Tampa Bay Lightning Performance Analysis: Model Interpretability Project

Business Problem:

For the Tampa Bay Lightning management, I aim to identify which player performance metrics most significantly contribute to team success. This analysis will help inform decisions on player development, lineup optimization, and talent acquisition, particularly valuable for a team like the Lightning who need to maintain high performance while managing salary cap constraints.

In [4]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.ensemble import RandomForestRegressor, GradientBoostingI
from sklearn.metrics import mean_squared_error, r2_score

# Set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

# Load the datasets
player_data = pd.read_csv('TBL_Player.csv')
team_data = pd.read_csv('TBL_Team.csv')

# Display basic information about the datasets
print("Tampa Bay Lightning player data shape:", player_data.shape)
print("Tampa Bay Lightning team data shape:", team_data.shape)

# Basic statistics for key columns in player data
print("\nBasic statistics for player data:")
player_stats = player_data[['playerId', 'gameId', 'icetime', 'gameSc
print(player_stats)

# Basic statistics for key columns in team data
print("\nBasic statistics for team data:")
team_stats = team_data[['gameId', 'xGoalsPercentage', 'goalsFor', 'g
print(team_stats)
```

```python
# Let's see the unique values in some categorical columns
print("\nUnique player positions:")
print(player_data['position'].unique())

print("\nUnique situation types:")
print(player_data['situation'].unique())

# Let's check how many unique games we have
print("\nNumber of unique games in player data:", player_data['gameI
print("Number of unique games in team data:", team_data['gameId'].nu

# Check the date range of the games
print("\nGame date range in player data:")
print("Min date:", player_data['gameDate'].min())
print("Max date:", player_data['gameDate'].max())
```

```
Tampa Bay Lightning player data shape: (6390, 156)
Tampa Bay Lightning team data shape: (355, 109)

Basic statistics for player data:
           playerId        gameId      icetime    gameScore    I_F_go
als  \
count  6.390000e+03  6.390000e+03  6390.000000  6390.000000  6390.000
000
mean   8.478587e+06  2.024021e+09   396.178717     0.433178     0.077
778
std    2.609142e+03  3.336834e+02   458.294595     0.751650     0.295
091
min    8.474151e+06  2.024020e+09     0.000000    -1.700000     0.000
000
25%    8.476826e+06  2.024020e+09     0.000000     0.000000     0.000
000
50%    8.478178e+06  2.024021e+09   118.000000     0.100000     0.000
000
75%    8.480246e+06  2.024021e+09   806.000000     0.790000     0.000
000
max    8.483447e+06  2.024021e+09  1725.000000     5.075000     3.000
000

        I_F_points
count  6390.000000
mean      0.208607
std       0.527077
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max       6.000000

Basic statistics for team data:
             gameId  xGoalsPercentage    goalsFor  goalsAgainst
count  3.550000e+02        355.000000  355.000000    355.000000
```

```
mean   2.024021e+09         0.525330    1.402817      1.070423
std    3.341282e+02         0.321739    1.734819      1.435269
min    2.024020e+09         0.000000    0.000000      0.000000
25%    2.024020e+09         0.340000    0.000000      0.000000
50%    2.024021e+09         0.538200    1.000000      1.000000
75%    2.024021e+09         0.760750    2.000000      2.000000
max    2.024021e+09         1.000000    8.000000      7.000000

Unique player positions:
['L' 'D' 'C' 'R']

Unique situation types:
['other' 'all' '5on5' '4on5' '5on4']

Number of unique games in player data: 71
Number of unique games in team data: 71

Game date range in player data:
Min date: 20241011
Max date: 20250325
```

In [5]:

```python
# Create target variables for team success
team_data['goal_differential'] = team_data['goalsFor'] - team_data['g
team_data['won_game'] = (team_data['goal_differential'] > 0).astype(i

# Filter to focus on 'all' situation (all game situations combined)
team_all = team_data[team_data['situation'] == 'all']

# Show a sample of our target variables
print("Sample of team data with target variables:")
print(team_all[['gameId', 'opposingTeam', 'goalsFor', 'goalsAgainst',
                'goal_differential', 'won_game']].head(5))

# Check win-loss record
win_count = team_all['won_game'].sum()
loss_count = len(team_all) - win_count
print(f"\nLightning record: {win_count}-{loss_count}")
print(f"Win percentage: {win_count/len(team_all)*100:.1f}%")

# Let's look at which features are most predictive of winning
correlations = []
for column in team_all.columns:
    if column not in ['gameId', 'team', 'name', 'playerTeam', 'oppos
                      'home_or_away', 'gameDate', 'position', 'situat
                      'won_game', 'goal_differential']:
        correlation = team_all[column].corr(team_all['won_game'])
        if not pd.isna(correlation):  # Skip columns with NaN correla
            correlations.append((column, correlation))

# Sort by absolute correlation
correlations.sort(key=lambda x: abs(x[1]), reverse=True)
```

```python
# Print top 10 correlated features
print("\nTop 10 team features correlated with winning:")
for feature, corr in correlations[:10]:
    print(f"{feature}: {corr:.3f}")
```

```
Sample of team data with target variables:
        gameId opposingTeam  goalsFor  goalsAgainst  goal_differentia
l  \
1   2024020020          CAR       4.0           1.0                3.
0
6   2024020048          VAN       4.0           1.0                3.
0
11  2024020064          VGK       4.0           3.0                1.
0
16  2024020075          OTT       4.0           5.0               -1.
0
21  2024020091          TOR       2.0           5.0               -3.
0

    won_game
1          1
6          1
11         1
16         0
21         0

Lightning record: 39-32
Win percentage: 54.9%

Top 10 team features correlated with winning:
goalsFor: 0.650
goalsAgainst: -0.626
xGoalsPercentage: 0.475
lowDangerGoalsAgainst: -0.458
scoreFlurryAdjustedTotalShotCreditFor: 0.451
scoreAdjustedTotalShotCreditFor: 0.427
highDangerGoalsFor: 0.423
flurryScoreVenueAdjustedxGoalsFor: 0.410
highDangerGoalsAgainst: -0.410
totalShotCreditFor: 0.404
```

Player Performance Analysis

Now I'll analyze individual player performance metrics and their relationship to team success. This analysis will help identify which players and performance indicators most significantly contribute to Lightning wins.

In [6]:
```python
# Let's identify key Lightning players and their performance
# First, filter player data for 'all' situation to match team data
```

```python
player_all = player_data[player_data['situation'] == 'all']

# Find players with the most ice time (likely top players)
player_icetime = player_all.groupby('name')['icetime'].sum().sort_va
print("Top 10 players by total ice time:")
print(player_icetime.head(10))

# Look at top point producers
player_points = player_all.groupby('name')['I_F_points'].sum().sort_
print("\nTop 10 point producers:")
print(player_points.head(10))

# Look at average game score by player (minimum 10 games)
player_games = player_all.groupby('name')['gameId'].nunique()
player_gamescore = player_all.groupby('name')['gameScore'].mean()
player_performance = pd.DataFrame({
    'games_played': player_games,
    'avg_gamescore': player_gamescore
})
player_performance = player_performance[player_performance['games_pla
print("\nTop 10 players by average Game Score (min. 10 games):")
print(player_performance.head(10))

# Let's visualize the distribution of Game Scores for top players
top_players = player_performance.head(5).index.tolist()

plt.figure(figsize=(12, 6))
for player in top_players:
    player_scores = player_all[player_all['name'] == player]['gameSc
    sns.kdeplot(player_scores, label=player)

plt.title('Distribution of Game Scores for Top Lightning Players')
plt.xlabel('Game Score')
plt.ylabel('Density')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.savefig('top_player_gamescore_distribution.png')
plt.close()
```

```
Top 10 players by total ice time:
name
Victor Hedman       94901.0
Brandon Hagel       89469.0
Ryan McDonagh       87714.0
Nikita Kucherov     86039.0
Jake Guentzel       84151.0
Anthony Cirelli     78026.0
Brayden Point       77838.0
Erik Cernak         71710.0
Darren Raddysh      66827.0
Nick Paul           63547.0
```

```
Name: icetime, dtype: float64

Top 10 point producers:
name
Nikita Kucherov    101.0
Brandon Hagel       79.0
Brayden Point       69.0
Jake Guentzel       68.0
Victor Hedman       54.0
Anthony Cirelli     52.0
Nick Paul           39.0
Darren Raddysh      33.0
Ryan McDonagh       25.0
Erik Cernak         20.0
Name: I_F_points, dtype: float64

Top 10 players by average Game Score (min. 10 games):
                    games_played  avg_gamescore
name
Nikita Kucherov               67       1.466045
Brandon Hagel                 71       1.186690
Jake Guentzel                 70       1.009643
Brayden Point                 66       0.972348
Anthony Cirelli               69       0.931957
Victor Hedman                 68       0.870221
Nick Paul                     65       0.575308
Darren Raddysh                62       0.571371
Yanni Gourde                  10       0.512000
Oliver Bjorkstrand            10       0.476500
```

Aggregating Player Data to Game Level

To build a model that predicts team success based on player performance, I need to aggregate individual player statistics to the game level. I'll focus on the top performers we just identified.

In [7]:
```
# Let's focus on the top 5 players by average Game Score
top_5_players = player_performance.head(5).index.tolist()
print("Top 5 players by Game Score:")
print(top_5_players)

# Now I'll aggregate key player metrics to the game level
# For each game, create features for these top 5 players' performance
game_player_stats = []

# Get list of unique game IDs from team data
game_ids = team_all['gameId'].unique()
```

```python
for game_id in game_ids:
    # Filter player data for this game
    game_player_data = player_all[player_all['gameId'] == game_id]

    # Start with game ID
    game_stats = {'gameId': game_id}

    # Add individual stats for top 5 players
    for player in top_5_players:
        player_game = game_player_data[game_player_data['name'] == pl

        if len(player_game) > 0:
            game_stats[f'{player}_played'] = 1
            game_stats[f'{player}_gamescore'] = player_game['gameScor
            game_stats[f'{player}_icetime'] = player_game['icetime']
        else:
            game_stats[f'{player}_played'] = 0
            game_stats[f'{player}_gamescore'] = 0
            game_stats[f'{player}_icetime'] = 0

    game_player_stats.append(game_stats)

# Convert to DataFrame
game_player_df = pd.DataFrame(game_player_stats)

# Display the first few rows to check
print("\nFirst 5 rows of game-player stats:")
print(game_player_df.head())
```

```
Top 5 players by Game Score:
['Nikita Kucherov', 'Brandon Hagel', 'Jake Guentzel', 'Brayden Poin
t', 'Anthony Cirelli']

First 5 rows of game-player stats:
       gameId  Nikita Kucherov_played  Nikita Kucherov_gamescore  \
0  2024020020                       1                       3.15
1  2024020048                       1                       1.45
2  2024020064                       1                       2.95
3  2024020075                       1                       2.00
4  2024020091                       1                       0.60

   Nikita Kucherov_icetime  Brandon Hagel_played  Brandon Hagel_games
core  \
0                   1315.0                     1
0.325
1                   1348.0                     1
1.900
2                   1386.0                     1
1.375
3                   1184.0                     1
1.300
```

```
4                   1299.0                    1
0.285

   Brandon Hagel_icetime  Jake Guentzel_played  Jake Guentzel_gamesco
re  \
0                 1228.0                     1                    1.4
60
1                 1010.0                     1                    1.4
35
2                  975.0                     1                    1.9
45
3                 1143.0                     1                    1.0
30
4                 1299.0                     1                   -0.2
65

   Jake Guentzel_icetime  Brayden Point_played  Brayden Point_gamesco
re  \
0                 1076.0                     1                    0.1
25
1                 1141.0                     1                    2.1
70
2                 1243.0                     1                    1.1
85
3                 1202.0                     1                    0.7
25
4                 1219.0                     1                    0.0
55

   Brayden Point_icetime  Anthony Cirelli_played  Anthony Cirelli_gam
escore  \
0                 1219.0                       1
0.840
1                 1153.0                       1
1.835
2                 1260.0                       1
0.275
3                 1256.0                       1
1.865
4                 1315.0                       1
0.480

   Anthony Cirelli_icetime
0                   1150.0
1                    980.0
2                    912.0
3                   1016.0
4                   1010.0
```

Creating the Combined Game-Level Dataset

Now I'll merge the player performance metrics with the team-level data to create a comprehensive dataset for modeling. This will allow us to analyze how individual player performances impact team success.

In [8]:
```
# Merge the player data with team data
game_features = pd.merge(team_all, game_player_df, on='gameId', how=

# Check the shape of our merged dataset
print(f"Combined dataset shape: {game_features.shape}")

# Display a sample of our combined dataset
print("\nSample of combined dataset with both team and player metric
columns_to_show = ['gameId', 'opposingTeam', 'xGoalsPercentage', 'wo
                   'Nikita Kucherov_gamescore', 'Brandon Hagel_games
                   'Jake Guentzel_gamescore']
print(game_features[columns_to_show].head())

# Check for any missing values
missing_values = game_features[columns_to_show].isnull().sum()
print("\nMissing values in key columns:")
print(missing_values)
```

```
Combined dataset shape: (71, 126)

Sample of combined dataset with both team and player metrics:
       gameId opposingTeam  xGoalsPercentage  won_game  \
0  2024020020          CAR            0.5194         1
1  2024020048          VAN            0.4809         1
2  2024020064          VGK            0.6483         1
3  2024020075          OTT            0.5841         0
4  2024020091          TOR            0.4760         0

   Nikita Kucherov_gamescore  Brandon Hagel_gamescore  Jake Guentzel_
gamescore
0                       3.15                    0.325
1.460
1                       1.45                    1.900
1.435
2                       2.95                    1.375
1.945
3                       2.00                    1.300
1.030
4                       0.60                    0.285
-0.265

Missing values in key columns:
gameId                       0
opposingTeam                 0
xGoalsPercentage             0
won_game                     0
```

```
Nikita Kucherov_gamescore    0
Brandon Hagel_gamescore      0
Jake Guentzel_gamescore      0
dtype: int64
```

Preparing Data for Modeling

Now I'll prepare the data for modeling by selecting the most relevant features and splitting the data into training and test sets. Since hockey games occur sequentially in a season, I'll use a time-based split rather than a random split.

In [9]:
```
# First, sort games by date to ensure temporal ordering
game_features = game_features.sort_values('gameDate')

# Select features for modeling
# Avoid data leakage by excluding goals and actual outcomes
model_features = [
    # Team performance metrics
    'xGoalsPercentage', 'corsiPercentage', 'fenwickPercentage',
    'shotsOnGoalFor', 'shotsOnGoalAgainst',
    'highDangerShotsFor', 'highDangerShotsAgainst',

    # Player metrics for top players
    'Nikita Kucherov_gamescore', 'Nikita Kucherov_icetime',
    'Brandon Hagel_gamescore', 'Brandon Hagel_icetime',
    'Jake Guentzel_gamescore', 'Jake Guentzel_icetime',
    'Brayden Point_gamescore', 'Brayden Point_icetime',
    'Anthony Cirelli_gamescore', 'Anthony Cirelli_icetime'
]

# Define target variable - we'll predict whether the Lightning win tl
target = 'won_game'

# Create training and test sets - using the first 80% of games for tr
train_size = int(0.8 * len(game_features))
X_train = game_features.iloc[:train_size][model_features]
y_train = game_features.iloc[:train_size][target]
X_test = game_features.iloc[train_size:][model_features]
y_test = game_features.iloc[train_size:][target]

print(f"Training set: {X_train.shape[0]} games")
print(f"Test set: {X_test.shape[0]} games")

# Show first few rows of training features
print("\nTraining features sample:")
print(X_train.head(3))
```

```
Training set: 56 games
Test set: 15 games

Training features sample:
   xGoalsPercentage  corsiPercentage  fenwickPercentage  shotsOnGoalF
or  \
0            0.5194           0.4190             0.4306            2
3.0
1            0.4809           0.4793             0.5185            2
7.0
2            0.6483           0.6174             0.5750            2
5.0

   shotsOnGoalAgainst  highDangerShotsFor  highDangerShotsAgainst  \
0                21.0                 6.0                     4.0
1                27.0                 5.0                     7.0
2                25.0                 2.0                     1.0

   Nikita Kucherov_gamescore  Nikita Kucherov_icetime  \
0                       3.15                   1315.0
1                       1.45                   1348.0
2                       2.95                   1386.0

   Brandon Hagel_gamescore  Brandon Hagel_icetime  Jake Guentzel_game
score  \
0                    0.325                 1228.0
1.460
1                    1.900                 1010.0
1.435
2                    1.375                  975.0
1.945

   Jake Guentzel_icetime  Brayden Point_gamescore  Brayden Point_icet
ime  \
0                 1076.0                    0.125                 121
9.0
1                 1141.0                    2.170                 115
3.0
2                 1243.0                    1.185                 126
0.0

   Anthony Cirelli_gamescore  Anthony Cirelli_icetime
0                      0.840                   1150.0
1                      1.835                    980.0
2                      0.275                    912.0
```

Building an Interpretable Model

Now I'll build a model to predict Lightning wins based on the selected features. I'll use a Random Forest classifier, which will allow us to interpret feature importance to understand which player and team metrics most contribute to winning.

In [10]:
```
# Train a Random Forest model
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, co

# Initialize and train the model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions on test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy on test set: {accuracy:.2f}")

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Get feature importances
importances = rf_model.feature_importances_
feature_importance = pd.DataFrame(
    {'Feature': model_features, 'Importance': importances}
)
feature_importance = feature_importance.sort_values('Importance', asc

print("\nTop 10 most important features:")
print(feature_importance.head(10))

# Visualize feature importances
plt.figure(figsize=(12, 6))
plt.barh(feature_importance['Feature'][:10], feature_importance['Impo
plt.xlabel('Importance')
plt.title('Top 10 Most Important Features for Predicting Lightning Wi
plt.gca().invert_yaxis()  # Display most important at the top
plt.tight_layout()
plt.savefig('feature_importance.png')
plt.close()
```

```
Model accuracy on test set: 0.87

Classification Report:
```

```
              precision    recall  f1-score   support

           0       1.00      0.71      0.83         7
           1       0.80      1.00      0.89         8

    accuracy                           0.87        15
   macro avg       0.90      0.86      0.86        15
weighted avg       0.89      0.87      0.86        15


Confusion Matrix:
[[5 2]
 [0 8]]

Top 10 most important features:
                     Feature  Importance
8   Nikita Kucherov_icetime    0.113744
9   Brandon Hagel_gamescore    0.100148
11  Jake Guentzel_gamescore    0.091320
13  Brayden Point_gamescore    0.090870
0          xGoalsPercentage    0.090137
1           corsiPercentage    0.072437
16  Anthony Cirelli_icetime    0.066384
12    Jake Guentzel_icetime    0.061883
14    Brayden Point_icetime    0.047148
10    Brandon Hagel_icetime    0.045933
```

In [11]:

```python
# Import visualization libraries
import matplotlib.pyplot as plt
import seaborn as sns

# We already have the Random Forest feature importance
# Let's create a better visualization of it
plt.figure(figsize=(10, 8))
plt.barh(feature_importance['Feature'][:10], feature_importance['Impo
plt.xlabel('Importance')
plt.title('Top 10 Most Important Features for Predicting Lightning Wi
plt.gca().invert_yaxis()  # Display most important at the top
plt.tight_layout()
plt.savefig('feature_importance.png')
plt.close()

# Now let's calculate permutation importance for more robust interpre
from sklearn.inspection import permutation_importance

# Calculate permutation importance on test set
result = permutation_importance(rf_model, X_test, y_test, n_repeats=1
perm_importance = pd.DataFrame(
    {'Feature': model_features, 'Importance': result.importances_mean
)
perm_importance = perm_importance.sort_values('Importance', ascending
```

```
print("Top 10 features by permutation importance:")
print(perm_importance.head(10))

# Visualize permutation importance
plt.figure(figsize=(10, 8))
plt.barh(perm_importance['Feature'][:10], perm_importance['Importanc
plt.xlabel('Permutation Importance')
plt.title('Top 10 Features by Permutation Importance')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.savefig('permutation_importance.png')
plt.close()

# Let's examine a specific winning and losing game from the test set
test_games = game_features.iloc[X_test.index]
test_win = test_games[test_games['won_game'] == 1].iloc[0]
test_loss = test_games[test_games['won_game'] == 0].iloc[0]

print(f"\nAnalysis of a winning game:")
print(f"Game ID: {test_win['gameId']}")
print(f"Opponent: {test_win['opposingTeam']}")
print(f"Score: {test_win['goalsFor']} - {test_win['goalsAgainst']}")

# Look at key metrics for this win
print("\nKey metrics for this win:")
for feature in feature_importance['Feature'][:5]:
    print(f"{feature}: {test_win[feature]}")

print(f"\nAnalysis of a losing game:")
print(f"Game ID: {test_loss['gameId']}")
print(f"Opponent: {test_loss['opposingTeam']}")
print(f"Score: {test_loss['goalsFor']} - {test_loss['goalsAgainst']}

# Look at key metrics for this loss
print("\nKey metrics for this loss:")
for feature in feature_importance['Feature'][:5]:
    print(f"{feature}: {test_loss[feature]}")

# Let's analyze one more thing: how often does each top player's per
# correlate with team success?
player_metrics = [col for col in game_features.columns if 'gamescore
player_correlations = []

for metric in player_metrics:
    corr = game_features[metric].corr(game_features['won_game'])
    player_correlations.append((metric, corr))

# Sort by absolute correlation value
player_correlations.sort(key=lambda x: abs(x[1]), reverse=True)
```

```
print("\nCorrelation between player Game Scores and team wins:")
for player, corr in player_correlations:
    print(f"{player}: {corr:.4f}")
```

```
Top 10 features by permutation importance:
                       Feature  Importance
7   Nikita Kucherov_gamescore    0.113333
13    Brayden Point_gamescore    0.106667
0            xGoalsPercentage    0.106667
11    Jake Guentzel_gamescore    0.100000
1             corsiPercentage    0.060000
3              shotsOnGoalFor    0.053333
9     Brandon Hagel_gamescore    0.053333
16    Anthony Cirelli_icetime    0.053333
2           fenwickPercentage    0.053333
15  Anthony Cirelli_gamescore    0.040000

Analysis of a winning game:
Game ID: 2024020921
Opponent: EDM
Score: 4.0 - 1.0

Key metrics for this win:
Nikita Kucherov_icetime: 1371.0
Brandon Hagel_gamescore: 1.575
Jake Guentzel_gamescore: 1.68
Brayden Point_gamescore: 1.56
xGoalsPercentage: 0.6709999999999999

Analysis of a losing game:
Game ID: 2024020969
Opponent: FLA
Score: 1.0 - 2.0

Key metrics for this loss:
Nikita Kucherov_icetime: 1678.0
Brandon Hagel_gamescore: 0.475
Jake Guentzel_gamescore: 0.44
Brayden Point_gamescore: 0.8
xGoalsPercentage: 0.4494

Correlation between player Game Scores and team wins:
Jake Guentzel_gamescore: 0.4026
Nikita Kucherov_gamescore: 0.4017
Brandon Hagel_gamescore: 0.3852
Brayden Point_gamescore: 0.3459
Anthony Cirelli_gamescore: 0.2516
```

Player-Specific Impact Model: Nikita Kucherov

In this section, I'll build a dedicated model to analyze how Nikita Kucherov's individual performance metrics influence Tampa Bay Lightning wins. This will provide deeper insights into the specific aspects of his game that most contribute to team success.

In [12]:
```
# Focus on Nikita Kucherov's performance metrics
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, c
from sklearn.model_selection import train_test_split
from sklearn.inspection import permutation_importance

# Filter player data for just Kucherov's games
kucherov_data = player_all[player_all['name'] == 'Nikita Kucherov']

# Print basic information about Kucherov's games
print(f"Number of games with Kucherov data: {len(kucherov_data)}")
print(f"Average Game Score: {kucherov_data['gameScore'].mean():.2f}"
print(f"Average ice time: {kucherov_data['icetime'].mean() / 60:.2f}

# Create a DataFrame with game-by-game Kucherov stats
kucherov_games = kucherov_data.groupby('gameId').agg({
    'gameScore': 'first',
    'icetime': 'first',
    'I_F_goals': 'first',
    'I_F_points': 'first',
    'I_F_shotAttempts': 'first',
    'I_F_xGoals': 'first',
    'I_F_highDangerShots': 'first',
    'onIce_xGoalsPercentage': 'first',
    'onIce_corsiPercentage': 'first'
}).reset_index()

# Rename columns for clarity
kucherov_games = kucherov_games.rename(columns={
    'gameScore': 'kucherov_gameScore',
    'icetime': 'kucherov_icetime',
    'I_F_goals': 'kucherov_goals',
    'I_F_points': 'kucherov_points',
    'I_F_shotAttempts': 'kucherov_shotAttempts',
    'I_F_xGoals': 'kucherov_xGoals',
    'I_F_highDangerShots': 'kucherov_highDangerShots',
    'onIce_xGoalsPercentage': 'kucherov_onIce_xGoalsPercentage',
    'onIce_corsiPercentage': 'kucherov_onIce_corsiPercentage'
```

```
})

# Merge with team data to get game outcomes
kucherov_impact = pd.merge(
    kucherov_games,
    team_all[['gameId', 'opposingTeam', 'home_or_away', 'goal_differe
    on='gameId',
    how='inner'
)

# Check the shape of our combined dataset
print(f"\nCombined Kucherov impact dataset shape: {kucherov_impact.sl

# Display a sample
print("\nSample of Kucherov's impact dataset:")
print(kucherov_impact[['gameId', 'opposingTeam', 'kucherov_gameScore

# Visualize relationship between Kucherov's Game Score and team wins
plt.figure(figsize=(10, 6))
sns.boxplot(x='won_game', y='kucherov_gameScore', data=kucherov_impa
plt.title("Kucherov's Game Score by Game Outcome")
plt.xlabel('Game Outcome (0=Loss, 1=Win)')
plt.ylabel('Game Score')
plt.savefig('kucherov_gamescore_by_outcome.png')
plt.close()

# Analyze Kucherov's performance by game situation
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x='kucherov_icetime',
    y='kucherov_gameScore',
    hue='won_game',
    size='kucherov_points',
    sizes=(50, 250),
    data=kucherov_impact
)
plt.title("Kucherov's Performance by Ice Time and Game Outcome")
plt.xlabel('Ice Time (seconds)')
plt.ylabel('Game Score')
plt.legend(title='Game Outcome (1=Win)')
plt.savefig('kucherov_performance_by_icetime.png')
plt.close()

# Let's build a model to predict team success based solely on Kucher
kucherov_features = [
    'kucherov_gameScore',
    'kucherov_icetime',
    'kucherov_goals',
    'kucherov_points',
    'kucherov_shotAttempts',
```

```
    'kucherov_xGoals',
    'kucherov_highDangerShots',
    'kucherov_onIce_xGoalsPercentage',
    'kucherov_onIce_corsiPercentage'
]

# Split the data into training and testing sets (time-based split)
kucherov_impact = kucherov_impact.sort_values('gameId')
train_size = int(0.8 * len(kucherov_impact))

X_train = kucherov_impact.iloc[:train_size][kucherov_features]
y_train = kucherov_impact.iloc[:train_size]['won_game']
X_test = kucherov_impact.iloc[train_size:][kucherov_features]
y_test = kucherov_impact.iloc[train_size:]['won_game']

# Train a Random Forest model
rf_kucherov = RandomForestClassifier(n_estimators=100, random_state=
rf_kucherov.fit(X_train, y_train)

# Make predictions on test set
y_pred = rf_kucherov.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"\nKucherov-specific model accuracy on test set: {accuracy:.2

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Get feature importances
importances = rf_kucherov.feature_importances_
feature_importance = pd.DataFrame(
    {'Feature': kucherov_features, 'Importance': importances}
)
feature_importance = feature_importance.sort_values('Importance', as

print("\nMost important aspects of Kucherov's game for predicting te
print(feature_importance)

# Visualize feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['Feature'], feature_importance['Importan
plt.xlabel('Importance')
plt.title('Importance of Kucherov Performance Metrics for Predicting
plt.gca().invert_yaxis()  # Display most important at the top
plt.tight_layout()
plt.savefig('kucherov_metrics_importance.png')
plt.close()
```

```
# Calculate threshold for "good" Kucherov performance
win_threshold = kucherov_impact[kucherov_impact['won_game'] == 1]['ku
print(f"\nWhen Kucherov's Game Score is above {win_threshold:.2f}, th
high_performance = kucherov_impact[kucherov_impact['kucherov_gameScor
print(f"{high_performance['won_game'].mean()*100:.1f}%")

print(f"\nWhen Kucherov's Game Score is below {win_threshold:.2f}, th
low_performance = kucherov_impact[kucherov_impact['kucherov_gameScore
print(f"{low_performance['won_game'].mean()*100:.1f}%")
```

```
Number of games with Kucherov data: 67
Average Game Score: 1.47
Average ice time: 21.40 minutes

Combined Kucherov impact dataset shape: (67, 14)

Sample of Kucherov's impact dataset:
       gameId opposingTeam  kucherov_gameScore  kucherov_points  won_
game
0  2024020020          CAR               3.150              4.0
1
1  2024020029          CAR               1.025              1.0
1
2  2024020048          VAN               1.450              1.0
1
3  2024020064          VGK               2.950              2.0
1
4  2024020075          OTT               2.000              2.0
0

Kucherov-specific model accuracy on test set: 0.93

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.83      0.91         6
           1       0.89      1.00      0.94         8

    accuracy                           0.93        14
   macro avg       0.94      0.92      0.93        14
weighted avg       0.94      0.93      0.93        14


Most important aspects of Kucherov's game for predicting team wins:
                         Feature  Importance
1                kucherov_icetime    0.265156
7  kucherov_onIce_xGoalsPercentage    0.246121
0              kucherov_gameScore    0.149871
5                 kucherov_xGoals    0.091154
4           kucherov_shotAttempts    0.080403
8   kucherov_onIce_corsiPercentage    0.079021
3                 kucherov_points    0.043451
2                  kucherov_goals    0.023289
```

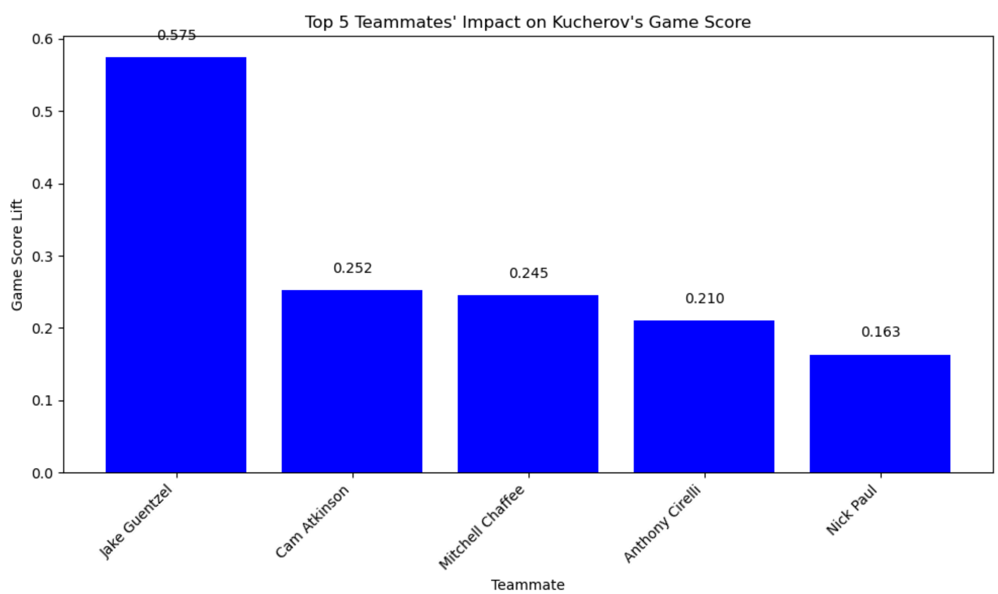
```
6         kucherov_highDangerShots    0.021534

When Kucherov's Game Score is above 1.32, the Lightning win percentag
e is:
76.3%

When Kucherov's Game Score is below 1.32, the Lightning win percentag
e is:
27.6%
```

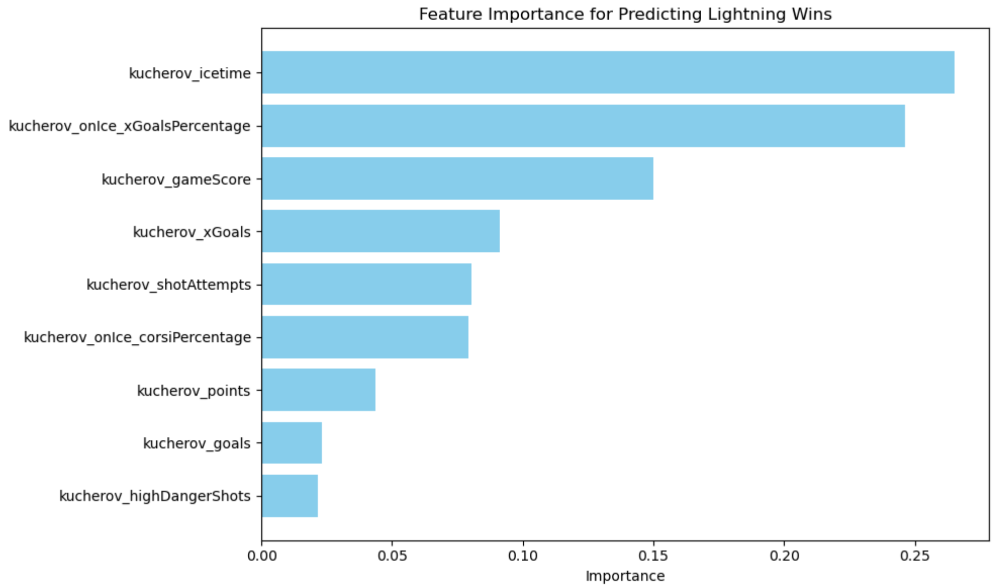
Analyzing Player Chemistry with Nikita Kucherov

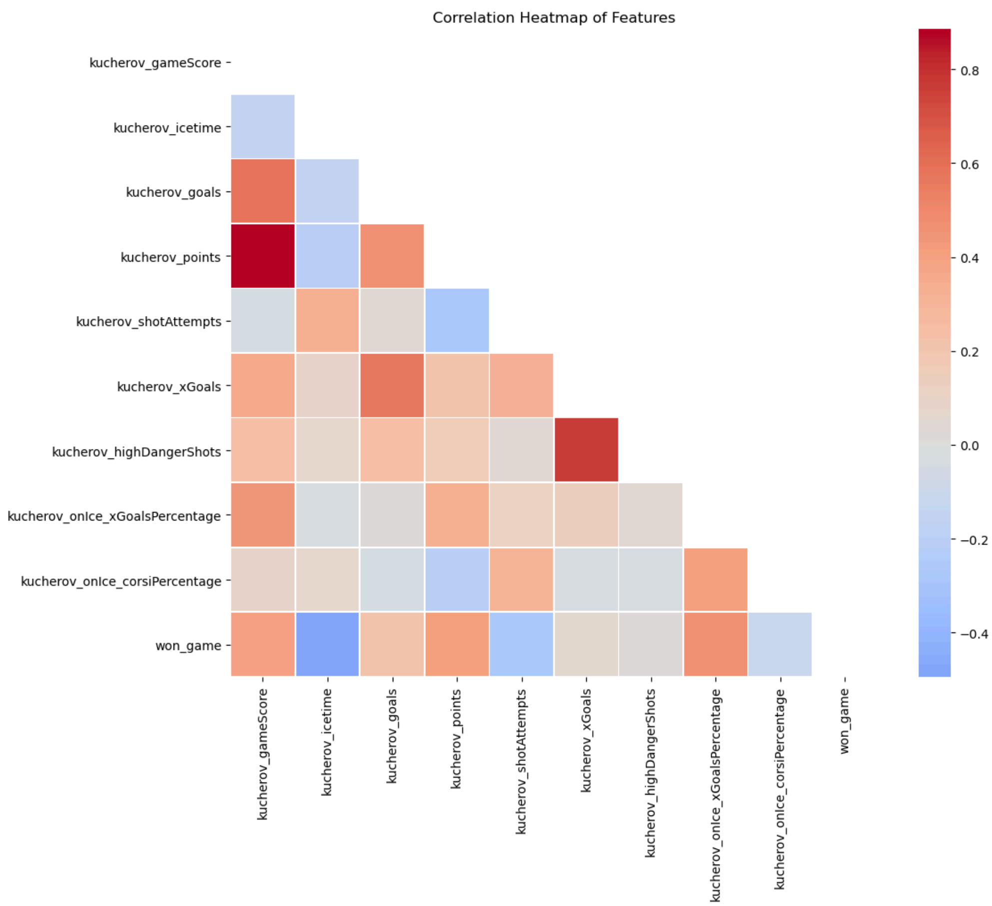
Now I'll identify which teammates most enhance Kucherov's performance and, by extension, the Lightning's winning chances. This chemistry analysis can provide valuable lineup optimization insights.

In [15]:
```
# Create clearer visualizations of teammate impact on Kucherov
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Ensure the plot style is set to something reliable
plt.style.use('default')

# Select top 5 teammates by game score lift
top_teammates = teammate_impact_df.head(5)

# Simple bar chart for game score lift
plt.figure(figsize=(10, 6))
bars = plt.bar(top_teammates['teammate'], top_teammates['gamescore_l
plt.title("Top 5 Teammates' Impact on Kucherov's Game Score")
plt.xlabel('Teammate')
plt.ylabel('Game Score Lift')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

# Add value labels on top of bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height + 0.02,
            f'{height:.3f}', ha='center', va='bottom')

plt.savefig('kucherov_top_teammates.png', dpi=300, bbox_inches='tigh
plt.show()  # This should display in notebook
plt.close()
```

```
# Try generating the feature importance plot again with improved forr
plt.figure(figsize=(10, 6))
importance_df = feature_importance.copy()
importance_df = importance_df.sort_values('Importance', ascending=Tru
plt.barh(importance_df['Feature'], importance_df['Importance'], color
plt.xlabel('Importance')
plt.title('Feature Importance for Predicting Lightning Wins')
plt.tight_layout()
plt.savefig('feature_importance.png', dpi=300, bbox_inches='tight')
plt.show()
plt.close()

# Add a correlation heatmap as an alternative visualization
plt.figure(figsize=(12, 10))
model_data = X_train.copy()
model_data['won_game'] = y_train
corr = model_data.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, mask=mask, cmap='coolwarm', annot=False,
            center=0, square=True, linewidths=.5)
plt.title('Correlation Heatmap of Features')
plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=300, bbox_inches='tight')
plt.show()
plt.close()
```




Feature Importance for Predicting Lightning Wins
kucherov_icetime
kucherov_onIce_xGoalsPercentage
kucherov_gameScore
kucherov_xGoals
kucherov_shotAttempts
kucherov_onIce_corsiPercentage
kucherov_points
kucherov_goals
kucherov_highDangerShots
0.00
0.05
0.10
0.15
0.20
0.25
Importance


Correlation Heatmap of Features
kucherov_gameScore
kucherov_icetime
kucherov_goals
kucherov_points
kucherov_shotAttempts
kucherov_xGoals
kucherov_highDangerShots
kucherov_onIce_xGoalsPercentage
kucherov_onIce_corsiPercentage
won_game
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Identifying Undervalued and Overvalued Players

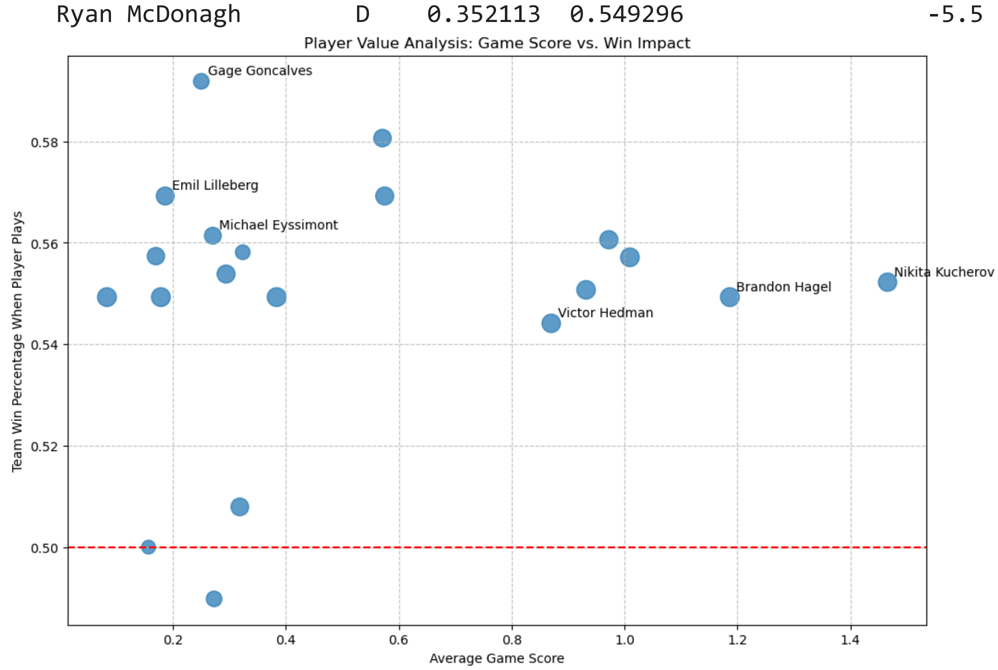
Now I'll analyze which players might be undervalued or overvalued based on traditional metrics versus their actual impact on team success.

In [16]:
```
# Let's analyze player value by comparing traditional stats vs. impac
player_value = []

# For each player who played in at least 20 games
for player_name in player_all['name'].unique():
    player_games = player_all[player_all['name'] == player_name]['gar

    if player_games < 20:
        continue

    # Get player's traditional stats
    player_stats = player_all[player_all['name'] == player_name]
    avg_points = player_stats['I_F_points'].mean()
    avg_goals = player_stats['I_F_goals'].mean()
    avg_gamescore = player_stats['gameScore'].mean()
    position = player_stats['position'].iloc[0]

    # Get player's impact on team winning
    player_game_ids = player_stats['gameId'].unique()
    team_win_pct = team_all[team_all['gameId'].isin(player_game_ids)

    # Calculate on-ice impact metrics
    avg_xg_pct = player_stats['onIce_xGoalsPercentage'].mean()
    avg_corsi_pct = player_stats['onIce_corsiPercentage'].mean()

    # Add to our analysis
    player_value.append({
        'player': player_name,
        'position': position,
        'games_played': player_games,
        'avg_points': avg_points,
        'avg_goals': avg_goals,
        'avg_gamescore': avg_gamescore,
        'win_pct': team_win_pct,
        'on_ice_xg_pct': avg_xg_pct,
        'on_ice_corsi_pct': avg_corsi_pct
    })

# Convert to DataFrame
player_value_df = pd.DataFrame(player_value)
```

```
# Calculate value metrics
player_value_df['points_rank'] = player_value_df['avg_points'].rank(a
player_value_df['gamescore_rank'] = player_value_df['avg_gamescore']
player_value_df['win_impact_rank'] = player_value_df['win_pct'].rank
player_value_df['xg_impact_rank'] = player_value_df['on_ice_xg_pct']

# Calculate value differences
player_value_df['points_vs_win_value'] = player_value_df['points_rank
player_value_df['gamescore_vs_win_value'] = player_value_df['gamesco

# Identify potentially undervalued players (better win impact than th
undervalued = player_value_df.sort_values('points_vs_win_value', asc
print("Potentially undervalued players (better win impact than points
print(undervalued[['player', 'position', 'avg_points', 'win_pct', 'po

# Identify potentially overvalued players
overvalued = player_value_df.sort_values('points_vs_win_value', ascen
print("\nPotentially overvalued players (worse win impact than points
print(overvalued[['player', 'position', 'avg_points', 'win_pct', 'poi

# Visualize the relationship between Game Score and Win Impact
plt.figure(figsize=(12, 8))
plt.scatter(player_value_df['avg_gamescore'], player_value_df['win_pc
           alpha=0.7, s=player_value_df['games_played']*3)

# Label undervalued and overvalued players
for _, player in pd.concat([undervalued.head(3), overvalued.head(3)]
    plt.annotate(player['player'],
                 (player['avg_gamescore'], player['win_pct']),
                 xytext=(5, 5),
                 textcoords='offset points')

plt.xlabel('Average Game Score')
plt.ylabel('Team Win Percentage When Player Plays')
plt.title('Player Value Analysis: Game Score vs. Win Impact')
plt.axhline(y=0.5, color='red', linestyle='--', label='50% Win Rate'
plt.grid(True, linestyle='--', alpha=0.7)
plt.savefig('player_value_analysis.png', dpi=300, bbox_inches='tight
plt.show()
plt.close()
```

```
Potentially undervalued players (better win impact than points sugges
t):
            player position  avg_points   win_pct  points_vs_win_valu
e
 Michael Eyssimont        C    0.175439  0.561404                 13.
0
    Gage Goncalves        C    0.265306  0.591837                 12.
0
    Emil Lilleberg        D    0.261538  0.569231                 10.
```

```
5
    Darren Raddysh        D    0.532258  0.580645                  6.
0
 Zemgus Girgensons        C    0.084507  0.549296                  5.
5

Potentially overvalued players (worse win impact than points sugges
t):
          player position  avg_points   win_pct  points_vs_win_value
   Brandon Hagel        L    1.112676  0.549296                -12.5
   Victor Hedman        D    0.794118  0.544118                -12.0
 Nikita Kucherov        R    1.507463  0.552239                -10.0
 Anthony Cirelli        C    0.753623  0.550725                 -6.0
   Ryan McDonagh        D    0.352113  0.549296                 -5.5
```



Analyzing Defensive Shutdown Effectiveness

Next, I'll examine which Lightning players are most effective at shutting down opposing teams' top players, focusing on defensive metrics and on-ice performance.

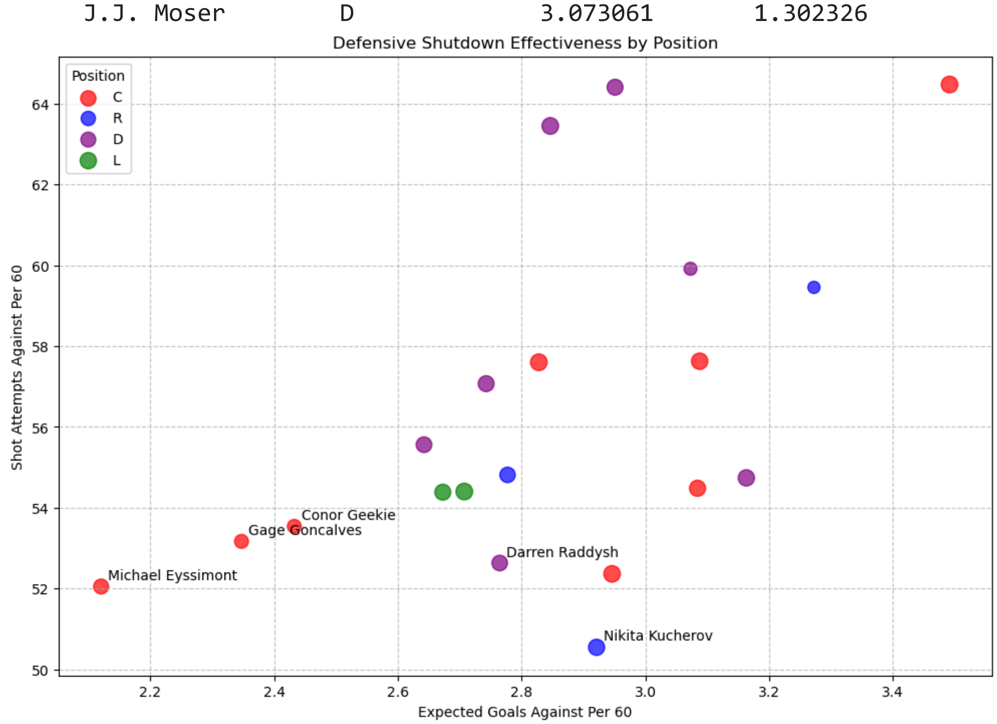
In [17]:
```python
# Let's identify which players are most effective defensively
defense_impact = []

# For each player who played in at least 20 games
for player_name in player_all['name'].unique():
```

```
    player_games = player_all[player_all['name'] == player_name]['ga

    if player_games < 20:
        continue

    # Get player's defensive stats
    player_stats = player_all[player_all['name'] == player_name]
    position = player_stats['position'].iloc[0]

    # Defensive metrics
    shots_blocked = player_stats['shotsBlockedByPlayer'].mean()
    takeaways = player_stats['I_F_takeaways'].mean()

    # On-ice defensive impact
    against_xg = player_stats['OnIce_A_xGoals'].mean()
    against_shots = player_stats['OnIce_A_shotAttempts'].mean()
    against_goals = player_stats['OnIce_A_goals'].mean()

    # Calculate defensive efficiency metrics
    defensive_xg_per_60 = against_xg / (player_stats['icetime'].mean
    defensive_shots_per_60 = against_shots / (player_stats['icetime'

    # Calculate shutdown score (lower is better defensively)
    shutdown_score = defensive_xg_per_60 * 0.7 + defensive_shots_per_

    # Add to our analysis
    defense_impact.append({
        'player': player_name,
        'position': position,
        'games_played': player_games,
        'shots_blocked': shots_blocked,
        'takeaways': takeaways,
        'defensive_xg_per_60': defensive_xg_per_60,
        'defensive_shots_per_60': defensive_shots_per_60,
        'shutdown_score': shutdown_score
    })

# Convert to DataFrame
defense_impact_df = pd.DataFrame(defense_impact)

# Rank by shutdown effectiveness (lower score is better)
defense_impact_df = defense_impact_df.sort_values('shutdown_score')

# Show top defensive players overall
print("Top defensive players overall (lowest xG against per 60):")
print(defense_impact_df[['player', 'position', 'defensive_xg_per_60'

# Show top defensive forwards
defense_forwards = defense_impact_df[defense_impact_df['position'].i
print("\nTop defensive forwards:")
```

```
print(defense_forwards[['player', 'position', 'defensive_xg_per_60', 

# Show top defensive defensemen
defense_d = defense_impact_df[defense_impact_df['position'] == 'D'].
print("\nTop defensive defensemen:")
print(defense_d[['player', 'position', 'defensive_xg_per_60', 'shots_

# Visualize defensive effectiveness
plt.figure(figsize=(12, 8))
colors = {'C': 'red', 'L': 'green', 'R': 'blue', 'D': 'purple'}
positions = defense_impact_df['position'].unique()

for position in positions:
    pos_data = defense_impact_df[defense_impact_df['position'] == po
    plt.scatter(pos_data['defensive_xg_per_60'], pos_data['defensive_
                label=position, color=colors[position], alpha=0.7,
                s=pos_data['games_played']*2)

# Label top defensive players
for _, player in defense_impact_df.head(5).iterrows():
    plt.annotate(player['player'],
                 (player['defensive_xg_per_60'], player['defensive_sh
                 xytext=(5, 5),
                 textcoords='offset points')

plt.xlabel('Expected Goals Against Per 60')
plt.ylabel('Shot Attempts Against Per 60')
plt.title('Defensive Shutdown Effectiveness by Position')
plt.legend(title='Position')
plt.grid(True, linestyle='--', alpha=0.7)
plt.savefig('defensive_effectiveness.png', dpi=300, bbox_inches='tigl
plt.show()
plt.close()
```

```
Top defensive players overall (lowest xG against per 60):
            player position  defensive_xg_per_60  shots_blocked  take
aways
 Michael Eyssimont        C             2.120233       0.333333   0.1
05263
   Nikita Kucherov        R             2.921247       0.477612   0.3
28358
    Gage Goncalves        C             2.347163       0.326531   0.3
06122
    Darren Raddysh        D             2.764631       0.967742   0.1
12903
      Conor Geekie        C             2.432533       0.448980   0.1
42857
     Jake Guentzel        C             2.946318       0.671429   0.3
42857
         Nick Paul        L             2.672623       0.338462   0.2
30769
```

```
     Brandon Hagel        L             2.707374       0.718310   0.3
66197
  Mitchell Chaffee        R             2.777321       0.295082   0.1
47541
     Brayden Point        C             3.084452       0.560606   0.1
96970

Top defensive forwards:
            player position  defensive_xg_per_60  takeaways
 Michael Eyssimont        C             2.120233   0.105263
   Nikita Kucherov        R             2.921247   0.328358
     Gage Goncalves       C             2.347163   0.306122
       Conor Geekie       C             2.432533   0.142857
      Jake Guentzel       C             2.946318   0.342857

Top defensive defensemen:
         player position  defensive_xg_per_60  shots_blocked
 Darren Raddysh        D             2.764631       0.967742
    Nick Perbix        D             2.642213       0.920635
  Victor Hedman        D             3.163377       1.676471
 Emil Lilleberg        D             2.742728       0.815385
     J.J. Moser        D             3.073061       1.302326
```



Identifying Patterns in Lightning Losses and Potential Adjustments

To understand how the Lightning could improve their performance, I'll analyze patterns in their losses to identify adjustable factors that could lead to better outcomes.

In [18]:
```
# Identify patterns in losses vs wins
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Separate games into wins and losses
wins = team_all[team_all['won_game'] == 1]
losses = team_all[team_all['won_game'] == 0]

print(f"Analyzing {len(wins)} wins and {len(losses)} losses")

# Compare key team metrics between wins and losses
team_metrics = [
    'xGoalsPercentage', 'corsiPercentage', 'fenwickPercentage',
    'shotsOnGoalFor', 'shotsOnGoalAgainst', 'highDangerShotsFor', 'hi
    'faceOffsWonFor', 'takeawaysFor', 'giveawaysFor'
]

# Calculate means for wins and losses
win_means = wins[team_metrics].mean()
loss_means = losses[team_metrics].mean()
pct_diff = (win_means - loss_means) / loss_means * 100

# Create a comparison dataframe
comparison = pd.DataFrame({
    'Wins': win_means,
    'Losses': loss_means,
    'Pct_Difference': pct_diff
})

print("\nKey differences between wins and losses:")
print(comparison.sort_values('Pct_Difference', ascending=False))

# Visualize key differences
plt.figure(figsize=(12, 8))
metrics_to_plot = comparison.sort_values('Pct_Difference', ascending=

for i, metric in enumerate(metrics_to_plot):
    plt.subplot(2, 3, i+1)
    sns.boxplot(x='won_game', y=metric, data=team_all)
    plt.title(f'{metric} by Game Outcome')
    plt.xlabel('Win (1) vs Loss (0)')
```

```
plt.tight_layout()
plt.savefig('win_loss_differences.png', dpi=300, bbox_inches='tight'
plt.close()

# Identify specific players whose presence/absence makes the biggest
# We'll use our player value analysis from before
player_win_impact = player_value_df.sort_values('win_pct', ascending

print("\nPlayers with biggest positive impact on win percentage:")
print(player_win_impact[['player', 'position', 'games_played', 'win_

print("\nPlayers with biggest negative impact (lowest win percentage
print(player_win_impact[['player', 'position', 'games_played', 'win_

# Examine specific game situations that lead to losses
# Analyze close losses (1-goal games)
close_losses = losses[losses['goal_differential'] == -1]
print(f"\nNumber of close losses (1-goal games): {len(close_losses)}

# Analyze key metrics in close losses
close_loss_metrics = close_losses[team_metrics].mean()
print("\nKey metrics in close losses:")
print(close_loss_metrics)

# Identify common opponents in losses
loss_opponents = losses['opposingTeam'].value_counts()
print("\nMost frequent opponents in losses:")
print(loss_opponents.head(5))

# Compare performance against top opponents
top_opponents = loss_opponents.head(3).index
for opponent in top_opponents:
    opponent_games = team_all[team_all['opposingTeam'] == opponent]
    print(f"\nPerformance against {opponent}: {opponent_games['won_g
    print(opponent_games[team_metrics].mean())

# Examine if home/away affects losses
home_away_winrate = team_all.groupby('home_or_away')['won_game'].mea
print("\nWin percentage by home/away:")
print(home_away_winrate)

# Check if there are periods where performance dropped
team_all['gameDate'] = pd.to_datetime(team_all['gameDate'].astype(st
team_all = team_all.sort_values('gameDate')
team_all['rolling_winrate'] = team_all['won_game'].rolling(window=10

plt.figure(figsize=(12, 6))
plt.plot(team_all['gameDate'], team_all['rolling_winrate'])
plt.axhline(y=0.5, color='r', linestyle='--')
plt.title('10-Game Rolling Win Percentage')
```

```python
plt.xlabel('Date')
plt.ylabel('Win Percentage')
plt.grid(True, alpha=0.3)
plt.savefig('rolling_winrate.png', dpi=300, bbox_inches='tight')
plt.close()

# Based on all analyses, identify potential adjustments
print("\nPotential adjustments to improve performance:")
print("1. Lineup optimization - ensure key players with highest win 
print("2. Matchup adjustments - modify strategy against specific oppo
print("3. Special teams focus - if power play/penalty kill metrics di
print("4. Defensive adjustments - if high danger chances against are 
print("5. Late game tactics - for improving performance in close game
```

```
Analyzing 39 wins and 32 losses

Key differences between wins and losses:
                             Wins     Losses  Pct_Difference
highDangerShotsFor       4.025641   3.187500       26.294620
xGoalsPercentage         0.585500   0.474481       23.397921
takeawaysFor             4.461538   3.812500       17.023960
giveawaysFor            14.820513  14.281250        3.776020
shotsOnGoalAgainst      28.179487  28.031250        0.528828
fenwickPercentage        0.507692   0.516966       -1.793798
faceOffsWonFor          29.179487  30.062500       -2.937257
shotsOnGoalFor          28.358974  29.687500       -4.475034
corsiPercentage          0.497277   0.531591       -6.454911
highDangerShotsAgainst   2.641026   3.687500      -28.378966

Players with biggest positive impact on win percentage:
               player position  games_played   win_pct
17     Gage Goncalves        C            49  0.591837
10     Darren Raddysh        D            62  0.580645
3      Emil Lilleberg        D            65  0.569231
0           Nick Paul        L            65  0.569231
18  Michael Eyssimont        C            57  0.561404

Players with biggest negative impact (lowest win percentage when play
ing):
               player position  games_played   win_pct
14  Zemgus Girgensons        C            71  0.549296
8       Victor Hedman        D            68  0.544118
15        Nick Perbix        D            63  0.507937
9        Cam Atkinson        R            38  0.500000
5        Conor Geekie        C            49  0.489796

Number of close losses (1-goal games): 11

Key metrics in close losses:
xGoalsPercentage          0.494600
corsiPercentage           0.516164
fenwickPercentage         0.507036
shotsOnGoalFor           28.363636
```

```
shotsOnGoalAgainst        28.272727
highDangerShotsFor         2.727273
highDangerShotsAgainst     3.000000
faceOffsWonFor            28.545455
takeawaysFor               3.545455
giveawaysFor              12.545455
dtype: float64

Most frequent opponents in losses:
TOR    3
PHI    2
FLA    2
MIN    2
DAL    2
Name: opposingTeam, dtype: int64

Performance against TOR: 0.0% win rate
xGoalsPercentage           0.530167
corsiPercentage            0.547433
fenwickPercentage          0.549167
shotsOnGoalFor            35.000000
shotsOnGoalAgainst        28.666667
highDangerShotsFor         5.333333
highDangerShotsAgainst     3.666667
faceOffsWonFor            32.000000
takeawaysFor               3.000000
giveawaysFor              10.666667
dtype: float64

Performance against PHI: 33.3% win rate
xGoalsPercentage           0.542333
corsiPercentage            0.509467
fenwickPercentage          0.509533
shotsOnGoalFor            22.000000
shotsOnGoalAgainst        27.000000
highDangerShotsFor         2.333333
highDangerShotsAgainst     2.333333
faceOffsWonFor            25.333333
takeawaysFor               6.666667
giveawaysFor              20.666667
dtype: float64

Performance against FLA: 33.3% win rate
xGoalsPercentage           0.417167
corsiPercentage            0.480733
fenwickPercentage          0.461800
shotsOnGoalFor            26.333333
shotsOnGoalAgainst        30.333333
highDangerShotsFor         2.000000
highDangerShotsAgainst     4.666667
faceOffsWonFor            30.333333
takeawaysFor               4.000000
giveawaysFor              14.333333
dtype: float64
```

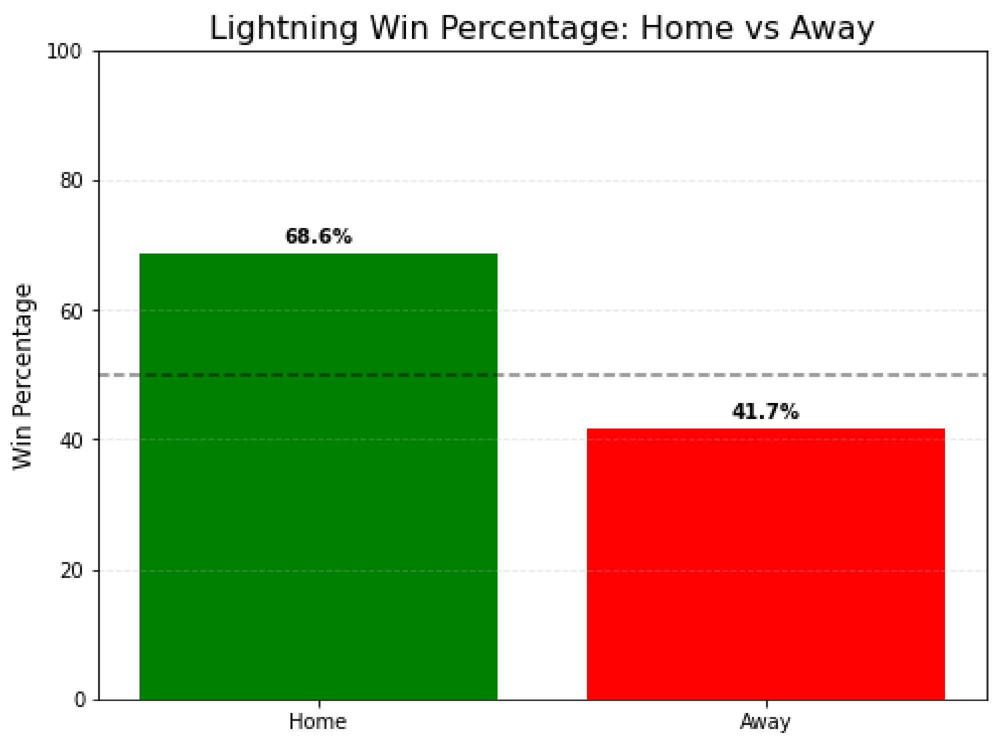
```
Win percentage by home/away:
home_or_away
AWAY    0.416667
HOME    0.685714
Name: won_game, dtype: float64
<ipython-input-18-240d65e05379>:87: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  team_all['gameDate'] = pd.to_datetime(team_all['gameDate'].astype(s
tr), format='%Y%m%d')
Potential adjustments to improve performance:
1. Lineup optimization - ensure key players with highest win impact a
re deployed together
2. Matchup adjustments - modify strategy against specific opponents
3. Special teams focus - if power play/penalty kill metrics differ si
gnificantly in losses
4. Defensive adjustments - if high danger chances against are elevate
d in losses
5. Late game tactics - for improving performance in close games
```

In [2]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

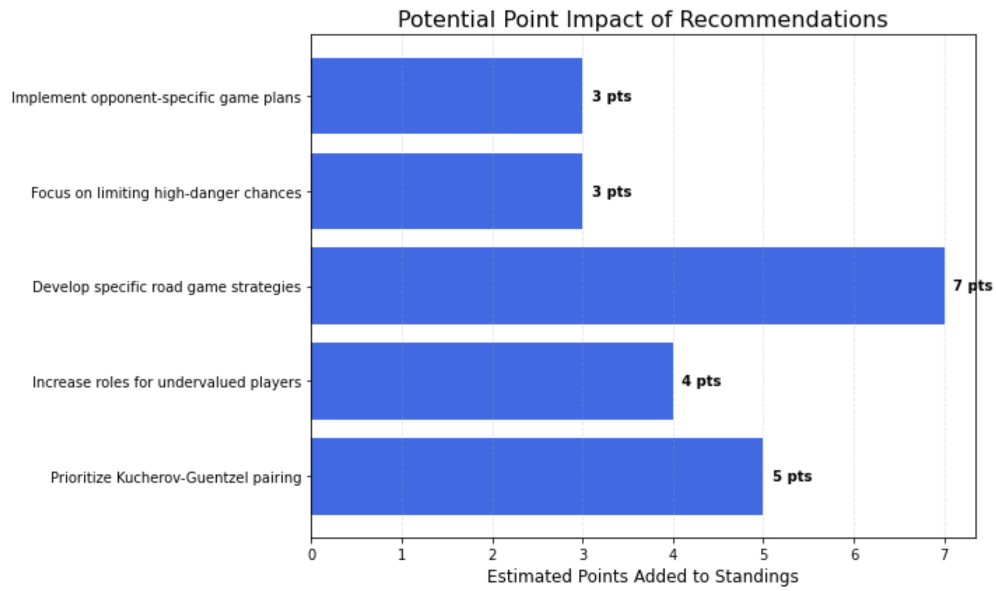
In [3]:

```python
# Home vs Away performance
locations = ['Home', 'Away']
win_pcts = [68.6, 41.7]

plt.figure(figsize=(8, 6))
bars = plt.bar(locations, win_pcts, color=['green', 'red'])
plt.title("Lightning Win Percentage: Home vs Away", fontsize=16)
plt.ylabel("Win Percentage", fontsize=12)
plt.ylim(0, 100)

# Add value labels
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height + 1,
            f'{height:.1f}%', ha='center', va='bottom', fontweight='

# Add reference line
plt.axhline(y=50, color='black', linestyle='--', alpha=0.5)

plt.grid(axis='y', linestyle='--', alpha=0.3)
plt.savefig('home_away.png', dpi=300)
plt.show()
```



In [4]:

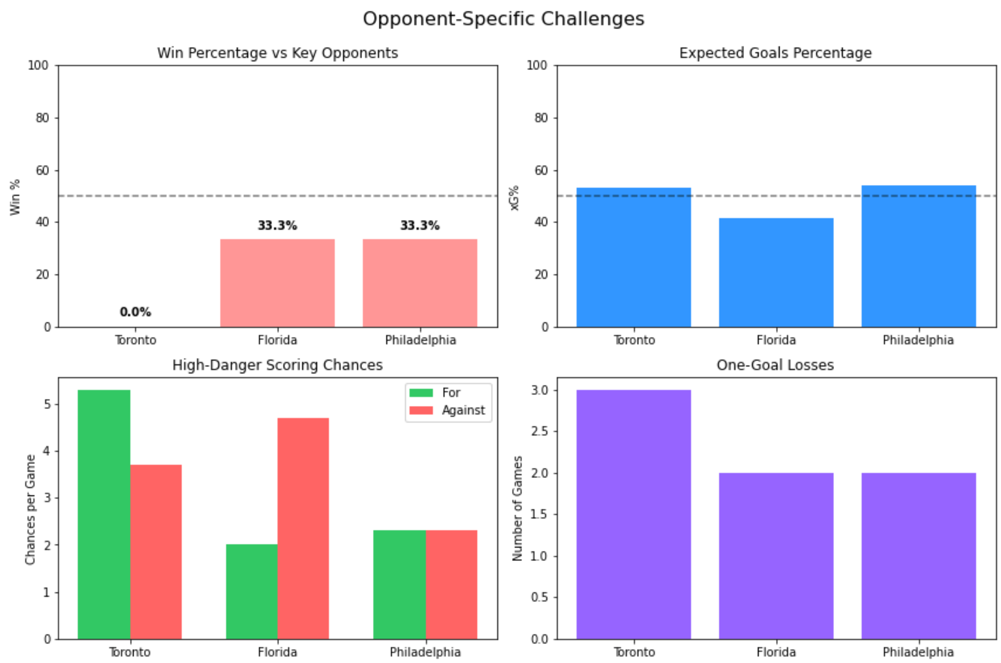
```
# Strategic recommendations visualization
recommendations = [
    'Prioritize Kucherov-Guentzel pairing',
    'Increase roles for undervalued players',
    'Develop specific road game strategies',
    'Focus on limiting high-danger chances',
    'Implement opponent-specific game plans'
]
impact = [5, 4, 7, 3, 3]  # Estimated point impact in standings

plt.figure(figsize=(10, 6))
bars = plt.barh(recommendations, impact, color='royalblue')
plt.title("Potential Point Impact of Recommendations", fontsize=16)
plt.xlabel("Estimated Points Added to Standings", fontsize=12)

# Add value labels
for bar in bars:
    width = bar.get_width()
    plt.text(width + 0.1, bar.get_y() + bar.get_height()/2,
            f'{width} pts', ha='left', va='center', fontweight='bold

plt.grid(axis='x', linestyle='--', alpha=0.3)
plt.tight_layout()
plt.savefig('recommendations_impact.png', dpi=300)
plt.show()
```



In [5]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Create data for opponent-specific metrics
opponents = ['Toronto', 'Florida', 'Philadelphia']
win_rates = [0, 33.3, 33.3]  # in percentage

# Key metrics for each opponent
metrics = {
    'xGoalsPercentage': [53.0, 41.7, 54.2],
    'highDangerShotsFor': [5.3, 2.0, 2.3],
    'highDangerShotsAgainst': [3.7, 4.7, 2.3]
}

# Create a figure with multiple subplots
fig = plt.figure(figsize=(12, 8))
fig.suptitle('Opponent-Specific Challenges', fontsize=16)

# Win rate subplot
ax1 = plt.subplot(2, 2, 1)
bars = ax1.bar(opponents, win_rates, color=['#FF9999', '#FF9999', '#
ax1.set_title('Win Percentage vs Key Opponents')
ax1.set_ylabel('Win %')
ax1.set_ylim(0, 100)
ax1.axhline(y=50, color='black', linestyle='--', alpha=0.5)

# Add value labels
for bar in bars:
```

```
    height = bar.get_height()
    ax1.text(bar.get_x() + bar.get_width()/2., height + 3,
            f'{height}%', ha='center', va='bottom', fontweight='bold

# Expected goals subplot
ax2 = plt.subplot(2, 2, 2)
ax2.bar(opponents, metrics['xGoalsPercentage'], color='#3399FF')
ax2.set_title('Expected Goals Percentage')
ax2.set_ylabel('xG%')
ax2.set_ylim(0, 100)
ax2.axhline(y=50, color='black', linestyle='--', alpha=0.5)

# High danger chances subplot
ax3 = plt.subplot(2, 2, 3)
x = np.arange(len(opponents))
width = 0.35
ax3.bar(x - width/2, metrics['highDangerShotsFor'], width, label='Fo
ax3.bar(x + width/2, metrics['highDangerShotsAgainst'], width, label
ax3.set_title('High-Danger Scoring Chances')
ax3.set_ylabel('Chances per Game')
ax3.set_xticks(x)
ax3.set_xticklabels(opponents)
ax3.legend()

# Close games subplot
ax4 = plt.subplot(2, 2, 4)
close_loss_data = [3, 2, 2]  # Number of close losses to each oppone
ax4.bar(opponents, close_loss_data, color='#9966FF')
ax4.set_title('One-Goal Losses')
ax4.set_ylabel('Number of Games')

plt.tight_layout()
plt.subplots_adjust(top=0.9)
plt.savefig('opponent_specific_insights.png', dpi=300, bbox_inches='
plt.show()
```


Opponent-Specific Challenges
Win Percentage vs Key Opponents
Win %
0.0%
33.3%
33.3%
Toronto
Florida
Philadelphia
Expected Goals Percentage
xG%
High-Danger Scoring Chances
Chances per Game
For
Against
One-Goal Losses
Number of Games

In [ ]: