In [7]:
```
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import GridSearchCV
import statsmodels.formula.api as smf
import statsmodels.regression.linear_model as slm
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.graphics.gofplots import qqplot
import scipy.stats as stats
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix,
import seaborn as sns
from sklearn.preprocessing import RobustScaler
from xgboost import plot_importance
import os
```

In [8]:
```
os.getcwd()
```

Out[8]:
```
'/Users/jaykepastis/Football Analytics/OSU_Hackathon'
```

In [ ]:
```
pip install seaborn
```

In [4]:
```
pip install Pandoc
```

```
Collecting Pandoc
  Downloading pandoc-2.4.tar.gz (34 kB)
  Preparing metadata (setup.py) ... done
Collecting plumbum (from Pandoc)
  Downloading plumbum-1.9.0-py3-none-any.whl.metadata (10 kB)
Collecting ply (from Pandoc)
  Downloading ply-3.11-py2.py3-none-any.whl.metadata (844 bytes)
Downloading plumbum-1.9.0-py3-none-any.whl (127 kB)
Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
Building wheels for collected packages: Pandoc
  Building wheel for Pandoc (setup.py) ... done
  Created wheel for Pandoc: filename=pandoc-2.4-py3-none-any.whl size=34792
sha256=0a1062d769cfed5efdba07f190e5760a333e832b20d3de87e840ef1914283fac
  Stored in directory: /Users/jaykepastis/Library/Caches/pip/wheels/14/79/8
c/5d7a023cc8df1aa0381c1739d69da18ae7f90c08b2dc9a1bf5
Successfully built Pandoc
Installing collected packages: ply, plumbum, Pandoc
Successfully installed Pandoc-2.4 plumbum-1.9.0 ply-3.11
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install TeX
```

```
Collecting TeX
  Downloading tex-1.8.tar.gz (4.2 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: TeX
  Building wheel for TeX (setup.py) ... done
  Created wheel for TeX: filename=tex-1.8-py3-none-any.whl size=5110 sha256=
a9b00974d9e7608cbb0a7a76dc928155e8e53ade740f9629849159c318e2a7ef
  Stored in directory: /Users/jaykepastis/Library/Caches/pip/wheels/b3/d1/8
3/ce91035b92ccac87592a5e008c7a08f1cf28f8679045dce96d
Successfully built TeX
Installing collected packages: TeX
Successfully installed TeX-1.8
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
#Ohio State Team Dataset: Game by Game
OSU_df = pd.read_csv('OSUHackathon2025_TeamGameData.csv')
```

In [ ]:
```
#Ohio State Player Dataset
OSU_Player_df = pd.read_csv('OSUHackathon2025_PlayerTotalPointsData.csv')
OSU_Player_df
```

In [ ]:
```
#Elo Ratings
CFP_Elo_Ratings = pd.read_excel("CFP_Elo_Ratings.xlsx", engine="openpyxl")
```

In [ ]:
```
#ESPN Strength of Record
CFB_SOS = pd.read_csv('CFB_SOS.csv')
```

In [ ]:
```
OSU_df['Date'] = pd.to_datetime(OSU_df['Date'])
```

In [ ]:
```
filtered_data = OSU_df.loc[
    ((OSU_df['Season'] == 2018) & (OSU_df['Date'] < '2018-12-15')) |
    ((OSU_df['Season'] == 2019) & (OSU_df['Date'] < '2019-12-20')) |
    ((OSU_df['Season'] == 2021) & (OSU_df['Date'] < '2021-12-17')) |
    ((OSU_df['Season'] == 2022) & (OSU_df['Date'] < '2022-12-16')) |
    ((OSU_df['Season'] == 2023) & (OSU_df['Date'] < '2023-12-16')) |
    ((OSU_df['Season'] == 2024) & (OSU_df['Date'] < '2024-12-14'))
]

print(filtered_data)
```

In [ ]:
```
merged_df = pd.merge(filtered_data, CFB_SOS, on=['Team', 'Season'], how='inn
merged_df
```

In [ ]:
```
y = merged_df['WasCFPTeam']

selected_features = ['Sum_PassBlockPoints','OffensiveEPAPerPlay_Pass', 'Offe
                    'Sum_PassRushPoints', 'Sum_RunDefPoints']

X = merged_df.loc[:, selected_features]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rar

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

In [ ]:
```
model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)

model.fit(X_train, y_train)
```

In [ ]:
```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

In [ ]:
```
merged_df['Win'] = (merged_df['Score'] > merged_df['OppScore']).astype(int)

merged_df['CumulativeWins'] = (
    merged_df.groupby(['Team', 'Season'])['Win']
    .cumsum()
    .shift(fill_value=0)
)

merged_df['GamesPlayed'] = (
    merged_df.groupby(['Team', 'Season']).cumcount()
)

merged_df['WinPct_BeforeGame'] = merged_df['CumulativeWins'] / merged_df['Ga
merged_df['WinPct_BeforeGame'] = merged_df['WinPct_BeforeGame'].fillna(0)
```

In [ ]:
```
y = merged_df['WasCFPTeam']

selected_features = [
    'Sum_PassBlockPoints', 'OffensiveEPAPerPlay_Pass',
    'OffensiveEPAPerPlay_Run', 'SOR', 'Sum_RunBlockPoints',
    'Sum_PassRushPoints', 'Sum_RunDefPoints',
]

X = merged_df.loc[:, selected_features]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    min_child_weight=3,
    random_state=42
)

model.fit(X_train_scaled, y_train)
```

In [ ]:
```
y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]

X_test_with_preds = X_test.copy()
X_test_with_preds['Predicted_Playoff_Prob'] = y_pred_prob

X_test_with_preds[['Team', 'Season']] = merged_df.loc[X_test.index, ['Team',

X_test_with_preds = (
    X_test_with_preds.groupby(['Team', 'Season'])['Predicted_Playoff_Prob']
    .mean()
    .reset_index()
)

X_test_with_preds['Projected_Rank'] = (
    X_test_with_preds.groupby('Season')['Predicted_Playoff_Prob']
    .rank(method='dense', ascending=False)
    .astype(int)
)

print(X_test_with_preds[['Team', 'Season', 'Predicted_Playoff_Prob', 'Projec
      .sort_values(['Season', 'Projected_Rank']))
```

In [ ]:
```
y = merged_df['WasCFPTeam']

selected_features = [
    'Sum_PassBlockPoints', 'OffensiveEPAPerPlay_Pass',
    'OffensiveEPAPerPlay_Run', 'SOR', 'Sum_RunBlockPoints',
    'Sum_PassRushPoints', 'Sum_RunDefPoints'
]

X = merged_df.loc[:, selected_features]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = xgb.XGBClassifier(
    objective='binary:logistic',
    eval_metric='logloss',
    n_estimators=200,
    learning_rate=0.05,
    max_depth=4,
    min_child_weight=3,
    random_state=42
)

model.fit(X_train_scaled, y_train)

y_pred_prob = model.predict_proba(X_test_scaled)[:, 1]

X_test_with_preds = X_test.copy()
X_test_with_preds['Predicted_Playoff_Prob'] = y_pred_prob

X_test_with_preds[['Team', 'Season']] = merged_df.loc[X_test.index, ['Team',

X_test_with_preds = (
    X_test_with_preds.groupby(['Team', 'Season'])['Predicted_Playoff_Prob']
    .mean()
    .reset_index()
)

X_test_with_preds['Projected_Rank'] = (
    X_test_with_preds.groupby('Season')['Predicted_Playoff_Prob']
    .rank(method='dense', ascending=False)
    .astype(int)
)

print(X_test_with_preds[['Team', 'Season', 'Predicted_Playoff_Prob', 'Projec
      .sort_values(['Season', 'Projected_Rank']))
```

In [ ]:
```
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

plot_importance(model, importance_type='weight')
plt.show()
```

In [ ]:
```
scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [ ]:
```
final_df = X_test_with_preds[['Team', 'Season', 'Predicted_Playoff_Prob', 'P
    .sort_values(['Season', 'Projected_Rank'])
```

In [ ]:
```
final_df
```

In [ ]:
```
final_df[(final_df['Team'] == 'SMU') & (final_df['Season'] == 2024)]
```

In [ ]:
```
final_df.to_csv('predicted_playoff_rankings.csv', index=False)
```

In [ ]:
```
top_2024 = (
    X_test_with_preds[X_test_with_preds['Season'] == 2024]
    .sort_values('Projected_Rank')
    .head(20)
)
```

```
print(top_2024[['Team', 'Season', 'Predicted_Playoff_Prob', 'Projected_Rank'
```

In [ ]:
```
top_2023 = (
    X_test_with_preds[X_test_with_preds['Season'] == 2023]
    .sort_values('Projected_Rank')
    .head(20)
)

print(top_2023[['Team', 'Season', 'Predicted_Playoff_Prob', 'Projected_Rank'
```

In [ ]:
```
data = {
    'Year': [year for year in range(2014, 2025) if year != 2020 for _ in rar
    'Rank': list(range(1, 26)) * 10,
    'Team': [
        # 2014
        "Alabama", "Oregon", "Florida State", "Ohio State", "Baylor",
        "TCU", "Mississippi State", "Michigan State", "Ole Miss", "Arizona",
        "Kansas State", "Georgia Tech", "Georgia", "UCLA", "Arizona State",
        "Missouri", "Clemson", "Wisconsin", "Auburn", "Boise State",
        "Louisville", "Utah", "LSU", "USC", "Minnesota",

        # 2015
        "Alabama", "Clemson", "Stanford", "Ohio State", "Oklahoma",
        "Michigan State", "TCU", "Houston", "Iowa", "Ole Miss",
        "Notre Dame", "Michigan", "Baylor", "Florida State", "North Carolina
        "LSU", "Utah", "Navy", "Oregon", "Oklahoma State",
        "Wisconsin", "Tennessee", "Northwestern", "Western Kentucky", "Flori

        # 2016
        "Clemson", "Alabama", "USC", "Washington", "Oklahoma",
        "Ohio State", "Penn State", "Florida State", "Wisconsin", "Michigan"
        "Oklahoma State", "Stanford", "LSU", "Florida", "Western Michigan",
        "Virginia Tech", "Colorado", "West Virginia", "South Florida", "Miam
        "Louisville", "Tennessee", "Utah", "Auburn", "San Diego State",

        # 2017
        "Alabama", "Georgia", "Oklahoma", "Clemson", "Ohio State",
        "UCF", "Wisconsin", "Penn State", "TCU", "Auburn",
        "Notre Dame", "USC", "Miami", "Oklahoma State", "Michigan State",
        "Washington", "Northwestern", "LSU", "Mississippi State", "Stanford"
        "South Florida", "Boise State", "NC State", "Virginia Tech", "Memphi

        # 2018
        "Clemson", "Alabama", "Ohio State", "Oklahoma", "Notre Dame",
        "LSU", "Florida", "Georgia", "Texas", "Washington State",
        "UCF", "Kentucky", "Washington", "Michigan", "Syracuse",
        "Texas A&M", "Penn State", "Fresno State", "Army", "West Virginia",
        "Northwestern", "Utah State", "Boise State", "Cincinnati", "Iowa",

        # 2019
        "LSU", "Clemson", "Ohio State", "Georgia", "Oregon",
        "Florida", "Oklahoma", "Alabama", "Penn State", "Minnesota",
        "Wisconsin", "Notre Dame", "Baylor", "Auburn", "Iowa",
        "Utah", "Memphis", "Michigan", "Appalachian State", "Navy",
        "Cincinnati", "Air Force", "Boise State", "UCF", "Texas",

        # 2021
        "Georgia", "Alabama", "Michigan", "Cincinnati", "Baylor",
        "Ohio State", "Oklahoma State", "Notre Dame", "Michigan State", "Okl
        "Ole Miss", "Utah", "Pittsburgh", "Clemson", "Wake Forest",
        "Louisiana", "Houston", "Kentucky", "BYU", "NC State",
        "Arkansas", "Oregon", "Iowa", "Utah State", "San Diego State",

        # 2022
        "Georgia", "TCU", "Michigan", "Ohio State", "Alabama",
        "Tennessee", "Penn State", "Washington", "Tulane", "Utah",
        "Florida State", "Southern California Trojans", "Clemson", "Kansas S
        "LSU", "Oregon State", "Notre Dame", "Troy", "Mississippi State",
        "UCLA", "Pittsburgh", "South Carolina", "Fresno State", "Texas",

        # 2023
        "Michigan", "Washington", "Texas", "Georgia", "Alabama",
        "Florida State", "Oregon", "Missouri", "Ole Miss", "Ohio State",
        "Arizona", "LSU", "Penn State", "Notre Dame", "Oklahoma",
        "Oklahoma State", "Tennessee", "Kansas State", "Louisville", "Clemso
        "NC State", "SMU", "Kansas", "Iowa", "Liberty",

        # 2024
        "Ohio State", "Notre Dame", "Oregon", "Texas", "Penn State",
        "Georgia", "Arizona State", "Boise State", "Tennessee", "Indiana",
        "Ole Miss", "SMU", "BYU", "Clemson", "Iowa State",
        "Illinois", "Alabama", "Miami", "South Carolina", "Syracuse",
        "Army", "Missouri", "UNLV", "Memphis", "Colorado"
    ]
}

print("Length of Year array:", len(data['Year']))
print("Length of Rank array:", len(data['Rank']))
print("Length of Team array:", len(data['Team']))

df = pd.DataFrame(data)
print(df)
```

In [ ]:
```
df = df.rename(columns={'Year': 'Season'})
```

In [ ]:
```
df
```

In [ ]:
```
df.to_csv('CFP_Rankings.csv', index=False)
```

In [ ]:
```
Elo_Ratings = CFP_Elo_Ratings[CFP_Elo_Ratings['Season'] != 2020]
Elo_Ratings
```

In [ ]:
```
dataset = pd.merge(df, Elo_Ratings, on=['Team', 'Season'], how='inner')
dataset
```

In [ ]:
```
def win_probability(elo1, elo2):
    return 1 / (1 + 10 ** ((elo2 - elo1) / 400))
```

```
def assign_seeds(df):
    seed_df = pd.DataFrame()

    for season, group in df.groupby('Season'):
        top_winners = group[group['Conf_Winner'] == 1].nlargest(5, 'Elo').cc
        top_winners['Seed'] = range(1, 6)
        top_non_winners = group[group['Conf_Winner'] == 0].nlargest(5, 'Elo'
        top_non_winners['Seed'] = range(6, 11)
        season_seeds = pd.concat([top_winners, top_non_winners])
        seed_df = pd.concat([seed_df, season_seeds])

    return seed_df

seeded_teams = assign_seeds(dataset)
teams = {row['Seed']: row['Elo'] for _, row in seeded_teams.iterrows()}

n_simulations = 10000
champion_counts = {seed: 0 for seed in range(1, 11)}

for _ in range(n_simulations):
    winners = {}
    for matchup in [(7, 10), (8, 9)]:
        elo1 = teams.get(matchup[0], 1500)
        elo2 = teams.get(matchup[1], 1500)
        winner = matchup[0] if np.random.rand() < win_probability(elo1, elo2
        winners[winner] = winner

    quarterfinal_matchups = [
        (3, winners.get(7, 10)),
        (4, winners.get(8, 9)),
        (5, 6)
    ]

    quarterfinal_winners = {}
    for matchup in quarterfinal_matchups:
        elo1 = teams.get(matchup[0], 1500)
        elo2 = teams.get(matchup[1], 1500)
        winner = matchup[0] if np.random.rand() < win_probability(elo1, elo2
        quarterfinal_winners[winner] = winner

    semifinal_matchups = [
        (1, max(quarterfinal_winners, key=quarterfinal_winners.get, default=
        (2, min(quarterfinal_winners, key=quarterfinal_winners.get, default=
    ]

    semifinal_winners = {}
    for matchup in semifinal_matchups:
        elo1 = teams.get(matchup[0], 1500)
        elo2 = teams.get(matchup[1], 1500)
        winner = matchup[0] if np.random.rand() < win_probability(elo1, elo2
        semifinal_winners[winner] = winner

    champ_matchup = list(semifinal_winners.keys())
    elo1 = teams.get(champ_matchup[0], 1500)
    elo2 = teams.get(champ_matchup[1], 1500)
    winning_seed = champ_matchup[0] if np.random.rand() < win_probability(el
    champion_counts[winning_seed] += 1

win_probs = {seed: (champion_counts[seed] / n_simulations) * 100 for seed ir

plt.figure(figsize=(10, 6))
plt.bar(win_probs.keys(), win_probs.values(), color='skyblue', edgecolor='bl
plt.xlabel("Playoff Seed", fontsize=12)
plt.ylabel("Championship Win Probability (%)", fontsize=12)
plt.title("CFP 10-Team Monte Carlo Simulation (10,000 Runs)", fontsize=14)
plt.xticks(range(1, 11))
plt.show()
```

In [ ]:
```
plt.figure(figsize=(10, 6))
bars = plt.bar(win_probs.keys(), win_probs.values(), color='skyblue', edgecc

for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height, f'{height:.2f}%',
             ha='center', va='bottom', fontsize=10)

plt.xlabel("Playoff Seed", fontsize=12)
plt.ylabel("Championship Win Probability (%)", fontsize=12)
plt.title("CFP 10-Team Monte Carlo Simulation (10,000 Runs)", fontsize=14)
plt.xticks(range(1, 11))
plt.show()
```