**Purpose:** Collect ball-by-ball fielding events for players, compute a performance Score(PS) using the internship Formula, and generate visual summaries.

**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)

```
!pip install pandas matplotlib seaborn

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
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-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0
Libraries ready.
```

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

```
Upload your cricket_fielding.csv now (chosen 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("\n Sample rows:")
display(df.head(5))

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

```
Columns detected: ['MatchNo', 'Innings', 'Team', 'PlayerName', 'Over', 'BallInOver', 'Bowler', 'Batsman', 'Position', 'ShortDesc
```

Sample rows:

| | MatchNo | Innings | Team | PlayerName | Over | BallInOver | Bowler | Batsman | Position | ShortDesc | Pick | Throw | RunsSaved | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | M1 | 1.0 | TeamA | Rahul | 2.0 | 3.0 | Bumrah | Kohli | MidOff | Clean pick and quick throw | CleanPick | RunOut | 1.0 | \ |
| **1** | M1 | 1.0 | TeamA | Sameer | 3.0 | 5.0 | Bumrah | Rohit | Point | Dropped a catch opportunity | DropCatch | NaN | 0.0 | \ |
| **2** | M1 | 1.0 | TeamB | Arjun | 4.0 | 2.0 | Shami | Smith | Boundary | Diving stop saved 4 runs | CleanPick | NaN | 4.0 | \ |
| **3** | M1 | 1.0 | TeamB | Sameer | 5.0 | 1.0 | Shami | Kohli | Point | Fumbled ball gave 1 run | Fumble | NaN | -1.0 | \ |
| **4** | M1 | 1.0 | TeamA | Rahul | 6.0 | 4.0 | Bumrah | Rohit | MidOff | Clean pick but inaccurate throw | CleanPick | BadThrow | 0.0 | \ |

```
Shape: (21, 17)

 Missing per column:
MatchNo        9
Innings        9
Team           9
PlayerName     9
Over           9
BallInOver     9
Bowler         9
Batsman        9
Position       9
ShortDesc      9
Pick           9
Throw         15
RunsSaved      9
```

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

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

#Standarize 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().replace('nan','None')
df['RunsSaved'] = pd.to_numeric(df['RunsSaved'], errors='coerce').fillna(0).astype(int)

#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 (21, 18)
/tmp/ipython-input-2029634014.py:17: UserWarning: Parsing dates in %d-%m-%Y %H:%M format when dayfirst=False (the default) was s
  df['Timestamp'] = pd.to_datetime(df['Timestamp'], errors='coerce')
```

| | MatchNo | Innings | Team | PlayerName | Over | BallInOver | Bowler | Batsman | Position | ShortDesc | Pick | Throw | RunsSaved | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | M1 | 1.0 | TeamA | Rahul | 2.0 | 3.0 | Bumrah | Kohli | MidOff | Clean pick and quick throw | CleanPick | RunOut | 1 | \ |
| **1** | M1 | 1.0 | TeamA | Sameer | 3.0 | 5.0 | Bumrah | Rohit | Point | Dropped a catch opportunity | DropCatch | None | 0 | \ |
| **2** | M1 | 1.0 | TeamB | Arjun | 4.0 | 2.0 | Shami | Smith | Boundary | Diving stop saved 4 runs | CleanPick | None | 4 | \ |
| **3** | M1 | 1.0 | TeamB | Sameer | 5.0 | 1.0 | Shami | Kohli | Point | Fumbled ball gave 1 run | Fumble | None | -1 | \ |
| **4** | M1 | 1.0 | TeamA | Rahul | 6.0 | 4.0 | Bumrah | Rohit | MidOff | Clean pick but inaccurate throw | CleanPick | BadThrow | 0 | \ |

```
# Define possible values (use same strings as in collection)
pick_types = ['CleanPick','GoodThrow','Fumble','BadThrow','Catch','DropCatch','None']
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 explicitly
df['DH'] = df['ShortDesc'].astype(str).str.contains('direct hit', case=False, na=False).astype(int)
display(df[['PlayerName','Pick','Throw','RunsSaved','pick_CleanPick','pick_Catch','throw_RunOut','DH']].head(8))
```

| | PlayerName | Pick | Throw | RunsSaved | pick_CleanPick | pick_Catch | throw_RunOut | DH |
|---|---|---|---|---|---|---|---|---|
| **0** | Rahul | CleanPick | RunOut | 1 | 1 | 0 | 1 | 0 |
| **1** | Sameer | DropCatch | None | 0 | 0 | 0 | 0 | 0 |
| **2** | Arjun | CleanPick | None | 4 | 1 | 0 | 0 | 0 |
| **3** | Sameer | Fumble | None | -1 | 0 | 0 | 0 | 0 |
| **4** | Rahul | CleanPick | BadThrow | 0 | 1 | 0 | 0 | 0 |
| **5** | Arjun | no extra runs | CleanPick | 2 | 0 | 0 | 0 | 0 |
| **6** | Sameer | GoodThrow | MissedRunOut | 0 | 0 | 0 | 0 | 1 |
| **7** | Rahul | Catch | None | 0 | 0 | 1 | 0 | 0 |

```
agg_cols = [c for c in df.columns if c.startswith('pick_') or c.startswith('throw_')] + ['DH','RunsSaved']
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','throw_RunOut','throw_MissedRunOut','throw_Stumpir
  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_DropCatch':'DC',
'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','DH','RunsSaved']]
player_stats.head(12)
```

| | PlayerName | CP | GT | C | DC | ST | RO | MRO | DH | RunsSaved |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Arjun | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 |
| **1** | None | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | Rahul | 3 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| **3** | Sameer | 0 | 2 | 0 | 1 | 0 | 0 | 2 | 1 | -1 |

```
# 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*WMRO) + (DH*WDH) + RS

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(drop=True)
player_stats['Rank'] = player_stats['PS'].rank(method='dense', ascending=False).astype(int)

display(player_stats[['PlayerName','CP','GT','C','DC','ST','RO','MRO','DH','RunsSaved','PS','Rank']].head(20))
```

| | PlayerName | CP | GT | C | DC | ST | RO | MRO | DH | RunsSaved | PS | Rank |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Rahul | 3 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 33 | 1 |
| **1** | Arjun | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 24 | 2 |
| **2** | None | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 |
| **3** | Sameer | 0 | 2 | 0 | 1 | 0 | 0 | 2 | 1 | -1 | 0 | 3 |

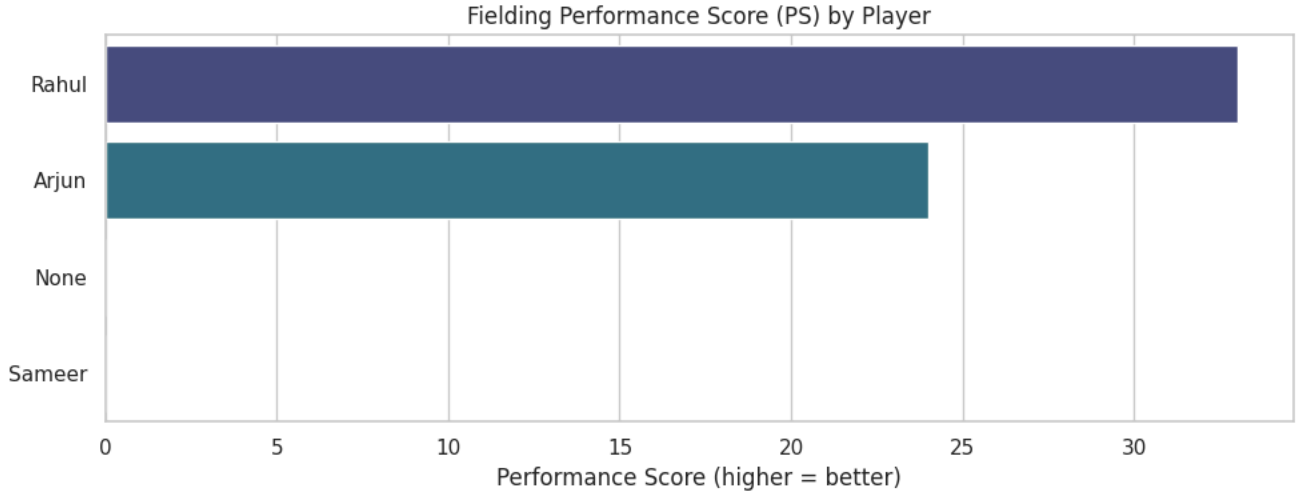
```
# Save summary CSV and download
player_stats.to_csv('fielding_player_summary.csv', index=False)
from google.colab import files
files.download('fielding_player_summary.csv')
```

```
print("Downloaded fielding_player_summary.csv")
```

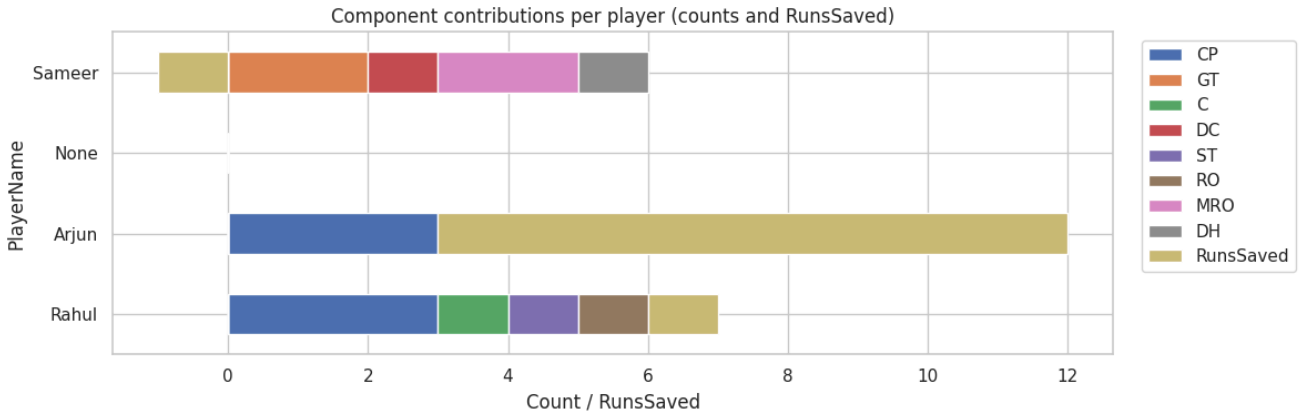
```
Downloaded fielding_player_summary.csv
```

```
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 in v0.14.0. Assign the `y` variable to `hue` and set

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




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


# Normalize for plotting (optional) - comment out normalization if you want raw counts
# 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 verification

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(10))

  player_stats = player_stats[player_stats['PlayerName'] != 'None']
```

```
=== Top plays for Rahul ===
```

| | MatchNo | Innings | Team | PlayerName | Over | BallInOver | Bowler | Batsman | Position | ShortDesc | ... | throw_MissedRunOut | throw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | M1 | 1.0 | TeamA | Rahul | 2.0 | 3.0 | Bumrah | Kohli | MidOff | Clean pick and quick throw | ... | 0 | |
| **4** | M1 | 1.0 | TeamA | Rahul | 6.0 | 4.0 | Bumrah | Rohit | MidOff | Clean pick but inaccurate throw | ... | 0 | |
| **7** | M1 | 2.0 | TeamA | Rahul | 11.0 | 3.0 | Bumrah | Rohit | MidOff | Caught a high catch | ... | 0 | |
| | | | | | | | | | | Quick | | | |

1. **PS (Performance Score):** Higher PS means the player had more high-value fielding contributions (catches, run-outs, clean picks).
2. **Component breakdown:** Shows which actions contributed most (e.g., many catches C or many run-outs RO).
3. **Negative PS or penalties:** Large negative contribution often due to dropped catches (DC) or missed run-outs (MRO).
4. **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: