FinalProject

February 9, 2026

[48]:
```
import pandas as pd
import numpy as np
import re
from pathlib import Path
```

[49]:
```
DATA_DIR = Path(".")

season_files = [
    "nba-2018-EasternStandardTime.csv",
    "nba-2019-EasternStandardTime.csv",
    "nba-2020-UTC.csv",
    "nba-2021-UTC.csv",
    "nba-2022-UTC.csv",
    "nba-2023-UTC.csv",
    "nba-2024-UTC.csv",
]
```

[50]:
```
def infer_season_from_filename(filename: str) -> int:

    m = re.search(r"(19|20)\d{2}", filename)
    if not m:
        raise ValueError(f"Could not infer season year from filename:␣
 ↪{filename}")
    return int(m.group(0))
```

[51]:
```
def process_one_season(csv_path: Path) -> pd.DataFrame:
    df = pd.read_csv(csv_path)

    required_cols = {"Date", "Home Team", "Away Team", "Result"}
    missing = required_cols - set(df.columns)
    if missing:
        raise ValueError(f"{csv_path.name} is missing columns: {missing}")

    df["Date"] = pd.to_datetime(df["Date"], errors="coerce", dayfirst=True)
    if df["Date"].isna().any():
        # Some dates were failing so this is to show the debug
        bad_rows = df[df["Date"].isna()].head(5)
```

```
        raise ValueError(f"{csv_path.name}: Some 'Date' values could not be␣
↪parsed. Examples:\n{bad_rows}")

    # Parse "Result" -> HOME_PTS, AWAY_PTS
    # For the format: "119 - 107"
    scores = df["Result"].astype(str).str.split(" - ", expand=True)

    if scores.shape[1] != 2:
        scores = df["Result"].astype(str).str.extract(r"(\d+)\s*-\s*(\d+)")

    df["HOME_PTS"] = pd.to_numeric(scores[0], errors="coerce")
    df["AWAY_PTS"] = pd.to_numeric(scores[1], errors="coerce")

    # Remove rows without scores ( this would be for like future games)
    df = df.dropna(subset=["HOME_PTS", "AWAY_PTS"]).copy()
    df["HOME_PTS"] = df["HOME_PTS"].astype(int)
    df["AWAY_PTS"] = df["AWAY_PTS"].astype(int)

    # Our target label
    df["home_win"] = (df["HOME_PTS"] > df["AWAY_PTS"]).astype(int)

    # Add season from filename year
    df["SEASON"] = infer_season_from_filename(csv_path.name)

    out = df[["SEASON", "Date", "Home Team", "Away Team", "HOME_PTS",␣
↪"AWAY_PTS", "home_win"]].copy()

    out["Home Team"] = out["Home Team"].astype(str).str.strip()
    out["Away Team"] = out["Away Team"].astype(str).str.strip()

    return out
```

[52]:
```
all_seasons = []
for fname in season_files:
    path = DATA_DIR / fname
    if not path.exists():
        raise FileNotFoundError(f"Could not find: {path.resolve()}")

    season_df = process_one_season(path)
    print(f"{fname}: {len(season_df)} games")
    all_seasons.append(season_df)

games_all = pd.concat(all_seasons, ignore_index=True)

# pre sorting for later use
games_all = games_all.sort_values(["SEASON", "Date", "Home Team", "Away Team"]).
↪reset_index(drop=True)
```

```
print("Total games:", len(games_all))
games_all.head()
```

```
nba-2018-EasternStandardTime.csv: 1230 games
nba-2019-EasternStandardTime.csv: 1059 games
nba-2020-UTC.csv: 1079 games
nba-2021-UTC.csv: 1230 games
nba-2022-UTC.csv: 1230 games
nba-2023-UTC.csv: 1231 games
nba-2024-UTC.csv: 1231 games
Total games: 8290
```

[52]:

	SEASON	Date	Home Team	Away Team
0	2018	2018-10-16 20:00:00	Boston Celtics	Philadelphia 76ers
1	2018	2018-10-16 22:30:00	Golden State Warriors	Oklahoma City Thunder
2	2018	2018-10-17 19:00:00	Charlotte Hornets	Milwaukee Bucks
3	2018	2018-10-17 19:00:00	Detroit Pistons	Brooklyn Nets
4	2018	2018-10-17 19:00:00	Indiana Pacers	Memphis Grizzlies

	HOME_PTS	AWAY_PTS	home_win
0	105	87	1
1	108	100	1
2	112	113	0
3	103	100	1
4	111	83	1

Our total number of games in our dataset is 8290 !

[53]:
```
## Checking for dupes

dupes = games_all.duplicated(subset=["SEASON", "Date", "Home Team", "Away␣
↪Team"]).sum()
print("Duplicate games:", dupes)
```

```
Duplicate games: 0
```

[54]:
```
games_all.to_csv("nba_games_2018_2024_clean.csv", index=False)
print("Saved: nba_games_2018_2024_clean.csv")
```

```
Saved: nba_games_2018_2024_clean.csv
```

[55]:
```
# games per season
print(games_all["SEASON"].value_counts().sort_index())

# home team win rate
print("Home win rate:", games_all["home_win"].mean())
```

```
SEASON
2018    1230
```

```
2019    1059
2020    1079
2021    1230
2022    1230
2023    1231
2024    1231
Name: count, dtype: int64
Home win rate: 0.5574185765983112
```

Here I am collecting all of the data from the source: https://fixturedownload.com/results/ That has csv results of all games from all the seasons, we can see that 2019 and 2020 differ, that is due to Covid-19 shutting down the season early. Also, in 2023 the NBA In Season Tournament was created which added 1 extra game.

Then we do a at a glance check. To see that the home team won 55.7% of games in our dataset, consistent with known home-court advantage in the sports.

0.1 Next: We are going to handle the Advanced Stats data for 2018-2025

[56]:
```
DATA_DIR = Path(".")

# mapping our filename
season_map = {
    "18-19 Advanced Stats.csv": 2018,
    "19-20 Advanced Stats.csv": 2019,
    "20-21 Advanced Stats.csv": 2020,
    "21-22 Advanced Stats.csv": 2021,
    "22-23 Advanced Stats.csv": 2022,
    "23-24 Advanced Stats.csv": 2023,
    "24-25 Advanced Stats.csv": 2024,
}
```

[57]:
```
KEEP_COLS = ["Team", "ORtg", "DRtg", "NRtg", "SRS", "Pace", "TOV%", "TS%"]

def load_one_team_adv(file_path, season_year: int) -> pd.DataFrame:
    # Reading raw lines to find the real header row that contains "Rk" and␣
 ↪"Team"
    with open(file_path, "r", encoding="utf-8", errors="ignore") as f:
        lines = f.readlines()

    header_idx = None
    for i, line in enumerate(lines):
        # Basketball Reference header row seems to start with "Rk,Team"
        if line.strip().startswith("Rk,Team"):
            header_idx = i
            break

    if header_idx is None:
```

```
        raise ValueError(f"Could not find header row starting with 'Rk,Team' in␣
↪{file_path}")

    df = pd.read_csv(file_path, skiprows=header_idx)

    # Droping blank rows + League Average which BBallRefrence Includes but isnt␣
↪useful to us
    df = df[df["Team"].notna()].copy()
    df = df[~df["Team"].astype(str).str.contains("League Average", na=False)].
↪copy()

    # removing playoff *'s from some of the team names
    df["Team"] = df["Team"].astype(str).str.replace("*", "", regex=False).str.
↪strip()

    missing = [c for c in KEEP_COLS if c not in df.columns]
    if missing:
        raise ValueError(
            f"{file_path} is missing columns: {missing}\n"
            f"Columns found:\n{list(df.columns)}"
        )

    df = df[KEEP_COLS].copy()
    df["SEASON"] = season_year

    # Convert numeric columns
    for c in KEEP_COLS:
        if c != "Team":
            df[c] = pd.to_numeric(df[c], errors="coerce")

    return df
```

[58]:
```
team_frames = []
for fname, season_year in season_map.items():
    path = DATA_DIR / fname
    temp = load_one_team_adv(path, season_year)
    print(fname, "-> rows:", len(temp))
    team_frames.append(temp)

team_adv_all = pd.concat(team_frames, ignore_index=True)

print("Total team rows:", len(team_adv_all))
print("Seasons:", sorted(team_adv_all["SEASON"].unique()))
team_adv_all.head()
```

```
18-19 Advanced Stats.csv -> rows: 30
19-20 Advanced Stats.csv -> rows: 30
```

```
20-21 Advanced Stats.csv -> rows: 30
21-22 Advanced Stats.csv -> rows: 30
22-23 Advanced Stats.csv -> rows: 30
23-24 Advanced Stats.csv -> rows: 30
24-25 Advanced Stats.csv -> rows: 30
Total team rows: 210
Seasons: [np.int64(2018), np.int64(2019), np.int64(2020), np.int64(2021),
np.int64(2022), np.int64(2023), np.int64(2024)]
```

[58]:

	Team	ORtg	DRtg	NRtg	SRS	Pace	TOV%	TS%	SEASON
0	Milwaukee Bucks	113.8	105.2	8.6	8.04	103.3	12.0	0.583	2018
1	Golden State Warriors	115.9	109.5	6.4	6.42	100.9	12.6	0.596	2018
2	Toronto Raptors	113.1	107.1	6.0	5.49	100.2	12.4	0.579	2018
3	Utah Jazz	110.9	105.7	5.2	5.28	100.3	13.4	0.572	2018
4	Houston Rockets	115.5	110.7	4.8	4.96	97.9	12.0	0.581	2018

[87]:
```
set(games_all["Home Team"].unique()) - set(team_adv_all["Team"].unique())
```

[87]:
```
set()
```

The code above is a saftey check to make sure the names are matching from the dataset, when we ran it we found that a name inconsistency (e.g., 'LA Clippers' vs. 'Los Angeles Clippers') had to be resolved prior to merging datasets!

[60]:
```
TEAM_NAME_FIXES = {
    "LA Clippers": "Los Angeles Clippers"
}

games_all["Home Team"] = games_all["Home Team"].replace(TEAM_NAME_FIXES)
games_all["Away Team"] = games_all["Away Team"].replace(TEAM_NAME_FIXES)

set(games_all["Home Team"].unique()) - set(team_adv_all["Team"].unique())
```

[60]:
```
set()
```

[61]:
```
## Merging team stats into either home or away games
games_feat = games_all.merge(
    team_adv_all,
    left_on=["Home Team", "SEASON"],
    right_on=["Team", "SEASON"],
    how="left"
).drop(columns=["Team"])

games_feat = games_feat.rename(columns={
    "ORtg": "ORtg_home",
    "DRtg": "DRtg_home",
    "NRtg": "NRtg_home",
    "SRS":  "SRS_home",
```

```
    "Pace": "Pace_home",
    "TOV%": "TOV_home",
    "TS%":  "TS_home",
})

games_feat = games_feat.merge(
    team_adv_all,
    left_on=["Away Team", "SEASON"],
    right_on=["Team", "SEASON"],
    how="left"
).drop(columns=["Team"])

games_feat = games_feat.rename(columns={
    "ORtg": "ORtg_away",
    "DRtg": "DRtg_away",
    "NRtg": "NRtg_away",
    "SRS":  "SRS_away",
    "Pace": "Pace_away",
    "TOV%": "TOV_away",
    "TS%":  "TS_away",
})
```

[62]:
```
for stat in ["ORtg", "DRtg", "NRtg", "SRS", "Pace", "TOV", "TS"]:
    games_feat[f"{stat}_diff"] = games_feat[f"{stat}_home"] -␣
 ↪games_feat[f"{stat}_away"]
```

0.2 Home-Team Baseline

We are just going to predict that the home team wins everytime, this following code will help get our first benchmark!

[63]:
```
from sklearn.metrics import accuracy_score

y_true = games_feat["home_win"]
y_pred_home = [1] * len(y_true)

home_baseline_acc = accuracy_score(y_true, y_pred_home)
home_baseline_acc
```

[63]:
```
0.5574185765983112
```

0.3 It predicts that 55.741% of the time the home team is going to win

Next, Train / Test split by the season, we're not doing a random split. We train on older seasons and test on newer seasons.

[64]:
```
train_seasons = [2018, 2019, 2020, 2021, 2022]
test_seasons = [2023, 2024]
```

```
train_df = games_feat[games_feat["SEASON"].isin(train_seasons)].copy()
test_df  = games_feat[games_feat["SEASON"].isin(test_seasons)].copy()

print("Train games:", len(train_df))
print("Test games:", len(test_df))
```

```
Train games: 5828
Test games: 2462
```

[65]:
```
feature_cols = [
    "ORtg_diff",
    "DRtg_diff",
    "NRtg_diff",
    "SRS_diff",
    "Pace_diff",
    "TOV_diff",
    "TS_diff"
]

X_train = train_df[feature_cols]
y_train = train_df["home_win"]

X_test = test_df[feature_cols]
y_test = test_df["home_win"]
```

[66]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (5828, 7)
y_train shape: (5828,)
X_test shape: (2462, 7)
y_test shape: (2462,)
```

[67]:
```
X_train.head()
##testing
```

[67]:

	ORtg_diff	DRtg_diff	NRtg_diff	SRS_diff	Pace_diff	TOV_diff	TS_diff
0	-0.4	-2.2	1.8	1.65	-2.0	-1.4	-0.007
1	5.6	2.5	3.1	2.86	-1.9	0.9	0.051
2	-2.4	7.3	-9.7	-9.36	-4.6	-1.1	-0.029
3	-0.6	-0.5	-0.1	-0.16	-3.4	-0.7	-0.012
4	3.8	-2.3	6.1	4.84	1.5	-0.5	0.013

[68]:
```
X_train.isna().sum()
## testing
```

[68]:
```
ORtg_diff    0
DRtg_diff    0
NRtg_diff    0
SRS_diff     0
Pace_diff    0
TOV_diff     0
TS_diff      0
dtype: int64
```

0.4 Baseline Logisitic Regression

[69]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

log_reg = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=1000))
])

log_reg.fit(X_train, y_train)
```

[69]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('clf', LogisticRegression(max_iter=1000))])
```

[70]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

y_pred_lr = log_reg.predict(X_test)

lr_acc = accuracy_score(y_test, y_pred_lr)
lr_acc
```

[70]:
```
0.6835905767668562
```

[71]:
```
print(f"Home-team baseline accuracy: {home_baseline_acc:.3f}")
print(f"Logistic regression accuracy: {lr_acc:.3f}")
```

```
Home-team baseline accuracy: 0.557
Logistic regression accuracy: 0.684
```

[72]:
```
from sklearn.metrics import confusion_matrix, classification_report

print(confusion_matrix(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

```
[[ 640  484]
 [ 295 1043]]
```

```
              precision    recall  f1-score   support

           0       0.68      0.57      0.62      1124
           1       0.68      0.78      0.73      1338

    accuracy                           0.68      2462
   macro avg       0.68      0.67      0.67      2462
weighted avg       0.68      0.68      0.68      2462
```

0.5 Baseline and Logistic Regression Results

0.5.1 Baseline Model

As you can see earlier, we listed a home team only predictor, which classifies every game as a home win. Across all seasons in the dataset, this baseline achieved an accuracy of **55.7%**, reflecting the known home-court advantage in the NBA.

0.5.2 Logistic Regression Model

We then trained a logistic regression model using **difference based team features** (home team minus away team), including: - Offensive Rating (ORtg) - Defensive Rating (DRtg) - Net Rating (NRtg) - Simple Rating System (SRS) - Pace - Turnover Percentage (TOV%) - True Shooting Percentage (TS%)

All features were standardized using `StandardScaler`, and the model was trained on historical seasons with evaluation performed on held-out seasons to avoid data leakage.

0.5.3 Overall Performance

The logistic regression model achieved an accuracy of **68.4%**, substantially outperforming the home-team baseline by nearly **13 percentage points**. This improvement indicates that team-level advanced statistics provide meaningful predictive signal beyond home-court advantage alone.

0.5.4 Confusion Matrix

	Predicted Loss	Predicted Win
Actual Loss	640	484
Actual Win	295	1043

The model correctly identified a large majority of home wins while still capturing many away-team victories, demonstrating balanced performance across both classes.

0.5.5 Classification Metrics

- **Precision (Home Win):** 0.68
- **Recall (Home Win):** 0.78

- **F1-score (Home Win):** 0.73

The high recall for home wins indicates that the model is effective at identifying games where the home team is likely to win, while maintaining reasonable precision. Performance on away wins is slightly weaker but still substantially better than random guessing or the baseline.

0.5.6 Summary

These results show that a relatively simple linear model, when combined with well-chosen advanced team metrics, can achieve strong predictive performance on NBA game outcomes. The logistic regression model serves as a solid benchmark for comparison with more complex models explored later in the project.

SVM

[73]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline

svm_linear = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="linear"))
])

param_grid_linear = {
    "clf__C": [0.01, 0.1, 1, 10, 100]
}

grid_linear = GridSearchCV(
    svm_linear,
    param_grid_linear,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid_linear.fit(X_train, y_train)

print("Best Linear SVM C:", grid_linear.best_params_)
print("CV Accuracy:", grid_linear.best_score_)
```

```
Best Linear SVM C: {'clf__C': 0.01}
CV Accuracy: 0.6607737100538136
```

[74]:
```
from sklearn.metrics import accuracy_score

svm_linear_best = grid_linear.best_estimator_
y_pred_linear = svm_linear_best.predict(X_test)
```

```
linear_acc = accuracy_score(y_test, y_pred_linear)
print("Linear SVM Test Accuracy:", linear_acc)
```

```
Linear SVM Test Accuracy: 0.6839967506092608
```

0.6 RBF (Nonlinear) SVM

[75]:
```
svm_rbf = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", SVC(kernel="rbf"))
])

param_grid_rbf = {
    "clf__C": [0.1, 1, 10],
    "clf__gamma": ["scale", 0.01, 0.1, 1]
}

grid_rbf = GridSearchCV(
    svm_rbf,
    param_grid_rbf,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid_rbf.fit(X_train, y_train)

print("Best RBF params:", grid_rbf.best_params_)
print("CV Accuracy:", grid_rbf.best_score_)
```

```
Best RBF params: {'clf__C': 1, 'clf__gamma': 0.1}
CV Accuracy: 0.6650639359830388
```

[76]:
```
svm_rbf_best = grid_rbf.best_estimator_
y_pred_rbf = svm_rbf_best.predict(X_test)

rbf_acc = accuracy_score(y_test, y_pred_rbf)
print("RBF SVM Test Accuracy:", rbf_acc)
```

```
RBF SVM Test Accuracy: 0.6868399675060926
```

0.7 Random Forest Classifier

[77]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)

param_grid_rf = {
```

```
    "n_estimators": [200, 500],
    "max_depth": [None, 5, 10],
    "min_samples_leaf": [1, 5, 10]
}

grid_rf = GridSearchCV(
    rf,
    param_grid_rf,
    cv=5,
    scoring="accuracy",
    n_jobs=-1
)

grid_rf.fit(X_train, y_train)

print("Best RF params:", grid_rf.best_params_)
print("CV Accuracy:", grid_rf.best_score_)
```

```
Best RF params: {'max_depth': 5, 'min_samples_leaf': 5, 'n_estimators': 500}
CV Accuracy: 0.6664354125103984
```

[78]:
```
rf_best = grid_rf.best_estimator_
y_pred_rf = rf_best.predict(X_test)

rf_acc = accuracy_score(y_test, y_pred_rf)
print("Random Forest Test Accuracy:", rf_acc)
```

```
Random Forest Test Accuracy: 0.6864337936636881
```

0.8 Final Model Comparison

[79]:
```
print("Home-team baseline accuracy: 0.557")
print(f"Logistic Regression accuracy: {log_reg.score(X_test, y_test):.3f}")
print(f"Linear SVM accuracy: {linear_acc:.3f}")
print(f"RBF SVM accuracy: {rbf_acc:.3f}")
print(f"Random Forest accuracy: {rf_acc:.3f}")
```

```
Home-team baseline accuracy: 0.557
Logistic Regression accuracy: 0.684
Linear SVM accuracy: 0.684
RBF SVM accuracy: 0.687
Random Forest accuracy: 0.686
```

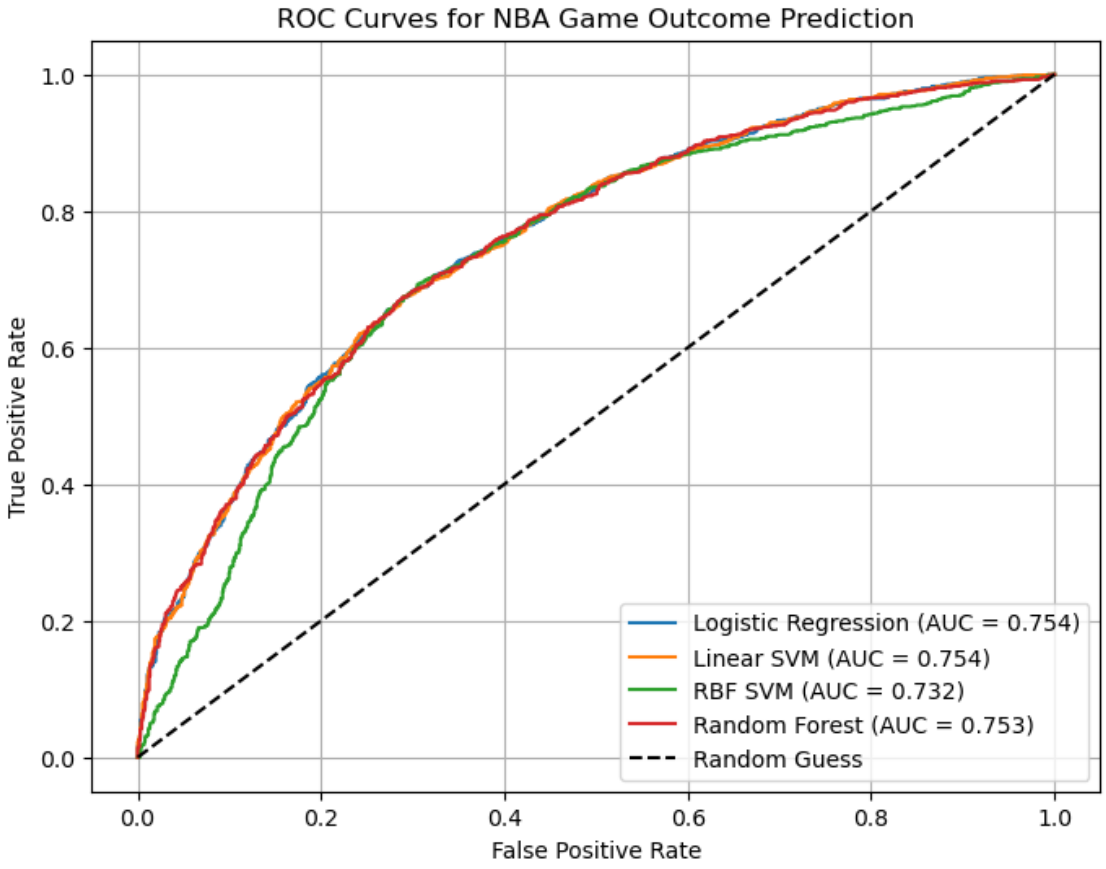
0.9 ROC Curves - Get Predicted Probabilities

[80]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Logistic Regression
```

```
y_score_lr = log_reg.predict_proba(X_test)[:, 1]

# Linear SVM
y_score_linear = svm_linear_best.decision_function(X_test)

# RBF SVM
y_score_rbf = svm_rbf_best.decision_function(X_test)

# Random Forest
y_score_rf = rf_best.predict_proba(X_test)[:, 1]
```

[81]:
```
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_score_lr)
fpr_lin, tpr_lin, _ = roc_curve(y_test, y_score_linear)
fpr_rbf, tpr_rbf, _ = roc_curve(y_test, y_score_rbf)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_score_rf)

auc_lr  = auc(fpr_lr, tpr_lr)
auc_lin = auc(fpr_lin, tpr_lin)
auc_rbf = auc(fpr_rbf, tpr_rbf)
auc_rf  = auc(fpr_rf, tpr_rf)
```

[82]:
```
plt.figure(figsize=(8, 6))

plt.plot(fpr_lr, tpr_lr, label=f"Logistic Regression (AUC = {auc_lr:.3f})")
plt.plot(fpr_lin, tpr_lin, label=f"Linear SVM (AUC = {auc_lin:.3f})")
plt.plot(fpr_rbf, tpr_rbf, label=f"RBF SVM (AUC = {auc_rbf:.3f})")
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {auc_rf:.3f})")

plt.plot([0, 1], [0, 1], "k--", label="Random Guess")

plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for NBA Game Outcome Prediction")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



0.10 ROC Curve Analysis

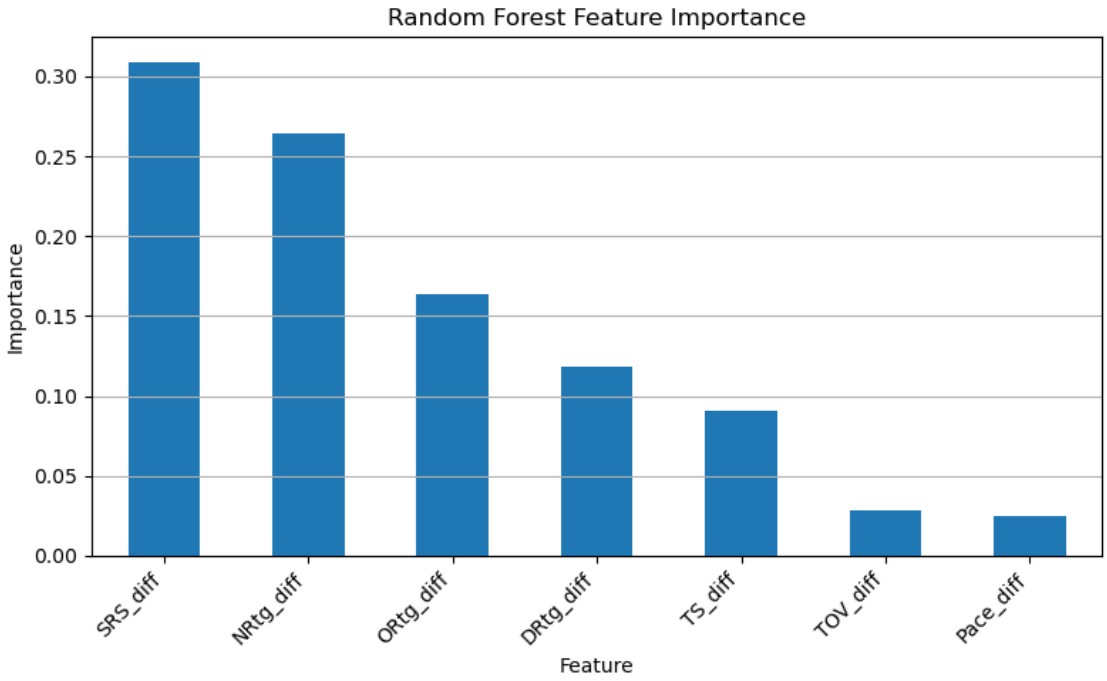
To further evaluate model performance, we generated Receiver Operating Characteristic (ROC) curves for all models. ROC curves illustrate the trade-off between true positive rate and false positive rate across different classification thresholds.

[83]:
```
feature_importance = pd.Series(
    rf_best.feature_importances_,
    index=feature_cols
).sort_values(ascending=False)

feature_importance
```

[83]:
```
SRS_diff     0.309133
NRtg_diff    0.264782
ORtg_diff    0.163592
DRtg_diff    0.118654
TS_diff      0.090645
TOV_diff     0.028732
Pace_diff    0.024462
```

```
dtype: float64
```

[84]:
```
plt.figure(figsize=(8, 5))
feature_importance.plot(kind="bar")
plt.title("Random Forest Feature Importance")
plt.ylabel("Importance")
plt.xlabel("Feature")
plt.xticks(rotation=45, ha="right")
plt.grid(axis="y")
plt.tight_layout()
plt.show()
```



0.11 Feature Importance Analysis

To better understand the factors that influence our predictions the most, we examined feature importance from the Random Forest model. Looking at all of our data, the most influential features were Net Rating difference and Simple Rating System (SRS) difference.

Offensive and Defensive rating differences also contributed meaningfully, while pace and turnover differences played smaller roles.

0.12 Limitations of the Model

However when looking at some drawbacks to our model, the analysis relies on season-aggregated team statistics, which do not capture game-specific factors such as injuries, lineup changes, or minutes restrictions. Basketball is an ever changing sport, and a season is full of highs and downs

momentum wise which is hard to account for. Additionally, the model does not explicitly account for scheduling effects such as back-to-back games, travel fatigue, or rest disparities between teams.

0.13 Error Analysis

An examination of misclassified games shows that many errors occur during unexpected outcomes, such as upsets where lower rated teams defeat stronger opponents. These cases often coincide with high variance events, including unusually poor shooting performances by favored teams or out of the ordianry performances by "underdogs".

Because the model relies on season-level averages, it is less sensitive to one-off events or short-term fluctuations. As a result, games with extreme variance or unusual circumstances are more likely to be misclassified.

0.14 Conclusion

In this project, we developed and evaluated multiple machine learning models to predict NBA regular-season game outcomes using team level advanced statistics. By combining game logs with Basketball Reference efficiency metrics, we created a structured dataset that captures meaningful differences in team quality.

All machine learning models significantly outperformed a simple home team baseline, with accuracies approaching 69%. Linear models performed comparably to more complex classifiers, indicating that overall team strength metrics such as Simple Rating System and Net Rating capture much of the predictive signal. Feature importance analysis further supported this conclusion, highlighting aggregate measures of team efficiency as the dominant factors in game outcomes.

Overall, this project demonstrates that well-designed feature engineering and sound experimental design can yield strong predictive performance even with relatively simple models. The results highlight both the strengths and limitations of team-level data in modeling inherently noisy sports outcomes.