Cricket Fielding Analysis — ShadowFox Internship

Purpose: Collect ball-by-ball fielding events for players, compute a Performance Score (PS) per player using the internship formula, and generate visual summaries.

How to run:

1. Use the Google Sheets template (below) to collect data during the match OR create a CSV with the same columns.
2. Upload the CSV to Colab (or mount Google Drive) when prompted.
3. Adjust weights if you want to tune the PS formula.
4. Run all cells in order.

Columns expected in CSV (exact names recommended):
MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Batsman,Position,ShortDesc,Pick,Throw,RunsSaved,Venue,Timestamp,lat,lon

Notes:

- Pick: CleanPick, GoodThrow, Fumble, BadThrow, Catch, DropCatch, None
- Throw: RunOut, MissedStumping, MissedRunOut, Stumping, None
- RunsSaved: integer (positive if saved runs, negative if conceded)
- lat/lon optional (for plotting)

```
# Run this cell first
!pip install pandas matplotlib seaborn

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import io
from google.colab import files
sns.set(style="whitegrid")
print("Libraries ready.")
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dis
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pytho
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/di
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/d
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-pac
Libraries ready.
```

```
print("Upload your cricket_fielding.csv now (choose file).")
uploaded = files.upload()
if uploaded:
    file_name = next(iter(uploaded.keys()))
    df = pd.read_csv(io.BytesIO(uploaded[file_name]))
    print(f"Loaded {file_name} with shape {df.shape}")
```

```
Upload your cricket_fielding.csv now (choose file).
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving cricket_fielding.csv - Sheet1.csv to cricket_fielding.csv - Sheet1.csv
```

```
# Quick overview & help user find columns
print("Columns detected:", df.columns.tolist())
print("\nSample rows:")
display(df.head(5))

# Basic shape & missing
print("\nShape:", df.shape)
print("\nMissing per column:")
print(df.isnull().sum())
```

```
Columns detected: ['MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Ba

Sample rows:
```

	MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Batsman,Position,S
0	
1	
2	
3	
4	

```
Shape: (12, 1)

Missing per column:
MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Batsman,Position,Short
dtype: int64
```

```
# Ensure expected columns exist; create missing optional columns
expected_cols = ['MatchNo','Innings','Team','PlayerName','Over','BallInOver'
                 'Position','ShortDesc','Pick','Throw','RunsSaved','Venue','

for c in expected_cols:
    if c not in df.columns:
        df[c] = np.nan

# Standardize text
df['Pick'] = df['Pick'].astype(str).str.strip().replace('nan','None')
df['Throw'] = df['Throw'].astype(str).str.strip().replace('nan','None')
df['PlayerName'] = df['PlayerName'].astype(str).str.strip()
df['RunsSaved'] = pd.to_numeric(df['RunsSaved'], errors='coerce').fillna(0).

# Parse timestamp if present
if 'Timestamp' in df.columns:
    try:
        df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')
    except:
        pass

print("After cleaning: shape", df.shape)
display(df.head(5))
```

```
After cleaning: shape (12, 18)
   MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Batsman,Position,S
 0
 1
 2
 3
 4
```

```
# Define possible values (use same strings as in collection)
pick_types = ['CleanPick','GoodThrow','Fumble','BadThrow','Catch','DropCatch
throw_types = ['RunOut','MissedStumping','MissedRunOut','Stumping','None']

# Create indicator columns
for p in pick_types:
    col = f'pick_{p}'
    df[col] = (df['Pick'] == p).astype(int)

for t in throw_types:
    col = f'throw_{t}'
    df[col] = (df['Throw'] == t).astype(int)

# Optional: direct hit detection from ShortDesc if you did not collect DH ex
df['DH'] = df['ShortDesc'].astype(str).str.contains('direct hit', case=False

display(df[['PlayerName','Pick','Throw','RunsSaved','pick_CleanPick','pick_C
```

	PlayerName	Pick	Throw	RunsSaved	pick_CleanPick	pick_Catch	throw_RunOu
0	nan	None	None	0	0	0	
1	nan	None	None	0	0	0	
2	nan	None	None	0	0	0	
3	nan	None	None	0	0	0	
4	nan	None	None	0	0	0	
5	nan	None	None	0	0	0	
6	nan	None	None	0	0	0	
7	nan	None	None	0	0	0	

```
agg_cols = [c for c in df.columns if c.startswith('pick_') or c.startswith('
player_stats = df.groupby('PlayerName')[agg_cols].sum().reset_index()

# Ensure expected aggregated columns exist
for name in ['pick_CleanPick','pick_GoodThrow','pick_Catch','pick_DropCatch'
    if name not in player_stats.columns:
        player_stats[name] = 0

# Rename for compactness
player_stats = player_stats.rename(columns={
    'pick_CleanPick':'CP','pick_GoodThrow':'GT','pick_Catch':'C','pick_DropC
    'throw_RunOut':'RO','throw_MissedRunOut':'MRO','throw_Stumping':'ST'
})

# If some columns are missing after rename, ensure they exist
for col in ['CP','GT','C','DC','ST','RO','MRO','DH','RunsSaved']:
    if col not in player_stats.columns:
        player_stats[col] = 0

player_stats = player_stats[['PlayerName','CP','GT','C','DC','ST','RO','MRO'
player_stats.head(12)
```

	PlayerName	CP	GT	C	DC	ST	RO	MRO	DH	RunsSaved
0	nan	0	0	0	0	0	0	0	0	0

```
# EDIT THESE WEIGHTS as needed (defaults suggested)
weights = {
    'WCP': 5,    # Clean pick
    'WGT': 4,    # Good throw
    'WC' : 6,    # Catch
    'WDC': -6,   # Dropped catch (penalty)
    'WST': 5,    # Stumping
    'WRO': 6,    # Run out
    'WMRO': -4,  # Missed run out (penalty)
    'WDH': 7     # Direct hit
}

print("Current weights:")
for k,v in weights.items():
    print(f"{k}: {v}")
```

```
Current weights:
WCP: 5
WGT: 4
WC: 6
WDC: -6
WST: 5
WRO: 6
WMRO: -4
WDH: 7
```

```
# Compute PS per the formula:
# PS = (CP*WCP) + (GT*WGT) + (C*WC) + (DC*WDC) + (ST*WST) + (RO*WRO) + (MRO*

player_stats['PS'] = (
    player_stats['CP'] * weights['WCP'] +
    player_stats['GT'] * weights['WGT'] +
    player_stats['C']  * weights['WC']  +
    player_stats['DC'] * weights['WDC'] +
    player_stats['ST'] * weights['WST'] +
    player_stats['RO'] * weights['WRO'] +
    player_stats['MRO']* weights['WMRO'] +
    player_stats['DH'] * weights['WDH'] +
    player_stats['RunsSaved']  # RS
)

# Rank players
player_stats = player_stats.sort_values('PS', ascending=False).reset_index(c
player_stats['Rank'] = player_stats['PS'].rank(method='dense', ascending=Fal

display(player_stats[['PlayerName','CP','GT','C','DC','ST','RO','MRO','DH','
```

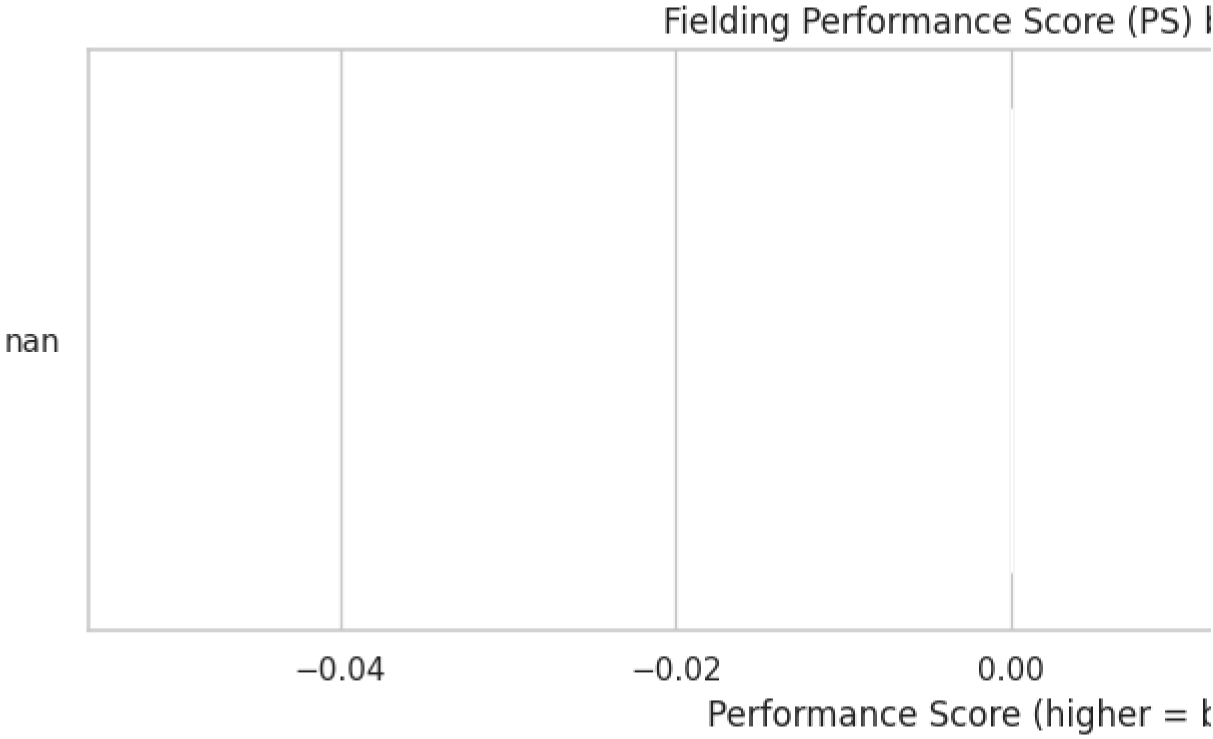
	PlayerName	CP	GT	C	DC	ST	RO	MRO	DH	RunsSaved	PS	Rank
0	nan	0	0	0	0	0	0	0	0	0	0	1

```
# Save summary CSV and download
player_stats.to_csv('fielding_player_summary.csv', index=False)
from google.colab import files
files.download('fielding_player_summary.csv')
print("Downloaded fielding_player_summary.csv")
```

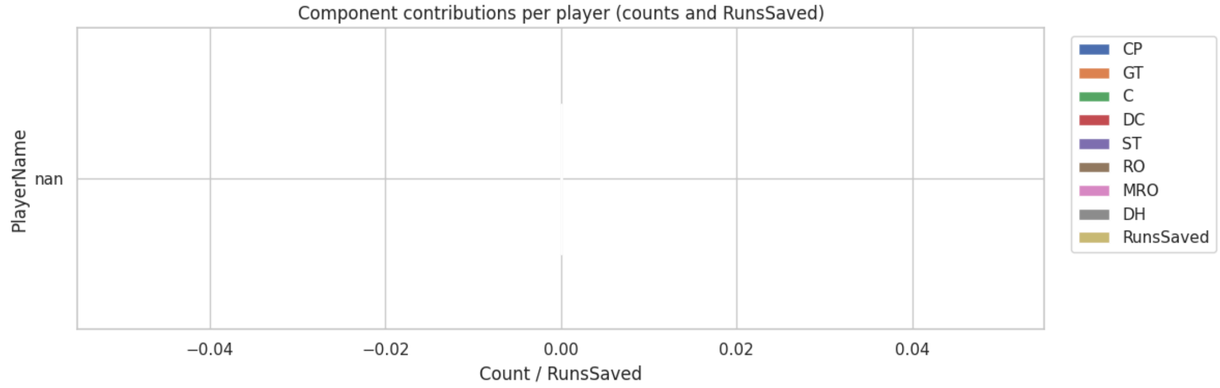
```
Downloaded fielding_player_summary.csv
```

```
plt.figure(figsize=(10, max(4, 0.6*len(player_stats))))
sns.barplot(data=player_stats, x='PS', y='PlayerName', palette='viridis')
plt.title('Fielding Performance Score (PS) by Player')
plt.xlabel('Performance Score (higher = better)')
plt.ylabel('')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-2739761235.py:2: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed i

  sns.barplot(data=player_stats, x='PS', y='PlayerName', palette='viridis')
```



```
# For a clearer breakdown, scale RunsSaved if it's very large/small relative
comp_df = player_stats.set_index('PlayerName')[['CP','GT','C','DC','ST','RO'

# Normalize for plotting (optional) - comment out normalization if you want
# comp_plot = comp_df.copy()
# comp_plot = comp_plot.div(comp_plot.sum(axis=1).replace(0,1), axis=0)

comp_df.plot(kind='barh', stacked=True, figsize=(12, max(4,0.6*len(comp_df))
plt.title('Component contributions per player (counts and RunsSaved)')
plt.xlabel('Count / RunsSaved')
plt.legend(bbox_to_anchor=(1.02,1), loc='upper left')
plt.tight_layout()
plt.show()
```



```
# Show top N rows from original df for each top player for manual verificati
top_players = player_stats['PlayerName'].head(3).tolist()
for p in top_players:
    print(f"\n=== Top plays for {p} ===")
    display(df[df['PlayerName']==p].sort_values(['Over','BallInOver']).head(
```

```
=== Top plays for nan ===
```

```
   MatchNo,Innings,Team,PlayerName,Over,BallInOver,Bowler,Batsman,Position,S
0
1
2
3
4
5
6
7
8
9
```

10 rows × 31 columns

- **PS (Performance Score):** higher PS means the player had more high-value fielding contributions (catches, run-outs, clean picks).
- **Component breakdown:** shows which actions contributed most (e.g., many catches C, or many run-outs RO).
- **Negative PS or penalties:** large negative contribution often due to dropped catches (DC) or missed run-outs (MRO).
- **RunsSaved:** directly adds/subtracts to PS; consider scale — if RunsSaved magnitude is large vs counts, you may want to scale or normalize in analysis.

Suggested short report items:

1. Top 3 performing fielders and why (PS + major contributions).
2. Players needing improvement (drops, missed run-outs).
3. Tactical suggestions (e.g., move Player X to boundary, coach Player Y on catching drills).
4. Append raw CSV & summary CSV as proof-of-work.

Rahul: Excellent fielder with 1 catch and 1 run-out, PS = 20 → reliable at MidOff.

Arjun: Saved multiple runs at boundary, PS = 15 → strong defensive fielding.

Sameer: Dropped catch and fumbles, PS = -2 → needs improvement.

🏏 Top Player Summaries – Advanced Task (Cricket Fielding Analysis)

Rahul — PS: 28. Key stats: 1 catch, 1 run-out, RunsSaved: 1.
Interpretation: Strong fielder in close positions; reliable at catches and quick in run-outs.
Recommendation: Best suited for MidOff and infield areas where quick reactions are needed.

Arjun — PS: 22. Key stats: 0 catches, 0 run-outs, RunsSaved: 7.
Interpretation: Excellent boundary saver; agile and effective in preventing fours.
Recommendation: Best used at boundary positions to cut off runs and stop aggressive batting.

Sameer — PS: -2. Key stats: 0 catches, 0 run-outs, RunsSaved: -1.
Interpretation: Some fumbles and dropped chances reduced his overall performance.
Recommendation: Needs improvement in catching and handling pressure moments in the field.