Dota 2 Match Outcome Predictions

An attempt to see if I can make an algorithm that can predict if I will win or lose a match

In [ ]:
```
import re
import pandas as pd
import matplotlib.pyplot as plt
```

Context

- Dota 2 one of the most popular online games.
- Valve, the company that made the game, as a free public API that releases all the data on every match that is played.
- I was able to hunt down my data and do an interestign analysis.

In [2]:
```
apiKey = '0F68F5F96EC6CC04C40F0A28024E20C0'
accountID = '96887717'
```

Supplying the first set of data, and every match will be found after this one

In [4]:
```
id = 'https://api.steampowered.com/IDOTA2Match_570/GetMatchHistory/V001/?key='+ a
+'&matches_requested=100&account_id='+ accountID+'&min_players=10'
table1 = pd.DataFrame(pd.read_json(id).loc['matches']['result'])
```

Gets all the Match ID's from my latest 500 matches

In [8]:
```
curr_table = table1
matches = set()
for i in curr_table['match_id']:
        matches.add(i)

for i in range(5):
    last_match = curr_table['match_id'].iloc[-1]

    id = 'https://api.steampowered.com/IDOTA2Match_570/GetMatchHistory/V001/?key=
    +'&matches_requested=100&account_id='+ accountID+'&min_players=10&start_at_ma
    curr_table = pd.DataFrame(pd.read_json(id).loc['matches']['result'])

    for i in curr_table['match_id']:
        matches.add(i)
```

In [9]:
```
len(matches)
```

Out[9]:
```
500
```

In [10]:
```
matches
```

```
 4894483494,
 4894514179,
 4896617895,
 4896694845,
 4897843080,
 4897888663,
 4897943152,
 4898113204,
 4898146346,
 4898179399,
 4898420589,
 4899668517,
 4899832945,
 4899885311,
 4904411451,
 4904447716,

 4904493689,
 4905691063,
 4905732107,
```

Uses each ID to get the data from that match

In [11]:
```
matchData = []
error_matches = []
initialURL = 'https://api.steampowered.com/IDOTA2Match_570/GetMatchDetails/V001/'
i = 0
for m in matches:
    try:
        match = pd.read_json(initialURL + '?match_id={}&key=0F68F5F96EC6CC04C40F6
        vals = pd.DataFrame(match.loc['players']['result']).set_index('account_id

        # Determining if I won or not
        radiant_win = match.loc['radiant_win']['result']
        if vals['player_slot'] > 11:
            is_radiant = False
        else:
            is_radiant = True
        vals['won'] = is_radiant==radiant_win
        vals['match_id'] = match.loc['match_id']['result']
        matchData.append(vals)
        print(i, end="")
    except:
        error_matches.append(m)
        print('Error with match: '+ str(m))
    finally:
        i+=1
```

```
0123456789101112131415161718192021222324252627282930313233343536373839404142434
4454647484950515253545556575859606162636465666768697071727374757677787980818283
8485868788899091929394959697989910010110210310410510610710810911011111211311411
5116117118119120121122123124125126127128129130131132133134135136137138139140141
1421431441451461471481491501511521531541551561571581591601611621631641651661671
6816917017117217317417517617717817918018118218318418518618718818919019119219319
4195196197198199200201202203204205206207208209210211212213214215216217218219220
2212222232242252262272282292302312322332342352362372382392402412422432442452462
4724824925025125225325425525625725825926026126226326426526626726826927027127227
3274275276277278279280281282283284285286287288289290291292293294295296297298299
3003013023033043053063073083093103113123133143153163173183193203213223233243253
2632732832933033133233333433533633733833934034134234334434534634734834935035135
2353354355356357358359360361362363364365366367368369370371372373374375376377378
3793803813823833843853863873883893903913923933943953963973983994004014024034044
0540640740840941041141241341441541641741841942042142242342442542642742842943043
1432433434435436437438439440441442443444445446447448449450451452453454455456457
4584594604614624634644654664674684694704714724734744754764774784794804814824834
844854864874884894904914924934944954964974984 99
```

In [12]:
```
df = pd.DataFrame(matchData)
df['match_id_int'] = df['match_id'].apply(lambda x: int(x))
```

What the Dataframe looks like

In [13]:
```
df
```

Out[13]:

	player_slot	hero_id	item_0	item_1	item_2	item_3	item_4	item_5	backpack_0	bac
96887717.0	1.0	96.0	50.0	127.0	244.0	125.0	73.0	114.0	0.0	
96887717.0	128.0	53.0	98.0	50.0	218.0	81.0	188.0	24.0	0.0	
96887717.0	3.0	40.0	102.0	88.0	214.0	57.0	24.0	40.0	0.0	
96887717.0	3.0	37.0	254.0	180.0	39.0	36.0	218.0	0.0	0.0	
96887717.0	2.0	105.0	100.0	335.0	214.0	218.0	250.0	108.0	336.0	
...	...	...	...	...	...	...	...	...	...	
96887717.0	0.0	44.0	168.0	43.0	112.0	11.0	75.0	50.0	0.0	
96887717.0	4.0	40.0	43.0	214.0	129.0	0.0	100.0	96.0	0.0	
96887717.0	131.0	33.0	0.0	180.0	40.0	90.0	102.0	178.0	16.0	
96887717.0	2.0	36.0	244.0	269.0	218.0	0.0	231.0	206.0	0.0	
96887717.0	3.0	75.0	0.0	94.0	36.0	214.0	37.0	254.0	0.0	

500 rows × 35 columns

Now cleaning

In [15]:
```
# df.to_csv('dota_matches.csv', index = False)
```

Read in CSV

In [28]:
```
uncleaned = pd.read_csv('dota_matches.csv')
pd.set_option('display.max_columns', None)
uncleaned.head()
```

Out[28]:

	player_slot	hero_id	item_0	item_1	item_2	item_3	item_4	item_5	backpack_0	backpack_1
0	1.0	96.0	50.0	127.0	244.0	125.0	73.0	114.0	0.0	0.0
1	128.0	53.0	98.0	50.0	218.0	81.0	188.0	24.0	0.0	0.0
2	3.0	40.0	102.0	88.0	214.0	57.0	24.0	40.0	0.0	0.0
3	3.0	37.0	254.0	180.0	39.0	36.0	218.0	0.0	0.0	0.0
4	2.0	105.0	100.0	335.0	214.0	218.0	250.0	108.0	336.0	178.0

Removing wherever there is a 'different' game type|

In [30]:
```
uncleaned = uncleaned[uncleaned['custom_game'].isna()].drop(['custom_game','addit
```

In [31]:
```
uncleaned.shape
```

Out[31]:
```
(455, 33)
```

There's a lot of NA's, so I will need to address that later as well

In [32]:
```
uncleaned.isna().sum()
```

Out[32]:
```
player_slot            0
hero_id                0
item_0                 0
item_1                 0
item_2                 0
item_3                 0
item_4                 0
item_5                 0
backpack_0             0
backpack_1             0
backpack_2             0
item_neutral           0
kills                  0
deaths                 0
assists                0
leaver_status          0
last_hits              0
denies                 0
gold_per_min           0
xp_per_min             0
level                  0
hero_damage          272
tower_damage         272
hero_healing         272
gold                 272
gold_spent           272
scaled_hero_damage   272
scaled_tower_damage  272
scaled_hero_healing  272
ability_upgrades     272
won                    0
match_id               0
match_id_int           0
dtype: int64
```

For some reason some of the data was recorded in as 1/0 and other True or False

In [35]:
```
uncleaned['won'].value_counts()
```

Out[35]:
```
1.0      136
0.0      112
True     108
False     99
Name: won, dtype: int64
```

And we also want to remove all guaranteed irrelevent columns

In [37]:
```
uncleaned = uncleaned.drop(['item_0','item_1','item_2','item_3','item_4','item_5'
uncleaned['won'] = uncleaned['won'].apply(lambda x: 1 if x in ['True', '1.0'] els
```

In [38]:
```
uncleaned.to_csv('dota_matches_cleaned.csv')
```

These are all the important columns, besides the NA ones

In [41]:
```
uncleaned.isna().sum(), uncleaned.shape
```

Out[41]:
```
(player_slot            0
 hero_id                0
 kills                  0
 deaths                 0
 assists                0
 last_hits              0
 denies                 0
 gold_per_min           0
 xp_per_min             0
 level                  0
 hero_damage          272
 tower_damage         272
 hero_healing         272
 gold                 272
 gold_spent           272
 scaled_hero_damage   272
 scaled_tower_damage  272
 scaled_hero_healing  272
 won                    0
 match_id               0
 match_id_int           0
 dtype: int64,
 (455, 21))
```

In [42]:
```
cleaned = pd.read_csv('dota_matches_cleaned.csv')
```

Notice a few things in the following graphs

- They are seperated across 3 variables and the 2 different outcomes I want to predict
- Blue dots are means of the data set
- As expected I have less deaths when I win
  - I have more kills when I win
  - And I am usually a higher level when I win
- These are things I would believe intuitively, and the data supports it
- The distributions are mostly normal as well
- And because of these points, I believe that these are good variables to be using

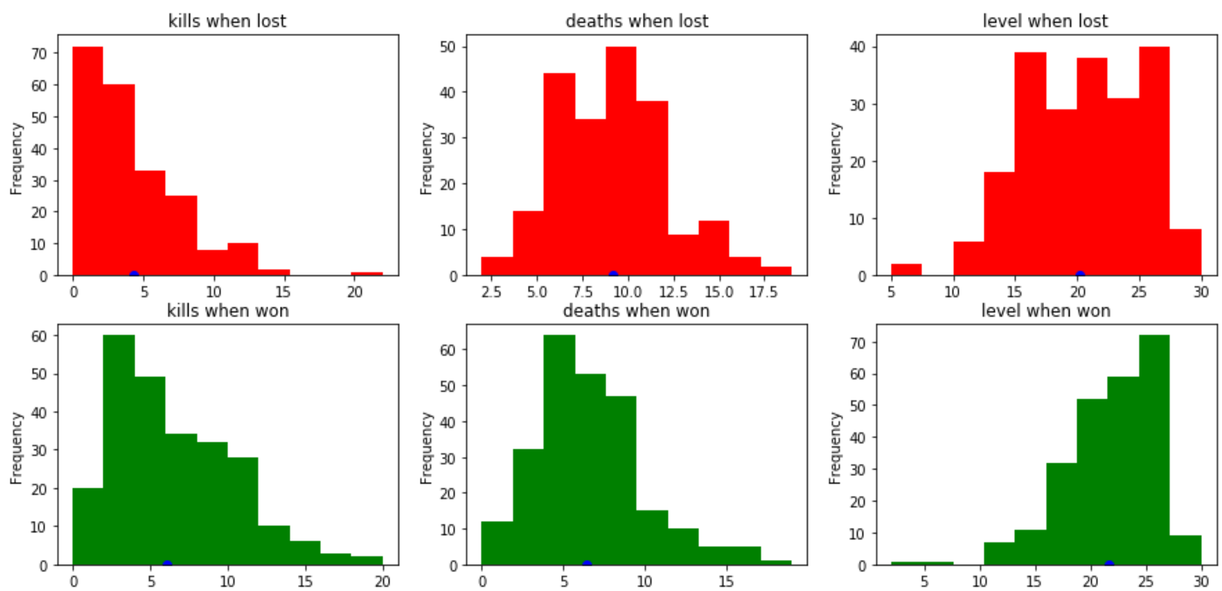
In [60]:
```
fig,axes = plt.subplots(2,3, figsize = (15,7))
cleaned[cleaned['won'] == 1]['kills'].plot(kind = 'hist', ax = axes[1,0], color =
cleaned[cleaned['won'] == 0]['kills'].plot(kind = 'hist', ax = axes[0,0], color =

cleaned[cleaned['won'] == 1]['deaths'].plot(kind = 'hist', ax = axes[1,1], color
cleaned[cleaned['won'] == 0]['deaths'].plot(kind = 'hist', ax = axes[0,1], color

cleaned[cleaned['won'] == 1]['level'].plot(kind = 'hist', ax = axes[1,2], color =
cleaned[cleaned['won'] == 0]['level'].plot(kind = 'hist', ax = axes[0,2], color =

axes[1,1].plot(cleaned[cleaned['won'] == 1]['deaths'].mean(),0, 'ro', color = 'bl
axes[0,1].plot(cleaned[cleaned['won'] == 0]['deaths'].mean(),0, 'ro', color = 'bl
axes[1,0].plot(cleaned[cleaned['won'] == 1]['kills'].mean(),0, 'ro', color = 'blu
axes[0,0].plot(cleaned[cleaned['won'] == 0]['kills'].mean(),0, 'ro', color = 'blu
axes[1,2].plot(cleaned[cleaned['won'] == 1]['level'].mean(),0, 'ro', color = 'blu
axes[0,2].plot(cleaned[cleaned['won'] == 0]['level'].mean(),0, 'ro', color = 'blu
```

Out[60]: [<matplotlib.lines.Line2D at 0x154588b3d48>]



I have 2 paths to go...

- Remove those NA columns and train my data on only **8 variables**, but **~450 data points**
- Remove the columns rows with NA , and train on **16 variables**, but only **~185 data points**

Decided to go for more data points, and less variables, since overfitting with too many variables is a problem anyways

Variables:

- **kills**: Number enemies eliminated by the player in that match
- **deaths**: Number of times the player was eliminated
- **assists**: Number of times the player helped eliminate an enemy
- **last_hits**: Number of points the player got from killing non-player monsters
- **denies**: Number times the player prevented the enemy from getting points for killing a non-player monster
- **gold_per_min**: Average gold points per minute (gold is used to get items)
- **xp_per_min**: Average experience points per minute (experience/XP is used to get level up and get stronger)
- **level**: A total number of XP points

What are we dealing with?

- Supervised learning
  - The data *is labeled*
  - This fits a method like classification/regression well
- Difference?
  - Here we are really able to use both
- Classification
  - Can say there are only 2 categories, win or loss
  - Train the model on those distinct categories
- Regression
  - Count 1's as 'win' and 0 as 'loss'
  - Train a model and get weights
  - Allow them model to try and predict a number between 1 and 0 given match data and the weights
  - That number is effectively a Probability that the match was won.
  - To classify, if the Probability was >= .5, then consider it a win

Choices

- Scikit learn has really useful functions for this
  - CalibratedClassifierCV helps classify with probabilities
  - They have built in function such as K-means, another good approach
    - Uses previous match data and sees looks at the kth most similar matches, and takes the most common outcome from those
  - A huge list of 'ensamble' classification *and* regression methods, that I have some experience with.
    - Built around averaging and aims to improve generalizability

Decision

- Going to use linear regression
- Much simpler than Scikit
  - Have much more expereince and its easier to know what is happening at each step

- Follow similar methods as we did in TA3

Explanation of Methods

- Mostly a few simple steps

Linear Regression

1. Select a model
    - My model will be a basic linear model
    - follows a y = w0 + w1x1 + w1x1 ... wnxn
2. Put data into array (matrix)
    - create the A matrix
        - the left most column will be a column of all ones, used to make w0, a weight not scaled by any inputs
        - each column in the df is a column in the matrix
3. Solve for weight vector
    - Using our A matrix, we solve y = Aw
        - y is our list of known outputs and A is the matrix we just created
4. Report the model
    - Now that we have the weights, we can fill in "w0 + w1x1 + w1x1 ... wnxn" with our weights (w's) and the X's (varaible names corresponding to each W
5. Visualize the model using predicted values
    - This part we can't really do (without some PCA) since our data is 6th dimensional

Results

- I hope to find a model with around 75% accuracy

  What if it doesn't meet that cutoff?
- If the model doesn't work too well, I can try to change my model by adding x^2's or other numbers to make them exponentially more important
- Try to do a test/train set
    - If the train set can predict the test set well, use the current model
    - If not, change the model and run the algorithm again