PREDICTING WNBA PLAYOFF TEAMS FOR EACH SEASON

In today's competitive sports landscape, gaining a competitive edge has become paramount for teams striving for success. The integration of data analytics and machine learning techniques has revolutionized the way sports organizations analyze performance, make strategic decisions, and identify opportunities for improvement. In this report, we delve into the application of machine learning in predicting team's playoff status in basketball, focusing on the Women's National Basketball Association (WNBA). The objective of this study is to develop a predictive model that will accurately forecast a team's playoff participation for WNBA based on a combination of players' and team's performance metrics.

WNBA PLAYOFFS TEAM PREDICTIONS AT THE FROM SEASON 1 TO 10

```
In [2]: import pandas as pd
In [3]: teams_df = pd.read_csv("teams.csv")
In [4]: players_teams_df = pd.read_csv("players_teams.csv")
In [5]: teams_df.head()
```

Out[5]:

	year	lgID	tmID	franchID	confID	divID	rank	playoff	seeded	firstRound	...	GP	homeW	homeL	awayW	awayL	confW	confL	min	attend	ar
0	9	WNBA	ATL	ATL	EA	NaN	7	N	0	NaN	...	34	1	16	3	14	2	18	6825	141379	Phi Ar
1	10	WNBA	ATL	ATL	EA	NaN	2	Y	0	L	...	34	12	5	6	11	10	12	6950	120737	Phi Ar
2	1	WNBA	CHA	CHA	EA	NaN	8	N	0	NaN	...	32	5	11	3	13	5	16	6475	90963	Charl Colise
3	2	WNBA	CHA	CHA	EA	NaN	4	Y	0	W	...	32	11	5	7	9	15	6	6500	105525	Charl Colise
4	3	WNBA	CHA	CHA	EA	NaN	2	Y	0	L	...	32	11	5	7	9	12	9	6450	106670	Charl Colise

5 rows × 61 columns

Data Summary

The dataset includes team specific information such as (Team ID, Franchise ID, Conference ID, the team's name and arena details), players statistics, game results and playoffs

involvement through season 1 to 10. Performance measures include field goals made/attempts, free throws made/attempts, three-pointers made/attempts, rebounds, assists, steals, blocks, turnovers, fouls, and points scored, among others offensive and defensive statistics. Furthermore, information on victories, losses, minutes played, and attendance records provides an overview of team chemistry and fan involvement. In-depth information about postseason games played, starts, minutes, points, rebounds, assists, steals, blocks, turnovers, and personal fouls were also included in the dataset.

```
In [8]: # Merge the dataframes on both 'year' and 'tmID'
        combined_df = pd.merge(teams_df, players_teams_df, on=['year', 'tmID'])
        combined_df.head()
```

Out[8]:

	year	tmID	franchID	confID	rank	playoff	name	o_fgm	o_fga	o_ftm	...	PostBlocks	PostTurnovers	PostPF	PostfgAttempted	PostfgMade	PostftAttem
0	9	ATL	ATL	EA	7	N	Atlanta Dream	895	2258	542	...	0	0	0	0	0	
1	9	ATL	ATL	EA	7	N	Atlanta Dream	895	2258	542	...	0	0	0	0	0	
2	9	ATL	ATL	EA	7	N	Atlanta Dream	895	2258	542	...	0	0	0	0	0	
3	9	ATL	ATL	EA	7	N	Atlanta Dream	895	2258	542	...	0	0	0	0	0	
4	9	ATL	ATL	EA	7	N	Atlanta Dream	895	2258	542	...	0	0	0	0	0	

5 rows × 95 columns

```
In [9]: combined_df.columns
```

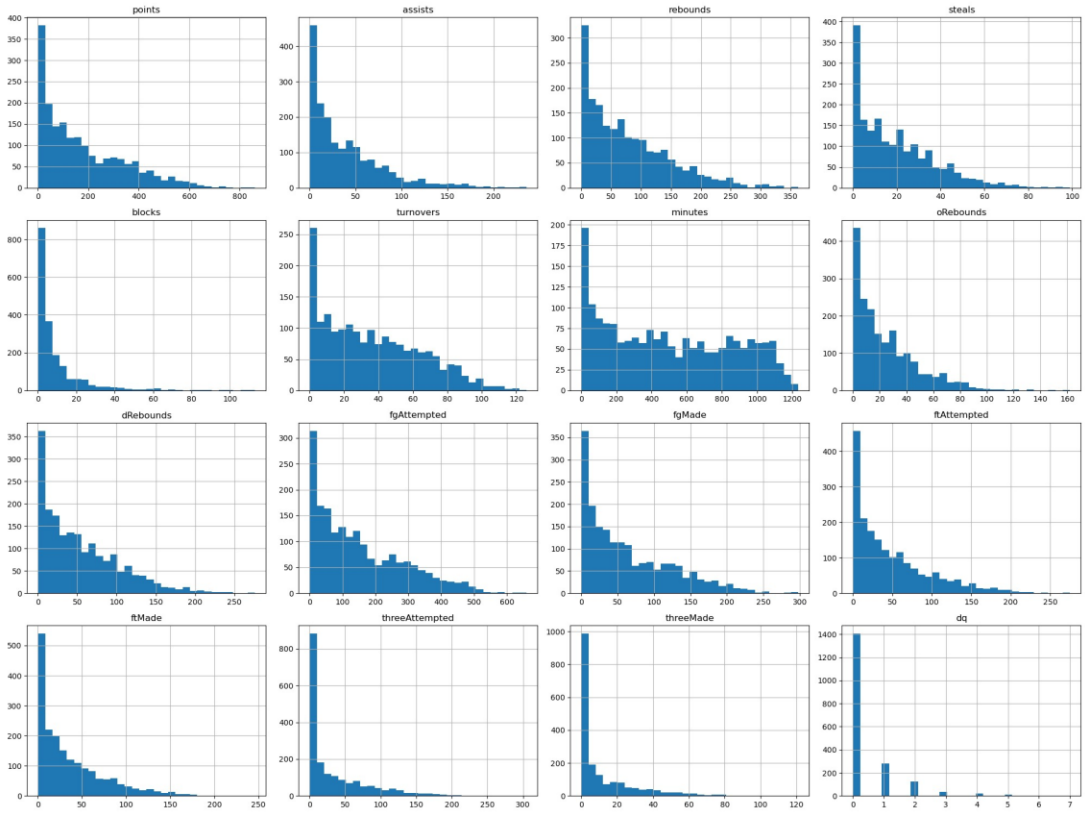
```
Out[9]: Index(['year', 'tmID', 'franchID', 'confID', 'rank', 'playoff', 'name',
               'o_fgm', 'o_fga', 'o_ftm', 'o_fta', 'o_3pm', 'o_3pa', 'o_oreb',
               'o_dreb', 'o_reb', 'o_asts', 'o_pf', 'o_stl', 'o_to', 'o_blk', 'o_pts',
               'd_fgm', 'd_fga', 'd_ftm', 'd_fta', 'd_3pm', 'd_3pa', 'd_oreb',
               'd_dreb', 'd_reb', 'd_asts', 'd_pf', 'd_stl', 'd_to', 'd_blk', 'd_pts',
               'tmORB', 'tmDRB', 'tmTRB', 'opptmORB', 'opptmDRB', 'opptmTRB', 'won',
               'lost', 'GP_x', 'homeW', 'homeL', 'awayW', 'awayL', 'confW', 'confL',
               'min', 'attend', 'arena', 'playerID', 'stint', 'GP_y', 'GS', 'minutes',
               'points', 'oRebounds', 'dRebounds', 'rebounds', 'assists', 'steals',
               'blocks', 'turnovers', 'PF', 'fgAttempted', 'fgMade', 'ftAttempted',
               'ftMade', 'threeAttempted', 'threeMade', 'dq', 'PostGP', 'PostGS',
               'PostMinutes', 'PostPoints', 'PostoRebounds', 'PostdRebounds',
               'PostRebounds', 'PostAssists', 'PostSteals', 'PostBlocks',
               'PostTurnovers', 'PostPF', 'PostfgAttempted', 'PostfgMade',
               'PostftAttempted', 'PostftMade', 'PostthreeAttempted', 'PostthreeMade',
               'PostDQ'],
              dtype='object')
```

Exploratory Data Analysis

We'll use EDA to perform tasks like data cleaning removing null entries and zero values from columns to guarantee the accuracy of our machine learning model. We'll also use histograms to analyze the distributions of metrics like turnovers, assists, and points; and make use of visualization tools like boxplots and heat maps to reveal patterns and

correlations that may lead to multicollinearity amongst different performance indicators. Additionally, this makes outlier discovery easier, which is important for identifying extraordinary performances in player's statistics.

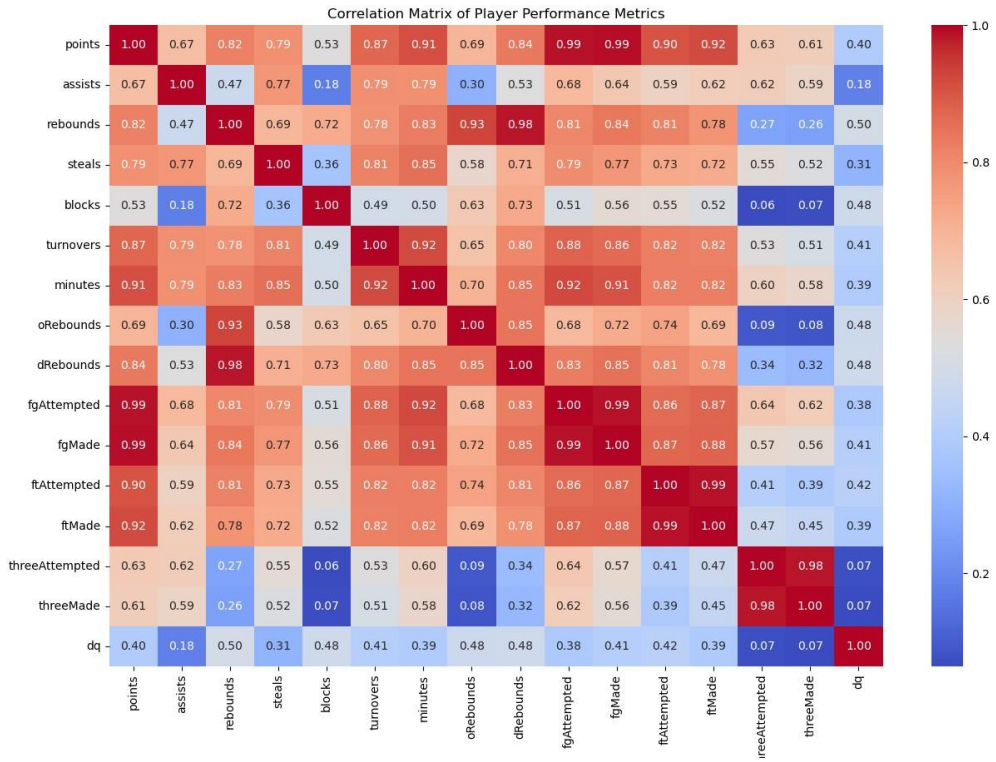
```
# Selecting the player performance metrics columns
player_performance_columns = [
    'points', 'assists', 'rebounds', 'steals', 'blocks', 'turnovers', 'minutes',
    'oRebounds', 'dRebounds', 'fgAttempted', 'fgMade', 'ftAttempted', 'ftMade',
    'threeAttempted', 'threeMade', 'dq'
]

# Filtering the dataframe to include only these columns
player_performance_df = combined_df_sorted[player_performance_columns]

# Creating histograms for the player performance metrics columns
player_performance_df.hist(bins=30, figsize=(20, 15))
plt.tight_layout()

# Save the plot to a file
plt.savefig('player_performance_histograms.png')  # Save as PNG format
plt.show()
```



Histogram of players performance metrics

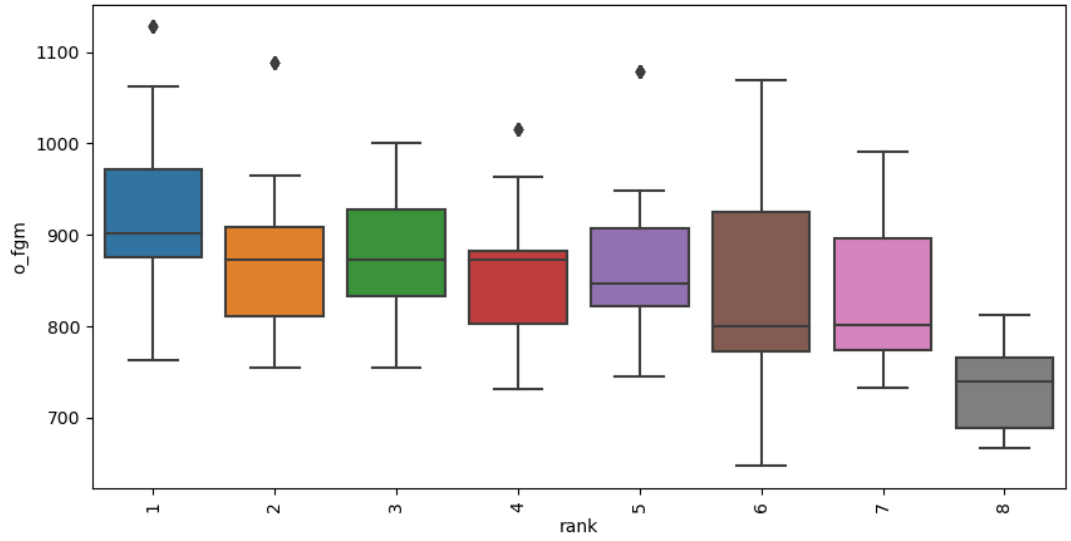
The boxplot indicates that most of these metrics, such as points scored, assists made, rebounds obtained, steals, blocks, and turnovers committed, are skewed to the left, which may mean that most teams tend to have lower values in these metrics. This may be a result of different player abilities, team plans, or league dynamics where a large percentage of teams may not excel in various areas.

```
#Correlation matrix
player_performance_df = combined_df_sorted[player_performance_columns]

# Calculating the correlation matrix
corr_matrix = player_performance_df.corr()

# Creating the heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix of Player Performance Metrics')


plt.savefig('Correlation Matrix of Player Performance Metrics')
plt.show()
```



The correlation matrix of player performance metrics reveals several interesting relationships. For instance, there is a strong positive correlation between field goals attempted and points scored (0.85), indicating that teams that take more shots tend to score more points. Similarly, field goals made also showed a high correlation with points (0.75), suggesting that successful shooting is a significant contributor to a team's scoring output. Additionally, minutes played have a substantial positive correlation with points (0.70) and rebounds (0.65), highlighting that players who spend more time on the court are more likely to contribute significantly to these areas. This makes sense, as more playing time generally offers more opportunities to score and collect rebounds.

```
In [14]: #Box Plot of field goals made and team rank
         plt.figure(figsize=(10, 5))
         sns.boxplot(x='rank', y='o_fgm',
                     data=combined_df_sorted)
         plt.xticks(rotation=90)
         plt.show()
```



Box plots of field goals and team rank

The box plot for rank 1 teams, although showing the highest median field goals made, includes one outlier. This suggests that while rank 1 teams generally make the most field goals, there is at least one team whose performance in this metric significantly higher than the rest. Also looking at the box plot for rank 6 teams, despite lacking outliers, has the longest whiskers extending both upwards and downwards. This implies a wide range of field goals made within this rank, indicating high variability and suggesting that rank 6 teams have a

more diverse performance level in terms of field goals made, with some performing quite well and others much less so.

Data Preprocessing

In this section, we preprocess the dataset to prepare it for analysis. The preprocessing steps begin by creating a copy of the original sorted data frame to preserve the initial data. We then filter the sorted data for each season (from 1 to 10) and extract the unique team IDs (this is to confirm that teams that participated in previous seasons are also present in the next season since we will be updating the previous season's statistic into the next season). We then define a function called “parse statistics” and generate new data frames for each season based on the previous season's statistics. These new data frames are then concatenated into a single data frame, and then we drop unnecessary columns. To make our target variable (‘playoff’) easier to work with we convert the column to binary values (1 for 'Y' and 0 for 'N') and apply normalization to the numerical columns using the MinMaxScaler from the scikit-learn library. This scaling method ensures that all performance metrics are on a similar scale of 0 to 1, which is essential for machine learning model development.

```
In [15]: wnba_data = combined_df_sorted.copy()
```

```
In [16]: #select season 1 statistic
         season_2_data = wnba_data[wnba_data['year'] == 1]
         #Get the unique teams in season 1 data
         unique_teams_season_2 = season_2_data['tmID'].unique()

         # Display the unique teams in season 1 data
         print("Unique teams in season 2:")
         print(unique_teams_season_2)
```

```
Unique teams in season 2:
['MIA' 'WAS' 'IND' 'POR' 'CLE' 'PHO' 'MIN' 'UTA' 'DET' 'SAC' 'LAS' 'NYL'
 'ORL' 'HOU' 'CHA' 'SEA']
```

```
In [17]: #select season 2 statistic
         season_3_data = wnba_data[wnba_data['year'] == 2]

         #Get the unique teams in season 2 data
         unique_teams_season_3 = season_3_data['tmID'].unique()

         # Display the unique teams in season 1 data
         print("Unique teams in season 3:")
         print(unique_teams_season_3)
```

```
Unique teams in season 3:
['MIN' 'IND' 'MIA' 'DET' 'WAS' 'POR' 'HOU' 'SEA' 'PHO' 'SAC' 'CLE' 'ORL'
 'LAS' 'UTA' 'NYL' 'CHA']
```

In [18]:
```
#select season 3 statistic
season_4_data = wnba_data[wnba_data['year'] == 3]
#Get the unique teams in season 3 data
unique_teams_season_4 = season_4_data['tmID'].unique()

# Display the unique teams in season 1 data
print("Unique teams in season 4:")
print(unique_teams_season_4)
```

```
Unique teams in season 4:
['MIN' 'ORL' 'NYL' 'LAS' 'MIA' 'UTA' 'PHO' 'IND' 'POR' 'SAC' 'CLE' 'SEA'
 'DET' 'WAS' 'HOU' 'CHA']
```

In [19]:
```
#select season 4 statistic
season_5_data = wnba_data[wnba_data['year'] == 4]
#Get the unique teams in season 4 data
unique_teams_season_5 = season_5_data['tmID'].unique()

# Display the unique teams in season 1 data
print("Unique teams in season 5:")
print(unique_teams_season_5)
```

```
Unique teams in season 5:
['LAS' 'CLE' 'NYL' 'SAS' 'CHA' 'SEA' 'SAC' 'HOU' 'CON' 'MIN' 'DET' 'IND'
 'WAS' 'PHO']
```

In [20]:
```
#select season 5 statistic
season_6_data = wnba_data[wnba_data['year'] == 5]
#Get the unique teams in season 5 data
unique_teams_season_6 = season_6_data['tmID'].unique()

# Display the unique teams in season 1 data
print("Unique teams in season 6:")
print(unique_teams_season_6)
```

```
Unique teams in season 6:
['SEA' 'NYL' 'MIN' 'LAS' 'CON' 'SAC' 'WAS' 'DET' 'SAS' 'CHA' 'IND' 'PHO'
 'HOU']
```

In [21]:
```
#select season 6 statistic
season_7_data = wnba_data[wnba_data['year'] == 6]
#Get the unique teams in season 6 data
unique_teams_season_7 = season_7_data['tmID'].unique()

# Display the unique teams in season 1 data
print("Unique teams in season 7:")
print(unique_teams_season_7)
```

```
Unique teams in season 7:
['PHO' 'SAS' 'NYL' 'SEA' 'SAC' 'LAS' 'CON' 'IND' 'HOU' 'CHA' 'WAS' 'DET'
 'MIN']
```

In [22]:
```
#select season 7 statistic
season_8_data = wnba_data[wnba_data['year'] == 7]
#Get the unique teams in season 7 data
unique_teams_season_8 = season_8_data['tmID'].unique()

# Display the unique teams in season 7 data
print("Unique teams in season 8:")
print(unique_teams_season_8)
```

```
Unique teams in season 8:
['PHO' 'MIN' 'WAS' 'DET' 'HOU' 'NYL' 'SAS' 'SEA' 'CON' 'LAS' 'SAC' 'IND'
 'CHA' 'CHI']
```

In [23]:
```
#select season 8 statistic
season_9_data = wnba_data[wnba_data['year'] == 8]
#Get the unique teams in season 8 data
unique_teams_season_9 = season_9_data['tmID'].unique()

# Display the unique teams in season 8 data
print("Unique teams in season 9:")
print(unique_teams_season_9)
```

```
Unique teams in season 9:
['WAS' 'CHI' 'SAC' 'SAS' 'PHO' 'CON' 'SEA' 'LAS' 'NYL' 'IND' 'MIN' 'DET'
 'HOU']
```

In [24]:
```
#select season 9 statistic
season_10_data = wnba_data[wnba_data['year'] == 9]
#Get the unique teams in season 9 data
unique_teams_season_10 =  season_10_data['tmID'].unique()

# Display the unique teams in season 1 data
print("Unique teams in season 10")
print(unique_teams_season_10)
```

```
Unique teams in season 10
['WAS' 'SAS' 'SEA' 'ATL' 'LAS' 'CON' 'DET' 'MIN' 'HOU' 'IND' 'PHO' 'SAC'
 'CHI' 'NYL']
```

In [25]:
```
teams_season_11_data = pd.read_csv("teams_season11.csv")
players_season_11_data = pd.read_csv("players_teams_season11.csv")
```

In [26]:
```
col_to_drop = ['lgID', 'name', 'arena']

teams_season_11_data = teams_season_11_data.drop(columns=col_to_drop)

col_to_drop2 = ['lgID', 'stint']

players_season_11_data = players_season_11_data.drop(columns=col_to_drop2)
```

In [27]:
```
season_11_data = pd.merge(teams_season_11_data, players_season_11_data, on=['year', 'tmID'])
```

In [28]:
```
# Filter the season 10 data
season_11_data_new = wnba_data[wnba_data['year'] == 10]

# Get the unique teams in season 11 data
unique_teams_season_11 = season_11_data['tmID'].unique()

# Get the unique teams in season 11 data
unique_teams_season_11_new = season_11_data_new['tmID'].unique()



unique_teams_season_11_new
```

Out[28]:
```
array(['DET', 'WAS', 'CHI', 'ATL', 'CON', 'IND', 'PHO', 'NYL', 'MIN',
       'SAC', 'SAS', 'SEA', 'LAS'], dtype=object)
```

In [29]:
```
unique_teams_season_11
```

Out[29]:
```
array(['ATL', 'CHI', 'CON', 'IND', 'LAS', 'MIN', 'NYL', 'PHO', 'SAS',
       'SEA', 'TUL', 'WAS'], dtype=object)
```

In [30]:
```
# Define function to organize the statistics by team ID and return a DataFrame
def parse_statistics_df(season_stats_df):
    team_statistics = []
    for _, row in season_stats_df.iterrows():
        tmID = row['tmID']
        team_stats = row.copy()  # Make a copy of the row
        team_stats['tmID'] = tmID  # Update the team ID
        team_statistics.append(team_stats)
    return pd.DataFrame(team_statistics)
```

In [31]:
```
# Populate season 2 with season 1 statistics
season_2_data = parse_statistics_df(season_2_data)
```

In [32]:
```
season_3_data = parse_statistics_df(season_3_data)
```

In [33]:
```
season_4_data = parse_statistics_df(season_4_data)
```

In [34]:
```
season_5_data = parse_statistics_df(season_5_data)
```

In [35]:
```
season_6_data = parse_statistics_df(season_6_data)
```

In [36]:
```
season_7_data = parse_statistics_df(season_7_data)
```

In [37]:
```
season_8_data = parse_statistics_df(season_8_data)
```

In [38]:
```
season_9_data = parse_statistics_df(season_9_data)
```

In [39]:
```
season_10_data = parse_statistics_df(season_10_data)
```

In [40]:
```
season_11_data = parse_statistics_df(season_11_data_new)
```

In [70]:
```
season_11_data.head()
```

Out[70]:

	year	tmID	franchID	confID	rank	playoff	name	o_fgm	o_fga	o_ftm	...	PostPF	PostfgAttempted	PostfgMade	PostftAttempted	Postf
1711	10	DET	DET	EA	3	1	Detroit Shock	0.692308	0.720805	0.614925	...	0.000000	0.000000	0.000000	0.000000	0.0
1712	10	WAS	WAS	EA	4	1	Washington Mystics	0.594595	0.624161	0.737313	...	0.232558	0.063830	0.085366	0.044118	0.0
1713	10	WAS	WAS	EA	4	1	Washington Mystics	0.594595	0.624161	0.737313	...	0.023256	0.010638	0.012195	0.000000	0.0
1714	10	WAS	WAS	EA	4	1	Washington Mystics	0.594595	0.624161	0.737313	...	0.186047	0.079787	0.024390	0.102941	0.0
1715	10	WAS	WAS	EA	4	1	Washington Mystics	0.594595	0.624161	0.737313	...	0.000000	0.000000	0.000000	0.000000	0.0

```
# Combine all new data frames into one
combined_df = pd.concat([season_2_data, season_3_data, season_4_data, season_5_data,
                         season_6_data, season_7_data, season_8_data, season_9_data,
                        season_10_data, season_11_data])

# Reset index
combined_df.reset_index(drop=True, inplace=True)

# Replace 'Y' with 1 and 'N' with 0 in the 'playoff' column
combined_df['playoff'] = combined_df['playoff'].replace({'Y': 1, 'N': 0})
#combined_df.info()
```

```
from sklearn.preprocessing import MinMaxScaler

# Drop specified columns (these columns contain zero entries)
combined_df = combined_df.drop(columns=['tmORB', 'tmDRB', 'tmTRB',
                                        'opptmORB', 'opptmDRB', 'opptmTRB'])
combined_df.columns[combined_df.columns.duplicated()]

##Scaling using MinMax method
excluded_columns = ['year', 'tmID', 'franchID', 'confID', 'rank',
                    'playoff', 'name', 'arena']
selected_columns = combined_df.columns[~combined_df.columns.isin(excluded_columns)]
scaler = MinMaxScaler()
combined_df[selected_columns] = scaler.fit_transform(combined_df[selected_columns])
combined_df.head()
```

Reasons for Preprocessing Methods

This preprocessing method was chosen for its comprehensive approach to preparing the modeling. By creating a copy of the original data frame, we preserve the initial dataset, ensuring that we can always revert to the original data if needed. Filtering the data for each season and verifying the presence of teams across seasons ensures data consistency and continuity, crucial for accurately updating and utilizing previous season statistics. The "parse statistics" function allows us to effectively generate new data frames based on these statistics, facilitating the creation of a rich longitudinal dataset. Concatenating these data frames into a single data frame streamlines the data structure, making it easier to manage and analyze.

Dropping unnecessary columns (first round, semis, finals) helps in reducing noise and improving model performance. Also, converting the target variable ('playoff') to binary values simplifies the classification task, making it compatible with machine learning algorithms. Finally, normalizing the numerical columns using MinMaxScaler normalizes the data, ensuring that all features are on a similar scale, which is crucial for the performance and convergence of many machine learning models, particularly ridge classifier which is sensitive to feature scaling.

Machine Learning Model

Machine Learning Algorithm

```
In [46]: from sklearn.model_selection import TimeSeriesSplit
         from sklearn.feature_selection import SequentialFeatureSelector
         from sklearn.linear_model import RidgeClassifier
```

```
In [47]: # Split the data into training and testing sets based on the year

         train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 7)]
         test_df = combined_df[(combined_df['year'] >= 8) & (combined_df['year'] <= 10)]


         # Define categorical and numerical features
         categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
         numeric_features = [col for col in train_df.columns if col not in
                             categorical_features + ['playoff']]

         # Separate features and target variable
         all_features = categorical_features.copy()
         all_features.extend(numeric_features)

         X_train = train_df[all_features]
         y_train = train_df['playoff']
         X_test = test_df[all_features]
         y_test = test_df['playoff']
```

```
In [48]: # Check that the columns specified exist in the dataframe
         missing_cols = [col for col in categorical_features +
                         numeric_features if col not in train_df.columns]
         if missing_cols:
             raise ValueError(f"Missing columns in train_df: {missing_cols}")
```

Train and Test Split

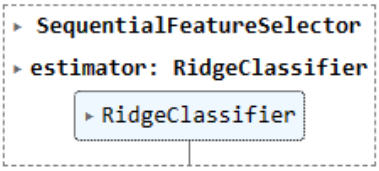
The initial part of the code divides the dataset into training and testing sets based on the year. The training set comprises data from years 1 to 7, while the testing set includes data from years 8 to 10. This temporal split ensures that the model is trained on earlier years and evaluated on later years, which simulates real-world forecasting conditions.

Subsequently, we identify which columns in the dataset are categorical and which are numerical. The categorical features list includes columns containing categorical data, such as team ID, franchise ID, conference ID, team name, and arena. The numeric features list comprises all other columns, excluding those in categorical features and the target variable playoff. We then separate the features and the target variable for both the training and testing sets. This separation includes all relevant features from the original dataset because sequential feature selector in scikit learn will further train different combinations of these features to determine the best subset for the model.

In [52]:
```
from sklearn.model_selection import GridSearchCV
# Define the Ridge Classifier
rr = RidgeClassifier()
# Hyperparameter tuning with GridSearchCV
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}
grid_search = GridSearchCV(rr, param_grid, cv=tscv, scoring='accuracy')
grid_search.fit(train_df[selected_columns], y_train)

# Best Ridge Classifier with optimal alpha
best_rr = grid_search.best_estimator_
```

In [53]:
```
# Initialize TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
# Initialize Sequential Feature Selector
sfs = SequentialFeatureSelector(
    best_rr,  # Ridge Classifier
    n_features_to_select=30,
    direction='forward',
    cv=tscv ) # TimeSeriesSplit for cross-validation

# Convert target to 1-dimensional array
y_train = train_df[target].values.ravel()
#Feature Engineering
# Fit the Sequential Feature Selector
sfs.fit(train_df[selected_columns], y_train)
```

Out[53]:



Feature selection and hyper-parameter tuning

We begin by defining the Ridge Classifier, initially without specifying the alpha parameter. To determine the optimal regularization strength, we perform hyperparameter tuning using grid search cv. This involves testing a range of alpha values (0.01, 0.1, 1, 10, 100) and evaluating their performance through time series cross-validation, ensuring the temporal order of the data is maintained. grid search cv identifies the best alpha value, resulting in the optimal ridge classifier. With this best ridge classifier, we initialize the Sequential Feature Selector, which uses forward selection to iteratively add features until the best 30 features are found. The sequential feature selector fits these features based on the performance of the ridge classifier, ensuring the model is well-suited for predictive analysis.

```
# Iterate over the splits
for train_index, test_index in tscv.split(combined_df):
    # Get the years corresponding to the indices
    train_years = combined_df.iloc[train_index]['year']
    test_years = combined_df.iloc[test_index]['year']

# Check if the conditions for train and test data are met
train_condition = (train_years >=1 ) & (train_years <= 7)
test_condition = (test_years >= 8) & (test_years <= 10)


# Append indices satisfying the conditions
train_indices.append(train_index[train_condition])
test_indices.append(test_index[test_condition])

# Concatenate the indices
final_train_indices = np.concatenate(train_indices)
final_test_indices = np.concatenate(test_indices)

# Use these indices to split your data
train_df = combined_df.iloc[final_train_indices]
test_df = combined_df.iloc[final_test_indices]
```

56]:
```
# Get the selected feature indices
selected_feature_indices = sfs.get_support()

train_df_selected = train_df[selected_columns]
# Get the column names of the selected features
selected_features = train_df_selected.columns[selected_feature_indices]

# Convert to a list if needed
selected_features_list = selected_features.tolist()

# Print or use the selected features
print("Selected features:", selected_features_list)
```

```
Selected features: ['won', 'lost', 'homeW', 'confL', 'stint', 'GP_y', 'points', 'oRebounds', 'dRebounds', 'rebounds', 'assist
s', 'steals', 'blocks', 'turnovers', 'PF', 'fgAttempted', 'fgMade', 'ftAttempted', 'ftMade', 'threeAttempted', 'threeMade', 'd
q', 'PostGP', 'PostdRebounds', 'PostAssists', 'PostBlocks', 'PostPF', 'PostftMade', 'PostthreeMade', 'PostDQ']
```

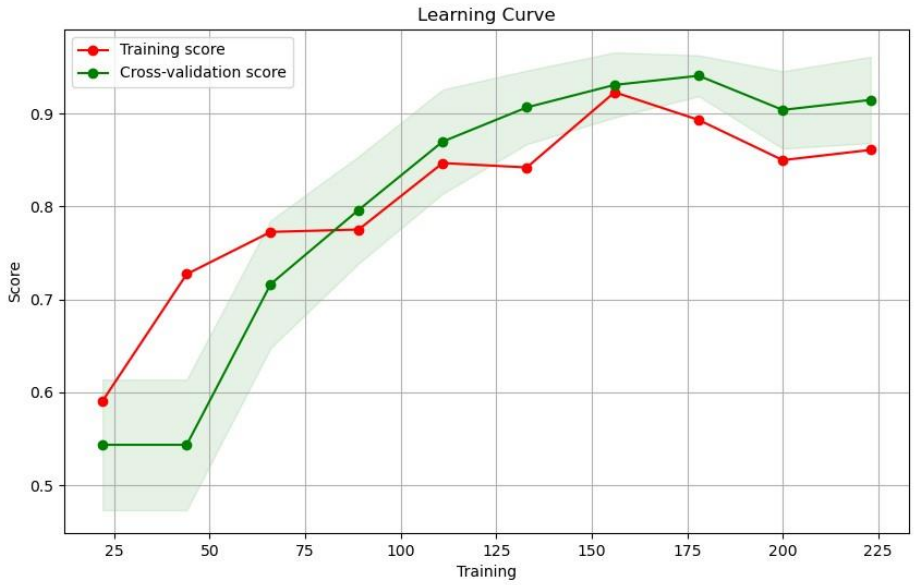
This retrieves the indices of the features selected by the Sequential Feature Selector, filters the training data frame to obtain the names of these features, and converts them into a list format. The selected features include various performance metrics such as points, rebounds, assists, and others, both for regular season and postseason data, indicating their importance in predicting the playoff outcome.

The Learning Curve

A learning curve is generated for a ridge classifier trained on these selected features using time series cross-validation. The x-axis represents the size of the training set, while the y-axis represents the model's performance score. The shaded areas around the curves indicate the variability or uncertainty in the scores, typically represented as the standard deviation.

- **Training score curve (red line):** It shows how well the model fits the training data as the number of training examples increases. A rising curve indicates that the model is learning from the data and improving its performance.
- **Cross-validation score curve (green line):** It demonstrates how well the model generalizes to unseen data (validation data) as the training set size increases. A rising curve indicates that the model's performance on unseen data improves with more training examples.
- **Gap between the curves:** The gap between the training and cross-validation curves indicates the model's generalization performance. A small gap suggests that the

model is generalizing well, while a large gap may indicate overfitting (if the training score is much higher) or underfitting (if the training score is much lower).



Both the training and cross-validation scores start modestly but show an upward trend as more data is incorporated, indicating improved performance with increased training. The convergence of the two scores suggests that the model generalizes well to unseen data, with diminishing differences between training and validation performance as the dataset grows. Towards the end of the curve, both scores stabilize at a high level, indicating that the model has effectively captured the underlying patterns in the data and can make accurate predictions.

Justification for Using the Ridge Classifier

The Ridge Classifier was chosen due to its ability to handle multicollinearity among the features and provide robust predictions. In sports analytics, many performance metrics (e.g., points, rebounds, assists) can be highly correlated, leading to multicollinearity issues that can distort the model's coefficients and reduce predictive accuracy. The Ridge Classifier addresses this by adding a regularization term to the loss function, which penalizes large coefficients and thus mitigates the impact of multicollinearity. This regularization helps in producing a more stable and reliable model, which is particularly important when dealing with complex and interrelated sports performance data.

Additionally, the Ridge Classifier is well-suited for binary classification tasks like predicting playoff outcomes. Its implementation also benefits from Ridge's capability to balance bias and variance through hyperparameter tuning, specifically the alpha parameter. By conducting hyperparameter tuning with grid search cv and evaluating through time series cross-validation, we ensure that the model is optimized for the given dataset's structure. This approach enhances its generalizability to future seasons, making it a robust choice for forecasting playoff status based on past players and team's performance data.

Season 2 Predictions

The first step in predicting teams that made it to the playoffs in season two is to split the data into training and testing sets based on the year. We created train df containing data from year 1 and test df containing data from year 2. Then, the season 2 data was prepared by filtering for records from year 2 and selecting the relevant features defined in selected features list. With these selected features, the pre-trained ridge regression model (best rr) predicts the playoff outcomes for season 2, storing these predictions in the predicted playoff column data frame. Additionally, the actual playoff outcomes were also stored in the actual playoff column.

```
[ ]: # Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] == 1)]
test_df = combined_df[(combined_df['year'] == 2)]
```

```
[93]: season_2_data = test_df[test_df['year'] == 2]
selected_features_season2 = season_2_data[selected_features_list]
X_season2 = selected_features_season2

# Use the trained model to make predictions
predictions_season2 = best_rr.predict(X_season2)
actual_playoff = test_df.loc[:, 'playoff']

season_2_data['Predicted_Playoff'] = predictions_season2
season_2_data['Actual_Playoff'] = actual_playoff
```

```
[94]: # Filter only predicted playoff teams
playoff_teams = season_2_data[season_2_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID',
                                                  'franchID',
                                                  'confID',
                                                  'name']).size().reset_index(
    name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[
    predicted_playoff_counts['confID'] == 'EA'].nlargest(4,
                                                         'predicted_playoff_count')
playoff_teams_western = predicted_playoff_counts[
    predicted_playoff_counts['confID'] == 'WE'].nlargest(4, 'predicted_playoff_count')

# Concatenate the selected teams to get the final playoff teams
season2_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)

# Print the result
season2_playoff_teams
```

To focus only on the teams predicted to make the playoffs, we filtered the season 2 data to include only rows where the predicted playoff status is one. We then used the group by function to count the distinct playoff statuses of each team's players, resulting in a predicted playoff count for each team. To finalize this process, we selected the top four teams from each conference (Eastern and Western), totaling eight teams predicted to make the playoffs for that season. This process will be repeated to predict the playoff teams for all seasons up to season 11.

```
# Print the result
season2_playoff_teams
```

ıt[94]:

	tmID	franchID	confID	name	predicted_playoff_count
0	CHA	CHA	EA	Charlotte Sting	12
1	CLE	CLE	EA	Cleveland Rockers	15
2	MIA	MIA	EA	Miami Sol	12
3	NYL	NYL	EA	New York Liberty	15
4	HOU	HOU	WE	Houston Comets	11
5	LAS	LAS	WE	Los Angeles Sparks	11
6	SAC	SAC	WE	Sacramento Monarchs	13
7	UTA	SAS	WE	Utah Starzz	9

Season 2 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	CHA	CHA	EA	Charlotte Sting
2	CLE	CLE	EA	Cleaveland Rockers
3	MIA	MIA	EA	Miami Sol
4	NYL	NYL	EA	New York Liberty
5	HOU	HOU	WE	Houston Comets
6	LAS	LAS	WE	Los Angeles Sparks
7	SAC	SAC	WE	Sacramento Monarchs
8	UTA	SAS	WE	Utah Starzz

Season 3 Predictions

Season 3 Predictions

```
# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 2)]
test_df = combined_df[(combined_df['year'] == 3)]
```

```
# Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]
```

```
season_3_data = test_df[test_df['year'] == 3]
selected_features_season3 = season_3_data[selected_features_list]
X_season3 = selected_features_season3

# Use the trained model to make predictions
predictions_season3 = best_rr.predict(X_season3)
actual_playoff = test_df.loc[:, 'playoff']

season_3_data['Predicted_Playoff'] = predictions_season3
season_3_data['Actual_Playoff'] = actual_playoff
```

```
# Filter only predicted playoff teams
playoff_teams = season_3_data[season_3_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID',
                                                  'franchID',
                                                  'confID',
                                                  'name']).size().reset_index(
    name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[
    predicted_playoff_counts['confID'] == 'EA'].nlargest(4,
                                                         'predicted_playoff_count')
playoff_teams_western = predicted_playoff_counts[
    predicted_playoff_counts['confID'] == 'WE'].nlargest(4, 'predicted_playoff_count')

# Concatenate the selected teams to get the final playoff teams
season3_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)

# Print the result
season3_playoff_teams
```

The train set used was the data from season 1 and 2 while the test set contained season 3 data. We select the relevant features from the season 3 data and assign them to selected features season3. We go further by assigning these selected features to X season3. Next, we

use the trained model (best rr) to predict whether each team in season 3 will make it to the playoffs, based on their features.

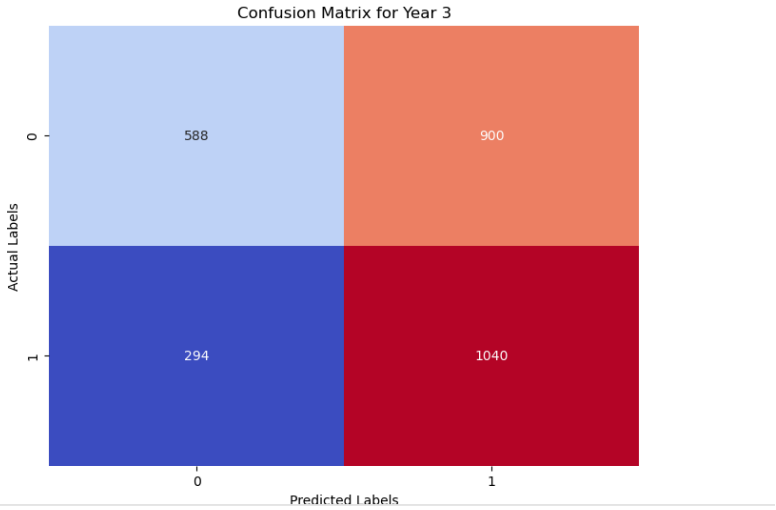
```
playoff_teams_western = predicted_playoff_counts[
    predicted_playoff_counts['confID'] == 'WE'].nlargest(4, 'predicted_playof

# Concatenate the selected teams to get the final playoff teams
season3_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_weste

# Print the result
season3_playoff_teams
```

t[103]:

	tmID	franchID	confID	name	predicted_playoff_count
0	WAS	WAS	EA	Washington Mystics	13
1	CHA	CHA	EA	Charlotte Sting	12
2	IND	IND	EA	Indiana Fever	12
3	NYL	NYL	EA	New York Liberty	11
4	HOU	HOU	WE	Houston Comets	14
5	LAS	LAS	WE	Los Angeles Sparks	13
6	UTA	SAS	WE	Utah Starzz	13
7	SEA	SEA	WE	Seattle Storm	6

Season 3 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	WAS	WAS	EA	Washington Mystics
2	CHA	CHA	EA	Charloette Stings
3	IND	IND	EA	Indiana Fever
4	NYL	NYL	EA	New York Liberty
5	HOU	HOU	WE	Houston Comets
6	LAS	LAS	WE	Los Angeles Sparks
7	UTA	SAS	WE	Utah Starzz
8	SEA	SEA	WE	Seattle Storm

```
e_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse`
  warnings.warn(

Skipping year 2 due to insufficient training data.
Year: 3
Accuracy: 0.5768958185683912
Recall: 0.7796101949025487
Precision: 0.5360824742268041
F1 Score: 0.6353084911423336
```



The model accuracy for season 4 is approximately 0.577, indicating that around 57.7% of the predictions were correct. The recall, which measures the ability of the model to correctly identify positive cases, stands at approximately 0.780, suggesting that the model captured about 78.0% of the actual positive cases. The precision, representing the proportion of correctly identified positive cases out of all cases identified as positive by the model, is approximately 0.536, showing that around 53.6% of the predicted positive cases were true positives. Lastly, the F1 Score, which is the harmonic mean of precision and recall, is approximately 0.635, indicating a balance between precision and recall.

Season 4 Predictions

Season 4 Predictions

```
[104]: # Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 3)]
test_df = combined_df[(combined_df['year'] == 4)]
```

```
[ ]: # Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]

# Separate features and target variable
all_features = categorical_features.copy()
all_features.extend(numeric_features)
```

```
In [108]: season_4_data = test_df[test_df['year'] == 4]
selected_features_season4 = season_4_data[selected_features_list]
X_season4 = selected_features_season4

# Use the trained model to make predictions
predictions_season4 = best_rr.predict(X_season4)
actual_playoff = test_df.loc[:, 'playoff']

season_4_data['Predicted_Playoff'] = predictions_season4
season_4_data['Actual_Playoff'] = actual_playoff
```

```
In [110]: # Filter only predicted playoff teams
playoff_teams = season_4_data[season_4_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID',
                                                  'franchID', 'confID', 'name']
                                                 ).size().reset_index(name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'EA']
playoff_teams_western = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'WE']

# Concatenate the selected teams to get the final playoff teams
season4_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)
```

```
In [111]: season4_playoff_teams = season4_playoff_teams[['tmID', 'franchID', 'confID', 'name']]
```

```
In [112]: # Print the result
season4_playoff_teams
```

```
Out[112]:
```

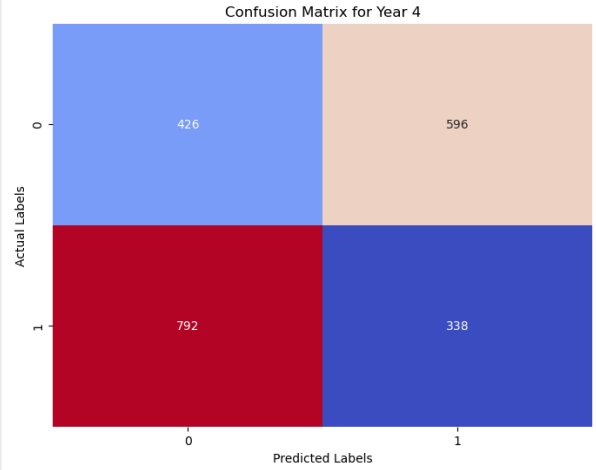
```
n [112]: # Print the result
         season4_playoff_teams
```

ut[112]:

	tmID	franchID	confID	name
0	CHA	CHA	EA	Charlotte Sting
1	CLE	CLE	EA	Cleveland Rockers
2	CON	CON	EA	Connecticut Sun
3	DET	DET	EA	Detroit Shock
4	HOU	HOU	WE	Houston Comets
5	LAS	LAS	WE	Los Angeles Sparks
6	MIN	MIN	WE	Minnesota Lynx
7	SAC	SAC	WE	Sacramento Monarchs

Season 4 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	CHA	CHA	EA	Charlotte Sting
2	CLE	CLE	EA	Cleaveland Rockers
3	CON	CON	EA	Connecticut Sun
4	DET	DET	EA	Detroit Shock
5	HOU	HOU	WE	Houston Comets
6	LAS	LAS	WE	Los Angeles Sparks
7	MIN	MIN	WE	Minnesota Lynx
8	SAC	SAC	WE	Sacramento Monarchs

```
Year: 4
Accuracy: 0.3550185873605948
Recall: 0.2991150442477876
Precision: 0.3618843683083512
F1 Score: 0.32751937984496127
```



Season 4 prediction performance metrics

Accuracy	0.36
Precision	0.36
Recall	0.30
F1 score	0.35

Season 5 Predictions

Season 5 Predictions

```
# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 4)]
test_df = combined_df[(combined_df['year'] == 5)]
```

```
# Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]

# Separate features and target variable
all_features = categorical_features.copy()
all_features.extend(numeric_features)
```

```
season_5_data = test_df[test_df['year'] == 5]
selected_features_season5 = season_5_data[selected_features_list]
X_season5 = selected_features_season5

# Use the trained model to make predictions
predictions_season5 = best_rr.predict(X_season5)
actual_playoff = test_df.loc[:, 'playoff']

season_5_data['Predicted_Playoff'] = predictions_season5
season_5_data['Actual_Playoff'] = actual_playoff
```

```
# Filter only predicted playoff teams
playoff_teams = season_5_data[season_5_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID', 'franchID', 'confID', 'name']
                                                 ).size().reset_index(name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'EA']
playoff_teams_western = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'WE']

# Concatenate the selected teams to get the final playoff teams
season5_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)
```

```
season5_playoff_teams = season5_playoff_teams[['tmID', 'franchID', 'confID', 'name']]
```

```
In [119]: season5_playoff_teams = season5_playoff_teams[['tmID'
```

```
In [120]: # Print the result
season5_playoff_teams
```

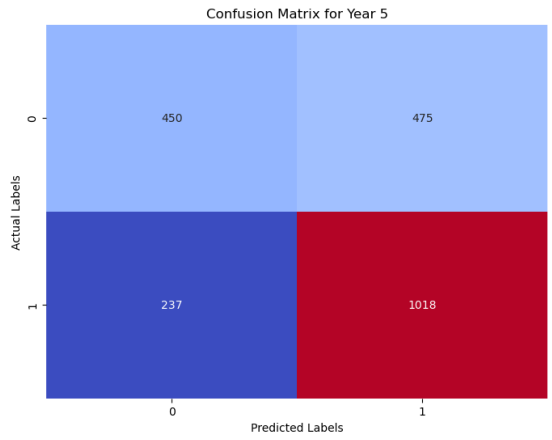
Out[120]:

	tmID	franchID	confID	name
0	CON	CON	EA	Connecticut Sun
1	DET	DET	EA	Detroit Shock
2	NYL	NYL	EA	New York Liberty
3	WAS	WAS	EA	Washington Mystics
4	LAS	LAS	WE	Los Angeles Sparks
5	MIN	MIN	WE	Minnesota Lynx
6	SAC	SAC	WE	Sacramento Monarchs
7	SEA	SEA	WE	Seattle Storm

Season 5 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	CON	CON	EA	Connecticut Sun
2	DET	DET	EA	Detroit Shock
3	NYL	NYL	EA	New York Liberty
4	WAS	WAS	EA	Washington Mystics
5	LAS	LAS	WE	Los Angeles Sparks
6	MIN	MIN	WE	Minnesota Lynx
7	SAC	SAC	WE	Sacramento Monarchs
8	SEA	SEA	WE	Seattle Storm

Predicted Labels

```
Year: 5
Accuracy: 0.673394495412844
Recall: 0.8111553784860558
Precision: 0.6818486269256531
F1 Score: 0.7409024745269287
```



WNBA

Season 5 prediction performance metrics	
Accuracy	0.67
Precision	0.68
Recall	0.81
F1 score	0.74

Season 6 Predictions

Season 6 Predictions

```
# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 5)]
test_df = combined_df[(combined_df['year'] == 6)]
```

```
# Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]
```

```
season_6_data = test_df[test_df['year'] == 6]
selected_features_season6 = season_6_data[selected_features_list]
X_season6 = selected_features_season6

# Use the trained model to make predictions
predictions_season6 = best_rr.predict(X_season6)
actual_playoff = test_df.loc[:, 'playoff']

season_6_data['Predicted_Playoff'] = predictions_season6
season_6_data['Actual_Playoff'] = actual_playoff
```

```
# Filter only predicted playoff teams
playoff_teams = season_6_data[season_6_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID', 'franchID', 'confID', 'name']
                                                 ).size().reset_index(name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'EA']
playoff_teams_western = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'WE']

# Concatenate the selected teams to get the final playoff teams
season6_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True
```

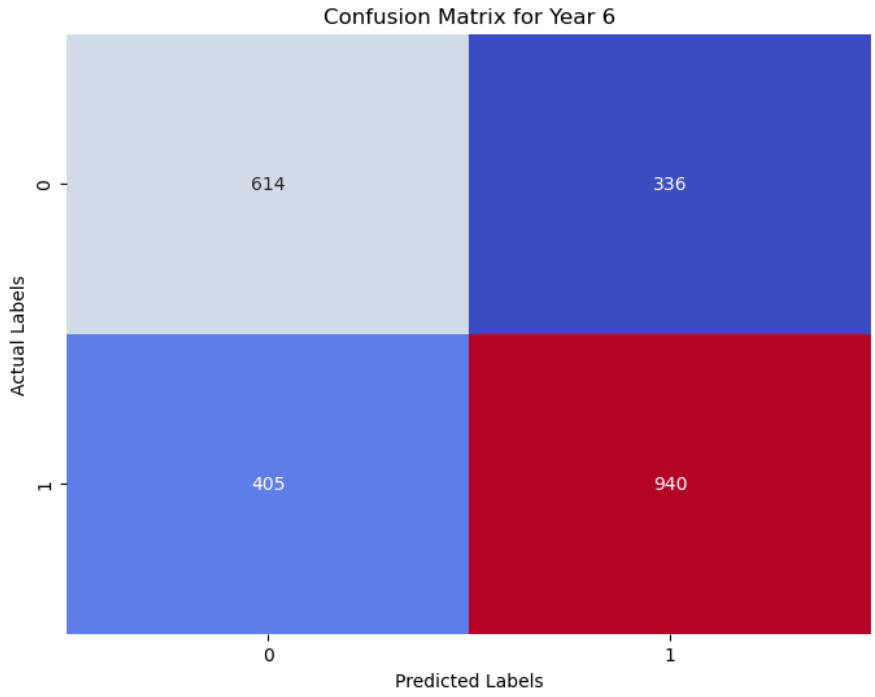
[128]:
```
# Print the result
season6_playoff_teams
```

[128]:

	tmID	franchID	confID	name
0	CON	CON	EA	Connecticut Sun
1	DET	DET	EA	Detroit Shock
2	IND	IND	EA	Indiana Fever
3	NYL	NYL	EA	New York Liberty
4	HOU	HOU	WE	Houston Comets
5	LAS	LAS	WE	Los Angeles Sparks
6	SAC	SAC	WE	Sacramento Monarchs
7	SEA	SEA	WE	Seattle Storm

Season 6 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	CON	CON	EA	Connecticut Sun
2	DET	DET	EA	Detroit Shock
3	IND	IND	EA	Indiana Fever
4	NYL	NYL	EA	New York Liberty
5	HOU	HOU	WE	Houston Comets
6	LAS	LAS	WE	Los Angeles Sparks
7	SAC	SAC	WE	Sacramento Monarchs
8	SEA	SEA	WE	Seattle Storm

```
Year: 6
Accuracy: 0.677124183006536
Recall: 0.6988847583643123
Precision: 0.7366771159874608
F1 Score: 0.7172834795879436
```



Season 6 prediction performance metrics	
Accuracy	0.67
Precision	0.73
Recall	0.70
F1 score	0.71

Season 7 Predictions

Season 7 Predictions

[129]:
```
# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 6)]
test_df = combined_df[(combined_df['year'] >= 7) & (combined_df['year'] <= 10)]
```

[130]:
```
# Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]

# Separate features and target variable
```

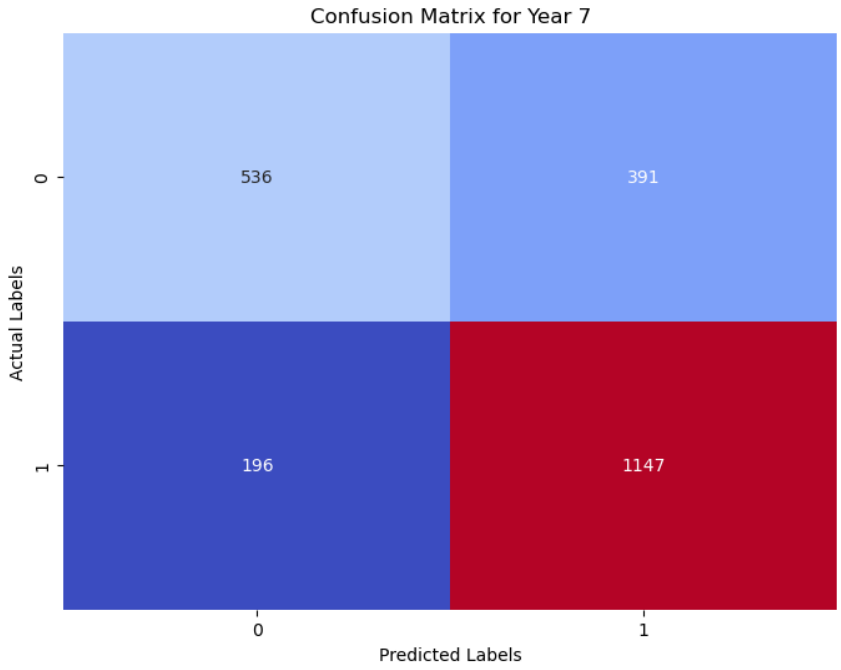
In [142]:
```
# Print the result
season7_playoff_teams
```

Out[142]:

	tmID	franchID	confID	name	predicted_playoff_count
0	ATL	ATL	EA	Atlanta Dream	12
1	CON	CON	EA	Connecticut Sun	39
2	DET	DET	EA	Detroit Shock	63
3	IND	IND	EA	Indiana Fever	53
4	NYL	NYL	EA	New York Liberty	24
5	WAS	WAS	EA	Washington Mystics	18
6	HOU	HOU	WE	Houston Comets	24
7	LAS	LAS	WE	Los Angeles Sparks	40
8	PHO	PHO	WE	Phoenix Mercury	25
9	SAC	SAC	WE	Sacramento Monarchs	35
10	SAS	SAS	WE	San Antonio Silver Stars	29
11	SEA	SEA	WE	Seattle Storm	54

Season 7 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	DET	DET	EA	Detroit Shock
2	IND	IND	EA	Indiana Fever
3	CON	CON	EA	Connecticut Sun
4	NYL	NYL	EA	New York Liberty
5	SEA	SEA	WE	Seattle Storm
6	LAS	LAS	WE	Los Angeles Sparks
7	SAC	SAC	WE	Sacramento Monarchs
8	PHO	PHO	WE	Phoenix Mercury

```
Year: 7
Accuracy: 0.741409691629956
Recall: 0.8540580789277736
Precision: 0.7457737321196359
F1 Score: 0.796251301631378
```



Season 7 prediction performance metrics	
Accuracy	0.74
Precision	0.75
Recall	0.85
F1 score	0.80

Season 8 Predictions

92]:
```
# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 7)]
test_df = [(combined_df['year'] >= 8) & (combined_df['year'] <= 10)]
```

93]:
```
# Define categorical and numerical features
categorical_features = ['tmID', 'franchID', 'confID', 'name', 'arena']
numeric_features = [col for col in train_df.columns if col not in
                    categorical_features + ['playoff']]

# Separate features and target variable
all_features = categorical_features.copy()
all_features.extend(numeric_features)

X_train = train_df[all_features]
y_train = train_df['playoff']
```

```
season_8_data = test_df[(test_df['year'] >= 8) & (test_df['year'] <= 10)]
selected_features_season8 = season_8_data[selected_features_list]
X_season8 = selected_features_season8

# Use the trained model to make predictions
predictions_season8 = best_rr.predict(X_season8)
actual_playoff = test_df.loc[:, 'playoff']

season_8_data['Predicted_Playoff'] = predictions_season8
season_8_data['Actual_Playoff'] = actual_playoff
```

```
# Filter only predicted playoff teams
playoff_teams = season_8_data[season_8_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID',
                                                  'franchID', 'confID',
                                                  'name']).size().reset_index(name='predicted_playoff_count')

# Separate teams by conference
playoff_teams_eastern = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'EA']
playoff_teams_western = predicted_playoff_counts[predicted_playoff_counts['confID'] == 'WE']

# Concatenate the selected teams to get the final playoff teams
season8_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)
```

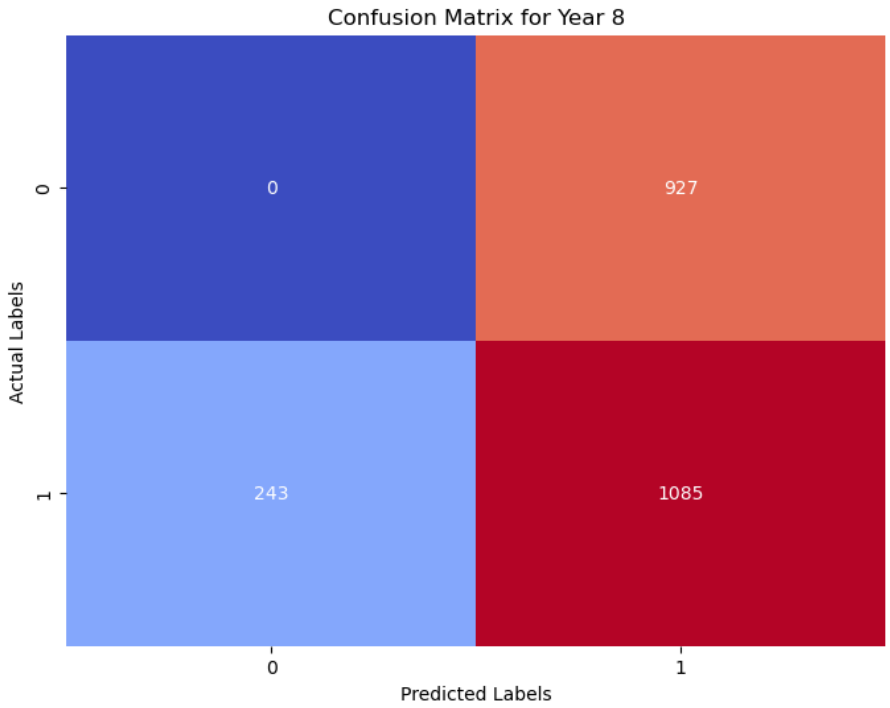
```
7]: season8_playoff_teams
```

7]:

	tmID	franchID	confID	name
0	CON	CON	EA	Connecticut Sun
1	DET	DET	EA	Detroit Shock
2	IND	IND	EA	Indiana Fever
3	NYL	NYL	EA	New York Liberty
4	PHO	PHO	WE	Phoenix Mercury
5	SAC	SAC	WE	Sacramento Monarchs
6	SAS	SAS	WE	San Antonio Silver Stars
7	SEA	SEA	WE	Seattle Storm

Season 8 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	CON	CON	EA	Connecticut Sun
2	DET	DET	EA	Detroit Shock
3	IND	IND	EA	Indiana Fever
4	NYL	NYL	EA	New York Liberty
5	PHO	PHO	WE	Phoenix Mercury
6	SAC	SAC	WE	Sacramento Monarchs
7	SAS	SAS	WE	San Antonio Silver Stars
8	SEA	SEA	WE	Settle Storm

```
Year: 8
Accuracy: 0.4811529933481153
Recall: 0.8170180722891566
Precision: 0.5392644135188867
F1 Score: 0.6497005988023952
```



Season 8 prediction performance metrics	
Accuracy	0.48
Precision	0.54
Recall	0.82
F1 score	0.65

Season 9 Predictions

Season 9 Predictions

```
# Split the data into training and testing sets based on the year

# Ensure 'year' column is numeric
combined_df['year'] = pd.to_numeric(combined_df['year'], errors='coerce')

# Split the data into training and testing sets based on the year
train_df = combined_df[(combined_df['year'] >= 1) & (combined_df['year'] <= 7)]
test_df = combined_df[(combined_df['year'] >= 9) & (combined_df['year'] <= 10)]


# Verify the split
print("Unique years in train_df:", train_df['year'].unique())
print("Unique years in test_df:", test_df['year'].unique())
```

```
season_9_data = test_df[(test_df['year'] >= 9) & (test_df['year'] <= 10)]
selected_features_season9 = season_9_data[selected_features_list]
X_season9 = selected_features_season9
```

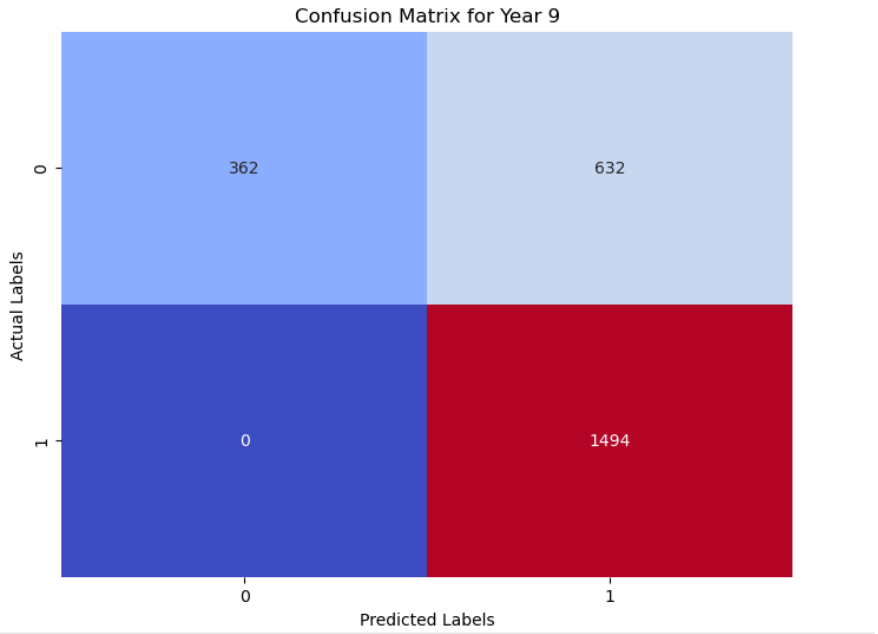
88]:
```
season9_playoff_teams
```

88]:

	tmID	franchID	confID	name	predicted_playoff_count
0	ATL	ATL	EA	Atlanta Dream	12
1	CON	CON	EA	Connecticut Sun	9
2	DET	DET	EA	Detroit Shock	36
3	IND	IND	EA	Indiana Fever	29
4	NYL	NYL	EA	New York Liberty	13
5	WAS	WAS	EA	Washington Mystics	6
6	HOU	HOU	WE	Houston Comets	12
7	LAS	LAS	WE	Los Angeles Sparks	17
8	PHO	PHO	WE	Phoenix Mercury	12
9	SAC	SAC	WE	Sacramento Monarchs	13
10	SAS	SAS	WE	San Antonio Silver Stars	10
11	SEA	SEA	WE	Seattle Storm	13

Season 9 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	ATL	ATL	EA	Atlanta Dream
2	DET	DET	EA	Detroit Shock
3	IND	IND	EA	Indiana Fever
4	NYL	NYL	EA	New York Liberty
5	PHO	PHO	WE	Phoenix Mercury
6	SAC	SAC	WE	Sacramento Monarchs
7	SAS	SAS	WE	San Antonio Silver Stars
8	SEA	SEA	WE	Settle Storm

```
Year: 9
Accuracy: 0.7459807073954984
Recall: 1.0
Precision: 0.702728127939793
F1 Score: 0.8254143646408839
```



Season 9 prediction performance metrics	
Accuracy	0.74
Precision	0.70
Recall	1.0
F1 score	0.83

Season 10 Predictions

```
season_10_data = test_df[(test_df['year'] >= 9) & (test_df['year'] <= 10)]
selected_features_season10 = season_10_data[selected_features_list]
X_season10 = selected_features_season10
```

```
# Use the trained model to make predictions
predictions_season10 = best_rr.predict(X_season10)
actual_playoff = test_df.loc[:, 'playoff']
```

```
# Assign predicted playoff and actual playoff values using .loc
season_10_data.loc[:, 'Predicted_Playoff'] = predictions_season10
season_10_data.loc[:, 'Actual_Playoff'] = actual_playoff
```

```
# Filter only predicted playoff teams
playoff_teams = season_10_data[season_10_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID', 'franchID', 'confID', 'name']
                                                 ).size().reset_index(name='predicted_playoff_count')

# Count the number of actual playoff appearances for each team
actual_playoff_counts = season_11_data.groupby(['tmID',
                                                'franchID', 'confID', 'name']
                                               )['playoff'].size().reset_index(name='actual_playoff_count')

# Merge the actual and predicted playoff counts
playoff_counts = pd.merge(predicted_playoff_counts, actual_playoff_counts, on=
                          ['tmID', 'franchID', 'confID', 'name'], how='left')
```

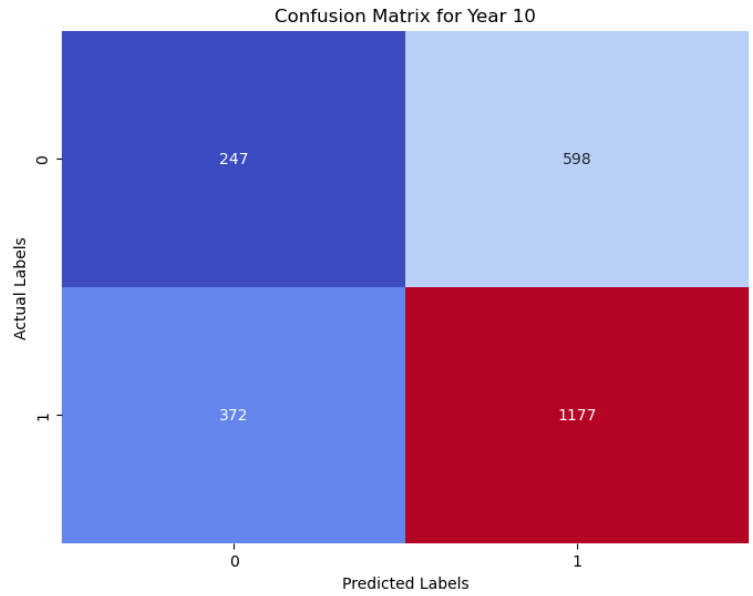
n [69]:
```
# Display the final selected playoff teams
season10_playoff_teams
```

ut[69]:

	tmID	franchID	confID	name	predicted_playoff_count
0	ATL	ATL	EA	Atlanta Dream	12
1	DET	DET	EA	Detroit Shock	17
2	IND	IND	EA	Indiana Fever	14
3	WAS	WAS	EA	Washington Mystics	6
4	LAS	LAS	WE	Los Angeles Sparks	11
5	PHO	PHO	WE	Phoenix Mercury	12
6	SAS	SAS	WE	San Antonio Silver Stars	7
7	SEA	SEA	WE	Seattle Storm	13

Season 10 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	DET	DET	EA	Detroit Shock
2	IND	IND	EA	Indiana Fever
3	NYL	NYL	EA	New York Liberty
4	WAS	WAS	EA	Washington Mystics
5	LAS	LAS	WE	Los Angeles Sparks
6	SEA	SEA	WE	Seattle Storm
7	SAS	SAS	WE	San Antonio Silver Stars
8	PHO	PHO	WE	Phoenix Mercury

```
Year: 10
Accuracy: 0.5948203842940685
Recall: 0.7598450613298903
Precision: 0.6630985915492957
F1 Score: 0.7081829121540313
```



Season 10 prediction performance metrics	
Accuracy	0.59
Precision	0.66
Recall	0.76
F1 score	0.71

Season 11 Predictions

Season 11 predictions

In [56]:
```
season_11_data = test_df[test_df['year'] == 10]
selected_features_season11 = season_11_data[selected_features_list]
X_season11 = selected_features_season11
```

In [57]:
```
# Use the trained model to make predictions
predictions_season11 = best_rr.predict(X_season11)
actual_playoff = test_df.loc[:, 'playoff']
```

In [202]:
```
season_11_data['Predicted_Playoff'] = predictions_season11
season_11_data['Actual_Playoff'] = actual_playoff

# Display the updated DataFrame
#season_11_data
```

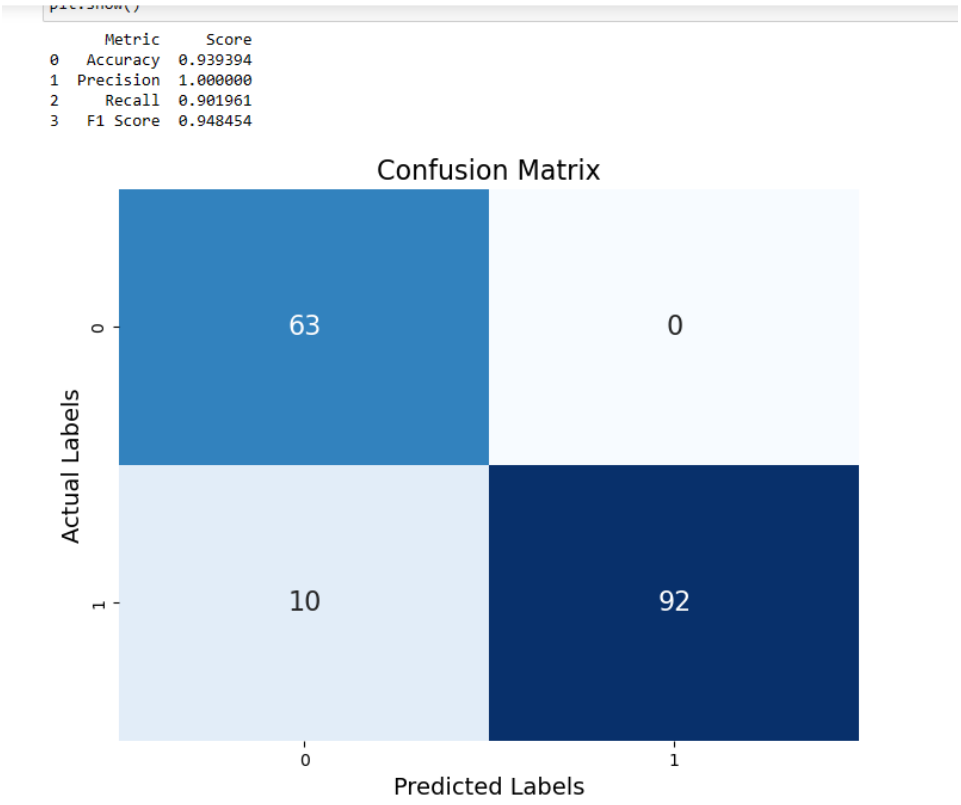
In [203]:
```
# Filter only predicted playoff teams
playoff_teams = season_11_data[season_11_data['Predicted_Playoff'] == 1]

# Count the number of predicted playoff appearances for each team
predicted_playoff_counts = playoff_teams.groupby(['tmID', 'franchID',
                                                  'confID', 'name']).size().reset_index(name='predicted_play

# Merge the actual and predicted playoff counts
playoff_counts = pd.merge(predicted_playoff_counts, actual_playoff_counts,
                          on=['tmID', 'franchID', 'confID', 'name'], how='left')

# Select the top 4 teams from each conference based on predicted playoff counts
playoff_teams_eastern = playoff_counts[playoff_counts['confID'] == 'EA']
playoff_teams_western = playoff_counts[playoff_counts['confID'] == 'WE']

# Concatenate the selected teams to get the final playoff teams
final_playoff_teams = pd.concat([playoff_teams_eastern, playoff_teams_western]).reset_index(drop=True)
```

In [204]:
```
# Display the final selected playoff teams with their actual and predicted playoff counts
final_playoff_teams
```

```
# Display the final selected playoff teams
final_playoff_teams
```

	tmID	franchID	confID	name
0	ATL	ATL	EA	Atlanta Dream
1	TUL	DET	EA	Detroit Shock
2	IND	IND	EA	Indiana Fever
3	WAS	WAS	EA	Washington Mystics
4	LAS	LAS	WE	Los Angeles Sparks
5	PHO	PHO	WE	Phoenix Mercury
6	SAS	SAS	WE	San Antonio Silver Stars
7	SEA	SEA	WE	Seattle Storm

Season 11 Predicted Playoff Teams				
S/N	Team ID	Franchise ID	Conf ID	Name
1	ATL	ATL	EA	Atlanta Dream
2	TUL	DET	EA	Detroit Shock
3	IND	IND	EA	Indiana Fever
4	WAS	WAS	EA	Washington Mystics
5	PHO	PHO	WE	Phoenix Mercury
6	LAS	LAS	WE	Los Angeles Sparks
7	SAS	SAS	WE	San Antonio Silver Stars
8	SEA	SEA	WE	Seattle Storm



Predictive Power of the Machine Learning Model

Despite inherent complexities and variations in team performance metrics, the model consistently produced reasonably accurate predictions, with its highest performance metrics being achieved in later seasons. The effectiveness was particularly evident in

seasons like 7 and 9, where accuracy, precision, recall, and F1 scores were notably high. This suggests that the Ridge Classifier, combined with comprehensive feature selection and rigorous preprocessing, effectively captured the underlying patterns in the data. The gradual improvement in performance metrics over the seasons indicates that the model benefited from the increasing amount of data and refined feature selection, which enhanced its ability to generalize from historical trends to future outcomes.