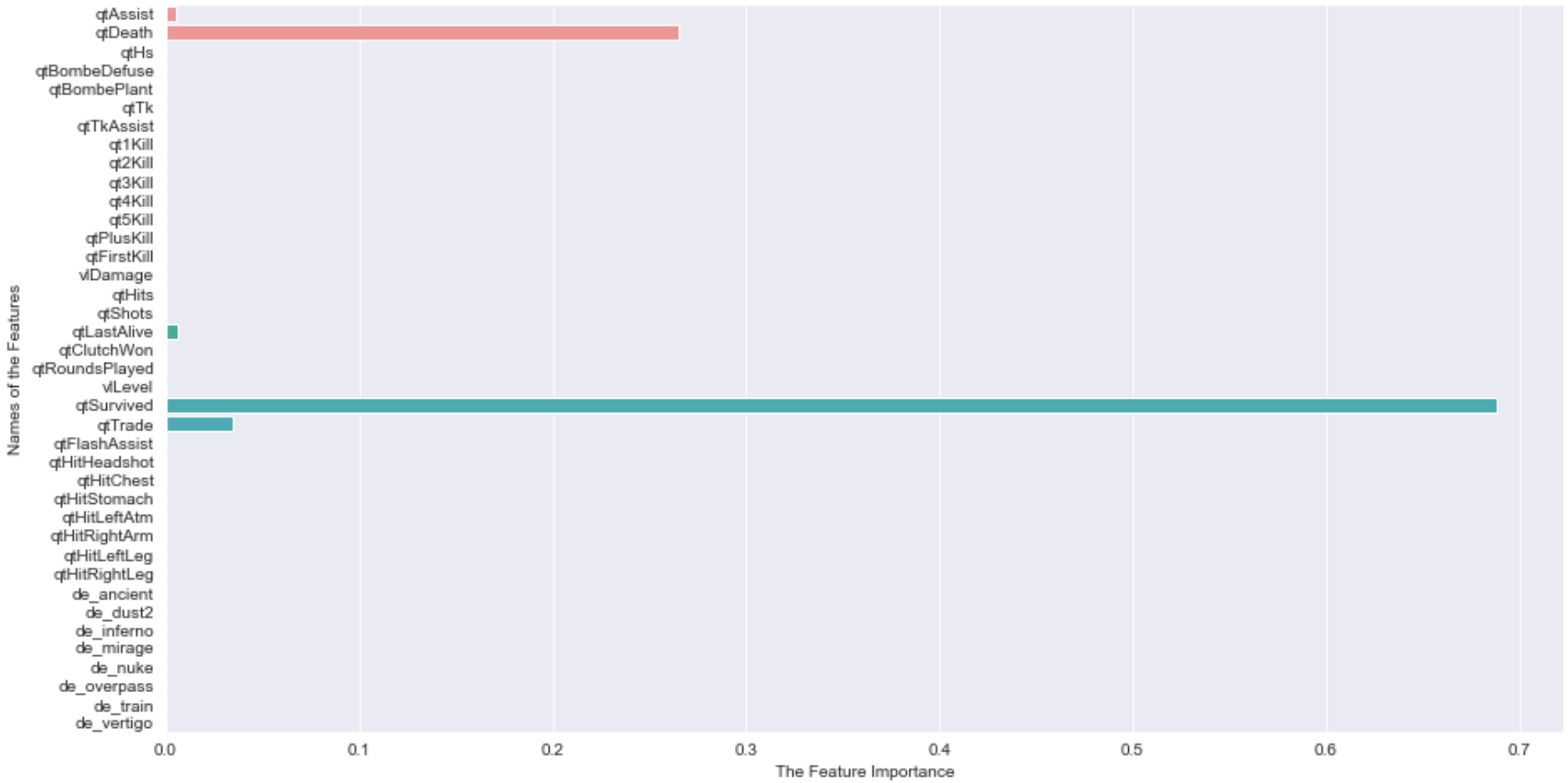
1.9: List the top 3 most important features for this trained tree? How would you justify these features being the most important?

In [159…

```
plt.figure(figsize = (15, 8))
```

```
sns.barplot(y = X_dev.columns.tolist(), x = clf_pruned.feature_importances_)

plt.ylabel('Names of the Features')
plt.xlabel('The Feature Importance')
```

Out[159… Text(0.5, 0, 'The Feature Importance')



Answer

The top 3 most important features are 1) qtSurvived

2) qtDeath

3) qtTrade

Top 3 most important features for the trained tree can be justified by the barchart and also the code that I have provided below that justifies the features importances. It is understandable by the nature of the data as well why did it choose these values due to this contribution to the Decision Tree Model

In [164…

```
indices = (-clf_pruned.feature_importances_).argsort()[:3]
#We have stored the top 3 features indices in the indices variable
X_dev.iloc[:,indices]
```

Out[164…

	qtSurvived	qtDeath	qtTrade
82481	8.0	23	4.0
4172	4.0	22	2.0
44676	6.0	20	2.0
151259	3.0	18	0.0
6933	6.0	21	4.0
...	...	...	...
120216	6.0	16	3.0
104018	14.0	5	1.0
132300	10.0	15	4.0
147344	7.0	19	1.0
122295	7.0	22	5.0

146757 rows × 3 columns

Question 2: Random Forests

2.1: Train a Random Forest model on the development dataset using RandomForestClassifier class in sklearn. Use the default parameters. Evaluate the performance of the model on test dataset. Does this perform better than Decision Tree on the test dataset (compare to results in Q 1.6)?

In [165…

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
rf = RandomForestClassifier()
pipe_rf = make_pipeline( rf)
pipe_rf.fit(X_dev, y_dev)
print('Score (Development data): {:.3f}'.format(pipe_rf.score(X_dev, y_dev)))
print('Score (Test data): ', pipe_rf.score(X_test, y_test))
```

```
Score (Development data): 1.000
Score (Test data):  0.7886345053147997
```

We have used the default parameters abd evaluated the performances

Yes this perform better than the Decision Tree on test data as

```
The score of test data of decision tree = 0.7231
```

```
The score of test data of random forests = 0.7886
```

So we may clearly say that the random forests perform better than the decision tree for serving our purpose

2.2: Does all trees in the trained random forest model have pure leaves? How would you verify this?

In [166…

```
training_score = pipe_rf.score(X_dev, y_dev)
print('Training score of our model is: ', training_score)
```

```
Training score of our model is:  1.0
```

Answer As observed from the fact that the score for the pipe of random forest on its learning is giving us a rate of 1.0. This might infer us that the training score is high enough to give us the confidence that all trees and their leaves in them in the training random forest model have pure leaves. This might not be enough to state that but for our purpose of learning and comprehending as taught by Professor in Lecture we might say yes the trees leaves are pure if the training score is perfect 1.0.

2.3: Assume you want to improve the performance of this model. Also, assume that you had to pick two hyperparameters that you could tune to improve its performance. Which hyperparameters would you choose and why?

Answer: If I want to improve the performance of this model, the hyperparameters that I choose to hypertune are:

1) n_estimators

2) max_depth

Reason:

The `n_estimators` is chosen as it is proprtional to the performance of the model or maybe be higher the number of estimators highers would be accuracy performance, but it is quite enough to say that it will definitely makes code slower.

The `max_depth` is chosen as it would allow me to constraint the depth to which I want each tree in the random forest to grow unless the best performance of the model might be achieved just before the overfitting of model over the data

2.4: Now, assume you had to choose up to 5 different values (each) for these two hyperparameters. How would you choose these values that could potentially give you a performance lift?

Answer I have assumed that I have to choose up to 5 different values for these two hyperparameters. The way I might choose these values that could potentially give me a performance lift is none othe r the Cross Validation that I would perform using the Grid Search. We are well known to GridSearch Randomality and it would perfectly define the range for n_estimators and max_depth that I want to hypertune in my model to get the performance upliftment

2.5: Perform model selection using the chosen values for the hyperparameters. Use cross-validation for finding the optimal hyperparameters. Report on the optimal hyperparameters. Estimate the performance of the optimal model (model trained with optimal hyperparameters) on test dataset? Has the performance improved over your plain-vanilla random forest model trained in Q2.1?

In [169…

```
#Keeping these parameters to understand and run the code a little quicker
n_estimators = [100,150,200]
max_depth = [10,20,30,40]

grid = {'randomforestclassifier__n_estimators': n_estimators,
       'randomforestclassifier__max_depth': max_depth}
pipe_rf_search = make_pipeline(rf)
gs = GridSearchCV(estimator = pipe_rf_search, param_grid = grid, cv = 5)
gs_results = gs.fit(X_dev, y_dev)

best_params = gs_results.best_params_
print('max_depth: ', best_params['randomforestclassifier__max_depth'])
print('n_estimators: ', best_params['randomforestclassifier__n_estimators'])
print('Score (development data): ',gs.score(X_dev, y_dev))
print('Score (test data): ',gs.score(X_test, y_test))
```

```
max_depth:  20
n_estimators:  150
Score (development data):  0.9878370367341932
Score (test data):  0.7907604251839738
```

***Answer**

I have used the cross-validation for finding the optimal hyperparameters. The above cell reports on the optimal hyperparameters. The hyperparameters have been estimated for the performance of the optimal model as following :

- max_depth: 20
- n_estimators: 150
- Score (development data): 0.9878370367341932
- Score (test data): 0.7907604251839738

The score here is 0.79076 which is higher than the 0.7886. I might say that the performance has significantly improved over the plain_vanilla random forest model trained in Q2.1.

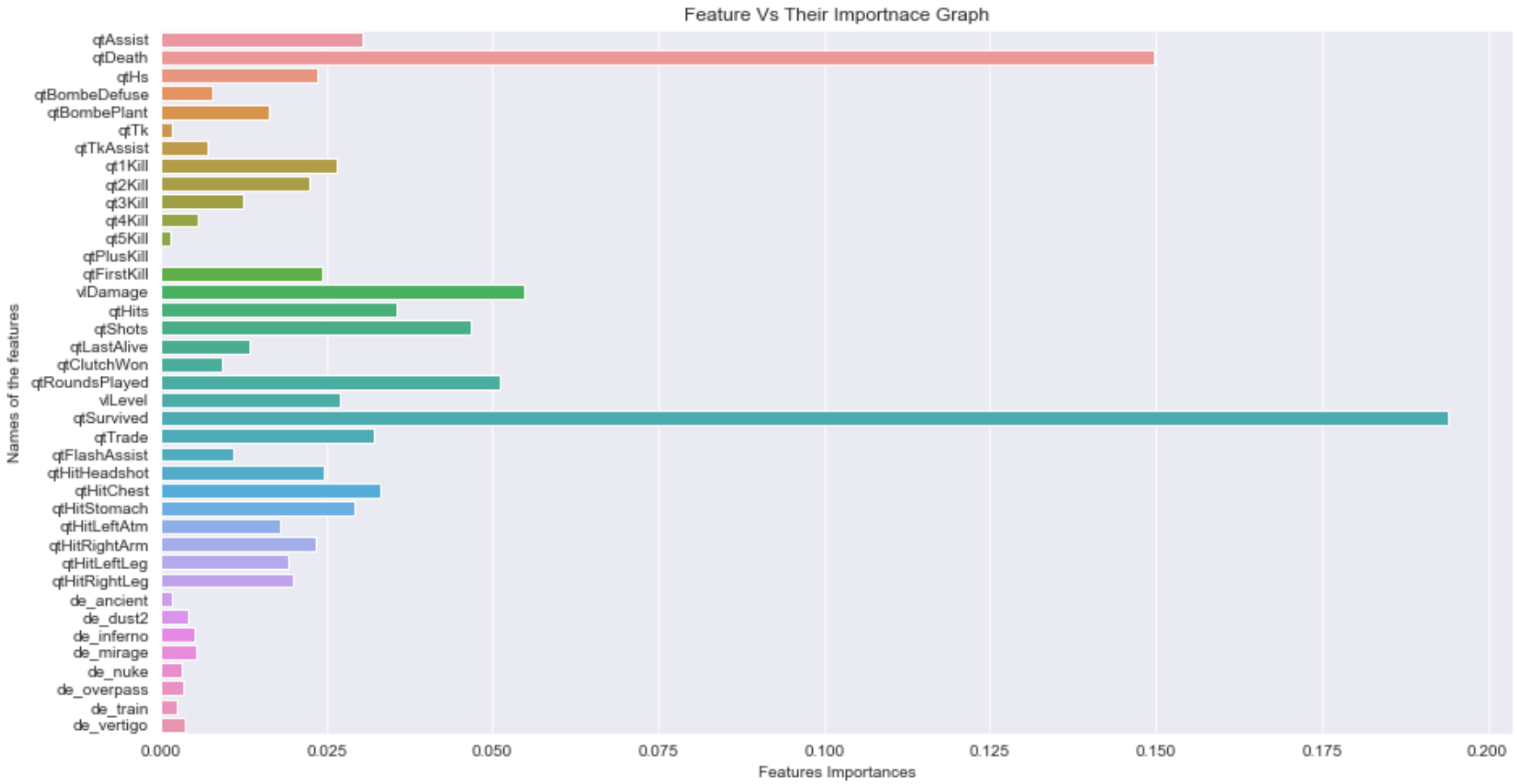
2.6: Can you find the top 3 most important features from the model trained in Q2.5? How do these features compare to the important features that you found from Q1.9? If they differ, which feature set makes more sense?

Answer Yes I can find the top 3 important features from the trained model in 2.5

In [170…

```
plt.figure(figsize = (15, 8))
sns.barplot(y = X_dev.columns.tolist(), x = rf.feature_importances_)
plt.ylabel('Names of the features')
plt.xlabel('Features Importances')
plt.title('Feature Vs Their Importnace Graph')
```

Out[170… `Text(0.5, 1.0, 'Feature Vs Their Importnace Graph')`



In [171…

```
indices = (-rf.feature_importances_).argsort()[:3]
#We have stored the top 3 features indices in the indices variable
X_dev.iloc[:,indices]
```

Out[171…

	qtSurvived	qtDeath	vlDamage
82481	8.0	23	2842
4172	4.0	22	1279
44676	6.0	20	1760
151259	3.0	18	1411
6933	6.0	21	3397
...	...	...	...

	qtSurvived	qtDeath	vlDamage
120216	6.0	16	3439
104018	14.0	5	2200
132300	10.0	15	2726
147344	7.0	19	3882
122295	7.0	22	3665

146757 rows × 3 columns

Answer (Continued)

The top 3 most important features from the model trained in Q2.5 are

- qtSurvived,
- qtDeath
- vlDamage The difference that we see here is that the top 2 features are the same, but the third one is different. The top 3 for the random forest model makes more sense to me according to the nature of the data set

Question 3: Gradient Boosted Trees

3.1: Choose three hyperparameters to tune GradientBoostingClassifier and HistGradientBoostingClassifier on the development dataset using 5-fold cross validation. Report on the time taken to do model selection for both the models. Also, report the performance of the test dataset from the optimal models.

In [49]:
```
from time import perf_counter
from sklearn.ensemble import GradientBoostingClassifier

grid = {'gradientboostingclassifier__n_estimators': [100,150,200],
       'gradientboostingclassifier__learning_rate': np.logspace(-2, -1, 2),
       'gradientboostingclassifier__max_depth': [10,20,30]}

gb = GradientBoostingClassifier(random_state = 1)

pipe_gb_search = make_pipeline( gb)

gs_gb = GridSearchCV(estimator = pipe_gb_search, param_grid = grid, cv = 3,verbose=3)#cv=5 would have been bett
```

```
t1_s = perf_counter()
gs_gb_results = gs_gb.fit(X_dev, y_dev)
t1_e = perf_counter()

print('Elapsed time for grid search on 3 hyperparameters & 5-fold CV on GradientBoostingClassifier is: ',
    t1_e - t1_s)
best_params = gs_gb_results.best_params_
print('n_estimators: ', best_params['gradientboostingclassifier__n_estimators'])
print('learning_rate: ', best_params['gradientboostingclassifier__learning_rate'])
print('max_depth: ', best_params['gradientboostingclassifier__max_depth'])
print('Score on development data: ', gs_gb.score(X_dev, y_dev))
print('Score on test data: ', gs_gb.score(X_test, y_test))
```

```
Fitting 3 folds for each of 18 candidates, totalling 54 fits
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=100;, score=0.790 total time= 1.1min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=100;, score=0.793 total time= 1.1min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=100;, score=0.795 total time= 1.1min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=150;, score=0.792 total time= 1.7min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=150;, score=0.794 total time= 1.7min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=150;, score=0.797 total time= 1.7min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=200;, score=0.792 total time= 2.2min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=200;, score=0.795 total time= 2.3min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=10, gradient
boostingclassifier__n_estimators=200;, score=0.798 total time= 2.2min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=100;, score=0.747 total time= 3.5min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=100;, score=0.752 total time= 3.8min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=100;, score=0.749 total time= 3.6min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=150;, score=0.750 total time= 5.6min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=150;, score=0.755 total time= 6.0min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=150;, score=0.753 total time= 5.7min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=200;, score=0.752 total time= 7.7min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=200;, score=0.759 total time= 8.2min
```

```
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=20, gradient
boostingclassifier__n_estimators=200;, score=0.758 total time= 7.9min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=100;, score=0.728 total time= 3.4min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=100;, score=0.730 total time= 3.4min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=100;, score=0.731 total time= 3.4min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=150;, score=0.727 total time= 5.1min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=150;, score=0.730 total time= 5.2min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=150;, score=0.732 total time= 5.1min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=200;, score=0.727 total time= 6.9min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=200;, score=0.730 total time= 6.7min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.01, gradientboostingclassifier__max_depth=30, gradient
boostingclassifier__n_estimators=200;, score=0.732 total time= 6.7min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=100;, score=0.791 total time= 1.1min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=100;, score=0.796 total time= 1.1min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=100;, score=0.796 total time= 1.1min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=150;, score=0.791 total time= 1.6min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=150;, score=0.795 total time= 1.6min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=150;, score=0.795 total time= 1.6min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=200;, score=0.791 total time= 2.2min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=200;, score=0.793 total time= 2.2min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=10, gradientb
oostingclassifier__n_estimators=200;, score=0.795 total time= 2.2min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=100;, score=0.776 total time= 4.6min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=100;, score=0.778 total time= 4.6min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=100;, score=0.780 total time= 4.6min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=150;, score=0.783 total time= 7.1min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=150;, score=0.786 total time= 7.0min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
```

```
oostingclassifier__n_estimators=150;, score=0.786 total time= 7.1min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=200;, score=0.785 total time= 9.3min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=200;, score=0.789 total time= 9.4min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=20, gradientb
oostingclassifier__n_estimators=200;, score=0.790 total time= 9.6min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=100;, score=0.727 total time= 3.4min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=100;, score=0.730 total time= 3.5min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=100;, score=0.731 total time= 3.5min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=150;, score=0.727 total time= 5.0min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=150;, score=0.730 total time= 5.1min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=150;, score=0.730 total time= 5.0min
[CV 1/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=200;, score=0.732 total time= 6.0min
[CV 2/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=200;, score=0.734 total time= 6.1min
[CV 3/3] END gradientboostingclassifier__learning_rate=0.1, gradientboostingclassifier__max_depth=30, gradientb
oostingclassifier__n_estimators=200;, score=0.734 total time= 6.1min
Elapsed time for grid search on 3 hyperparameters & 5-fold CV on GradientBoostingClassifier is:  14277.81347029
2
n_estimators:  200
learning_rate:  0.01
max_depth:  10
Score on development data:  0.8376636208153615
Score on test data:  0.7943853911147452
```

In [61]:

```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier

grid = {'max_iter': [100,150,200,250],
       'learning_rate': [0.1,0.02,0.01,0.2],
       'max_depth': [10,20,30]}

hgb = HistGradientBoostingClassifier(random_state = 1)

gs_hgb = GridSearchCV(estimator = hgb, param_grid = grid, cv = 5)

t2_s = perf_counter()
gs_hgb_results = gs_hgb.fit(X_dev, y_dev)
t2_e = perf_counter()
```

NOTE Some parametric values won't match with expected results as we all students are using different hyperparametric tuning variables and different processors and due to time constraints I myself would have performed better on training the data but it was taking a lot of time thus we would just try to understand the nature of working of the project through this assignment

In [174…

```
print('The elapsed time for grid search on 3 hyperparameters and the performed 5-fold CV on HistGradientBoostin
    t2_e - t2_s,"(seconds)")
best_params = gs_hgb_results.best_params_
print('Number of estimators', best_params['max_iter'])
print('The best computed learning rate: {:0.3}'.format(best_params['learning_rate']))
print('The best computed max_depth: ', best_params['max_depth'])
print('Score (development data):', gs_hgb.score(X_dev, y_dev))
print('Score(test data):', gs_hgb.score(X_test, y_test))
```

```
The elapsed time for grid search on 3 hyperparameters and the performed 5-fold CV on HistGradientBoostingClassi
fier is:  23.53154404199813 (seconds)
Number of estimators 150
The best computed learning rate: 0.01
The best computed max_depth:  20
Score (development data): 0.7919554092820104
Score(test data): 0.787653311529027
```

3.2: Train an XGBoost model by tuning 3 hyperparameters using 5 fold cross-validation. Compare the performance of the trained XGBoost model on the test dataset against the performances obtained from 3.1

In [180…

```
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')

grid = {'max_estimators': [100,150,200],
       'learning_rate': [0.01,0.1,0.2],
       'max_depth': [10,20,30]}

xgb = XGBClassifier(eval_metric = 'error', random_state = 1)
gs_xgb = GridSearchCV(estimator = xgb, param_grid = grid, cv = 5)

t3_s = perf_counter()
gs_xgb_results = gs_xgb.fit(X_dev, y_dev)
t3_e = perf_counter()
```

```
[22:23:00] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.
```

```
  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[22:23:29] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[22:23:58] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[22:24:28] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[22:24:56] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.


[22:25:25] WARNING: /opt/concourse/worker/volumes/live/7a2b9f41-3287-451b-6691-43e9a6c0910f/volume/xgboost-spli
t_1619728204606/work/src/learner.cc:541:
Parameters: { max_estimators } might not be used.

  This may not be accurate due to some parameters are only used in language bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip through this
  verification. Please open an issue if you find above cases.
```

In [193…

```
print('The elapsed time for grid search on 3 hyperparameters and the performed 5-fold CV on HistGradientBoostin
     (t3_e - t3_s),'(seconds)')
best_params = gs_xgb_results.best_params_
print('The number of estimators: ', best_params['max_estimators'])
print('The best computed learning rate: ', best_params['learning_rate'])
print('The best computed max_depth: ',best_params['max_depth'])
print('Score (development data): {:0.3}'.format(gs_xgb.score(X_dev, y_dev)))
print('Score (Test data):', gs_xgb.score(X_test, y_test))
```

```
The elapsed time for grid search on 3 hyperparameters and the performed 5-fold CV on HistGradientBoostingClassi
fier is:  181199.367833 (seconds)
The number of estimators:  100
The best computed learning rate:  0.1
The best computed max_depth:  20
Score (development data): 1.0
Score (Test data): 0.7917688743526846
```

3.3: Compare the results on the test dataset from XGBoost, HistGradientBoostingClassifier, GradientBoostingClassifier with results from Q1.6 and Q2.1. Which model tends to perform the best and which one does the worst? How big is the difference between the two? Which model would you choose among these 5 models and why?

Answer It is quite evident from the performed computations that clearly the **XGBoost** perform best with the highest accuracy among all the models while the **Decision Trees** basically done at the very beginning gives the worst accuracy as compared to the other models.

It is observed by me that the Decision trees and XGBoost have varied metrics and the difference between them is clearly observed.

If given the five models, I will choose the model with the XGBoost as first of all the performance is the highest as compared to the rest of the models, and the running time was also comparatively lesser than the other models. It took the longer time than the HistGradientBoostingClassifier to train it, but it is also true that it got trained faster than the GradientBoostingClassifier. Thus XGBoost model is the one that I am going to use for my ML model for this problem set , we can improve its performance with time but due to time constraint I had to settle for less accuracy for now and due to low GPU power too on my Mac, I have compromised on the speed of training. Still XGBoosting is the best among the rest 5 models here in my solution.

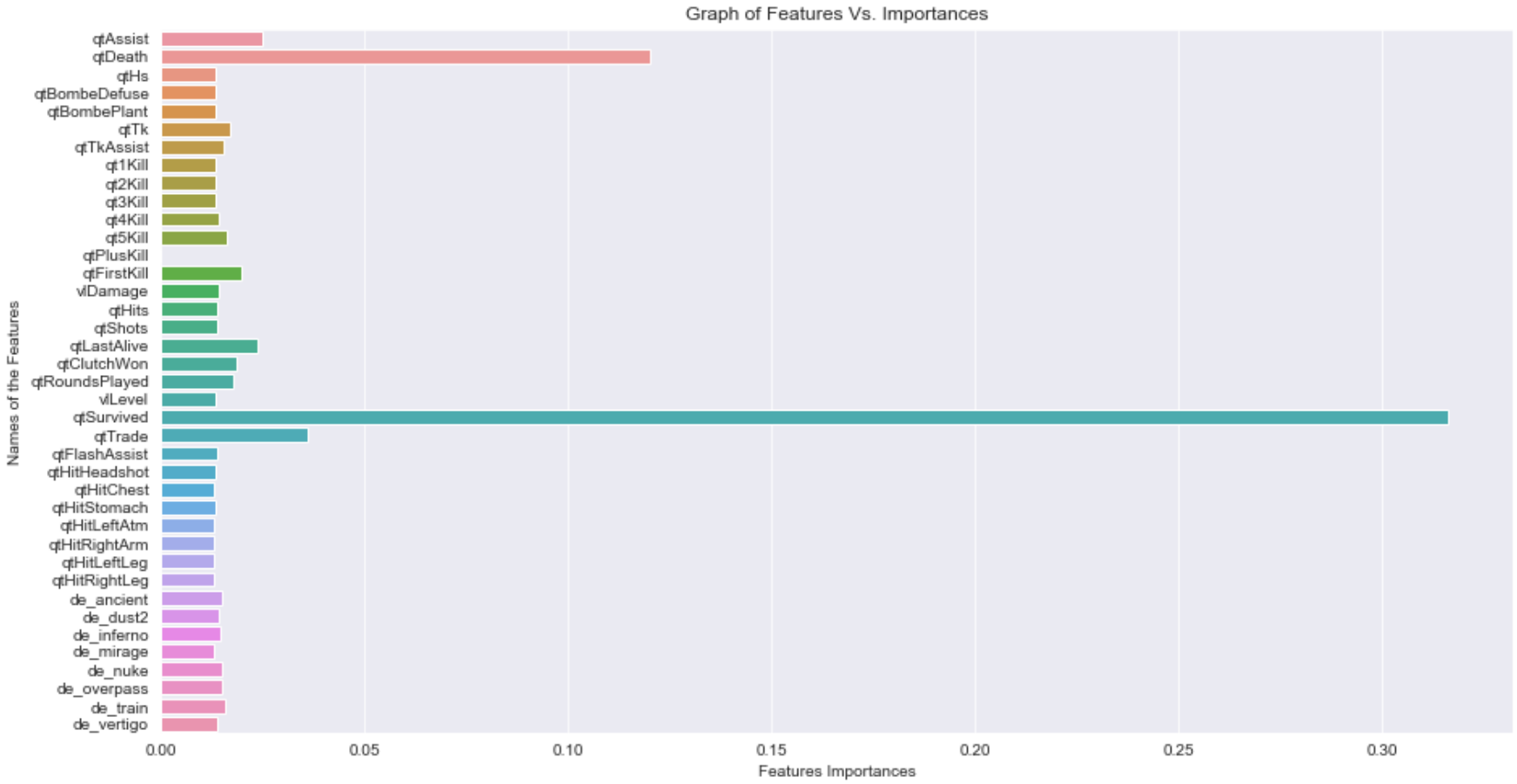
3.4: Can you list the top 3 features from the trained XGBoost model? How do they differ from the features found from Random Forest and Decision Tree? Which one would you trust the most?

In [196…

```
plt.figure(figsize = (15, 8))
```

```
sns.barplot(y = X_dev.columns.tolist(), x = gs_xgb_results.best_estimator_.feature_importances_)
plt.ylabel('Names of the Features')
plt.xlabel('Features Importances')
plt.title('Graph of Features Vs. Importances')
```

Out[196… Text(0.5, 1.0, 'Graph of Features Vs. Importances')



We may choose the best parameters from the below code:-

In [189…

```
indices = (- gs_xgb_results.best_estimator_.feature_importances_).argsort()[:3]
#We have stored the top 3 features indices in the indices variable
X_dev.iloc[:,indices]
```

Out[189…

	qtSurvived	qtDeath	qtTrade

	qtSurvived	qtDeath	qtTrade
82481	8.0	23	4.0
4172	4.0	22	2.0
44676	6.0	20	2.0
151259	3.0	18	0.0
6933	6.0	21	4.0
...	...	...	...
120216	6.0	16	3.0
104018	14.0	5	1.0
132300	10.0	15	4.0
147344	7.0	19	1.0
122295	7.0	22	5.0

146757 rows × 3 columns

Answer

The top 3 features from the trained XGBoost model are :-

1) qtSurvived

2) qtDeath

3) qtTrade

It is same as the decision tree as I see from the plot but still has the third feature to be different with the random forest This is another reason I would rely on the XGBoost Model than compared to any other model as its performance metrics have justified reasoning and have performed well as compare to rest of the models.

3.5: Can you choose the top 7 features (as given by feature importances from XGBoost) and repeat Q3.2? Does this model perform better than the one trained in Q3.2? Why or why not is the performance better?

In [190…

```
top_features = ['qtDeath','qtAssist','qtSurvived','qtTrade','qtLastAlive','qtFirstKill','qtClutchWon']
```

```
grid = {'n_estimators': [100,150],
       'learning_rate': [0.01,0.1,0.2],
       'max_depth': [10,20,30]}

gs_xgb_v2 = GridSearchCV(estimator = xgb, param_grid = grid, cv = 5)

t4_s = perf_counter()
gs_xgb_v2_results = gs_xgb_v2.fit(X_dev[top_features], y_dev)
t4_e = perf_counter()
```

In [195…

```
print('The elapsed time for grid search on 3 hyperparameters & 5-fold CV on top 7 features using XGBoost is: {:
    (t4_e - t4_s),'(seconds)'))
best_params = gs_xgb_v2_results.best_params_
print('The best computed n_estimators: {}'.format(best_params['n_estimators']))
print('The best computed learning_rate: {}'.format(best_params['learning_rate']))
print('The best computed max_depth: {}\n'.format(best_params['max_depth']))
print('Score (Development data): {:.3f}'.format(gs_xgb_v2.score(X_dev[top_features], y_dev)))
print('Score (Test data): {:.3f}'.format(gs_xgb_v2.score(X_test[top_features], y_test)))
```

```
The elapsed time for grid search on 3 hyperparameters & 5-fold CV on top 7 features using XGBoost is: 385294.11
The best computed n_estimators: 150
The best computed learning_rate: 0.1
The best computed max_depth: 20

Score (Development data): 0.905
Score (Test data): 0.774
```

Observation

The model with the top 7 features does not perform better than the one trained in Q3.2. tyhe performance has degraded !

The reson is that we should not be ignoring the features of the dataset,evem if they are not that much contributing to the computation of the predictor label of the target label under the study. All in all we may say that all the features be with little importances still contributes to the model computations to learn and obtaining the predictor.

Question 4: Calibration

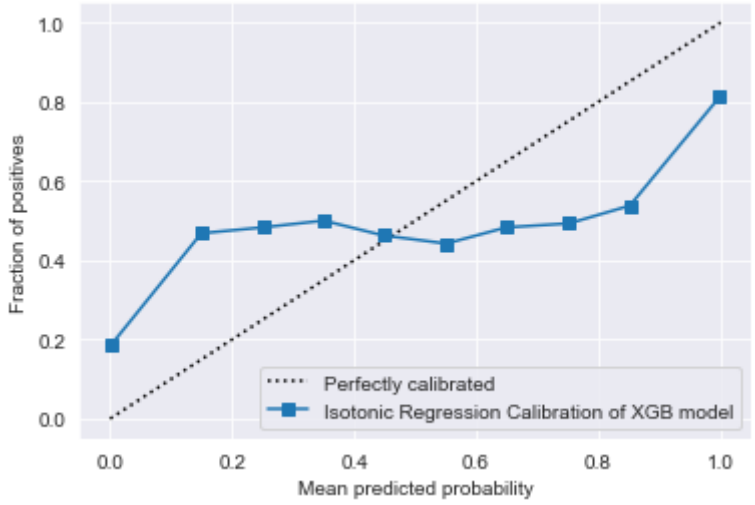
4.1: Estimate the brier score for the XGBoost model (trained with optimal hyperparameters from Q3.2) scored on the test dataset.

In [198…
```
from sklearn.metrics import brier_score_loss
probs = gs_xgb_results.predict_proba(X_test)
probs = probs[:, 1]
y_true = y_test
loss = brier_score_loss(y_true, probs)
print('The brier score loss (XGBoost model) is: ', loss)
```

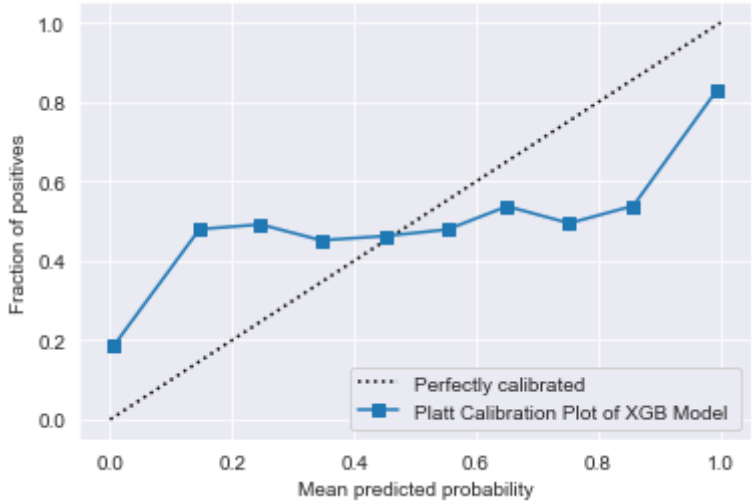
```
The brier score loss (XGBoost model) is:  0.1411260642605482
```

4.2: Calibrate the trained XGBoost model using isotonic regression as well as Platt scaling. Plot predicted v.s. actual on test datasets from both the calibration methods

In [211…
```
from sklearn.calibration import CalibratedClassifierCV, CalibrationDisplay
X_train, X_calib, y_train, y_calib = train_test_split(X_dev, y_dev, test_size = 0.2, random_state = 123)

cal_xgb_iso = CalibratedClassifierCV(gs_xgb_results, cv = 'prefit', method = 'isotonic')
cal_xgb_iso.fit(X_calib, y_calib)
display = CalibrationDisplay.from_estimator(cal_xgb_iso, X_test, y_test, n_bins = 10,
                                            name = 'Isotonic Regression Calibration of XGB model')
```



In [215…
```
#Second Calibration Platt Type
cal_xgb_platt = CalibratedClassifierCV(gs_xgb_results, cv = 'prefit', method = 'sigmoid')
cal_xgb_platt.fit(X_calib, y_calib)
display = CalibrationDisplay.from_estimator(cal_xgb_platt, X_test, y_test, n_bins = 10,
                                            name = 'Platt Calibration Plot of XGB Model ')
```



4.3: Report brier scores from both the calibration methods. Do the calibration methods help in having better predicted probabilities?

In [91]:
```
probs = cal_xgb_iso.predict_proba(X_test)
probs = probs[:, 1]
y_true = y_test
loss = brier_score_loss(y_true, probs)
print('The brier score for the XGBoost model is: ', loss)


probs = cal_xgb_platt.predict_proba(X_test)
probs = probs[:, 1]
y_true = y_test
loss = brier_score_loss(y_true, probs)
print('The brier score for the XGBoost model is: ',loss)
```

```
The brier score for the XGBoost model is:  0.1937846062701516
The brier score for the XGBoost model is:  0.1869344224680692
```

The calibration methods should help in having better predicted probabilities according to the brier score loss. but here in our cases the brier score loss has rised which is not we expected it ois may be due to low GPU and less time ran on the computer that claibration did not contribute that much to the performance but it generally helps in increasing the performances. But today on my solution as a case study it did not helped much as I expected as the brier score loss values has risen from before the claibration been performed on the dataset.

Conclusion

I feel happy I have completed the whole assignment. I want to Thank the TA and our Professor Vijay , for such nice excercise that really helped me understamnding the concepts. thank you so much!