European Soccer Data - Mod3 Project

Harrison Hardin and Joe Down

About the Data Set

From Kaggle.com:

The ultimate Soccer database for data analysis and machine learning

What you get:

- +25,000 matches
- +10,000 players
- 11 European Countries with their lead championship
- Seasons 2008 to 2016
- Players and Teams' attributes* sourced from EA Sports' FIFA video game series, including the weekly updates
- Team line up with squad formation (X, Y coordinates)
- Betting odds from up to 10 providers
- Detailed match events (goal types, possession, corner, cross, fouls, cards etc...) for +10,000 matches

- Data stored in SQLite database

Data Wrangling/Cleaning

```
#making a string to add all the bookies to the query
bookies = "d.B365H, d.B365D, d.B365A, d.BWH, d.BWD, d.BWA, d.IWH, d.IWD, d.IWA, d.LBH, d.LBD,
bookies = bookies.replace('d','m')


#main query gets win loss data and bookie odds
q=("""
        SELECT m.home_team_goal, m.away_team_goal, m.home_team_api_id, {} FROM
        Match m

        """.format(bookies))

df = pd.read_sql_query(q, conn)
df_copy = pd.read_sql_query(q, conn)

#Sets up columns to see who won or lost or draw
df['HomeWin']=df.home_team_goal>df.away_team_goal
df['AwayWin']=df.away_team_goal>df.home_team_goal
df['Draw']=df.home_team_goal==df.away_team_goal
```

Data Wrangling/Cleaning

```
#checking for null values
df.isna().sum()
```

```
home_team_goal          0
away_team_goal          0
HomeWin                 0
AwayWin                 0
Draw                    0
home_team_api_id        0
B365H                3387
B365D                3387
B365A                3387
BWH                  3404
BWD                  3404
BWA                  3404
IWH                  3459
IWD                  3459
IWA                  3459
LBH                  3423
LBD                  3423
LBA                  3423
PSH                 14811
PSD                 14811
PSA                 14811
WHH                  3408
WHD                  3408
WHA                  3408
SJH                  8882
SJD                  8882
SJA                  8882
VCH                  3411
VCD                  3411
VCA                  3411
GBH                 11817
GBD                 11817
GBA                 11817
BSH                 11818
BSD                 11818
BSA                 11818
dtype: int64
```

```
#drop columns with over 5000 null value - 22432 total rows
df.drop(['PSA','PSH','PSD','GBH','GBD','GBA','BSH','BSD','BSA','SJH','SJD','SJA'], axis=1, inplace=True)
```

Slide Ty

```
#check for null values again
df.isna().sum()
```

```
home_team_goal          0
away_team_goal          0
HomeWin                 0
AwayWin                 0
Draw                    0
home_team_api_id        0
B365H                3387
B365D                3387
B365A                3387
BWH                  3404
BWD                  3404
BWA                  3404
IWH                  3459
IWD                  3459
IWA                  3459
LBH                  3423
LBD                  3423
LBA                  3423
WHH                  3408
WHD                  3408
WHA                  3408
VCH                  3411
VCD                  3411
VCA                  3411
dtype: int64
```

Question 1: Bookie Comparisons

Is there a significant difference between the underlying distributions of the various bookies? First, is there a difference specifically between each bookie to each other (ttest) within each category (home win, away win, and draw)? Second, is there a difference between each bookie and the underlying distributions of all the other bookies (anova test)?

Log Transformation - Bookie Odds

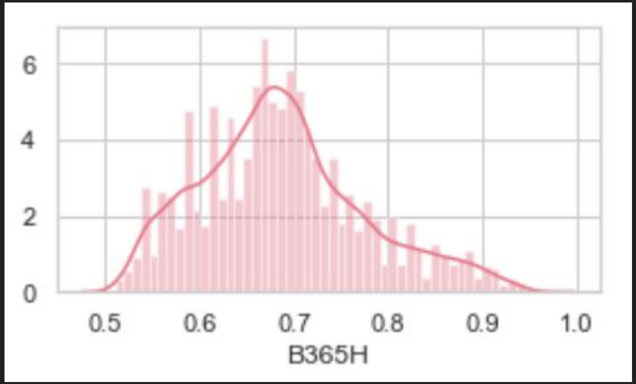
```
#This is to check and see which columns to transform
#The result of the test is that home win and draw columns should be transformed but not the away win column
#(True means transform improves normality)

def log_transform_test(df):
    improved_list = []
    for column in df.columns:
        pre_transform_stat = stats.normaltest(df[column])
        transformed_col = df[column].apply(lambda x: np.log(x))
        post_transformed_stat = stats.normaltest(transformed_col)
        improved_list.append([column, pre_transform_stat>post_transformed_stat])
    return improved_list
log_transform_test(df.loc[:,'B365H':'VCD'])
```

```
[['B365H', True],
 ['B365D', True],
 ['B365A', False],
 ['BWH', True],
 ['BWD', True],
 ['BWA', False],
 ['IWH', True],
 ['IWD', True],
 ['IWA', False],
 ['LBH', True],
 ['LBD', True],
 ['LBA', False],
 ['WHH', True],
 ['WHD', True],
 ['WHA', False],
 ['VCH', True],
 ['VCD', True]]
```

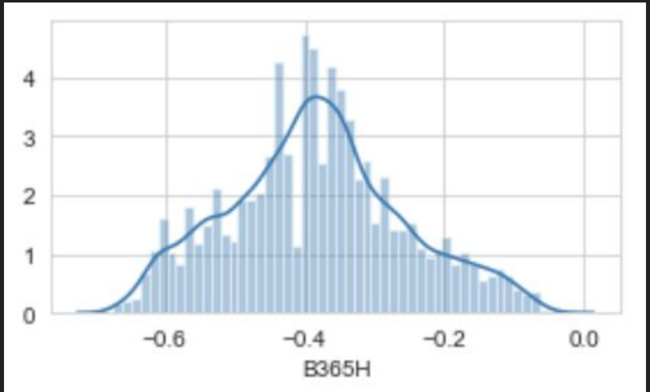
Log Transformation

```
def log_transform(df):
    for column in df.columns:
        df[column] = df[column].apply(lambda x: np.log(x))

    return df

df_log_transformed = df.copy()
df_log_transformed.loc[:,'B365H':'VCH':3] = log_transform(df_log_transformed.copy().loc[:,'B365H':'VCH':3])
df_log_transformed.loc[:,'B365D':'VCD':3] = log_transform(df_log_transformed.copy().loc[:,'B365D':'VCD':3])
```

Before



After



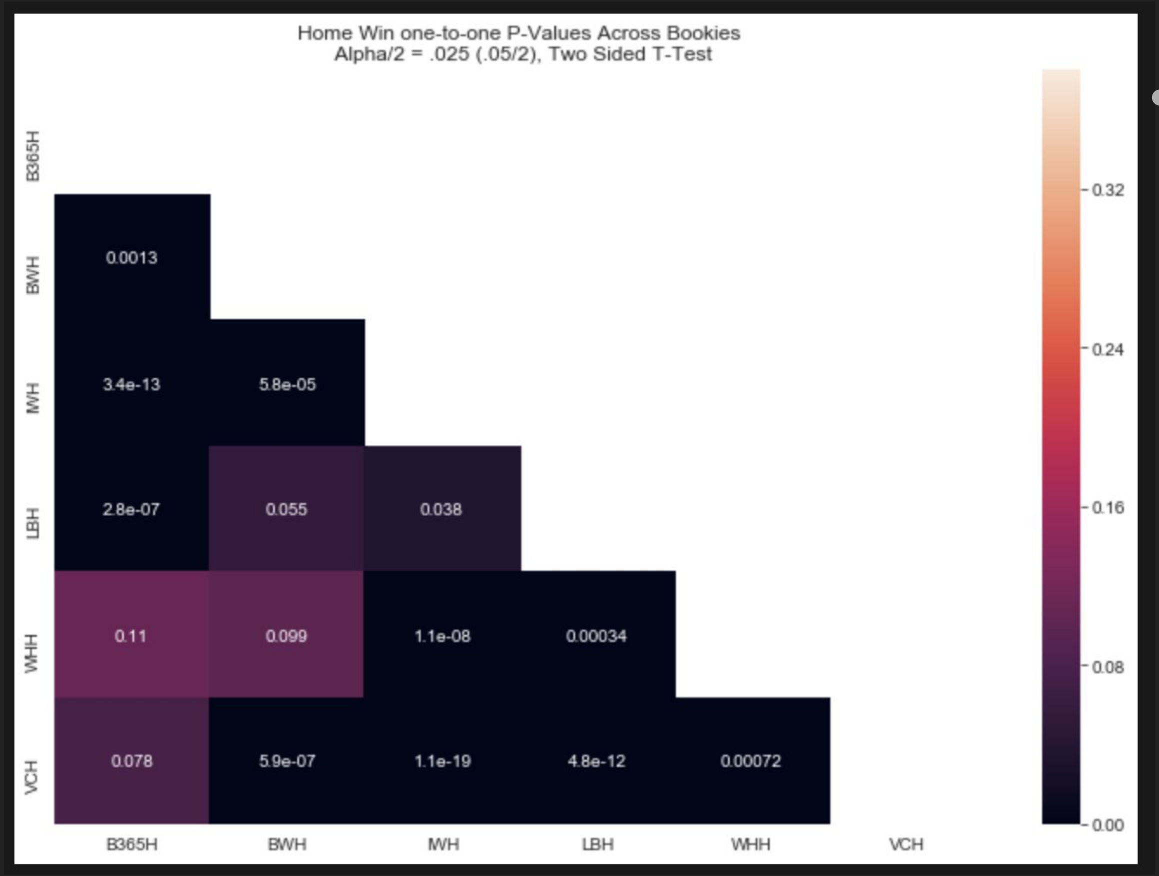
Independent Two Sided T-tests

```
for bookie_df in list_of_dataframes:
    for bookie_1 in bookie_df:
        for index, bookie_2 in enumerate(bookie_df):
            x = df[bookie_1]
            y = df[bookie_2]
            ttest=stats.ttest_ind(x,y)
            bookie_df[bookie_1].iloc[index] = ttest[1]
```

- Iterate through bookies
- Conduct T-tests
- Create Seaborn heatmaps

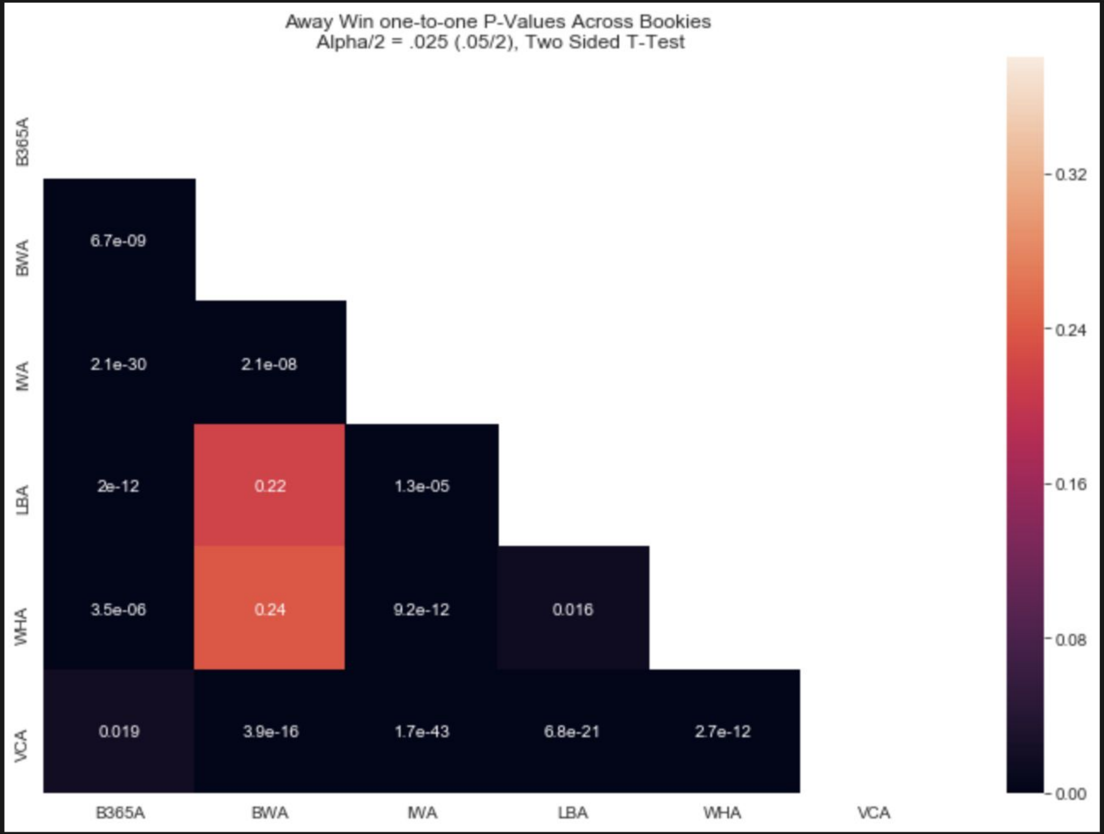
```
#This generates a heatmap of p-values for home wins
sns.set_style("whitegrid")
fig, ax = plt.subplots(figsize= (12,8))
sns.heatmap(home_win_df, vmin=0, vmax=.38, annot = True, ax = ax)
ax.set_title("Home Win one-to-one P-Values Across Bookies\n Alpha/2 = .025 (.05/2), Two Sided T-Test")
ax.patch.set_alpha(0.5)
ax.set_ylabel('')
ax.set_xlabel('')
```

Independent Two Sided T-tests - Home Win

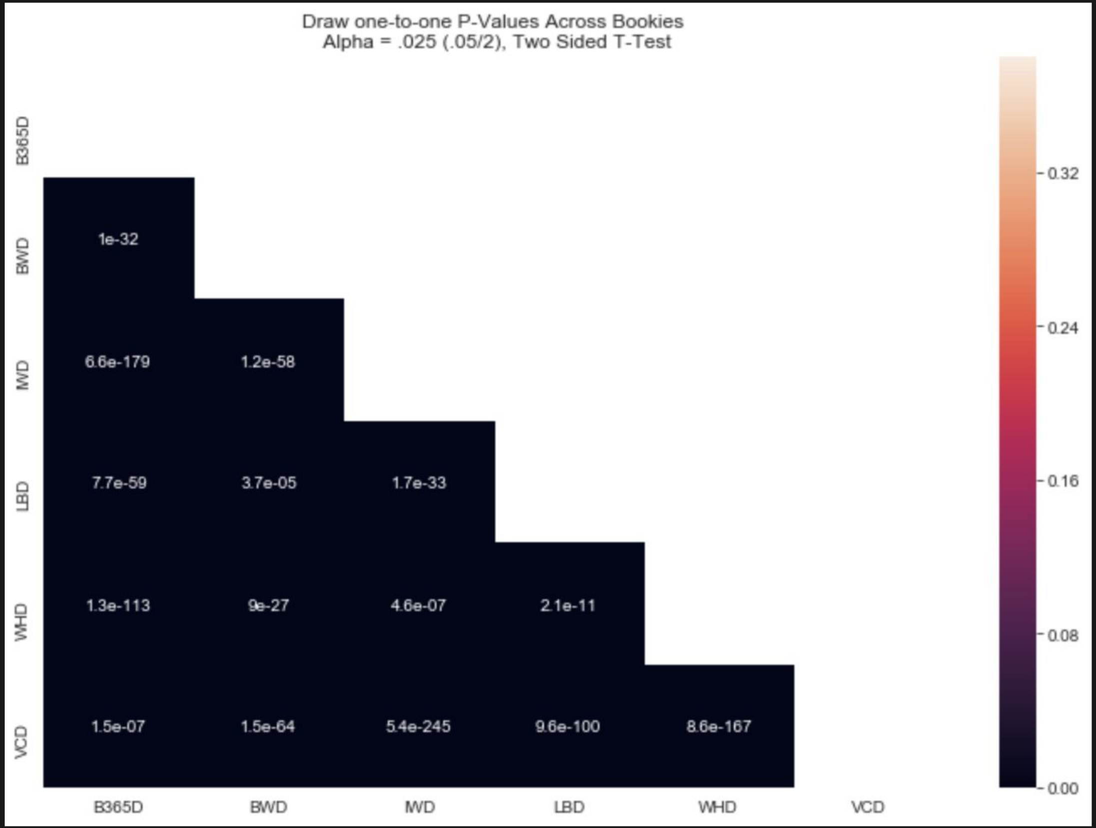


- Values above .025 reject the null hypothesis that the underlying distributions are the same

Independent Two Sided T-tests Away Win



Independent Two Sided T-tests Draw



One Way Anova Test

```
result = stats.f_oneway(df.B365H, df.BWH, df.IWH, df.LBH, df.WHH, df.VCH)
print(' ')
print(f'The p-value for this one way ANOVA test is {result[1]}')
print('Therefore, the null hypothesis that these populations come from the same underlying distrubtion')
print(f"was {('rejected' if result[1]<=.05 else 'not rejected')} at the .95 level of confindence.")
print(' ')

result = stats.f_oneway(df.B365A, df.BWA, df.IWA, df.LBA, df.WHA, df.VCA)
print(f'The p-value for this one way ANOVA test is {result[1]}')
print('Therefore, the null hypothesis that these populations come from the same underlying distrubtion')
print(f"was {('rejected' if result[1]<=.05 else 'not rejected')} at the .95 level of confindence.")
print(' ')

stats.f_oneway(df.B365D, df.BWD, df.IWD, df.LBD, df.WHD, df.VCD)
print(f'The p-value for this one way ANOVA test is {result[1]}')
print('Therefore, the null hypothesis that these populations come from the same underlying distrubtion')
print(f"was {('rejected' if result[1]<=.05 else 'not rejected')} at the .95 level of confindence.")
print(' ')

#For all three of my one-way ANOVA tests, the null hypothesis that the individual distributions
#come from the same underlying distribution was rejected at the 95% level of confidence.

#This means essentially there is no single common distribution that can predict all the bookie's
#behavior for all three categories of odds, home wins, away wins, and draws.
```

```
The p-value for this one way ANOVA test is 2.750142525557248e-22
Therefore, the null hypothesis that these populations come from the same underlying distrubtion
was rejected at the .95 level of confindence.

The p-value for this one way ANOVA test is 3.3042277037498044e-51
Therefore, the null hypothesis that these populations come from the same underlying distrubtion
was rejected at the .95 level of confindence.

The p-value for this one way ANOVA test is 3.3042277037498044e-51
Therefore, the null hypothesis that these populations come from the same underlying distrubtion
was rejected at the .95 level of confindence.
```

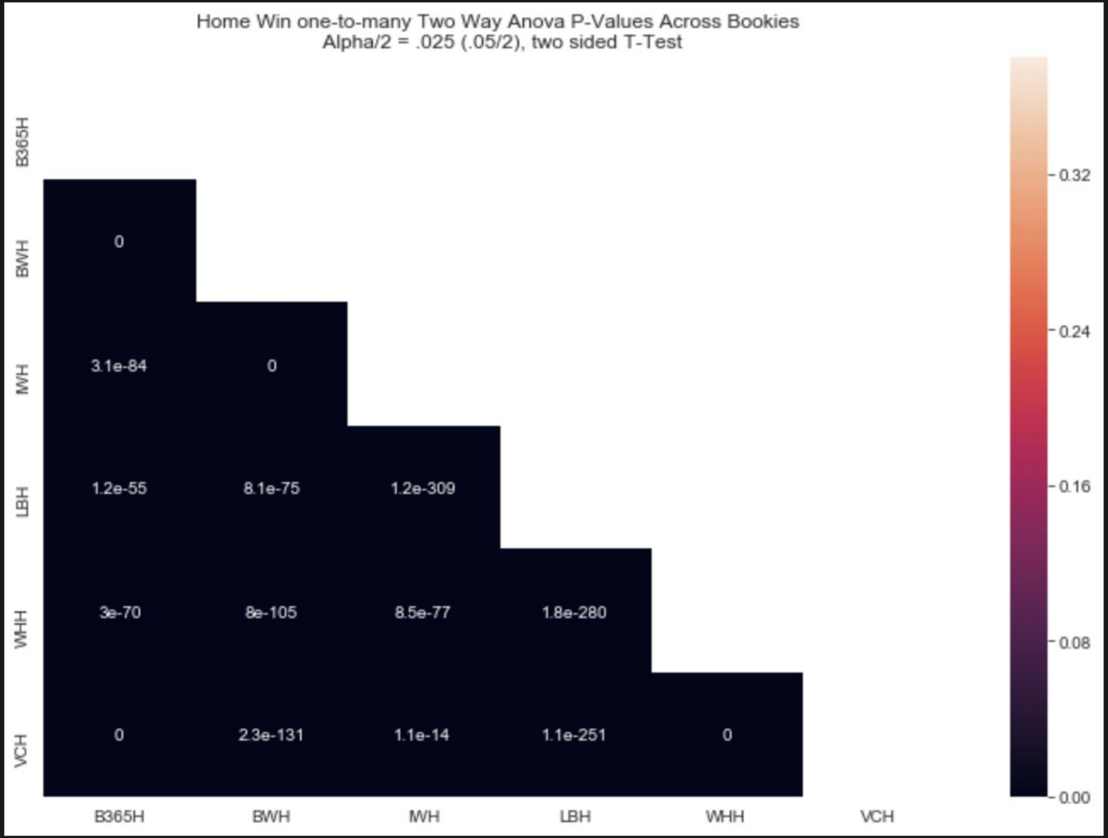
- Do all the bookie odds distributions come from the same underlying distributions? No.

Two Way Anova Test

```
for bookie in anova_df_list[index]:
    formula = f'{bookie} ~ ' +' + '.join([bookie for bookie in anova_df_list[index].drop(bookie, axis=1).columns])
    if str(dfl_item)==str(away_win_df):
        match_list.append([index, "df"])
        model = sm.ols(formula, df).fit()
    else:
        #switch commenting to choose between log transformed or not
        #model = sm.ols(formula, df).fit()
        model = sm.ols(formula, df_log_transformed).fit()
        match_list.append([index, "df_log_transformed"])

    aov_table = sm2.stats.anova_lm(model, typ=2)
    for bookie_2 in anova_df_list[index].drop(bookie, axis=1).columns:
        anova_df_list[index].loc[bookie][bookie_2] = aov_table.loc[bookie_2]['PR(>F)']
```

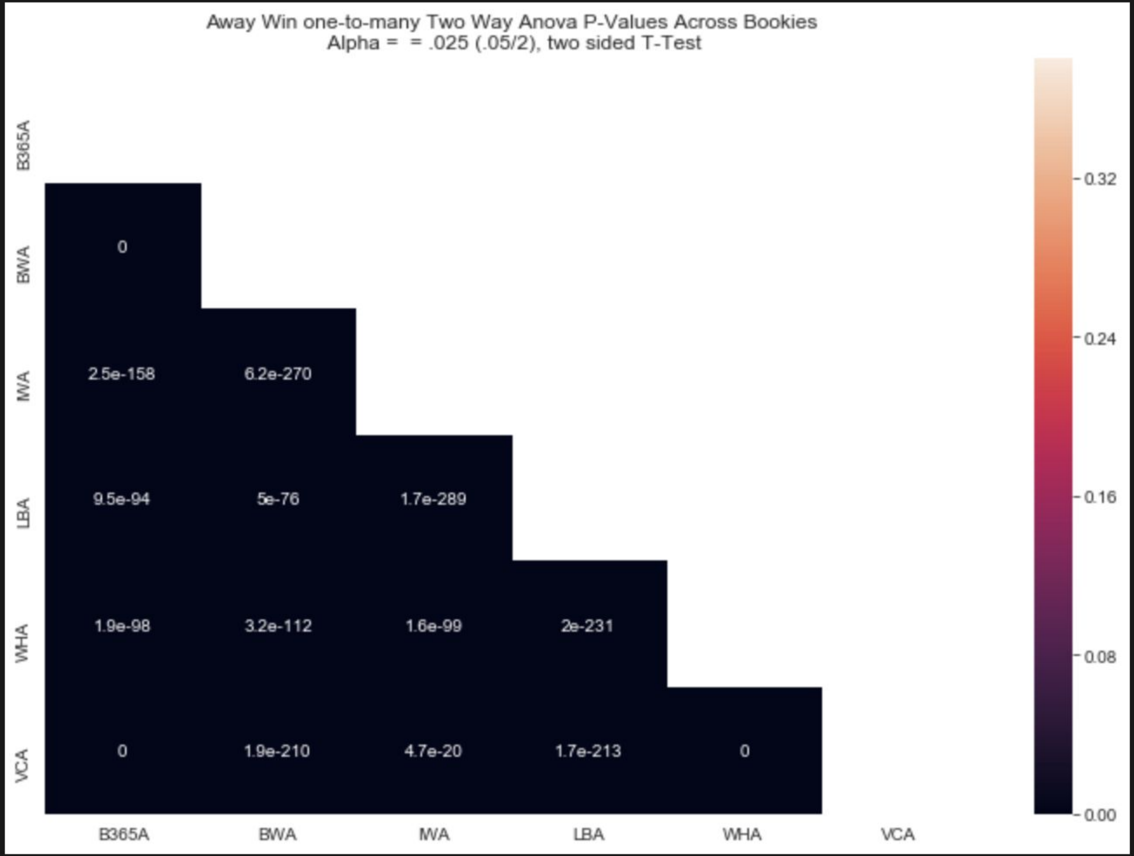
- Does one “target” bookie come from the same distribution as all the other bookie odds?

Two Way Anova Test - Home Win

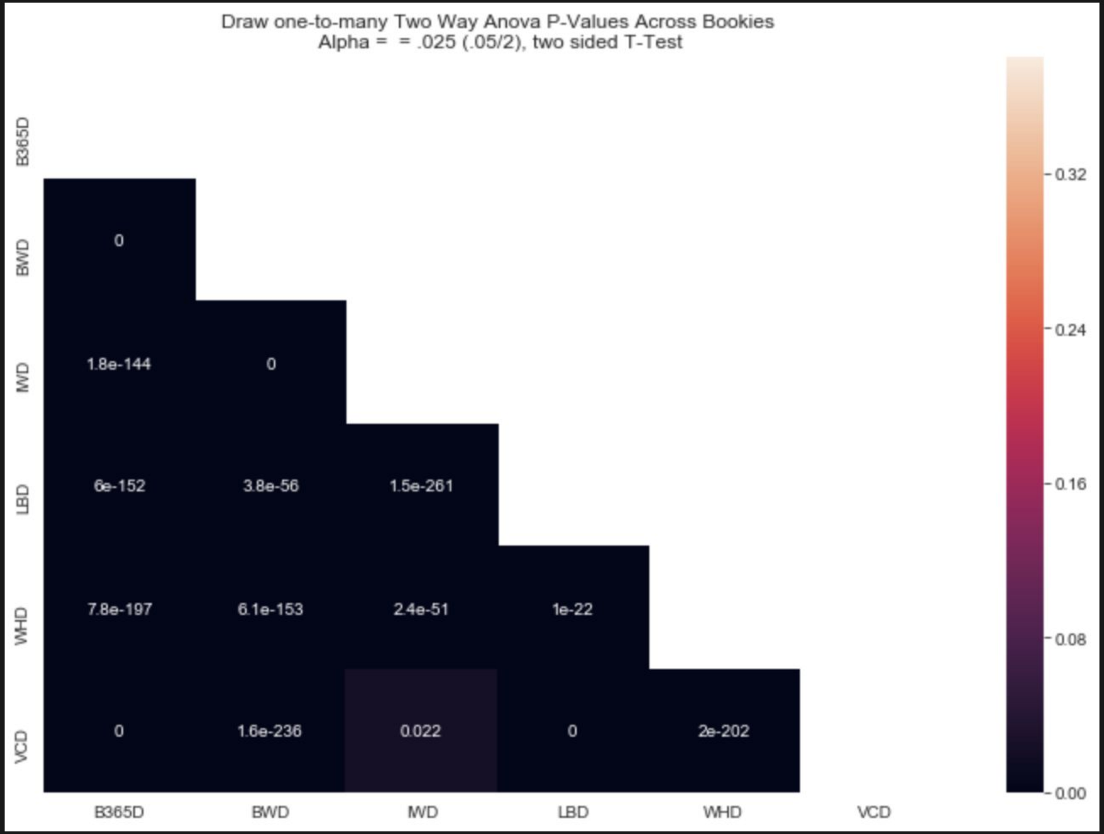


- All of these two way anova tests reject the null hypothesis and therefore the distributions do not come from the same underlying distribution.

Two Way Anova Test - Away Win



Two Way Anova Test - Draw



- VCD/IWD just barely rejects the null hypothesis but it does: (.022<.025)

Question 2: Athletic Measures Vs. Soccer Skill

Is there a significant difference between the upper and lower groupings of players rated by various athletic measures vs those same players rated by various soccer skills?

Log Transformation - Athletic Measures vs. Skills

```
log_transform_test(ath_df[skills_list])
#False indicates a log transformation will not improve normality
#We decided not to log transform due to 10/16 of these column tests resulting in false
```

```
[['crossing', False],
 ['finishing', True],
 ['heading_accuracy', False],
 ['short_passing', False],
 ['dribbling', False],
 ['free_kick_accuracy', False],
 ['long_passing', False],
 ['ball_control', False],
 ['long_shots', False],
 ['aggression', False],
 ['interceptions', True],
 ['positioning', False],
 ['penalties', False],
 ['marking', True],
 ['standing_tackle', True],
 ['sliding_tackle', True]]
```

```
log_transform_test(ath_df[ath_list])
```

```
[['acceleration', False],
 ['sprint_speed', False],
 ['agility', False],
 ['reactions', False],
 ['balance', False],
 ['jumping', False],
 ['stamina', False],
 ['strength', False]]
```

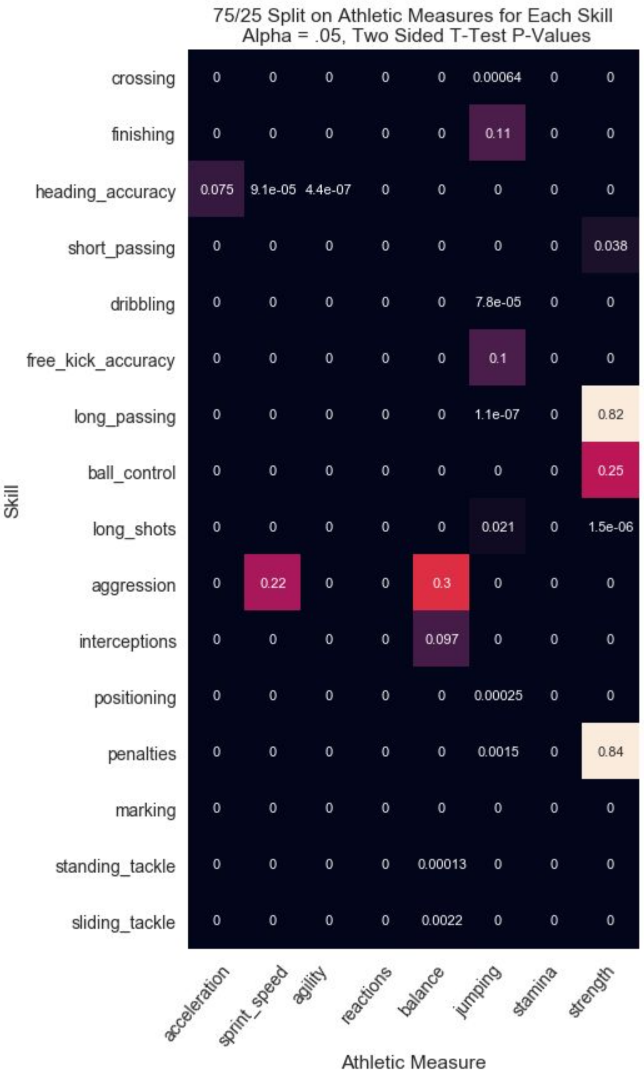
Log transform not chosen because testing did not clearly indicate necessity.

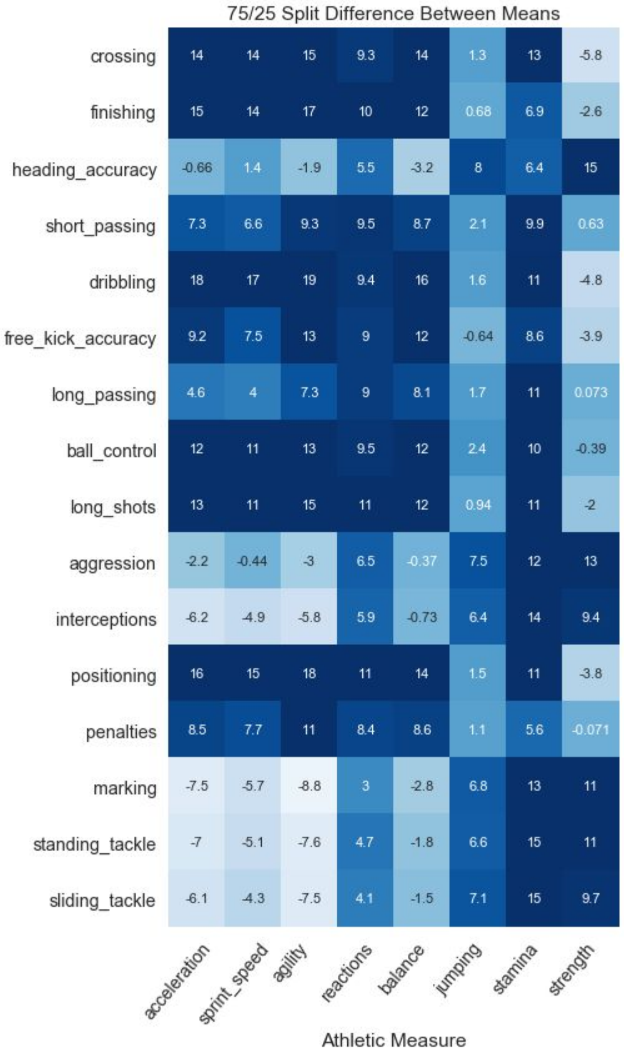
Athletic Measures vs. Skills - Steps

```
#This will populate two df's, one of pvalues and one of the difference between means

#function takes in the main_df, a p-values holder df, the percentage desired for split (must be 75, 50, or 25),
#a list of athletic measures and a list of osccer skills
def ath_to_skill_ttest(ath_df, qualities_df, top_percent, ath_list, skills_list):

    #setting up empty dictionary
    dict_means = {}

    #reset values to zero
    for col in qualities_df.columns:
        qualities_df[col].values[:] = 0

    #creating a string with % after the percent
    #this is needed for grabbing values off series.describe()
    top_percent_string = str(top_percent)+'%'
    bottom_percent_string = str(100-top_percent)+'%'

    #create columns classifying players into upper and lower groups for each athletic ability
    for ath in ath_list:
        ath_df[ath+'_top'+top_percent_string]=ath_df[ath]>= ath_df[ath].describe()[top_percent_string]
        ath_df[ath+'_bottom'+bottom_percent_string]=ath_df[ath]< ath_df[ath].describe()[top_percent_string]

    #this is a dictionary that hold the 25/75 splits
    dict_skill_split = {}
    for i, ath in enumerate(ath_list):
        for j, skill in enumerate(skills_list):

            #Creates a dictionary entry for each ath/skill combo with an array the skill of top quartile of the ath
            #and array of the skill of the bottom 75
            dict_skill_split.update({ath+skill : [ath_df.loc[ath_df[ath+'_top'+top_percent_string]][skill],
                                            ath_df.loc[ath_df[ath+'_bottom'+bottom_percent_string]][skill]]})

            dict_means.update({ath+skill : ath_df.loc[ath_df[ath+'_top'+top_percent_string]][skill].mean()-
                                            ath_df.loc[ath_df[ath+'_bottom'+bottom_percent_string]][skill].mean()})



#This loops through the qualities to conduct individual one-to-one ttests
    fail_reject_list = []
    reject_list = []
    mean_df = qualities_df.copy()
    for a, ath in enumerate(ath_list):
        for s, skill in enumerate(skills_list):
```

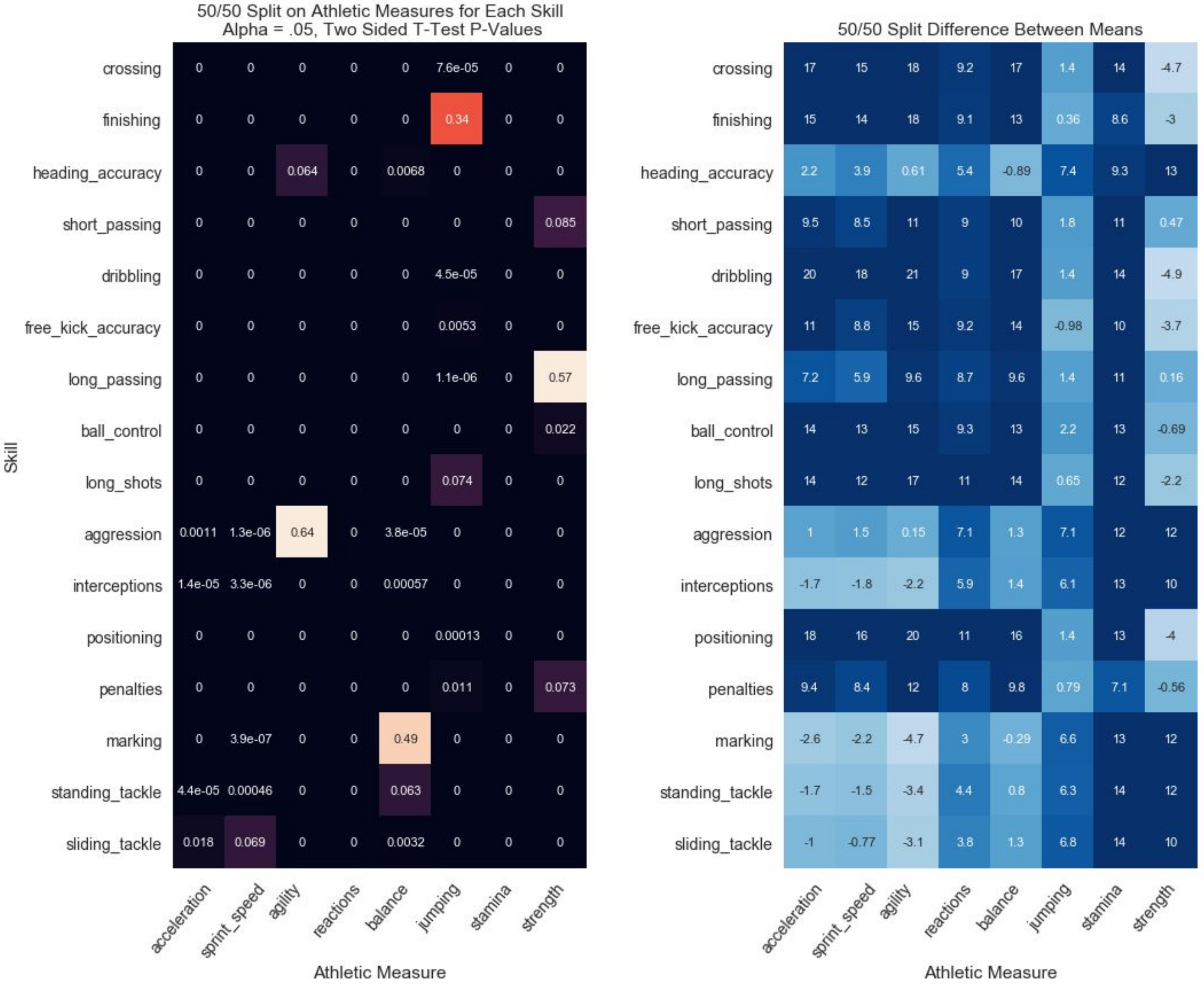
- Split athletic measures by upper and lower groups
- Compare those splits across skills
- Perform individual ttests
- Perform anova tests

Individual T-tests between upper and lower groupings, in this case 75%/25% split on the athletic measure.

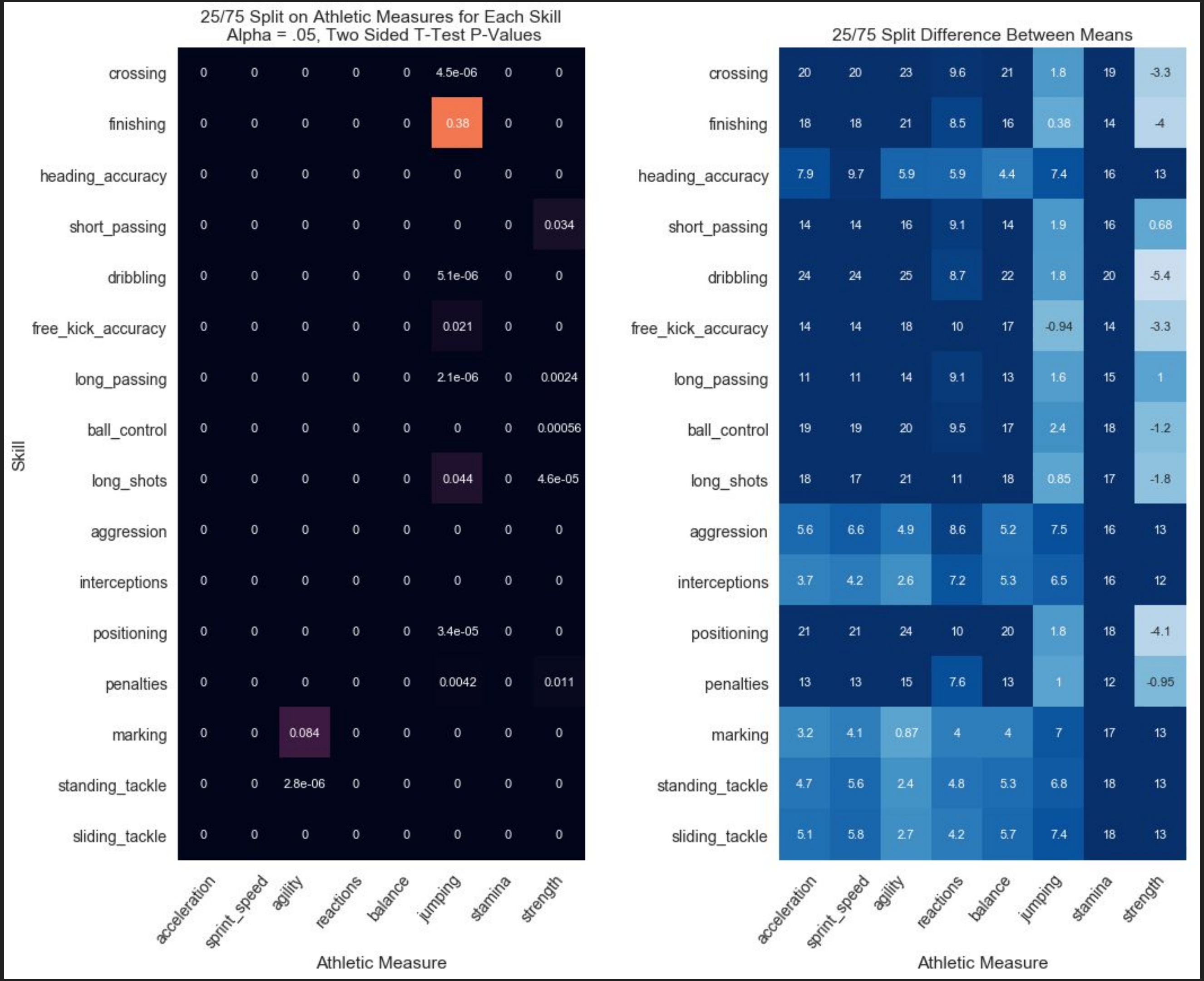


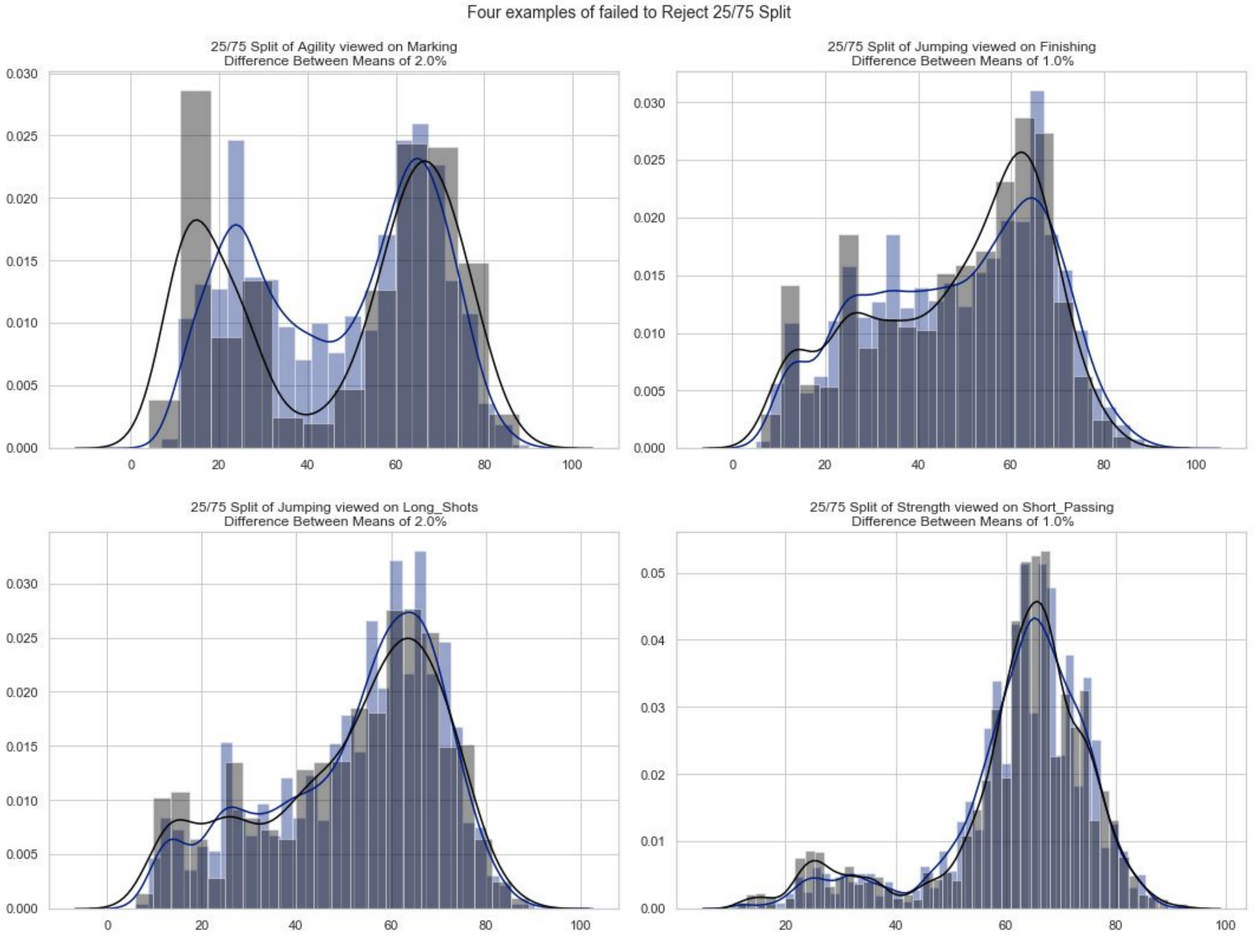


Difference between means of upper and lower groupings.

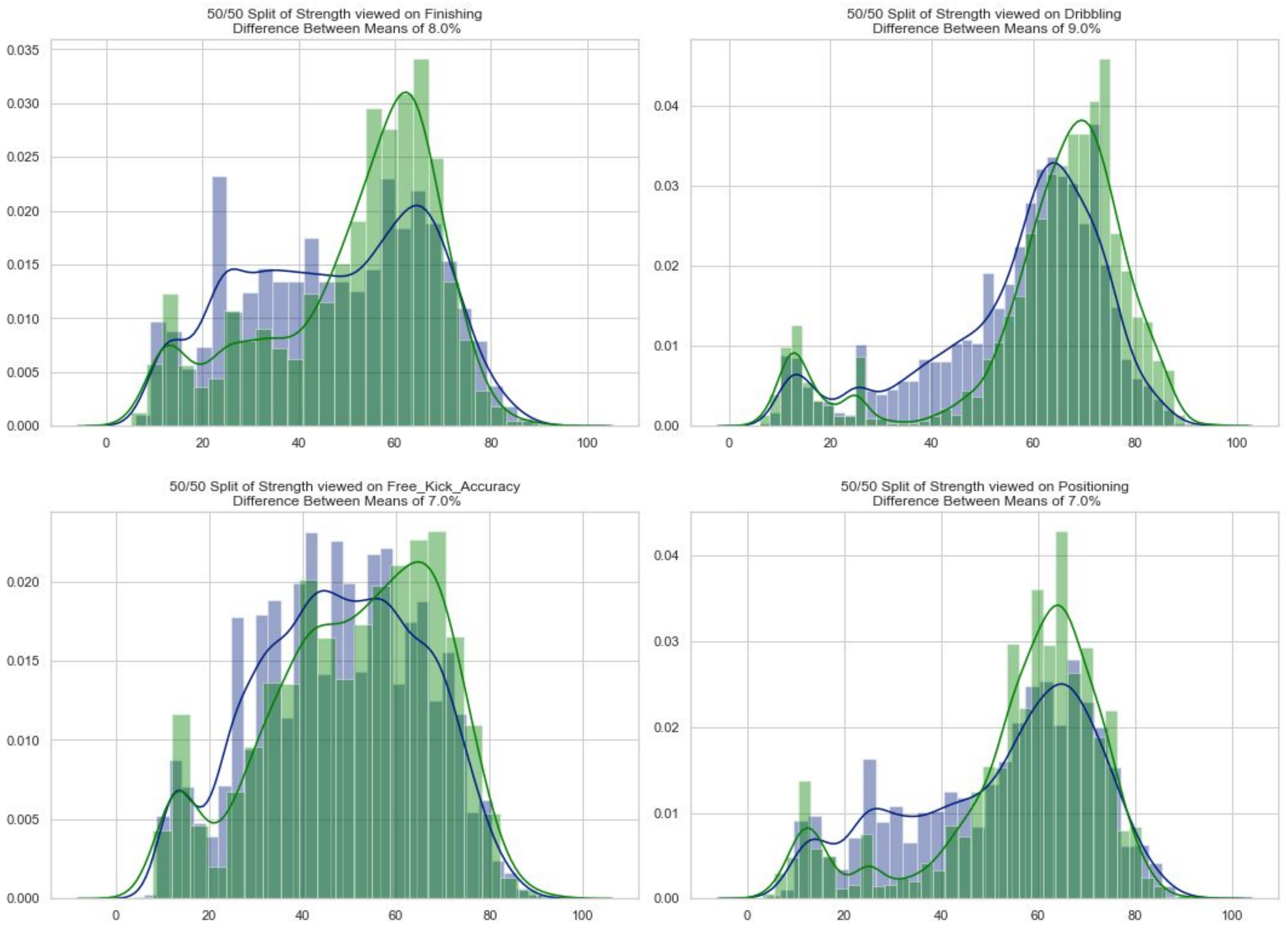


50/50 Split on Athletic Measures for Each Skill
Alpha = .05, Two Sided T-Test P-Values
50/50 Split Difference Between Means
Skill
crossing
finishing
heading_accuracy
short_passing
dribbling
free_kick_accuracy
long_passing
ball_control
long_shots
aggression
interceptions
positioning
penalties
marking
standing_tackle
sliding_tackle
acceleration
sprint_speed
agility
reactions
balance
jumping
stamina
strength
Athletic Measure

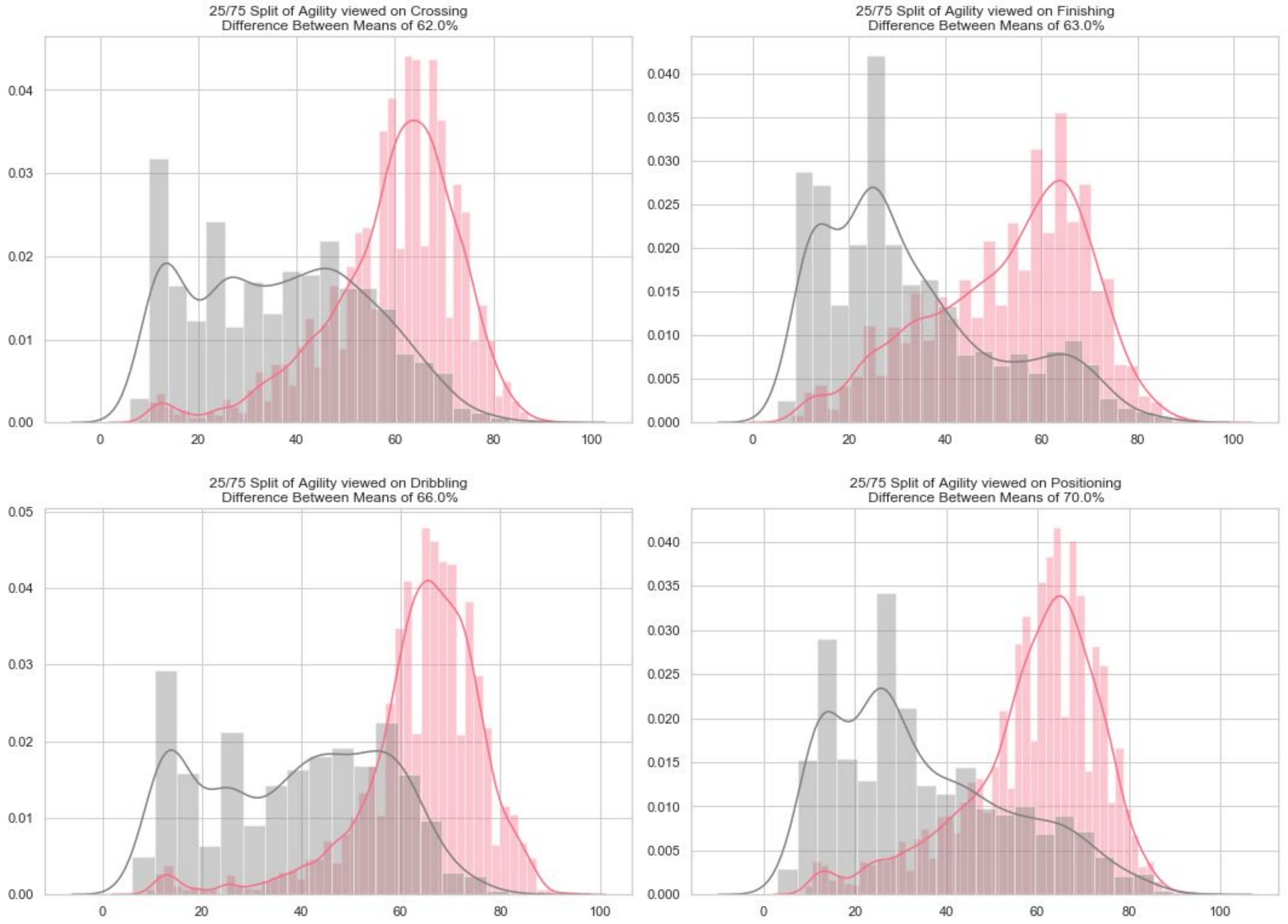

25/75 Split on Athletic Measures for Each Skill
Alpha = .05, Two Sided T-Test P-Values
crossing 0 0 0 0 0 4.5e-06 0 0
finishing 0 0 0 0 0 0.38 0 0
heading_accuracy 0 0 0 0 0 0 0 0
short_passing 0 0 0 0 0 0 0 0.034
dribbling 0 0 0 0 0 5.1e-06 0 0
free_kick_accuracy 0 0 0 0 0 0.021 0 0
long_passing 0 0 0 0 0 2.1e-06 0 0.0024
ball_control 0 0 0 0 0 0 0 0.00056
long_shots 0 0 0 0 0 0.044 0 4.6e-05
aggression 0 0 0 0 0 0 0 0
interceptions 0 0 0 0 0 0 0 0
positioning 0 0 0 0 0 3.4e-05 0 0
penalties 0 0 0 0 0 0.0042 0 0.011
marking 0 0 0.084 0 0 0 0 0
standing_tackle 0 0 2.8e-06 0 0 0 0 0
sliding_tackle 0 0 0 0 0 0 0 0
Skill
acceleration
sprint_speed
agility
reactions
balance
jumping
stamina
strength
Athletic Measure
25/75 Split Difference Between Means
crossing 20 20 23 9.6 21 1.8 19 -3.3
finishing 18 18 21 8.5 16 0.38 14 -4
heading_accuracy 7.9 9.7 5.9 5.9 4.4 7.4 16 13
short_passing 14 14 16 9.1 14 1.9 16 0.68
dribbling 24 24 25 8.7 22 1.8 20 -5.4
free_kick_accuracy 14 14 18 10 17 -0.94 14 -3.3
long_passing 11 11 14 9.1 13 1.6 15 1
ball_control 19 19 20 9.5 17 2.4 18 -1.2
long_shots 18 17 21 11 18 0.85 17 -1.8
aggression 5.6 6.6 4.9 8.6 5.2 7.5 16 13
interceptions 3.7 4.2 2.6 7.2 5.3 6.5 16 12
positioning 21 21 24 10 20 1.8 18 -4.1
penalties 13 13 15 7.6 13 1 12 -0.95
marking 3.2 4.1 0.87 4 4 7 17 13
standing_tackle 4.7 5.6 2.4 4.8 5.3 6.8 18 13
sliding_tackle 5.1 5.8 2.7 4.2 5.7 7.4 18 13
acceleration
sprint_speed
agility
reactions
balance
jumping
stamina
strength
Athletic Measure


Four examples of failed to Reject 25/75 Split
25/75 Split of Agility viewed on Marking
Difference Between Means of 2.0%
25/75 Split of Jumping viewed on Finishing
Difference Between Means of 1.0%
25/75 Split of Jumping viewed on Long_Shots
Difference Between Means of 2.0%
25/75 Split of Strength viewed on Short_Passing
Difference Between Means of 1.0%

Reject 25/75 Split AND Lower Group (green) has at least 6.5% Higher Mean (top 4 highest mean differences)



Reject 25/75 Split AND Upper Group (pink) has at least 61% Greater Mean (top 4 highest mean differences)



Harrison's part of the presentation

Works consulted:

https://machinelearningmastery.com/how-to-code-the-students-t-test-from-scratch-in-python/

https://www.sagepub.com/sites/default/files/upm-binaries/33663_Chapter4.pdf

https://www.kaggle.com/efezinoerome/analyzing-soccer-data

http://www.statstutor.ac.uk/resources/uploaded/tutorsquickguidetostatistics.pdf

https://math.stackexchange.com/questions/2173385/semantics-binomial-vs-binary

https://towardsdatascience.com/hypothesis-testing-in-the-northwind-dataset-using-anova-db3ab16b5eba

https://www.quora.com/What-does-a-high-F-value-usually-mean-and-why

Is there a statistical difference in the odds of winning a game when a team is playing in front of their home crowd?

Special thanks to Joe for help with this

```
In [335]: df2['HomeWin']=df2.home_team_goal>df2.away_team_goal
          df2['AwayWin']=df2.away_team_goal>df2.home_team_goal
```

```
In [337]: #As shown below, the effect size of playing home/away appears to be substantial:

          #Percentage of wins at home (across the dataset)
          print(df2.HomeWin.sum()/df2.HomeWin.shape[0])
          #Percentage of wins across the dataset when away
          print(df2.AwayWin.sum()/df2.AwayWin.shape[0])

          #This will be confirmed with statistical testing.

          0.45871665576042187
          0.28738596558759
```

In [331]:
```python
def homewinbinary(df):
    win_dict={}
    games_home = df2.groupby(df.home_team_api_id) #slice by home id
    games_away = df2.groupby(df.away_team_api_id) #slice by away id
    team_ids = list(games_home.groups.keys()) #get individual team ids
    #calculate and store home win percentages
    for team in team_ids:
        x=games_home.get_group(team) #grab home wins
        y=games_away.get_group(team) #grab everything else

        home_per=x.HomeWin.sum()/len(x.HomeWin) #calculate Home win percentage
        else_per=(1 - home_per) #calculate complement of Home win percentage to include draws as well

        win_dict[team]=[home_per, else_per] #store

    win_df=pd.DataFrame(win_dict).T #Transpose DF to have teams as rows
    return win_df
binary = homewinbinary(df2)
binary.head()
```

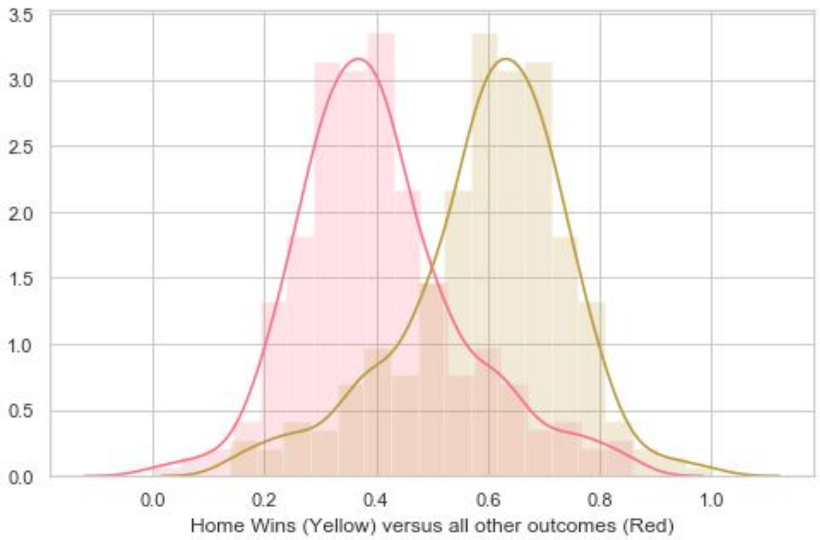
Out[331]:

	0	1
1601	0.450000	0.550000
1773	0.355556	0.644444
1957	0.525000	0.475000
2033	0.253333	0.746667
2182	0.616667	0.383333

Effect size of home-field advantage

In [338]:

```
#Plotting the distributions for Home Win Binary metric
plt.figure(figsize=(8,5))
for skill in binary.columns:
    sns.distplot((binary[skill]), hist_kws=dict(alpha=0.2))
    plt.xlabel('Home Wins (Yellow) versus all other outcomes (Red)')
```



Not a groundbreaking revelation, but still a valid real-world application of statistics, and good practice. I accept my alternative hypothesis.

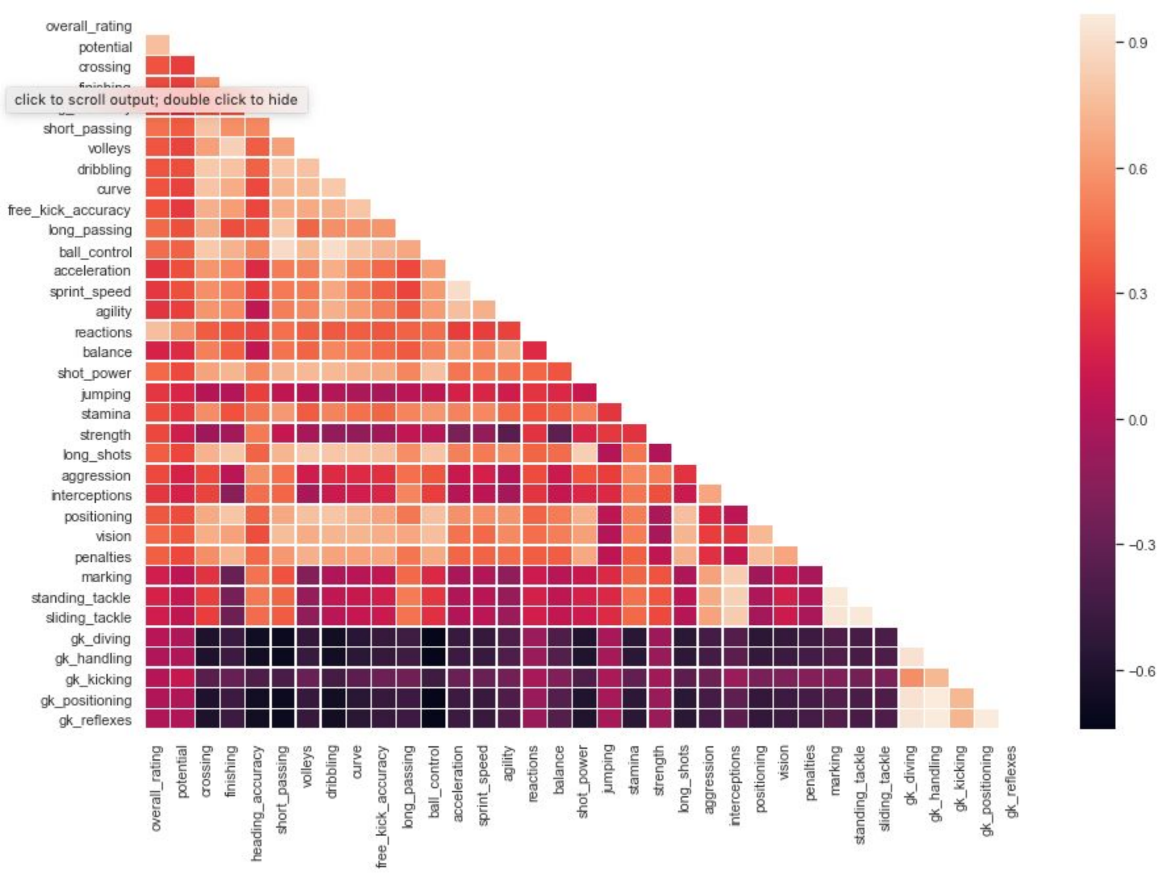
```
In [333]: #Run a dependent Ttest with Stats Model
          x = binary[0] #All other results
          y = binary[1] #Wins at home
          ttest=stats.ttest_rel(x,y)
          print(' ')
          print(f'The p-value for this dependent T-test is {ttest[1]}')
          print('Therefore, the null hypothesis that playing at home does not have a statistically significant effect on winning'
          print(f"was {('rejected' if ttest[1]<=.05 else 'not rejected')} at the .95 level of confidence.")
          print(' ')

          The p-value for this dependent T-test is 9.70767116964689e-25
          Therefore, the null hypothesis that playing at home does not have a statistically significant effect on winning
          was rejected at the .95 level of confidence.
```

High Cohen's D value, large effect size

In [300]:
```python
def Cohen_d(group1, group2):
    diff = group1.mean() - group2.mean()
    n1, n2 = len(group1), len(group2)
    var1 = group1.var()
    var2 = group2.var()
    # Calculate the pooled variance
    pooled_var = (n1 * var1 + n2 * var2) / (n1 + n2)
    # Calculate Cohen's d statistic
    d = diff / np.sqrt(pooled_var)
    return d
#The two groups under investigation here have given a Cohen's D greater than 0.8.
#Therefore, playing a game at home is considered to have large effect size on your odds of victory
Cohen_d(y, x)
```

Out[300]:
```
1.3022709371161278
```

Inquiries into the Player Attributes database: efforts at insight from ANOVA



Although high F-values will lead to low P-values and generally indicate a good predictor of the response, #these F-values are so large as to reduce the P-value to 0, and don't offer insight here.

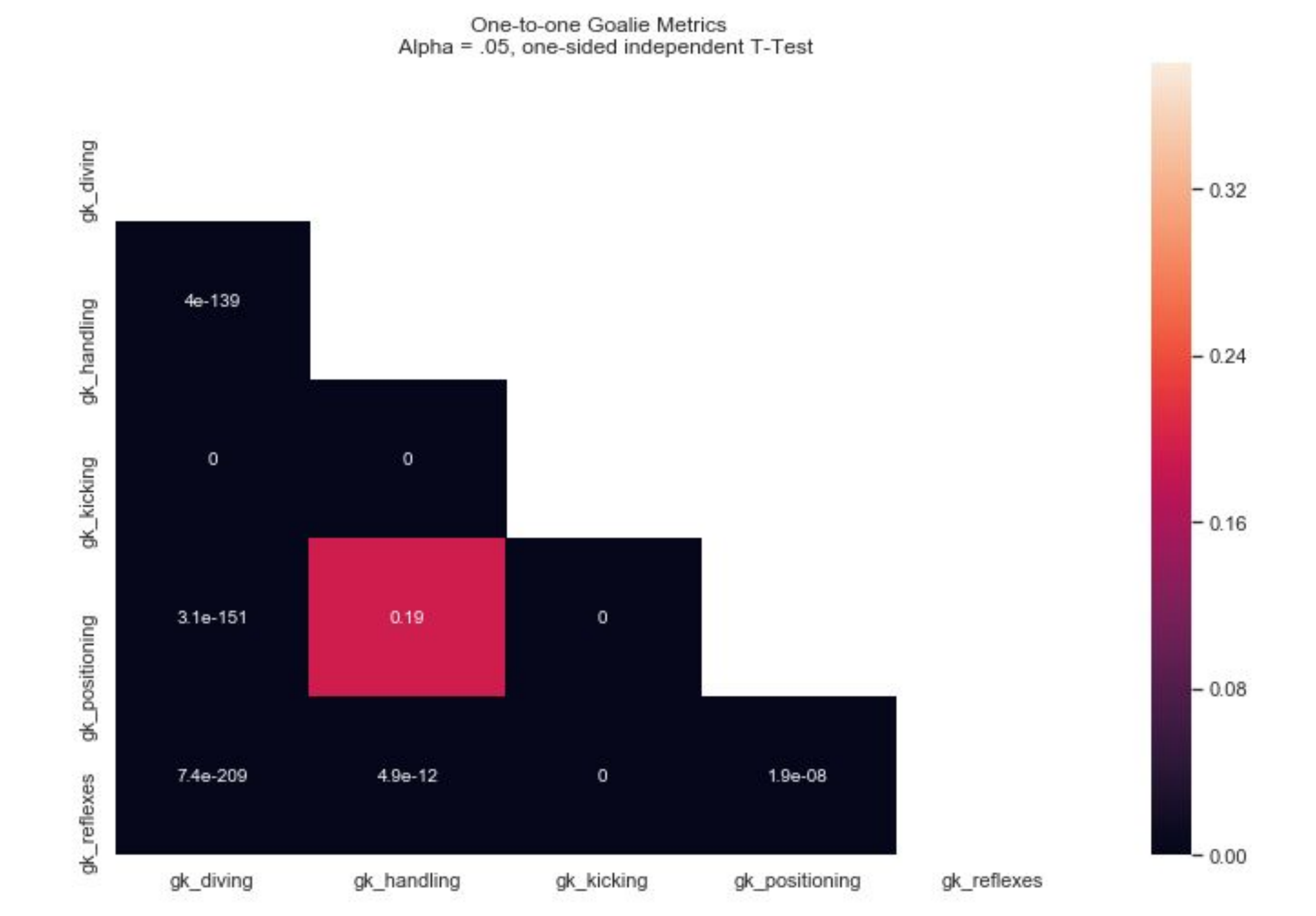
```
strength ~ sprint_speed + acceleration
                    sum_sq        df              F  PR(>F)
sprint_speed  9.251548e+05       1.0    6880.867712     0.0
acceleration  1.757729e+06       1.0   13073.161155     0.0
Residual      2.462363e+07  183139.0            NaN     NaN
sprint_speed ~ strength + acceleration
                    sum_sq        df               F  PR(>F)
strength      1.905996e+05       1.0     6880.867712     0.0
acceleration  2.352731e+07       1.0   849363.366244     0.0
Residual      5.072939e+06  183139.0             NaN     NaN
acceleration ~ strength + sprint_speed
                    sum_sq        df               F  PR(>F)
strength      3.741563e+05       1.0    13073.161155     0.0
sprint_speed  2.430894e+07       1.0   849363.366244     0.0
Residual      5.241473e+06  183139.0             NaN     NaN
```

```
gk_diving ~ gk_reflexes + ball_control
                    sum_sq        df               F  PR(>F)
gk_reflexes   1.725937e+07       1.0   506868.848760     0.0
ball_control  2.956282e+05       1.0     8681.933974     0.0
Residual      6.236059e+06  183139.0             NaN     NaN
gk_reflexes ~ gk_diving + ball_control
                    sum_sq        df               F  PR(>F)
gk_diving     1.812554e+07       1.0   506868.848760     0.0
ball_control  2.428963e+05       1.0     6792.434757     0.0
Residual      6.549020e+06  183139.0             NaN     NaN
ball_control ~ gk_diving + gk_reflexes
                   sum_sq        df            F  PR(>F)
gk_diving    8.719733e+05       1.0  8681.933974     0.0
gk_reflexes  6.822007e+05       1.0  6792.434757     0.0
Residual     1.839363e+07  183139.0          NaN     NaN
```

Independent T-testing of goalkeeper metrics

In [ ]:
```
#Pair each Goal-keeping metric with each other one in an independent Ttest
ttest_result_dict = {}
for skill in df1c.columns:
    for skill_2 in df1c.columns:
        x = df1c[skill]
        y = df1c[skill_2]
        ttest = stats.ttest_ind(x,y)
        ttest_name = skill+' and ' + skill_2
        ttest_result_dict.update({ttest_name : ttest})

#Pair each log-transformed Goal-keeping metric with each other one in an independent Ttest
ttest_result_dict1 = {}
for skill in df1c_log.columns:
    for skill_2 in df1c_log.columns:
        x = df1c_log[skill]
        y = df1c_log[skill_2]
        ttest = stats.ttest_ind(x,y)
        ttest_name = skill+' and ' + skill_2
        ttest_result_dict1.update({ttest_name : ttest})

# Create list of statistical values
ttest_list = list(ttest_result_dict.values())
ttest_list1 = list(ttest_result_dict1.values())

#Created a list of only p-values
p = [ttest_list[i][1] for i in range(len(ttest_list))]
p1 = [ttest_list1[i][1] for i in range(len(ttest_list1))]

#Sliced up list into appropriate sub-lists for each column
p_diving, p_handling, p_kicking, p_positioning, p_reflexes = p[0:5], p[5:10], p[10:15], p[15:20], p[20:25]
p_vals = [p_diving, p_handling, p_kicking, p_positioning, p_reflexes]
p_diving, p_handling, p_kicking, p_positioning, p_reflexes = p1[0:5], p1[5:10], p1[10:15], p1[15:20], p1[20:25]
p_valslog = [p_diving, p_handling, p_kicking, p_positioning, p_reflexes]
```

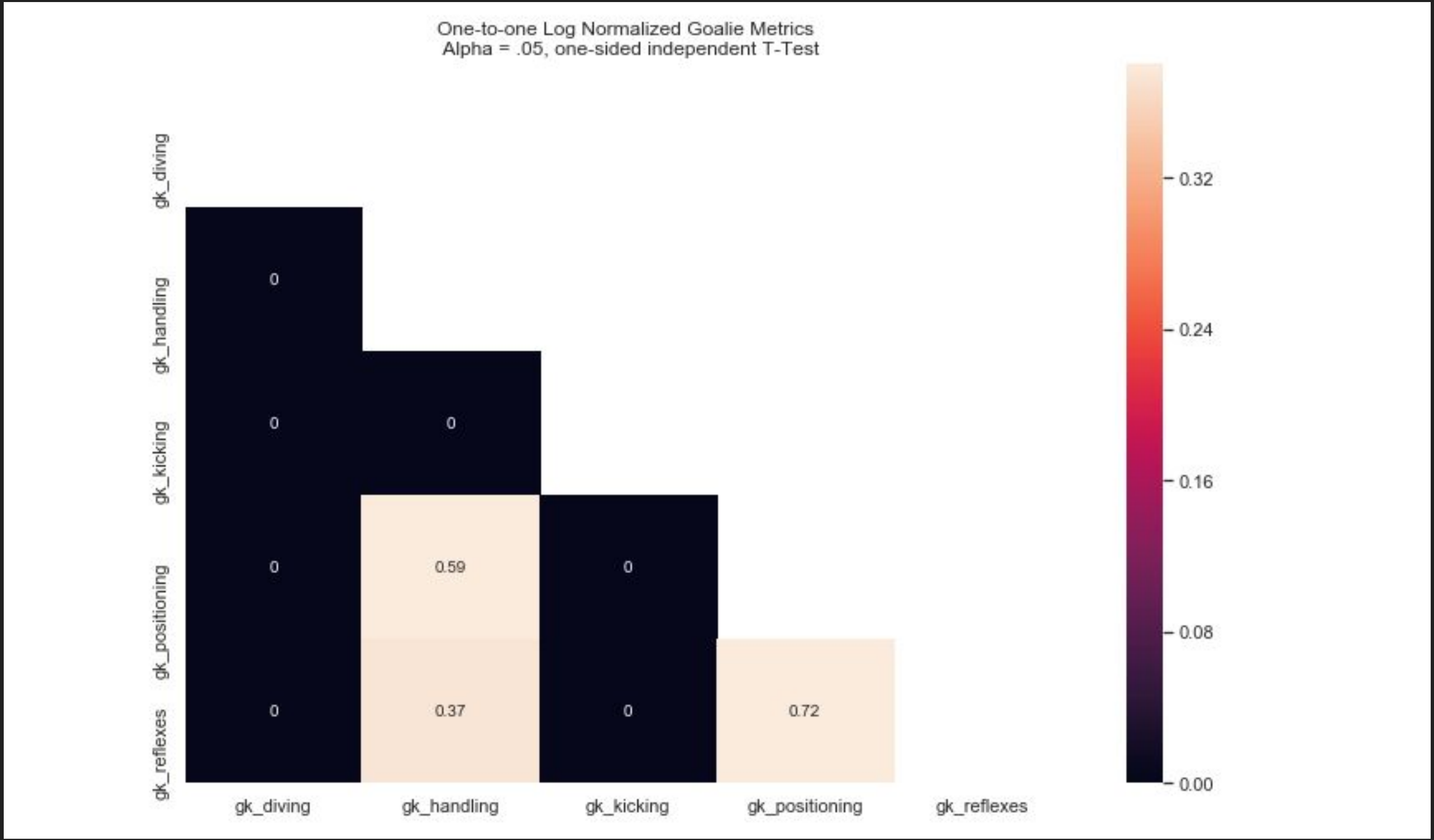
```python
# Create empty dataframe out of goal-keeper metrics
goalie_df = df1c.loc[:,'gk_diving':'gk_reflexes'].drop(df1c.index[0:foo2.shape[0]])
goalie_df['Index_'] = df1c.loc[:,'gk_diving':'gk_reflexes'].columns
goalie_df = goalie_df.set_index('Index_')
None

# Create empty dataframe out of log-transformed goal-keeper metrics
goalie_dflog = df1c_log.loc[:,'gk_diving':'gk_reflexes'].drop(df1c_log.index[0:df1c_log.shape[0]])
goalie_dflog['Index_'] = df1c_log.loc[:,'gk_diving':'gk_reflexes'].columns
goalie_dflog = goalie_dflog.set_index('Index_')
None

#Set columns for normal and log-transformed equal to respective p_values:

#index for p_vals
p_val = 0
#looping through columns
for column in goalie_df.columns:
    goalie_df[column] = p_vals[p_val]
    #increasing index
    p_val += 1
#repeat for log values
p_val1 = 0
for column in goalie_dflog.columns:
    goalie_dflog[column] = p_valslog[p_val1]
    p_val1 += 1
```



One-to-one Goalie Metrics
Alpha = .05, one-sided independent T-Test
gk_diving
gk_handling
gk_kicking
gk_positioning
gk_reflexes
4e-139
0
0
3.1e-151
0.19
0
7.4e-209
4.9e-12
0
1.9e-08
gk_diving
gk_handling
gk_kicking
gk_positioning
gk_reflexes
0.32
0.24
0.16
0.08
0.00

```
In [352]: for item in ttest_result_dict:
              print(' ')
              print(f'The p-value for this independent T-test is {ttest_result_dict[item][1]}')
              print("Therefore, the null hypothesis that FIFA's goalkeeper metrics")
              print("%s are independent of one another"%(item))
              print("was {result} at the .95 level of confidence.".format(result ='rejected' if ttest_result_dict[item][1]
                                                                          <=.05 else 'not rejected'))
              print(' ')
```

```
The p-value for this independent T-test is 0.1944046543429177
Therefore, the null hypothesis that FIFA's goalkeeper metrics
gk_handling and gk_positioning are independent of one another
was not rejected at the .95 level of confidence.


The p-value for this independent T-test is 4.856063130088822e-12
Therefore, the null hypothesis that FIFA's goalkeeper metrics
gk_handling and gk_reflexes are independent of one another
was rejected at the .95 level of confidence.
```


One-to-one Log Normalized Goalie Metrics
Alpha = .05, one-sided independent T-Test
gk_diving
gk_handling
gk_kicking
gk_positioning
gk_reflexes
0
0
0
0
0
0
0.59
0.37
0
0
0.72
0.32
0.24
0.16
0.08
0.00

In [346]:
```
for item in ttest_result_dict1:
    print(' ')
    print(f'The p-value for this independent T-test is {ttest_result_dict1[item][1]}')
    print("Therefore, the null hypothesis that FIFA's log-transformed goalkeeper metrics of")
    print("%s are independent of one another"%(item))
    print("was {result} at the .95 level of confidence.".format(result ='rejected' if ttest_result_dict1[item][1]
                                                                    <=.05 else 'not rejected'))
    print(' ')
```

```
The p-value for this independent T-test is 0.5938895749600349
Therefore, the null hypothesis that FIFA's log-transformed goalkeeper metrics of
gk_handling and gk_positioning are independent of one another
was not rejected at the .95 level of confidence.


The p-value for this independent T-test is 0.37261020623150465
Therefore, the null hypothesis that FIFA's log-transformed goalkeeper metrics of
gk_handling and gk_reflexes are independent of one another
was not rejected at the .95 level of confidence.
```