University of California, Berkeley

Democratizing Sports Analytics:

Predicting NBA Player Performance with Data-Driven Insights

Joseph Chen, Darren Chiang, Talha Tariq, Anh-Thu Bui

IEOR 142A

Professor Paul Grigas

December 13, 2024

Introduction: Motivation and Impact

Sports have always been a large part of how and why people congregate, build closer connections, and have fun. Through watching athletes, partaking in local leagues like college intramural teams, and analyzing sportsbooks, it seems to us that sports is and has always been largely a combination of skill and strategy as well as unpredictability. Here in the Bay Area, surrounded by Warriors fans and graced by the presence of the greatest NBA shooter in history, Steph Curry, our team was inspired to create a project that blended analytics with the excitement of the game. Initially, our goal was pretty simple and straight-forward: predict whether Steph Curry would go over or under specific stat lines in his next game. This project aimed to help out fans, fantasy sports players, and even bettors who appreciate data-driven insights. However, as we delved deeper into the project, it evolved into something far more ambitious: a scalable model capable of predicting performance for any NBA player. This shift dramatically expanded the potential utility and impact of our work, and honestly, the fun of the work.

We found that open-source predictors for sports data represent quite a significant opportunity to democratize access to predictive analytics. As fantasy sports and sports betting grow in popularity, especially as we have seen in the past few years with the rise of sports betting apps, accurate and accessible tools have become indispensable for enthusiasts. Furthermore, the fast-paced nature of sports and changing scene demands constantly updated data sources. Our model addresses this need by utilizing datasets that refresh regularly, ensuring predictions remain relevant and practical. This combination of accessibility, up-to-date data, and versatility highlights the value of our work for a wide range of applications.

Data Collection: NBA Data API

Our journey through data collection was as complicated as it was interesting. Initially, we relied on Kaggle for historical game statistics, but the datasets we found were incomplete and outdated. Although we had included the NBA Python API in our proposal as a potential fallback, we initially underestimated how useful it was. After exhausting a lot of other options, many of which required costly subscriptions, we revisited the NBA API and discovered that it's genuinely really informative and detailed. It provided everything from game logs to matchup-specific details, but extracting and cleaning the data required a lot of understanding, research, and effort.

The NBA API, which sourced data from NBA.com, proved perfect for our needs because of its immediate updates, which are crucial for sports predictions. Predictions grounded in stale data lose their relevance very very quickly, but this API allows us to build models that adapt dynamically to recent performance trends. This approach was also very aligned with our open-source interest as it offered a powerful and free solution without sacrificing quality.

One of the most interesting techniques we employed involved linking multiple endpoints within the API. For example, we needed to integrate player game logs with team statistics and historical rankings. This required careful mapping using player IDs, game IDs, and even regex-based parsing of matchup strings to identify opposing teams accurately. By combining these datasets, we created a comprehensive view of player performance that included rolling averages, opponent strengths, and contextual factors like game location and pace.

Here's the GitHub repository where the API Client package is: https://github.com/swar/nba_api

Modeling: Building the Pipeline, One Phase at a Time

Phase 1: Establishing a Baseline

We began with a few foundational models: Logistic Regression (simple and easy to interpret), Random Forest (more robust, can handle non-linear relationships), and Neural Networks (for complex patterns). At this stage, our feature set consisted mostly of rolling averages for key player stats, such as points, assists, and minutes. This allowed us to create a foundation and establish baseline performance metrics. While these early models lacked complexity, they gave a roadmap for improvement and better understanding of how we wanted to structure everything.

Phase 2: Expanding Features

Expanding the feature set was a key step in our project as it added important elements like opponent statistics and historical team rankings, nearly doubling the dataset's scope. By leveraging additional NBA API endpoints, we enriched the data with details such as team standings and rolling performance averages, which boosted predictive potential but also introduced challenges like multicollinearity and redundancy. As shown partially in **Appendix A**, we addressed these issues through a detailed feature selection process. Using correlation analysis, we removed ~15-20% of overlapping and highly interdependent features, such as redundant shooting metrics. We then applied Variance Inflation Factor (VIF) analysis to further refine the dataset, systematically eliminating features with excessive variance inflation while preserving key predictors like rolling points and minutes, which we identified based on domain expertise. This careful refinement process resulted in a cleaner, more balanced feature set, improving both model stability and interpretability while maintaining strong predictive power.

Phase 3: Optimizing Models

Optimization was the core of this phase. Logistic Regression required minimal adjustments, but Random Forest saw significant improvements through K-Fold Cross-Validation and hyperparameter tuning with GridSearchCV. For the Neural Network, we implemented a Multi-Layer Perceptron and fine-tuned parameters such as batch size and learning rate using RandomizedSearchCV. These refinements resulted in better precision, recall, and overall performance, although the Neural Network remained the most challenging to optimize due to its sensitivity to data imbalance and configuration and the fact that our dataset was relatively small. There are just not many NBA games played by a player a year, especially if we're splitting data.

Phase 4: Exploring Ensemble Methods
In the final phase, we experimented with ensemble techniques. Initially, we employed a VotingClassifier for its simplicity, combining the predictions of our base models. Later, we tried out a StackingClassifier, which uses a meta-model to learn from the combined outputs of individual models. For some analyses and predictions, this model proved to have the highest accuracy and recall metrics, showcasing the power of ensemble methods in leveraging the strengths of multiple models. While we considered adding Linear Discriminant Analysis (LDA), we ultimately decided against it due to its limitations in binary classification tasks.

Results and Observations:
Our results highlighted the complexities of sports prediction. Here are some examples of player-specific predictions for the NBA bets that were available on December 12, 2024.

- **Cade Cunningham (9.5 assists):** Predicted "Under" with 80% confidence. Logistic Regression did really well here, achieving an F1 score of 0.87. ***Result: Correct (8 assists)***
- **Tyler Herro (3.5 three-pointers made):** Predicted "Over" with 53% confidence. Gradient Boosting performed best, using shooting-related metrics to enhance precision. ***Result: Correct (4 three-pointers made)***
- **Giannis Antetokounmpo (32.5 points):** Predicted "Under" with 54% confidence. All models struggled due to data imbalance and the inherent unpredictability of high-scoring players, especially Mr. Antetokounmpo here. ***Result: Game is on Saturday 12/14***
- **Trae Young (11.5 assists):** Predicted "Over" with 72% confidence. Assist-related features were vital, and both Gradient Boosting and Neural Networks performed well. ***Result: Game is on Saturday 12/14***
- **Alperen Sengun (4.5 assists):** Predicted "Under" with strong performance from Logistic Regression, which achieved an accuracy of 88%. ***Result: Game is on Saturday 12/14***

One of the most striking findings was the variability in feature importance across players and stat lines. For instance, heavy scorers like Giannis required metrics such as rolling points and field goal efficiency, while assist-dominant players like Trae Young benefited from features related to team dynamics and game context. We can see this in **Appendix B** where different models assign varying levels of importance to the same features, even when predicting the same stat line for the same player. Additionally, we see it also in **Appendix C** where the same model assigns prioritize different features even when predicting the same statistic (assists) for different players. This variability showed us the challenge of creating a one-size-fits-all model for sports predictions, particularly for basketball.

As seen in the code and the five examples provided, there isn't one model that handles this task universally well. Different players, stat lines, and chosen statistics favor different models, further emphasizing the complexity and unpredictability of sports analytics. For example, while Gradient Boosting captured subtle shooting patterns for Tyler Herro, Logistic Regression excelled at simpler predictions like Cade Cunningham's assists, and ensemble methods worked better for other scenarios.

The naive baseline provided an important reference point. For every prediction task, we evaluated how well our models performed compared to simply predicting the most frequent outcome. For instance, predicting "Under" for all stat lines achieved reasonable accuracy due to the natural skew in performance data. However, our models consistently outperformed this baseline, particularly for more nuanced tasks like, again, predicting Herro's three-pointers. Keep in mind that all stat lines are assumed to be as balanced and fair as possible since the predictor is meant to deal with equilibrium stat lines set by sportsbooks and betting apps.

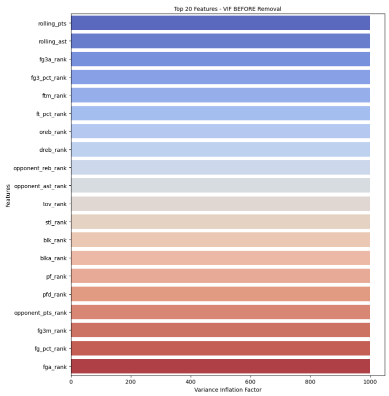
Finally, as highlighted in **Appendix D**, the challenges of imbalanced classes and a small dataset further complicate predictions. The ROC curves show that no model can fully overcome these limitations, with straight-line segments reflecting the restricted granularity of thresholds due to limited data and class imbalance. For example, Logistic Regression struggled with nuanced patterns, while more advanced models like Gradient Boosting showed stronger but still imperfect performance. These challenges demonstrate the need for larger datasets, better balance, and tailored modeling approaches to improve prediction accuracy in such a dynamic area like sports.

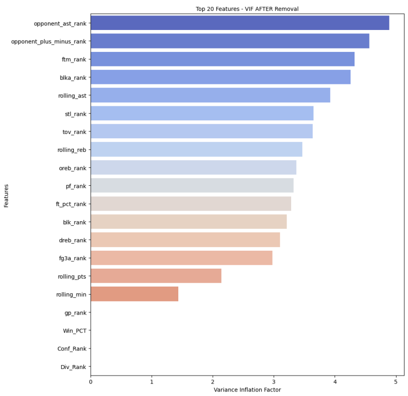
Reflections and Future Directions:
This project was both challenging and rewarding. What began as a focused exploration of Steph Curry's stats evolved into a more powerful tool for analyzing any NBA player's performance. Along the way, we navigated the so so so many difficulties and complexities of data collection, feature engineering, and model optimization, gaining really cool insights into the complexities of sports analytics and analytics in general.

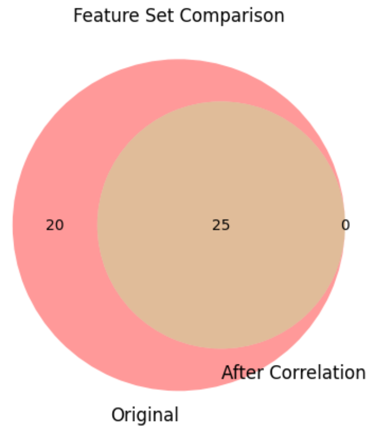
While we achieved promising results, several limitations remain, such as data imbalance due to extreme stat lines at times, ineffectiveness of oversampling techniques due to high dimensionality and feature interdependence, lack of understanding for real-time fluctuations such as injuries or changes in player roles, lack of enough training data, and overall model sensitivity.

If we were to continue working on this, we would work towards incorporating real-time game data and much more advanced contextual features that would make predictions more accurate and nuanced. Additionally, it would be really interesting to experiment with new techniques like generative oversampling or weighted loss functions, which just might improve model performance in imbalanced scenarios. Ultimately, this project has helped us work towards our motivation of not only trying to build an interesting predictive model, but also more importantly towards democratizing sports analytics for all, making it more accessible and equitable for fans and general enthusiasts who just want to better understand and engage with the game they love.

Appendix A: VIF Feature Removal



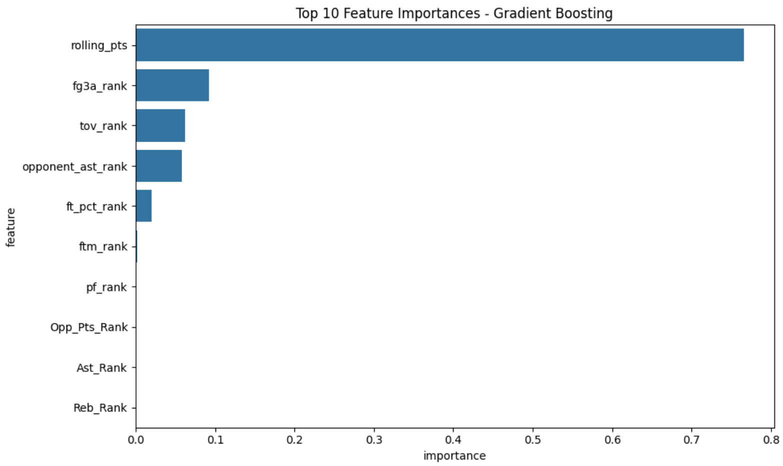


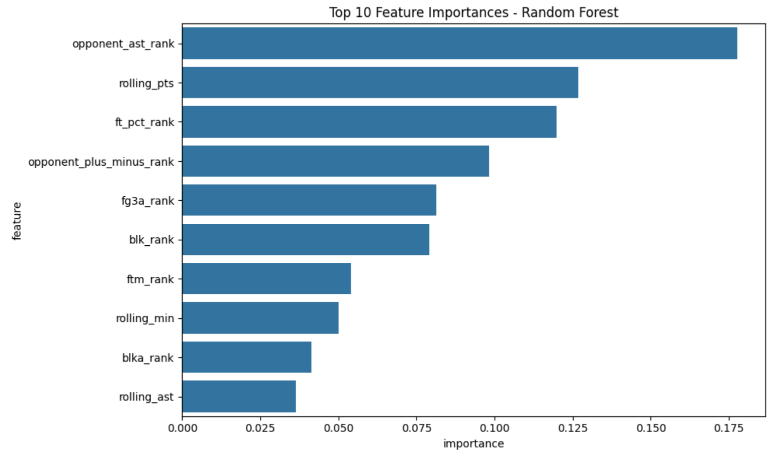


In this appendix, we show how we tackled multicollinearity in our dataset. At first, some features had infinite or really large VIF values, making it clear that there was too much overlap between them. To fix this, we carefully removed redundant features until all the remaining ones had VIF values below 5, which we considered a much healthier range.

After this cleanup, our feature set for one example shrank from 45 features down to 25. You can see the difference in the charts. This process not only simplified the dataset but also made our models more reliable and easier to interpret.

Appendix B: Differences in Feature Importances Between Models

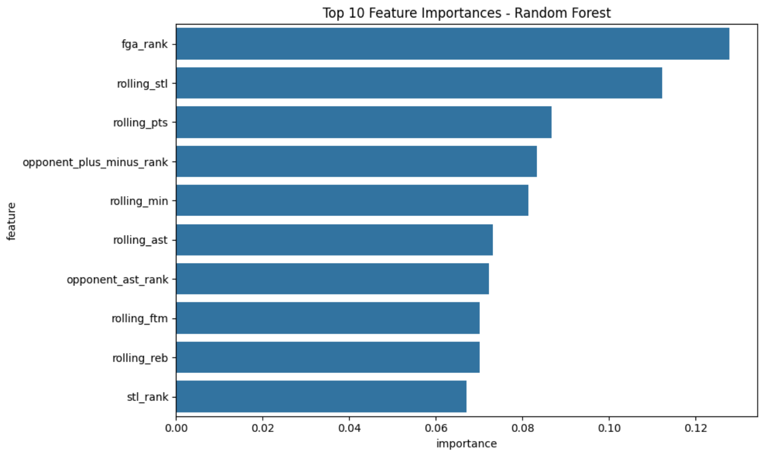




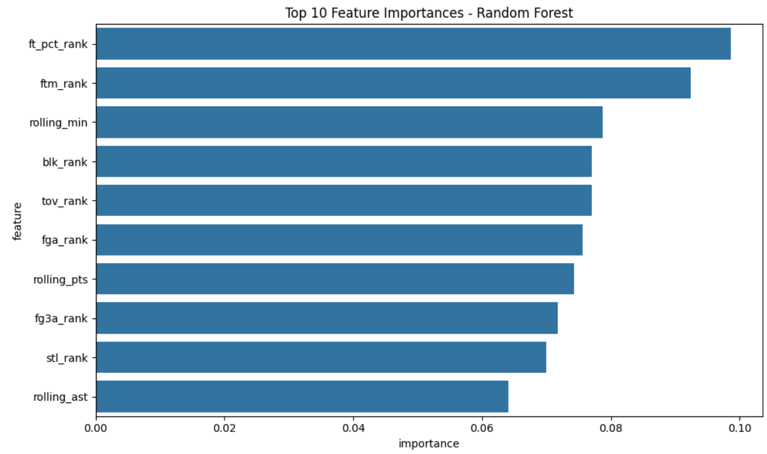
This appendix shows how different machine learning models prioritize features differently, even when predicting the same stat line for the same player using the same data. On the left, the Gradient Boosting model relies heavily on "rolling_pts" (recent scoring trends), making it the most important feature by far. Other features, like "fg3a_rank" (three-point attempts rank) and "opponent_ast_rank" (opponent assist rank), play much smaller roles. In contrast, the Random Forest model, shown on the right, spreads the importance more evenly across features. While "opponent_ast_rank" and "rolling_pts" are still important, other features like "ft_pct_rank" (free throw percentage rank) and "opponent_plus_minus_rank" are also highlighted.

Appendix C: Differences in Feature Importances Within Same Models and Statistics But Different Players

Cade Cunningham's Assists

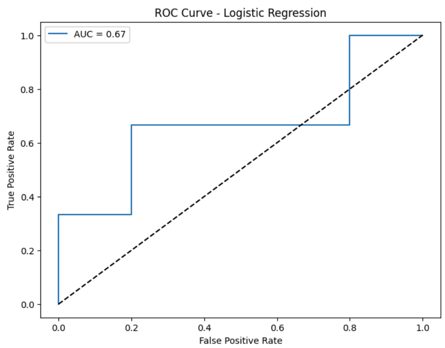


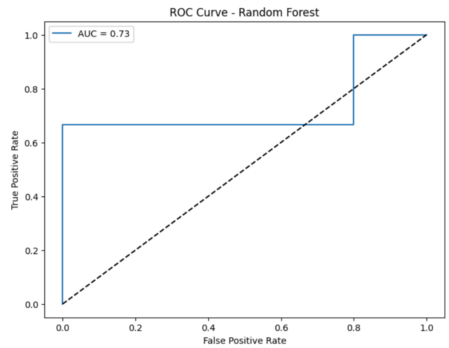
Trae Young's Assists

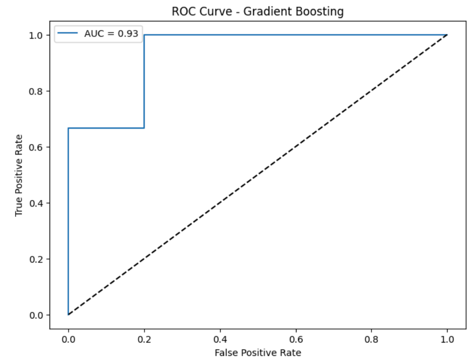


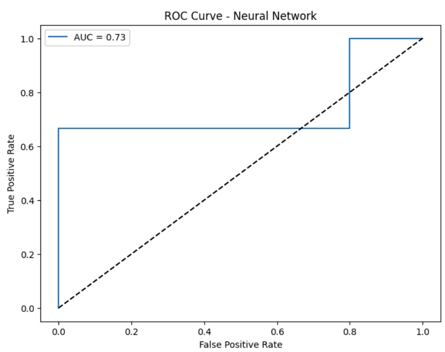
In this appendix, we see how the same model, Random Forest, assigns different feature importance when predicting assists for Cade Cunningham versus Trae Young, highlighting the unique factors influencing each player. For Cade, features like "fga_rank" (field goal attempts rank) and "rolling_stl" (recent steals) are most important, suggesting his assists are tied to scoring opportunities and defensive impact. In contrast, Trae's assist predictions prioritize "ft_pct_rank" (free throw percentage rank) and "ftm_rank" (free throws made), reflecting his focus on offensive efficiency and playmaking. This difference shows that even when predicting the same stat line, player-specific play styles and contexts lead to varying feature importance, making generalized sports predictions challenging and requiring models to adapt to individual circumstances.

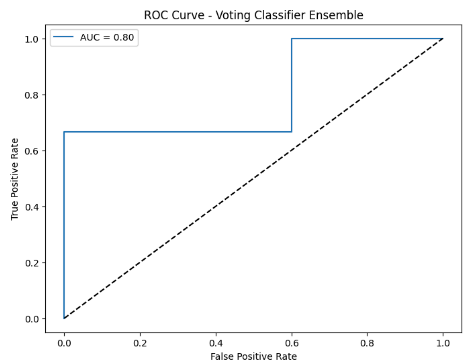
Appendix D: ROC Curves and the Difficulty of Imbalanced Classes and Lack of Data

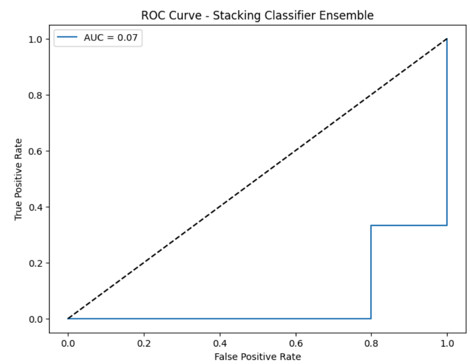












This appendix shows the challenges caused by imbalanced classes and limited data as seen in the ROC curves for various models. The straight-line segments in the curves suggest that the models often produce only a few distinct probability scores, very much likely due to small dataset size and limited positive or negative examples. This lack of granularity prevents the models from generating smoother thresholds for true and false positive rates. Additionally, highly imbalanced data skews the results, making it difficult for the models to handle class distributions effectively. Simpler models, like Logistic Regression, further struggle to learn complex patterns, while more advanced models like Gradient Boosting perform better but still face limitations due to the dataset's constraints. Overall, these results highlight the need for better data representation, larger datasets, and improved handling of imbalances to achieve more reliable predictions.

RELEVANT LINKS

GitHub for Notebook Code:
https://github.com/josephchen1/ieor-142a-nba-predictions/blob/main/final_nba_cleaned_up_joseph.ipynb

Shortened: https://tinyurl.com/ieor142anba

NBA API GitHub Link:
https://github.com/swar/nba_api/

Shortened: https://tinyurl.com/ieor142anbaapi

```
!pip install nba_api
```

```
Requirement already satisfied: nba_api in /usr/local/lib/python3.10/dist-packages (1.6.1)
Requirement already satisfied: numpy<2.0.0,>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from nba_api) (1.26.4)
Requirement already satisfied: requests<3.0.0,>=2.32.3 in /usr/local/lib/python3.10/dist-packages (from nba_api) (2.32.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.32.3->nba_ap
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.32.3->
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.32.3->
```

Defining all the needed functions and imports

Import statements

```
import pandas as pd
import numpy as np
from sklearn.model_selection import (
    train_test_split,
    cross_val_score,
    GridSearchCV,
    RandomizedSearchCV,
    StratifiedKFold,
    KFold,
    learning_curve
)
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import (
    RandomForestClassifier,
    GradientBoostingClassifier,
    VotingClassifier,
    StackingClassifier
)
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    roc_curve,
    roc_auc_score,
    precision_recall_curve,
    average_precision_score,
    f1_score,
    precision_score,
    recall_score
)
from statsmodels.stats.outliers_influence import variance_inflation_factor
from nba_api.stats.endpoints import (
    playergamelog,
    playernextngames,
    commonplayerinfo,
    leaguedashteamstats,
    teaminfocommon,
    teamgamelogs,
    playerindex
)
import matplotlib.pyplot as plt
from matplotlib_venn import venn2, venn3
import seaborn as sns
import warnings
```

Data Setup: Mappings + Utility Functions

```
TEAM_NAME_TO_ID = {
    "Atlanta": 1610612737, "Boston": 1610612738, "Brooklyn": 1610612751, "Charlotte": 1610612766,
    "Chicago": 1610612741, "Cleveland": 1610612739, "Dallas": 1610612742, "Denver": 1610612743,
    "Detroit": 1610612765, "Golden State": 1610612744, "Houston": 1610612745, "Indiana": 1610612754,
    "LA Clippers": 1610612746, "Lakers": 1610612747, "Memphis": 1610612763, "Miami": 1610612748,
```

```
    "Milwaukee": 1610612749, "Minnesota": 1610612750, "New Orleans": 1610612740, "New York": 1610612752,
    "Oklahoma City": 1610612760, "Orlando": 1610612753, "Philadelphia": 1610612755, "Phoenix": 1610612756,
    "Portland": 1610612757, "Sacramento": 1610612758, "San Antonio": 1610612759, "Toronto": 1610612761,
    "Utah": 1610612762, "Washington": 1610612764
}

TEAM_ABBR_TO_ID = {
    "ATL": 1610612737, "BOS": 1610612738, "BKN": 1610612751, "CHA": 1610612766,
    "CHI": 1610612741, "CLE": 1610612739, "DAL": 1610612742, "DEN": 1610612743,
    "DET": 1610612765, "GSW": 1610612744, "HOU": 1610612745, "IND": 1610612754,
    "LAC": 1610612746, "LAL": 1610612747, "MEM": 1610612763, "MIA": 1610612748,
    "MIL": 1610612749, "MIN": 1610612750, "NOP": 1610612740, "NYK": 1610612752,
    "OKC": 1610612760, "ORL": 1610612753, "PHI": 1610612755, "PHX": 1610612756,
    "POR": 1610612757, "SAC": 1610612758, "SAS": 1610612759, "TOR": 1610612761,
    "UTA": 1610612762, "WAS": 1610612764
}

def log(message, level="INFO"):
    """
    This is to help with logging stuff such as DEBUG, INFO, etc.
    """
    print(f"[{level}]: {message}")

warnings.filterwarnings('ignore', category=UserWarning)
warnings.filterwarnings('ignore', category=RuntimeWarning)
```

Data Fetching Functions

```
def fetch_player_details(player_id):
    """
    This helps fetch all the player details to verify we're getting the right player and player_id.
    """
    player_info = commonplayerinfo.CommonPlayerInfo(player_id=player_id)
    player_details = player_info.get_data_frames()[0].iloc[0]
    print("\nPlayer Details:")
    print(f"Name: {player_details['DISPLAY_FIRST_LAST']}")
    print(f"Team: {player_details['TEAM_NAME']}")
    print(f"Position: {player_details['POSITION']}")
    return player_details

def fetch_team_game_logs(season):
    """
    This fetches the team game logs for a given season.
    """
    try:
        team_game_logs = teamgamelogs.TeamGameLogs(
            season_nullable=season,
            season_type_nullable="Regular Season"
        )
        team_logs_df = team_game_logs.get_data_frames()[0]
        team_logs_df.columns = team_logs_df.columns.str.lower()
        print(f"Fetched team game logs for season {season}, shape: {team_logs_df.shape}")
        return team_logs_df
    except Exception as e:
        print(f"Error fetching team game logs: {e}")
        return pd.DataFrame()


def fetch_next_game_details(player_id, season):
    """
    This just fetches details of the player's next game, which we're trying to predict on.
    """
    player_next_games = playernextngames.PlayerNextNGames(
        player_id=player_id,
        season_type_all_star='Regular Season',
        number_of_games=1
    )
    next_game_details = player_next_games.get_data_frames()[0].iloc[0]
    print("\nNext Game Details:")
    print(next_game_details[['GAME_DATE', 'HOME_TEAM_NAME', 'VISITOR_TEAM_NAME', 'GAME_TIME']])
    return next_game_details


def fetch_player_data(player_id, seasons):
    """
```

```
    This fetches the game logs for a player across multiple seasons, which is the data we're using to predict with.
    """
    data_frames = []
    for season in seasons:
        game_log = playergamelog.PlayerGameLog(
            player_id=player_id,
            season=season,
            season_type_all_star='Regular Season'
        )
        df = game_log.get_data_frames()[0]

        df.columns = df.columns.str.lower()

        print(f"Columns in fetched player game log for season {season}: {df.columns}")
        if 'game_id' not in df.columns:
            raise ValueError(f"'game_id' not found in player game log for season {season}")

        df['season'] = season
        data_frames.append(df)
    print(f"\nFetched player data shape: {pd.concat(data_frames).shape}")
    return pd.concat(data_frames, ignore_index=True)


def fetch_team_info(team_id, season=None, season_type=None):
    """
    This fetches team info, including win percentage and rankings, which turned out to be honestly not as helpful as I thought
    """
    team_info = teaminfocommon.TeamInfoCommon(
        team_id=team_id,
        league_id="00"
    )

    try:
        team_info_common = team_info.get_data_frames()[0]
        team_season_ranks = team_info.get_data_frames()[1]

        team_features = {
            'Win_PCT': team_info_common.loc[0, 'PCT'],
            'Conf_Rank': team_info_common.loc[0, 'CONF_RANK'],
            'Div_Rank': team_info_common.loc[0, 'DIV_RANK'],
        }

        team_features.update({
            'Pts_Rank': team_season_ranks.loc[0, 'PTS_RANK'],
            'Reb_Rank': team_season_ranks.loc[0, 'REB_RANK'],
            'Ast_Rank': team_season_ranks.loc[0, 'AST_RANK'],
            'Opp_Pts_Rank': team_season_ranks.loc[0, 'OPP_PTS_RANK'],
        })

    except (KeyError, IndexError) as e:
        print(f"Error fetching team info: {e}")
        team_features = {}

    return team_features

def fetch_player_id(player_name, season="2024-25"):
    """
    This gets the PlayerID using the player's name and season.
    """
    try:
        player_index = playerindex.PlayerIndex(season=season)
        player_data = player_index.get_normalized_dict()["PlayerIndex"]

        for player in player_data:
            full_name = f"{player['PLAYER_FIRST_NAME']} {player['PLAYER_LAST_NAME']}"
            if full_name.lower() == player_name.lower():
                print(player["PERSON_ID"])
                return player["PERSON_ID"]

        raise ValueError(f"Player {player_name} not found in season {season}.")
    except Exception as e:
        raise RuntimeError(f"Error fetching PlayerID for {player_name}: {e}")
```

Feature Engineering Functions

```python
def calculate_rolling_averages(player_data):
    """
    This calculates all the rolling averages for key stats we need.
    """
    stats_to_average = [
        'pts', 'ast', 'reb', 'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a',
        'ftm', 'fta', 'oreb', 'dreb', 'stl', 'blk', 'tov', 'plus_minus'
    ]
    for stat in stats_to_average:
        rolling_col = f'rolling_{stat}'
        player_data[rolling_col] = player_data[stat].rolling(window=3, min_periods=1).mean()
    print("\nRolling averages calculated successfully.")
    return player_data

def add_opponent_rankings(player_data, season):
    """
    This adds opponent rankings to the player dataset, which expanded our feature set quite notably.
    """
    team_logs = fetch_team_game_logs(season)

    team_logs.columns = team_logs.columns.str.lower()

    opponent_rank_columns = {
        "w_pct_rank": "opponent_w_pct_rank",
        "pts_rank": "opponent_pts_rank",
        "reb_rank": "opponent_reb_rank",
        "ast_rank": "opponent_ast_rank",
        "plus_minus_rank": "opponent_plus_minus_rank",
    }
    team_logs = team_logs.rename(columns=opponent_rank_columns)
    team_logs_keyed = team_logs.set_index(["team_id", "game_id"])

    opponent_data = []
    for _, row in player_data.iterrows():
        matchup = row["matchup"]
        player_team_abbr = matchup.split(" ")[0]
        opponent_team_abbr = matchup.split(" ")[-1]

        if "@" in matchup:
            opponent_team_abbr = opponent_team_abbr
        elif "vs." in matchup:
            opponent_team_abbr = opponent_team_abbr

        player_team_id = TEAM_ABBR_TO_ID.get(player_team_abbr)
        opponent_team_id = TEAM_ABBR_TO_ID.get(opponent_team_abbr)

        if opponent_team_id is None:
            print(f"Error: Team abbreviation not found for {opponent_team_abbr}.")
            opponent_stats = {col: np.nan for col in opponent_rank_columns.values()}
        else:
            game_id = row["game_id"]
            try:
                opponent_stats = team_logs_keyed.loc[(opponent_team_id, game_id)].to_dict()
            except KeyError:
                opponent_stats = {col: np.nan for col in opponent_rank_columns.values()}

        opponent_data.append(opponent_stats)

    opponent_df = pd.DataFrame(opponent_data)
    player_data = pd.concat([player_data.reset_index(drop=True), opponent_df.reset_index(drop=True)], axis=1)

    return player_data


def extract_opponent_abbreviation(matchup):
    """
    This extracts the opponent team abbreviation from the matchup string.
    """
    if "vs." in matchup:
        return matchup.split("vs.")[-1].strip()
    elif "@" in matchup:
        return matchup.split("@")[-1].strip()
    else:
        print(f"Invalid matchup format: {matchup}")
        return None

def fetch_next_opponent_features(next_game_details, season):
```

```
    """
    This fetches the next opponent's info for feature engineering.
    """
    if 'HOME_TEAM_NAME' in next_game_details and 'VISITOR_TEAM_NAME' in next_game_details:
        if next_game_details['HOME_TEAM_NAME'] == next_game_details.get('TEAM_NAME', ''):
            opponent_team_name = next_game_details['VISITOR_TEAM_NAME']
        else:
            opponent_team_name = next_game_details['HOME_TEAM_NAME']
    else:
        raise KeyError("Required team name columns missing in next_game_details.")

    if opponent_team_name in TEAM_NAME_TO_ID:
        opponent_team_id = TEAM_NAME_TO_ID[opponent_team_name]

        current_features = fetch_team_info(opponent_team_id, season)

    else:
        log(f"Opponent team name {opponent_team_name} not found in mapping.")
        current_features = {}

    return current_features

def combine_features(player_data, opponent_features):
    """
    This combines all the features we created and wanted into model inputs.
    """
    features = pd.DataFrame()

    stats_to_include = ['pts', 'ast', 'reb', 'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a',
                        'ftm', 'fta', 'oreb', 'dreb', 'stl', 'blk', 'tov', 'plus_minus']
    for stat in stats_to_include:
        rolling_col = f'rolling_{stat}'
        if rolling_col in player_data.columns:
            features[rolling_col] = player_data[rolling_col]

    ranking_features = [
        'gp_rank', 'w_rank', 'l_rank', 'opponent_w_pct_rank', 'min_rank',
        'fgm_rank', 'fga_rank', 'fg_pct_rank', 'fg3m_rank', 'fg3a_rank',
        'fg3_pct_rank', 'ftm_rank', 'fta_rank', 'ft_pct_rank', 'oreb_rank',
        'dreb_rank', 'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank',
        'stl_rank', 'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank',
        'opponent_pts_rank', 'opponent_plus_minus_rank'
    ]
    for rank in ranking_features:
        if rank in player_data.columns:
            features[rank] = player_data[rank]

    for key, value in opponent_features.items():
        features[key] = value

    print("\n[Debug] Combined features shape:", features.shape)
    print("[Debug] Combined features columns:", features.columns)

    return features
```

Feature Selection Functions

```
def calculate_vif(X):
    """
    This just calculates VIF for each feature and handles the infinite VIF values we get."""
    vif_data = pd.DataFrame()
    vif_data["Feature"] = X.columns
    vif_values = []
    for i in range(X.shape[1]):
        try:
            vif = variance_inflation_factor(X.values, i)
            vif_values.append(vif)
        except np.linalg.LinAlgError:
            vif_values.append(np.inf)
    vif_data["VIF"] = vif_values
    return vif_data

def remove_high_vif_features(X, threshold=5, keep_features=None):
    """
    This removes the features with high VIF but preserves the specific rolling stats we want.
```

```python
    """
    if keep_features is None:
        keep_features = ["rolling_pts", "rolling_ast", "rolling_reb", "rolling_min"]

    while True:
        vif_data = pd.DataFrame()
        vif_data["Feature"] = X.columns
        vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

        high_vif = vif_data[vif_data["VIF"] > threshold]

        high_vif = high_vif[~high_vif["Feature"].isin(keep_features)]

        if high_vif.empty:
            break

        feature_to_remove = high_vif.loc[high_vif["VIF"].idxmax(), "Feature"]
        print(f"Removing {feature_to_remove} with VIF: {high_vif['VIF'].max():.2f}")
        X = X.drop(columns=[feature_to_remove])

    final_vif = pd.DataFrame()
    final_vif["Feature"] = X.columns
    final_vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return X, final_vif

def remove_high_correlation_features(X, threshold=0.95):
    """
    Does it same but removes features in regards to high correlation; does not preserve rollings.
    """
    corr_matrix = X.corr().abs()

    upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape, dtype=bool), k=1))
    to_drop = [column for column in upper_triangle.columns if any(upper_triangle[column] > threshold)]

    if to_drop:
        print(f"Removing {len(to_drop)} highly correlated features: {to_drop}")
    else:
        print("No highly correlated features found to remove.")

    return X.drop(columns=to_drop), to_drop
```

Dataset Preparation

```python
def prepare_dataset_with_weights(player_id, seasons, stat_col, stat_line, season, next_game_details):
    """
    Big function to basically prepare the dataset for training, including weights and opponent features. We use a lot of the pre
    """

    print("\n[Debug] Starting dataset preparation...")

    player_data = fetch_player_data(player_id, seasons)

    original_features = list(player_data.columns)

    # We weigh the seasons differently as performances typically vary by seasons, especially due to trading to different teams
    season_weights = {seasons[0]: 1.0, seasons[1]: 0.5}
    player_data['seasonweight'] = player_data['season'].map(season_weights)

    player_data = calculate_rolling_averages(player_data)

    player_data = add_opponent_rankings(player_data, season)

    opponent_features = fetch_next_opponent_features(next_game_details, season)

    features = combine_features(player_data, opponent_features)

    combined_features_before_correlation = list(features.columns)

    # Visualize the corr matrix BEFORE feature removal
    plt.figure(figsize=(20, 16))
    correlation_matrix_before = features.corr()

    plt.subplot(1, 2, 1)
    sns.heatmap(
```

```
        correlation_matrix_before,
        cmap="coolwarm",
        center=0,
        annot=False,
        cbar_kws={"shrink": .8},
        square=True
    )
    plt.title("Correlation Matrix BEFORE Feature Removal", fontsize=10)
    plt.tight_layout()

    if stat_col not in player_data.columns:
        raise ValueError(f"Column '{stat_col}' not found in player_data.")

    player_data = player_data.loc[:, ~player_data.columns.duplicated()]

    stat_series = player_data[stat_col]

    features['target'] = (stat_series > stat_line).astype(int)

    features['seasonweight'] = player_data['seasonweight']

    features = features.dropna()

    features_before_removal = features.copy()
    features, dropped_features = remove_high_correlation_features(features, threshold=0.85)

    remaining_features = list(features.columns)

    # Visualize corr matrix AFTER feature removal
    plt.subplot(1, 2, 2)
    correlation_matrix_after = features.corr()
    sns.heatmap(
        correlation_matrix_after,
        cmap="coolwarm",
        center=0,
        annot=False,
        cbar_kws={"shrink": .8},
        square=True
    )
    plt.title("Correlation Matrix AFTER Feature Removal", fontsize=10)
    plt.tight_layout()
    plt.show()

    print("\n--- Feature Analysis ---")
    print(f"Original Features Count: {len(original_features)}")
    print(f"Features Before Correlation Removal: {len(combined_features_before_correlation)}")
    print(f"Features After Correlation Removal: {len(remaining_features)}")

    print("\n--- Dropped Features ---")
    print(dropped_features)

    print("\n--- Remaining Features ---")
    print(remaining_features)

    # More visuals of dropped vs remaining features
    plt.figure(figsize=(10, 6))
    feature_sets = [
        ('Original', set(original_features)),
        ('Before Correlation', set(combined_features_before_correlation)),
        ('After Correlation', set(remaining_features))
    ]

    venn3([s for _, s in feature_sets], set_labels=[name for name, _ in feature_sets])
    plt.title("Feature Set Comparison")
    plt.show()

    # Taking the most recent game for prediction
    next_game_features = features.iloc[-1].drop('target')

    return features, next_game_features
```

Model Training

```
def advanced_model_evaluation(y_true, y_pred, y_pred_proba, model=None, X_test=None, feature_names=None, model_name=None):
    """
```

```
This just does all the calculating and visualizing model performances.
"""
results = {}

print(f"Evaluating {model_name}...")
results['classification_report'] = classification_report(y_true, y_pred)
print(f"Classification report for {model_name}:\n{results['classification_report']}")

results['precision'] = precision_score(y_true, y_pred, average='weighted')
results['recall'] = recall_score(y_true, y_pred, average='weighted')
results['f1'] = f1_score(y_true, y_pred, average='weighted')
print(f"Precision: {results['precision']:.4f}, Recall: {results['recall']:.4f}, F1 Score: {results['f1']:.4f}")

cm = confusion_matrix(y_true, y_pred)
results['confusion_matrix'] = cm
print(f"Confusion Matrix:\n{cm}")

# plotting everything
def plot_roc_curve():
    plt.figure(figsize=(8, 6))
    if y_pred_proba.ndim > 1:
        for i in range(y_pred_proba.shape[1]):
            fpr, tpr, _ = roc_curve(y_true == i, y_pred_proba[:, i])
            auc = roc_auc_score(y_true == i, y_pred_proba[:, i])
            plt.plot(fpr, tpr, label=f'Class {i} (AUC = {auc:.2f})')
    else:
        fpr, tpr, _ = roc_curve(y_true, y_pred_proba)
        auc = roc_auc_score(y_true, y_pred_proba)
        plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve - {model_name}')
    plt.legend()
    plt.show()

def plot_pr_curve():
    plt.figure(figsize=(8, 6))
    precision, recall, _ = precision_recall_curve(y_true, y_pred_proba)
    avg_precision = average_precision_score(y_true, y_pred_proba)
    plt.plot(recall, precision, label=f'AP = {avg_precision:.2f}')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title(f'Precision-Recall Curve - {model_name}')
    plt.legend()
    plt.show()

def plot_feature_importance():
    if feature_names is not None and model is not None:
        try:
            feature_importance = model.feature_importances_
            feature_imp_df = pd.DataFrame({
                'feature': feature_names,
                'importance': feature_importance
            }).sort_values('importance', ascending=False)
            plt.figure(figsize=(10, 6))
            sns.barplot(x='importance', y='feature', data=feature_imp_df.head(10))
            plt.title(f'Top 10 Feature Importances - {model_name}')
            plt.tight_layout()
            plt.show()
        except AttributeError:
            print(f"Feature importance not available for {model_name}.")

if y_pred_proba is not None:
    plot_roc_curve()
    plot_pr_curve()

if feature_names is not None and model is not None:
    plot_feature_importance()

misclassified_indices = np.where(y_true != y_pred)[0]
results['misclassified_samples'] = {
    'indices': misclassified_indices,
    'true_labels': y_true.iloc[misclassified_indices].values if isinstance(y_true, pd.Series) else y_true[misclassified_indi
    'predicted_labels': y_pred[misclassified_indices]
}
```

```
    print(f"Number of Misclassified Samples: {len(misclassified_indices)}")

    return results


def train_and_blend_models(features, next_game_features):
    """
    This is the part of the pipeline that trains all our models.
    """
    X = features.drop(columns=['target'])
    y = features['target']
    sample_weights = features['seasonweight']

    log("Starting feature scaling and preparation.")

    original_features = list(X.columns)

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    log("Feature scaling completed.")

    X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)

    # Visualize VIF BEFORE feature removal
    plt.figure(figsize=(20, 10))

    initial_vif_data = pd.DataFrame()
    initial_vif_data["Feature"] = X_scaled_df.columns
    vif_values = [variance_inflation_factor(X_scaled_df.values, i) for i in range(X_scaled_df.shape[1])]

    # Replace inf values with a large number (999) for better showing
    vif_values = [999 if np.isinf(x) else x for x in vif_values]
    initial_vif_data["VIF"] = vif_values

    initial_vif_sorted = initial_vif_data.sort_values(by="VIF", ascending=False)

    plt.subplot(1, 2, 1)
    sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
    plt.title("Top 20 Features - VIF BEFORE Removal", fontsize=10)
    plt.xlabel("Variance Inflation Factor")
    plt.ylabel("Features")
    plt.tight_layout()

    X_cleaned, dropped_correlation_features = remove_high_correlation_features(X_scaled_df, threshold=0.95)
    log(f"Dropped highly correlated features: {dropped_correlation_features}")

    X_cleaned, final_vif = remove_high_vif_features(X_cleaned, threshold=5)
    log(f"Features retained after VIF reduction: {list(X_cleaned.columns)}")

    # Visualize VIF AFTER feature removal
    plt.subplot(1, 2, 2)
    final_vif_data = pd.DataFrame()
    final_vif_data["Feature"] = X_cleaned.columns
    final_vif_data["VIF"] = [variance_inflation_factor(X_cleaned.values, i) for i in range(X_cleaned.shape[1])]

    final_vif_sorted = final_vif_data.sort_values(by="VIF", ascending=False)

    sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
    plt.title("Top 20 Features - VIF AFTER Removal", fontsize=10)
    plt.xlabel("Variance Inflation Factor")
    plt.ylabel("Features")
    plt.tight_layout()
    plt.show()

    # This is all the print stuff to track all the features and report on changes
    print("\n--- Feature Analysis ---")
    print(f"Original Features Count: {len(original_features)}")
    print(f"Features After Correlation Removal: {len(X_cleaned.columns)}")

    print("\n--- Dropped Features ---")
    print("Correlation Dropped Features:", dropped_correlation_features)

    print("\n--- Remaining Features ---")
    print(list(X_cleaned.columns))

    # Additional visualization of dropped vs remaining features
    plt.figure(figsize=(10, 6))
```

```
feature_sets = [
    ('Original', set(original_features)),
    ('After Correlation', set(X_cleaned.columns))
]

# Visualize correlation matrix AFTER feature removal
plt.subplot(1, 2, 2)
correlation_matrix_after = X_cleaned.corr()
sns.heatmap(
    correlation_matrix_after,
    cmap="coolwarm",
    center=0,
    annot=False,
    cbar_kws={"shrink": .8},
    square=True
)
plt.title("Correlation Matrix AFTER VIF Feature Removal", fontsize=10)
plt.tight_layout()
plt.show()

venn2([s for _, s in feature_sets], set_labels=[name for name, _ in feature_sets])
plt.title("Feature Set Comparison")
plt.show()

scaler_cleaned = StandardScaler()
X_scaled_cleaned = scaler_cleaned.fit_transform(X_cleaned)
X_scaled_cleaned_df = pd.DataFrame(X_scaled_cleaned, columns=X_cleaned.columns)

vif_data = pd.DataFrame()
vif_data["Feature"] = X_scaled_cleaned_df.columns
vif_data["VIF"] = [variance_inflation_factor(X_scaled_cleaned_df.values, i) for i in range(X_scaled_cleaned_df.shape[1])]
log("Calculated VIF data.")

log("Initial VIF Data:")
log(vif_data.sort_values(by="VIF", ascending=False).to_string(), level="DEBUG")

log("Preparing next_game_features for scaling and prediction.")
next_game_features_df = pd.DataFrame([next_game_features], columns=X.columns)
next_game_features_cleaned = next_game_features_df[X_cleaned.columns]

missing_cols = set(X_cleaned.columns) - set(next_game_features_cleaned.columns)
if missing_cols:
    log(f"Missing columns in next_game_features: {missing_cols}", level="ERROR")
    raise ValueError("next_game_features is missing required columns!")

try:
    next_game_scaled = scaler_cleaned.transform(next_game_features_cleaned)
    log("Successfully scaled next_game_features.")
except Exception as e:
    log(f"Error during scaling next_game_features: {e}", level="ERROR")
    raise

if next_game_scaled.shape[1] != X_scaled_cleaned.shape[1]:
    log("Mismatch in feature dimensions after scaling.", level="ERROR")
    raise ValueError(f"Feature mismatch! next_game_scaled has {next_game_scaled.shape[1]} features, "
                     f"but expected {X_scaled_cleaned.shape[1]} features.")

log("Splitting data into training and validation sets.")
X_train, X_val, y_train, y_val, sw_train, sw_val = train_test_split(
    X_scaled_cleaned, y, sample_weights, test_size=0.3, random_state=42
)

log(f"Training set shape: {X_train.shape}, Validation set shape: {X_val.shape}")

# Naive Model (Classification)
log("Training Naive Model.")
majority_class = y_train.mode()[0]
naive_predictions = [majority_class] * len(y_val)

# Logistic Regression
log("Training Logistic Regression model.")
log_model = LogisticRegression(class_weight='balanced', random_state=42)
log_model.fit(X_train, y_train, sample_weight=sw_train)

# Random Forest with K-Fold Cross-Validation
log("Training Random Forest model with cross-validation.")
rf_model = RandomForestClassifier(class_weight='balanced', random_state=42)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, scoring='accuracy', cv=kf)
rf_model.fit(X_train, y_train, sample_weight=sw_train)

log("Performing hyperparameter optimization for Random Forest.")
rf_params = {'n_estimators': [50, 100, 200], 'max_features': ['sqrt', 'log2'], 'max_depth': [None, 10, 20]}
rf_grid = GridSearchCV(rf_model, rf_params, scoring='roc_auc', cv=5, verbose=1)
rf_grid.fit(X_train, y_train, sample_weight=sw_train)
rf_best_model = rf_grid.best_estimator_

log("Random Forest training and optimization completed.")

# Gradient Boosting
log("Training Gradient Boosting model.")
gb_model = GradientBoostingClassifier(random_state=42)
gb_model.fit(X_train, y_train)

# Neural Network
log("Training Neural Network model with randomized hyperparameter search.")
nn_model = MLPClassifier(random_state=42)
nn_params = {
    'hidden_layer_sizes': [(32,), (32, 16), (64, 32)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'learning_rate': ['constant', 'adaptive'],
    'learning_rate_init': [0.001, 0.01],
    'max_iter': [500, 1000],
    'batch_size': [4],
    'early_stopping': [True],
    'validation_fraction': [0.1]
}

kf = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
nn_search = RandomizedSearchCV(nn_model, nn_params, n_iter=10, cv=kf, scoring='accuracy', random_state=42, verbose=1)

try:
    nn_search.fit(X_train, y_train)
    best_nn_model = nn_search.best_estimator_
    log("Neural Network training and optimization completed.")
except ValueError as e:
    log(f"Neural Network training failed: {e}", level="ERROR")
    best_nn_model = LogisticRegression()
    best_nn_model.fit(X_train, y_train)
    log(f"Neural Network replaced with Logistic Regression Model", level="ERROR")

log("Neural Network training and optimization completed.")

# Weighted Voting Classifier
log("Training Weighted Voting Classifier.")
weighted_ensemble = VotingClassifier(
    estimators=[
        ('lr', log_model),
        ('rf', rf_best_model),
        ('gb', gb_model),
        ('nn', best_nn_model)
    ],
    voting='soft',
    weights=[1, 2, 2, 1]
)
weighted_ensemble.fit(X_train, y_train)

meta_classifier = LogisticRegression()

# Stacking Classifier
stacking_ensemble = StackingClassifier(
    estimators=[
        ('lr', log_model),
        ('rf', rf_best_model),
        ('gb', gb_model),
        ('nn', best_nn_model)
    ],
    final_estimator=meta_classifier,
    cv=5,
    stack_method='predict_proba'
)

stacking_ensemble.fit(X_train, y_train)
```

```
y_pred_stacking = stacking_ensemble.predict(X_val)

print("Stacking Classifier Performance:")
print(classification_report(y_val, y_pred_stacking))

print("\nVoting Classifier Performance:")
weighted_ensemble.fit(X_train, y_train)
y_pred_voting = weighted_ensemble.predict(X_val)
print(classification_report(y_val, y_pred_voting))

evaluation_results = {}

# Naive Model Evaluation
log("Evaluating Naive Model.")

majority_class = y_train.mode()[0]
naive_pred = [majority_class] * len(y_val)

naive_pred_proba = [majority_class] * len(y_val)

naive_accuracy = accuracy_score(y_val, naive_pred)
naive_classification_report = classification_report(y_val, naive_pred, output_dict=True)
naive_confusion_matrix = confusion_matrix(y_val, naive_pred)

log(f"Naive Model Accuracy: {naive_accuracy:.4f}")
log("Naive Model Classification Report:")
log(naive_classification_report)
log("Naive Model Confusion Matrix:")
log(naive_confusion_matrix)

evaluation_results['naive_model'] = {
    'accuracy': naive_accuracy,
    'classification_report': naive_classification_report,
    'confusion_matrix': naive_confusion_matrix.tolist()
}

log("Naive Model evaluation completed.")


# Logistic Regression Evaluation
log_pred = log_model.predict(X_val)
log_pred_proba = log_model.predict_proba(X_val)[:, 1]
evaluation_results['logistic_regression'] = advanced_model_evaluation(
    y_val,
    log_pred,
    log_pred_proba,
    log_model,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Logistic Regression"
)

log("Logistic Regression evaluation completed.\n\n")

# Random Forest Evaluation
rf_pred = rf_best_model.predict(X_val)
rf_pred_proba = rf_best_model.predict_proba(X_val)[:, 1]
evaluation_results['random_forest'] = advanced_model_evaluation(
    y_val,
    rf_pred,
    rf_pred_proba,
    rf_best_model,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Random Forest"
)

log("Random Forest evaluation completed.\n\n")

# Gradient Boosting Evaluation
gb_pred = gb_model.predict(X_val)
gb_pred_proba = gb_model.predict_proba(X_val)[:, 1]
evaluation_results['gradient_boosting'] = advanced_model_evaluation(
    y_val,
    gb_pred,
    gb_pred_proba,
```

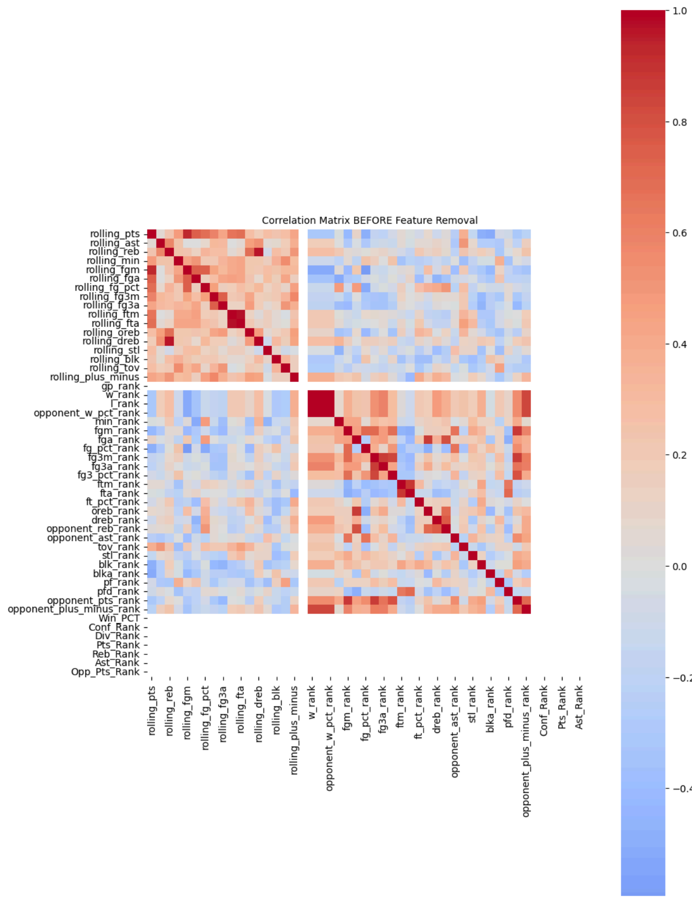
```
    gb_model,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Gradient Boosting"
)

log("Gradient Boosting evaluation completed.\n\n")

# Neural Network Evaluation
nn_pred = best_nn_model.predict(X_val)
nn_pred_proba = best_nn_model.predict_proba(X_val)[:, 1]
evaluation_results['neural_network'] = advanced_model_evaluation(
    y_val,
    nn_pred,
    nn_pred_proba,
    best_nn_model,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Neural Network"
)

log("Neural Network evaluation completed.\n\n")

# Voting Ensemble Evaluation
ensemble_pred = weighted_ensemble.predict(X_val)
ensemble_pred_proba = weighted_ensemble.predict_proba(X_val)[:, 1]
evaluation_results['weighted_ensemble'] = advanced_model_evaluation(
    y_val,
    ensemble_pred,
    ensemble_pred_proba,
    weighted_ensemble,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Voting Classifier Ensemble"
)

log("Voting Classifier Ensemble evaluation completed.\n\n")

# Stacking Classifier Ensemble Evaluation
stacking_pred = stacking_ensemble.predict(X_val)
stacking_pred_proba = stacking_ensemble.predict_proba(X_val)[:, 1]
evaluation_results['stacking_ensemble'] = advanced_model_evaluation(
    y_val,
    stacking_pred,
    stacking_pred_proba,
    stacking_ensemble,
    X_val,
    feature_names=X_cleaned.columns,
    model_name="Stacking Classifier Ensemble"
)

log("Stacking Classifier Ensemble evaluation completed.\n\n")

blended_probs = weighted_ensemble.predict_proba(next_game_scaled)[0]
blended_prediction = 1 if blended_probs[1] > blended_probs[0] else 0

print("\nBlended Prediction for Next Game:")
print(f"Prediction: {'Over' if blended_prediction == 1 else 'Under'}")
print(f"Probability of Under: {blended_probs[0]:.2f}")
print(f"Probability of Over: {blended_probs[1]:.2f}")

return_dict = {
    'models': {
        'Logistic Regression': log_model,
        'Random Forest': rf_best_model,
        'Gradient Boosting': gb_model,
        'Neural Network': best_nn_model,
        'Weighted Ensemble': weighted_ensemble,
        'Stacking Ensemble': stacking_ensemble
    },
    'feature_importances': pd.DataFrame({
        'Feature': X_cleaned.columns,
        'Importance': rf_best_model.feature_importances_
    }).sort_values(by='Importance', ascending=False),
    'vif': vif_data,
    'model_evaluations': evaluation_results,
    'next_game_prediction': {
```

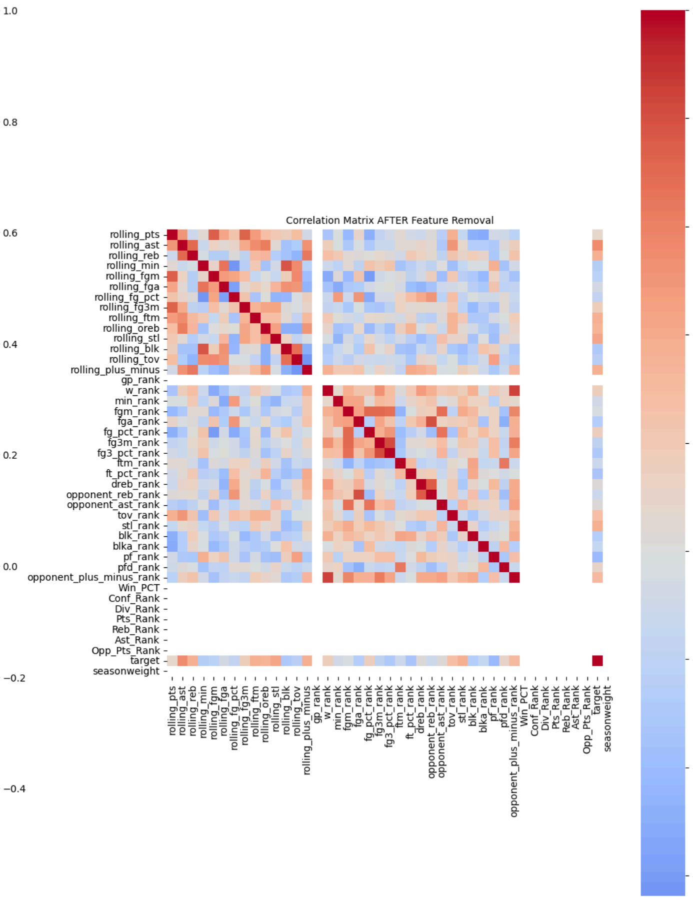
```
            'prediction': 'Over' if blended_prediction == 1 else 'Under',
            'probabilities': {
                'Under': blended_probs[0],
                'Over': blended_probs[1]
            }
        }
    }

    return return_dict
```

Main Predictor Execution

```
players_stats = [
    {"name": "Cade Cunningham", "stat_col": "ast", "stat_line": 9.5},
    # {"name": "Jaylen Brown", "stat_col": "ast", "stat_line": 4.5},
    # {"name": "Kristaps Porzingis", "stat_col": "reb", "stat_line": 7.5},
    {"name": "Tyler Herro", "stat_col": "fg3m", "stat_line": 3.5},
    # {"name": "Jimmy Butler", "stat_col": "pts", "stat_line": 19.5},
    # {"name": "Tyrese Haliburton", "stat_col": "pts", "stat_line": 16.5},
    # {"name": "Benedict Mathurin", "stat_col": "reb", "stat_line": 5.5},
    {"name": "Giannis Antetokounmpo", "stat_col": "pts", "stat_line": 32.5},
    {"name": "Trae Young", "stat_col": "ast", "stat_line": 11.5},
    # {"name": "Shai Gilgeous-Alexander", "stat_col": "pts", "stat_line": 30.5},
    {"name": "Alperen Sengun", "stat_col": "ast", "stat_line": 4.5},
]

for player in players_stats:
    print(f"\n--- Processing {player['name']} ---")

    player_id = fetch_player_id(player["name"])
    seasons = ['2024-25', '2023-24']
    next_game_details = fetch_next_game_details(player_id, seasons[0])

    features, next_game_features = prepare_dataset_with_weights(
        player_id, ["2024-25", "2023-24"], player["stat_col"], player["stat_line"], "2024-25", next_game_details
    )

    models = train_and_blend_models(features, next_game_features)

    blended_probs = models['next_game_prediction']['probabilities']
    blended_prediction = models['next_game_prediction']['prediction']

    print(f"Prediction for {player['stat_col']} (stat line {player['stat_line']}): {blended_prediction}")
    print(f"Probability of Under: {blended_probs['Under']:.2f}")
    print(f"Probability of Over: {blended_probs['Over']:.2f}")
```

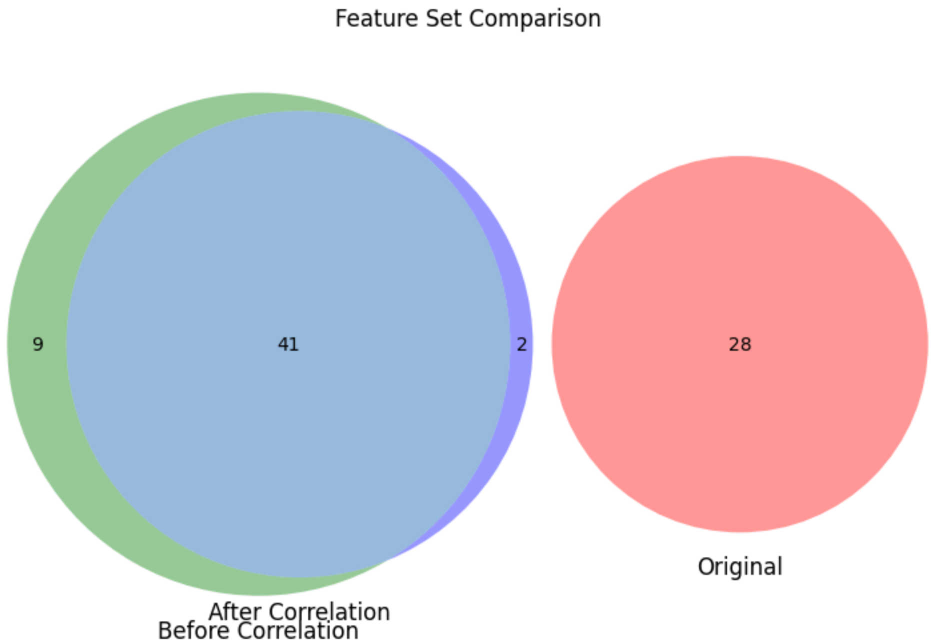
```
--- Processing Cade Cunningham ---
1630595

Next Game Details:
GAME_DATE            DEC 12, 2024
HOME_TEAM_NAME             Boston
VISITOR_TEAM_NAME         Detroit
GAME_TIME                07:30 PM
Name: 0, dtype: object

[Debug] Starting dataset preparation...
Columns in fetched player game log for season 2024-25: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
Columns in fetched player game log for season 2023-24: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')

Fetched player data shape: (83, 28)

Rolling averages calculated successfully.
Fetched team game logs for season 2024-25, shape: (726, 57)

[Debug] Combined features shape: (83, 50)
[Debug] Combined features columns: Index(['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min',
       'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
       'rolling_fg3a', 'rolling_ftm', 'rolling_fta', 'rolling_oreb',
       'rolling_dreb', 'rolling_stl', 'rolling_blk', 'rolling_tov',
       'rolling_plus_minus', 'gp_rank', 'w_rank', 'l_rank',
       'opponent_w_pct_rank', 'min_rank', 'fgm_rank', 'fga_rank',
       'fg_pct_rank', 'fg3m_rank', 'fg3a_rank', 'fg3_pct_rank', 'ftm_rank',
       'fta_rank', 'ft_pct_rank', 'oreb_rank', 'dreb_rank',
       'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank', 'stl_rank',
       'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank', 'opponent_pts_rank',
       'opponent_plus_minus_rank', 'Win_PCT', 'Conf_Rank', 'Div_Rank',
       'Pts_Rank', 'Reb_Rank', 'Ast_Rank', 'Opp_Pts_Rank'],
      dtype='object')
Removing 9 highly correlated features: ['rolling_fg3a', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fg3
```



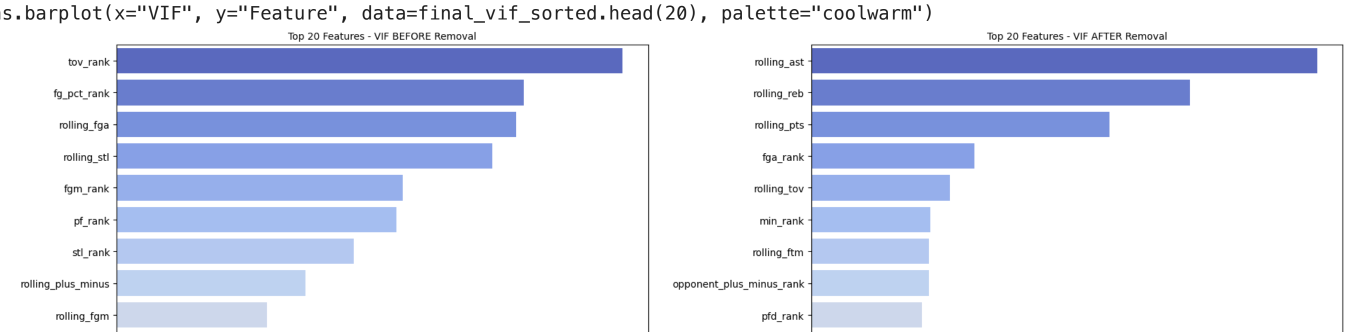


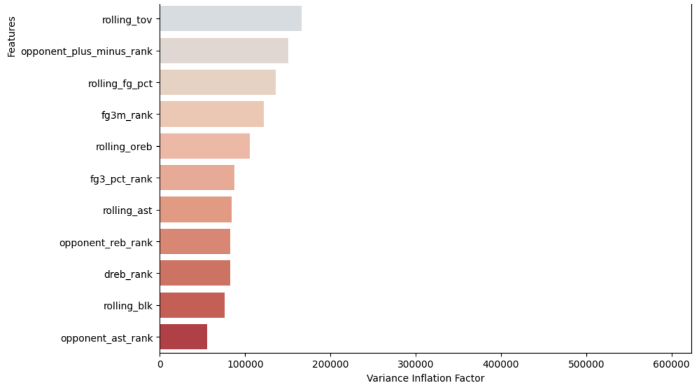

```
--- Feature Analysis ---
Original Features Count: 28
```

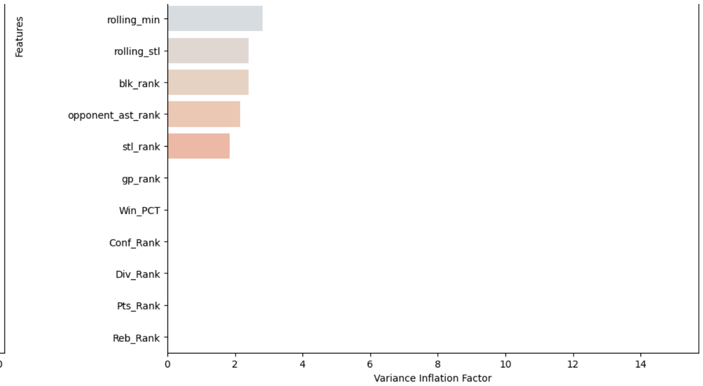
```
Original Features Count: 28
Features Before Correlation Removal: 50
Features After Correlation Removal: 43

--- Dropped Features ---
['rolling_fg3a', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fg3a_rank', 'fta_rank', 'oreb_rank', 'oppo

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
```



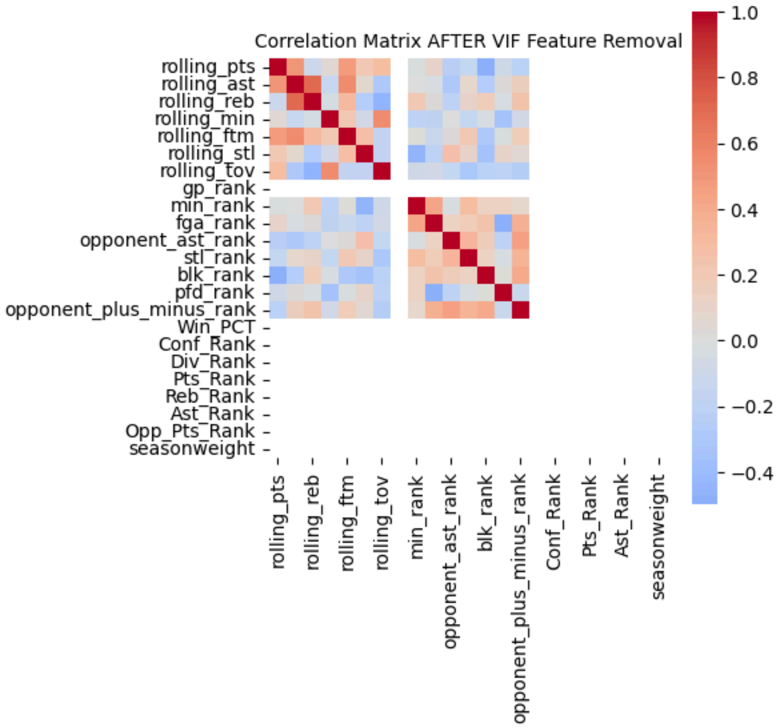

```
[INFO]: Starting feature scaling and preparation.
[INFO]: Feature scaling completed.
<ipython-input-111-9e74f1c2681d>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
No highly correlated features found to remove.
[INFO]: Dropped highly correlated features: []
Removing tov_rank with VIF: 593260.16
Removing fg3m_rank with VIF: 330910.41
Removing opponent_reb_rank with VIF: 229818.36
Removing dreb_rank with VIF: 217038.00
Removing rolling_fgm with VIF: 272474.28
Removing pf_rank with VIF: 155039.68
Removing fg3_pct_rank with VIF: 315405.79
Removing fg_pct_rank with VIF: 246115.04
Removing ft_pct_rank with VIF: 452349.57
Removing rolling_plus_minus with VIF: inf
Removing rolling_fga with VIF: inf
Removing rolling_fg_pct with VIF: inf
Removing rolling_fg3m with VIF: inf
Removing ftm_rank with VIF: 131.76
Removing fgm_rank with VIF: 50.94
Removing w_rank with VIF: 26.30
Removing rolling_blk with VIF: 16.92
Removing rolling_oreb with VIF: 8.67
Removing blka_rank with VIF: 6.40
[INFO]: Features retained after VIF reduction: ['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_ftm', '
<ipython-input-111-9e74f1c2681d>:53: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
```

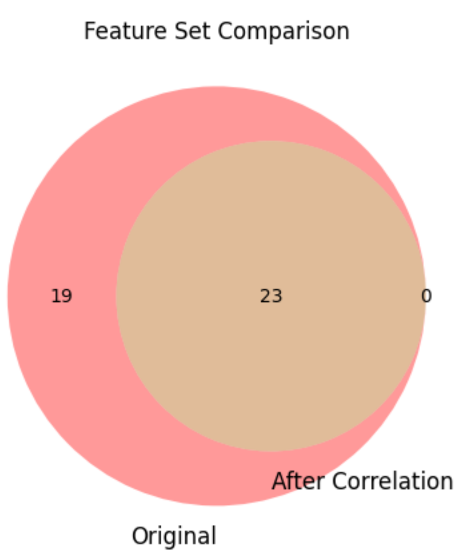







```
--- Feature Analysis ---
Original Features Count: 42
Features After Correlation Removal: 23

--- Dropped Features ---
Correlation Dropped Features: []

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_ftm', 'rolling_stl', 'rolling_tov', 'gp_rank', 'min_ra
```

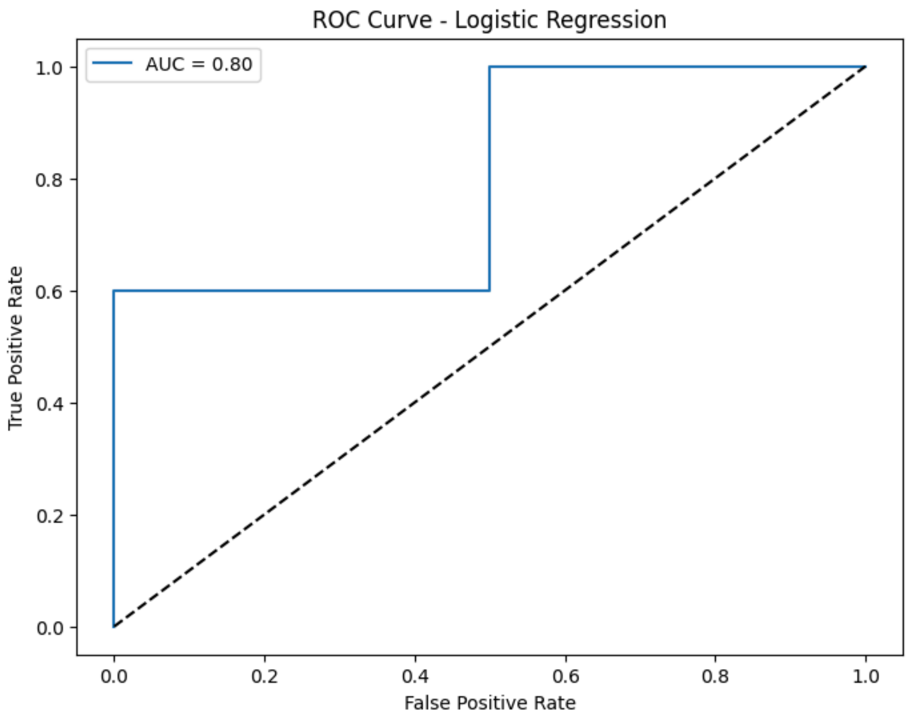


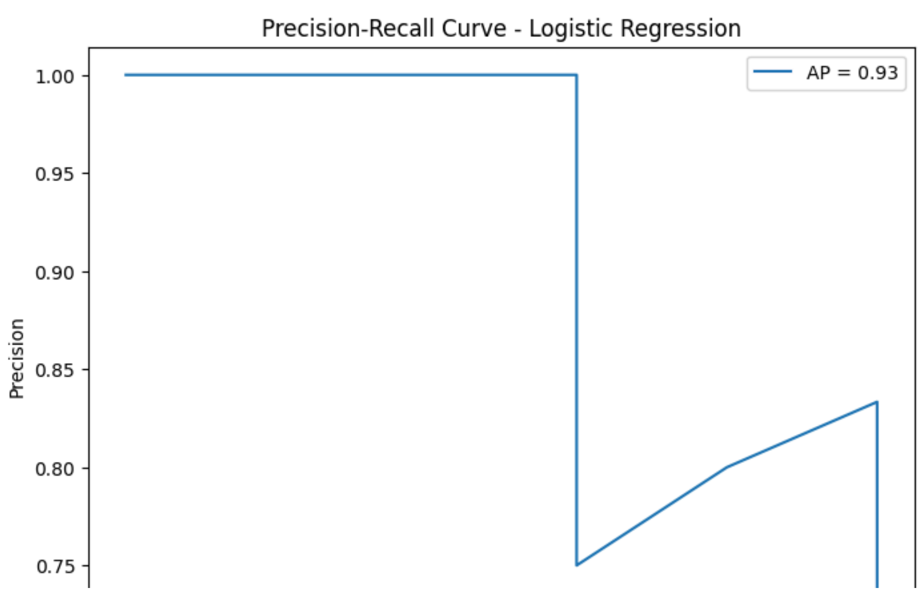


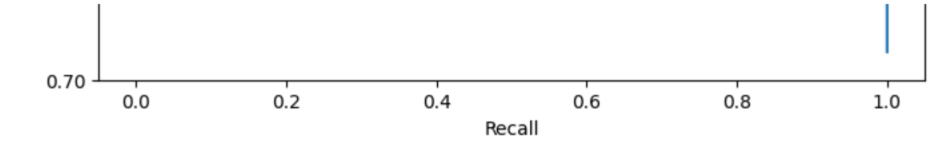

```
[INFO]: Calculated VIF data.
[INFO]: Initial VIF Data:
```

```
[DEBUG]:                      Feature        VIF
1               rolling_ast  14.964269
2               rolling_reb  11.207803
0               rolling_pts   8.810674
9                  fga_rank   4.834826
6               rolling_tov   4.101318
8                  min_rank   3.530729
4               rolling_ftm   3.481217
14  opponent_plus_minus_rank   3.480755
13                 pfd_rank   3.270190
3               rolling_min   2.807516
5               rolling_stl   2.412184
12                 blk_rank   2.406297
10        opponent_ast_rank   2.146462
11                 stl_rank   1.842553
7                   gp_rank        NaN
15                  Win_PCT        NaN
16                Conf_Rank        NaN
17                 Div_Rank        NaN
18                 Pts_Rank        NaN
19                 Reb_Rank        NaN
20                 Ast_Rank        NaN
21             Opp_Pts_Rank        NaN
22             seasonweight        NaN
[INFO]: Preparing next_game_features for scaling and prediction.
[INFO]: Successfully scaled next_game_features.
[INFO]: Splitting data into training and validation sets.
[INFO]: Training set shape: (14, 23), Validation set shape: (7, 23)
[INFO]: Training Naive Model.
[INFO]: Training Logistic Regression model.
[INFO]: Training Random Forest model with cross-validation.
[INFO]: Performing hyperparameter optimization for Random Forest.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[INFO]: Random Forest training and optimization completed.
[INFO]: Training Gradient Boosting model.
[INFO]: Training Neural Network model with randomized hyperparameter search.
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[ERROR]: Neural Network training failed:
All the 30 fits failed.
It is very likely that your model is misconfigured.
You can try to debug the error by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 751, in fit
    return self._fit(X, y, incremental=False)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 475, in _fit
    self._fit_stochastic(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 588, in _fit_stochas
    X, X_val, y, y_val = train_test_split(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 186, in wrapper
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2806, in train_test_split
    train, test = next(cv.split(X=arrays[0], y=stratify))
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 1843, in split
    for train, test in self._iter_indices(X, y, groups):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2265, in _iter_indices
    raise ValueError(
ValueError: The test_size = 1 should be greater or equal to the number of classes = 2

[ERROR]: Neural Network replaced with Logistic Regression Model
[INFO]: Neural Network training and optimization completed.
[INFO]: Training Weighted Voting Classifier.
Stacking Classifier Performance:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7


Voting Classifier Performance:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5
```

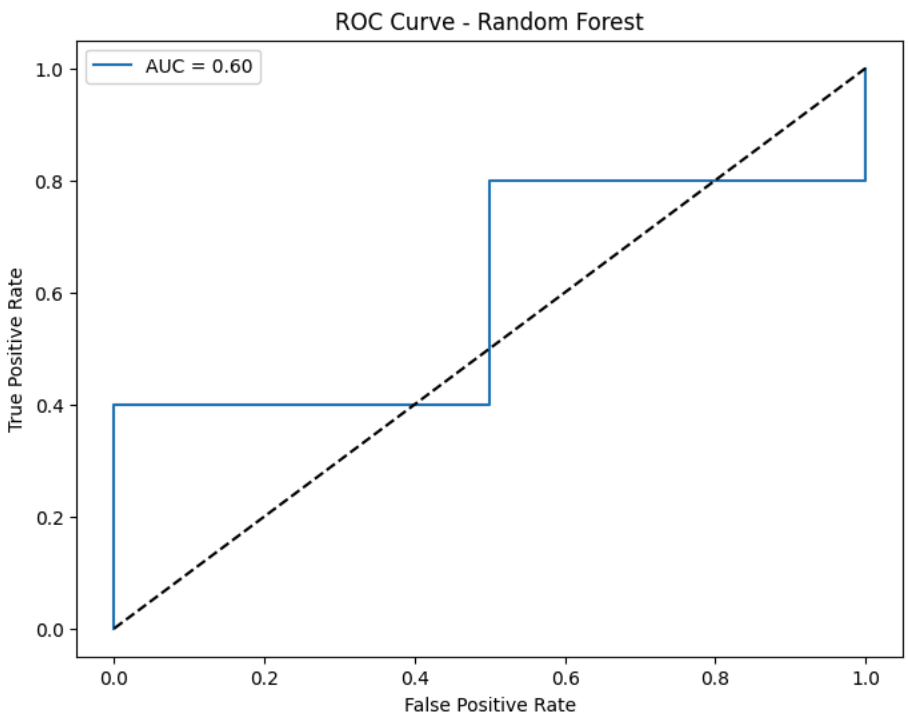
```
    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7

[INFO]: Evaluating Naive Model.
[INFO]: Naive Model Accuracy: 0.2857
[INFO]: Naive Model Classification Report:
[INFO]: {'0': {'precision': 0.2857142857142857, 'recall': 1.0, 'f1-score': 0.4444444444444444, 'support': 2.0}, '1': {'preci
[INFO]: Naive Model Confusion Matrix:
[INFO]: [[2 0]
 [5 0]]
[INFO]: Naive Model evaluation completed.
Evaluating Logistic Regression...
Classification report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7

Precision: 0.6310, Recall: 0.5714, F1 Score: 0.5905
Confusion Matrix:
[[1 1]
 [2 3]]
```

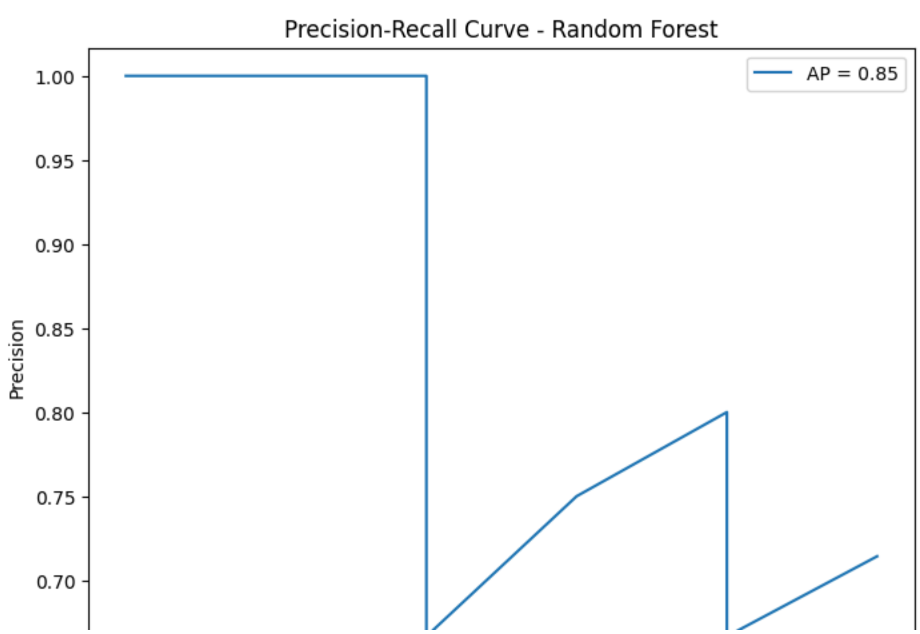


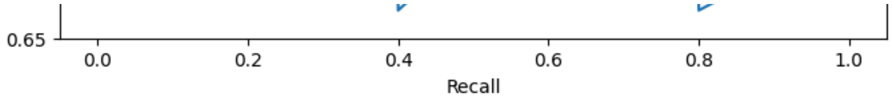


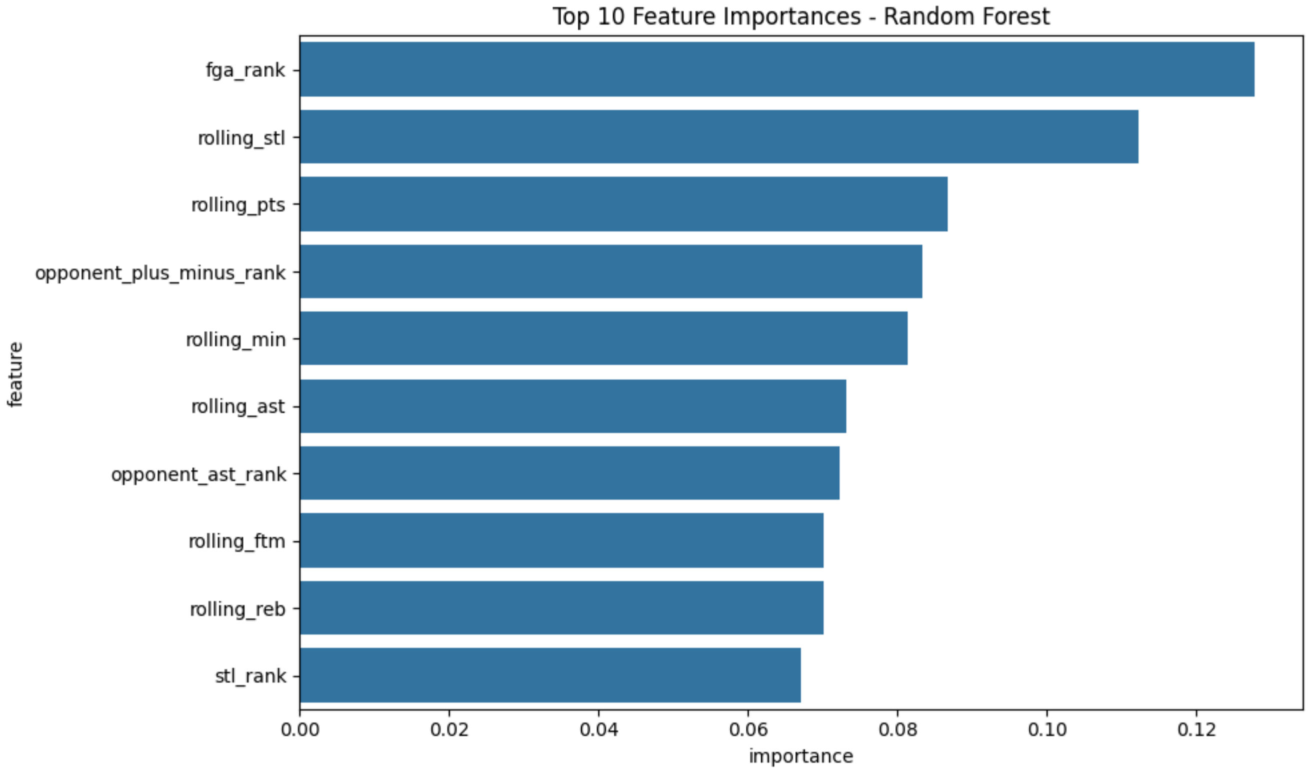



```
Feature importance not available for Logistic Regression.
Number of Misclassified Samples: 3
[INFO]: Logistic Regression evaluation completed.


Evaluating Random Forest...
Classification report for Random Forest:
              precision    recall  f1-score   support

           0       0.40      1.00      0.57         2
           1       1.00      0.40      0.57         5

    accuracy                           0.57         7
   macro avg       0.70      0.70      0.57         7
weighted avg       0.83      0.57      0.57         7

Precision: 0.8286, Recall: 0.5714, F1 Score: 0.5714
Confusion Matrix:
[[2 0]
 [3 2]]
```

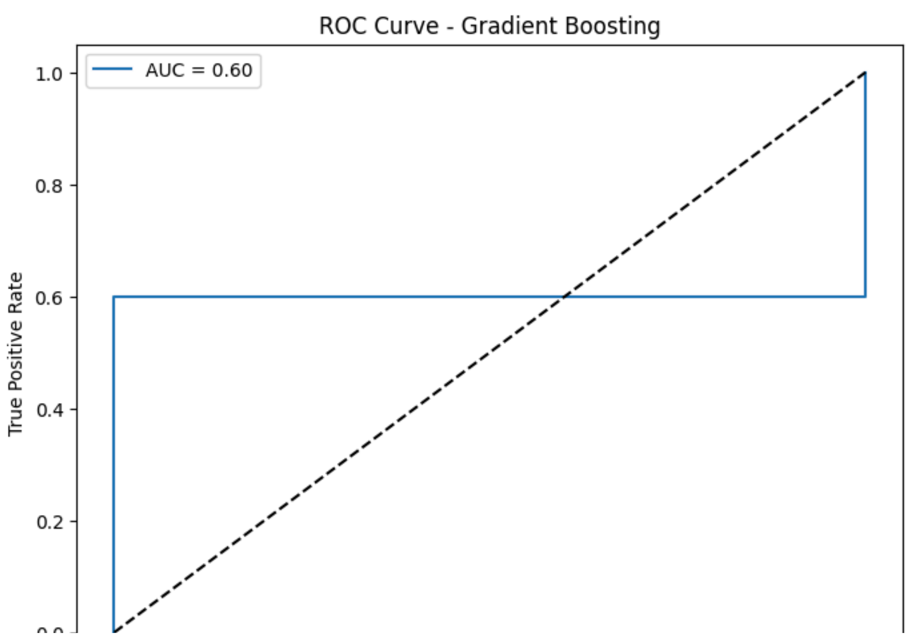


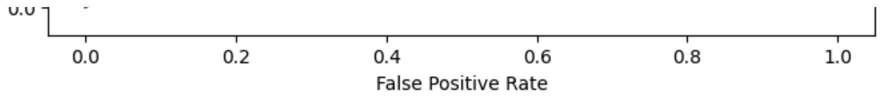


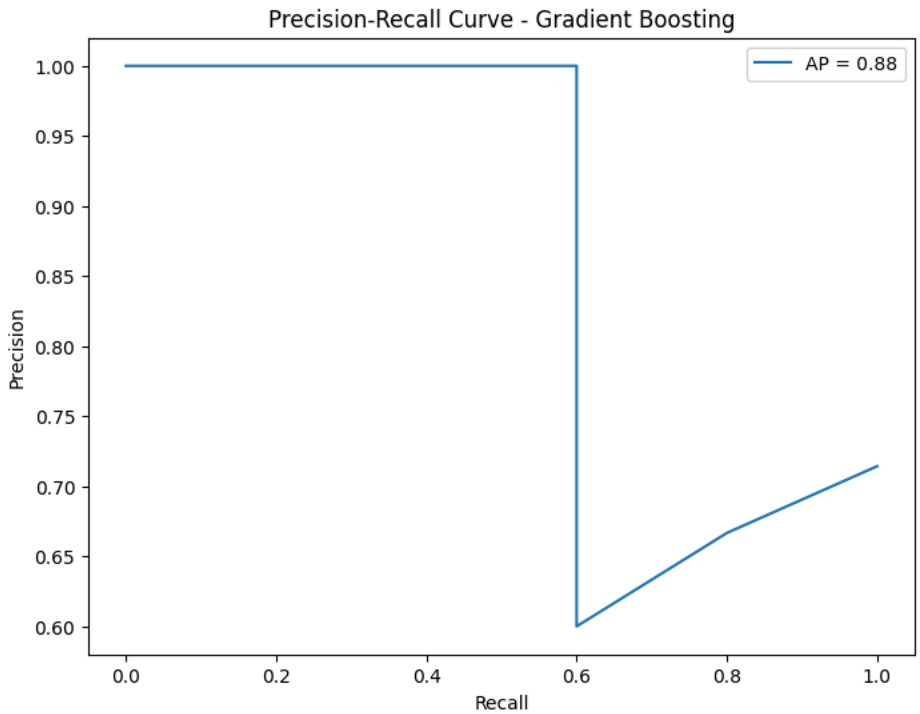


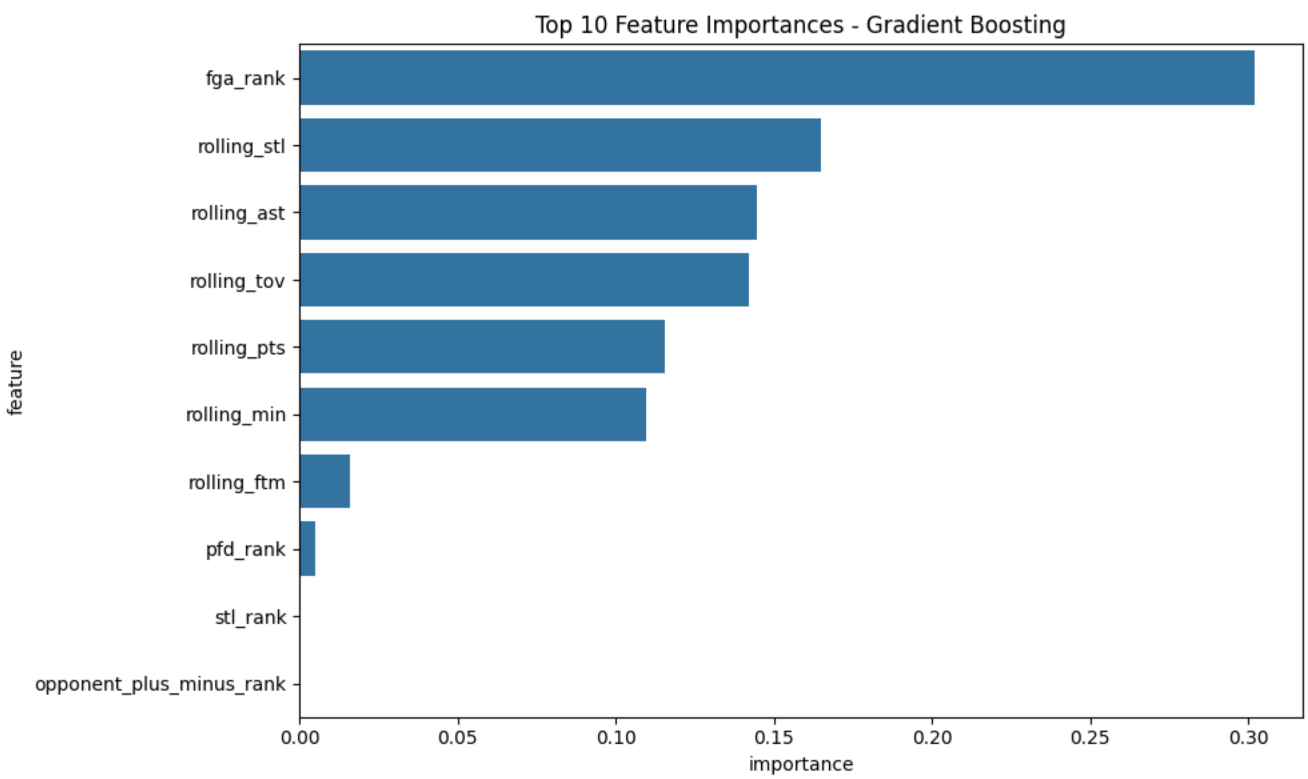



```
Number of Misclassified Samples: 3
[INFO]: Random Forest evaluation completed.


Evaluating Gradient Boosting...
Classification report for Gradient Boosting:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         2
           1       0.60      0.60      0.60         5

    accuracy                           0.43         7
   macro avg       0.30      0.30      0.30         7
weighted avg       0.43      0.43      0.43         7

Precision: 0.4286, Recall: 0.4286, F1 Score: 0.4286
Confusion Matrix:
[[0 2]
 [2 3]]
```



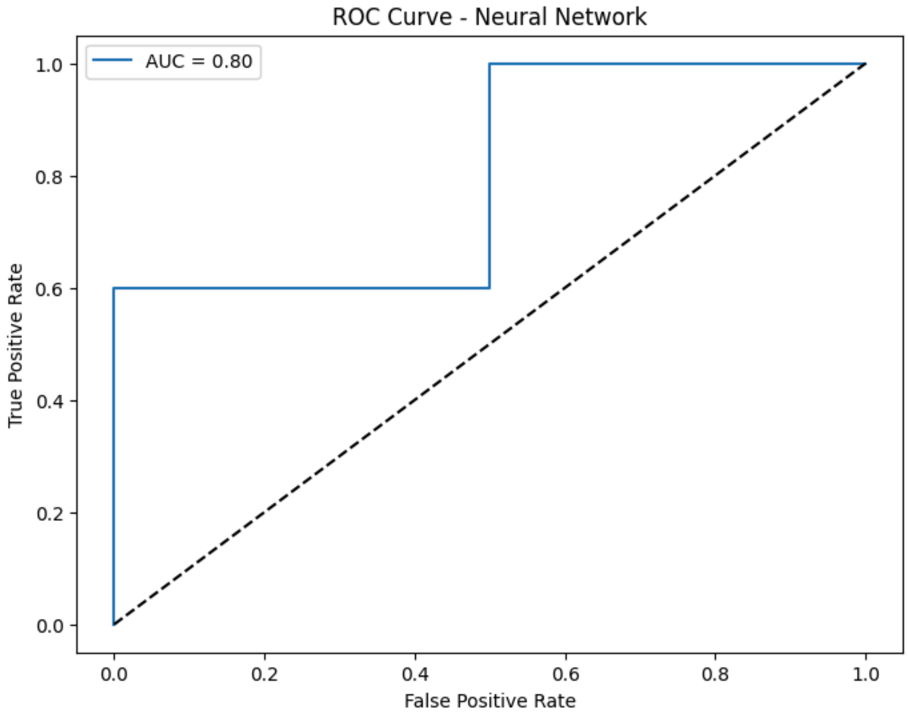


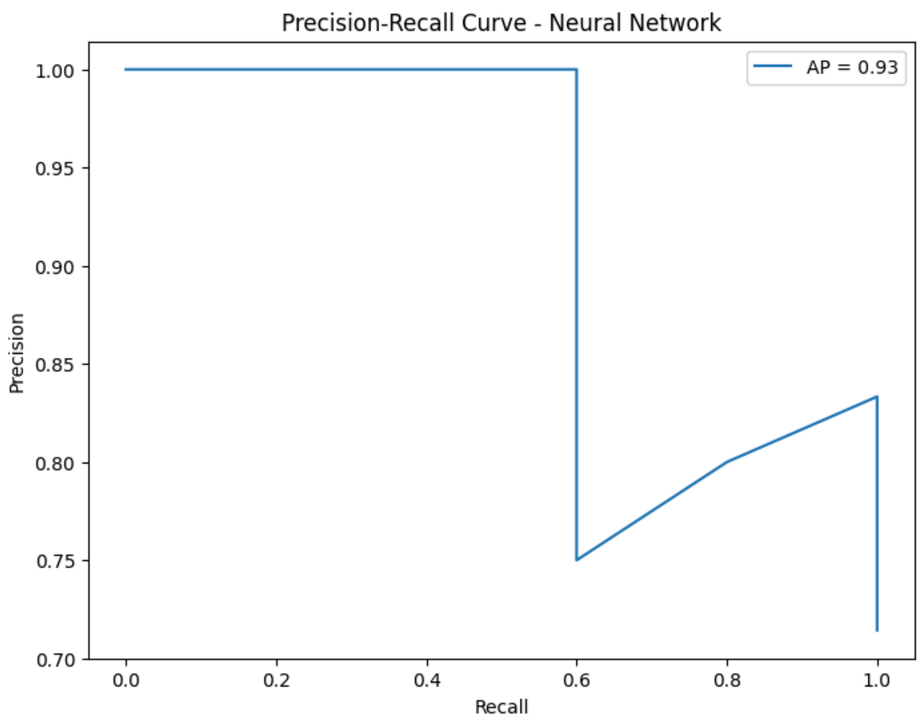





```
Number of Misclassified Samples: 4
[INFO]: Gradient Boosting evaluation completed.


Evaluating Neural Network...
Classification report for Neural Network:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7
```

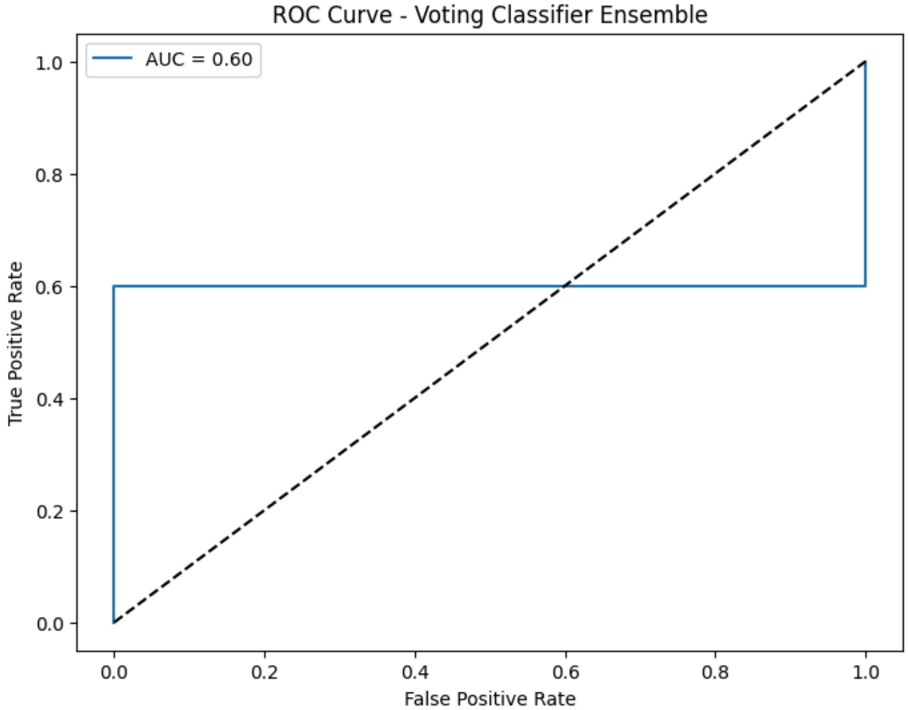
```
Precision: 0.6310, Recall: 0.5714, F1 Score: 0.5905
Confusion Matrix:
[[1 1]
 [2 3]]
```

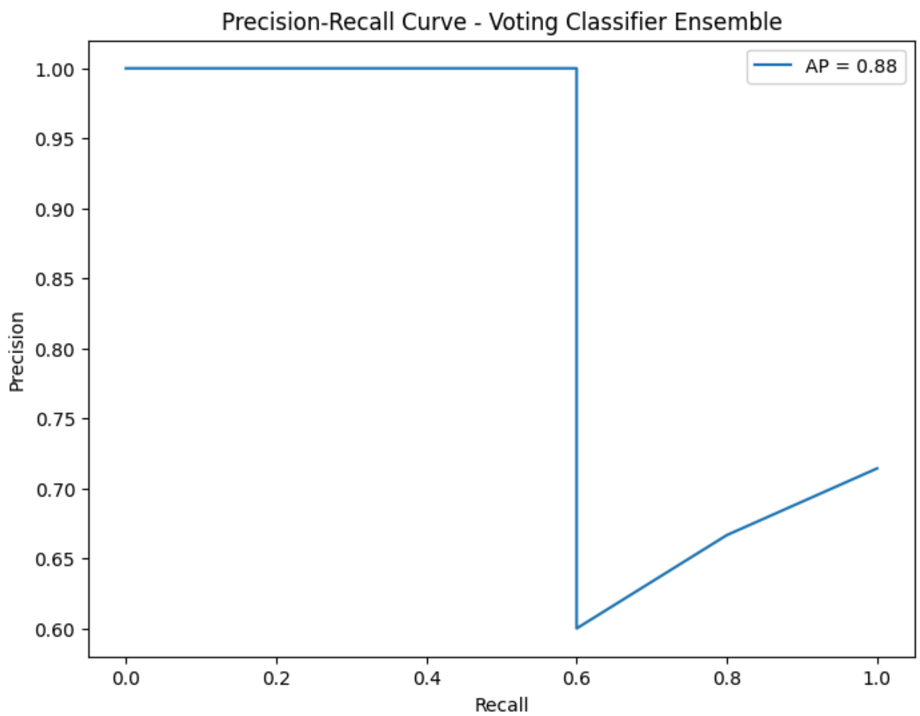





```
Feature importance not available for Neural Network.
Number of Misclassified Samples: 3
[INFO]: Neural Network evaluation completed.


Evaluating Voting Classifier Ensemble...
Classification report for Voting Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7

Precision: 0.6310, Recall: 0.5714, F1 Score: 0.5905
Confusion Matrix:
```

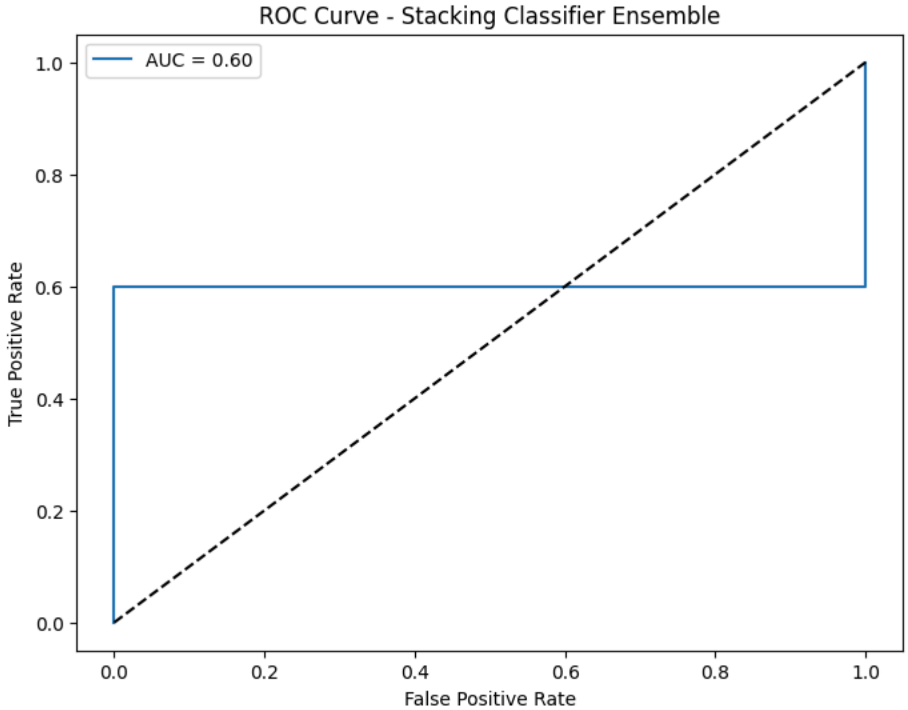
```
[[1 1]
 [2 3]]
```

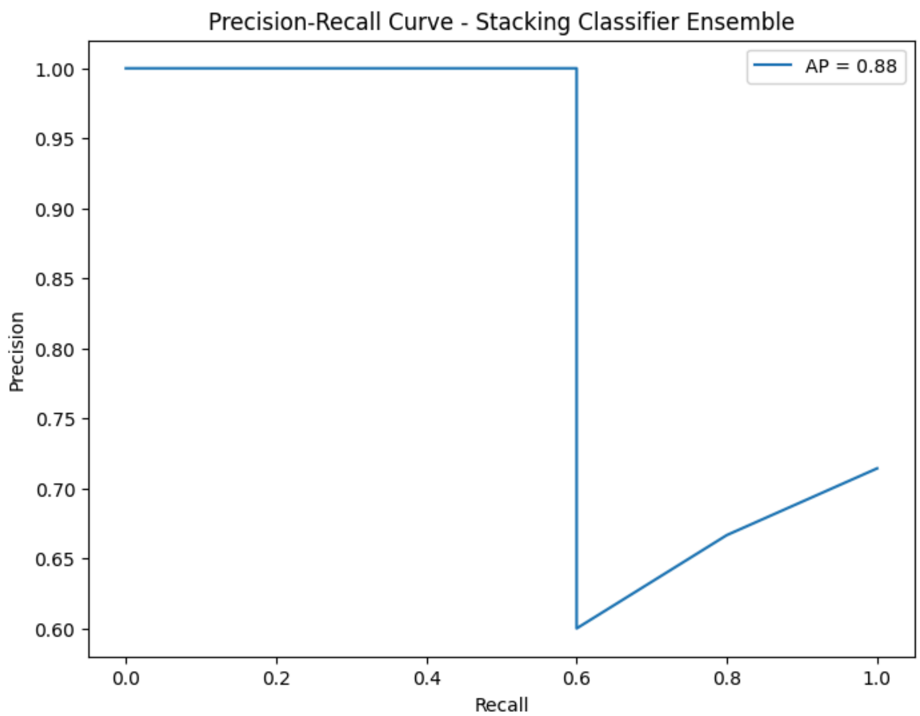





```
Feature importance not available for Voting Classifier Ensemble.
Number of Misclassified Samples: 3
[INFO]: Voting Classifier Ensemble evaluation completed.


Evaluating Stacking Classifier Ensemble...
Classification report for Stacking Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.33      0.50      0.40         2
           1       0.75      0.60      0.67         5

    accuracy                           0.57         7
   macro avg       0.54      0.55      0.53         7
weighted avg       0.63      0.57      0.59         7

Precision: 0.6310, Recall: 0.5714, F1 Score: 0.5905
Confusion Matrix:
[[1 1]
 [2 3]]
```

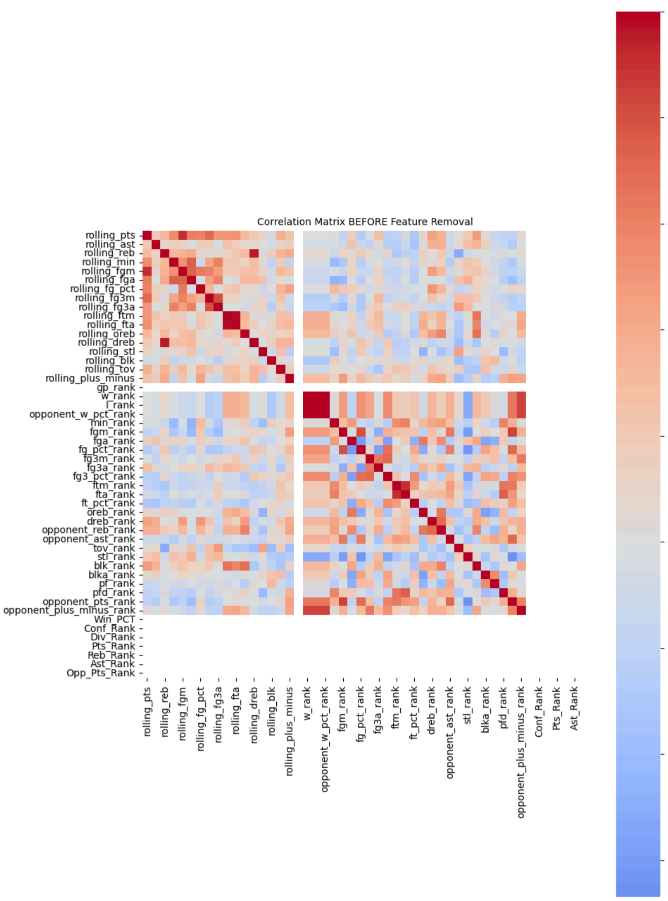
[2 3]]

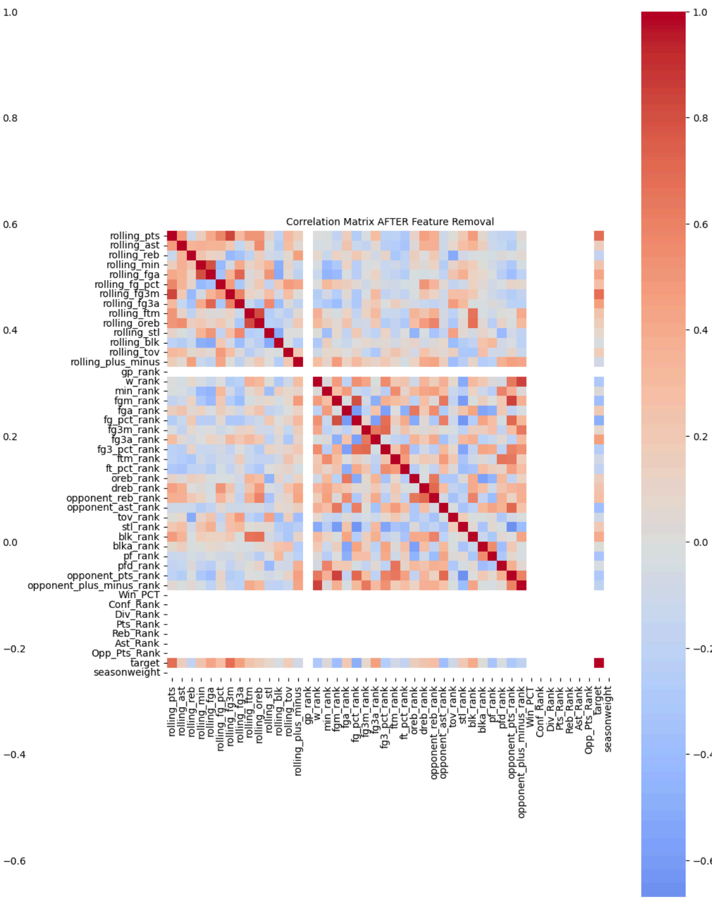




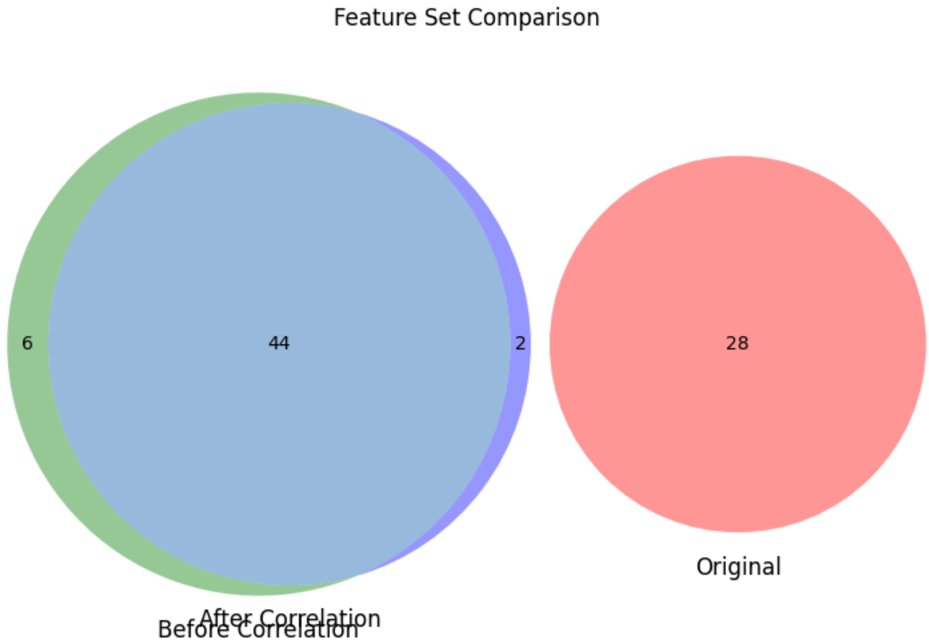

```
Feature importance not available for Stacking Classifier Ensemble.
Number of Misclassified Samples: 3
[INFO]: Stacking Classifier Ensemble evaluation completed.


Blended Prediction for Next Game:
Prediction: Under
Probability of Under: 0.80
Probability of Over: 0.20
Prediction for ast (stat line 9.5): Under
Probability of Under: 0.80
Probability of Over: 0.20

--- Processing Tyler Herro ---
1629639

Next Game Details:
GAME_DATE           DEC 12, 2024
HOME_TEAM_NAME             Miami
VISITOR_TEAM_NAME        Toronto
GAME_TIME               07:30 PM
```

```
GAME_TIME               07:30 PM
Name: 0, dtype: object

[Debug] Starting dataset preparation...
Columns in fetched player game log for season 2024-25: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
Columns in fetched player game log for season 2023-24: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')

Fetched player data shape: (64, 28)

Rolling averages calculated successfully.
Fetched team game logs for season 2024-25, shape: (726, 57)

[Debug] Combined features shape: (64, 50)
[Debug] Combined features columns: Index(['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min',
       'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
       'rolling_fg3a', 'rolling_ftm', 'rolling_fta', 'rolling_oreb',
       'rolling_dreb', 'rolling_stl', 'rolling_blk', 'rolling_tov',
       'rolling_plus_minus', 'gp_rank', 'w_rank', 'l_rank',
       'opponent_w_pct_rank', 'min_rank', 'fgm_rank', 'fga_rank',
       'fg_pct_rank', 'fg3m_rank', 'fg3a_rank', 'fg3_pct_rank', 'ftm_rank',
       'fta_rank', 'ft_pct_rank', 'oreb_rank', 'dreb_rank',
       'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank', 'stl_rank',
       'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank', 'opponent_pts_rank',
       'opponent_plus_minus_rank', 'Win_PCT', 'Conf_Rank', 'Div_Rank',
       'Pts_Rank', 'Reb_Rank', 'Ast_Rank', 'Opp_Pts_Rank'],
      dtype='object')
Removing 6 highly correlated features: ['rolling_fgm', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fta_
```

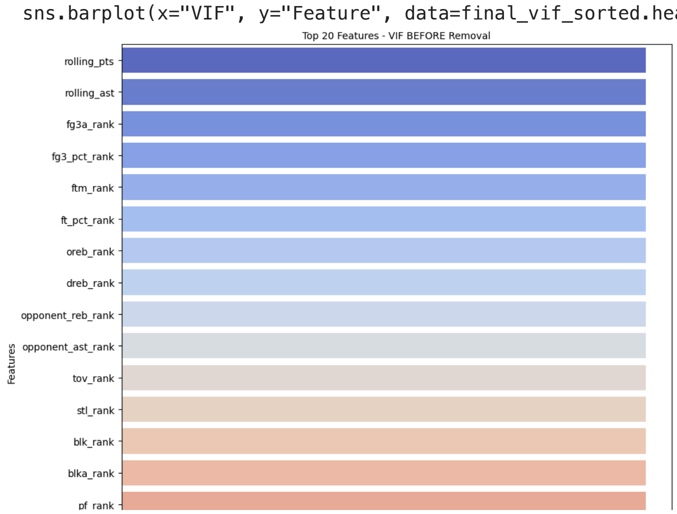


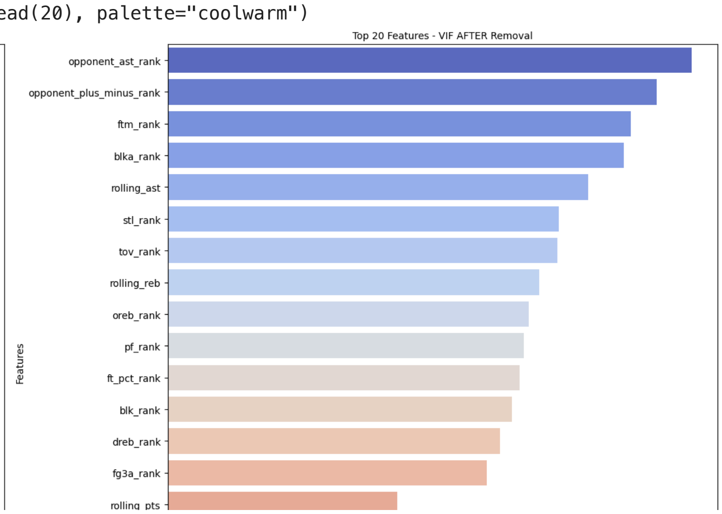


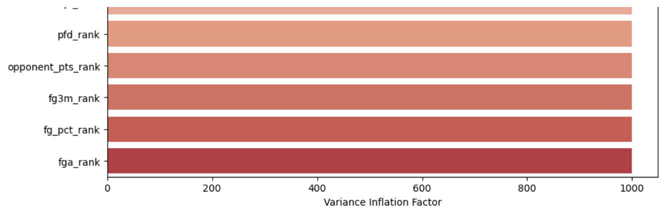

```
--- Feature Analysis ---
Original Features Count: 28
Features Before Correlation Removal: 50
Features After Correlation Removal: 46

--- Dropped Features ---
['rolling_fgm', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fta_rank']

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m', 'rolling_fg3a'
```

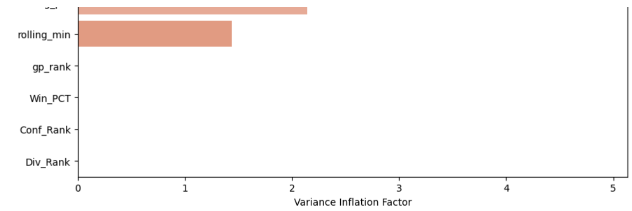



```
[INFO]: Starting feature scaling and preparation.
[INFO]: Feature scaling completed.
<ipython-input-111-9e74f1c2681d>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
No highly correlated features found to remove.
[INFO]: Dropped highly correlated features: []
Removing rolling_fga with VIF: inf
Removing rolling_fg_pct with VIF: inf
Removing rolling_fg3m with VIF: inf
Removing rolling_fg3a with VIF: inf
Removing rolling_ftm with VIF: inf
Removing rolling_oreb with VIF: inf
Removing rolling_stl with VIF: inf
Removing rolling_blk with VIF: inf
Removing rolling_tov with VIF: inf
Removing rolling_plus_minus with VIF: inf
Removing w_rank with VIF: inf
Removing min_rank with VIF: inf
Removing fgm_rank with VIF: inf
Removing fga_rank with VIF: inf
Removing fg_pct_rank with VIF: inf
Removing opponent_reb_rank with VIF: 247.43
Removing fg3m_rank with VIF: 132.31
Removing opponent_pts_rank with VIF: 40.42
Removing pfd_rank with VIF: 16.04
Removing fg3_pct_rank with VIF: 15.18
[INFO]: Features retained after VIF reduction: ['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fg3a
<ipython-input-111-9e74f1c2681d>:53: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
```

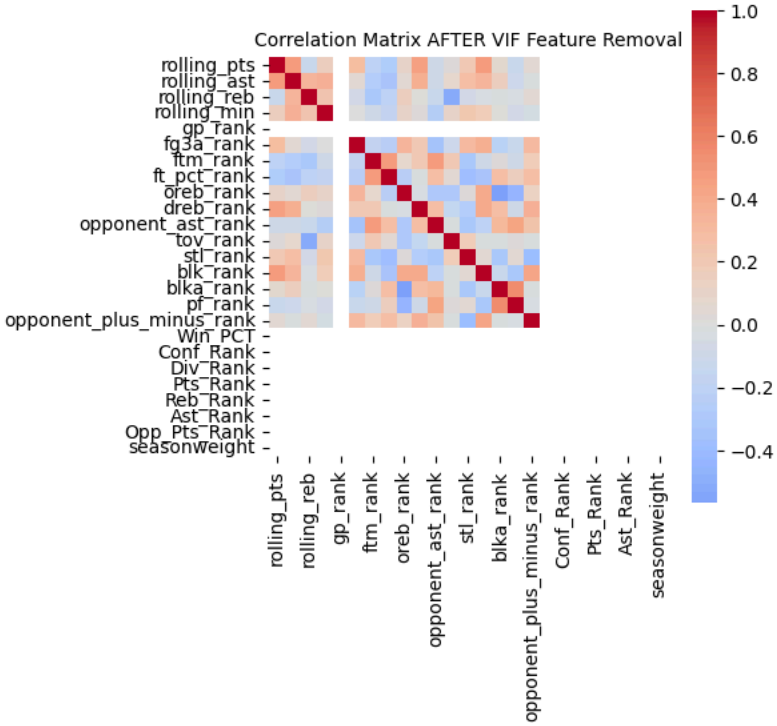


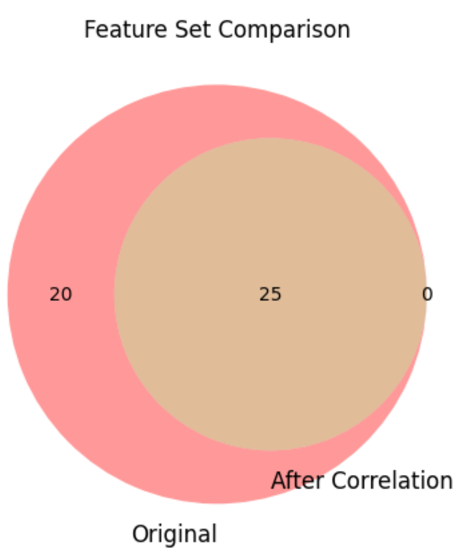







```
--- Feature Analysis ---
Original Features Count: 45
Features After Correlation Removal: 25

--- Dropped Features ---
Correlation Dropped Features: []

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fg3a_rank', 'ftm_rank', 'ft_pct_rank', 'oreb_rank',
```

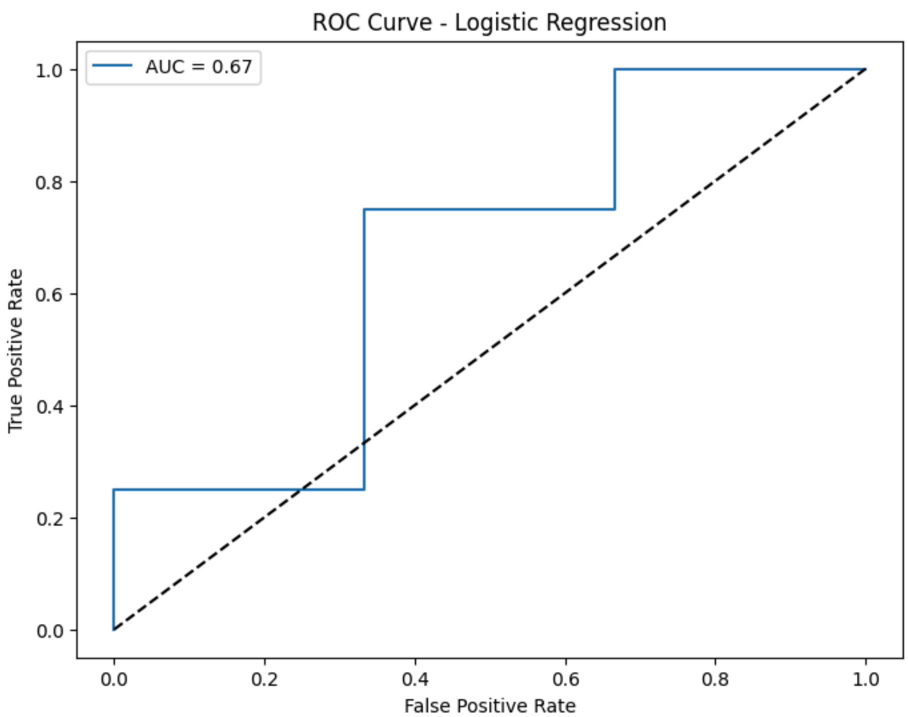


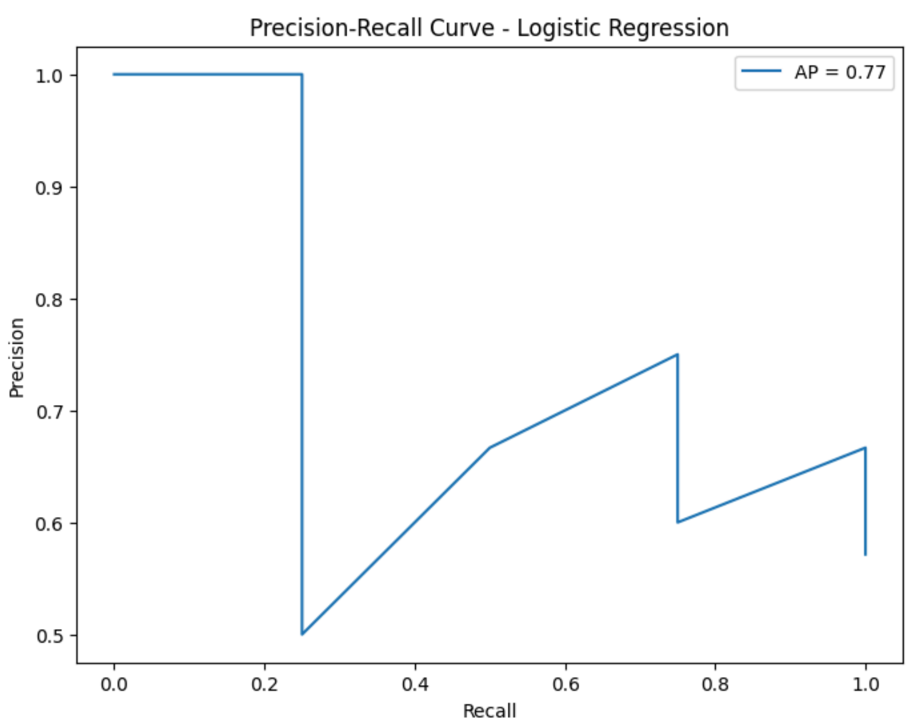



```
[INFO]: Calculated VIF data.
[INFO]: Initial VIF Data:
[DEBUG]:                    Feature       VIF
10         opponent_ast_rank  4.888624
16  opponent_plus_minus_rank  4.567950
6                   ftm_rank  4.321839
14                 blka_rank  4.260011
1                rolling_ast  3.926980
12                  stl_rank  3.654587
11                  tov_rank  3.639378
```

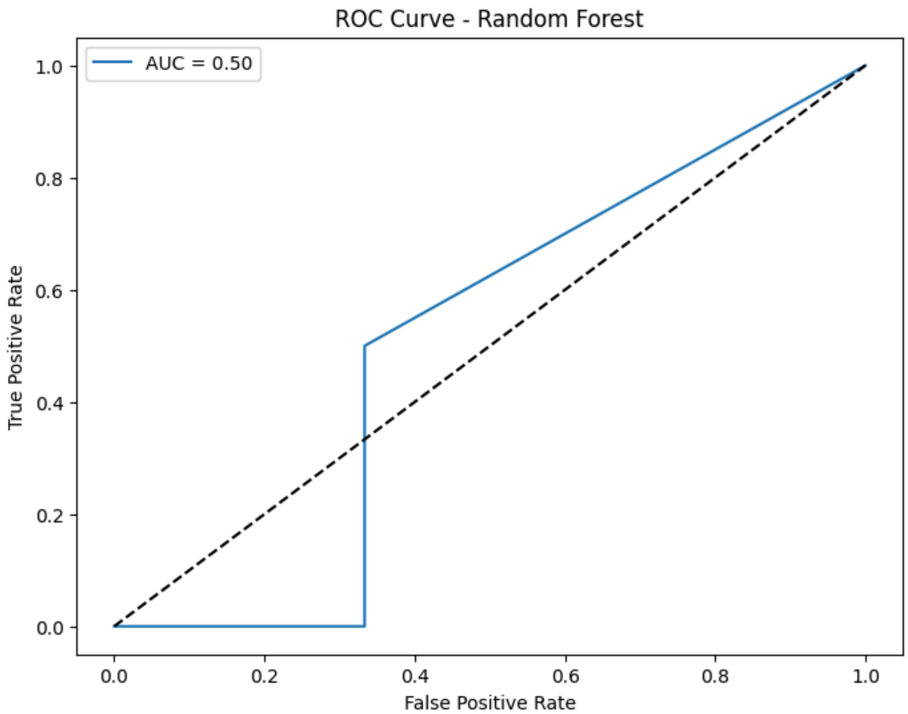
```
2                rolling_reb  3.471436
8                  oreb_rank  3.370303
15                   pf_rank  3.327771
7                ft_pct_rank  3.284585
13                  blk_rank  3.213146
9                  dreb_rank  3.103735
5                  fg3a_rank  2.977809
0                rolling_pts  2.144182
3                rolling_min  1.438733
4                    gp_rank       NaN
17                   Win_PCT       NaN
18                 Conf_Rank       NaN
19                  Div_Rank       NaN
20                  Pts_Rank       NaN
21                  Reb_Rank       NaN
22                  Ast_Rank       NaN
23              Opp_Pts_Rank       NaN
24              seasonweight       NaN
[INFO]: Preparing next_game_features for scaling and prediction.
[INFO]: Successfully scaled next_game_features.
[INFO]: Splitting data into training and validation sets.
[INFO]: Training set shape: (15, 25), Validation set shape: (7, 25)
[INFO]: Training Naive Model.
[INFO]: Training Logistic Regression model.
[INFO]: Training Random Forest model with cross-validation.
[INFO]: Performing hyperparameter optimization for Random Forest.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[INFO]: Random Forest training and optimization completed.
[INFO]: Training Gradient Boosting model.
[INFO]: Training Neural Network model with randomized hyperparameter search.
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[ERROR]: Neural Network training failed:
All the 30 fits failed.
It is very likely that your model is misconfigured.
You can try to debug the error by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 751, in fit
    return self._fit(X, y, incremental=False)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 475, in _fit
    self._fit_stochastic(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 588, in _fit_stochas
    X, X_val, y, y_val = train_test_split(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 186, in wrapper
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2806, in train_test_split
    train, test = next(cv.split(X=arrays[0], y=stratify))
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 1843, in split
    for train, test in self._iter_indices(X, y, groups):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2265, in _iter_indices
    raise ValueError(
ValueError: The test_size = 1 should be greater or equal to the number of classes = 2

[ERROR]: Neural Network replaced with Logistic Regression Model
[INFO]: Neural Network training and optimization completed.
[INFO]: Training Weighted Voting Classifier.
Stacking Classifier Performance:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.67      1.00      0.80         4

    accuracy                           0.71         7
   macro avg       0.83      0.67      0.65         7
weighted avg       0.81      0.71      0.67         7


Voting Classifier Performance:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.67      1.00      0.80         4

    accuracy                           0.71         7
   macro avg       0.83      0.67      0.65         7
weighted avg       0.81      0.71      0.67         7

[INFO]: Evaluating Naive Model.
```

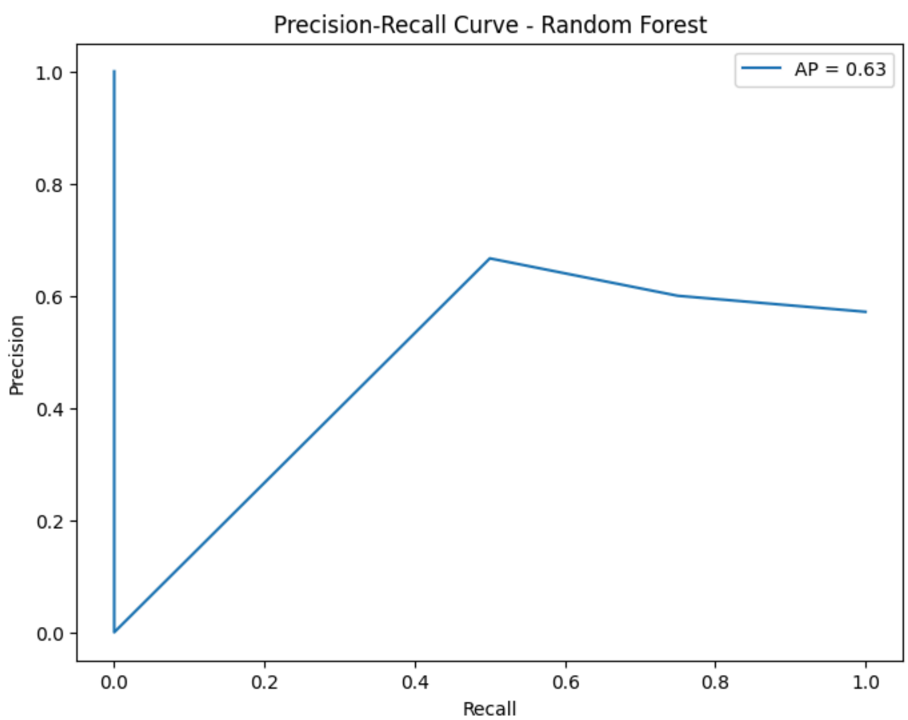
```
[INFO]: Evaluating Naive Model.
[INFO]: Naive Model Accuracy: 0.5714
[INFO]: Naive Model Classification Report:
[INFO]: {'0': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 3.0}, '1': {'precision': 0.5714285714285714, 're
[INFO]: Naive Model Confusion Matrix:
[INFO]: [[0 3]
 [0 4]]
[INFO]: Naive Model evaluation completed.
Evaluating Logistic Regression...
Classification report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       0.60      0.75      0.67         4

    accuracy                           0.57         7
   macro avg       0.55      0.54      0.53         7
weighted avg       0.56      0.57      0.55         7

Precision: 0.5571, Recall: 0.5714, F1 Score: 0.5524
Confusion Matrix:
[[1 2]
 [1 3]]
```



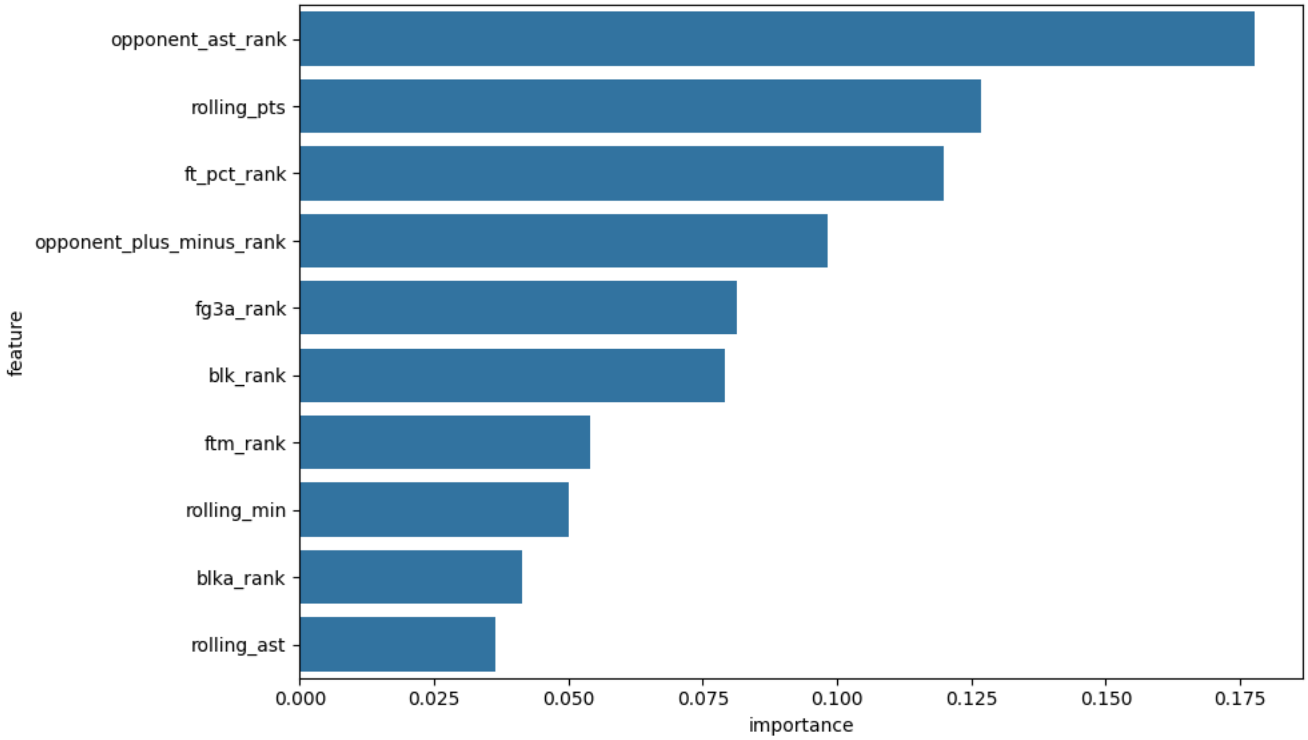


```
Feature importance not available for Logistic Regression.
Number of Misclassified Samples: 3
[INFO]: Logistic Regression evaluation completed.


Evaluating Random Forest...
Classification report for Random Forest:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.57      1.00      0.73         4

    accuracy                           0.57         7
   macro avg       0.29      0.50      0.36         7
weighted avg       0.33      0.57      0.42         7

Precision: 0.3265, Recall: 0.5714, F1 Score: 0.4156
Confusion Matrix:
[[0 3]
 [0 4]]
```

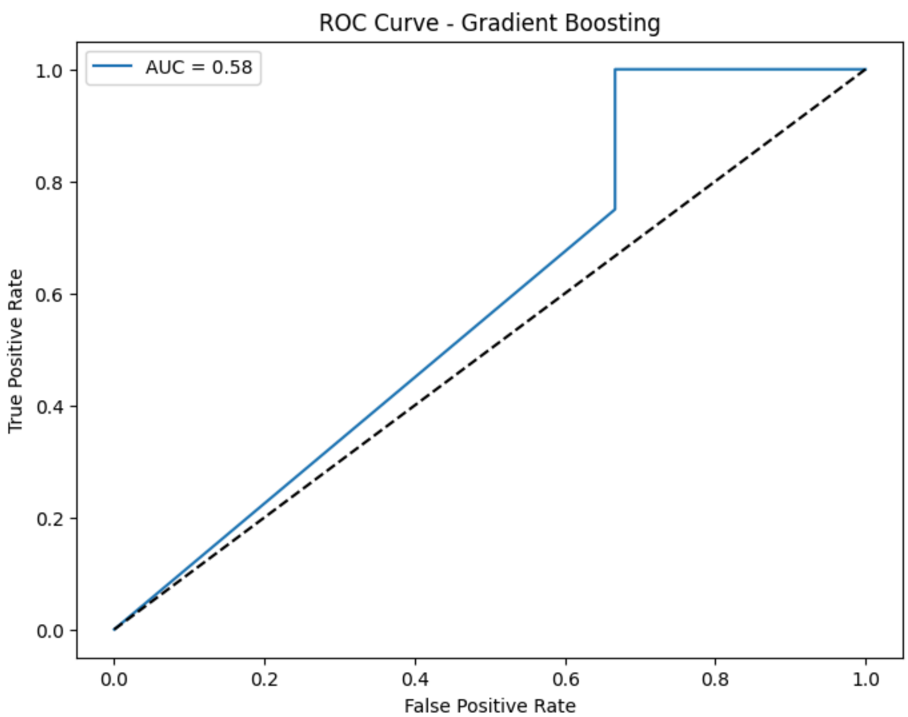




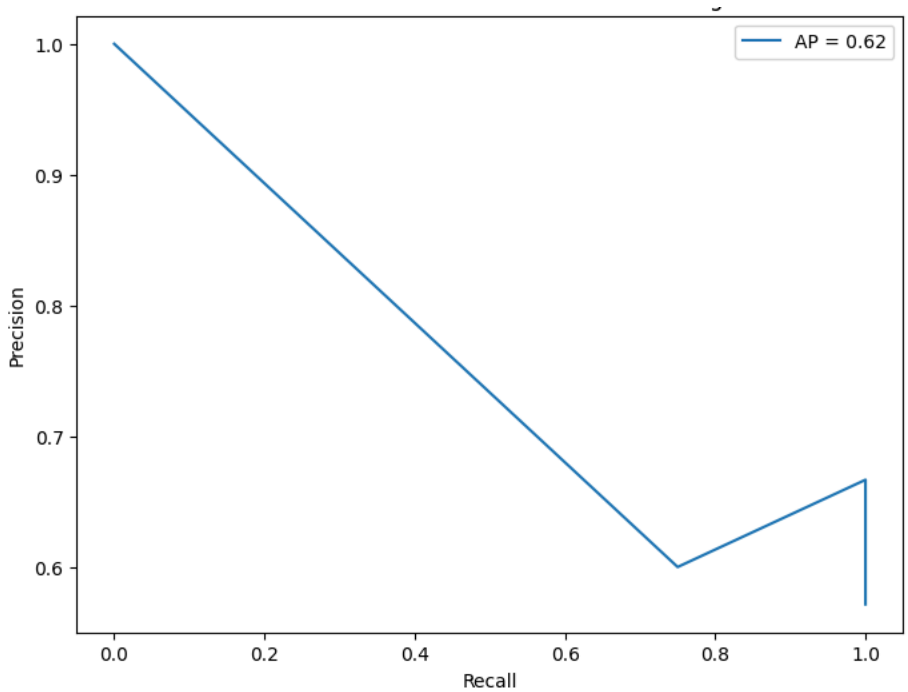
Top 10 Feature Importances - Random Forest



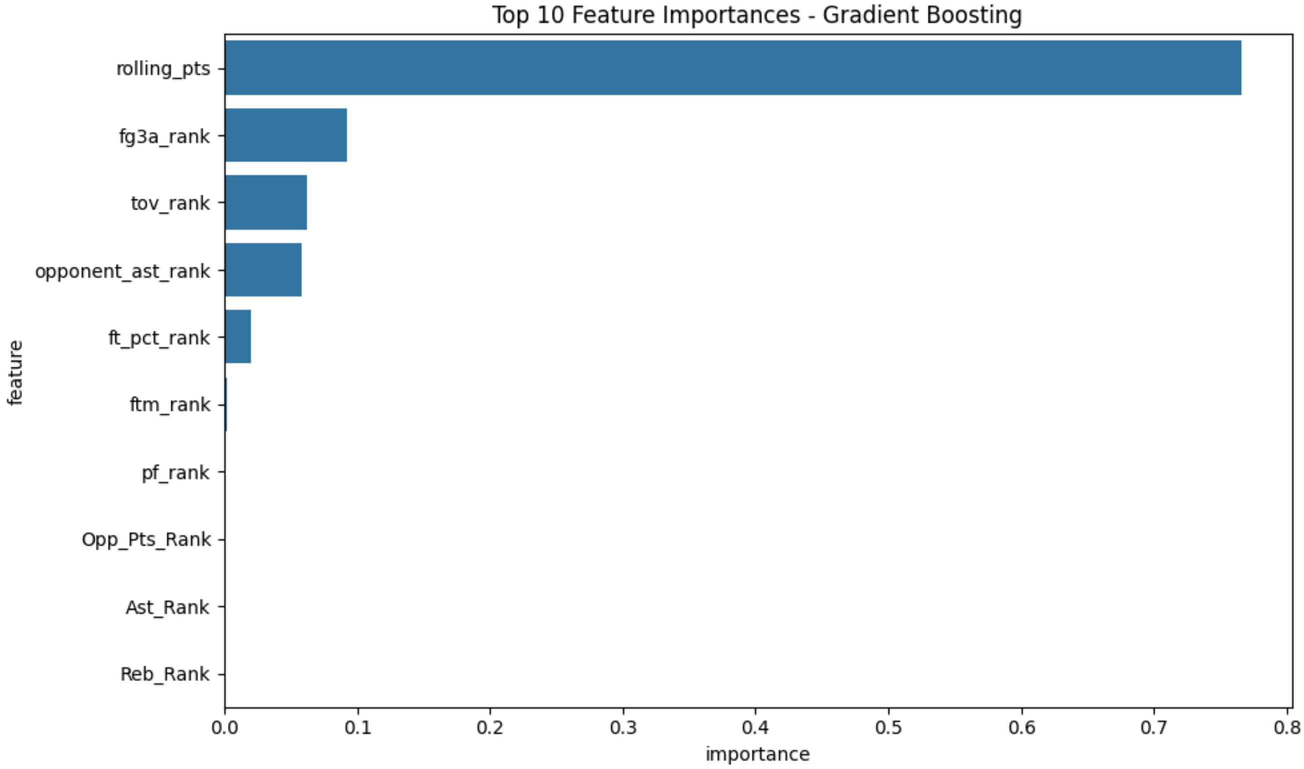

```
Number of Misclassified Samples: 3
[INFO]: Random Forest evaluation completed.


Evaluating Gradient Boosting...
Classification report for Gradient Boosting:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.67      1.00      0.80         4

    accuracy                           0.71         7
   macro avg       0.83      0.67      0.65         7
weighted avg       0.81      0.71      0.67         7

Precision: 0.8095, Recall: 0.7143, F1 Score: 0.6714
Confusion Matrix:
[[1 2]
 [0 4]]
```



Precision-Recall Curve - Gradient Boosting

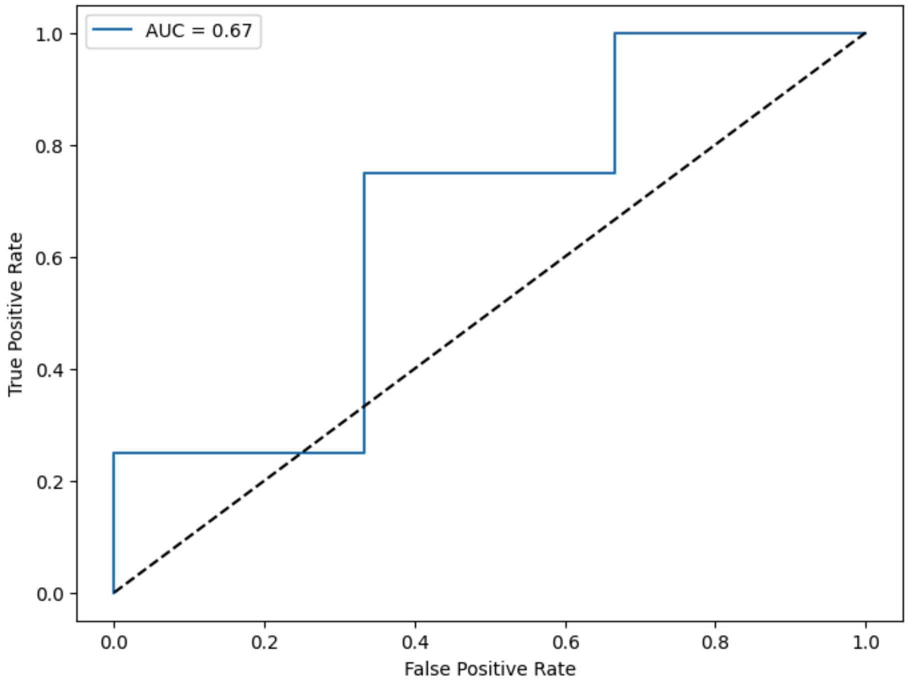


Top 10 Feature Importances - Gradient Boosting

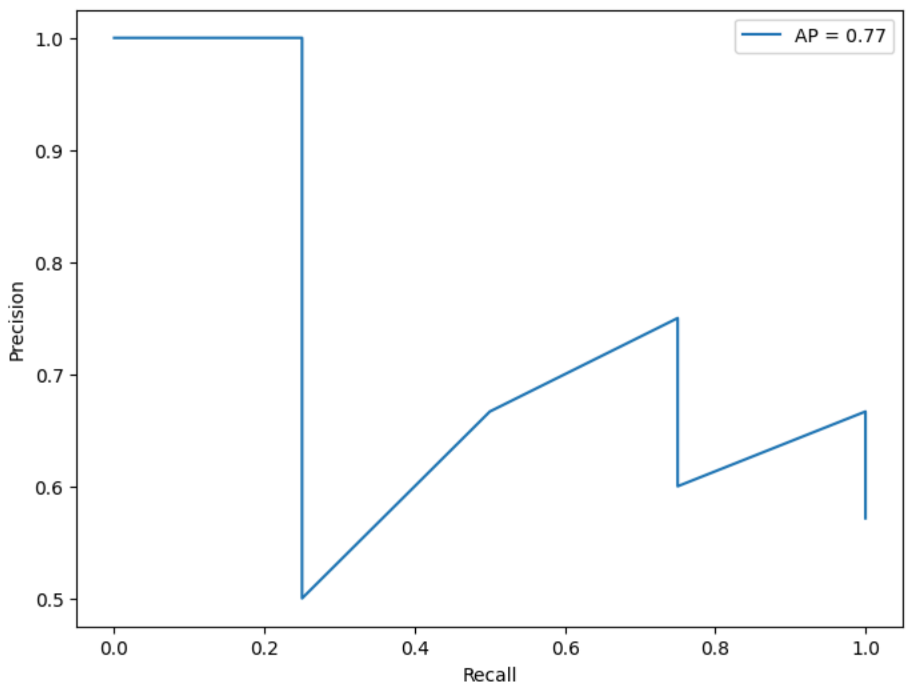



```
Number of Misclassified Samples: 2
[INFO]: Gradient Boosting evaluation completed.


Evaluating Neural Network...
Classification report for Neural Network:
              precision    recall  f1-score   support

           0       0.50      0.33      0.40         3
           1       0.60      0.75      0.67         4

    accuracy                           0.57         7
   macro avg       0.55      0.54      0.53         7
weighted avg       0.56      0.57      0.55         7

Precision: 0.5571, Recall: 0.5714, F1 Score: 0.5524
Confusion Matrix:
[[1 2]
 [1 3]]
```

ROC Curve - Neural Network

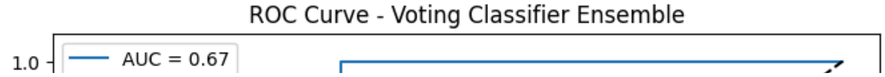


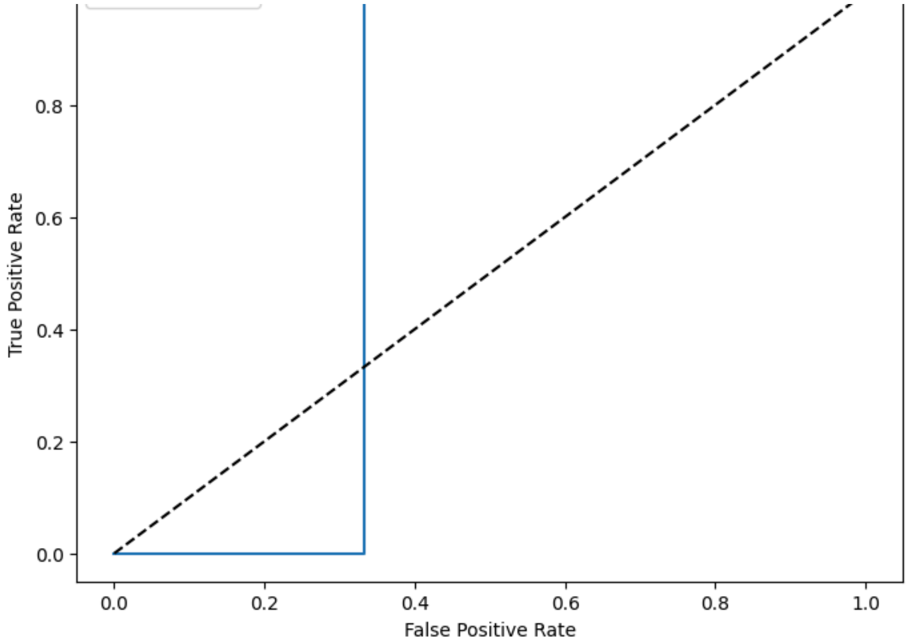
Precision-Recall Curve - Neural Network

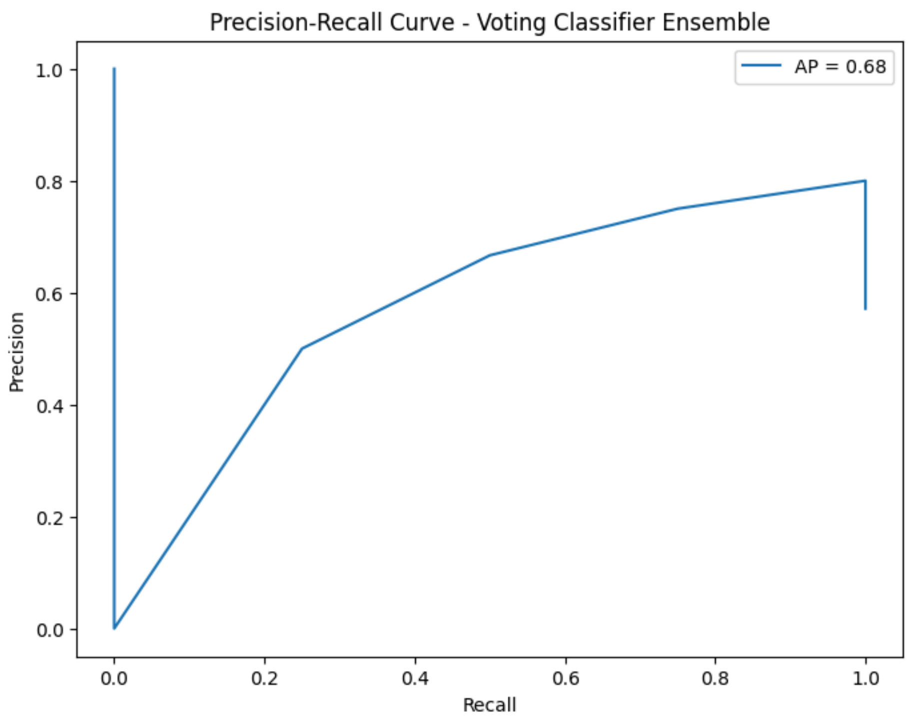



```
Feature importance not available for Neural Network.
Number of Misclassified Samples: 3
[INFO]: Neural Network evaluation completed.


Evaluating Voting Classifier Ensemble...
Classification report for Voting Classifier Ensemble:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.67      1.00      0.80         4

    accuracy                           0.71         7
   macro avg       0.83      0.67      0.65         7
weighted avg       0.81      0.71      0.67         7

Precision: 0.8095, Recall: 0.7143, F1 Score: 0.6714
Confusion Matrix:
[[1 2]
 [0 4]]
```

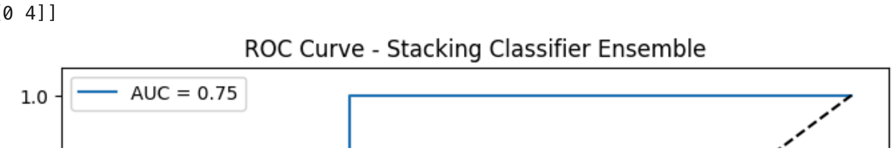
ROC Curve - Voting Classifier Ensemble

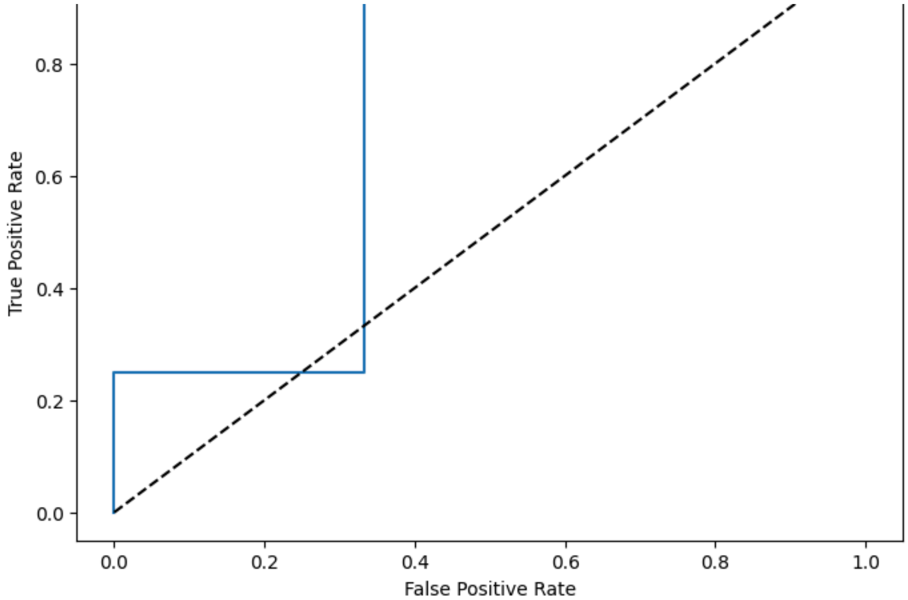


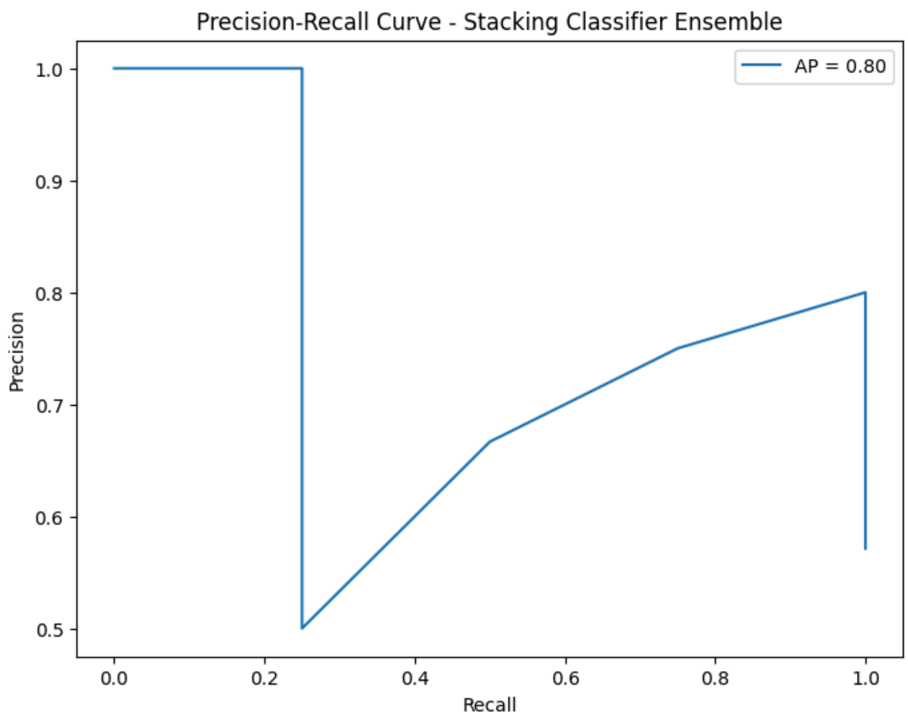





```
Feature importance not available for Voting Classifier Ensemble.
Number of Misclassified Samples: 2
[INFO]: Voting Classifier Ensemble evaluation completed.


Evaluating Stacking Classifier Ensemble...
Classification report for Stacking Classifier Ensemble:
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.67      1.00      0.80         4

    accuracy                           0.71         7
   macro avg       0.83      0.67      0.65         7
weighted avg       0.81      0.71      0.67         7

Precision: 0.8095, Recall: 0.7143, F1 Score: 0.6714
Confusion Matrix:
[[1 2]
 [0 4]]
```

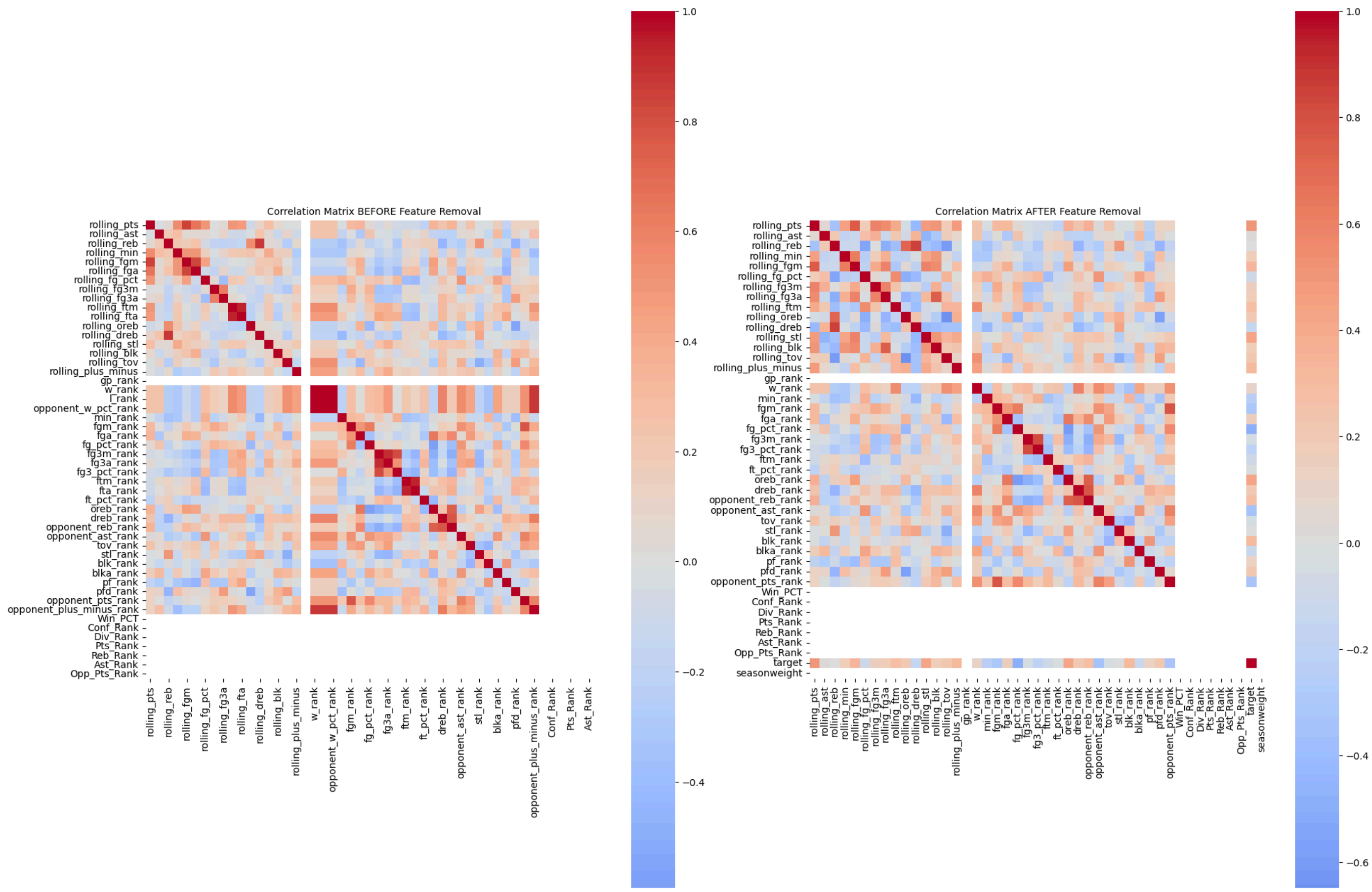




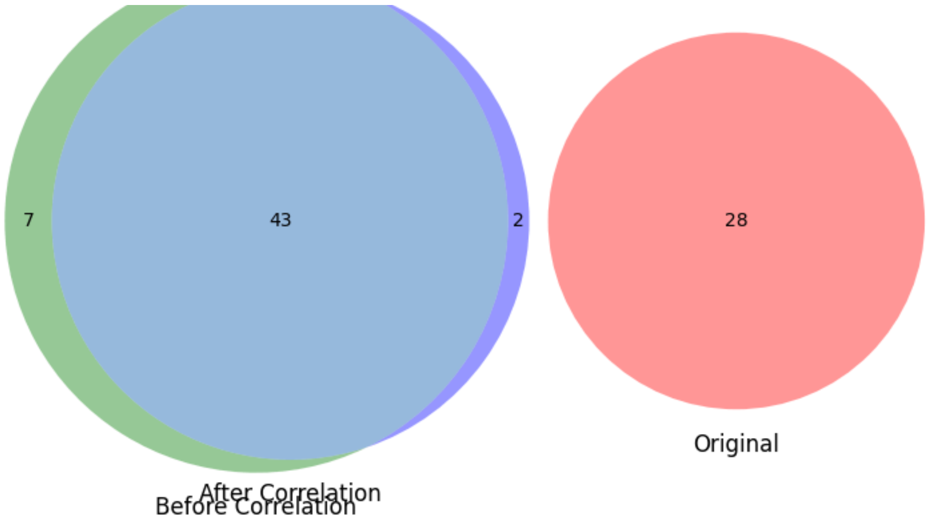



```
Feature importance not available for Stacking Classifier Ensemble.
Number of Misclassified Samples: 2
[INFO]: Stacking Classifier Ensemble evaluation completed.



Blended Prediction for Next Game:
Prediction: Over
Probability of Under: 0.47
Probability of Over: 0.53
Prediction for fg3m (stat line 3.5): Over
Probability of Under: 0.47
Probability of Over: 0.53

--- Processing Giannis Antetokounmpo ---
203507

Next Game Details:
GAME_DATE            DEC 14, 2024
HOME_TEAM_NAME          Milwaukee
VISITOR_TEAM_NAME         Atlanta
GAME_TIME                04:30 PM
Name: 0, dtype: object

[Debug] Starting dataset preparation...
Columns in fetched player game log for season 2024-25: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
```

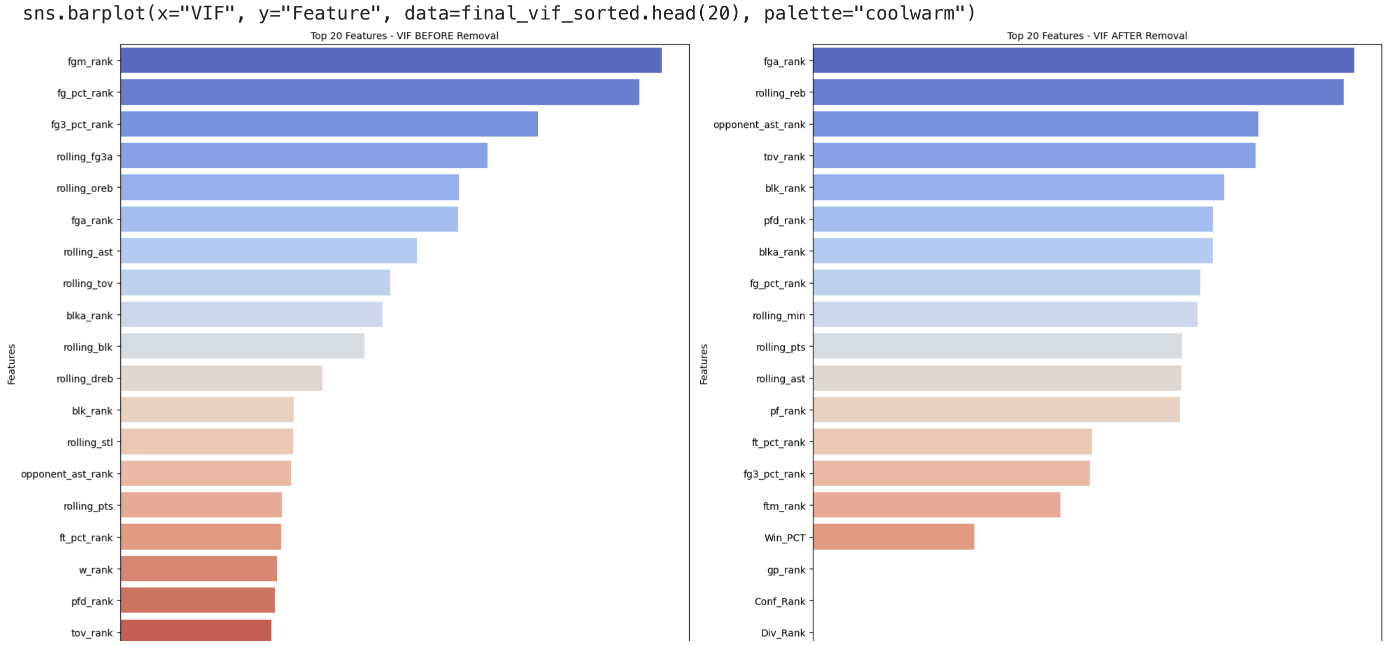
```
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
Columns in fetched player game log for season 2023-24: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')

Fetched player data shape: (95, 28)

Rolling averages calculated successfully.
Fetched team game logs for season 2024-25, shape: (726, 57)

[Debug] Combined features shape: (95, 50)
[Debug] Combined features columns: Index(['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min',
       'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
       'rolling_fg3a', 'rolling_ftm', 'rolling_fta', 'rolling_oreb',
       'rolling_dreb', 'rolling_stl', 'rolling_blk', 'rolling_tov',
       'rolling_plus_minus', 'gp_rank', 'w_rank', 'l_rank',
       'opponent_w_pct_rank', 'min_rank', 'fgm_rank', 'fga_rank',
       'fg_pct_rank', 'fg3m_rank', 'fg3a_rank', 'fg3_pct_rank', 'ftm_rank',
       'fta_rank', 'ft_pct_rank', 'oreb_rank', 'dreb_rank',
       'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank', 'stl_rank',
       'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank', 'opponent_pts_rank',
       'opponent_plus_minus_rank', 'Win_PCT', 'Conf_Rank', 'Div_Rank',
       'Pts_Rank', 'Reb_Rank', 'Ast_Rank', 'Opp_Pts_Rank'],
      dtype='object')
Removing 7 highly correlated features: ['rolling_fga', 'rolling_fta', 'l_rank', 'opponent_w_pct_rank', 'fg3a_rank', 'fta_ran
```

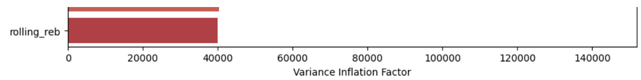


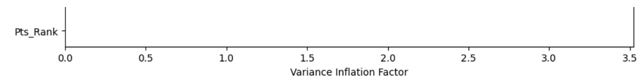

```
--- Feature Analysis ---
Original Features Count: 28
Features Before Correlation Removal: 50
Features After Correlation Removal: 45

--- Dropped Features ---
['rolling_fga', 'rolling_fta', 'l_rank', 'opponent_w_pct_rank', 'fg3a_rank', 'fta_rank', 'opponent_plus_minus_rank']

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_fgm', 'rolling_fg_pct', 'rolling_fg3m', 'rolling_fg3a'
```



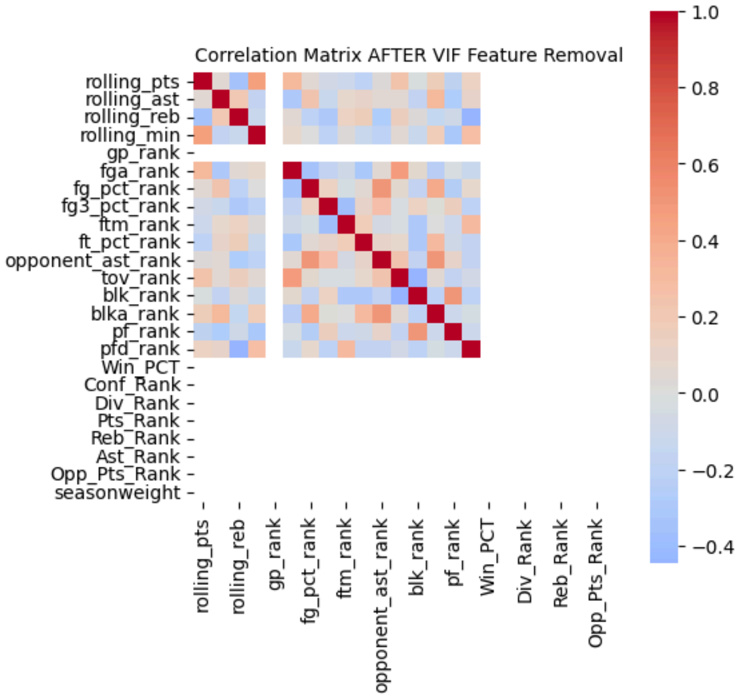

```
[INFO]: Starting feature scaling and preparation.
[INFO]: Feature scaling completed.
<ipython-input-111-9e74f1c2681d>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
No highly correlated features found to remove.
[INFO]: Dropped highly correlated features: []
Removing rolling_fgm with VIF: inf
Removing rolling_fg_pct with VIF: inf
Removing rolling_fg3m with VIF: inf
Removing rolling_fg3a with VIF: inf
Removing rolling_ftm with VIF: inf
Removing rolling_oreb with VIF: inf
Removing rolling_dreb with VIF: inf
Removing rolling_stl with VIF: inf
Removing rolling_blk with VIF: inf
Removing rolling_tov with VIF: inf
Removing rolling_plus_minus with VIF: inf
Removing w_rank with VIF: inf
Removing min_rank with VIF: inf
Removing fgm_rank with VIF: inf
Removing opponent_pts_rank with VIF: 599.76
Removing opponent_reb_rank with VIF: 86.48
Removing stl_rank with VIF: 44.84
Removing fg3m_rank with VIF: 11.22
Removing oreb_rank with VIF: 8.36
Removing dreb_rank with VIF: 7.37
[INFO]: Features retained after VIF reduction: ['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fga_
<ipython-input-111-9e74f1c2681d>:53: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
```

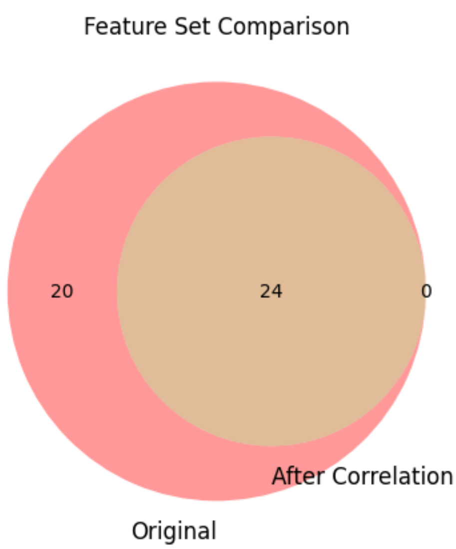







```
--- Feature Analysis ---
Original Features Count: 44
Features After Correlation Removal: 24

--- Dropped Features ---
Correlation Dropped Features: []

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fga_rank', 'fg_pct_rank', 'fg3_pct_rank', 'ftm_rank
```

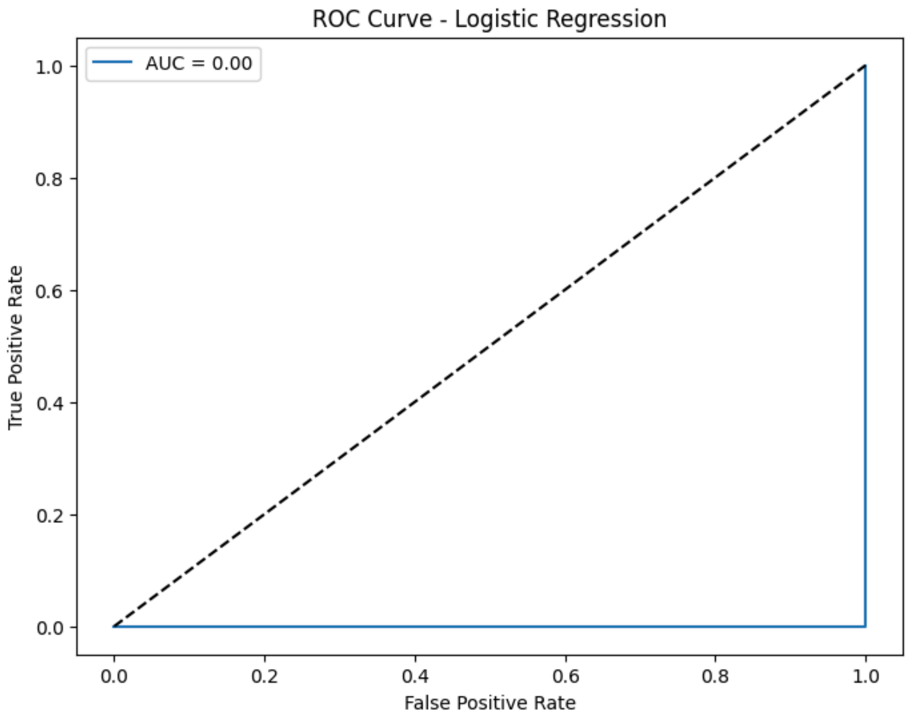


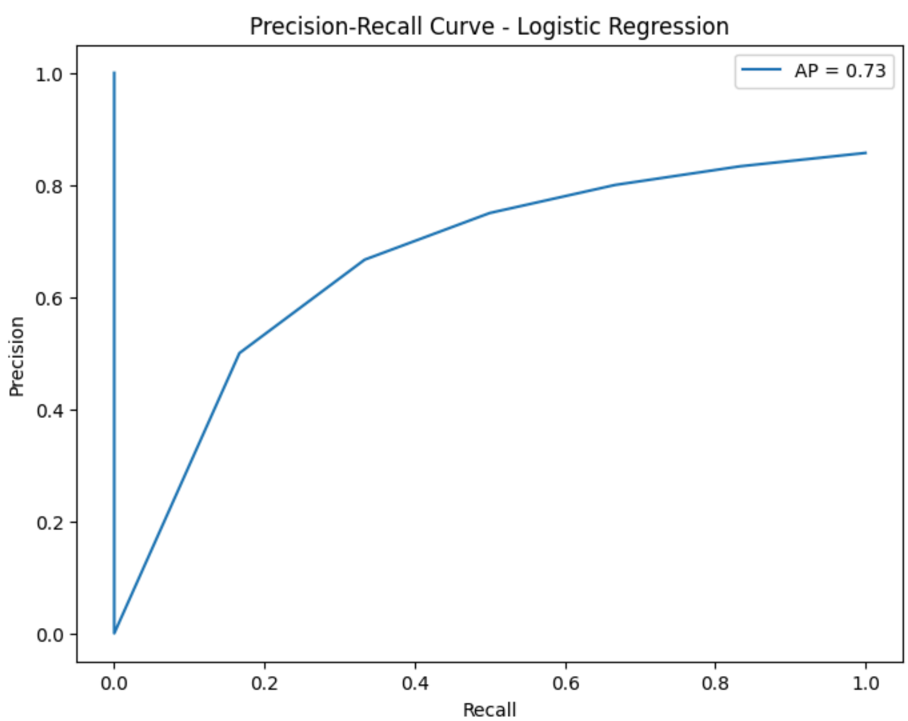



```
[INFO]: Calculated VIF data.
[INFO]: Initial VIF Data:
[DEBUG]:              Feature       VIF
5           fga_rank  3.356220
2        rolling_reb  3.287127
10  opponent_ast_rank  2.760411
11          tov_rank  2.743743
12          blk_rank  2.549496
15          pfd_rank  2.479079
13         blka_rank  2.477443
6        fg_pct_rank  2.400948
3        rolling_min  2.383772
0        rolling_pts  2.286361
1        rolling_ast  2.284489
14           pf_rank  2.275635
9       ft_pct_rank  1.728696
7      fg3_pct_rank  1.717841
```

```
8           ftm_rank  1.533881
4            gp_rank       NaN
16           Win_PCT       NaN
17         Conf_Rank       NaN
18          Div_Rank       NaN
19          Pts_Rank       NaN
20          Reb_Rank       NaN
21          Ast_Rank       NaN
22      Opp_Pts_Rank       NaN
23      seasonweight       NaN
[INFO]: Preparing next_game_features for scaling and prediction.
[INFO]: Successfully scaled next_game_features.
[INFO]: Splitting data into training and validation sets.
[INFO]: Training set shape: (15, 24), Validation set shape: (7, 24)
[INFO]: Training Naive Model.
[INFO]: Training Logistic Regression model.
[INFO]: Training Random Forest model with cross-validation.
[INFO]: Performing hyperparameter optimization for Random Forest.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[INFO]: Random Forest training and optimization completed.
[INFO]: Training Gradient Boosting model.
[INFO]: Training Neural Network model with randomized hyperparameter search.
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[ERROR]: Neural Network training failed:
All the 30 fits failed.
It is very likely that your model is misconfigured.
You can try to debug the error by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 888, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1473, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 751, in fit
    return self._fit(X, y, incremental=False)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 475, in _fit
    self._fit_stochastic(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py", line 588, in _fit_stochas
    X, X_val, y, y_val = train_test_split(
  File "/usr/local/lib/python3.10/dist-packages/sklearn/utils/_param_validation.py", line 186, in wrapper
    return func(*args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2806, in train_test_split
    train, test = next(cv.split(X=arrays[0], y=stratify))
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 1843, in split
    for train, test in self._iter_indices(X, y, groups):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_split.py", line 2265, in _iter_indices
    raise ValueError(
ValueError: The test_size = 1 should be greater or equal to the number of classes = 2

[ERROR]: Neural Network replaced with Logistic Regression Model
[INFO]: Neural Network training and optimization completed.
[INFO]: Training Weighted Voting Classifier.
Stacking Classifier Performance:
              precision    recall  f1-score   support

           0       0.14      1.00      0.25         1
           1       0.00      0.00      0.00         6

    accuracy                           0.14         7
   macro avg       0.07      0.50      0.12         7
weighted avg       0.02      0.14      0.04         7


Voting Classifier Performance:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       6.0

    accuracy                           0.00       7.0
   macro avg       0.00      0.00      0.00       7.0
weighted avg       0.00      0.00      0.00       7.0

[INFO]: Evaluating Naive Model.
[INFO]: Naive Model Accuracy: 0.1429
[INFO]: Naive Model Classification Report:
[INFO]: {'0': {'precision': 0.14285714285714285, 'recall': 1.0, 'f1-score': 0.25, 'support': 1.0}, '1': {'precision': 0.0, '
[INFO]: Naive Model Confusion Matrix:
[INFO]: [[1 0]
 [6 0]]
[INFO]: Naive Model evaluation completed.
Evaluating Logistic Regression
```

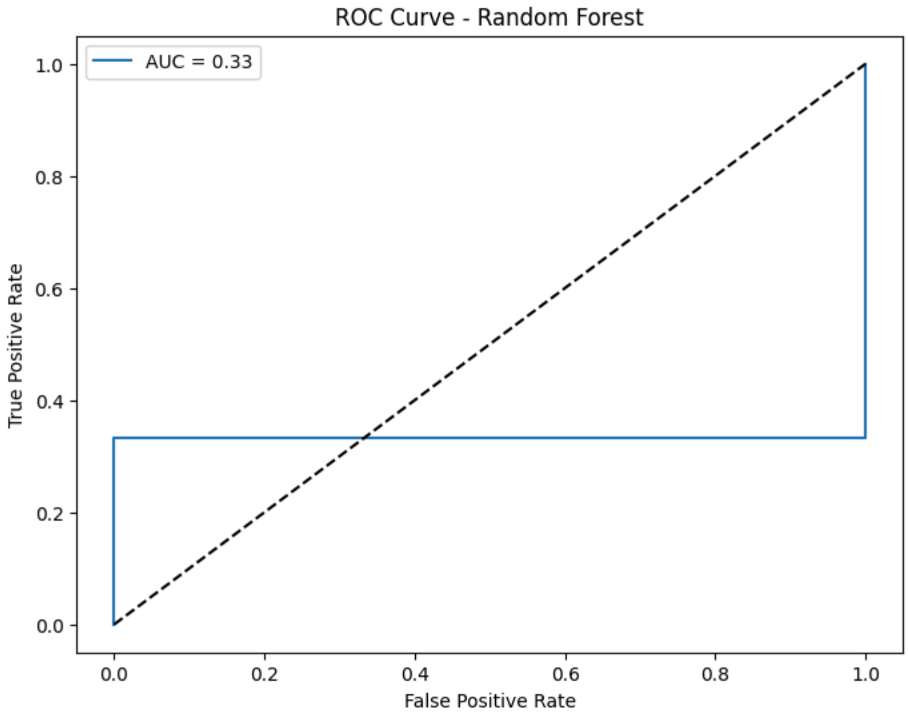
```
Evaluating Logistic Regression...
Classification report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       6.0

    accuracy                           0.00       7.0
   macro avg       0.00      0.00      0.00       7.0
weighted avg       0.00      0.00      0.00       7.0

Precision: 0.0000, Recall: 0.0000, F1 Score: 0.0000
Confusion Matrix:
[[0 1]
 [6 0]]
```

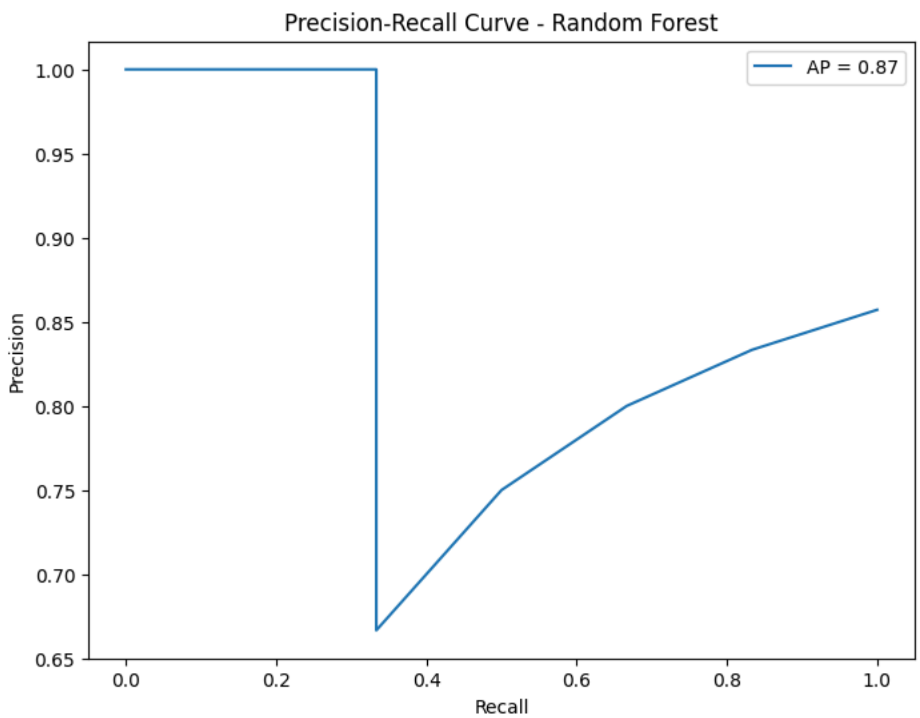


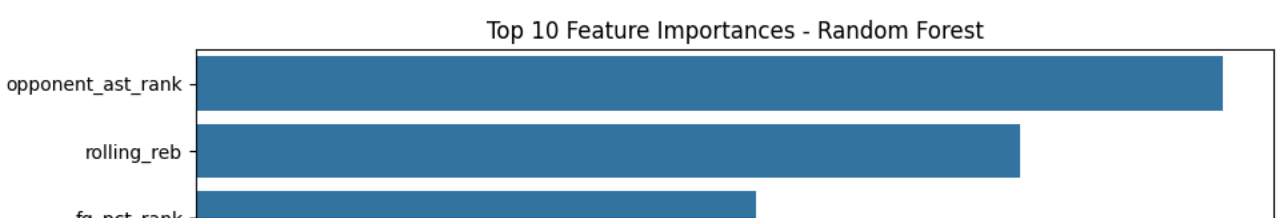


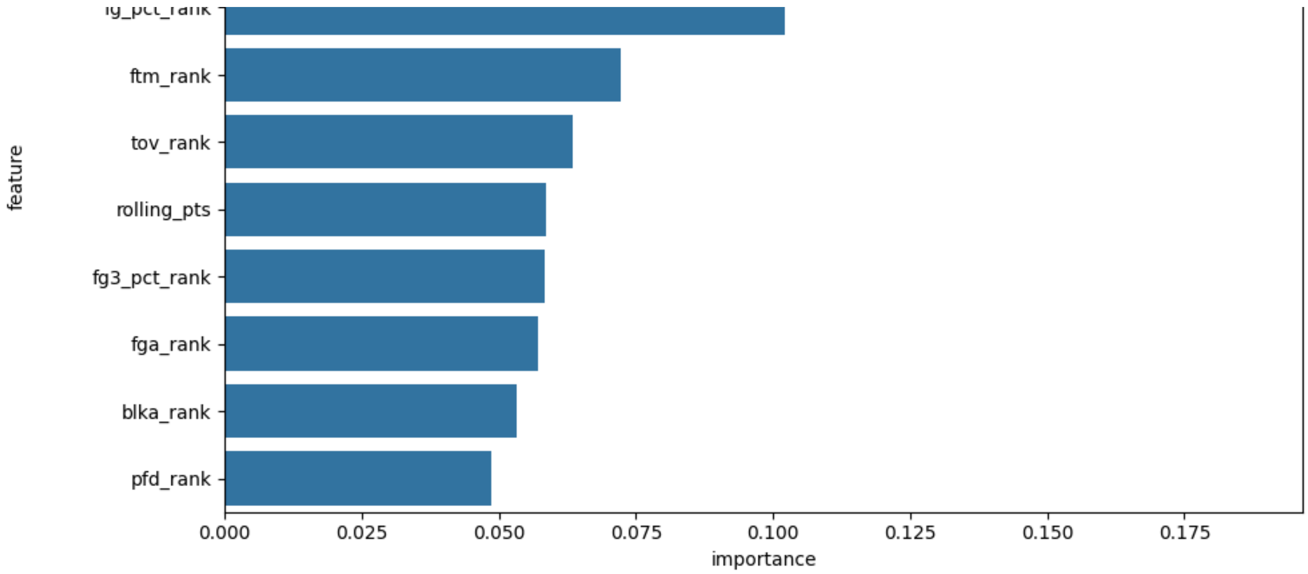

```
Feature importance not available for Logistic Regression.
Number of Misclassified Samples: 7
[INFO]: Logistic Regression evaluation completed.


Evaluating Random Forest...
Classification report for Random Forest:
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.14      1.00      0.25         1
           1       0.00      0.00      0.00         6

    accuracy                           0.14         7
   macro avg       0.07      0.50      0.12         7
weighted avg       0.02      0.14      0.04         7

Precision: 0.0204, Recall: 0.1429, F1 Score: 0.0357
Confusion Matrix:
[[1 0]
 [6 0]]
```

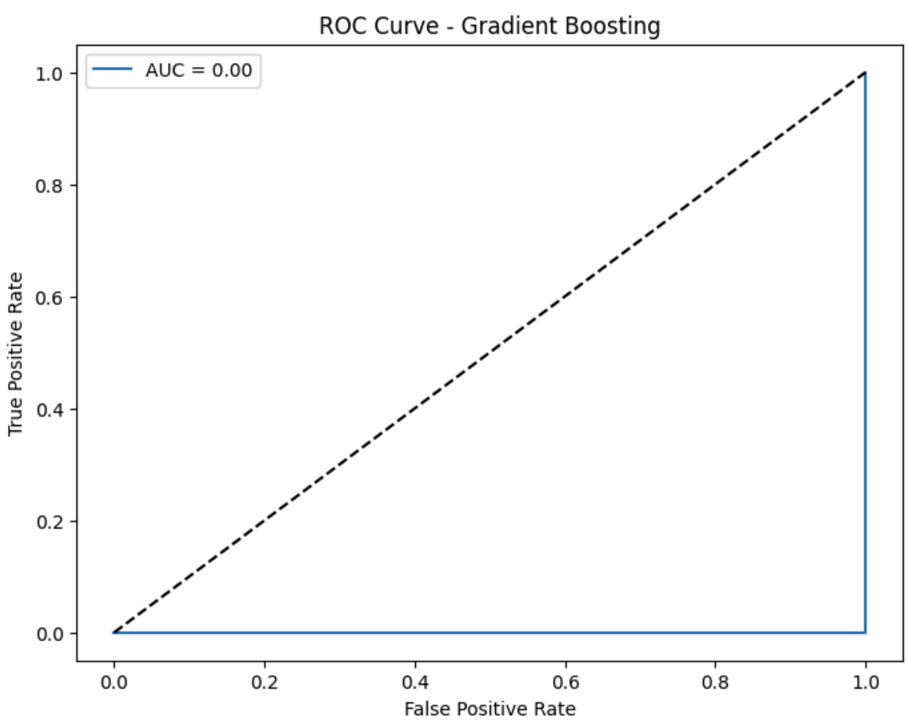


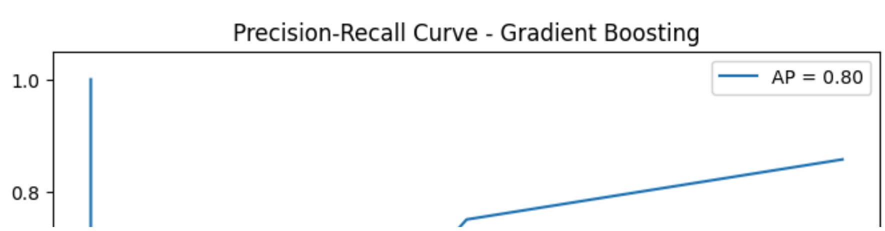


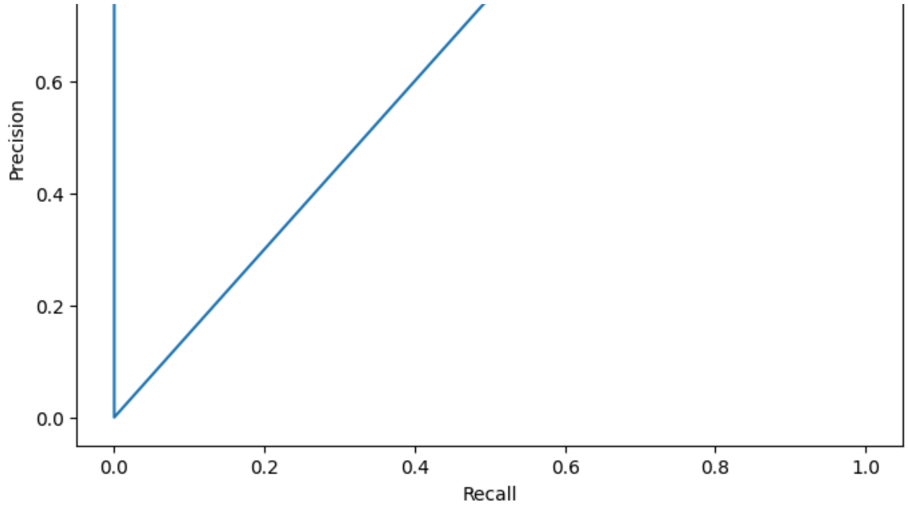


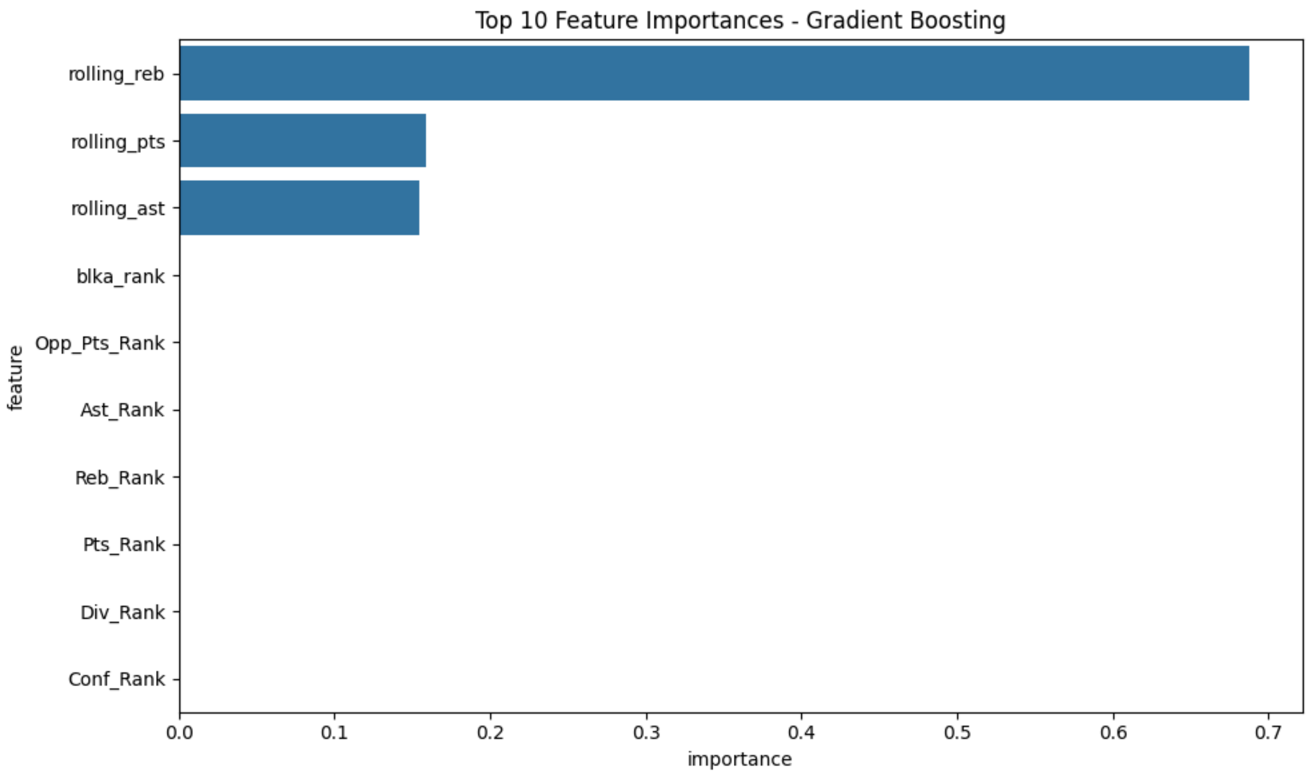



```
Number of Misclassified Samples: 6
[INFO]: Random Forest evaluation completed.


Evaluating Gradient Boosting...
Classification report for Gradient Boosting:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       6.0

    accuracy                           0.00       7.0
   macro avg       0.00      0.00      0.00       7.0
weighted avg       0.00      0.00      0.00       7.0

Precision: 0.0000, Recall: 0.0000, F1 Score: 0.0000
Confusion Matrix:
[[0 1]
 [6 0]]
```

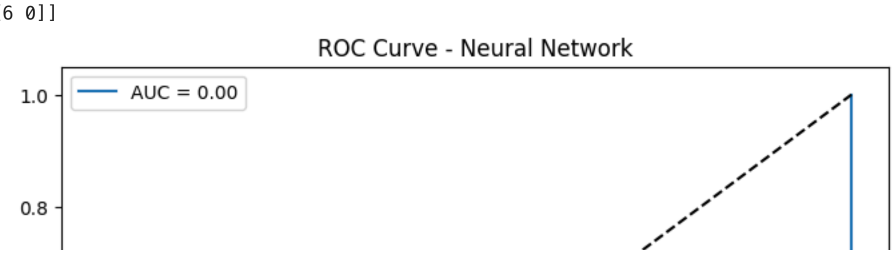


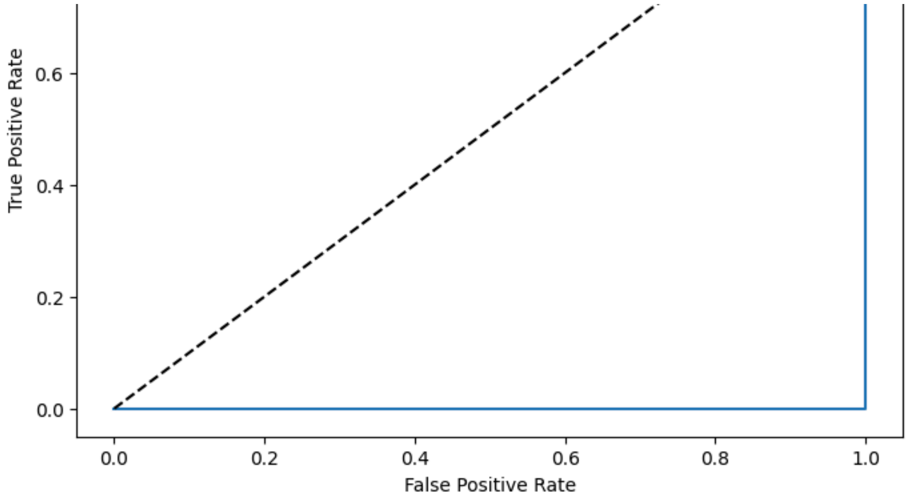


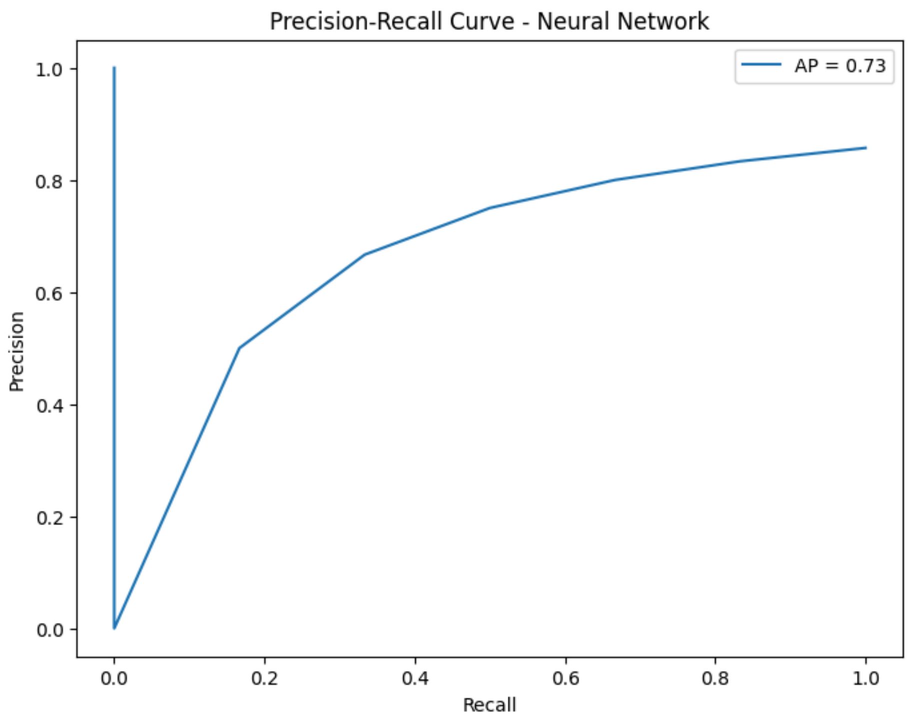




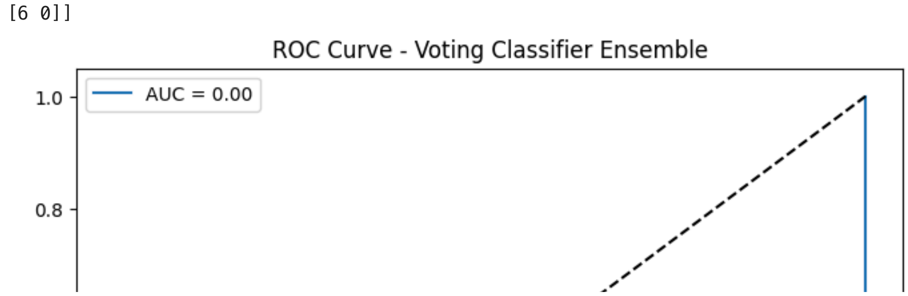

```
Number of Misclassified Samples: 7
[INFO]: Gradient Boosting evaluation completed.


Evaluating Neural Network...
Classification report for Neural Network:
              precision    recall  f1-score   support

           0       0.14      1.00      0.25         1
           1       0.00      0.00      0.00         6

    accuracy                           0.14         7
   macro avg       0.07      0.50      0.12         7
weighted avg       0.02      0.14      0.04         7

Precision: 0.0204, Recall: 0.1429, F1 Score: 0.0357
Confusion Matrix:
[[1 0]
 [6 0]]
```

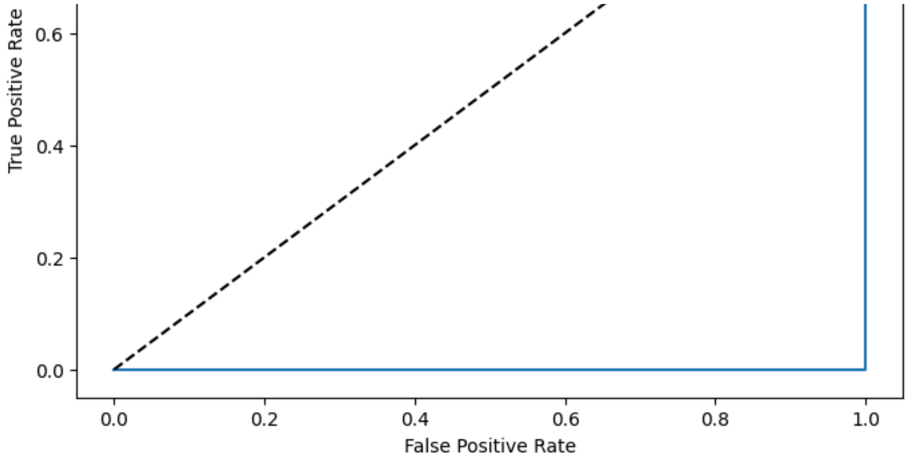


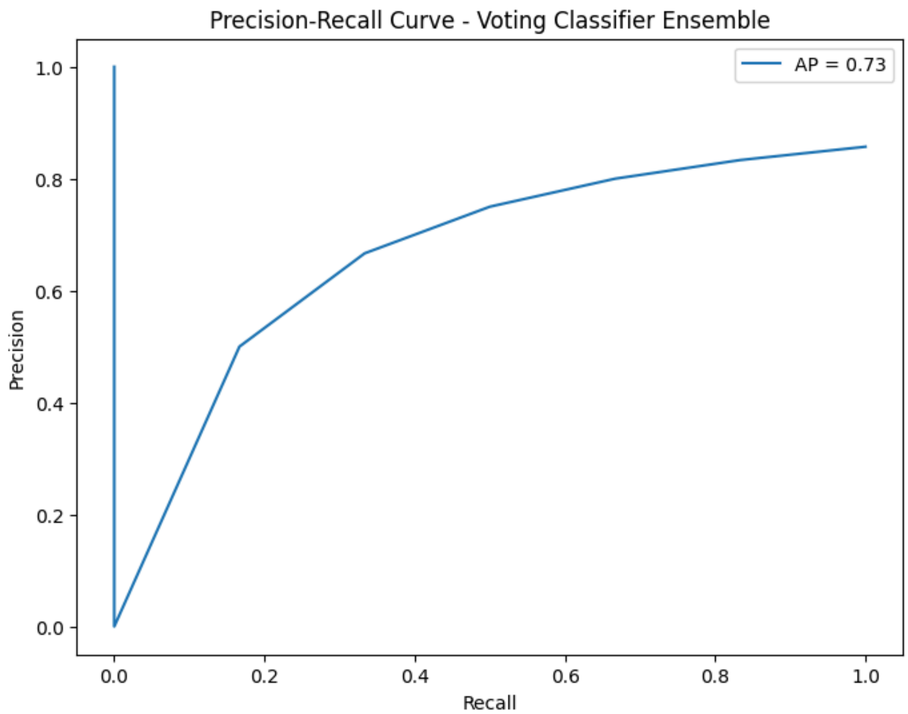




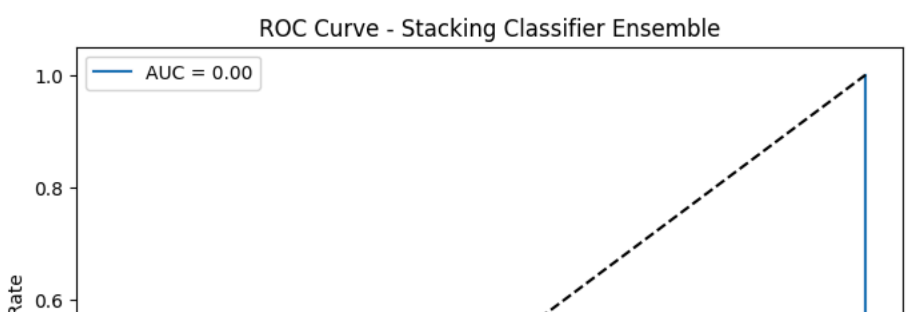

```
Feature importance not available for Neural Network.
Number of Misclassified Samples: 6
[INFO]: Neural Network evaluation completed.


Evaluating Voting Classifier Ensemble...
Classification report for Voting Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       1.0
           1       0.00      0.00      0.00       6.0

    accuracy                           0.00       7.0
   macro avg       0.00      0.00      0.00       7.0
weighted avg       0.00      0.00      0.00       7.0

Precision: 0.0000, Recall: 0.0000, F1 Score: 0.0000
Confusion Matrix:
[[0 1]
 [6 0]]
```

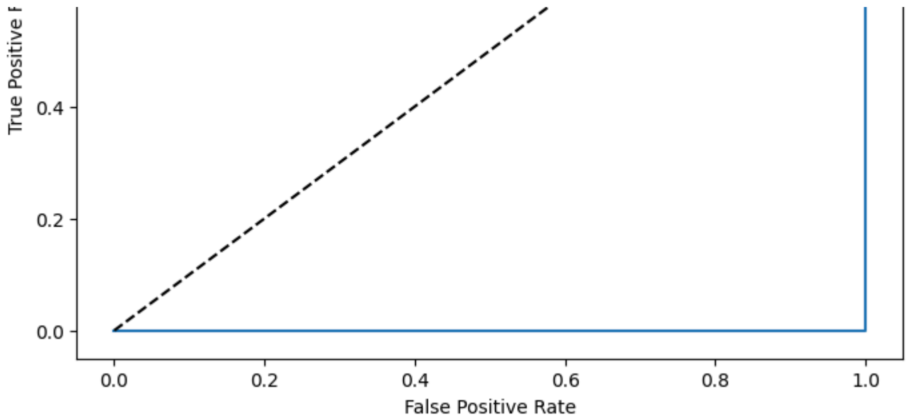


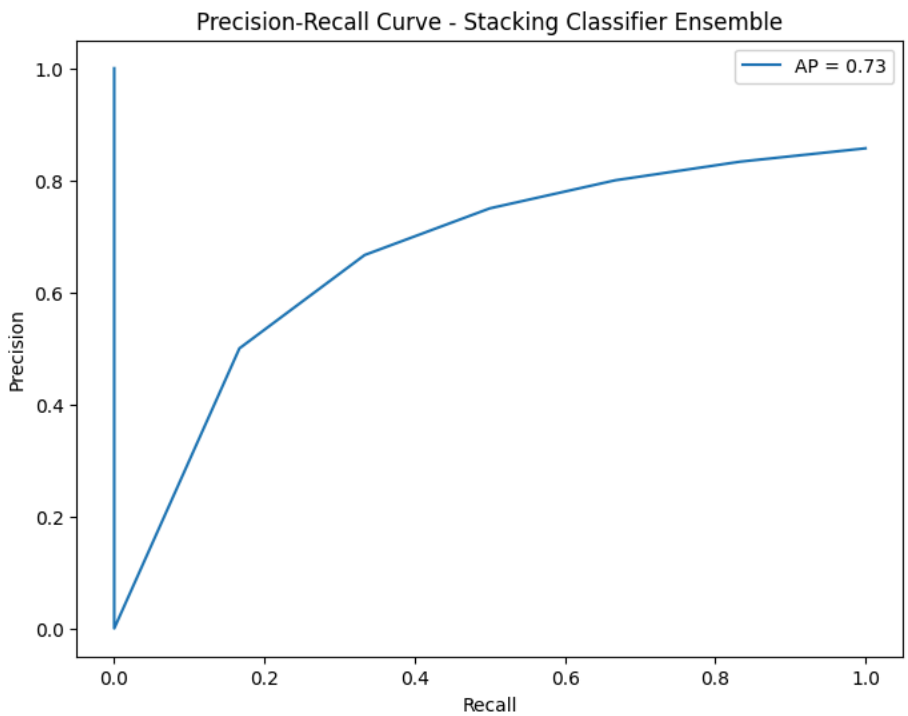





```
Feature importance not available for Voting Classifier Ensemble.
Number of Misclassified Samples: 7
[INFO]: Voting Classifier Ensemble evaluation completed.


Evaluating Stacking Classifier Ensemble...
Classification report for Stacking Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.14      1.00      0.25         1
           1       0.00      0.00      0.00         6

    accuracy                           0.14         7
   macro avg       0.07      0.50      0.12         7
weighted avg       0.02      0.14      0.04         7

Precision: 0.0204, Recall: 0.1429, F1 Score: 0.0357
Confusion Matrix:
[[1 0]
 [6 0]]
```

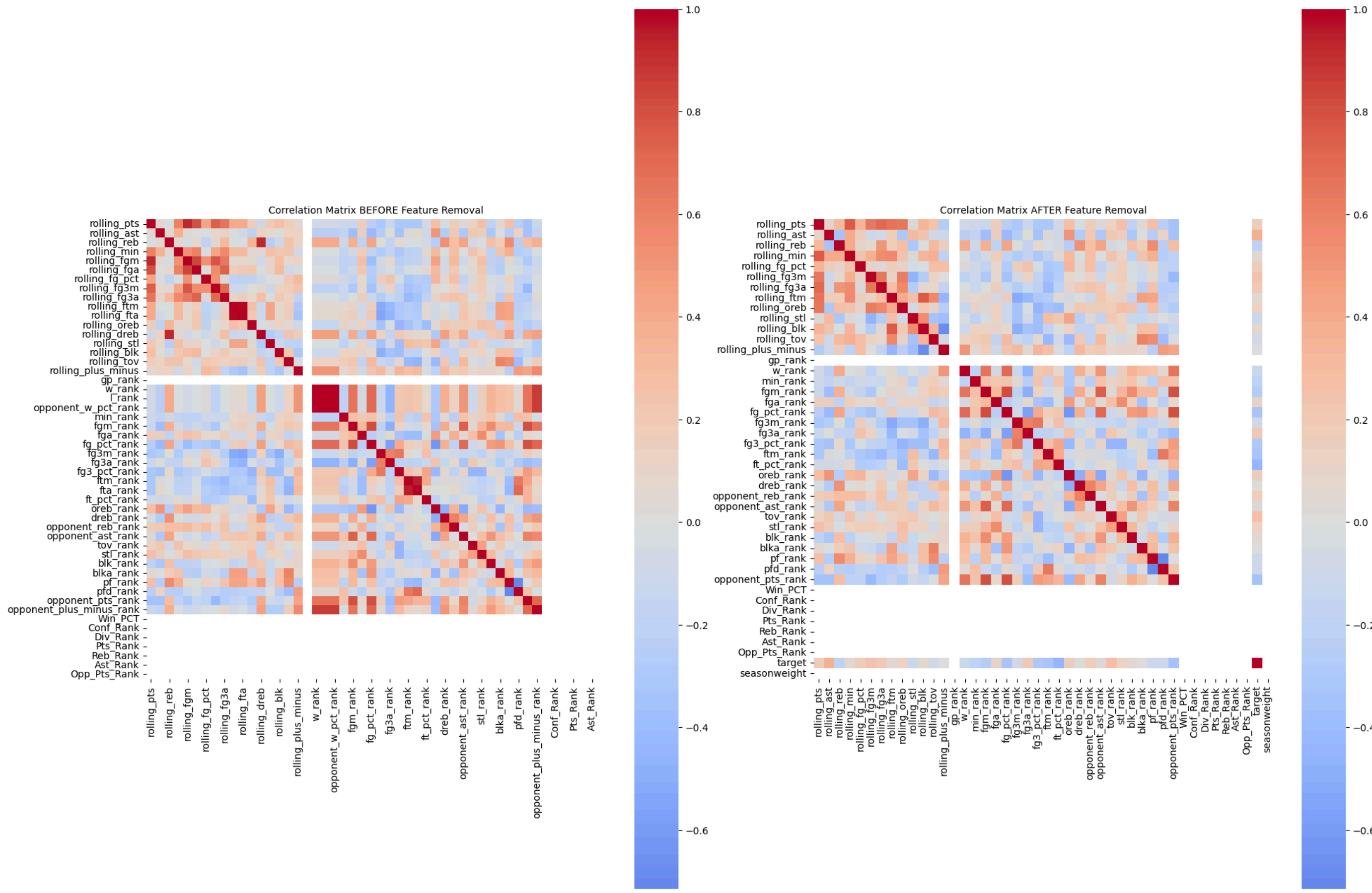






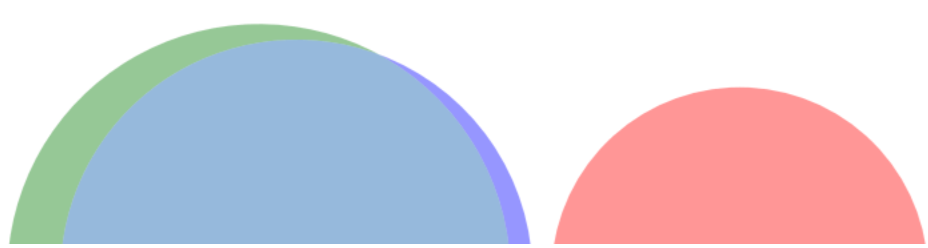

```
Feature importance not available for Stacking Classifier Ensemble.
Number of Misclassified Samples: 6
[INFO]: Stacking Classifier Ensemble evaluation completed.



Blended Prediction for Next Game:
Prediction: Under
Probability of Under: 0.54
Probability of Over: 0.46
Prediction for pts (stat line 32.5): Under
Probability of Under: 0.54
Probability of Over: 0.46

--- Processing Trae Young ---
1629027

Next Game Details:
GAME_DATE            DEC 14, 2024
HOME_TEAM_NAME          Milwaukee
VISITOR_TEAM_NAME         Atlanta
GAME_TIME                04:30 PM
Name: 0, dtype: object

[Debug] Starting dataset preparation...
Columns in fetched player game log for season 2024-25: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
Columns in fetched player game log for season 2023-24: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
```

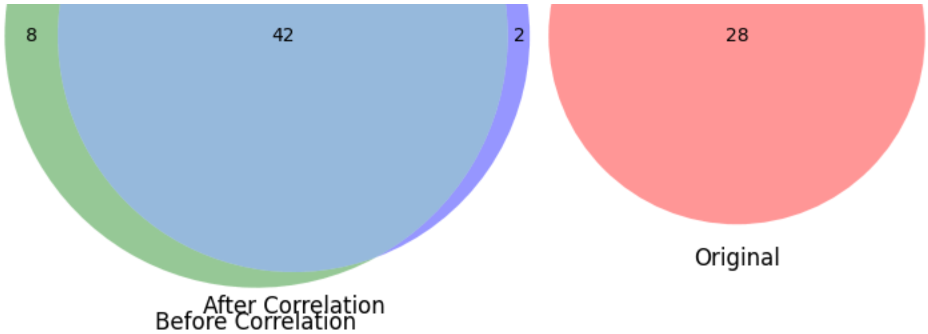
```
Fetched player data shape: (79, 28)

Rolling averages calculated successfully.
Fetched team game logs for season 2024-25, shape: (726, 57)

[Debug] Combined features shape: (79, 50)
[Debug] Combined features columns: Index(['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min',
       'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
       'rolling_fg3a', 'rolling_ftm', 'rolling_fta', 'rolling_oreb',
       'rolling_dreb', 'rolling_stl', 'rolling_blk', 'rolling_tov',
       'rolling_plus_minus', 'gp_rank', 'w_rank', 'l_rank',
       'opponent_w_pct_rank', 'min_rank', 'fgm_rank', 'fga_rank',
       'fg_pct_rank', 'fg3m_rank', 'fg3a_rank', 'fg3_pct_rank', 'ftm_rank',
       'fta_rank', 'ft_pct_rank', 'oreb_rank', 'dreb_rank',
       'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank', 'stl_rank',
       'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank', 'opponent_pts_rank',
       'opponent_plus_minus_rank', 'Win_PCT', 'Conf_Rank', 'Div_Rank',
       'Pts_Rank', 'Reb_Rank', 'Ast_Rank', 'Opp_Pts_Rank'],
      dtype='object')
Removing 8 highly correlated features: ['rolling_fgm', 'rolling_fga', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_p
```



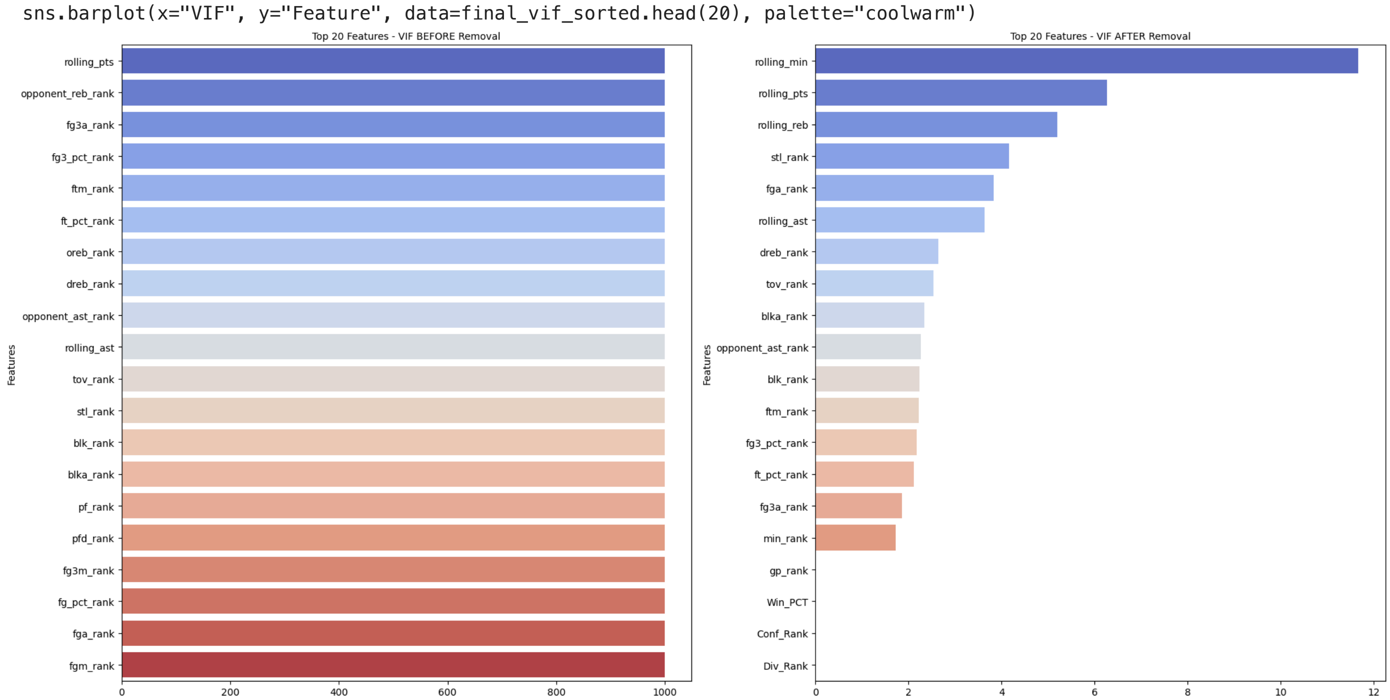

```
--- Feature Analysis ---
Original Features Count: 28
Features Before Correlation Removal: 50
Features After Correlation Removal: 44

--- Dropped Features ---
['rolling_fgm', 'rolling_fga', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fta_rank', 'opponent_plus_mi

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_fg_pct', 'rolling_fg3m', 'rolling_fg3a', 'rolling_ftm'
```

Feature Set Comparison



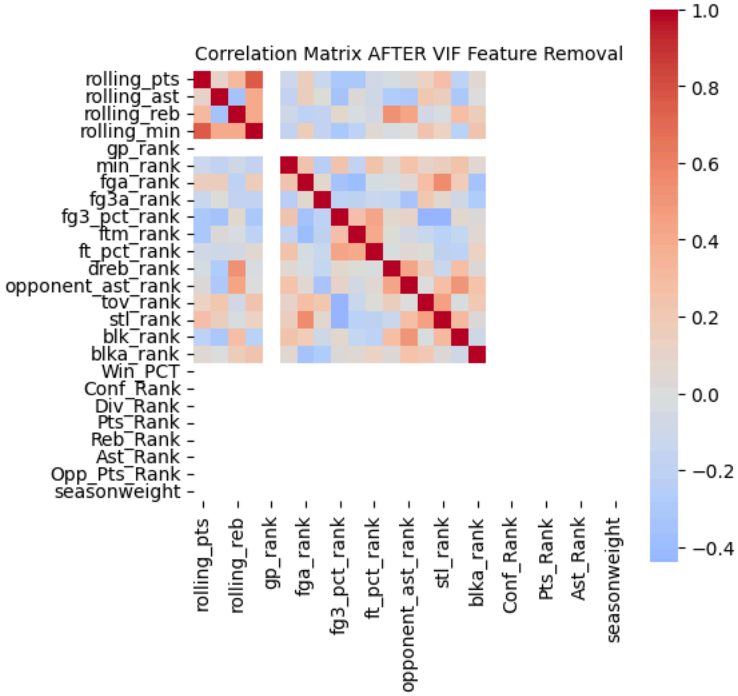


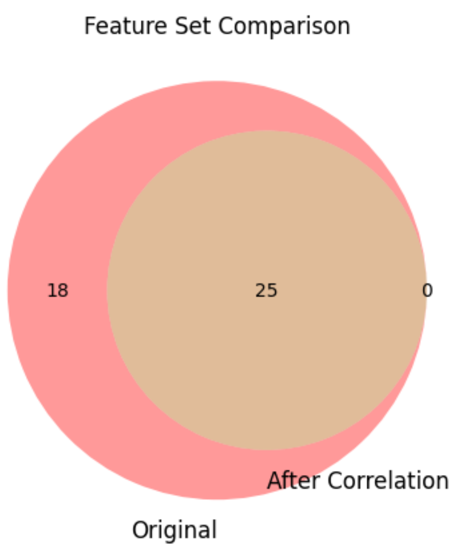

```
[INFO]: Starting feature scaling and preparation.
[INFO]: Feature scaling completed.
<ipython-input-111-9e74f1c2681d>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
No highly correlated features found to remove.
[INFO]: Dropped highly correlated features: []
Removing rolling_fg_pct with VIF: inf
Removing rolling_fg3m with VIF: inf
Removing rolling_fg3a with VIF: inf
Removing rolling_ftm with VIF: inf
Removing rolling_oreb with VIF: inf
Removing rolling_stl with VIF: inf
Removing rolling_blk with VIF: inf
Removing rolling_tov with VIF: inf
Removing rolling_plus_minus with VIF: inf
Removing w_rank with VIF: inf
Removing opponent_pts_rank with VIF: 11085.21
Removing fgm_rank with VIF: 534.02
Removing fg3m_rank with VIF: 119.37
Removing opponent_reb_rank with VIF: 42.15
Removing oreb_rank with VIF: 18.29
Removing fg_pct_rank with VIF: 17.39
Removing pfd_rank with VIF: 11.05
Removing pf_rank with VIF: 8.16
[INFO]: Features retained after VIF reduction: ['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'min_
<ipython-input-111-9e74f1c2681d>:53: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
```




```
--- Feature Analysis ---
Original Features Count: 43
Features After Correlation Removal: 25

--- Dropped Features ---
Correlation Dropped Features: []
```

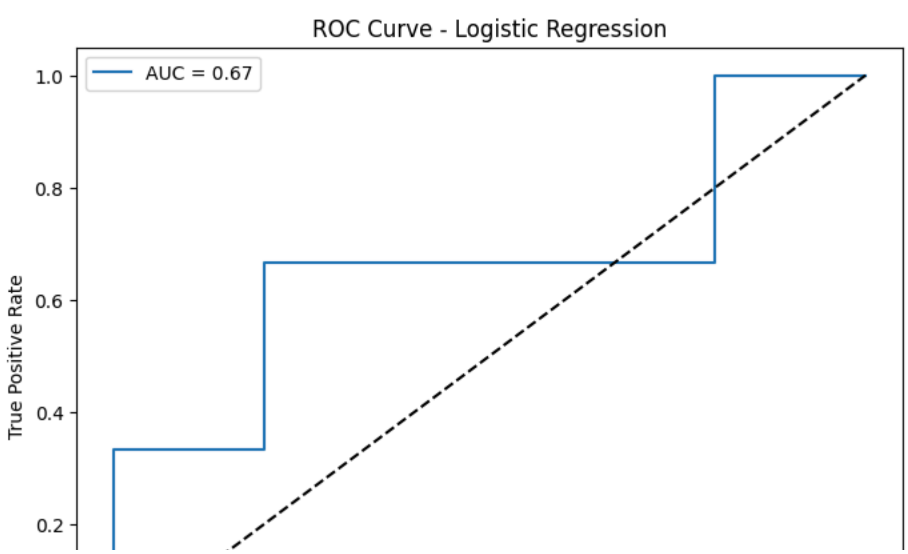
```
Correlation Dropped Features: []

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'min_rank', 'fga_rank', 'fg3a_rank', 'fg3_pct_rank',
```

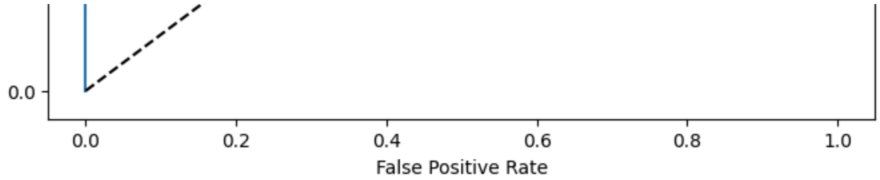


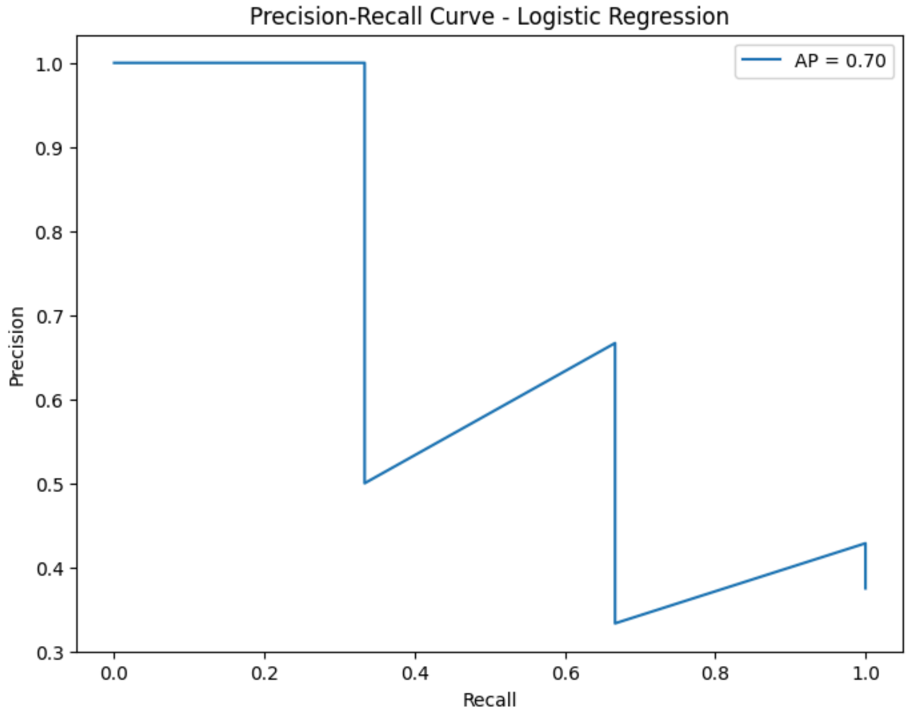



```
[INFO]: Calculated VIF data.
[INFO]: Initial VIF Data:
[DEBUG]:                Feature        VIF
3          rolling_min  11.672095
0          rolling_pts   6.268228
2          rolling_reb   5.192899
14            stl_rank   4.162409
6             fga_rank   3.827188
1          rolling_ast   3.642524
11           dreb_rank   2.641813
13            tov_rank   2.533123
16           blka_rank   2.334574
12   opponent_ast_rank   2.267486
15            blk_rank   2.243172
9             ftm_rank   2.218942
8         fg3_pct_rank   2.175257
10         ft_pct_rank   2.116735
7            fg3a_rank   1.860442
5             min_rank   1.719892
4              gp_rank        NaN
17             Win_PCT        NaN
18           Conf_Rank        NaN
19            Div_Rank        NaN
20            Pts_Rank        NaN
21            Reb_Rank        NaN
22            Ast_Rank        NaN
23        Opp_Pts_Rank        NaN
```

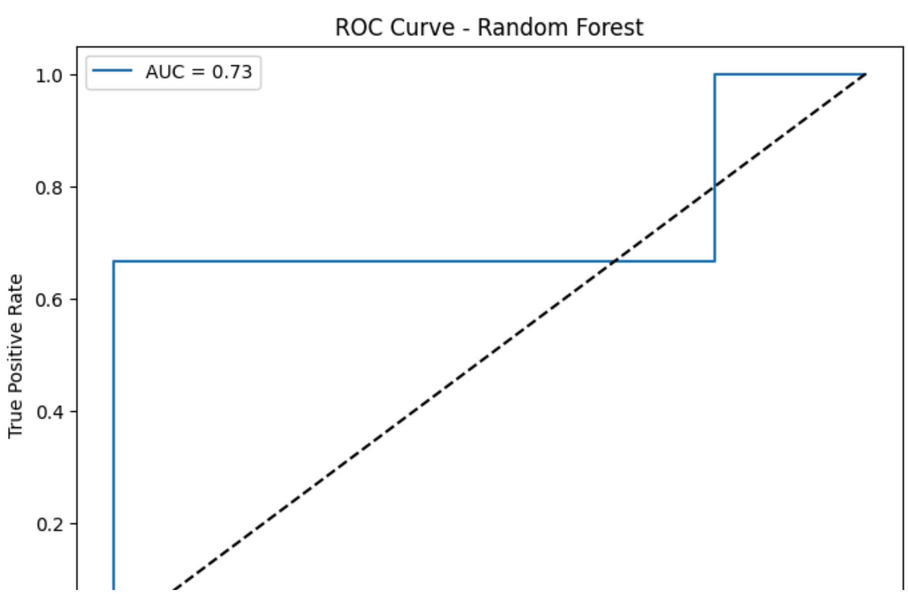
```
24       seasonweight        NaN
[INFO]: Preparing next_game_features for scaling and prediction.
[INFO]: Successfully scaled next_game_features.
[INFO]: Splitting data into training and validation sets.
[INFO]: Training set shape: (17, 25), Validation set shape: (8, 25)
[INFO]: Training Naive Model.
[INFO]: Training Logistic Regression model.
[INFO]: Training Random Forest model with cross-validation.
[INFO]: Performing hyperparameter optimization for Random Forest.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[INFO]: Random Forest training and optimization completed.
[INFO]: Training Gradient Boosting model.
[INFO]: Training Neural Network model with randomized hyperparameter search.
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[INFO]: Neural Network training and optimization completed.
[INFO]: Neural Network training and optimization completed.
[INFO]: Training Weighted Voting Classifier.
Stacking Classifier Performance:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.17      0.33      0.22         3

    accuracy                           0.12         8
   macro avg       0.08      0.17      0.11         8
weighted avg       0.06      0.12      0.08         8


Voting Classifier Performance:
              precision    recall  f1-score   support

           0       0.67      0.40      0.50         5
           1       0.40      0.67      0.50         3

    accuracy                           0.50         8
   macro avg       0.53      0.53      0.50         8
weighted avg       0.57      0.50      0.50         8

[INFO]: Evaluating Naive Model.
[INFO]: Naive Model Accuracy: 0.3750
[INFO]: Naive Model Classification Report:
[INFO]: {'0': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 5.0}, '1': {'precision': 0.375, 'recall': 1.0, '
[INFO]: Naive Model Confusion Matrix:
[INFO]: [[0 5]
 [0 3]]
[INFO]: Naive Model evaluation completed.
Evaluating Logistic Regression...
Classification report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.67      0.40      0.50         5
           1       0.40      0.67      0.50         3

    accuracy                           0.50         8
   macro avg       0.53      0.53      0.50         8
weighted avg       0.57      0.50      0.50         8

Precision: 0.5667, Recall: 0.5000, F1 Score: 0.5000
Confusion Matrix:
[[2 3]
 [1 2]]
```

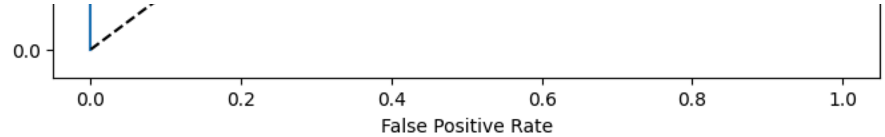


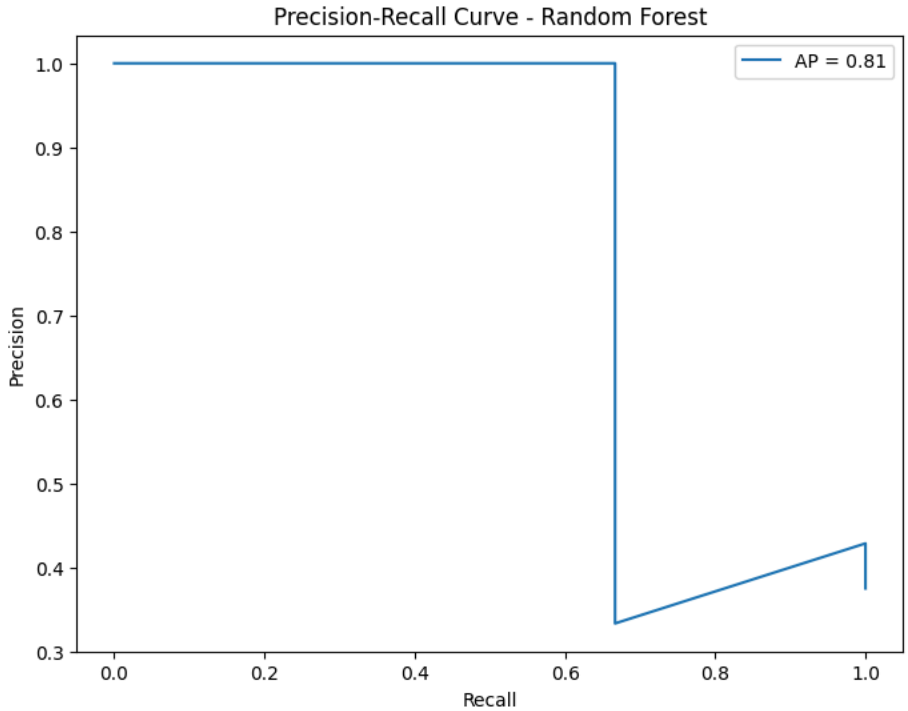


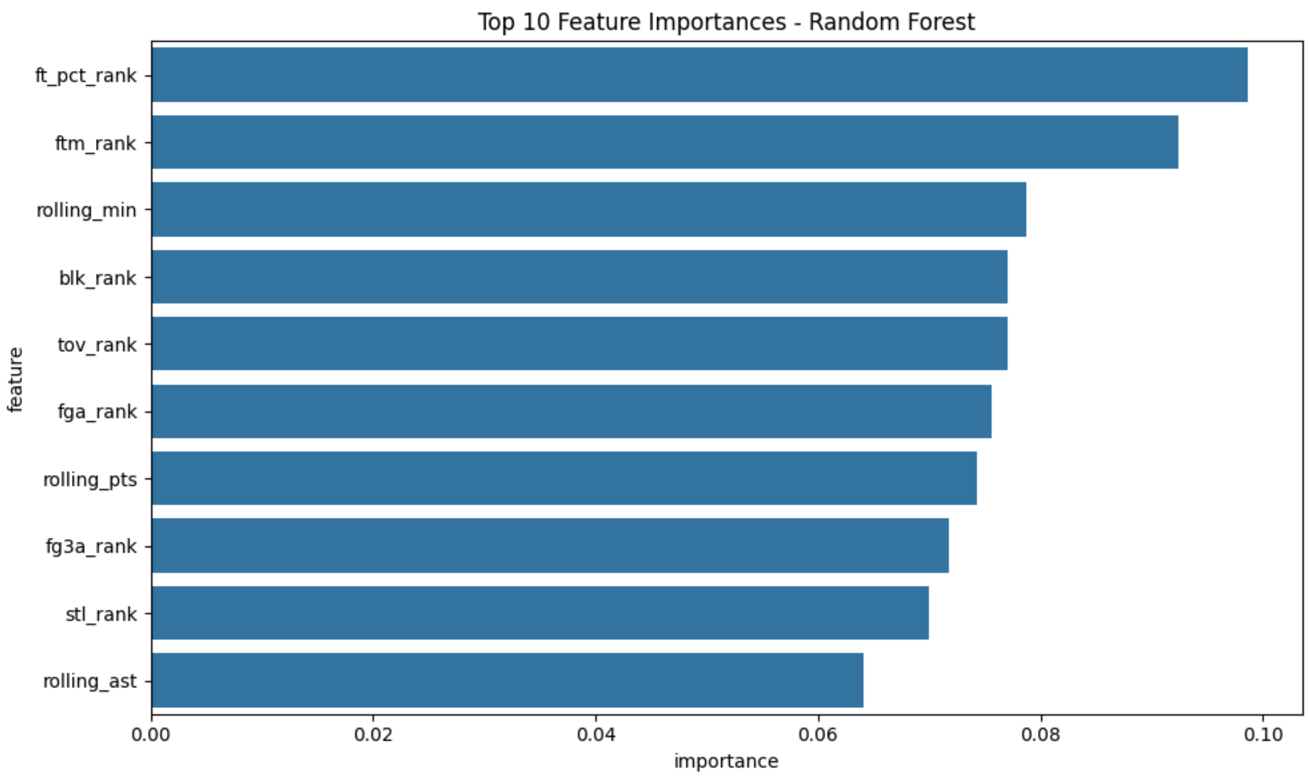



```
Feature importance not available for Logistic Regression.
Number of Misclassified Samples: 4
[INFO]: Logistic Regression evaluation completed.


Evaluating Random Forest...
Classification report for Random Forest:
              precision    recall  f1-score   support

           0       1.00      0.20      0.33         5
           1       0.43      1.00      0.60         3

    accuracy                           0.50         8
   macro avg       0.71      0.60      0.47         8
weighted avg       0.79      0.50      0.43         8

Precision: 0.7857, Recall: 0.5000, F1 Score: 0.4333
Confusion Matrix:
[[1 4]
 [0 3]]
```



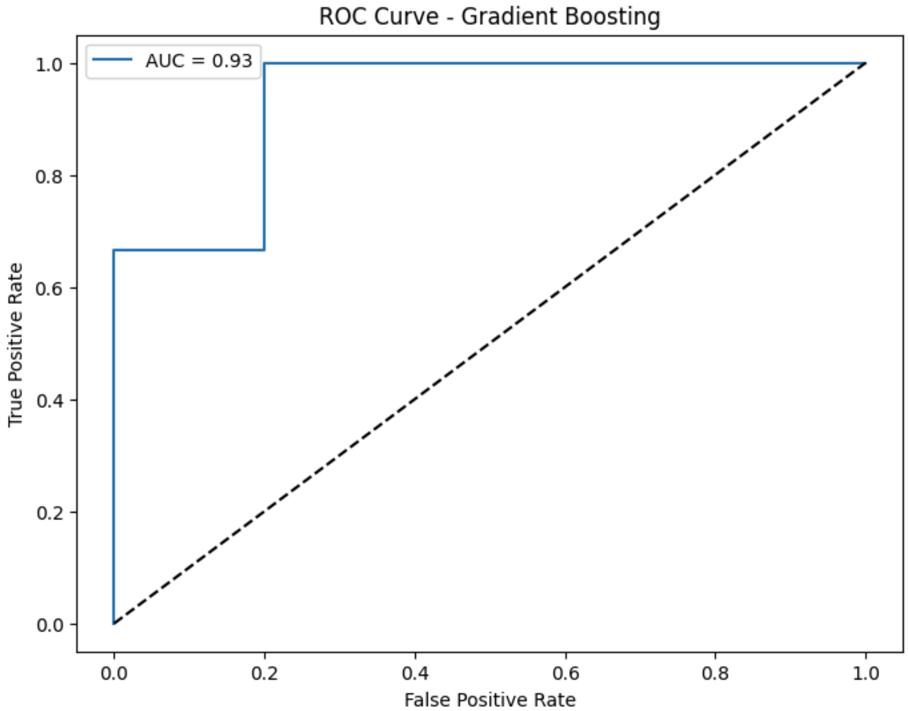


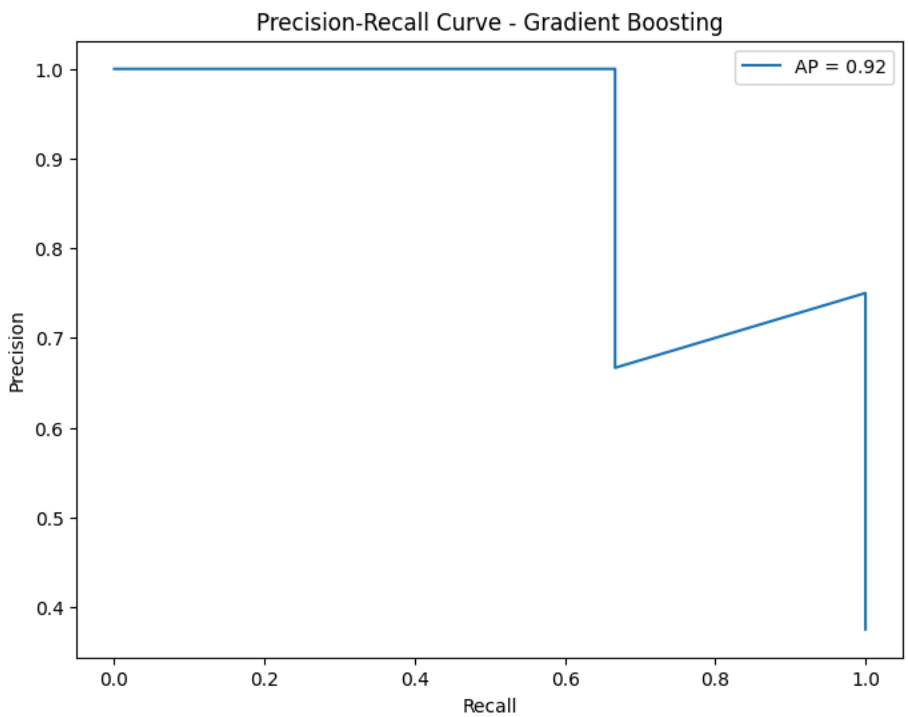


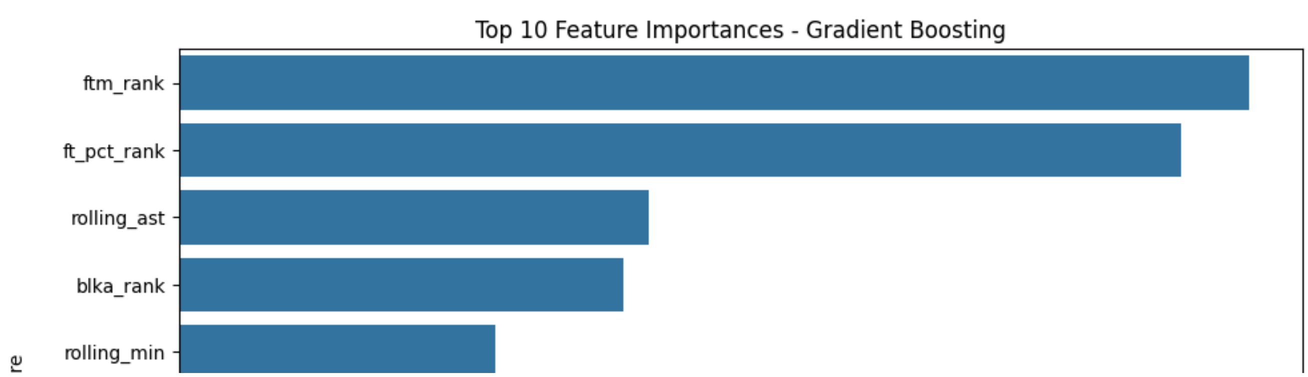


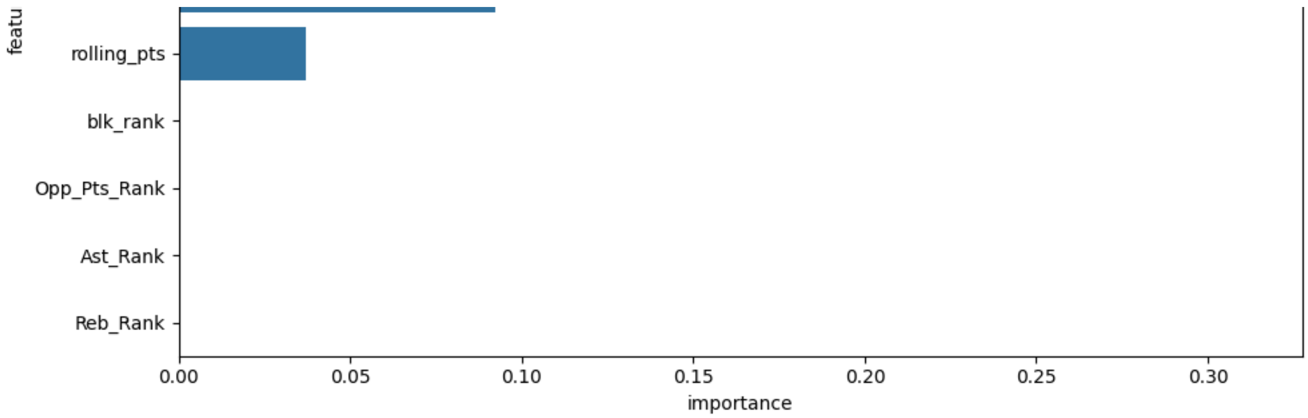

```
Number of Misclassified Samples: 4
[INFO]: Random Forest evaluation completed.


Evaluating Gradient Boosting...
Classification report for Gradient Boosting:
              precision    recall  f1-score   support

           0       1.00      0.60      0.75         5
           1       0.60      1.00      0.75         3

    accuracy                           0.75         8
   macro avg       0.80      0.80      0.75         8
```

```
weighted avg       0.85      0.75      0.75         8

Precision: 0.8500, Recall: 0.7500, F1 Score: 0.7500
Confusion Matrix:
[[3 2]
 [0 3]]
```

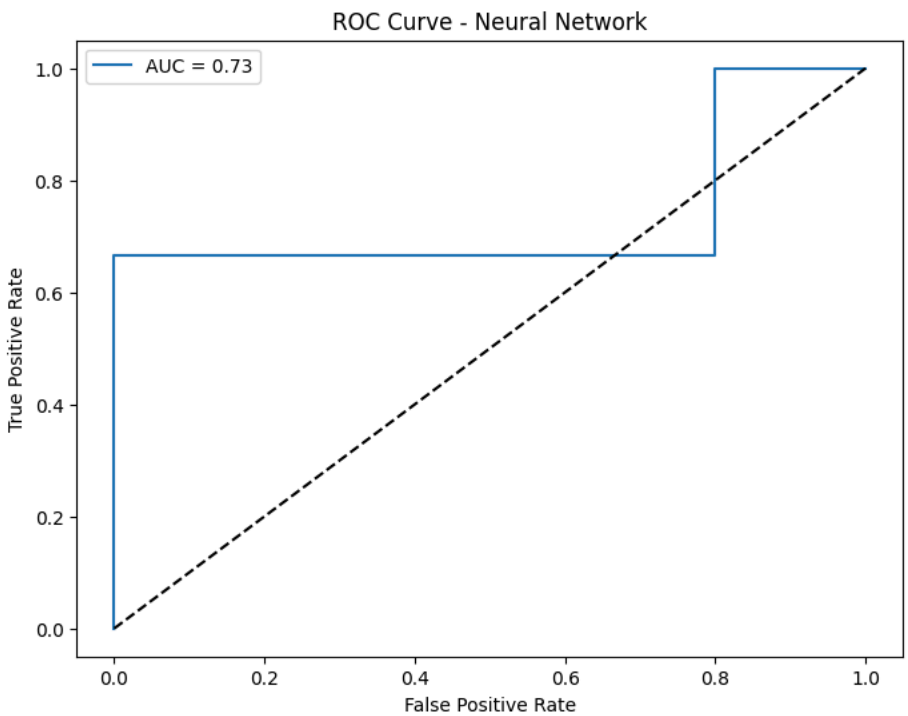


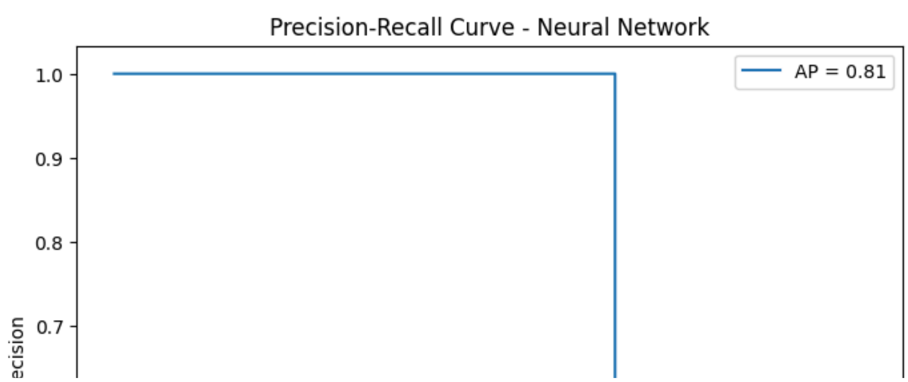


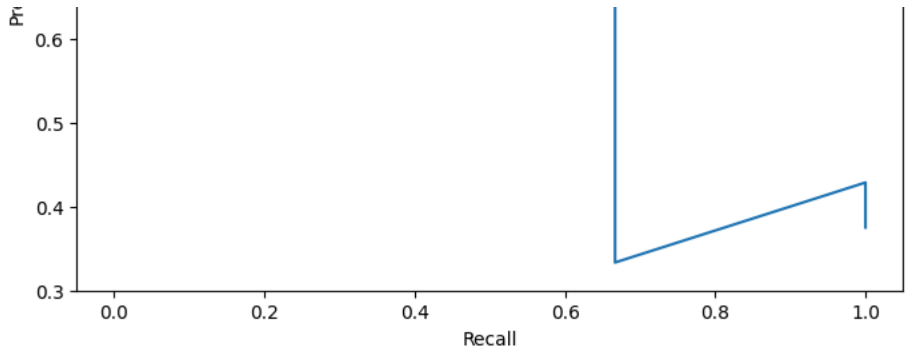




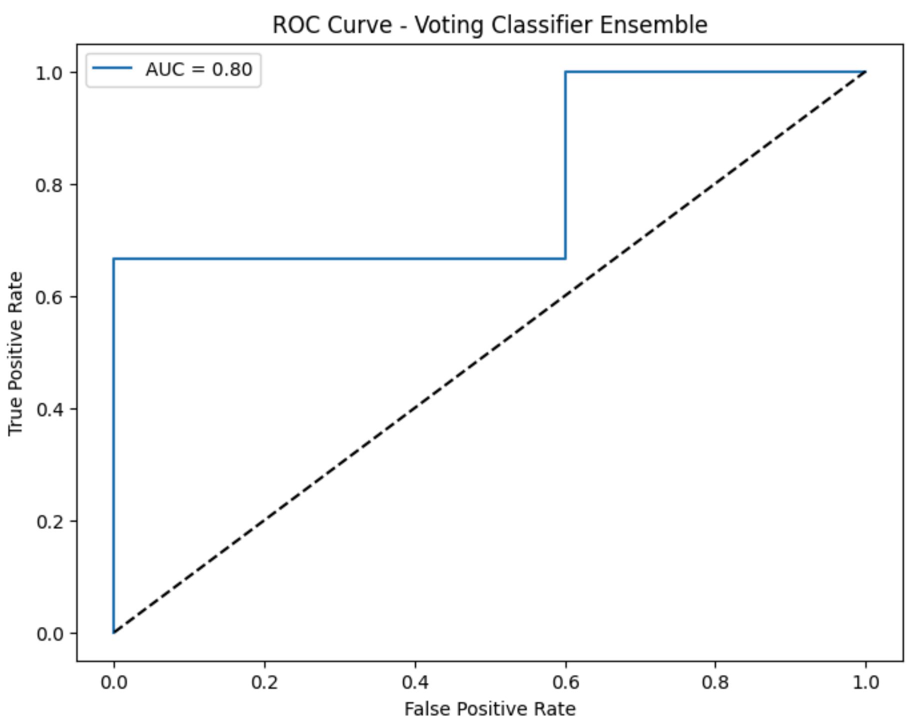

```
Number of Misclassified Samples: 2
[INFO]: Gradient Boosting evaluation completed.


Evaluating Neural Network...
Classification report for Neural Network:
              precision    recall  f1-score   support

           0       0.50      0.20      0.29         5
           1       0.33      0.67      0.44         3

    accuracy                           0.38         8
   macro avg       0.42      0.43      0.37         8
weighted avg       0.44      0.38      0.35         8

Precision: 0.4375, Recall: 0.3750, F1 Score: 0.3452
Confusion Matrix:
[[1 4]
 [1 2]]
```

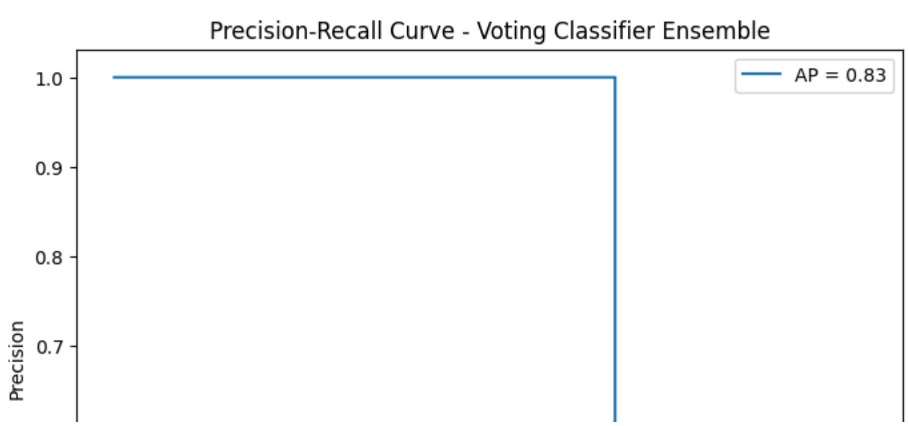


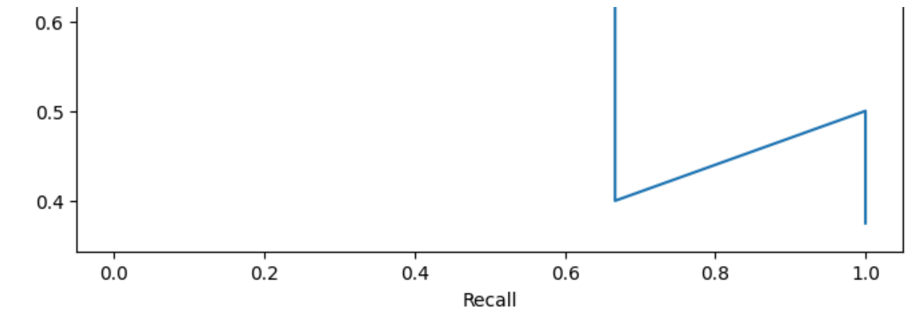




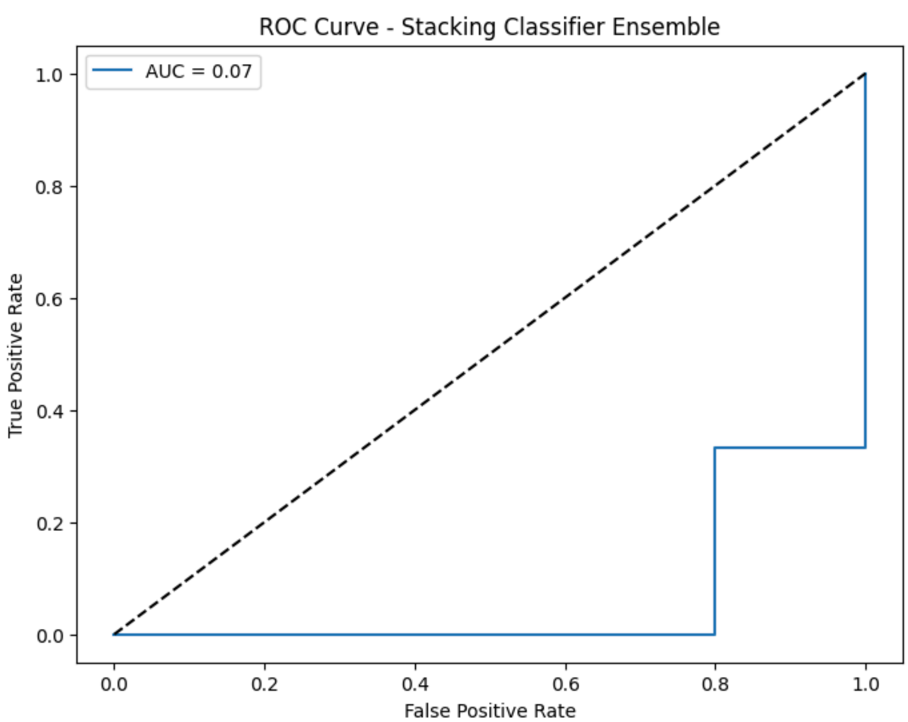

```
Feature importance not available for Neural Network.
Number of Misclassified Samples: 5
[INFO]: Neural Network evaluation completed.


Evaluating Voting Classifier Ensemble...
Classification report for Voting Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.67      0.40      0.50         5
           1       0.40      0.67      0.50         3

    accuracy                           0.50         8
   macro avg       0.53      0.53      0.50         8
weighted avg       0.57      0.50      0.50         8

Precision: 0.5667, Recall: 0.5000, F1 Score: 0.5000
Confusion Matrix:
[[2 3]
 [1 2]]
```

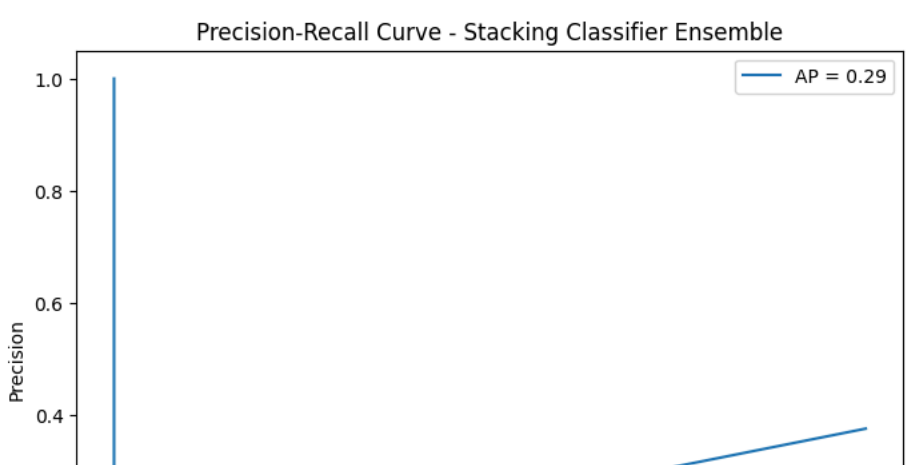


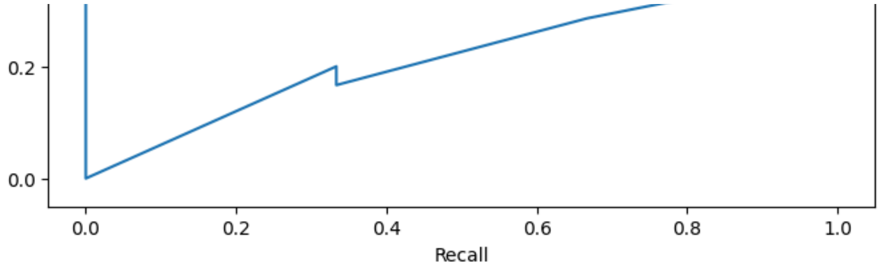




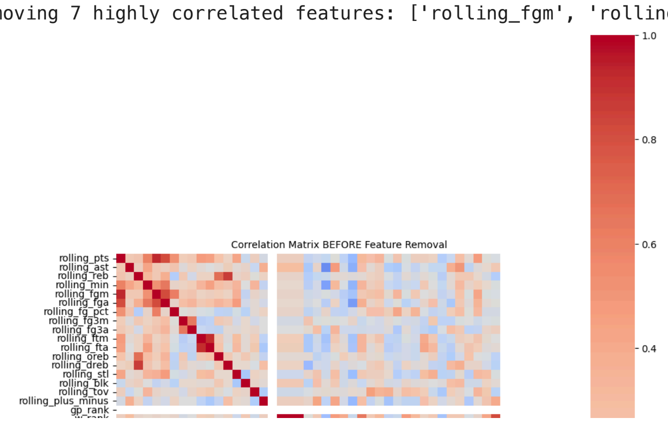

```
Feature importance not available for Voting Classifier Ensemble.
Number of Misclassified Samples: 4
[INFO]: Voting Classifier Ensemble evaluation completed.


Evaluating Stacking Classifier Ensemble...
Classification report for Stacking Classifier Ensemble:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         5
           1       0.17      0.33      0.22         3

    accuracy                           0.12         8
   macro avg       0.08      0.17      0.11         8
weighted avg       0.06      0.12      0.08         8

Precision: 0.0625, Recall: 0.1250, F1 Score: 0.0833
Confusion Matrix:
[[0 5]
 [2 1]]
```

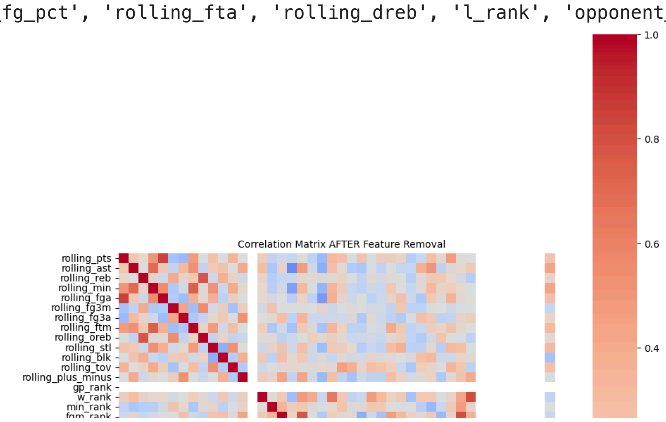


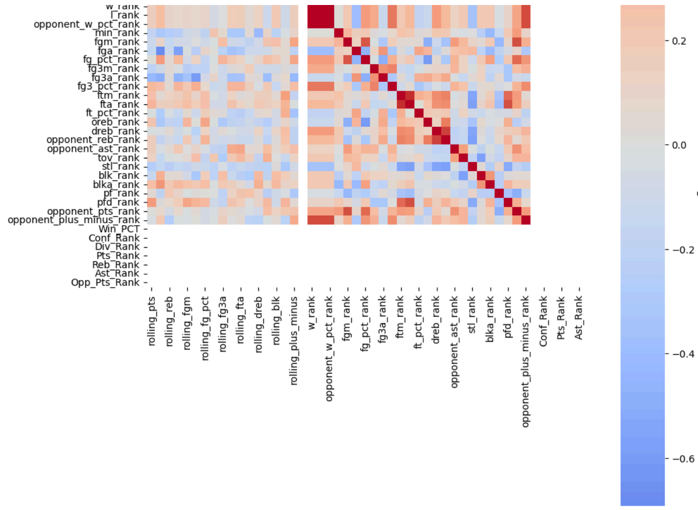


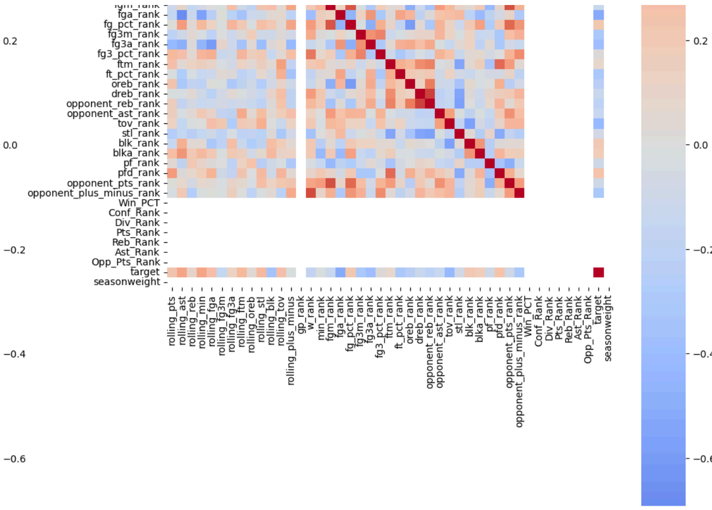



```
Feature importance not available for Stacking Classifier Ensemble.
Number of Misclassified Samples: 7
[INFO]: Stacking Classifier Ensemble evaluation completed.



Blended Prediction for Next Game:
Prediction: Over
Probability of Under: 0.28
Probability of Over: 0.72
Prediction for ast (stat line 11.5): Over
Probability of Under: 0.28
Probability of Over: 0.72

--- Processing Alperen Sengun ---
1630578

Next Game Details:
GAME_DATE            DEC 14, 2024
HOME_TEAM_NAME       Oklahoma City
VISITOR_TEAM_NAME          Houston
GAME_TIME                 08:30 PM
Name: 0, dtype: object

[Debug] Starting dataset preparation...
Columns in fetched player game log for season 2024-25: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')
Columns in fetched player game log for season 2023-24: Index(['season_id', 'player_id', 'game_id', 'game_date', 'matchup', '
       'min', 'fgm', 'fga', 'fg_pct', 'fg3m', 'fg3a', 'fg3_pct', 'ftm', 'fta',
       'ft_pct', 'oreb', 'dreb', 'reb', 'ast', 'stl', 'blk', 'tov', 'pf',
       'pts', 'plus_minus', 'video_available'],
      dtype='object')

Fetched player data shape: (88, 28)

Rolling averages calculated successfully.
Fetched team game logs for season 2024-25, shape: (726, 57)

[Debug] Combined features shape: (88, 50)
[Debug] Combined features columns: Index(['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min',
       'rolling_fgm', 'rolling_fga', 'rolling_fg_pct', 'rolling_fg3m',
       'rolling_fg3a', 'rolling_ftm', 'rolling_fta', 'rolling_oreb',
       'rolling_dreb', 'rolling_stl', 'rolling_blk', 'rolling_tov',
       'rolling_plus_minus', 'gp_rank', 'w_rank', 'l_rank',
       'opponent_w_pct_rank', 'min_rank', 'fgm_rank', 'fga_rank',
       'fg_pct_rank', 'fg3m_rank', 'fg3a_rank', 'fg3_pct_rank', 'ftm_rank',
       'fta_rank', 'ft_pct_rank', 'oreb_rank', 'dreb_rank',
       'opponent_reb_rank', 'opponent_ast_rank', 'tov_rank', 'stl_rank',
       'blk_rank', 'blka_rank', 'pf_rank', 'pfd_rank', 'opponent_pts_rank',
       'opponent_plus_minus_rank', 'Win_PCT', 'Conf_Rank', 'Div_Rank',
       'Pts_Rank', 'Reb_Rank', 'Ast_Rank', 'Opp_Pts_Rank'],
      dtype='object')
Removing 7 highly correlated features: ['rolling_fgm', 'rolling_fg_pct', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_
```

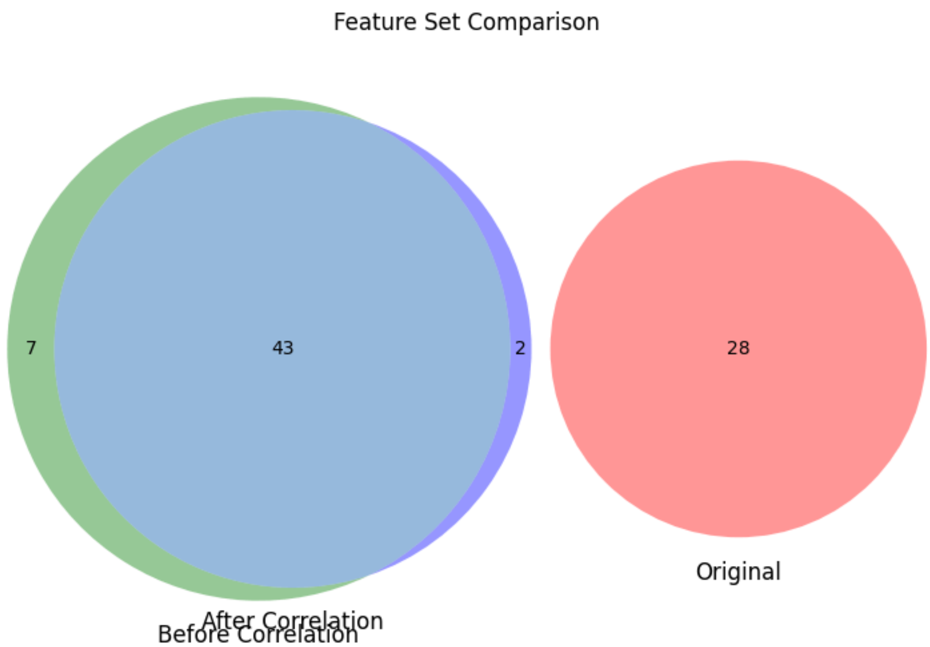






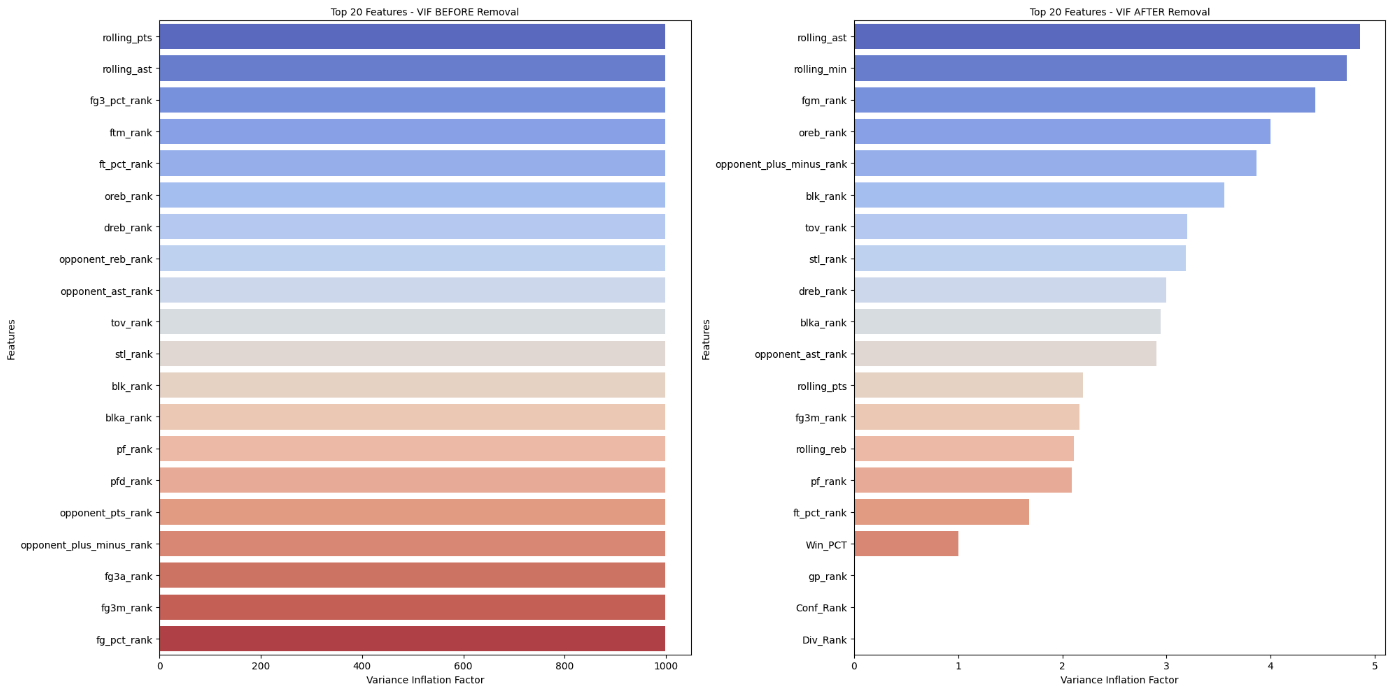


```
--- Feature Analysis ---
Original Features Count: 28
Features Before Correlation Removal: 50
Features After Correlation Removal: 45

--- Dropped Features ---
['rolling_fgm', 'rolling_fg_pct', 'rolling_fta', 'rolling_dreb', 'l_rank', 'opponent_w_pct_rank', 'fta_rank']

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'rolling_fga', 'rolling_fg3m', 'rolling_fg3a', 'rolling_ftm', '
```

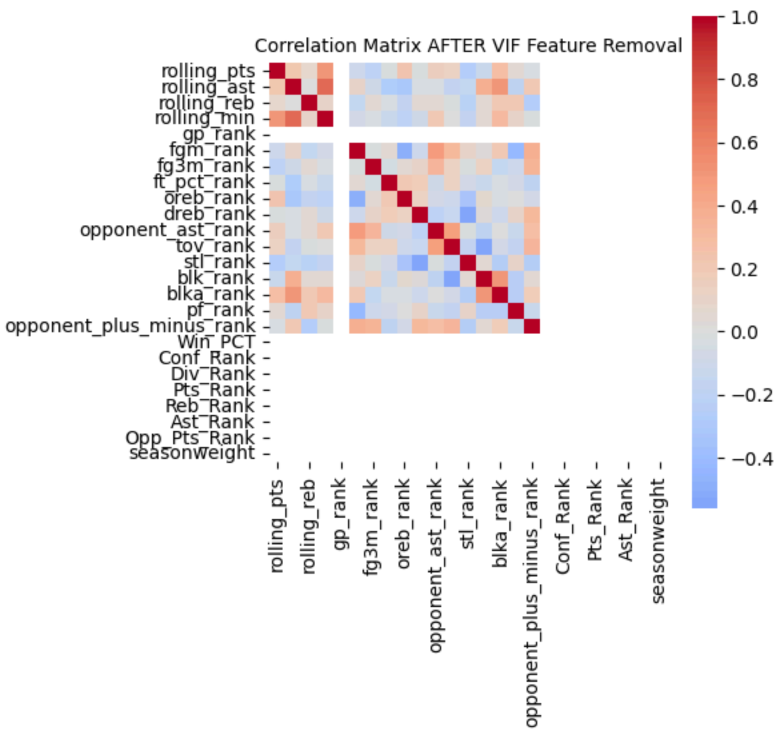



```
[INFO]: Starting feature scaling and preparation.
[INFO]: Feature scaling completed.
<ipython-input-111-9e74f1c2681d>:33: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=initial_vif_sorted.head(20), palette="coolwarm")
No highly correlated features found to remove.
[INFO]: Dropped highly correlated features: []
Removing rolling_fga with VIF: inf
Removing rolling_fg3m with VIF: inf
Removing rolling_fg3a with VIF: inf
Removing rolling_ftm with VIF: inf
Removing rolling_oreb with VIF: inf
Removing rolling_stl with VIF: inf
Removing rolling_blk with VIF: inf
Removing rolling_tov with VIF: inf
Removing rolling_plus_minus with VIF: inf
Removing w_rank with VIF: inf
Removing min_rank with VIF: inf
Removing opponent_pts_rank with VIF: 716.51
Removing fg_pct_rank with VIF: 414.34
Removing opponent_reb_rank with VIF: 101.56
```

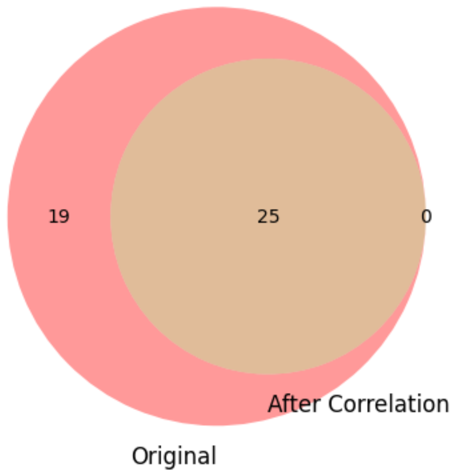
```
Removing fg3a_rank with VIF: 38.51
Removing pfd_rank with VIF: 18.53
Removing fga_rank with VIF: 17.71
Removing fg3_pct_rank with VIF: 12.63
Removing ftm_rank with VIF: 5.66
[INFO]: Features retained after VIF reduction: ['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fgm_
<ipython-input-111-9e74f1c2681d>:53: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and

  sns.barplot(x="VIF", y="Feature", data=final_vif_sorted.head(20), palette="coolwarm")
```




```
--- Feature Analysis ---
Original Features Count: 44
Features After Correlation Removal: 25

--- Dropped Features ---
Correlation Dropped Features: []

--- Remaining Features ---
['rolling_pts', 'rolling_ast', 'rolling_reb', 'rolling_min', 'gp_rank', 'fgm_rank', 'fg3m_rank', 'ft_pct_rank', 'oreb_rank',
```

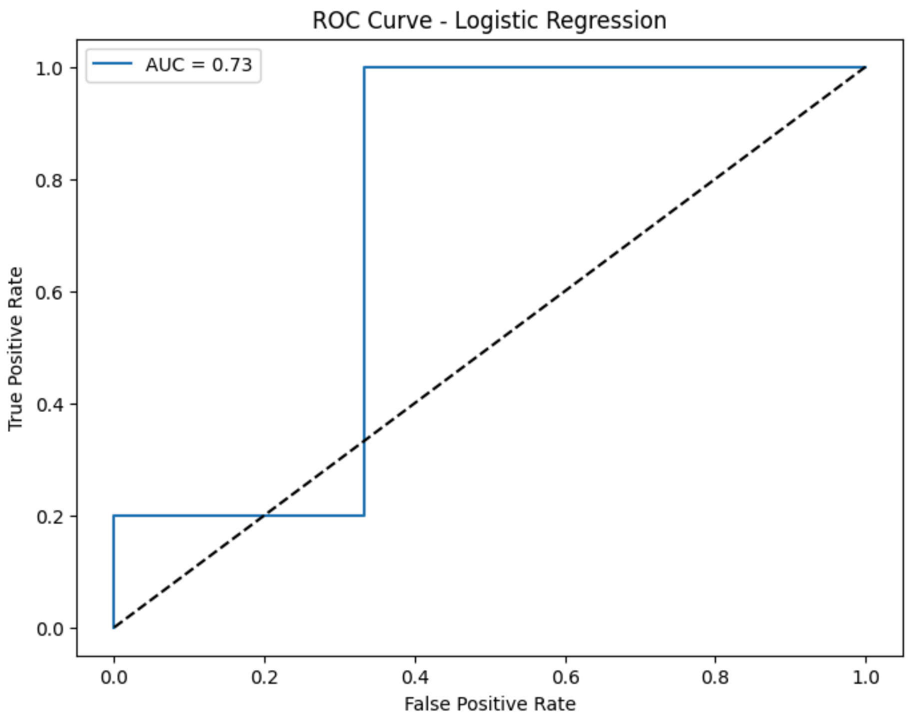


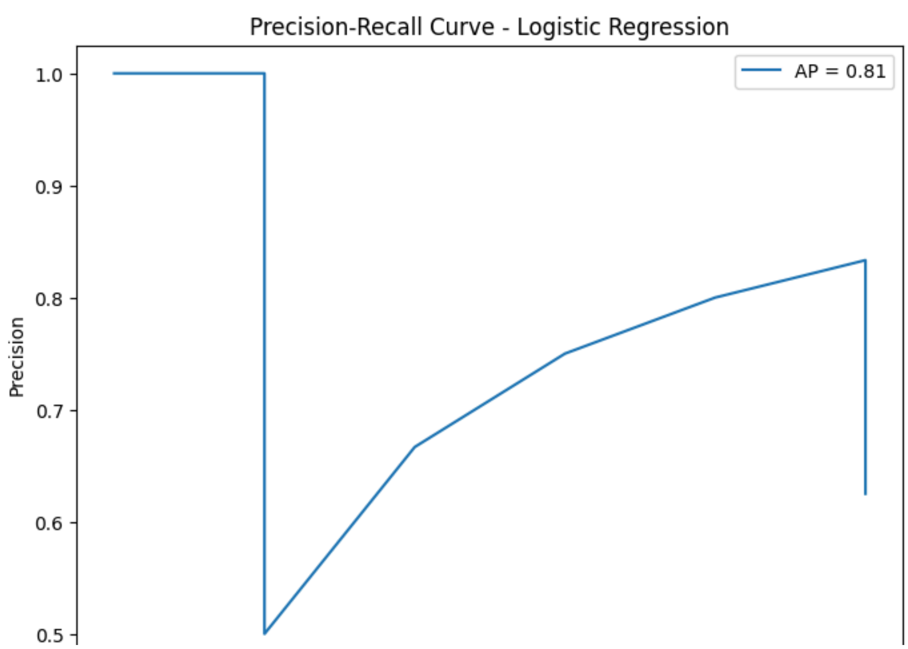
Feature Set Comparison

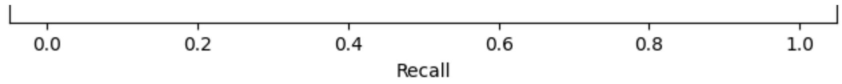



```
[INFO]: Calculated VIF data.
[INFO]: Initial VIF Data:
[DEBUG]:                       Feature       VIF
1                 rolling_ast  4.857982
3                 rolling_min  4.728853
5                    fgm_rank  4.429672
8                   oreb_rank  4.001910
16  opponent_plus_minus_rank  3.865039
13                   blk_rank  3.556291
11                   tov_rank  3.201641
12                   stl_rank  3.188202
9                   dreb_rank  2.997908
14                  blka_rank  2.943034
10         opponent_ast_rank  2.903818
0                 rolling_pts  2.196778
6                   fg3m_rank  2.161401
2                 rolling_reb  2.113190
15                    pf_rank  2.087191
7                 ft_pct_rank  1.680525
4                     gp_rank       NaN
17                    Win_PCT       NaN
18                  Conf_Rank       NaN
19                   Div_Rank       NaN
20                   Pts_Rank       NaN
21                   Reb_Rank       NaN
22                   Ast_Rank       NaN
23               Opp_Pts_Rank       NaN
24               seasonweight       NaN
[INFO]: Preparing next_game_features for scaling and prediction.
[INFO]: Successfully scaled next_game_features.
[INFO]: Splitting data into training and validation sets.
[INFO]: Training set shape: (17, 25), Validation set shape: (8, 25)
[INFO]: Training Naive Model.
[INFO]: Training Logistic Regression model.
[INFO]: Training Random Forest model with cross-validation.
[INFO]: Performing hyperparameter optimization for Random Forest.
Fitting 5 folds for each of 18 candidates, totalling 90 fits
[INFO]: Random Forest training and optimization completed.
[INFO]: Training Gradient Boosting model.
[INFO]: Training Neural Network model with randomized hyperparameter search.
Fitting 3 folds for each of 10 candidates, totalling 30 fits
[INFO]: Neural Network training and optimization completed.
[INFO]: Neural Network training and optimization completed.
[INFO]: Training Weighted Voting Classifier.
Stacking Classifier Performance:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.62      1.00      0.77         5

    accuracy                           0.62         8
   macro avg       0.31      0.50      0.38         8
weighted avg       0.39      0.62      0.48         8


Voting Classifier Performance:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.50      0.60      0.55         5

    accuracy                           0.38         8
   macro avg       0.25      0.30      0.27         8
```

```
   macro avg       0.25      0.30      0.27         8
weighted avg       0.31      0.38      0.34         8

[INFO]: Evaluating Naive Model.
[INFO]: Naive Model Accuracy: 0.6250
[INFO]: Naive Model Classification Report:
[INFO]: {'0': {'precision': 0.0, 'recall': 0.0, 'f1-score': 0.0, 'support': 3.0}, '1': {'precision': 0.625, 'recall': 1.0, '
[INFO]: Naive Model Confusion Matrix:
[INFO]: [[0 3]
 [0 5]]
[INFO]: Naive Model evaluation completed.
Evaluating Logistic Regression...
Classification report for Logistic Regression:
              precision    recall  f1-score   support

           0       1.00      0.67      0.80         3
           1       0.83      1.00      0.91         5

    accuracy                           0.88         8
   macro avg       0.92      0.83      0.85         8
weighted avg       0.90      0.88      0.87         8

Precision: 0.8958, Recall: 0.8750, F1 Score: 0.8682
Confusion Matrix:
[[2 1]
 [0 5]]
```








```
Feature importance not available for Logistic Regression.
Number of Misclassified Samples: 1
[INFO]: Logistic Regression evaluation completed.


Evaluating Random Forest...
Classification report for Random Forest:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.62      1.00      0.77         5

    accuracy                           0.62         8
   macro avg       0.31      0.50      0.38         8
weighted avg       0.39      0.62      0.48         8

Precision: 0.3906, Recall: 0.6250, F1 Score: 0.4808
Confusion Matrix:
[[0 3]
 [0 5]]
```