Sports Analytics Project

April 21, 2025

[2]:
```
import pandas as pd

df = pd.read_csv(r"C:\Users\chara\Downloads\badminton_worldtour.csv")

print(df.head())
print(df.info())
print(df.isnull().sum())
```

```
   Unnamed: 0           Player Category  Matches  Wins  Losses  pts_for  \
0           1       Aaron CHIA  Doubles       17    14       3      781
1           2    Adnan MAULANA  Doubles        1     1       0       62
2           3  Akane YAMAGUCHI  Singles       78    54      24     3732
3           4       Akira KOGA  Doubles        7     5       2      355
4           5      Alex LANIER  Singles        3     2       1      171

   pts_agst   win_pct  shot_pct
0       658  0.823529  0.542738
1        54  1.000000  0.534483
2      3337  0.692308  0.527939
3       327  0.714286  0.520528
4       171  0.666667  0.500000
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  185 non-null    int64
 1   Player      185 non-null    object
 2   Category    185 non-null    object
 3   Matches     185 non-null    int64
 4   Wins        185 non-null    int64
 5   Losses      185 non-null    int64
 6   pts_for     185 non-null    int64
 7   pts_agst    185 non-null    int64
 8   win_pct     185 non-null    float64
 9   shot_pct    185 non-null    float64
dtypes: float64(2), int64(6), object(2)
memory usage: 14.6+ KB
```

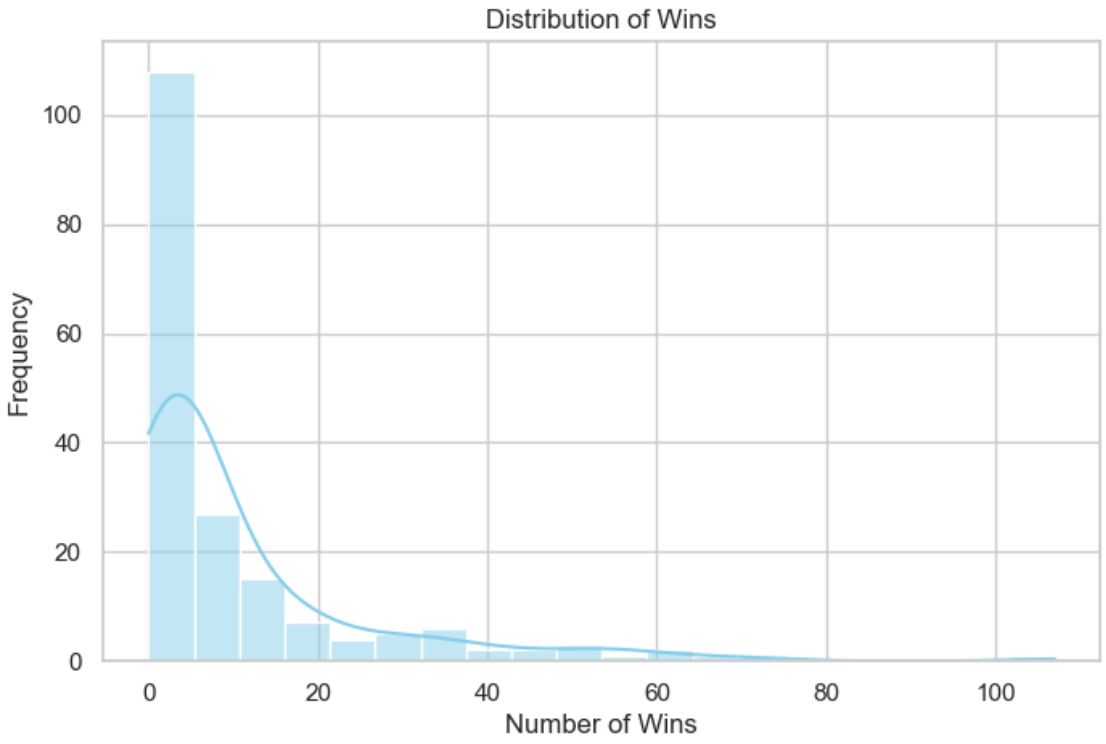
```
None
Unnamed: 0    0
Player        0
Category      0
Matches       0
Wins          0
Losses        0
pts_for       0
pts_agst      0
win_pct       0
shot_pct      0
dtype: int64
```

[3]:
```
#Column name cleanup
df.drop(columns=["Unnamed: 0"], inplace=True)
```

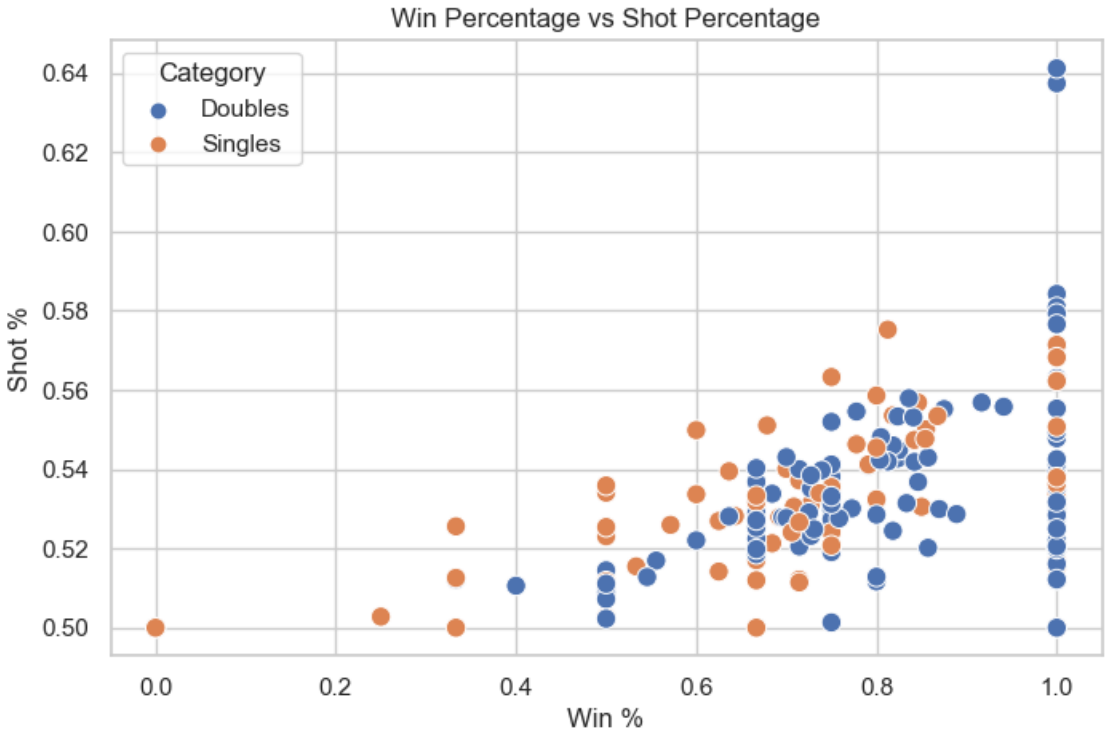
[10]:
```
#Feature Enigneering
# Feature 1: Win/Loss Ratio
df['win_loss_ratio'] = df['Wins'] / df['Losses'].replace(0, 1)  # avoid␣
 ↪division by zero

# Feature 2: Average Points per Match
df['avg_points_per_match'] = df['pts_for'] / df['Matches'].replace(0, 1)

# Feature 3: Net Points (Points For - Points Against)
df['net_points'] = df['pts_for'] - df['pts_agst']

# Feature 4: Convert Category to Numerical (optional but useful for models)
df['category_type'] = df['Category'].map({'Singles': 0, 'Doubles': 1})

# Show the new columns added
print(df[['Player', 'win_loss_ratio', 'avg_points_per_match', 'net_points',␣
 ↪'category_type']].head())
```

```
           Player  win_loss_ratio  avg_points_per_match  net_points  \
0       Aaron CHIA        4.666667             45.941176         123
1    Adnan MAULANA        1.000000             62.000000           8
2  Akane YAMAGUCHI        2.250000             47.846154         395
3       Akira KOGA        2.500000             50.714286          28
4      Alex LANIER        2.000000             57.000000           0

   category_type
0              1
1              1
2              0
3              1
4              0
```

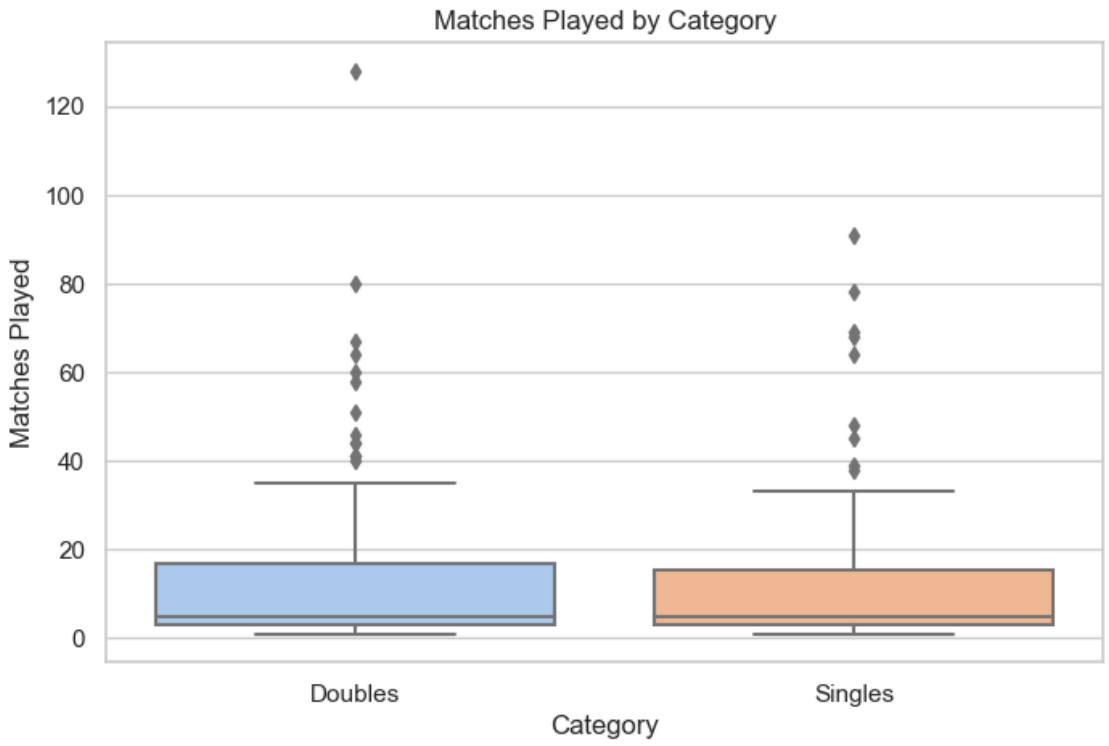
[12]:
```
import matplotlib.pyplot as plt
import seaborn as sns

#For better plot appearance
sns.set(style="whitegrid")

# 1. Distribution of Wins
plt.figure(figsize=(8, 5))
sns.histplot(df['Wins'], kde=True, bins=20, color='skyblue')
plt.title('Distribution of Wins')
plt.xlabel('Number of Wins')
plt.ylabel('Frequency')
plt.show()
```



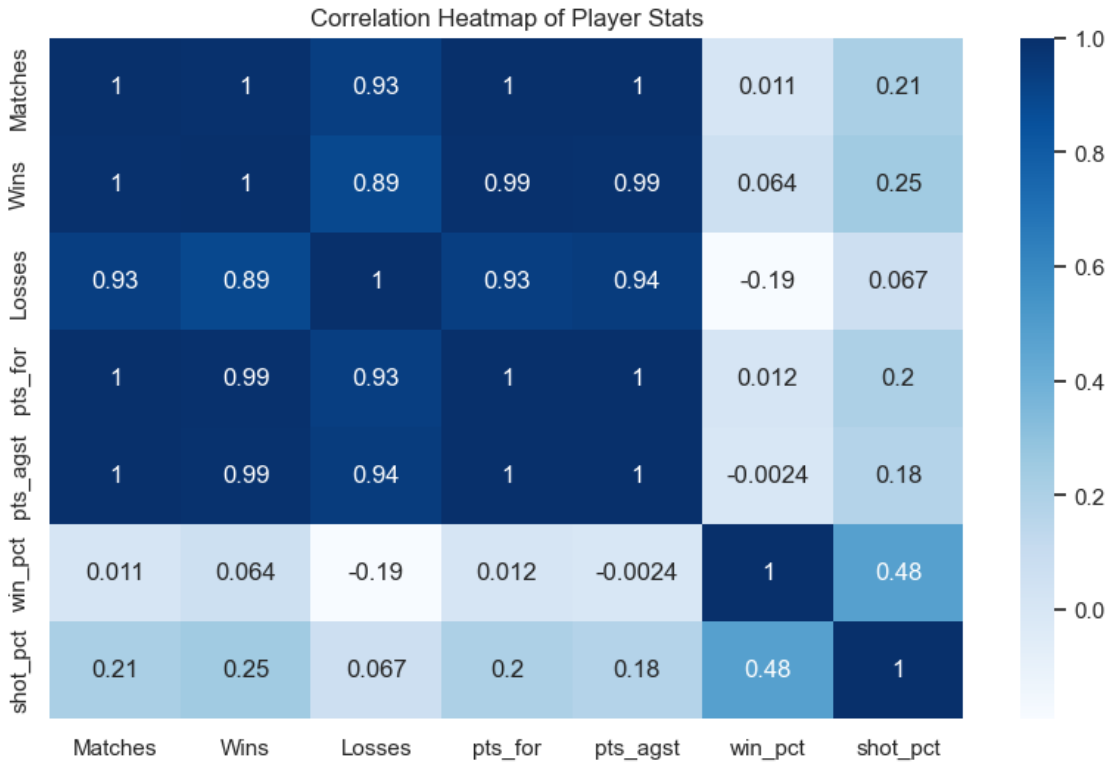
[13]:
```
# 2. Win % vs Shot % Scatter Plot
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='win_pct', y='shot_pct', hue='Category', s=80)
plt.title('Win Percentage vs Shot Percentage')
plt.xlabel('Win %')
plt.ylabel('Shot %')
plt.legend(title='Category')
plt.show()
```



[14]:
```
# 3. Boxplot of Matches by Category
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='Category', y='Matches', palette='pastel')
plt.title('Matches Played by Category')
plt.ylabel('Matches Played')
plt.show()
```



[15]:
```
# 4. Correlation Heatmap
plt.figure(figsize=(10, 6))
corr = df[['Matches', 'Wins', 'Losses', 'pts_for', 'pts_agst', 'win_pct',␣
 ↪'shot_pct']].corr()
sns.heatmap(corr, annot=True, cmap='Blues')
plt.title('Correlation Heatmap of Player Stats')
plt.show()
```



[17]:
```
features = ['Matches', 'Wins', 'Losses', 'pts_for', 'pts_agst', 'shot_pct']
target = 'win_pct'
```

[18]:
```
from sklearn.model_selection import train_test_split

# Define features and target
X = df[['Matches', 'Wins', 'Losses', 'pts_for', 'pts_agst', 'shot_pct']]
y = df['win_pct']

# Split into train and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print("Training samples:", len(X_train))
print("Testing samples:", len(X_test))
```

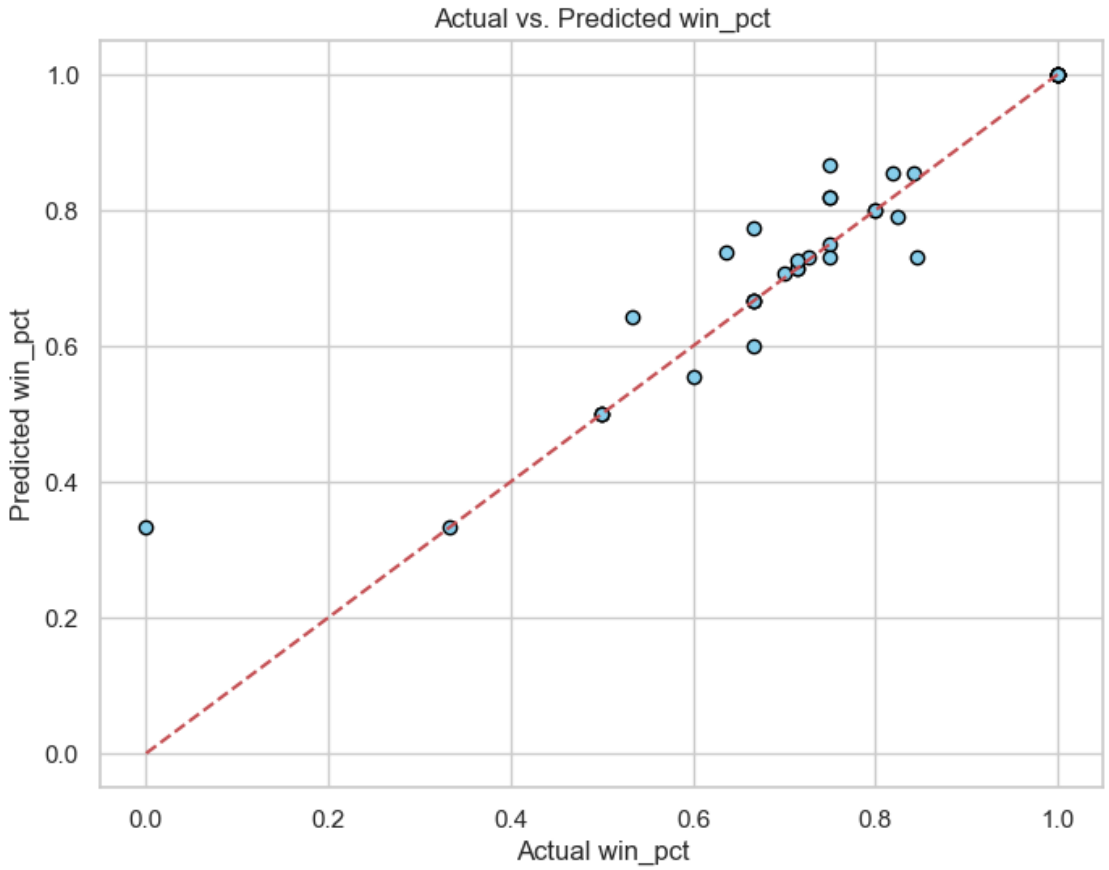
```
Training samples: 148
Testing samples: 37
```

[19]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# Initialize the model
```

```
model = DecisionTreeRegressor(random_state=42)

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

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error (MAE):", round(mae, 4))
print("R-squared Score (R²):", round(r2, 4))
```

```
Mean Absolute Error (MAE): 0.0338
R-squared Score (R²): 0.8788
```

[20]:
```
import matplotlib.pyplot as plt

# Scatter plot of Actual vs. Predicted values
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='skyblue', edgecolor='black')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')  #␣
 ↪Ideal line
plt.xlabel('Actual win_pct')
plt.ylabel('Predicted win_pct')
plt.title('Actual vs. Predicted win_pct')
plt.grid(True)
plt.show()
```



[24]:
```
importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
})

importance = importance.sort_values(by='Importance', ascending=False)
print(importance)
```

```
    Feature  Importance
2    Losses    0.594514
1      Wins    0.315892
0   Matches    0.043415
5  shot_pct    0.026289
4  pts_agst    0.014740
3   pts_for    0.005150
```

[28]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python
import numpy as np

# Initialize models
dt_model = DecisionTreeRegressor(random_state=42)  # Added Decision Tree model
rf_model = RandomForestRegressor(random_state=42)
lr_model = LinearRegression()

# Fit models
dt_model.fit(X_train, y_train)  # Fit Decision Tree model
rf_model.fit(X_train, y_train)
lr_model.fit(X_train, y_train)

# Predictions
dt_preds = dt_model.predict(X_test)  # Added predictions for Decision Tree
rf_preds = rf_model.predict(X_test)
lr_preds = lr_model.predict(X_test)

# Performance comparison
print("Model Comparison:\n")

print("Decision Tree:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, dt_preds)):.4f}")
print(f"R² Score: {r2_score(y_test, dt_preds):.4f}")

print("\nRandom Forest:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, rf_preds)):.4f}")
print(f"R² Score: {r2_score(y_test, rf_preds):.4f}")

print("\nLinear Regression:")
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, lr_preds)):.4f}")
print(f"R² Score: {r2_score(y_test, lr_preds):.4f}")
```

```
Model Comparison:

Decision Tree:
RMSE: 0.0717
R² Score: 0.8788

Random Forest:
RMSE: 0.0560
R² Score: 0.9262

Linear Regression:
RMSE: 0.1487
R² Score: 0.4795
```

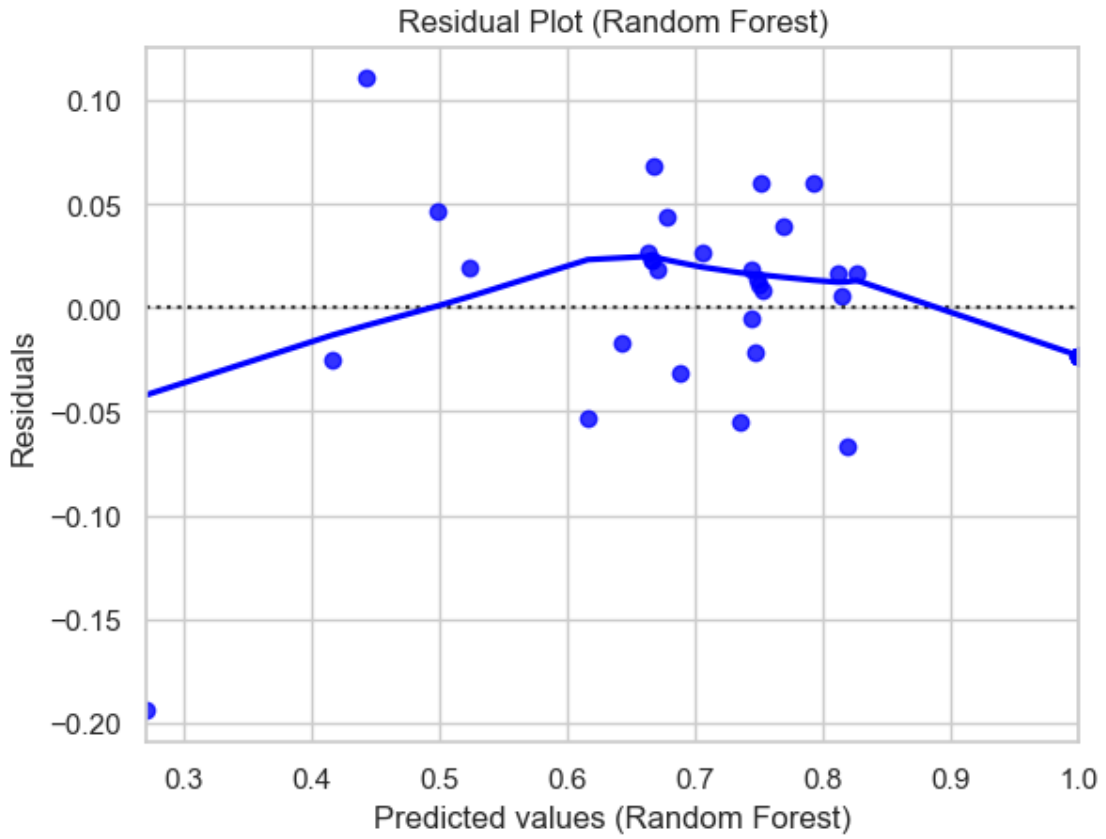
[29]:
```
from sklearn.metrics import mean_absolute_error

print("Decision Tree MAE:", mean_absolute_error(y_test, dt_preds))
print("Random Forest MAE:", mean_absolute_error(y_test, rf_preds))
print("Linear Regression MAE:", mean_absolute_error(y_test, lr_preds))
```

```
Decision Tree MAE: 0.03381697369613828
Random Forest MAE: 0.028750208278802307
Linear Regression MAE: 0.10517957425046069
```

[30]:
```
def adjusted_r2_score(r2, n, p):
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

n = len(y_test)  # Number of observations
p = X_test.shape[1]  # Number of features

print("Adjusted R² (Random Forest):", adjusted_r2_score(r2_score(y_test,␣
 ↪rf_preds), n, p))
```

```
Adjusted R² (Random Forest): 0.9114578954518373
```

[31]:
```
import matplotlib.pyplot as plt
import seaborn as sns

residuals_rf = y_test - rf_preds
sns.residplot(x=rf_preds, y=residuals_rf, lowess=True, color="blue")
plt.xlabel("Predicted values (Random Forest)")
plt.ylabel("Residuals")
plt.title("Residual Plot (Random Forest)")
plt.show()
```



[32]:
```
from sklearn.model_selection import cross_val_score

# Cross-validation for Random Forest
rf_cv_scores = cross_val_score(rf_model, X_train, y_train, cv=5,␣
 ↪scoring='neg_mean_squared_error')
print(f"Random Forest CV RMSE: {np.sqrt(-rf_cv_scores.mean()):.4f}")

# Cross-validation for Linear Regression
lr_cv_scores = cross_val_score(lr_model, X_train, y_train, cv=5,␣
 ↪scoring='neg_mean_squared_error')
print(f"Linear Regression CV RMSE: {np.sqrt(-lr_cv_scores.mean()):.4f}")
```

```
Random Forest CV RMSE: 0.0694
Linear Regression CV RMSE: 0.1475
```

[33]:
```
# Getting the feature importances for Random Forest
importances = rf_model.feature_importances_
feature_names = X_train.columns  # If using a DataFrame, otherwise use your␣
 ↪feature names
```

```
# Sort and display feature importance
sorted_indices = importances.argsort()
for i in sorted_indices[::-1]:
    print(f"{feature_names[i]}: {importances[i]:.4f}")
```

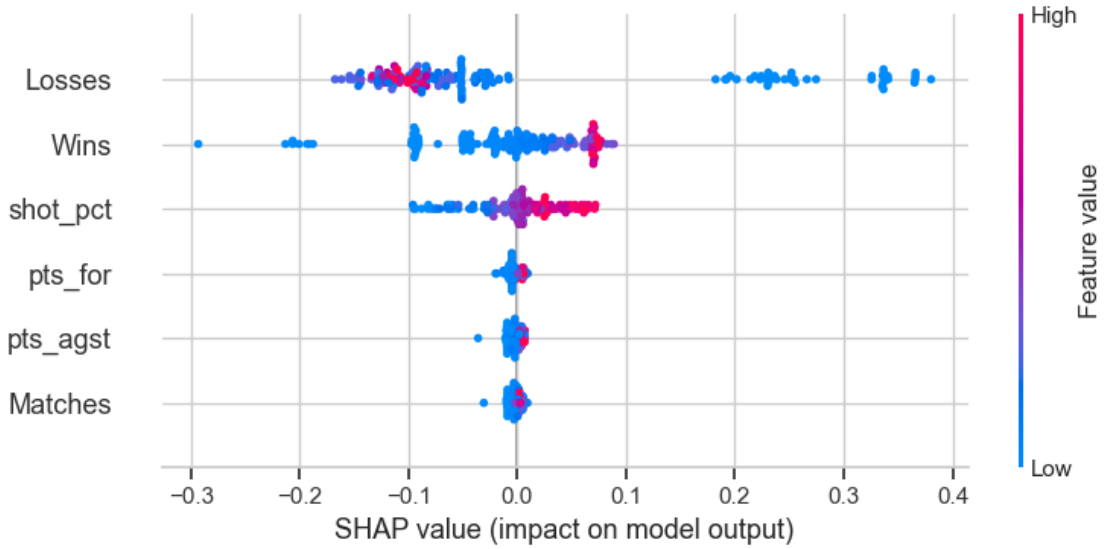
```
Losses: 0.5908
Wins: 0.2221
shot_pct: 0.1275
Matches: 0.0220
pts_for: 0.0191
pts_agst: 0.0186
```

[35]:
```
pip install shap
```

```
Collecting shap
  Downloading shap-0.47.2-cp311-cp311-win_amd64.whl (544 kB)
                                              0.0/544.4 kB ? eta -:--:--
     ----------                               153.6/544.4 kB 3.1 MB/s eta 0:00:01
     -------------------                      276.5/544.4 kB 2.9 MB/s eta 0:00:01
     ----------------------------             409.6/544.4 kB 3.2 MB/s eta 0:00:01
     ------------------------------------     542.7/544.4 kB 3.1 MB/s eta 0:00:01
     -------------------------------------- 544.4/544.4 kB 2.9 MB/s eta 0:00:00
Requirement already satisfied: numpy in e:\anaconda\lib\site-packages (from
shap) (1.24.3)
Requirement already satisfied: scipy in e:\anaconda\lib\site-packages (from
shap) (1.10.1)
Requirement already satisfied: scikit-learn in e:\anaconda\lib\site-packages
(from shap) (1.2.2)
Requirement already satisfied: pandas in e:\anaconda\lib\site-packages (from
shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in e:\anaconda\lib\site-packages
(from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in e:\anaconda\lib\site-packages
(from shap) (23.0)
Collecting slicer==0.0.8 (from shap)
  Downloading slicer-0.0.8-py3-none-any.whl (15 kB)
Requirement already satisfied: numba>=0.54 in e:\anaconda\lib\site-packages
(from shap) (0.57.0)
Requirement already satisfied: cloudpickle in e:\anaconda\lib\site-packages
(from shap) (2.2.1)
Requirement already satisfied: typing-extensions in e:\anaconda\lib\site-
packages (from shap) (4.9.0)
Requirement already satisfied: llvmlite<0.41,>=0.40.0dev0 in
e:\anaconda\lib\site-packages (from numba>=0.54->shap) (0.40.0)
Requirement already satisfied: colorama in e:\anaconda\lib\site-packages (from
tqdm>=4.27.0->shap) (0.4.6)
Requirement already satisfied: python-dateutil>=2.8.1 in e:\anaconda\lib\site-
packages (from pandas->shap) (2.8.2)
```

```
Requirement already satisfied: pytz>=2020.1 in e:\anaconda\lib\site-packages
(from pandas->shap) (2022.7)
Requirement already satisfied: joblib>=1.1.1 in e:\anaconda\lib\site-packages
(from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in e:\anaconda\lib\site-
packages (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in e:\anaconda\lib\site-packages (from
python-dateutil>=2.8.1->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.47.2 slicer-0.0.8
Note: you may need to restart the kernel to use updated packages.
```

[36]:
```python
import shap

explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_train)

# Plot the SHAP summary plot
shap.summary_plot(shap_values, X_train)
```



[38]:
```python
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# Define the parameter distribution for RandomizedSearchCV
param_dist = {
    'n_estimators': np.arange(100, 500, 100),
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize the RandomizedSearchCV object
random_search = RandomizedSearchCV(estimator=rf_model,␣
 ↪param_distributions=param_dist, n_iter=100, cv=5, n_jobs=-1, verbose=2)

# Fit the random search to find the best parameters
random_search.fit(X_train, y_train)

# Best parameters found by RandomizedSearchCV
print(f"Best Parameters: {random_search.best_params_}")

# Use the best model found from random search
best_rf_model = random_search.best_estimator_

# Predict using the best model
best_rf_preds = best_rf_model.predict(X_test)

# Evaluate the tuned model
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, best_rf_preds)):.4f}")
print(f"R² Score: {r2_score(y_test, best_rf_preds):.4f}")
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Best Parameters: {'n_estimators': 100, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_depth': 10, 'bootstrap': True}
RMSE: 0.0562
R² Score: 0.9257
```

[42]:
```
import joblib

# Save the best model
joblib.dump(best_rf_model, 'random_forest_model.pkl')

# Load the model later for predictions
loaded_model = joblib.load('random_forest_model.pkl')
```

[ ]:
```

```