League of Legends Role Analysis and Prediction

Name(s): Gihyeon Kwon

Website Link: https://gikwon.github.io/LeagueofLegends-role-Analysis-Prediction/

In [1]:
```
import pandas as pd
import numpy as np
from pathlib import Path

import plotly.express as px
pd.options.plotting.backend = 'plotly'

from scipy import stats
from sklearn.preprocessing import StandardScaler, QuantileTransformer, Label
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, Paramete
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
import warnings

warnings.filterwarnings('ignore', category=pd.errors.DtypeWarning)

pd.set_option('display.max_columns', 35)
pd.set_option('display.max_rows', 25)
```

Step 1: Introduction

Dataset of league of legends games played in the league.

Questions of focus:

- Which role “carries” (is the key position) in their team more often: ADCs (Bot lanes) or Mid laners?

Step 2: Data Cleaning and Exploratory Data Analysis

Read in csv file and dropping unnecessary columns and rows

- here I got rid of the rows that correspond to teams as they won't be useful in our analysis
- Also got rid of rows that are partially complete or with 'ignore' value in datacompleteness column

In [2]:
```
df = pd.read_csv("Data/2022_LoL_esports_match_data_from_OraclesElixir.csv")

df = df.drop(columns=[
    'url', 'league', 'year', 'split', 'playoffs',
    'date', 'ban1', 'ban2', 'ban3', 'ban4', 'ban5',
    'pick1', 'pick2', 'pick3', 'pick4', 'pick5'])

df = df[df['playername'].notna()]
```

In [3]:
```
original_columns = df.columns
```

In [4]:
```
df.datacompleteness.value_counts()
```

Out[4]:
```
complete    105530
partial      18190
ignore         440
Name: datacompleteness, dtype: int64
```

In [5]:
```
df = df[df['datacompleteness'] == 'complete']
```

In [6]:
```
df.head()
```

Out[6]:

	gameid	datacompleteness	game	patch	participantid	side	po
0	ESPORTSTMNT01_2690210	complete	1	12.01	1	Blue	
1	ESPORTSTMNT01_2690210	complete	1	12.01	2	Blue	
2	ESPORTSTMNT01_2690210	complete	1	12.01	3	Blue	
3	ESPORTSTMNT01_2690210	complete	1	12.01	4	Blue	
4	ESPORTSTMNT01_2690210	complete	1	12.01	5	Blue	

5 rows × 115 columns

Chaning results to True if won, False if lost

In [7]:
```
df.result = df.result.apply(lambda x: x == 1)
```

Making new variables that might be useful

- KD = Kills / (death + 1)
- KDA = (kills + assists)/ (deaths + 1)

In [8]:
```
df['gamelength_minutes'] = df['gamelength'] / 60

df['kills_per_minute'] = df['kills'] / df['gamelength_minutes']
df['assists_per_minute'] = df['assists'] / df['gamelength_minutes']
df['deaths_per_minute'] = df['deaths'] / df['gamelength_minutes']

df['kd_ratio'] = df['kills'] / (df['deaths'] + 1)
df['kda_ratio'] = (df['kills'] + df['assists']) / (df['deaths'] + 1)
```

Important note: some metric will be pointless in testing as they should be relative to the game. For example, a high kill from a mid is pointless if everyone else in the team had higher or similar kill count.

So we need to create metrics that are normalized

In [9]:
```
df['kill_participation'] = (df['kills'] + df['assists']) / (df['teamkills']


df['team_kda'] = df.groupby(['gameid', 'side'])['kda_ratio'].transform('mear
df['team_kd'] = df.groupby(['gameid', 'side'])['kd_ratio'].transform('mean')


df['kda_normal'] = df['kda_ratio'] / (df['team_kda'] + 1)
df['kd_normal'] = df['kd_ratio'] / (df['team_kda'] + 1)
```

In [10]:
```
df.head(10)
```

Out[10]:

	gameid	datacompleteness	game	patch	participantid	side	po:
0	ESPORTSTMNT01_2690210	complete	1	12.01	1	Blue	
1	ESPORTSTMNT01_2690210	complete	1	12.01	2	Blue	
2	ESPORTSTMNT01_2690210	complete	1	12.01	3	Blue	
3	ESPORTSTMNT01_2690210	complete	1	12.01	4	Blue	
4	ESPORTSTMNT01_2690210	complete	1	12.01	5	Blue	
5	ESPORTSTMNT01_2690210	complete	1	12.01	6	Red	
6	ESPORTSTMNT01_2690210	complete	1	12.01	7	Red	
7	ESPORTSTMNT01_2690210	complete	1	12.01	8	Red	
8	ESPORTSTMNT01_2690210	complete	1	12.01	9	Red	
9	ESPORTSTMNT01_2690210	complete	1	12.01	10	Red	

10 rows × 126 columns

Identifiers:

- `position`
- `gameid`

Useful variables:

- `gamelength`
- `gamelegnth_minutes`

Metrics that are relative to the team within game:

- `kill_participation`

- `kd_normal`
- `kda_normal`
- `damageshare`
- `earnedgoldshare`

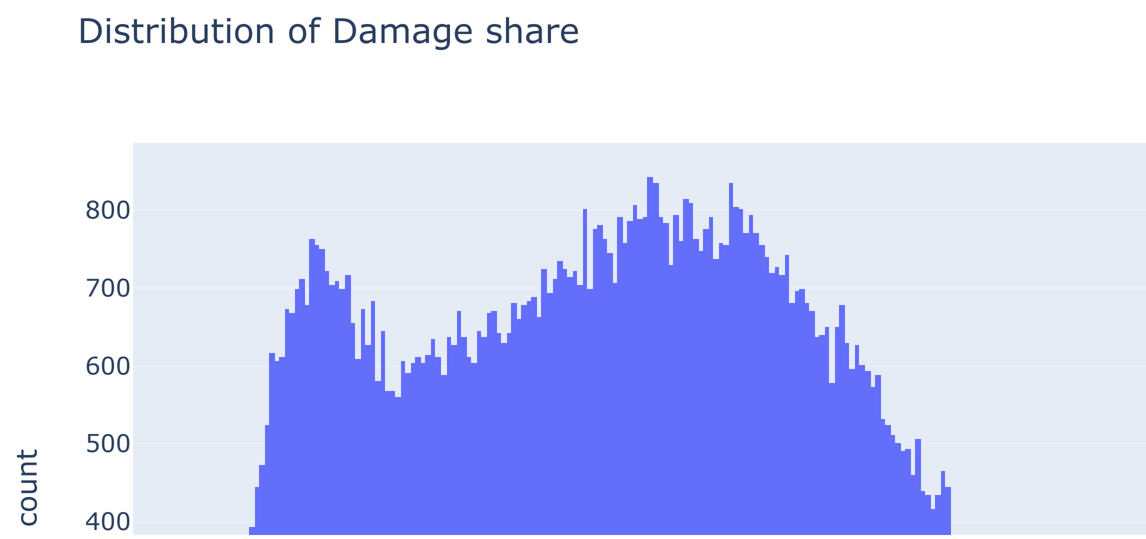
Some columns that contributes to 'carrying' the game that we may need to consider:

- `kills_per_minute`
- `assists_per_minute`
- `deaths_per_minute`
- `kills`
- `deaths`
- `assists`
- `doublekills`
- `triplekills`
- `quadrakills`
- `pentakills`
- `firstbloodkill`
- `firstbloodassist`
- `dpm`
- `earned gpm`
- `cspm`

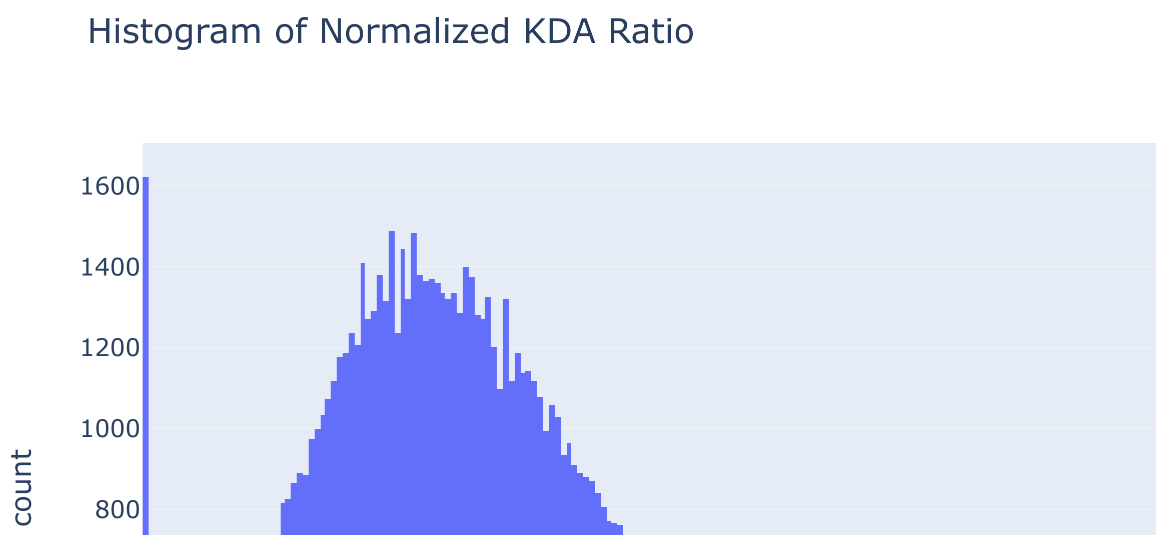
Univariate Analysis

In [11]:
```
mean_damageshare = df.damageshare.mean()
median_damageshare = df.damageshare.median()
```

In [12]:
```
fig = px.histogram(df, x='damageshare', title='Distribution of Damage share'

fig.update_layout(
    xaxis_title='Damage Share')
fig.show()

fig.write_html('LeagueofLegends-role-Analysis-Prediction/assets/hist_dmg.htm
```

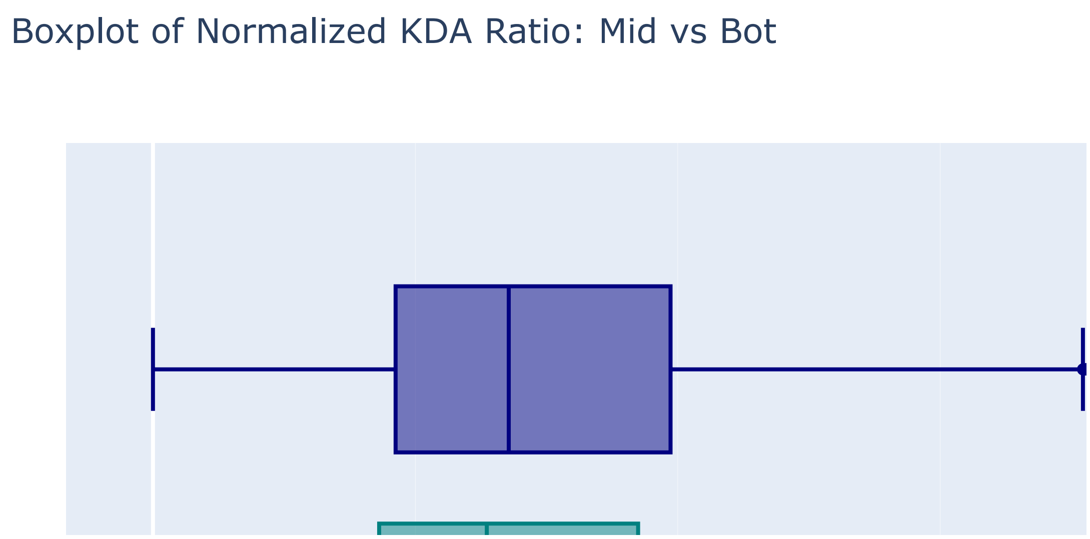


In [13]:
```
fig = px.histogram(df, x='kda_normal', title='Histogram of Normalized KDA Ra
fig.update_layout(
    xaxis_title='Normalized KDA')
fig.show()
```

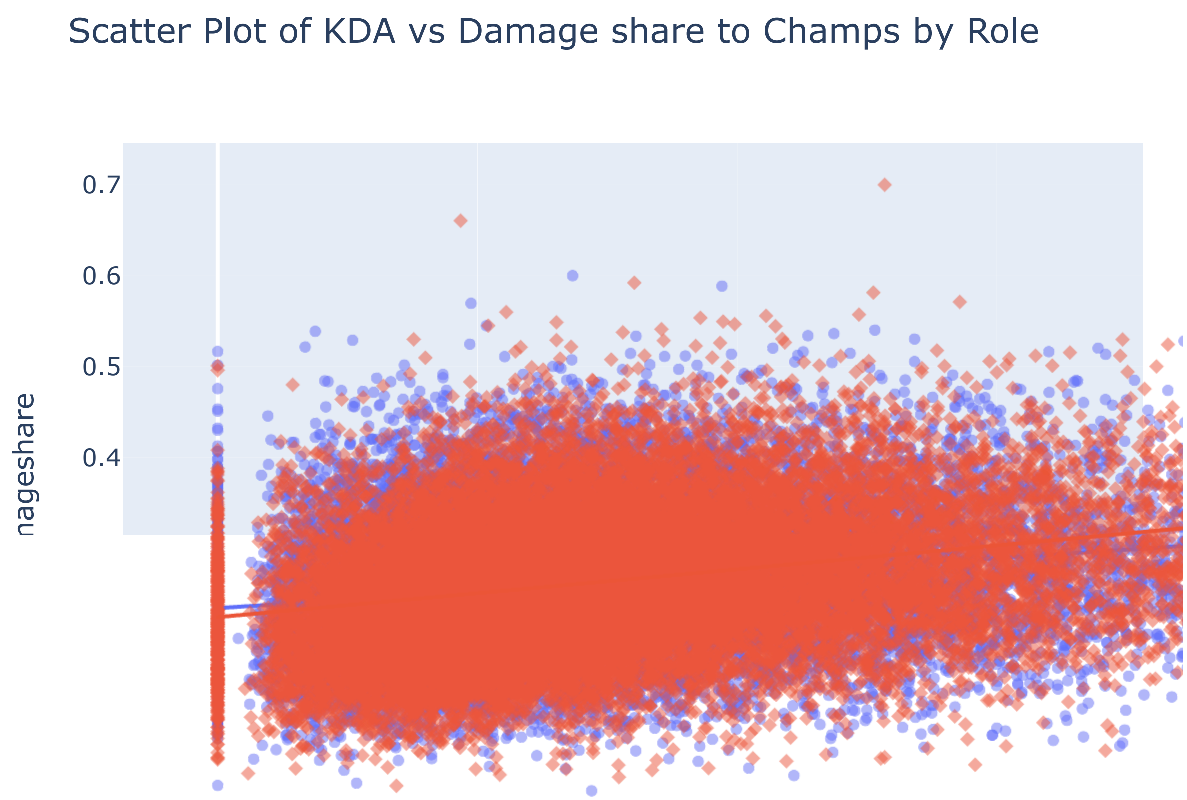


Bivariate Analysis

In [14]:
```
df_mid_bot = df[(df['position'] == 'mid') | (df['position'] == 'bot')]

fig = px.box(df_mid_bot, x='kda_normal', color= 'position', title='Boxplot c
            , color_discrete_map={
        'mid': 'teal',
        'bot': 'navy'
    })


fig.update_layout(
    xaxis_title='Normalized KDA Ratio',
    xaxis=dict(
        tickmode='array',
        tickvals=[0, 0.5, 1, 1.5, 2, 2.5, 3], # example tick values
        ticktext=['0', '0.5', '1', '1.5', '2', '2.5','3'] # example tick tex
    )
)

fig.show()
fig.write_html('LeagueofLegends-role-Analysis-Prediction/assets/Boxplot_KDA.
```



In [15]:
```
fig = px.scatter(df_mid_bot, x='kda_normal', y='damageshare',
                  color='position', symbol='position',
                 opacity=0.5,title='Scatter Plot of KDA vs Damage share to Ch
                 trendline='ols')
fig.show()
```



Making a table using the columns above

In [16]:
```
grouped_df = df_mid_bot.groupby('position').agg({
    'kd_normal': ['mean', 'std'],
    'kda_normal': ['mean', 'std'],
    'kill_participation': ['mean', 'std'],
    'kills_per_minute': ['mean', 'std'],
    'assists_per_minute': ['mean', 'std'],
    'deaths_per_minute': ['mean', 'std'],
    'dpm': ['mean', 'std'],
    'damageshare': ['mean', 'std'],
    'earned gpm': ['mean', 'std'],
    'earnedgoldshare': ['mean', 'std'],
    'cspm': ['mean', 'std'],
    'doublekills': ['mean', 'std'],
    'triplekills': ['mean', 'std'],
    'quadrakills': ['mean', 'std'],
    'pentakills': ['mean', 'std'],
    'firstbloodkill': ['mean', 'std'],
    'firstbloodassist': ['mean', 'std'],
})
grouped_df
```

Out[16]:

	kd_normal		kda_normal		kill_participation		kills_per_mir
	mean	std	mean	std	mean	std	mean
position							
bot	0.340107	0.268792	0.761290	0.427727	0.593615	0.181750	0.135902	0.107
mid	0.279429	0.234002	0.723186	0.419920	0.581186	0.175269	0.113512	0.091

So far, all average values **except** `assists_per_minute` are higher for the bottom position

Step 3: Assessment of Missingness

In [17]:
```
df.head()
```

Out[17]:

	gameid	datacompleteness	game	patch	participantid	side	po:
0	ESPORTSTMNT01_2690210	complete	1	12.01	1	Blue	
1	ESPORTSTMNT01_2690210	complete	1	12.01	2	Blue	
2	ESPORTSTMNT01_2690210	complete	1	12.01	3	Blue	
3	ESPORTSTMNT01_2690210	complete	1	12.01	4	Blue	
4	ESPORTSTMNT01_2690210	complete	1	12.01	5	Blue	

5 rows × 126 columns

In [18]:
```
df.isna().mean().sort_values(ascending=False).head(20)
```

Out[18]:
```
dragons (type unknown)    1.0
opp_heralds               1.0
elementaldrakes           1.0
opp_elementaldrakes       1.0
infernals                 1.0
mountains                 1.0
clouds                    1.0
oceans                    1.0
chemtechs                 1.0
hextechs                  1.0
elders                    1.0
opp_elders                1.0
firstherald               1.0
heralds                   1.0
void_grubs                1.0
dragons                   1.0
opp_void_grubs            1.0
firstbaron                1.0
firsttower                1.0
towers                    1.0
dtype: float64
```

As you can see above, the columns related to dragon, barons, and other monsters all have high missingness and all equal, but these aren't good columns to choose as they are missing by design as I have only kept rows with players and only rows with teams have values to the monster columns

In [19]:
```
(
    df.isna().mean().sort_values(ascending=False)
    [
        (df.isna().mean().sort_values(ascending=False) < 1)
        &
        (df.isna().mean().sort_values(ascending=False) > 0)
    ]
)
```

Out[19]:
```
inhibitors        0.030986
opp_barons        0.030986
barons            0.030986
opp_inhibitors    0.030986
playerid          0.018478
teamid            0.014640
teamname          0.000426
dtype: float64
```

Besides the monster related columns, we will try the next highest `barons`

In [20]:
```
df[df['barons'].isna()].head()
```

Out[20]:

	gameid	datacompleteness	game	patch	participantid	side
3348	ESPORTSTMNT01_2692918	complete	1	12.01	1	Blue
3349	ESPORTSTMNT01_2692918	complete	1	12.01	2	Blue
3350	ESPORTSTMNT01_2692918	complete	1	12.01	3	Blue
3351	ESPORTSTMNT01_2692918	complete	1	12.01	4	Blue
3352	ESPORTSTMNT01_2692918	complete	1	12.01	5	Blue

5 rows × 126 columns

Permutation to test dependency

- Null: Missingness of `barons` column is not dependent of another column
- Alternative: Missingness of `barons` column is dependent of another column

Test statistics:

- K-S Test statistics
- Using significant value of 0.01

Using the original columns given. Testing on columns that aren't entirely null or that aren't relevant.

For ex `game` has mostly value of 1. Same reasoning for `patch` and etc..

In [21]:

```
numeric_columns = (
    df[original_columns]
    .dropna(axis=1, how='all')
    .drop(columns=['barons', 'game', 'patch', 'participantid'])
    .select_dtypes(
        include=['float64', 'int64'])
    .columns
)
```

In [22]:

```
out_p = {}

for col in numeric_columns:
    missing_barons = df.loc[df['barons'].isna(), col]
    not_missing_barons = df.loc[df['barons'].notna(), col]
    out_p[col] = stats.ks_2samp(missing_barons, not_missing_barons).pvalue
```

In [23]:

```
out_p
```

Out[23]:

```
{'gamelength': 6.919952734752243e-12,
 'kills': 0.5556161835632021,
 'deaths': 0.006625437014808867,
 'assists': 0.004913965448192037 5,
 'teamkills': 5.5404593661053514e-08,
 'teamdeaths': 2.018567905976759e-07,
 'doublekills': 0.316263812578203,
 'triplekills': 1.0,
 'quadrakills': 1.0,
 'pentakills': 1.0,
 'firstblood': 0.9997894364157264,
 'firstbloodkill': 1.0,
 'firstbloodassist': 0.9997894364157264,
 'firstbloodvictim': 1.0,
 'team kpm': 1.0777501238401189e-11,
 'ckpm': 1.837264076903715e-32,
 'opp_barons': 0.0,
 'inhibitors': 0.0,
 'opp_inhibitors': 0.0,
 'damagetochampions': 0.09341365749060293,
 'dpm': 0.0006030612150309036,
 'damageshare': 0.15413488727924707,
 'damagetakenperminute': 0.001749410789684966,
 'damagemitigatedperminute': 0.19923545978348567,
 'wardsplaced': 2.1488523620917156e-08,
 'wpm': 4.2344127944851895e-07,
 'wardskilled': 2.1628398185703224e-14,
 'wcpm': 1.4325618471423052e-12,
 'controlwardsbought': 0.00014292578595776867,
 'visionscore': 2.2189626150911087e-09,
 'vspm': 1.7763161136255017e-07,
 'totalgold': 0.0015864157132329814,
 'earnedgold': 0.007073551247012911,
 'earned gpm': 0.7010266369249963,
 'earnedgoldshare': 0.3009786572538081,
 'goldspent': 0.000719198787705227,
 'total cs': 3.2110452787532215e-07,
 'minionkills': 1.1021169068628645e-110,
 'monsterkills': 0.003961194685262007,
 'cspm': 4.7847118988872665e-231,
 'goldat10': 0.8121425703487315,
 'xpat10': 0.8345738515431601,
 'csat10': 0.014584363311214888,
 'opp_goldat10': 0.8121425703487315,
 'opp_xpat10': 0.8345738515431601,
 'opp_csat10': 0.014584363311214888,
 'golddiffat10': 0.9587697195058094,
 'xpdiffat10': 0.7823762046389593,
 'csdiffat10': 0.9980372994093832,
 'killsat10': 0.19167930819667733,
 'assistsat10': 0.006843626937468948,
 'deathsat10': 0.07096588226239353,
 'opp_killsat10': 0.19167930819667733,
 'opp_assistsat10': 0.006843626937468948,
 'opp_deathsat10': 0.07096588226239353,
 'goldat15': 0.7271671414869816,
```

```
 'xpat15': 0.11645246024572942,
 'csat15': 0.0006033966449787389,
 'opp_goldat15': 0.7271671414869816,
 'opp_xpat15': 0.11645246024572942,
 'opp_csat15': 0.0006033966449787389,
 'golddiffat15': 0.936078947525773,
 'xpdiffat15': 0.9985907333072083,
 'csdiffat15': 0.994139162681523,
 'killsat15': 0.09394189161687816,
 'assistsat15': 3.9396179191285176e-05,
 'deathsat15': 0.000728653266545243,
 'opp_killsat15': 0.09394189161687816,
 'opp_assistsat15': 3.9396179191285176e-05,
 'opp_deathsat15': 0.000728653266545243}
```

In [24]:
```python
p_df = pd.DataFrame.from_dict(out_p, orient='index', columns=['value'])
p_df.sort_values('value')
```

Out[24]:

	value
opp_inhibitors	0.000000e+00
inhibitors	0.000000e+00
opp_barons	0.000000e+00
cspm	4.784712e-231
minionkills	1.102117e-110
...	...
quadrakills	1.000000e+00
pentakills	1.000000e+00
firstbloodkill	1.000000e+00
firstbloodvictim	1.000000e+00
triplekills	1.000000e+00

70 rows × 1 columns

You see strong relation ship in the top three with p-value of 0, but the 3 columns and the `barons` columns are similar where when barons is missing, the three are also missing. So lets analyze a column that is not entirely correlated with `barons`

In [25]:
```python
p_df[p_df['value'] > 0].sort_values('value')
```

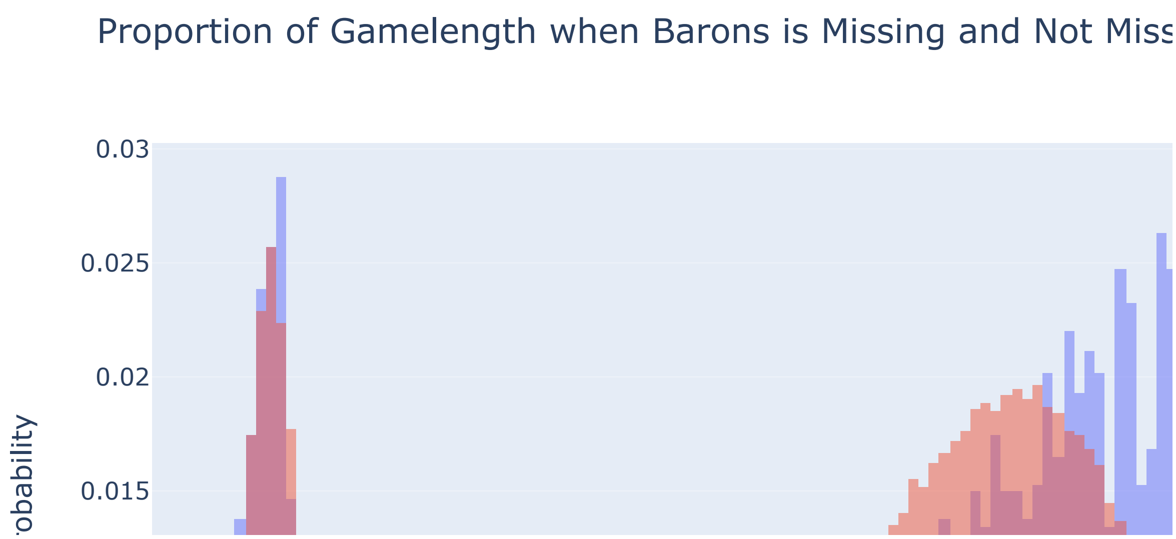
Out[25]:

	value
cspm	4.784712e-231
minionkills	1.102117e-110
ckpm	1.837264e-32
wardskilled	2.162840e-14
wcpm	1.432562e-12
...	...
quadrakills	1.000000e+00
firstbloodkill	1.000000e+00
triplekills	1.000000e+00
pentakills	1.000000e+00
firstbloodvictim	1.000000e+00

67 rows × 1 columns

- We see `cspm` has p value of 4.784712e-231, meaning there is strong evidence to reject the null
- Also `firstbloodvictim` has p value of 1, meaning there is strong evidence that fails to reject the null

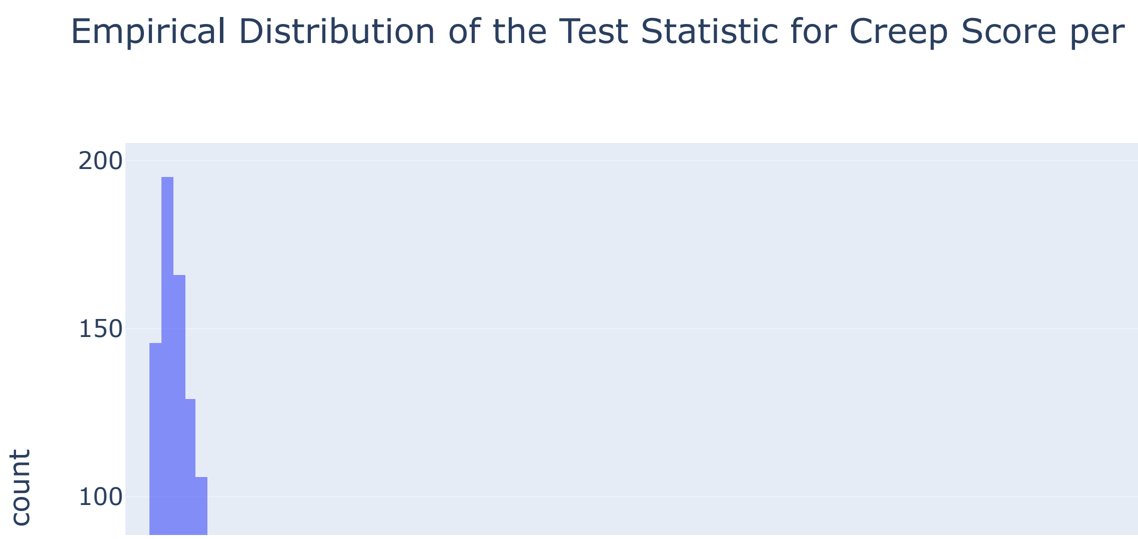
In [27]:
```
df_missing = df[df['barons'].isna()].copy()
df_not_missing = df[df['barons'].notna()].copy()

df_missing.loc[:,'missing'] = 'Missing'
df_not_missing.loc[:,'missing'] = 'Not Missing'
df_combined = pd.concat([df_missing, df_not_missing])

fig_gamelength_dist = px.histogram(df_combined, x='cspm', color='missing', b
                                   title="Proportion of Gamelength when Barc
                                   labels={'gamelength': 'Game Length', 'cou
                                    histnorm='probability')

fig_gamelength_dist.show()
```



In [28]:
```
missing_barons = df.loc[df['barons'].isna(), 'cspm']
not_missing_barons = df.loc[df['barons'].notna(), 'cspm']

ks_stats = []
combined_data = df[['cspm', 'barons']].copy()
combined_data['barons'] = combined_data['barons'].isna()
for _ in range(1000):

    permuted_labels = np.random.permutation(combined_data['barons'])
    permuted_missing = combined_data['cspm'][permuted_labels]
    permuted_not_missing = combined_data['cspm'][~permuted_labels]
    perm_stat = stats.ks_2samp(permuted_missing, permuted_not_missing).stati
    ks_stats.append(perm_stat)
```

In [29]:
```
observed_ks = stats.ks_2samp(df_missing['cspm'], df_not_missing['cspm']).sta

fig = px.histogram(pd.DataFrame(ks_stats), nbins=30, opacity=0.75, title="Em
fig.add_vline(x=observed_ks, line_color='red', line_width=1, opacity=1)

fig.add_annotation(x=observed_ks, y=max(np.histogram(ks_stats, bins=30)[0]),
fig.update_layout(
    xaxis_title="K-S Test Statistics for cspm",
```

```
    showlegend=False)

fig.show()

fig.write_html('LeagueofLegends-role-Analysis-Prediction/assets/cspm_ks.html
```



In [30]:
```
p_df.sort_values('value', ascending=False)
```

Out[30]:

	value
firstbloodkill	1.000000e+00
firstbloodvictim	1.000000e+00
triplekills	1.000000e+00
quadrakills	1.000000e+00
pentakills	1.000000e+00
...	...
minionkills	1.102117e-110
cspm	4.784712e-231
opp_barons	0.000000e+00
inhibitors	0.000000e+00
opp_inhibitors	0.000000e+00

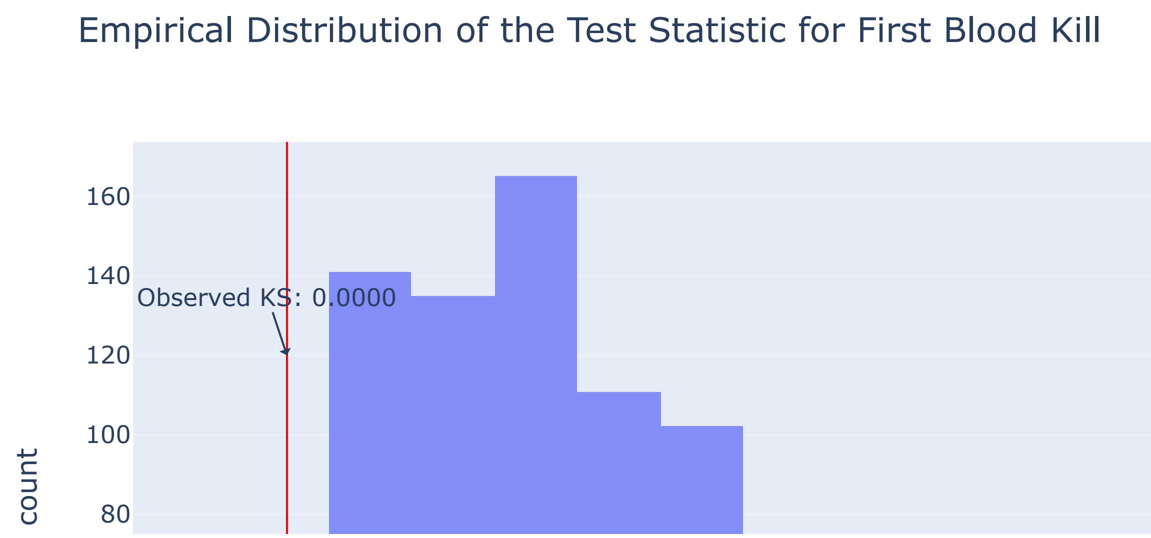
70 rows × 1 columns

And we see that `firstbloodkill` has a p_value of 1 which means we fail to reject the null. Meaning There is high change `barons` missingness is not dependent on `firstbloodkill`

In [31]:
```
missing_barons = df.loc[df['barons'].isna(), 'firstbloodkill']
not_missing_barons = df.loc[df['barons'].notna(), 'firstbloodkill']

ks_stats = []
combined_data = df[['firstbloodkill', 'barons']].copy()
combined_data['barons'] = combined_data['barons'].isna()
for _ in range(1000):

    permuted_labels = np.random.permutation(combined_data['barons'])
    permuted_missing = combined_data['firstbloodkill'][permuted_labels]
    permuted_not_missing = combined_data['firstbloodkill'][~permuted_labels]
    perm_stat = stats.ks_2samp(permuted_missing, permuted_not_missing).stati
    ks_stats.append(perm_stat)
```

In [32]:
```
observed_ks = stats.ks_2samp(df_missing['firstbloodkill'], df_not_missing['f

fig = px.histogram(pd.DataFrame(ks_stats), nbins=30, opacity=0.75, title="Em
fig.add_vline(x=observed_ks, line_color='red', line_width=1, opacity=1)

fig.add_annotation(x=observed_ks, y=max(np.histogram(ks_stats, bins=30)[0]),
fig.update_layout(
    xaxis_title="K-S Test Statistics for firstbloodkill",
    showlegend=False)

fig.show()
```

```
fig.write_html('LeagueofLegends-role-Analysis-Prediction/assets/firstbloodki
```



Step 4: Hypothesis Testing

Comparing the metrics between `position`'s `top` and `bot`

Choosing performance metrics:

- `kda_normal`
- `kd_normal`
- `kill_participation`
- `damangeshare`
- `earnedgoldshare`

But wait, we saw that the KDA and damageshare were bot good metrics

We will generate a new variable that is kda * damage share Sometimes KDA metric doesn't explain much, a player could last hit their enemies to steal kills, ending with high KDA but low damage.

In [33]:
```
df['kda_dmg'] = df['kda_normal'] * df['damageshare']
df_mid_bot = df[(df['position'] == 'bot') | (df['position'] == 'mid')]
```

Hypothesis testing:

- Null Hypothesis : There is no significant difference in the normalized KDA times Damage Share between ADCs and Mid laners.
- Alternative Hypothesis: The KDA times Damage Share is higher for bot position than the KDA times Damage Share for mids.

One sided test:

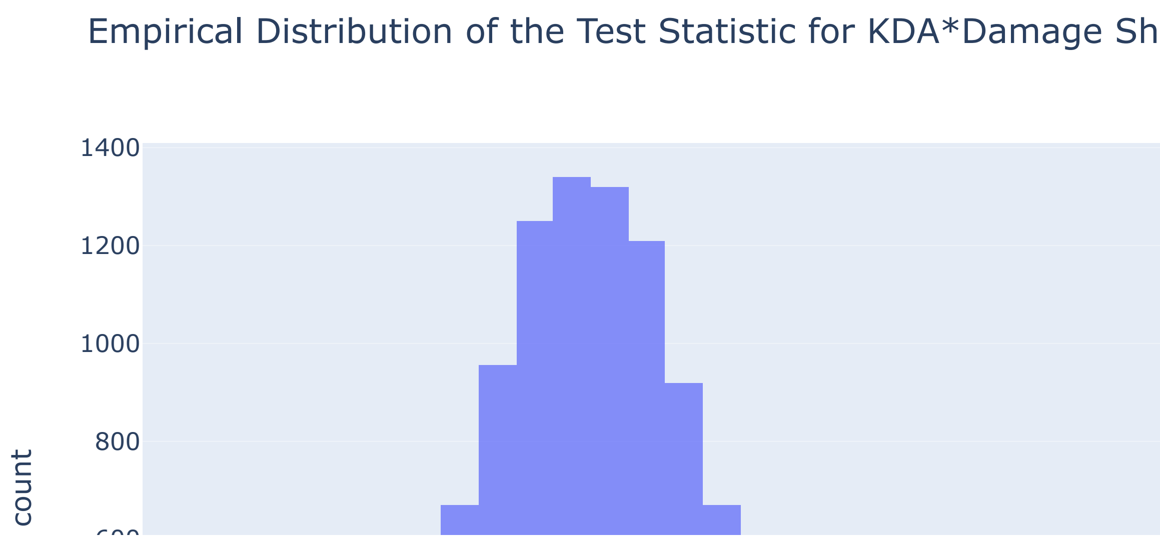
- Test Statistics: Difference in means
- Using significant value of 0.01

In [34]:
```
role_df = df_mid_bot[['position', 'kda_dmg']].copy()

observed_df = role_df.groupby('position')['kda_dmg'].mean()
observed = observed_df['bot'] - observed_df['mid']

stats = []
n = 10000
for i in range(n):
    shuffled_positions = np.random.permutation(role_df['position'])

    role_df['shuffled'] = shuffled_positions

    stat_df = role_df.groupby('shuffled')['kda_dmg'].mean()
    stat = stat_df['bot'] - stat_df['mid']
    stats.append(stat)

p_value = (stats >= observed).mean()
```

In [35]:
```
print(f"Observed Statistic is : {observed}")
print(f"P-value is: {p_value}")
```

```
Observed Statistic is : 0.015782391324528494
P-value is: 0.0
```

In [36]:
```
fig = px.histogram(pd.DataFrame(stats), nbins=30, opacity=0.75, title="Empir
fig.add_vline(x=observed, line_color='red', line_width=1, opacity=1)

fig.add_annotation(x=observed, y=max(np.histogram(stats, bins=30)[0]), text=
fig.update_layout(
    xaxis_title="Test Statistics for KDA * Damage Share",
    showlegend=False)

fig.show()

fig.write_html('LeagueofLegends-role-Analysis-Prediction/assets/normalKDA_pe
```



We reject the null hypothesis

Step 5: Framing a Prediction Problem

Prediction question:

- Predict the player's role given their post-game statistics
- multiclass classification

Predicting `position`

Metric:

- Will be using ***Accuracy***, as every game consits of the 5 positions we are predicting

Step 6: Baseline Model

Creating new features

In [37]:
```
df['kda_15'] = (df['killsat15'] + df['assistsat15']) / (df['deathsat15'] + 1
df['kd_15'] = df['killsat15']/ (df['deathsat15'] + 1)

df['kda_10'] = (df['killsat10'] + df['assistsat10']) / (df['deathsat10'] + 1
df['kd_10'] = df['killsat10']/ (df['deathsat10'] + 1)

df['monster_kpm'] = df['monsterkills'] / df['gamelength_minutes']
df['vision_pm'] = df['visionscore'] / df['gamelength_minutes']
```

Useful features to utilize:

- `kill_participation`
- `kda_normal`
- `kda_10`
- `kda_15`
- `goldat10`
- `xpat10`
- `csat10`
- `goldat15`
- `xpat15`
- `csat15`
- `monster_kpm`
- `vision_pm`
- `damagetakenperminute`
- `damageshare`
- `earnedgoldshare`
- `kills_per_minute`
- `assists_per_minute`
- `deaths_per_minute`
- `firstbloodkill`
- `firstbloodassist`
- `dpm`
- `earned gpm`
- `cspm`

Encoding the position column to the following:

Code	Position
0	Bot
1	Jungle
2	Mid
3	Support
4	Top

And keeping the feature we think are relevant

In [38]:
```
label_encoder = LabelEncoder()
df['position_encoded'] = label_encoder.fit_transform(df['position'])

model_df = df[['position_encoded', 'cspm', 'kda_normal'
    ,'xpat15', 'monster_kpm', 'damageshare','firstbloodkill', 'dpm'
              ,'goldat15', 'vision_pm']]

feature_columns = model_df.drop(columns=['position_encoded']).columns
```

Fitting the model

In [39]:
```
X = model_df
y = df['position_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra


preprocessor = ColumnTransformer(
    transformers=[
        ('imputer', SimpleImputer(strategy='mean'), feature_columns)
    ],
    remainder='passthrough'
)

pl = Pipeline([
    ('preproc', preprocessor),
    ('Random-Forest', RandomForestClassifier())
])

pl.fit(X_train.drop(columns = ['position_encoded']), y_train)

train_accuracy = pl.score(X_train.drop(columns = ['position_encoded']), y_tr
test_accuracy = pl.score(X_test.drop(columns = ['position_encoded']), y_test

print(f"Train accuracy: {train_accuracy}")
print(f"Test accuracy: {test_accuracy}")
```

```
Train accuracy: 1.0
Test accuracy: 0.8450517378615017
```

Step 7: Final Model

Using the same `model_df` and train & test data from the baseline model

Max_depth using:

- 'classifier__max_depth': [2, 3, 4, 5, 7, 10, 13, 15, 18, 20, 25, 30, None]
- Reason is to control the complexity, which ultimately helps control overfitting the data

Minimum samples split using:

- 'classifier__min_samples_split': [2, 5, 10, 20, 50, 100, 200]
- Also control overfitting of the data and improves generalization of the model

Standardized columns that are **per minute**

In [40]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('imputer', SimpleImputer(strategy='mean'), X.columns),
        ('scaler', StandardScaler(), ['cspm', 'dpm', 'monster_kpm'])
    ],
    remainder='passthrough'
)

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

param_grid = {
    'classifier__max_depth':[10, 20, 30, None],
    'classifier__min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1, scoring='a

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print(f'Best parameters found: {best_params}')
print(f'Best cross-validation accuracy: {best_score:.4f}')

final_model = pipeline.set_params(**best_params)
final_model.fit(X_train, y_train)

y_pred = final_model.predict(X_test)
test_accuracy = final_model.score(X_test, y_test)

print(f'Test set accuracy: {test_accuracy:.4f}')
```

```
Best parameters found: {'classifier__max_depth': 30, 'classifier__min_sample
s_split': 10}
Best cross-validation accuracy: 0.9998
Test set accuracy: 0.9997
```

Step 8: Fairness Analysis

Permutation test based on `damageshare`

- Median `damageshare` around 0.20, which makes sense as there are 5 players: 1/5

    - Classifying *low damage* as below 0.20 `damageshare`
    - *high damage* as above 0.20 `damageshare`
- Null Hypothesis: Our model is fair. Its accuracy for low damage share players and high damage share players are roughly the same, and any differences are due to random chance.

- Alternative Hypothesis: Our model is unfair. Its accuracy for high damage share players are higher then the accuracy of the model for low damage share players

X : Low damage Y : High damage

In [41]:
```
median_damageshare
```

Out[41]: 0.198369

In [42]:
```
median_damageshare = df.damageshare.median()
group_X_indices = X_test['damageshare'] <= median_damageshare
group_Y_indices = X_test['damageshare'] > median_damageshare

y_true_X = y_test[group_X_indices]
y_true_Y = y_test[group_Y_indices]
y_pred_X = y_pred[group_X_indices]
y_pred_Y = y_pred[group_Y_indices]

accuracy_X = accuracy_score(y_true_X, y_pred_X)
accuracy_Y = accuracy_score(y_true_Y, y_pred_Y)
observed_difference = accuracy_X - accuracy_Y
```

In [43]:
```
num_permutations = 10000
permutation_differences = []

combined_labels = np.concatenate((y_test, y_pred))
n = len(y_test)

for i in range(num_permutations):
    np.random.shuffle(combined_labels)

    y_test_shuffled = combined_labels[:n]
    y_pred_shuffled = combined_labels[n:]

    group_X_indices = X_test['damageshare'] <= median_damageshare
    group_Y_indices = X_test['damageshare'] > median_damageshare

    y_true_X_shuffled = y_test_shuffled[group_X_indices]
    y_true_Y_shuffled = y_test_shuffled[group_Y_indices]
    y_pred_X_shuffled = y_pred_shuffled[group_X_indices]
    y_pred_Y_shuffled = y_pred_shuffled[group_Y_indices]

    accuracy_X_shuffled = accuracy_score(y_true_X_shuffled, y_pred_X_shuffle
    accuracy_Y_shuffled = accuracy_score(y_true_Y_shuffled, y_pred_Y_shuffle

    permutation_differences.append(accuracy_X_shuffled - accuracy_Y_shufflec
```

```python
p_value = np.mean(permutation_differences >= observed_difference)
```

In [44]:
```python
print(f"P-value is: {p_value}")
```

```
P-value is: 0.5355
```

We fail to reject the null

Showing a strong evidence that our model is fair across the two groups