March Madness Wins Predictive Model

Final Report:

Predicting March Madness Wins with Gradient Boosting

Introduction:

The objective of this project was to develop a predictive model for March Madness wins using team statistics. By leveraging machine learning techniques, specifically gradient boosting, we aimed to identify the most important predictors contributing to a team's success in the tournament. This report summarizes our findings and recommendations based on the analysis conducted.

Data Collection and Preprocessing:

We collected historical data on NCAA basketball games, including team statistics such as points per possession, 3-point percentage, and free throw percentage. The dataset was preprocessed to handle missing values, normalize features, and encode categorical variables. The data was then split into training and testing sets for model development and evaluation.

Model Development:

We employed gradient boosting, a powerful ensemble learning technique, to build a predictive model for March Madness wins. This approach allowed us to iteratively train weak learners and combine their predictions to improve overall accuracy. The model was trained using the identified predictors and validated using cross-validation techniques to ensure robustness.

Key Predictors:

Our analysis revealed that the most important predictors for March Madness wins were points per possession, 3-point percentage, and free throw percentage. These metrics consistently emerged as significant contributors to a team's success in the tournament. Teams with higher efficiency in scoring, particularly from beyond the arc and the free throw line, demonstrated a competitive advantage in securing victories.

Recommendations:

Based on our findings, we recommend that teams prioritize recruiting players from high schools with a strong track record in 3-point and free throw shooting. Investing in players who excel in these areas can enhance a team's offensive efficiency and increase their likelihood of success in March Madness. Furthermore, coaches and recruiters should consider incorporating these metrics into their player evaluation process to identify talent that aligns with their team's strategic objectives.

Conclusion:

In conclusion, our analysis highlights the importance of points per possession, 3-point percentage, and free throw percentage as key predictors of March Madness wins. By leveraging gradient boosting and focusing on these critical metrics, teams can make informed decisions in player recruitment and strategic planning to optimize their performance in the tournament. This report provides valuable insights for coaches, recruiters, and stakeholders seeking to enhance their team's competitiveness in NCAA basketball.

In [2]:
```
import pandas as pd

df = pd.read_csv('Barttorvik Neutral.csv')
```

Data Prep

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

Out[3]:
```
Index(['YEAR', 'TEAM NO', 'TEAM ID', 'TEAM', 'SEED', 'ROUND', 'BADJ EM',
       'BADJ O', 'BADJ D', 'BARTHAG', 'GAMES', 'W', 'L', 'WIN%', 'EFG%',
       'EFG%D', 'FTR', 'FTRD', 'TOV%', 'TOV%D', 'OREB%', 'DREB%', 'OP OREB%',
       'OP DREB%', 'RAW T', '2PT%', '2PT%D', '3PT%', '3PT%D', 'BLK%', 'BLKED%',
       'AST%', 'OP AST%', '2PTR', '3PTR', '2PTRD', '3PTRD', 'BADJ T',
       'AVG HGT', 'EFF HGT', 'EXP', 'TALENT', 'FT%', 'OP FT%', 'PPPO', 'PPPD',
       'ELITE SOS', 'WAB', 'BADJ EM RANK', 'BADJ O RANK', 'BADJ D RANK',
       'BARTHAG RANK', 'EFG% RANK', 'EFGD% RANK', 'FTR RANK', 'FTRD RANK',
       'TOV% RANK', 'TOV%D RANK', 'OREB% RANK', 'DREB% RANK', 'OP OREB% RANK',
       'OP DREB% RANK', 'RAW T RANK', '2PT% RANK', '2PT%D RANK', '3PT% RANK',
       '3PT%D RANK', 'BLK% RANK', 'BLKED% RANK', 'AST% RANK', 'OP AST% RANK',
       '2PTR RANK', '3PTR RANK', '2PTRD RANK', '3PTRD RANK', 'BADJT RANK',
       'AVG HGT RANK', 'EFF HGT RANK', 'EXP RANK', 'TALENT RANK', 'FT% RANK',
       'OP FT% RANK', 'PPPO RANK', 'PPPD RANK', 'ELITE SOS RANK'],
      dtype='object')
```

In [4]:
```
df.shape
```

Out[4]:
```
(1079, 85)
```

Data Cleaning

In [5]:
```
df = df.dropna()

# Assuming 'W' is your target column
X = df.drop(columns=['WIN%', 'TEAM', 'W', 'L', 'YEAR', 'TEAM NO', 'TEAM ID', 'GAMES', 'WAB'])
y = df['W']
```

Linear Regression

In [7]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate the model's performance on the testing set
r_squared = model.score(X_test, y_test)
print(f"R-squared: {r_squared:.2f}")

# Make predictions on the testing set
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
R-squared: 0.35
Mean Squared Error: 1.76
```

KNN Feature Selection

In [9]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.feature_selection import SelectKBest, f_regression

# Feature selection using SelectKBest
selector = SelectKBest(score_func=f_regression, k=5)  # Choose the top 5 features
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

# Initialize KNN regressor
knn = KNeighborsRegressor(n_neighbors=5)  # You can adjust 'n_neighbors'

# Fit the model
knn.fit(X_train_selected, y_train)

# Calculate R-squared
r_squared = knn.score(X_test_selected, y_test)
print(f"R-squared: {r_squared:.2f}")

# Make predictions
y_pred = knn.predict(X_test_selected)

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
R-squared: 0.18
Mean Squared Error: 2.22
```

Hyperparameter Tuning

In [10]:
```
from sklearn.model_selection import GridSearchCV

# Define a range of k values to evaluate
param_grid = {'n_neighbors': [3, 5, 7, 9, 11]}

# Initialize KNN regressor
knn = KNeighborsRegressor()

# Perform grid search with cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

# Get the best k value
best_k = grid_search.best_params_['n_neighbors']
print(f"Best k value: {best_k}")

# Use the best k value to train the final model
best_knn = KNeighborsRegressor(n_neighbors=best_k)
best_knn.fit(X_train, y_train)

# Evaluate the model
y_pred = best_knn.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error with best k: {mse:.2f}")
```

```
Best k value: 11
Mean Squared Error with best k: 2.26
```

Random Forest

In [13]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize Random Forest regressor
random_forest = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
random_forest.fit(X_train, y_train)

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

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 1.52
```

Gradient Boosting

In [11]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Initialize Gradient Boosting regressor
gradient_boosting = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)

# Train the model
gradient_boosting.fit(X_train, y_train)

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

# Evaluate the model using Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
```

```
Mean Squared Error: 1.48
```

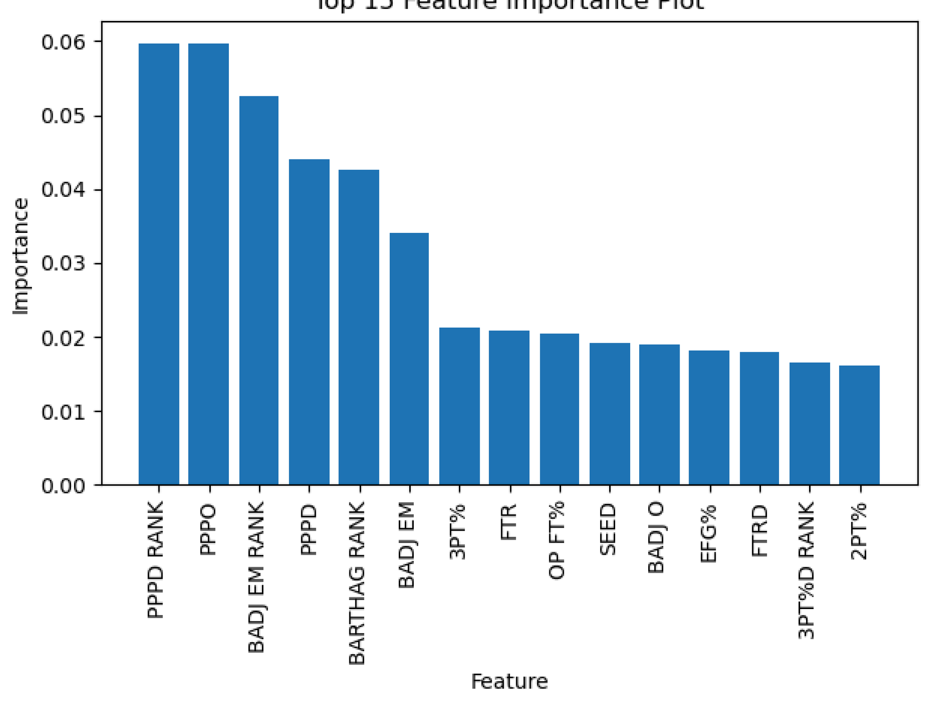
Model Comparison

In [14]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import GradientBoostingRegressor

# Initialize and train other regression models
linear_regression = LinearRegression()
linear_regression.fit(X_train, y_train)

gradient_boosting = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
gradient_boosting.fit(X_train, y_train)

# Evaluate the models
mse_random_forest = mean_squared_error(y_test, random_forest.predict(X_test))
mse_linear_regression = mean_squared_error(y_test, linear_regression.predict(X_test))
mse_gradient_boosting = mean_squared_error(y_test, gradient_boosting.predict(X_test))

print(f"MSE Random Forest: {mse_random_forest:.2f}")
print(f"MSE Linear Regression: {mse_linear_regression:.2f}")
print(f"MSE Gradient Boosting: {mse_gradient_boosting:.2f}")
```

```
MSE Random Forest: 1.52
MSE Linear Regression: 1.76
MSE Gradient Boosting: 1.48
```

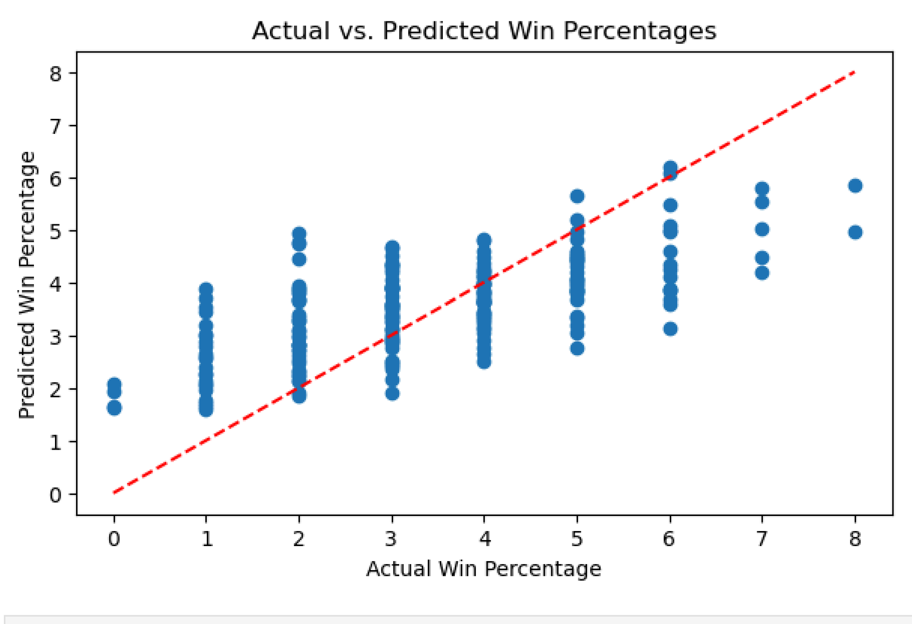
Data Visualizations

In [15]:
```
import matplotlib.pyplot as plt

# Get feature importances from the model
feature_importances = random_forest.feature_importances_

# Sort feature importances in descending order
sorted_indices = feature_importances.argsort()[::-1]

# Plot feature importances for the top 15 features
top_features = 15
plt.figure(figsize=(7, 4))
plt.bar(range(top_features), feature_importances[sorted_indices][:top_features], align='center')
plt.xticks(range(top_features), X_train.columns[sorted_indices][:top_features], rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Top 15 Feature Importance Plot')
plt.show()
```

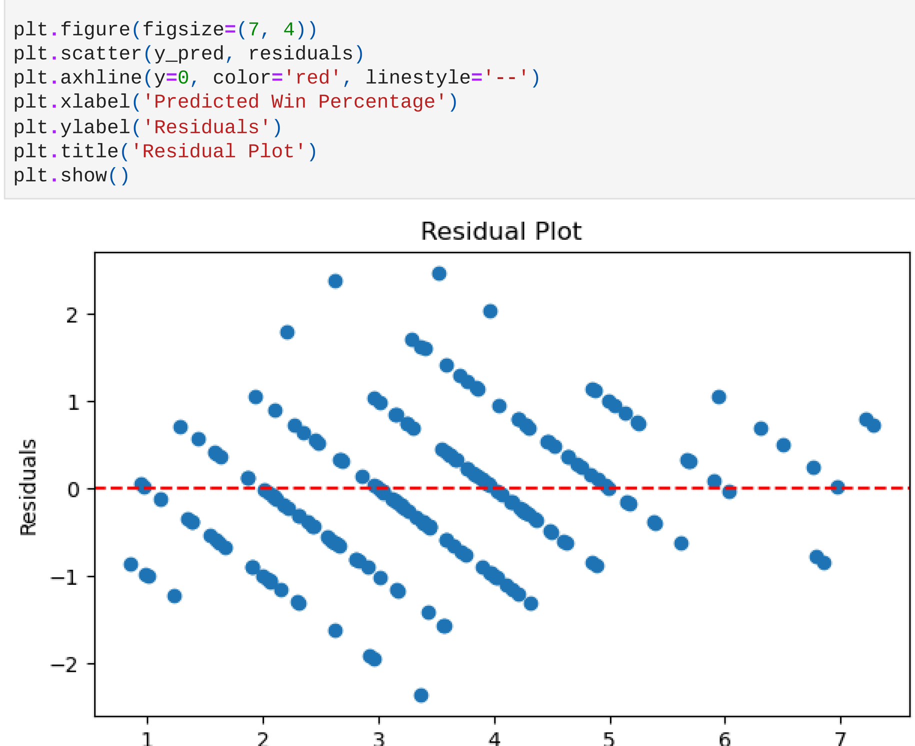


Point Per Possession, 3-Point Percentage and Free-Throw Percentage

In [16]:
```
plt.figure(figsize=(7,4))
plt.scatter(y_test, y_pred)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--', color='red')
plt.xlabel('Actual Win Percentage')
plt.ylabel('Predicted Win Percentage')
plt.title('Actual vs. Predicted Win Percentages')
plt.show()
```

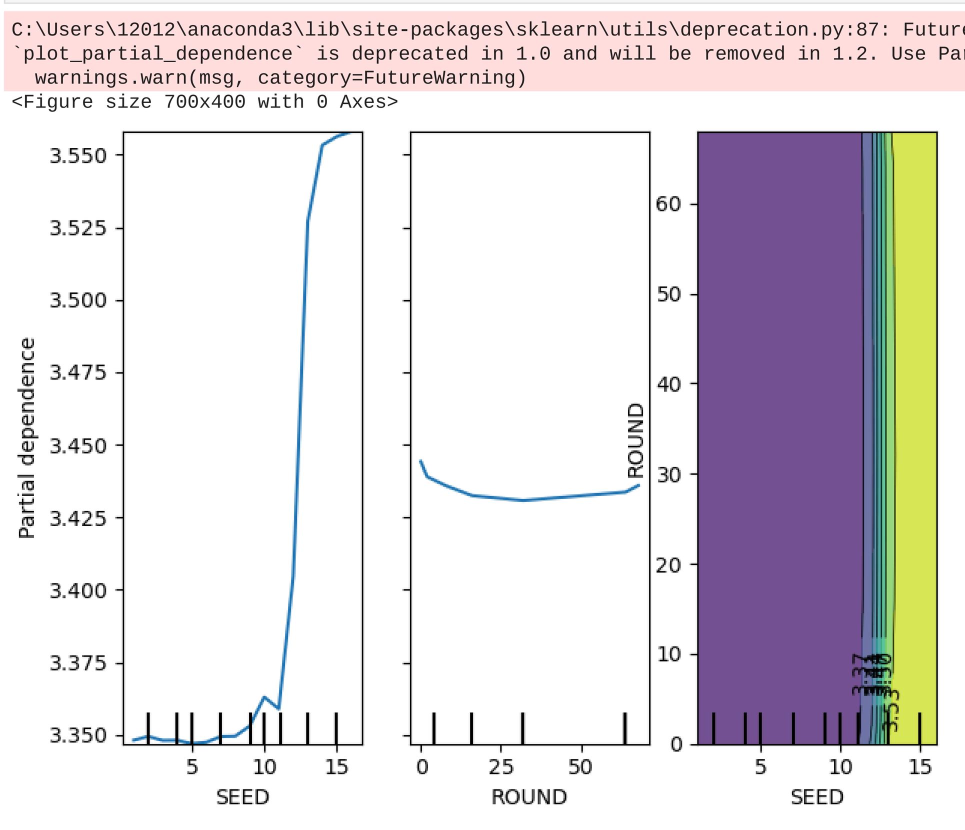


In [22]:
```
residuals = y_test - y_pred

plt.figure(figsize=(7, 4))
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Win Percentage')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



In [23]:
```
from sklearn.inspection import plot_partial_dependence

plt.figure(figsize=(7, 4))
plot_partial_dependence(random_forest, X_train, features=[0, 1, (0, 1)])  # Plot first three features
plt.tight_layout()
plt.show()
```

```
C:\Users\12012\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_partial_dependence is deprecated; Function `plot_partial_dependence` is deprecated in 1.0 and will be removed in 1.2. Use PartialDependenceDisplay.from_estimator instead
  warnings.warn(msg, category=FutureWarning)
<Figure size 700x400 with 0 Axes>
```



In [ ]: