full_project

December 8, 2024

0.0.1 Z-LPER - Developing a Performance Based Ranking for League of Legends Esports

ICCS261 Term Project, Artem Kiselev, 6580846

[135]:
```
import os

import pandas as pd
```

Combine all of Oracle's Elixir CSV files from 2014-2024 into one

[136]:
```
folder_path = 'OraclesElixir'

csv_files = [f for f in os.listdir(folder_path) if f.endswith('.csv')]

combined_df = pd.DataFrame()

for file in csv_files:
    for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
        combined_df = pd.concat([combined_df, chunk], ignore_index=True)
```

```
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/283514767.py:8:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/283514767.py:8:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/283514767.py:8:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/283514767.py:8:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/283514767.py:8:
DtypeWarning: Columns (2) have mixed types. Specify dtype option on import or
```

```
set low_memory=False.
  for chunk in pd.read_csv(os.path.join(folder_path, file), chunksize=100000):
```

Remove incomplete data and sort by date

[137]:
```python
combined_df = combined_df[combined_df['datacompleteness'] == 'complete']
combined_df = combined_df.sort_values(by='date').reset_index()
```

Remove any columns where at least 30% of them null

[138]:
```python
threshold = len(combined_df) * 0.7
columns_to_drop = [col for col in combined_df.columns if combined_df[col].
 ↪isnull().sum() > threshold and col not in ['playername', 'playerid']]
df_c = combined_df.drop(columns=columns_to_drop)
df_c = df_c.dropna(thresh=threshold, axis=1)
```

Remove any minute-based metrics such as goldat15 goldat20 etc. since they are very similar to the full game metric and will introduce unnecessary complexity into the model

[139]:
```python
columns_with_at = [col for col in df_c.columns if 'at' in col]
columns_with_at.remove('date')
columns_with_at.remove('patch')
columns_with_at.remove('deaths')
columns_with_at.remove('teamdeaths')
columns_with_at.remove('damagemitigatedperminute')

df_cd = df_c.drop(columns=columns_with_at)
```

participantid 100 or 200 are team rows. Remove all team rows, leave only individual player rows

[140]:
```python
df_cdg = df_cd[~df_cd['participantid'].isin([100, 200])]
```

Remove unnecessary descriptive columns as well as multikill and firstblood statistics, due to the incredibly low significane these statistics have for a game

[141]:
```python
df_cdg = df_cdg.
 ↪drop(columns=['year','date','patch','teamname','teamid','champion','ban1','ban2','ban3','ba
 ↪'playerid'])
```

Select numeric (statistical) columns

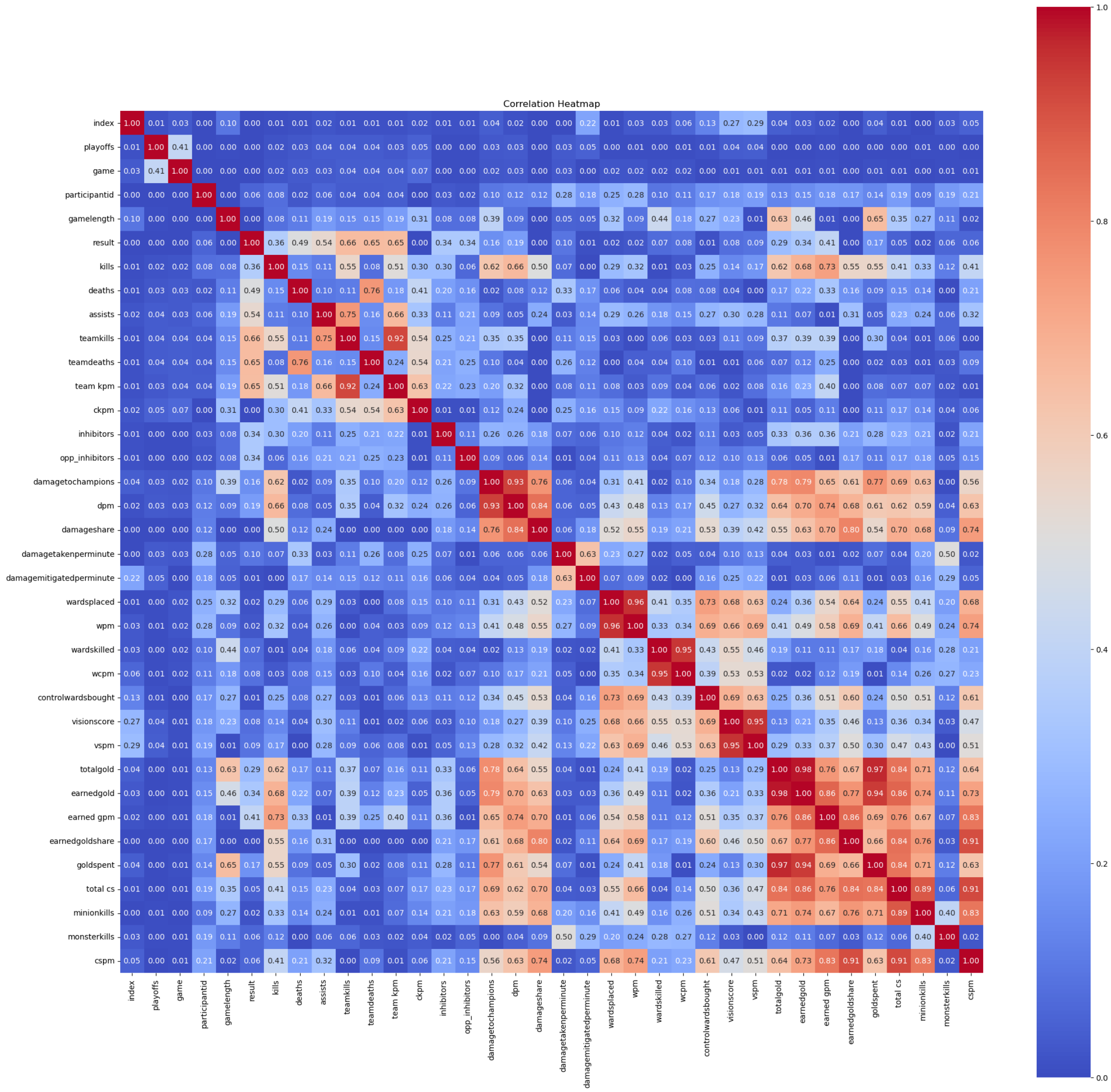
[142]:
```python
numeric = df_cdg.select_dtypes(include=['float64', 'int64'])
```

Create correlation matrix of all numeric columns

[143]:
```python
corr_matrix = numeric.corr()
corr_matrix = corr_matrix.abs()
```

Heatmap of correlation

[144]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(25, 25))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True,␣
 ↪square=True)
plt.title('Correlation Heatmap')
plt.show()
```



Find highly correlated columns

[145]:
```
threshold = 0.8
```

```
high_corr_pairs = [
    (col1, col2)
    for col1 in corr_matrix.columns
    for col2 in corr_matrix.columns
    if col1 != col2 and corr_matrix.loc[col1, col2] > threshold
]
```

Drop one of each pair of highly correlated columns

[146]:
```
to_drop = set()

for col1, col2 in high_corr_pairs:
    if col1 not in to_drop and col2 not in to_drop:
        to_drop.add(col2)

print("Features to drop:")
print(to_drop)

df_reduced = df_cdg.drop(columns=to_drop)
```

```
Features to drop:
{'earnedgoldshare', 'cspm', 'dpm', 'wpm', 'earnedgold', 'goldspent', 'team kpm',
'total cs', 'wcpm', 'vspm'}
```

Examine data

[147]:
```
df_reduced.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 726690 entries, 2 to 872027
Data columns (total 32 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   index                      726690 non-null  int64
 1   gameid                     726690 non-null  object
 2   league                     726690 non-null  object
 3   split                      556630 non-null  object
 4   playoffs                   726690 non-null  int64
 5   game                       726130 non-null  float64
 6   participantid              726690 non-null  int64
 7   side                       726690 non-null  object
 8   position                   726690 non-null  object
 9   playername                 726642 non-null  object
 10  gamelength                 726690 non-null  int64
 11  result                     726690 non-null  int64
 12  kills                      726690 non-null  int64
 13  deaths                     726690 non-null  int64
 14  assists                    726690 non-null  int64
 15  teamkills                  726690 non-null  int64
```

```
 16  teamdeaths                726690 non-null  int64
 17  ckpm                      726690 non-null  float64
 18  inhibitors                672450 non-null  float64
 19  opp_inhibitors            672450 non-null  float64
 20  damagetochampions         726490 non-null  float64
 21  damageshare               726490 non-null  float64
 22  damagetakenperminute      726490 non-null  float64
 23  damagemitigatedperminute  723550 non-null  float64
 24  wardsplaced               726490 non-null  float64
 25  wardskilled               726490 non-null  float64
 26  controlwardsbought        726490 non-null  float64
 27  visionscore               697820 non-null  float64
 28  totalgold                 725870 non-null  float64
 29  earned gpm                725870 non-null  float64
 30  minionkills               725170 non-null  float64
 31  monsterkills              726490 non-null  float64
dtypes: float64(16), int64(10), object(6)
memory usage: 183.0+ MB
```

Still some missing data, clean

[148]:
```
df_notna = df_reduced.dropna()
```

Dataset is now fully cleaned and is ready for the hypothesis test.

Finding out whether individual statisitcs of a professional League of Legends player contribute to the outcome of a game.

H0: There are no individual statisitcs for professional League of Legends players that significantly contribute towards the outcome of a game.

H1: There exist individual statistics for professional League of Legends players that significantly contribute towards the outcome of a game.

Filter by tier 1 leagues only for higher quality data

[149]:
```
tier1 = ['LCS', 'LEC', 'LPL', 'LCK']
tier1_df = df_notna[df_notna['league'].isin(tier1)]
```

Remove any players with less than 50 tier 1 games played to further increase quality

[150]:
```
player_counts = tier1_df['playername'].value_counts()
players_to_keep = player_counts[player_counts >= 50].index

tier1_df = tier1_df[tier1_df['playername'].isin(players_to_keep)]
df = tier1_df.reset_index(drop=True)
```

Separate the data into a features dataframe and the target variable

[151]:
```
X = df.
 ↪drop(columns=['gameid','split','league','playoffs','game','participantid','position','playe
 ↪'side', 'teamkills','teamdeaths','inhibitors','opp_inhibitors'])
```

```
y = df['result']
```

Inspect X

[152]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 68952 entries, 0 to 68951
Data columns (total 17 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   index                     68952 non-null  int64
 1   kills                     68952 non-null  int64
 2   deaths                    68952 non-null  int64
 3   assists                   68952 non-null  int64
 4   ckpm                      68952 non-null  float64
 5   damagetochampions         68952 non-null  float64
 6   damageshare               68952 non-null  float64
 7   damagetakenperminute      68952 non-null  float64
 8   damagemitigatedperminute  68952 non-null  float64
 9   wardsplaced               68952 non-null  float64
 10  wardskilled               68952 non-null  float64
 11  controlwardsbought        68952 non-null  float64
 12  visionscore               68952 non-null  float64
 13  totalgold                 68952 non-null  float64
 14  earned gpm                68952 non-null  float64
 15  minionkills               68952 non-null  float64
 16  monsterkills              68952 non-null  float64
dtypes: float64(13), int64(4)
memory usage: 8.9 MB
```

Random index column we dont need

[153]:
```
X = X.drop(columns=['index'])
```

Train Logistic Regression model on the features and the result of a game as the target

[154]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import statsmodels.api as sm

# train Logistic Regression model for p-values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
 ↪random_state=42)

X_train = sm.add_constant(X_train)
```

```
logit_model = sm.Logit(y_train, X_train)
result = logit_model.fit()

print("\nP-values from Logistic Regression:")
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.203071
         Iterations 8

P-values from Logistic Regression:
                           Logit Regression Results
==============================================================================
Dep. Variable:                 result   No. Observations:                48266
Model:                          Logit   Df Residuals:                    48249
Method:                           MLE   Df Model:                           16
Date:                Sun, 08 Dec 2024   Pseudo R-squ.:                  0.7070
Time:                        13:41:57   Log-Likelihood:                -9801.4
converged:                       True   LL-Null:                       -33447.
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
============
                              coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
------------
const                      -1.2752      0.169     -7.558      0.000      -1.606
-0.945
kills                       0.1811      0.015     11.865      0.000       0.151
0.211
deaths                     -0.4379      0.016    -27.676      0.000      -0.469
-0.407
assists                     0.6368      0.009     70.504      0.000       0.619
0.654
ckpm                       -7.3277      0.147    -49.946      0.000      -7.615
-7.040
damagetochampions           0.0001   5.16e-06     20.864      0.000    9.75e-05
0.000
damageshare               -20.7511      0.511    -40.632      0.000     -21.752
-19.750
damagetakenperminute       -0.0004      0.000     -2.958      0.003      -0.001
-0.000
damagemitigatedperminute 8.148e-05   7.29e-05      1.118      0.263   -6.13e-05
0.000
wardsplaced                -0.0198      0.002     -9.773      0.000      -0.024
-0.016
wardskilled                -0.0320      0.003     -9.594      0.000      -0.039
-0.025
```

```
controlwardsbought            0.0098      0.006      1.772      0.076      -0.001
0.021
visionscore                   0.0070      0.001      7.654      0.000       0.005
0.009
totalgold                    -0.0001   2.02e-05     -5.248      0.000      -0.000
-6.64e-05
earned gpm                    0.0570      0.001     65.329      0.000       0.055
0.059
minionkills                  -0.0206      0.001    -33.979      0.000      -0.022
-0.019
monsterkills                 -0.0228      0.001    -29.962      0.000      -0.024
-0.021
================================================================================
============
```

Analyze the accuracy of the model

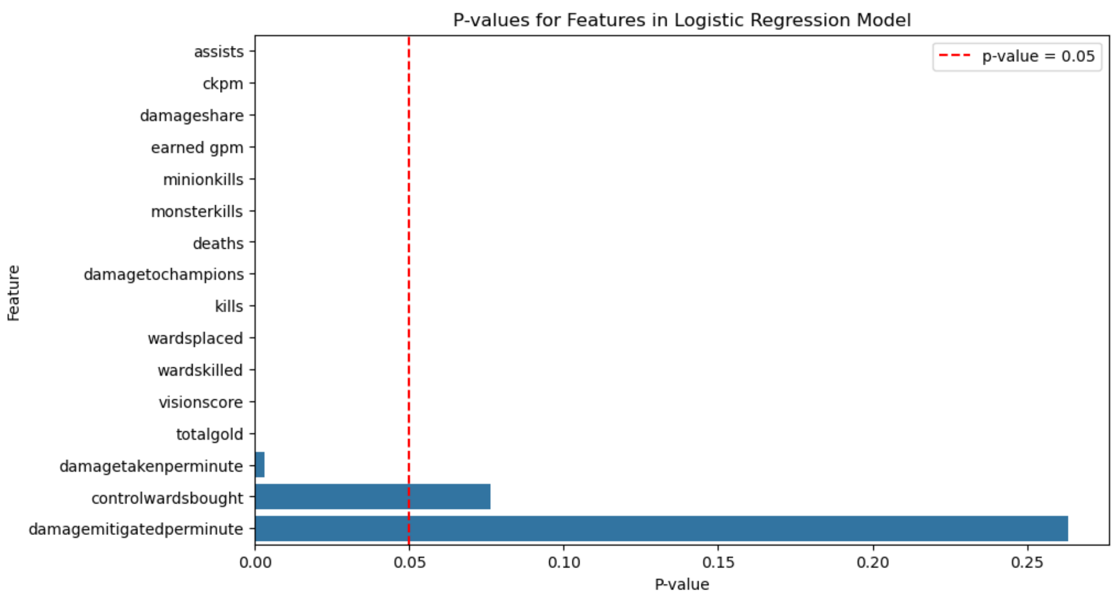
[155]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix

# add constant to the test set (for intercept)
X_test = sm.add_constant(X_test)
y_pred = result.predict(X_test)

# convert probabilities to binary predictions
y_pred_binary = (y_pred >= 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_binary)
error_rate = 1 - accuracy

print("\nAccuracy of the Logistic Regression model:", accuracy)

conf_matrix = confusion_matrix(y_test, y_pred_binary)
print("\nConfusion Matrix:")
print(conf_matrix)
```

```
Accuracy of the Logistic Regression model: 0.9181088658996422

Confusion Matrix:
[[9269  834]
 [ 860 9723]]
```

The accuracy of the model is high, therefore it is appropriate to utilize it for evaluating the hypothesis

[156]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

p_values = result.pvalues[1:]  # exclude constant
```

```
features = X_train.columns[1:]  # exclude constant

p_values_df = pd.DataFrame({
    'Feature': features,
    'P-value': p_values
}).sort_values(by='P-value', ascending=True)

plt.figure(figsize=(10, 6))
sns.barplot(x='P-value', y='Feature', data=p_values_df)

plt.axvline(x=0.05, color='r', linestyle='--', label='p-value = 0.05')
plt.title('P-values for Features in Logistic Regression Model')
plt.xlabel('P-value')
plt.ylabel('Feature')
plt.legend()
plt.show()
```



Conclusion:
Since there exist features with p-values less than 0.05, we reject the null hypothesis. This indicates that individual statistics of professional League of Legends players significantly contribute to the outcome of a game.

Extract all of the features that have a p-value less than 0.05 to use for the composite metric

[157]:
```
important_features = p_values_df[p_values_df['P-value'] < 0.05]

X_adj = df[important_features['Feature'].values]
```

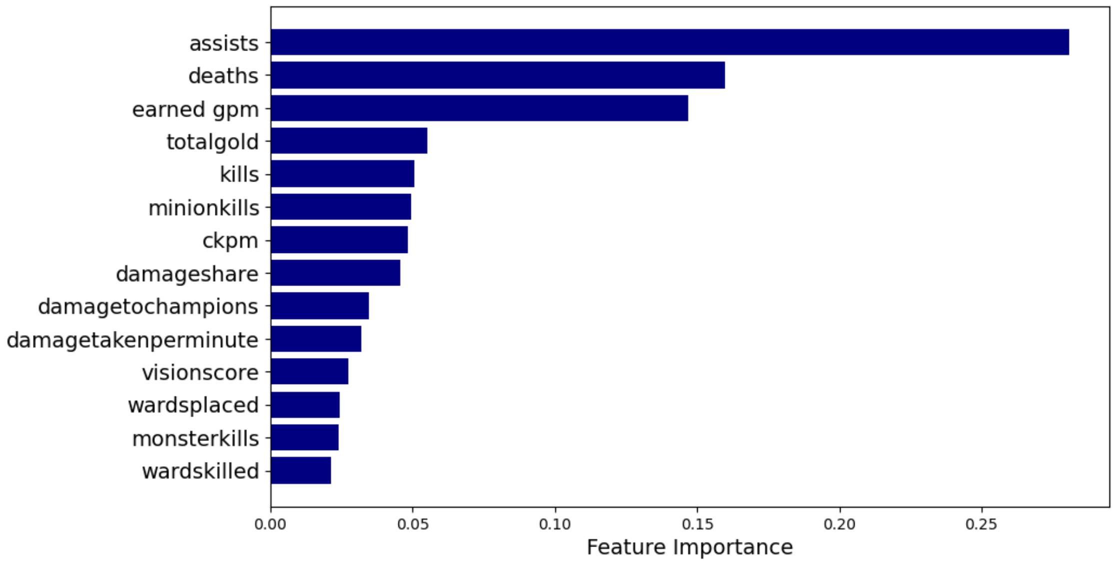
Train a Random Forest model on the important features, again using the result as a target to find out the feature weights

[158]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

rf_model = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)

# perform cross-validation
cv_scores = cross_val_score(rf_model, X_adj, y, cv=5, scoring='accuracy')
print(f'Cross-validation scores: {cv_scores}')
print(f'Average accuracy: {cv_scores.mean()}')

# train the Random Forest model on the entire dataset to get feature importances
rf_model.fit(X_adj, y)
```

```
Cross-validation scores: [0.90602567 0.90718585 0.9106599  0.9142132
0.90717912]
Average accuracy: 0.9090527453007287
```

[158]:
```
RandomForestClassifier(n_jobs=-1, random_state=42)
```

[159]:
```
feature_importances = rf_model.feature_importances_

importance_df = pd.DataFrame({
    'Feature': X_adj.columns,
    'Importance': feature_importances
})

# sort by importance
importance_df = importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'], color='#000080')
plt.xlabel('Feature Importance', fontsize=14)
plt.gca().invert_yaxis()

plt.yticks(fontsize=14)
plt.show()
```



Find the correlation between each feature and the result

[160]:
```
for feature in importance_df['Feature']:
    correlation = df[[feature, 'result']].corr(method='pearson')
    print(f"Correlation of {feature} with outcome: {correlation.iloc[0, 1]}")
```

```
Correlation of assists with outcome: 0.5571930198737738
Correlation of deaths with outcome: -0.5015048133368046
Correlation of earned gpm with outcome: 0.37962043649608035
Correlation of totalgold with outcome: 0.269633206331376
Correlation of kills with outcome: 0.3684680184870583
Correlation of minionkills with outcome: 0.010954710488547998
Correlation of ckpm with outcome: 0.004370908829536831
Correlation of damageshare with outcome: 8.93964728018843e-05
Correlation of damagetochampions with outcome: 0.15140217419638427
Correlation of damagetakenperminute with outcome: -0.0873248644946249
Correlation of visionscore with outcome: 0.07735297390952742
Correlation of wardsplaced with outcome: 0.009095645140868145
Correlation of monsterkills with outcome: 0.06020532536807729
Correlation of wardskilled with outcome: 0.07067377748574687
```

Looking at the correlation results, only deaths and damagetakenperminute negatively affect the result. It is too difficult to determine whether damageshare is helpful or harmful therefore I will not be utilizing it.

Examine the important features along with their weights

[161]:
```
print(importance_df)
```

```
                Feature  Importance
0               assists    0.280808
```

```
6                 deaths    0.159860
3             earned gpm    0.146779
12             totalgold    0.055072
8                  kills    0.050700
4            minionkills    0.049312
1                   ckpm    0.048318
2            damageshare    0.045502
7      damagetochampions    0.034693
13  damagetakenperminute    0.032021
11           visionscore    0.027501
9            wardsplaced    0.024304
5           monsterkills    0.023850
10           wardskilled    0.021279
```

Re-calculate the weights to account for the relationships between the values (reverse normalize) to be able to use them for raw data entries

[162]:
```python
import pandas as pd

features = importance_df['Feature'].values
importances = importance_df['Importance'].values

columns_in_df = [feature for feature in features if feature in df.columns]

std_vals = df[columns_in_df].std()

adjusted_weights = {feature: importance_df.loc[importance_df['Feature'] ==
 ↪feature, 'Importance'].values[0] / std_vals[feature]
                    for feature in columns_in_df}
```

[163]:
```python
adjusted_weights
```

[163]:
```
{'assists': 0.06930291698775495,
 'deaths': 0.09147490853291804,
 'earned gpm': 0.0017483451018731155,
 'totalgold': 1.4994026866621013e-05,
 'kills': 0.02116208548962984,
 'minionkills': 0.00037834917933246033,
 'ckpm': 0.19532001895058795,
 'damageshare': 0.45358071056950583,
 'damagetochampions': 3.8991705104160675e-06,
 'damagetakenperminute': 0.00012888790677417677,
 'visionscore': 0.0008162120088924586,
 'wardsplaced': 0.0012322949371729322,
 'monsterkills': 0.00041552051372437826,
 'wardskilled': 0.0029094160761692503}
```

Account for the negative weight of deaths and damagetakenperminute

[164]:
```
adjusted_weights['deaths'] = -abs(adjusted_weights['deaths'])
adjusted_weights['damagetakenperminute'] =␣
↪-abs(adjusted_weights['damagetakenperminute'])
```

Calculate the sum of these weights for each row in the dataset, name it LPER

[165]:
```
df['LPER'] = 0

for index, row in df.iterrows():
    LPER = 0
    for feature in columns_in_df:
        LPER += adjusted_weights[feature] * row[feature]
    df.at[index, 'LPER'] = LPER
```

```
/var/folders/0h/ch124vqs7dj_96gjw7qq3ksh0000gn/T/ipykernel_18676/2210354387.py:7
: FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise an error in a future version of pandas. Value '0.5562696567570568' has
dtype incompatible with int64, please explicitly cast to a compatible dtype
first.
  df.at[index, 'LPER'] = LPER
```

Find the average LPER per unique player

[166]:
```
average_lper_df = df.groupby('playername')['LPER'].mean().reset_index()

average_lper_df.columns = ['playername', 'LPER']
```

Sort players by LPER

[167]:
```
average_lper_df_sorted = average_lper_df.sort_values(by='LPER',␣
↪ascending=False).reset_index(drop=True)

# reset indexes starting from 1
average_lper_df_sorted.index = average_lper_df_sorted.index + 1

df_lper = average_lper_df_sorted
df_lper
```

[167]:
```
    playername      LPER
1         Peyz  1.683909
2      Rekkles  1.638864
3    Berserker  1.597246
4        Upset  1.593840
5       Aiming  1.588818
..         …         …
342       Zzus  0.765402
343      Pollu  0.753166
344     Secret  0.728566
345       Nova  0.714336
```

```
346   Jactroll  0.680108

[346 rows x 2 columns]
```

Clearly ADC's are too highly valued. Split the df into roles. First find the main position for each player.

[168]:
```python
player_roles = (
    df.groupby(['playername', 'position'])
    .size()
    .unstack(fill_value=0)  # Make positions into columns
    .reset_index()          # Ensure playername is a regular column
)

player_roles.columns.name = None
player_roles['position'] = player_roles.iloc[:, 1:].idxmax(axis=1)

player_roles
```

[168]:
```
      playername  bot  jng  mid  sup  top position
0            ADD    0    0    0    0  234      top
1            APA    0    0   92    0    0      mid
2      Abbedagge    0    0  255    0    0      mid
3    Ablazeolive    0    0  101    0    0      mid
4           Adam    0    0    0    0  197      top
..           ...  ...  ...  ...  ...  ...      ...
341      kyeahoo    0    0   58    0    0      mid
342          nuc    0    0  199    0    0      mid
343      promisq    0    0    0   56    0      sup
344         ucal    0    0  232    0    0      mid
345        vital   56    0    0   11    0      bot

[346 rows x 7 columns]
```

Add the position into the lper dataframe

[169]:
```python
df_lper = df_lper.merge(player_roles[['playername', 'position']],␣
 ↪on='playername', how='left')

df_lper.head()
```

[169]:
```
  playername      LPER position
0       Peyz  1.683909      bot
1    Rekkles  1.638864      bot
2  Berserker  1.597246      bot
3      Upset  1.593840      bot
4     Aiming  1.588818      bot
```

Separate df's for each position for clarity

[170]:
```
def split_df_by_position(df_lper):
    df_top = df_lper[df_lper['position'] == 'top']
    df_jng = df_lper[df_lper['position'] == 'jng']
    df_mid = df_lper[df_lper['position'] == 'mid']
    df_bot = df_lper[df_lper['position'] == 'bot']
    df_sup = df_lper[df_lper['position'] == 'sup']

    df_top = df_top.sort_values(by='LPER', ascending=False).
 ↪reset_index(drop=True)
    df_top.index = df_top.index + 1

    df_jng = df_jng.sort_values(by='LPER', ascending=False).
 ↪reset_index(drop=True)
    df_jng.index = df_jng.index + 1

    df_mid = df_mid.sort_values(by='LPER', ascending=False).
 ↪reset_index(drop=True)
    df_mid.index = df_mid.index + 1

    df_bot = df_bot.sort_values(by='LPER', ascending=False).
 ↪reset_index(drop=True)
    df_bot.index = df_bot.index + 1

    df_sup = df_sup.sort_values(by='LPER', ascending=False).
 ↪reset_index(drop=True)
    df_sup.index = df_sup.index + 1

    return df_top, df_jng, df_mid, df_bot, df_sup

df_top, df_jng, df_mid, df_bot, df_sup = split_df_by_position(df_lper)
```

Analyze one of the dataframes for accuracy

[171]:
```
df_top.head(30)
```

[171]:
```
      playername      LPER position
1         Nuguri  1.331674      top
2           Khan  1.327417      top
3          Fudge  1.307384      top
4    BrokenBlade  1.292232      top
5           Smeb  1.290666      top
6           Duke  1.276775      top
7        Alphari  1.275963      top
8         Photon  1.274620      top
9        Ssumday  1.259186      top
10         Armut  1.248439      top
11         Doran  1.245400      top
```

```
12          Kiin  1.244618      top
13         Canna  1.237021      top
14          Zeus  1.236126      top
15         Bwipo  1.219015      top
16        Wunder  1.215140      top
17     Oscarinin  1.205861      top
18          Huni  1.204419      top
19         Chasy  1.203131      top
20    Vizicsacsi  1.198212      top
21       Odoamne  1.195713      top
22         MaRin  1.191405      top
23        Impact  1.191056      top
24         Myrwn  1.187142      top
25         Orome  1.181355      top
26         HiRit  1.179685      top
27         SwOrd  1.171861      top
28        Summit  1.169306      top
29      Licorice  1.161412      top
30          Thal  1.156600      top
```

Very easy to notice that European and American players are way overvalued compared to Korean players if you possess knowledge of the scene. Let's adjust this.

Use Riot Games' Official Region Power Rankings to adjust the LPER of each player based on their region's strength. This is not perfect, but should improve the ranking's usability significantly.

[172]:

```python
league_strengths = {
    'LEC': 1542,
    'LCS': 1486,
    'LCK': 1873
}
```

Recalculate the LPER for each player based on region

[173]:

```python
df['Adj_LPER'] = df.apply(lambda row: round(row['LPER'] *␣
 ↪league_strengths[row['league']]), axis=1)

df
```

[173]:

```
        index           gameid league   split  playoffs  game  participantid  \
0      860025      TRKR1/750688    LCK  Spring         0   1.0             10
1      860017      TRKR1/750688    LCK  Spring         0   1.0              2
2      860016      TRKR1/750688    LCK  Spring         0   1.0              1
3      860020      TRKR1/750688    LCK  Spring         0   1.0              5
4      860037      TRKR1/750692    LCK  Spring         0   2.0             10
…         …                  …      …       …      …     …
68947  280720  LOLTMNT03_147405    LCK  Summer         1   5.0              5
68948  280719  LOLTMNT03_147405    LCK  Summer         1   5.0              4
68949  280718  LOLTMNT03_147405    LCK  Summer         1   5.0              3
```

```
68950  280717  LOLTMNT03_147405    LCK  Summer         1   5.0             2
68951  280716  LOLTMNT03_147405    LCK  Summer         1   5.0             1

       side position playername  …  wardsplaced  wardskilled  \
0       Red      sup      Jelly  …         27.0          9.0
1      Blue      jng      TusiN  …         31.0         16.0
2      Blue      top  Expession  …         10.0          4.0
3      Blue      sup     IgNar  …         41.0         12.0
4       Red      sup     IgNar  …         44.0          6.0
…       …      …        …  …          …          …
68947  Blue      sup    Lehends  …         92.0         17.0
68948  Blue      bot       Peyz  …         13.0         10.0
68949  Blue      mid      Chovy  …         17.0         12.0
68950  Blue      jng     Canyon  …         17.0         17.0
68951  Blue      top       Kiin  …         17.0          9.0

       controlwardsbought  visionscore  totalgold  earned gpm  minionkills  \
0                     4.0          0.0     5574.0     67.2936         39.0
1                     5.0          0.0    10245.0    227.9931         23.0
2                     1.0          0.0    13567.0    342.2821        204.0
3                     6.0          0.0     7539.0    134.8968         30.0
4                     8.0          0.0     6014.0     54.7685         27.0
…                       …            …          …           …            …
68947                28.0        149.0     7826.0     75.3470         32.0
68948                 3.0         30.0    14412.0    247.8306        365.0
68949                 6.0         56.0    14791.0    257.7564        360.0
68950                16.0         61.0    10826.0    153.9153         26.0
68951                11.0         55.0    12706.0    203.1515        271.0

       monsterkills      LPER  Adj_LPER
0               0.0  0.556270      1042
1              68.0  1.542899      2890
2               6.0  1.536750      2878
3               1.0  1.398530      2619
4               0.0  0.293348       549
…               …         …         …
68947           0.0  0.365269       684
68948          20.0  0.908849      1702
68949          11.0  1.004362      1881
68950         192.0  0.806346      1510
68951           4.0  0.856248      1604

[68952 rows x 34 columns]
```

[174]:

```
df_avg_lper = df.groupby('playername')['Adj_LPER'].mean().reset_index()
df_avg_lper['Adj_LPER'] = df_avg_lper['Adj_LPER'].round().astype(int)
df_avg_lper.rename(columns={'Adj_LPER': 'LPER'}, inplace=True)
```

```
df_avg_lper
```

[174]:

```
      playername  LPER
0            ADD  2009
1            APA  2239
2      Abbedagge  2125
3    Ablazeolive  1691
4           Adam  1768
..           …     …
341      kyeahoo  2193
342          nuc  2038
343      promisq  1205
344         ucal  2359
345        vital  2208

[346 rows x 2 columns]
```

[175]:

```
df_lper_adjusted = df_avg_lper.merge(player_roles[['playername', 'position']],
 ↪on='playername', how='left')
df_lper_adjusted
```

[175]:

```
      playername  LPER position
0            ADD  2009      top
1            APA  2239      mid
2      Abbedagge  2125      mid
3    Ablazeolive  1691      mid
4           Adam  1768      top
..           …     …        …
341      kyeahoo  2193      mid
342          nuc  2038      mid
343      promisq  1205      sup
344         ucal  2359      mid
345        vital  2208      bot

[346 rows x 3 columns]
```

Separate into roles again

[176]:

```
df_top, df_jng, df_mid, df_bot, df_sup = split_df_by_position(df_lper_adjusted)
```

Reanalyze the top df

[177]:

```
df_top.head(30)
```

[177]:

```
     playername  LPER position
1        Nuguri  2494      top
2          Khan  2486      top
3          Smeb  2417      top
```

```
4          Duke  2391       top
5         Doran  2333       top
6          Kiin  2331       top
7          Zeus  2315       top
8         Canna  2294       top
9         MaRin  2232       top
10        SwOrd  2195       top
11         Thal  2166       top
12       Kingen  2159       top
13        TrAce  2156       top
14      PerfecT  2149       top
15    Expession  2138       top
16       Rascal  2136       top
17       SoHwan  2094       top
18        Crazy  2086       top
19       Summit  2086       top
20        Clear  2086       top
21          Shy  2072       top
22        CuVee  2071       top
23      Ssumday  2026       top
24       Untara  2020       top
25       Burdol  2014       top
26          ADD  2009       top
27         Rich  1974       top
28  BrokenBlade  1971       top
29         DuDu  1966       top
30       Photon  1965       top
```

Despite being slightly too biased towards LCK players, this seems to be quite accurate, so let's proceed with these results

Find the z-score for each role to see deviation from mean for each player (how much better or worse a player is at their role than the average)

[178]:
```
def calculate_z_scores(df):
    mean_lper = df['LPER'].mean()
    std_lper = df['LPER'].std()
    df['z_score'] = (df['LPER'] - mean_lper) / std_lper
    return df

df_top = calculate_z_scores(df_top)
df_jng = calculate_z_scores(df_jng)
df_mid = calculate_z_scores(df_mid)
df_bot = calculate_z_scores(df_bot)
df_sup = calculate_z_scores(df_sup)
```

[179]:
```
df_top.head(30)
```

[179]:

	playername	LPER	position	z_score
1	Nuguri	2494	top	2.298505
2	Khan	2486	top	2.265921
3	Smeb	2417	top	1.984890
4	Duke	2391	top	1.878994
5	Doran	2333	top	1.642764
6	Kiin	2331	top	1.634618
7	Zeus	2315	top	1.569452
8	Canna	2294	top	1.483920
9	MaRin	2232	top	1.231399
10	Sw0rd	2195	top	1.080701
11	Thal	2166	top	0.962586
12	Kingen	2159	top	0.934076
13	TrAce	2156	top	0.921857
14	PerfecT	2149	top	0.893347
15	Expession	2138	top	0.848545
16	Rascal	2136	top	0.840399
17	SoHwan	2094	top	0.669336
18	Crazy	2086	top	0.636753
19	Summit	2086	top	0.636753
20	Clear	2086	top	0.636753
21	Shy	2072	top	0.579732
22	CuVee	2071	top	0.575659
23	Ssumday	2026	top	0.392377
24	Untara	2020	top	0.367940
25	Burdol	2014	top	0.343502
26	ADD	2009	top	0.323138
27	Rich	1974	top	0.180585
28	BrokenBlade	1971	top	0.168367
29	DuDu	1966	top	0.148002
30	Photon	1965	top	0.143929

The rankings of these players are fairly consistent with the general concensus of the community, let's try to merge everyone into one ranking

[180]:

```python
df_z_score = pd.concat([df_top, df_jng, df_mid, df_bot, df_sup],␣
↪ignore_index=True)
df_z_score = df_z_score.sort_values(by='z_score', ascending=False).
↪reset_index(drop=True)
df_z_score.index = df_z_score.index + 1

df_z_score
```

[180]:

	playername	LPER	position	z_score
1	Chovy	2913	mid	2.785904
2	Peyz	3154	bot	2.767598
3	Keria	2065	sup	2.312110
4	Nuguri	2494	top	2.298505

```
5           Khan  2486      top  2.265921
..           …     …       …        …
342         LIDER  1730      mid -1.807140
343   Ablazeolive  1691      mid -1.958559
344        Sniper  1439      top -1.998428
345         Kenvi  1213      jng -2.261907
346      Jactroll  1049      sup -2.490801

[346 rows x 4 columns]
```

Chovy being by far the highest ranked player (regarded by most of the community to be the best), followed by other statistically exceptional players like Peyz Keria and Khan signals that the ranking is insightful and meaningful

For easier comprehension, let's adjust z-score by multiplying by making it non-negative, and increasing it by 1000 to be able to round it to an integer. Name this Z-LPER.

[181]:
```
df_z_lper = df_z_score[['playername', 'position', 'z_score']]
df_z_lper['Z-LPER'] = (df_z_lper['z_score'] * 1000).round().astype(int)
df_z_lper['Z-LPER'] += abs(df_z_lper['Z-LPER'].min())
df_z_lper = df_z_lper.drop(columns='z_score')

df_z_lper
```

[181]:
```
       playername position  Z-LPER
1           Chovy      mid    5277
2            Peyz      bot    5259
3           Keria      sup    4803
4          Nuguri      top    4790
5            Khan      top    4757
..            …        …       …
342         LIDER      mid     684
343   Ablazeolive      mid     532
344        Sniper      top     493
345         Kenvi      jng     229
346      Jactroll      sup       0

[346 rows x 3 columns]
```

Inspect the top 30 players

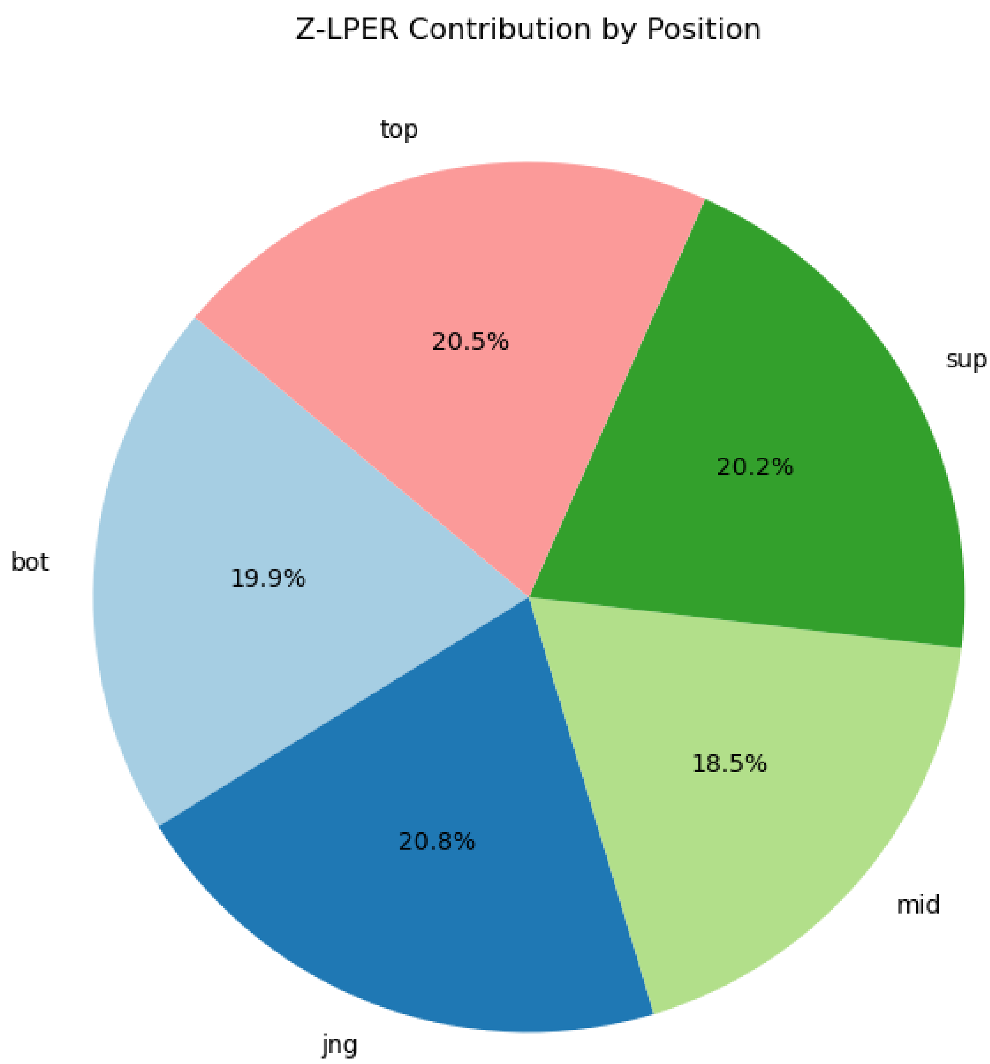
[182]:
```
df_z_lper.head(30)
```

[182]:
```
    playername position  Z-LPER
1        Chovy      mid    5277
2         Peyz      bot    5259
3        Keria      sup    4803
4       Nuguri      top    4790
5         Khan      top    4757
```

```
6       Aiming      bot    4616
7    ShowMaker      mid    4597
8       Peanut      jng    4514
9       Canyon      jng    4486
10        Smeb      top    4476
11     Delight      sup    4449
12    Gumayusi      bot    4377
13        Duke      top    4370
14        Zeka      mid    4349
15       Viper      bot    4349
16       Score      jng    4315
17        Yike      jng    4299
18     Lehends      sup    4288
19       Ruler      bot    4247
20        Wolf      sup    4245
21        Mata      sup    4231
22        PawN      mid    4209
23       Lucid      jng    4163
24         Bdd      mid    4163
25        Oner      jng    4155
26       Doran      top    4134
27        Kiin      top    4126
28       Faker      mid    4116
29       Teddy      bot    4063
30        Zeus      top    4060
```

Lets visualize and analyze the resulting ranking

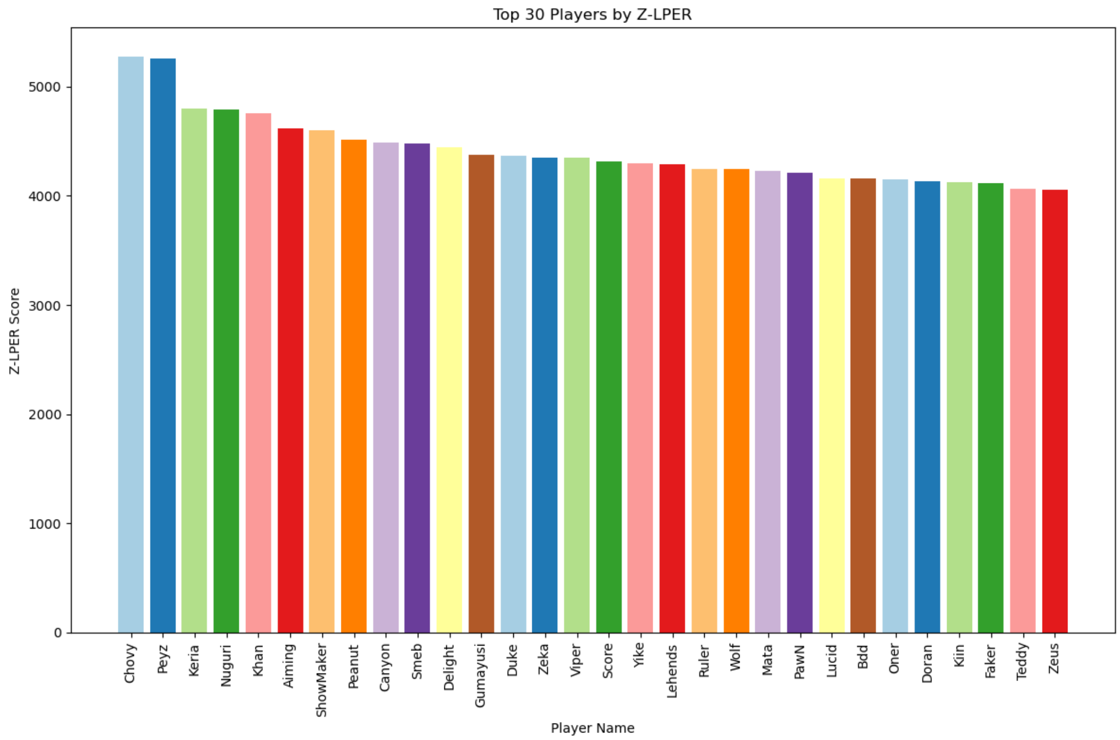
[183]:
```python
position_z_lper = df_z_lper.groupby("position")["Z-LPER"].sum()

# Plotting the pie chart
plt.figure(figsize=(8, 8))
plt.pie(
    position_z_lper,
    labels=position_z_lper.index,
    autopct='%1.1f%%',
    startangle=140,
    colors=plt.cm.Paired.colors
)
plt.title("Z-LPER Contribution by Position")
plt.show()
```



Each role is fairly even

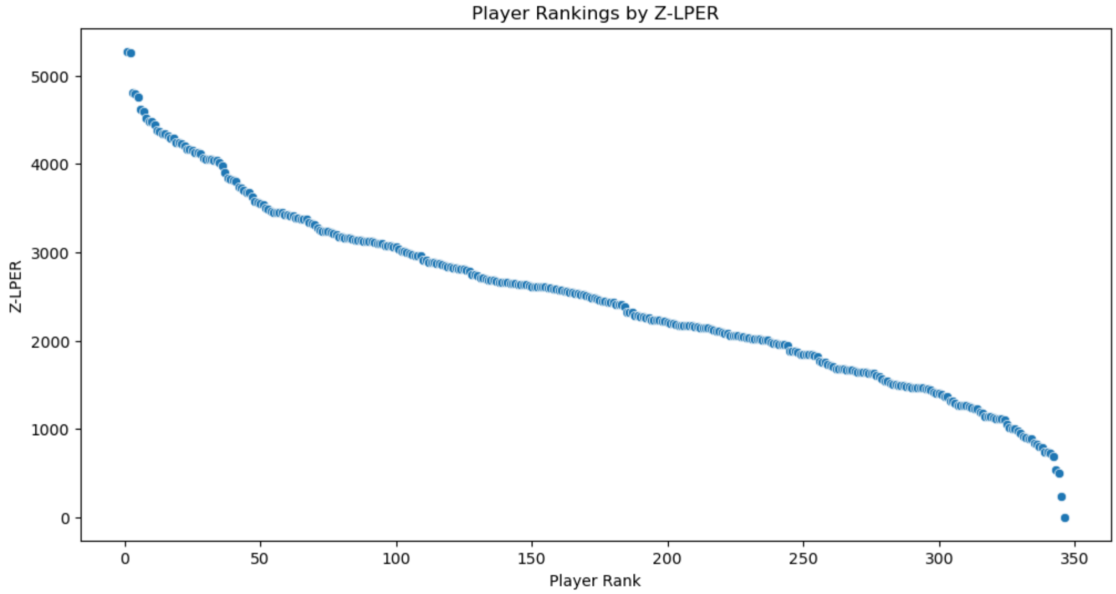
[184]:
```
df_top30 = df_z_lper.head(30)

# Plotting a bar chart
plt.figure(figsize=(12, 8))
plt.bar(df_top30["playername"], df_top30["Z-LPER"], color=plt.cm.Paired.colors)
plt.xticks(rotation=90)
plt.xlabel("Player Name")
plt.ylabel("Z-LPER Score")
plt.title("Top 30 Players by Z-LPER")
plt.tight_layout()
```

```
plt.show()
```



Chovy and Peyz seem to be very exceptional statistically

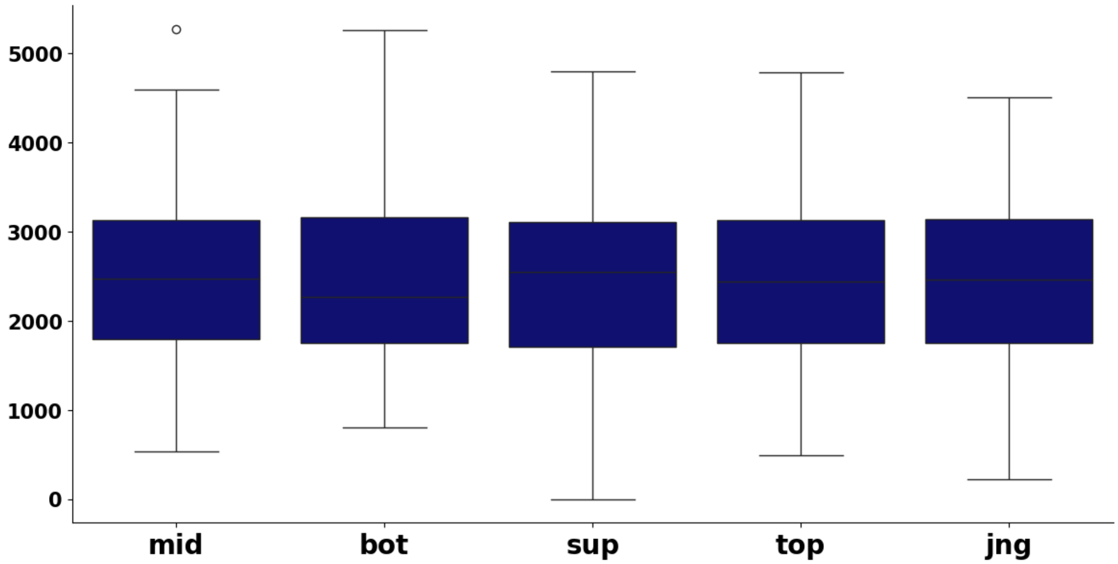
[185]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='index', y='Z-LPER', data=df_z_lper.reset_index(),␣
 ↪legend=None)
plt.xlabel('Player Rank')
plt.ylabel('Z-LPER')
plt.title('Player Rankings by Z-LPER')
plt.show()
```



Looks like the metric either starts to break down at the top and bottom or there are significant statistical outliers (Chovy and Peyz make sense to be so far ahead, the bottom is not as explainable)

[186]:
```
plt.figure(figsize=(14, 7))

# Draw the boxplot with navy blue bars and no lines
sns.boxplot(
    x='position',
    y='Z-LPER',
    data=df_z_lper,
    color='#000080',    # Navy blue color for the boxes
)

# Hide the axis labels (x and y)
plt.xlabel('')
plt.ylabel('')

# Adjust tick parameters for better readability
plt.xticks(fontsize=20, weight='bold')
plt.yticks(fontsize=15, weight='bold')

# Remove grid lines
plt.grid(False)

# Remove spines for a cleaner look
sns.despine()
```

```
# Show the plot
plt.show()
```



Seems that bot is still biased towards, likely due to the role usually having the best stats. Also we can see just how much of an outlier Chovy is statistically.

0.0.2 Conclusion:

Despite them not being completely perfect and having outliers, these results are still quite insightful and can initiate a lot of discussion towards improvement in the application of data science towards the LoL Esports ecosystem.