Data Analysis Report

(TEAM 18)

IPL DATA ANALYTICS

Team Members

Ayush Gumgaonkar, Rahul Aditya, Bommireddy Srivani

Infotact Solutions Internship

Introduction:

The Indian Premier League (IPL) has emerged as one of the most popular and competitive T20 cricket leagues in the world, captivating millions of fans globally. With its dynamic format, diverse player pool, and high-scoring matches, the IPL generates an immense amount of data, offering valuable insights into player performance, team strategies, match outcomes, and fan engagement trends.

This report aims to analyze IPL data comprehensively, leveraging statistical tools and visualization techniques to uncover patterns, trends, and anomalies. By examining key metrics such as batting and bowling performances, team dynamics, and match conditions, this analysis provides actionable insights that can benefit teams, analysts, and enthusiasts alike.

The findings presented here are intended to shed light on the factors influencing success in the IPL, highlight standout performances, and offer data-driven predictions for future seasons.

Datasets Used :

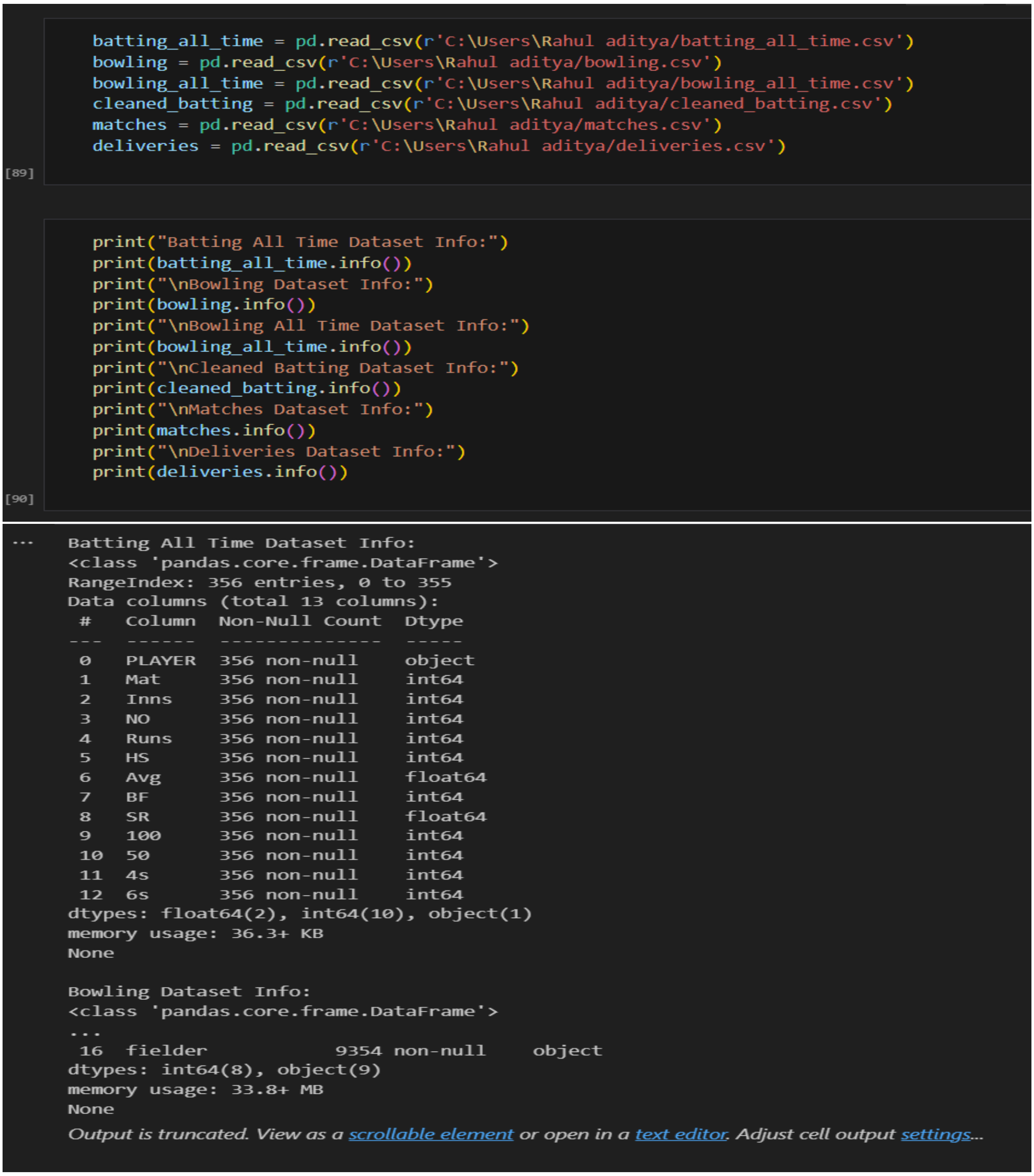
GitHub : cleaned_batting.xlsx , batting_all_time.xlsx , bowling_all_time.xlsx , bowling.xlsx

Kaggle : cleaned_matches.xlsx , cleaned_deliveries (1).xlsx , IPL-Winners.xlsx

Features Identified :

Screenshots of Pandas-Profiling Reports :

Cleaning:



```
batting_all_time = pd.read_csv(r'C:\Users\Rahul aditya/batting_all_time.csv')
bowling = pd.read_csv(r'C:\Users\Rahul aditya/bowling.csv')
bowling_all_time = pd.read_csv(r'C:\Users\Rahul aditya/bowling_all_time.csv')
cleaned_batting = pd.read_csv(r'C:\Users\Rahul aditya/cleaned_batting.csv')
matches = pd.read_csv(r'C:\Users\Rahul aditya/matches.csv')
deliveries = pd.read_csv(r'C:\Users\Rahul aditya/deliveries.csv')
```

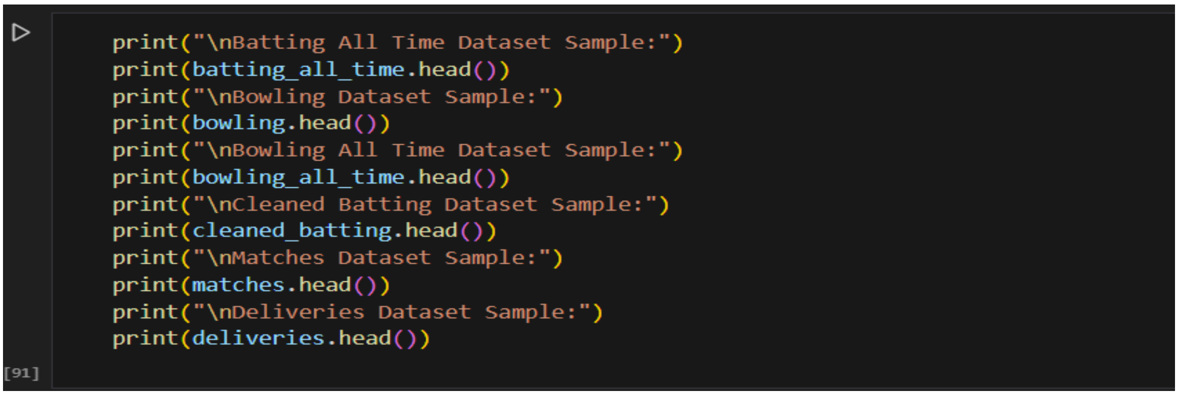
[89]

```
print("Batting All Time Dataset Info:")
print(batting_all_time.info())
print("\nBowling Dataset Info:")
print(bowling.info())
print("\nBowling All Time Dataset Info:")
print(bowling_all_time.info())
print("\nCleaned Batting Dataset Info:")
print(cleaned_batting.info())
print("\nMatches Dataset Info:")
print(matches.info())
print("\nDeliveries Dataset Info:")
print(deliveries.info())
```

[90]

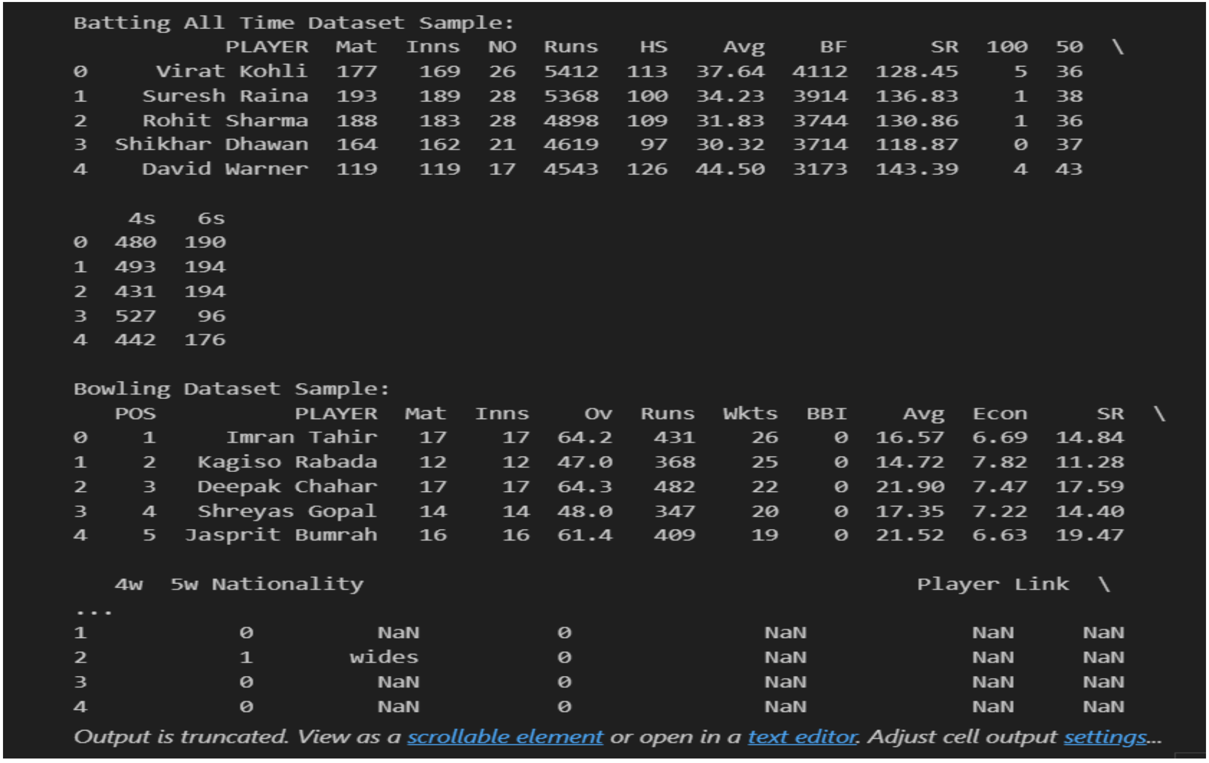
```
...  Batting All Time Dataset Info:
     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 356 entries, 0 to 355
     Data columns (total 13 columns):
      #   Column  Non-Null Count  Dtype
     ---  ------  --------------  -----
      0   PLAYER  356 non-null    object
      1   Mat     356 non-null    int64
      2   Inns    356 non-null    int64
      3   NO      356 non-null    int64
      4   Runs    356 non-null    int64
      5   HS      356 non-null    int64
      6   Avg     356 non-null    float64
      7   BF      356 non-null    int64
      8   SR      356 non-null    float64
      9   100     356 non-null    int64
      10  50      356 non-null    int64
      11  4s      356 non-null    int64
      12  6s      356 non-null    int64
     dtypes: float64(2), int64(10), object(1)
     memory usage: 36.3+ KB
     None

     Bowling Dataset Info:
     <class 'pandas.core.frame.DataFrame'>
     ...
      16  fielder            9354 non-null    object
     dtypes: int64(8), object(9)
     memory usage: 33.8+ MB
     None
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



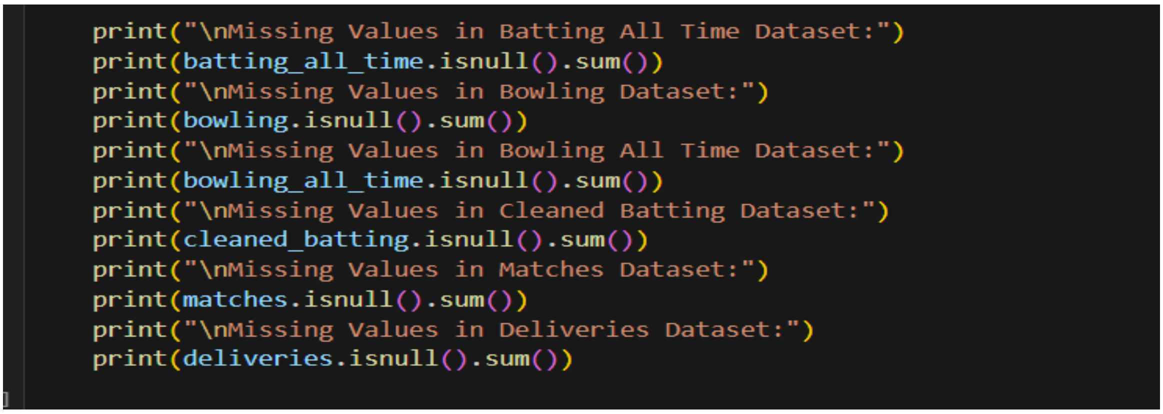
```python
print("\nBatting All Time Dataset Sample:")
print(batting_all_time.head())
print("\nBowling Dataset Sample:")
print(bowling.head())
print("\nBowling All Time Dataset Sample:")
print(bowling_all_time.head())
print("\nCleaned Batting Dataset Sample:")
print(cleaned_batting.head())
print("\nMatches Dataset Sample:")
print(matches.head())
print("\nDeliveries Dataset Sample:")
print(deliveries.head())
```

[91]

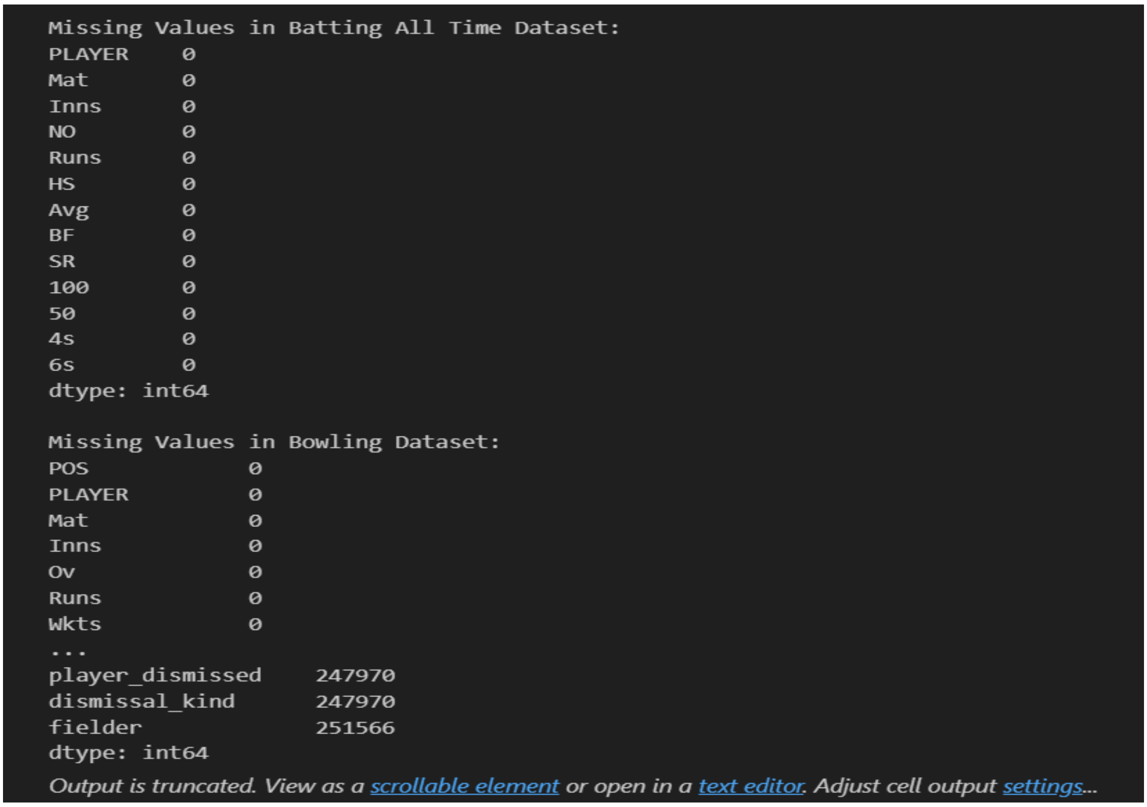


```
Batting All Time Dataset Sample:
           PLAYER  Mat  Inns  NO  Runs   HS    Avg    BF      SR  100  50  \
0     Virat Kohli  177   169  26  5412  113  37.64  4112  128.45    5  36
1    Suresh Raina  193   189  28  5368  100  34.23  3914  136.83    1  38
2    Rohit Sharma  188   183  28  4898  109  31.83  3744  130.86    1  36
3  Shikhar Dhawan  164   162  21  4619   97  30.32  3714  118.87    0  37
4    David Warner  119   119  17  4543  126  44.50  3173  143.39    4  43

    4s   6s
0  480  190
1  493  194
2  431  194
3  527   96
4  442  176

Bowling Dataset Sample:
   POS          PLAYER  Mat  Inns    Ov  Runs  Wkts  BBI    Avg  Econ     SR  \
0    1     Imran Tahir   17    17  64.2   431    26    0  16.57  6.69  14.84
1    2  Kagiso Rabada   12    12  47.0   368    25    0  14.72  7.82  11.28
2    3   Deepak Chahar   17    17  64.3   482    22    0  21.90  7.47  17.59
3    4   Shreyas Gopal   14    14  48.0   347    20    0  17.35  7.22  14.40
4    5  Jasprit Bumrah   16    16  61.4   409    19    0  21.52  6.63  19.47

   4w  5w Nationality                                Player Link  \
...
1            0        NaN          0              NaN          NaN      NaN
2            1      wides          0              NaN          NaN      NaN
3            0        NaN          0              NaN          NaN      NaN
4            0        NaN          0              NaN          NaN      NaN
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

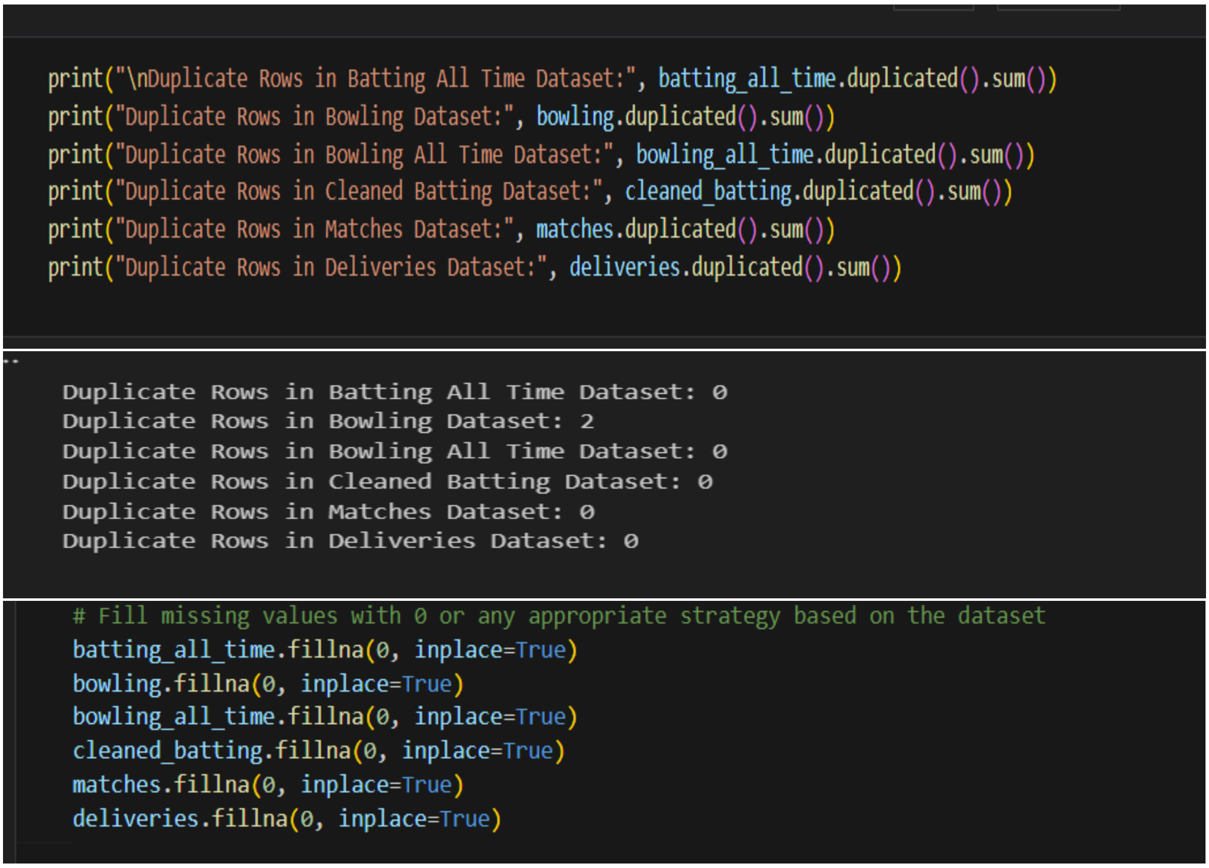


```python
print("\nMissing Values in Batting All Time Dataset:")
print(batting_all_time.isnull().sum())
print("\nMissing Values in Bowling Dataset:")
print(bowling.isnull().sum())
print("\nMissing Values in Bowling All Time Dataset:")
print(bowling_all_time.isnull().sum())
print("\nMissing Values in Cleaned Batting Dataset:")
print(cleaned_batting.isnull().sum())
print("\nMissing Values in Matches Dataset:")
print(matches.isnull().sum())
print("\nMissing Values in Deliveries Dataset:")
print(deliveries.isnull().sum())
```



```
Missing Values in Batting All Time Dataset:
PLAYER    0
Mat       0
Inns      0
NO        0
Runs      0
HS        0
Avg       0
BF        0
SR        0
100       0
50        0
4s        0
6s        0
dtype: int64

Missing Values in Bowling Dataset:
POS              0
PLAYER           0
Mat              0
Inns             0
Ov               0
Runs             0
Wkts             0
...
player_dismissed     247970
dismissal_kind       247970
fielder              251566
dtype: int64
```

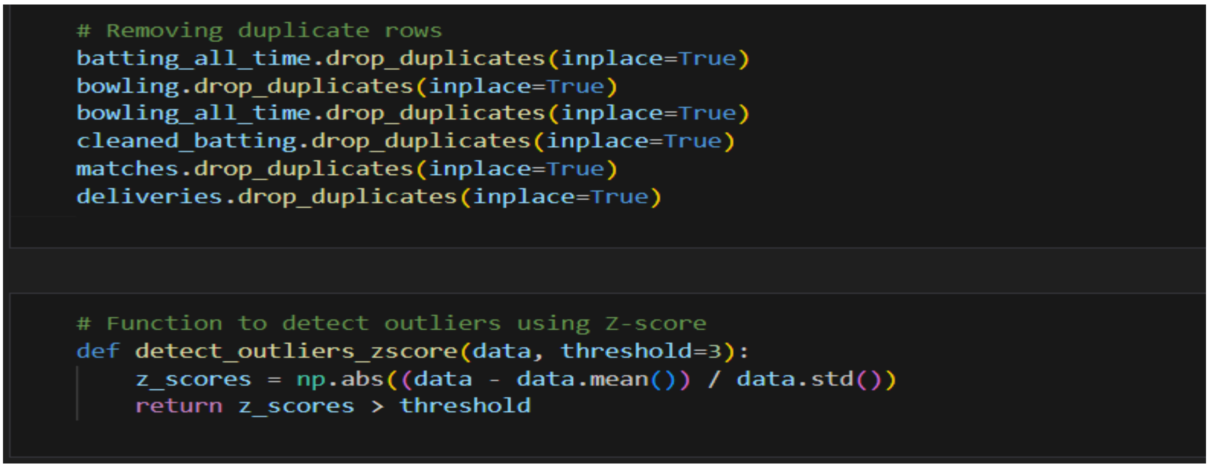
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



```python
print("\nDuplicate Rows in Batting All Time Dataset:", batting_all_time.duplicated().sum())
print("Duplicate Rows in Bowling Dataset:", bowling.duplicated().sum())
print("Duplicate Rows in Bowling All Time Dataset:", bowling_all_time.duplicated().sum())
print("Duplicate Rows in Cleaned Batting Dataset:", cleaned_batting.duplicated().sum())
print("Duplicate Rows in Matches Dataset:", matches.duplicated().sum())
print("Duplicate Rows in Deliveries Dataset:", deliveries.duplicated().sum())
```

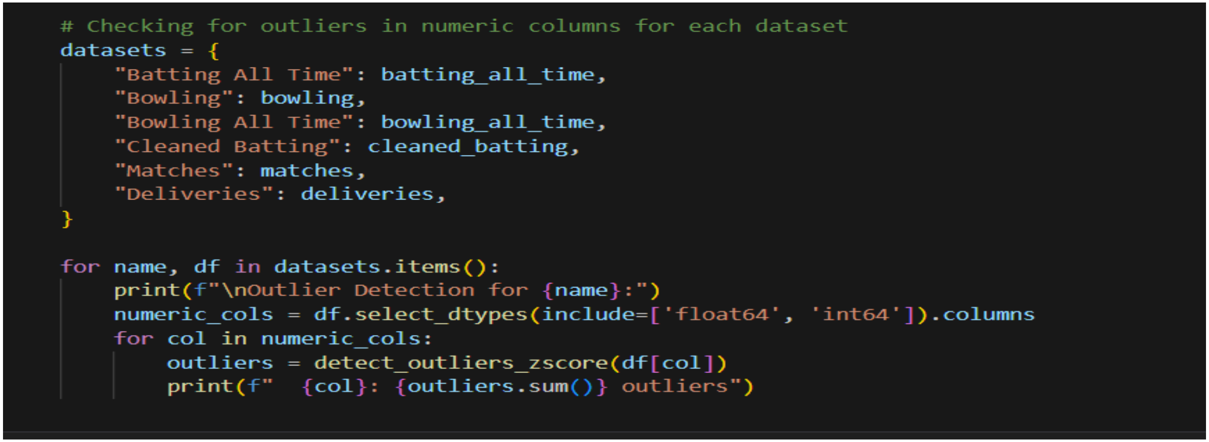
```
Duplicate Rows in Batting All Time Dataset: 0
Duplicate Rows in Bowling Dataset: 2
Duplicate Rows in Bowling All Time Dataset: 0
Duplicate Rows in Cleaned Batting Dataset: 0
Duplicate Rows in Matches Dataset: 0
Duplicate Rows in Deliveries Dataset: 0
```

```python
# Fill missing values with 0 or any appropriate strategy based on the dataset
batting_all_time.fillna(0, inplace=True)
bowling.fillna(0, inplace=True)
bowling_all_time.fillna(0, inplace=True)
cleaned_batting.fillna(0, inplace=True)
matches.fillna(0, inplace=True)
deliveries.fillna(0, inplace=True)
```

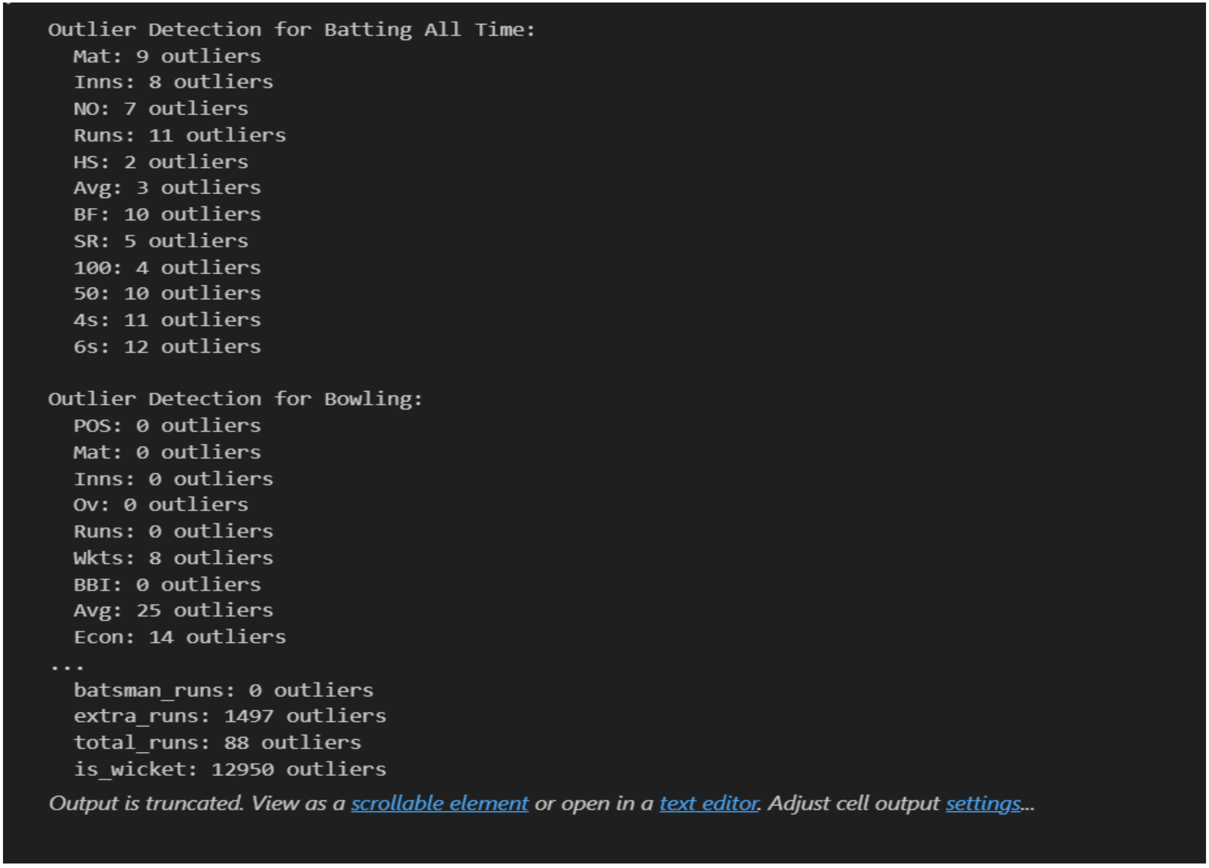


```python
# Removing duplicate rows
batting_all_time.drop_duplicates(inplace=True)
bowling.drop_duplicates(inplace=True)
bowling_all_time.drop_duplicates(inplace=True)
cleaned_batting.drop_duplicates(inplace=True)
matches.drop_duplicates(inplace=True)
deliveries.drop_duplicates(inplace=True)
```

```python
# Function to detect outliers using Z-score
def detect_outliers_zscore(data, threshold=3):
    z_scores = np.abs((data - data.mean()) / data.std())
    return z_scores > threshold
```

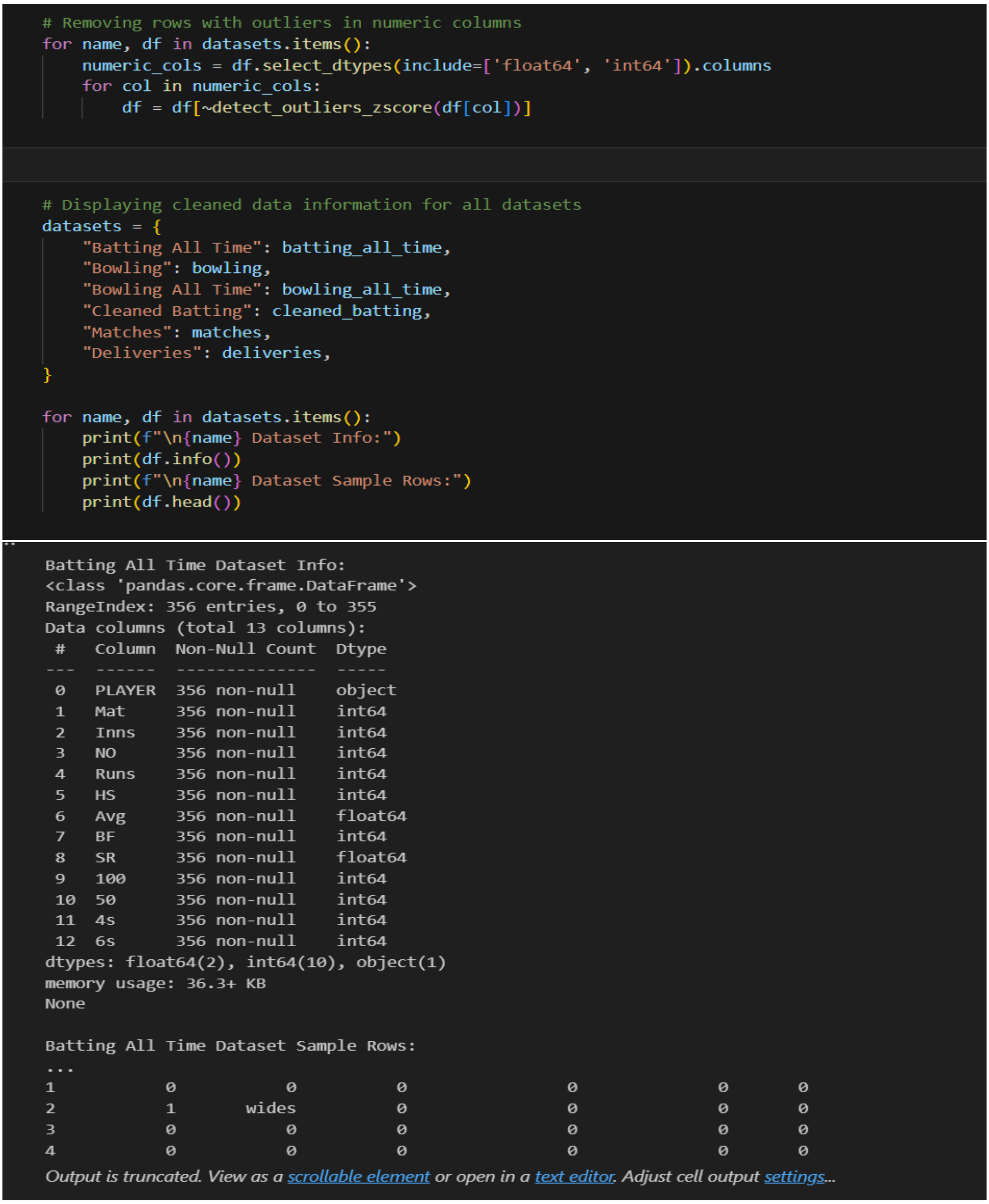


```python
# Checking for outliers in numeric columns for each dataset
datasets = {
    "Batting All Time": batting_all_time,
    "Bowling": bowling,
    "Bowling All Time": bowling_all_time,
    "Cleaned Batting": cleaned_batting,
    "Matches": matches,
    "Deliveries": deliveries,
}

for name, df in datasets.items():
    print(f"\nOutlier Detection for {name}:")
    numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
    for col in numeric_cols:
        outliers = detect_outliers_zscore(df[col])
        print(f"  {col}: {outliers.sum()} outliers")
```



```
Outlier Detection for Batting All Time:
  Mat: 9 outliers
  Inns: 8 outliers
  NO: 7 outliers
  Runs: 11 outliers
  HS: 2 outliers
  Avg: 3 outliers
  BF: 10 outliers
  SR: 5 outliers
  100: 4 outliers
  50: 10 outliers
  4s: 11 outliers
  6s: 12 outliers

Outlier Detection for Bowling:
  POS: 0 outliers
  Mat: 0 outliers
  Inns: 0 outliers
  Ov: 0 outliers
  Runs: 0 outliers
  Wkts: 8 outliers
  BBI: 0 outliers
  Avg: 25 outliers
  Econ: 14 outliers
...
  batsman_runs: 0 outliers
  extra_runs: 1497 outliers
  total_runs: 88 outliers
  is_wicket: 12950 outliers
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



```python
# Removing rows with outliers in numeric columns
for name, df in datasets.items():
    numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns
    for col in numeric_cols:
        df = df[~detect_outliers_zscore(df[col])]
```

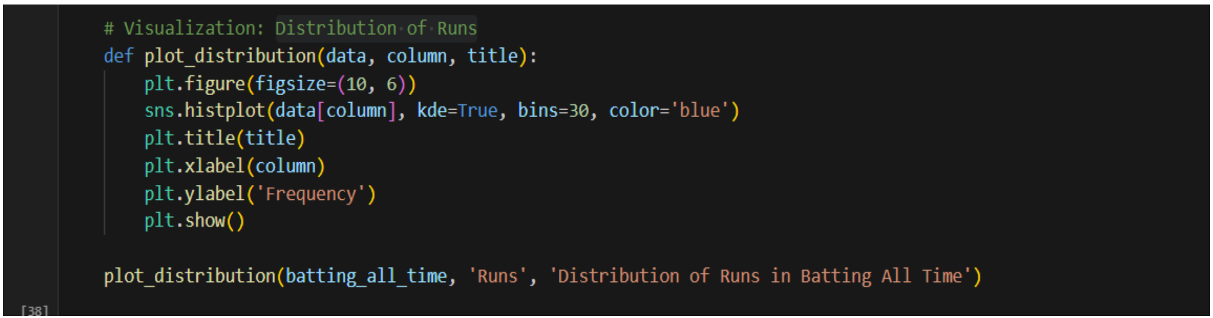
```python
# Displaying cleaned data information for all datasets
datasets = {
    "Batting All Time": batting_all_time,
    "Bowling": bowling,
    "Bowling All Time": bowling_all_time,
    "Cleaned Batting": cleaned_batting,
    "Matches": matches,
    "Deliveries": deliveries,
}

for name, df in datasets.items():
    print(f"\n{name} Dataset Info:")
    print(df.info())
    print(f"\n{name} Dataset Sample Rows:")
    print(df.head())
```

```
Batting All Time Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 356 entries, 0 to 355
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   PLAYER  356 non-null    object
 1   Mat     356 non-null    int64
 2   Inns    356 non-null    int64
 3   NO      356 non-null    int64
 4   Runs    356 non-null    int64
 5   HS      356 non-null    int64
 6   Avg     356 non-null    float64
 7   BF      356 non-null    int64
 8   SR      356 non-null    float64
 9   100     356 non-null    int64
 10  50      356 non-null    int64
 11  4s      356 non-null    int64
 12  6s      356 non-null    int64
dtypes: float64(2), int64(10), object(1)
memory usage: 36.3+ KB
None

Batting All Time Dataset Sample Rows:
...
1           0           0          0                0              0        0
2           1       wides          0                0              0        0
3           0           0          0                0              0        0
4           0           0          0                0              0        0
```

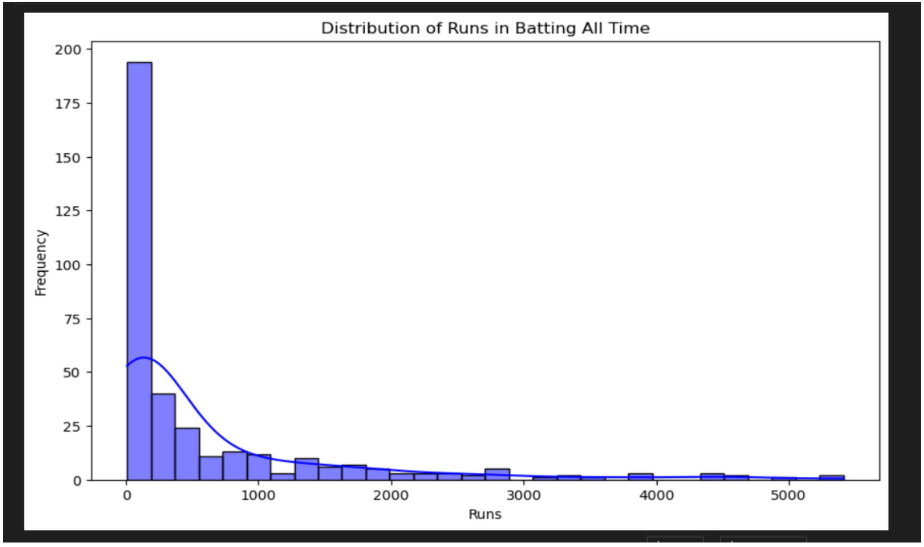
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Visualizations:

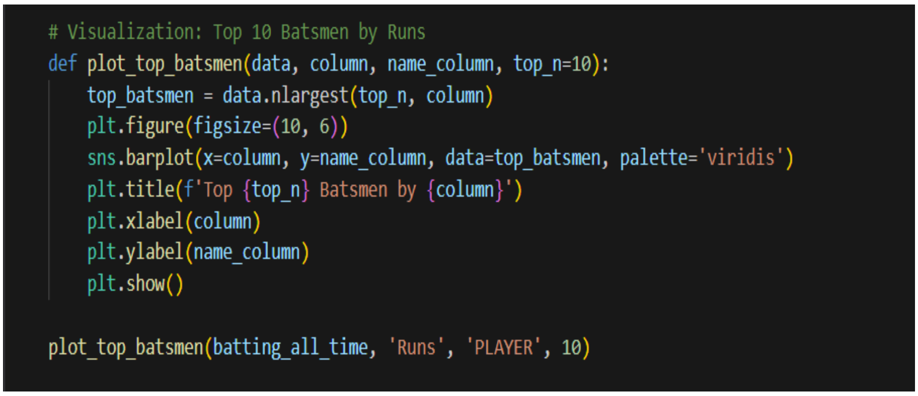
Distribution of Runs:



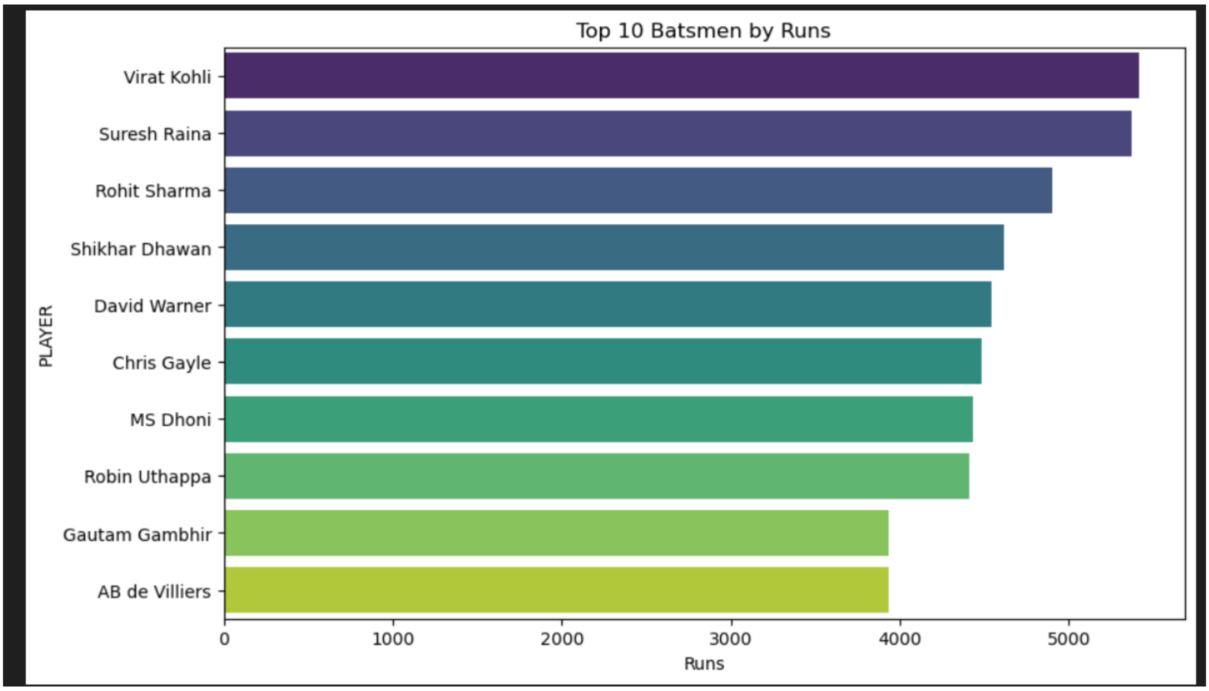
```python
# Visualization: Distribution of Runs
def plot_distribution(data, column, title):
    plt.figure(figsize=(10, 6))
    sns.histplot(data[column], kde=True, bins=30, color='blue')
    plt.title(title)
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()

plot_distribution(batting_all_time, 'Runs', 'Distribution of Runs in Batting All Time')
```



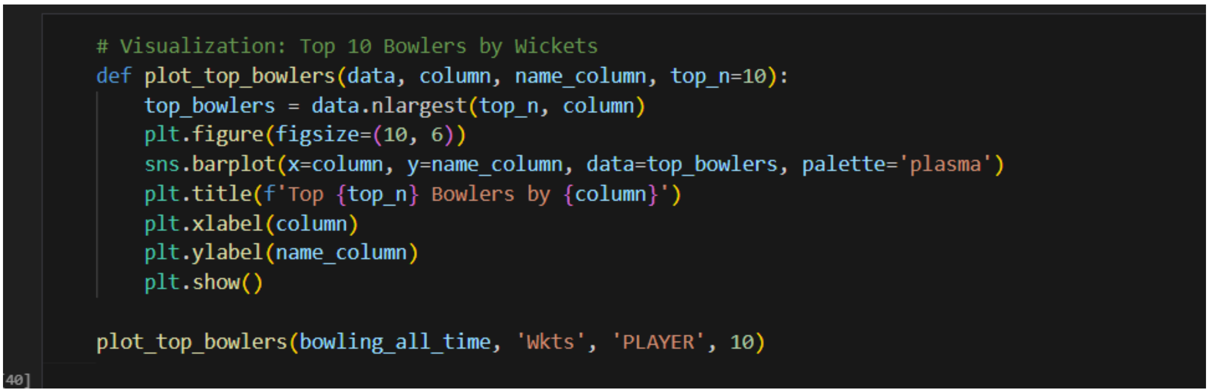
Top 10 Batsmen by Runs:



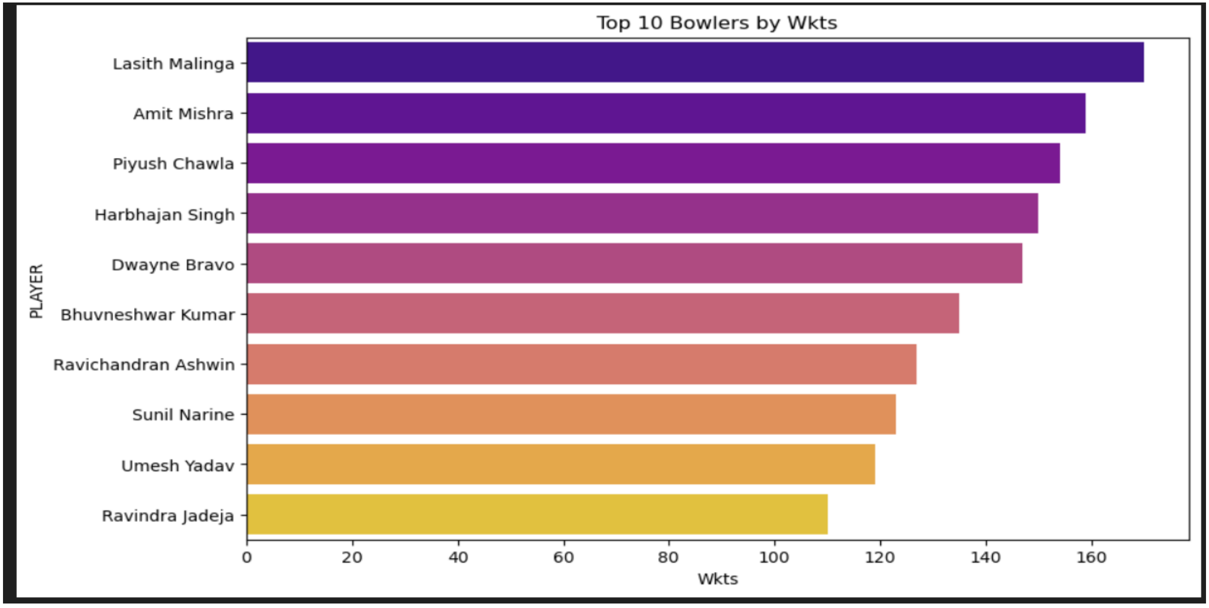
```python
# Visualization: Top 10 Batsmen by Runs
def plot_top_batsmen(data, column, name_column, top_n=10):
    top_batsmen = data.nlargest(top_n, column)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=column, y=name_column, data=top_batsmen, palette='viridis')
    plt.title(f'Top {top_n} Batsmen by {column}')
    plt.xlabel(column)
    plt.ylabel(name_column)
    plt.show()

plot_top_batsmen(batting_all_time, 'Runs', 'PLAYER', 10)
```



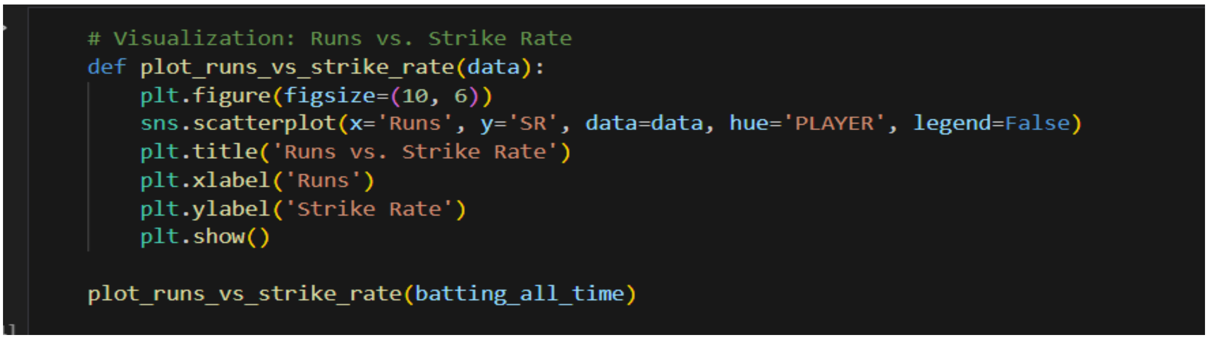
Top 10 Bowlers by Wickets:



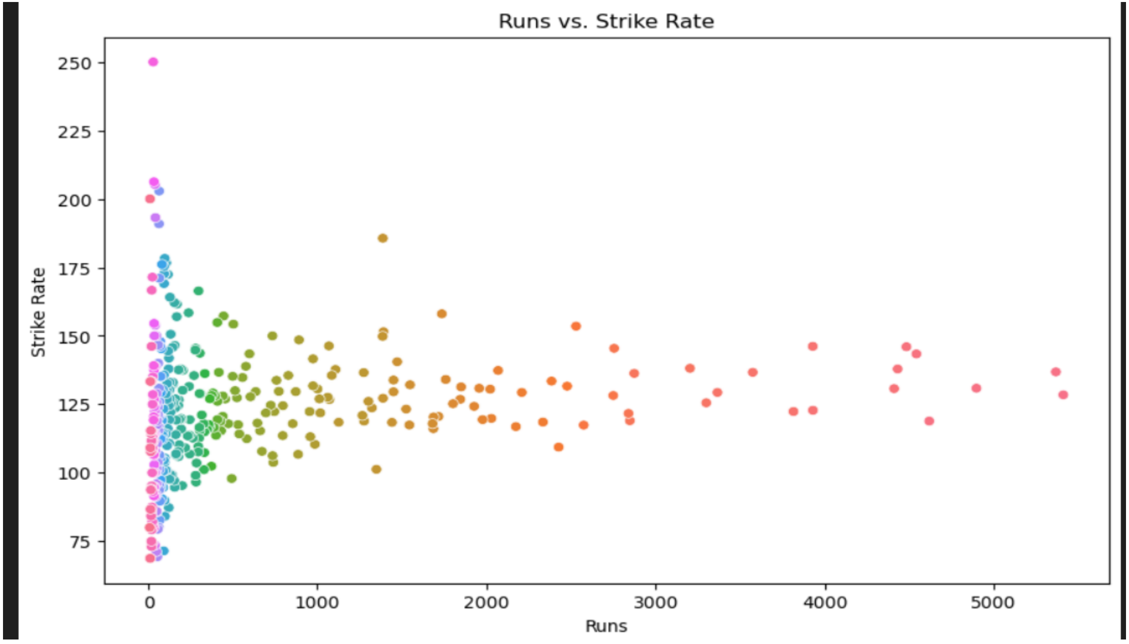
```
# Visualization: Top 10 Bowlers by Wickets
def plot_top_bowlers(data, column, name_column, top_n=10):
    top_bowlers = data.nlargest(top_n, column)
    plt.figure(figsize=(10, 6))
    sns.barplot(x=column, y=name_column, data=top_bowlers, palette='plasma')
    plt.title(f'Top {top_n} Bowlers by {column}')
    plt.xlabel(column)
    plt.ylabel(name_column)
    plt.show()

plot_top_bowlers(bowling_all_time, 'Wkts', 'PLAYER', 10)
```



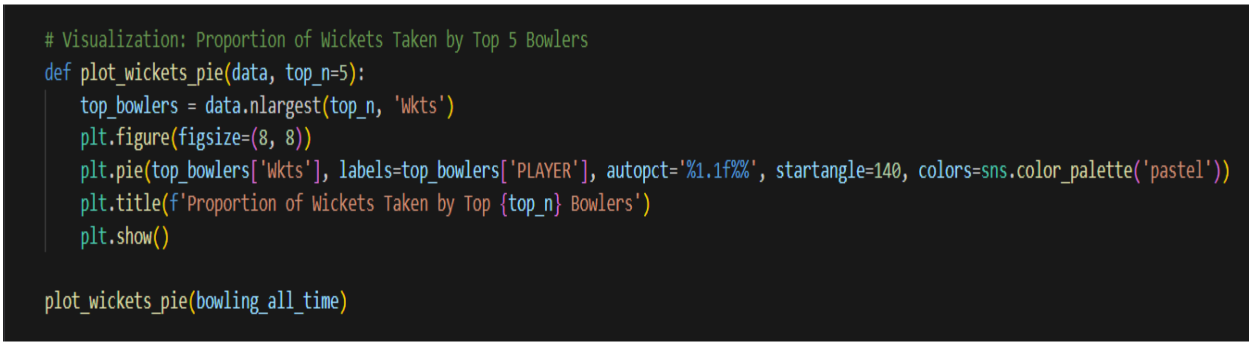
Runs vs Strike Rate:



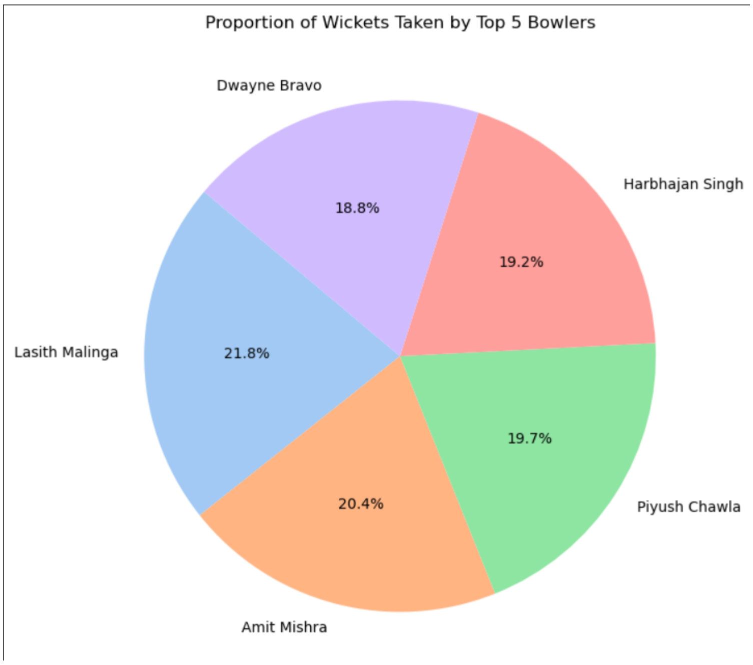
```
# Visualization: Runs vs. Strike Rate
def plot_runs_vs_strike_rate(data):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x='Runs', y='SR', data=data, hue='PLAYER', legend=False)
    plt.title('Runs vs. Strike Rate')
    plt.xlabel('Runs')
    plt.ylabel('Strike Rate')
    plt.show()

plot_runs_vs_strike_rate(batting_all_time)
```



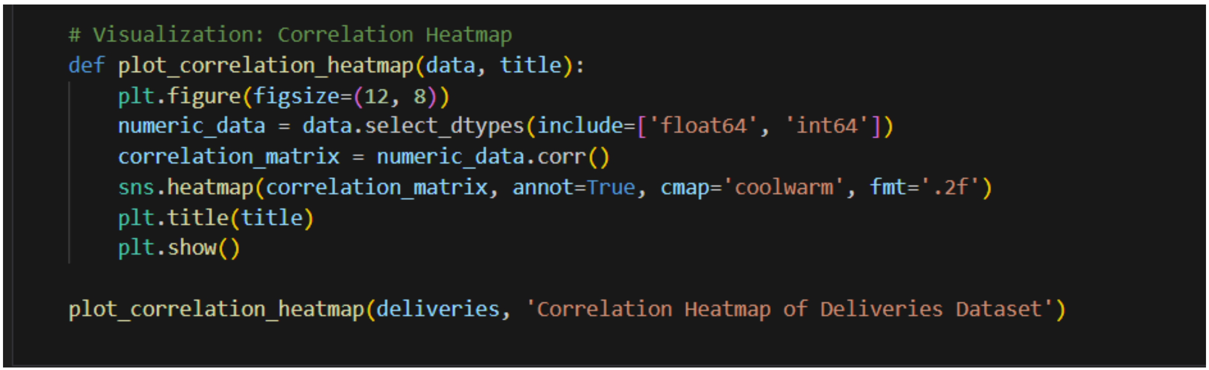
Proportion of Wickets taken by top 5 bowler:



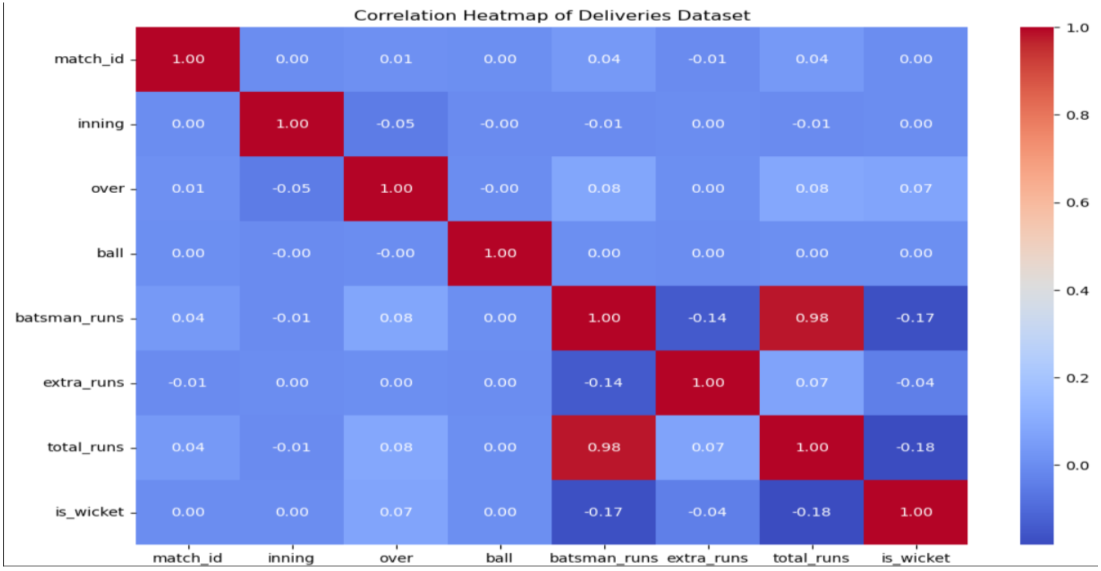
```
# Visualization: Proportion of Wickets Taken by Top 5 Bowlers
def plot_wickets_pie(data, top_n=5):
    top_bowlers = data.nlargest(top_n, 'Wkts')
    plt.figure(figsize=(8, 8))
    plt.pie(top_bowlers['Wkts'], labels=top_bowlers['PLAYER'], autopct='%1.1f%%', startangle=140, colors=sns.color_palette('pastel'))
    plt.title(f'Proportion of Wickets Taken by Top {top_n} Bowlers')
    plt.show()

plot_wickets_pie(bowling_all_time)
```

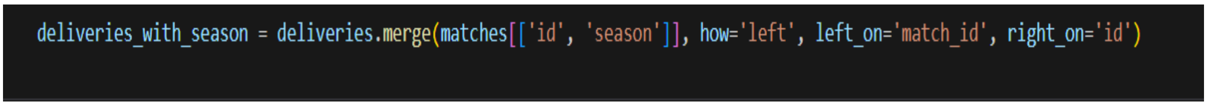


Corelation Heatmap:



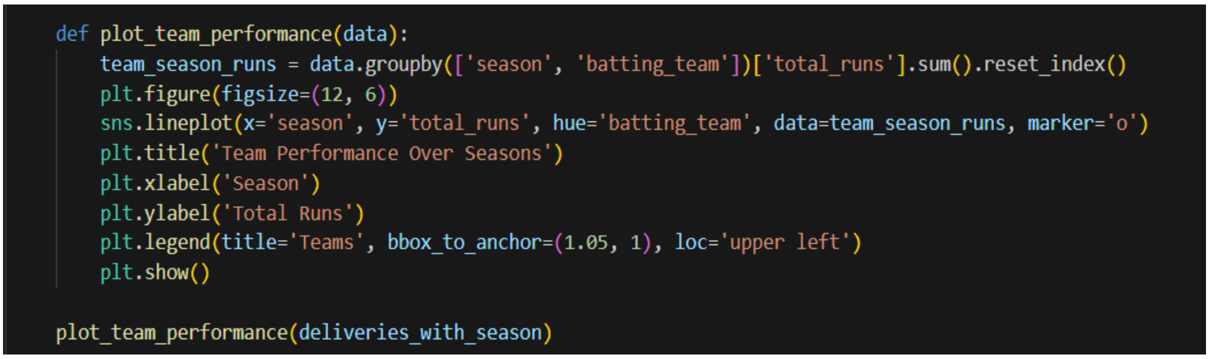
```
# Visualization: Correlation Heatmap
def plot_correlation_heatmap(data, title):
    plt.figure(figsize=(12, 8))
    numeric_data = data.select_dtypes(include=['float64', 'int64'])
    correlation_matrix = numeric_data.corr()
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
    plt.title(title)
    plt.show()

plot_correlation_heatmap(deliveries, 'Correlation Heatmap of Deliveries Dataset')
```



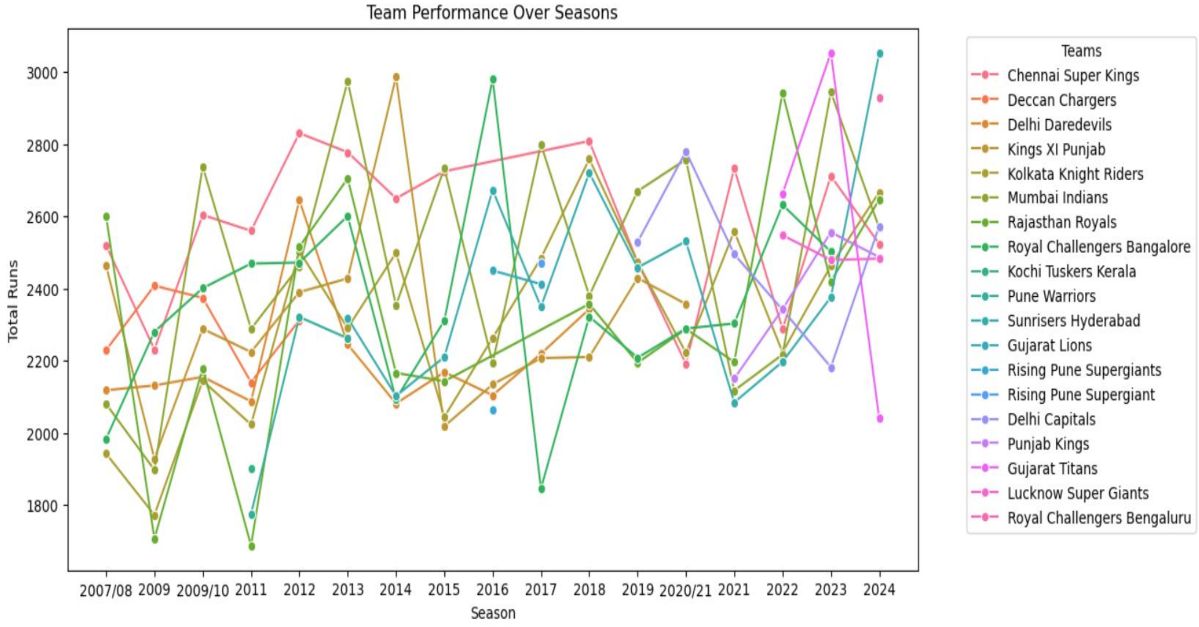


```
deliveries_with_season = deliveries.merge(matches[['id', 'season']], how='left', left_on='match_id', right_on='id')
```

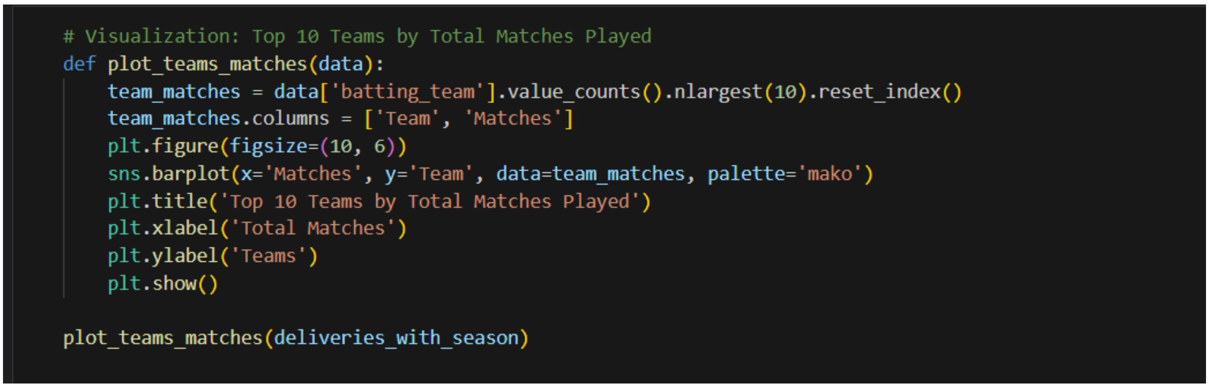
Team performance over seasons:



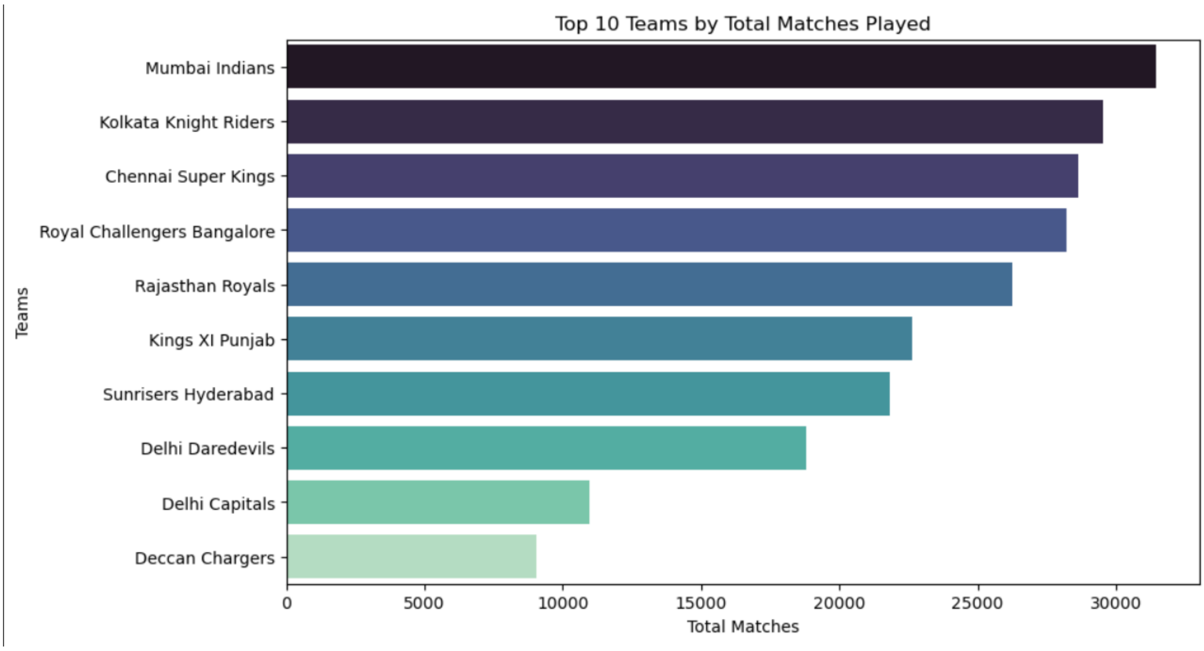
```
def plot_team_performance(data):
    team_season_runs = data.groupby(['season', 'batting_team'])['total_runs'].sum().reset_index()
    plt.figure(figsize=(12, 6))
    sns.lineplot(x='season', y='total_runs', hue='batting_team', data=team_season_runs, marker='o')
    plt.title('Team Performance Over Seasons')
    plt.xlabel('Season')
    plt.ylabel('Total Runs')
    plt.legend(title='Teams', bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()

plot_team_performance(deliveries_with_season)
```



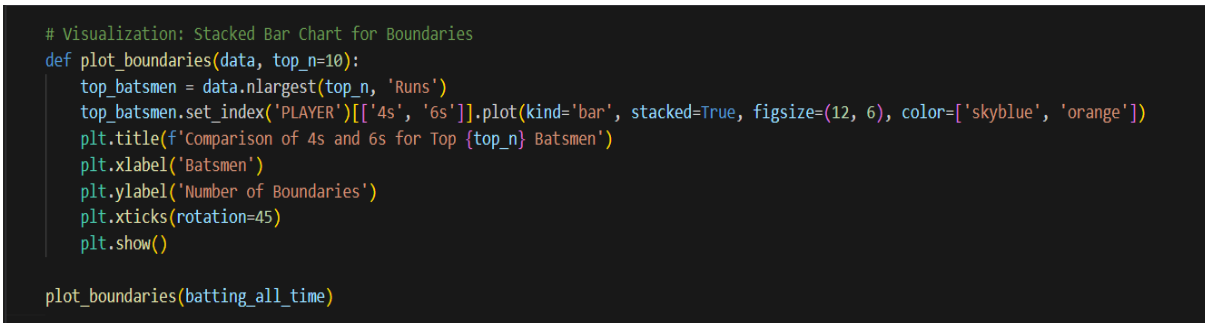
Top ten teams by total matches played:



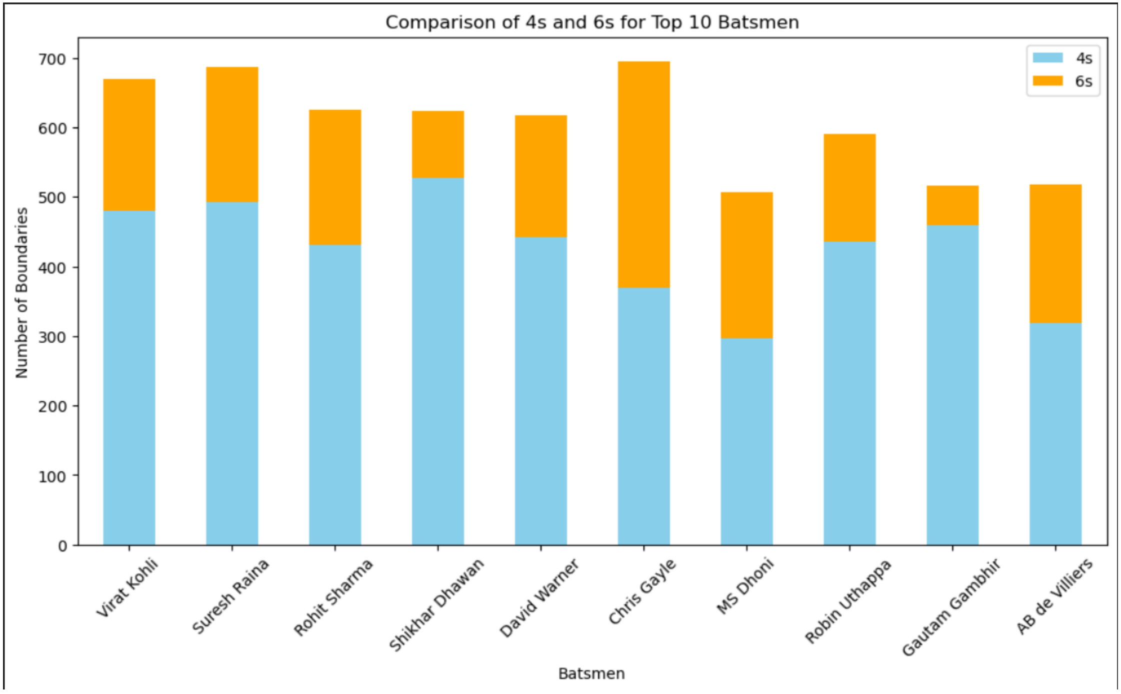
```
# Visualization: Top 10 Teams by Total Matches Played
def plot_teams_matches(data):
    team_matches = data['batting_team'].value_counts().nlargest(10).reset_index()
    team_matches.columns = ['Team', 'Matches']
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Matches', y='Team', data=team_matches, palette='mako')
    plt.title('Top 10 Teams by Total Matches Played')
    plt.xlabel('Total Matches')
    plt.ylabel('Teams')
    plt.show()

plot_teams_matches(deliveries_with_season)
```



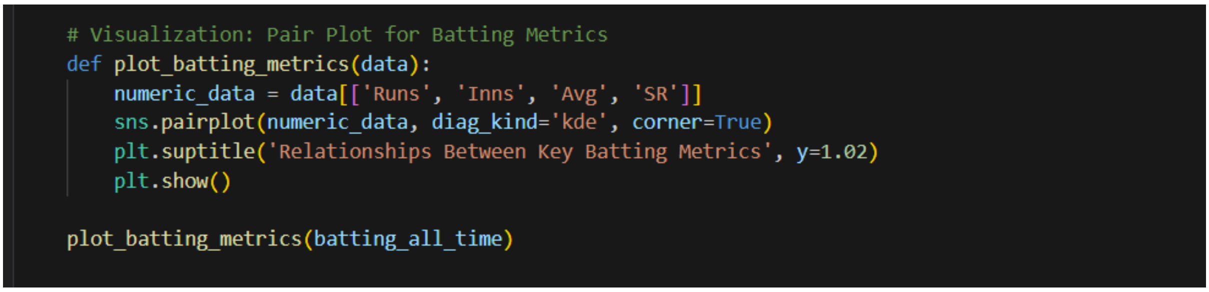
Stacked bar charts for Boundaries:



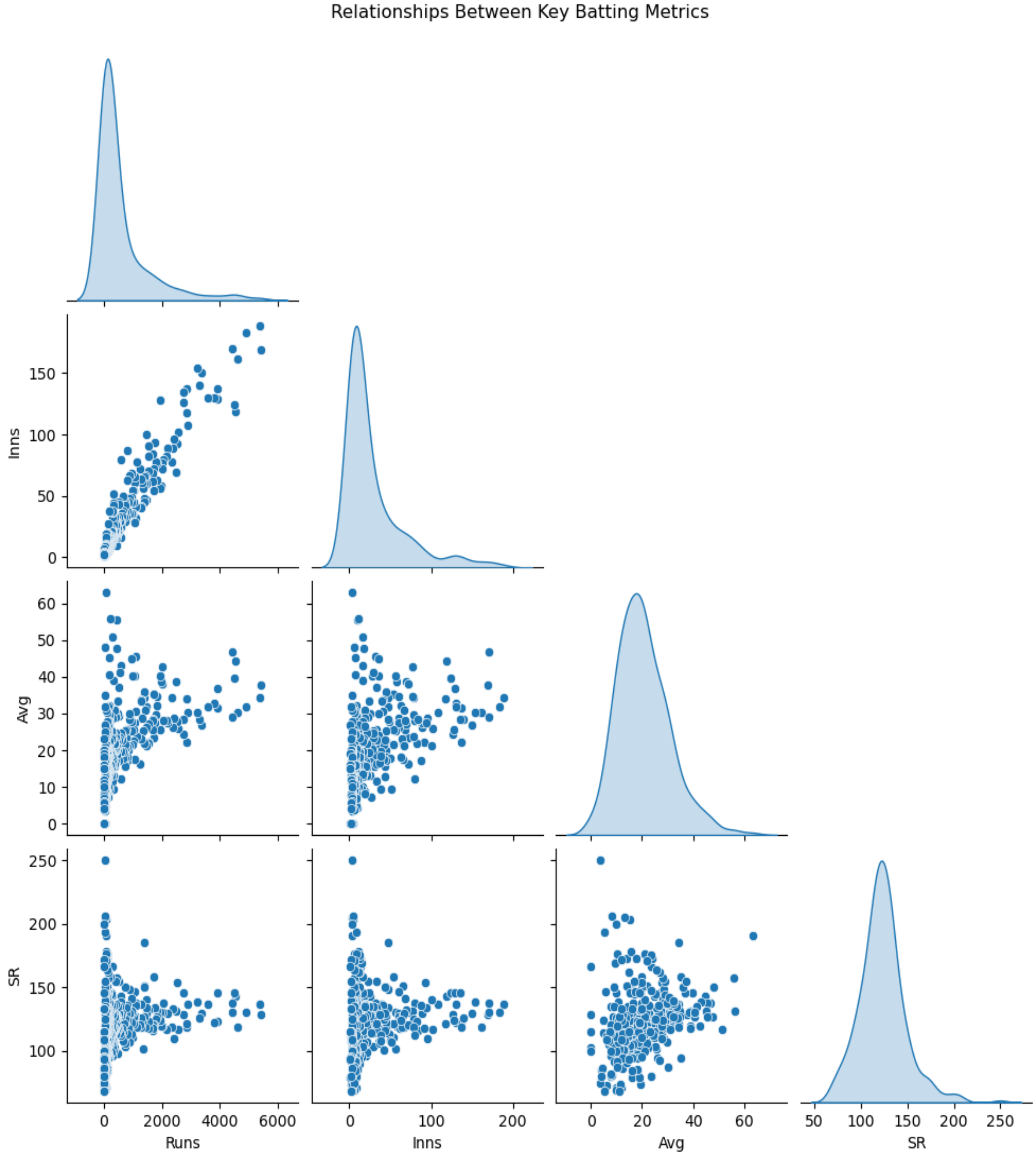
```
# Visualization: Stacked Bar Chart for Boundaries
def plot_boundaries(data, top_n=10):
    top_batsmen = data.nlargest(top_n, 'Runs')
    top_batsmen.set_index('PLAYER')[['4s', '6s']].plot(kind='bar', stacked=True, figsize=(12, 6), color=['skyblue', 'orange'])
    plt.title(f'Comparison of 4s and 6s for Top {top_n} Batsmen')
    plt.xlabel('Batsmen')
    plt.ylabel('Number of Boundaries')
    plt.xticks(rotation=45)
    plt.show()

plot_boundaries(batting_all_time)
```



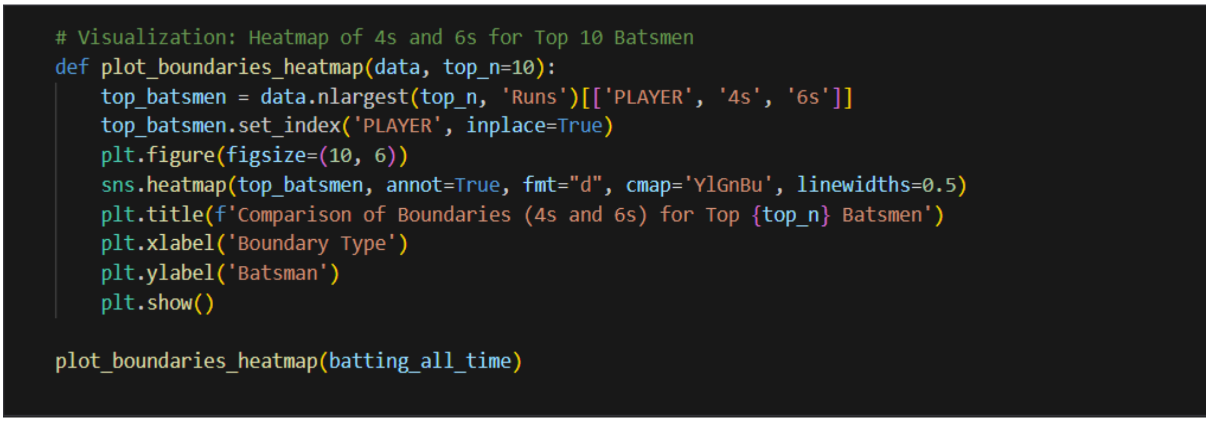
Pair plot for batting metrics:



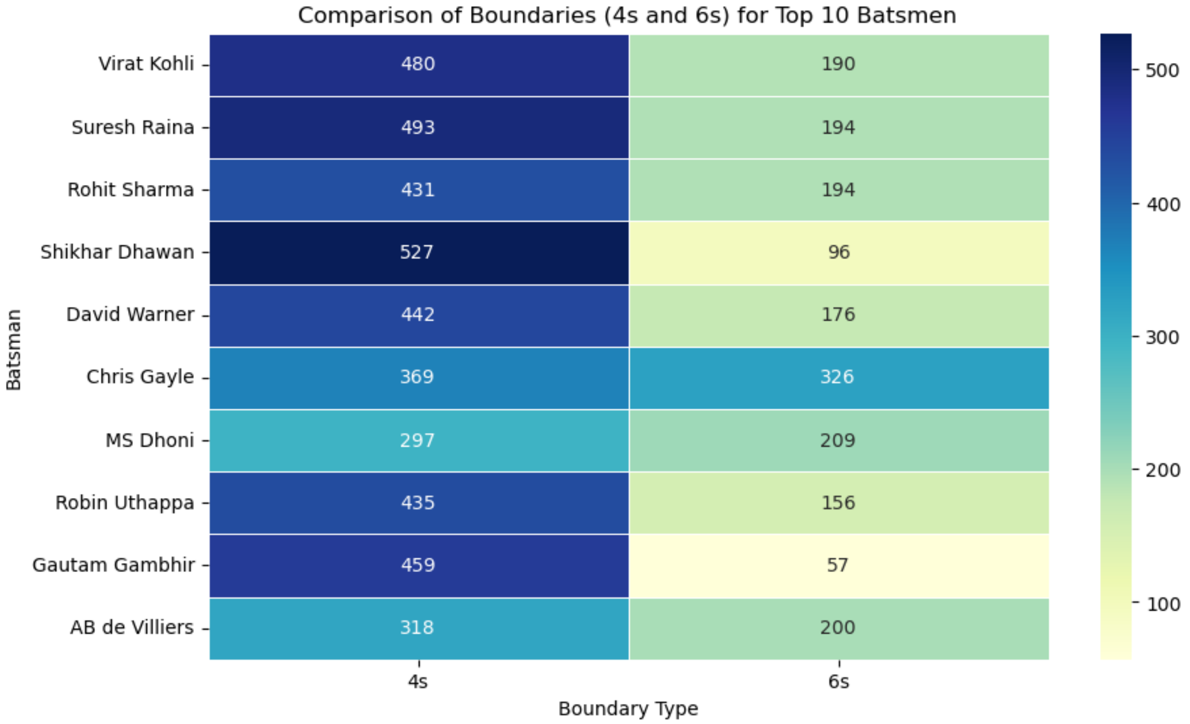
```
# Visualization: Pair Plot for Batting Metrics
def plot_batting_metrics(data):
    numeric_data = data[['Runs', 'Inns', 'Avg', 'SR']]
    sns.pairplot(numeric_data, diag_kind='kde', corner=True)
    plt.suptitle('Relationships Between Key Batting Metrics', y=1.02)
    plt.show()

plot_batting_metrics(batting_all_time)
```



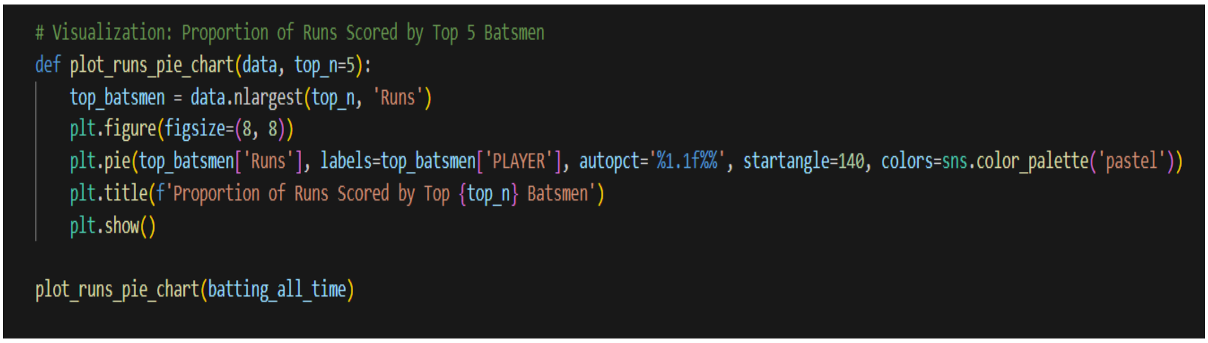
Heatmap of 4s and 6s for Top 10 Batsmen:



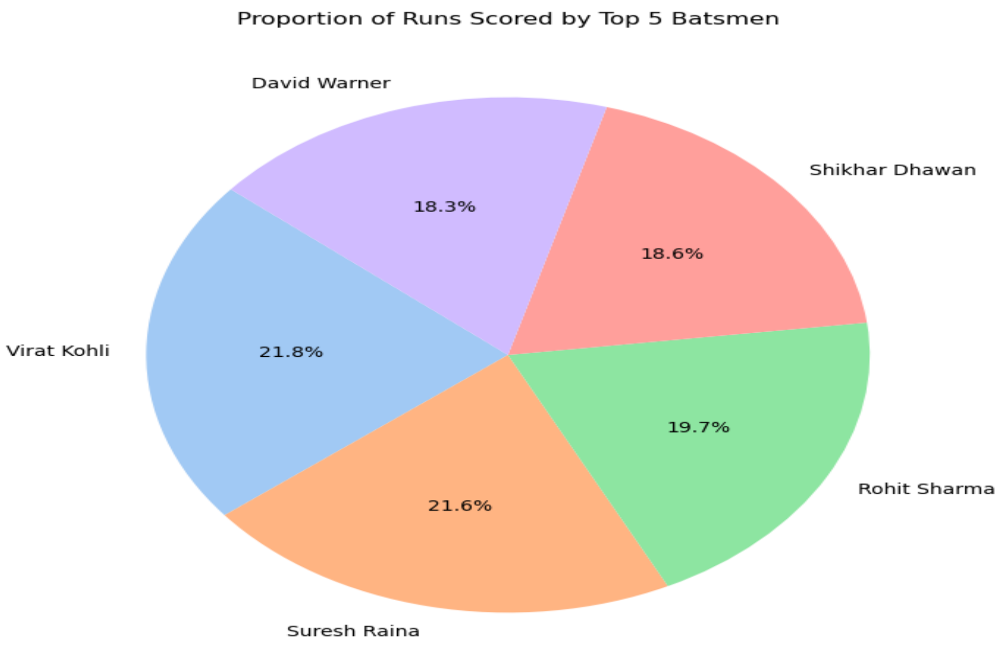
```
# Visualization: Heatmap of 4s and 6s for Top 10 Batsmen
def plot_boundaries_heatmap(data, top_n=10):
    top_batsmen = data.nlargest(top_n, 'Runs')[['PLAYER', '4s', '6s']]
    top_batsmen.set_index('PLAYER', inplace=True)
    plt.figure(figsize=(10, 6))
    sns.heatmap(top_batsmen, annot=True, fmt="d", cmap='YlGnBu', linewidths=0.5)
    plt.title(f'Comparison of Boundaries (4s and 6s) for Top {top_n} Batsmen')
    plt.xlabel('Boundary Type')
    plt.ylabel('Batsman')
    plt.show()

plot_boundaries_heatmap(batting_all_time)
```



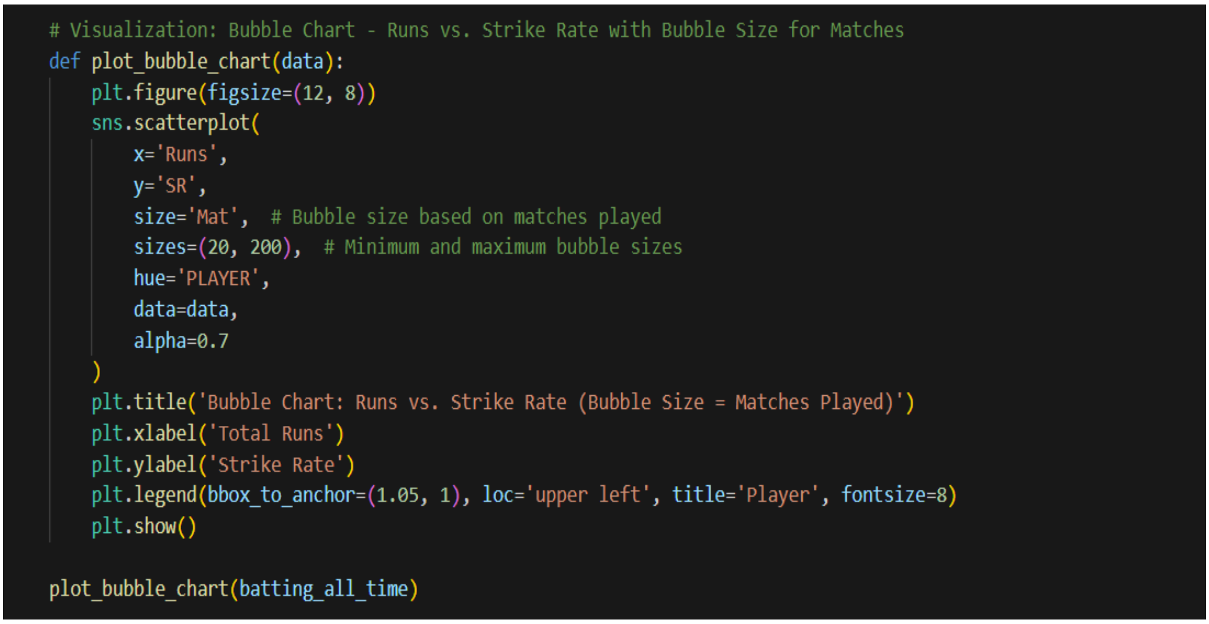
Proportion of Runs Scored by Top 5 Batsmen:



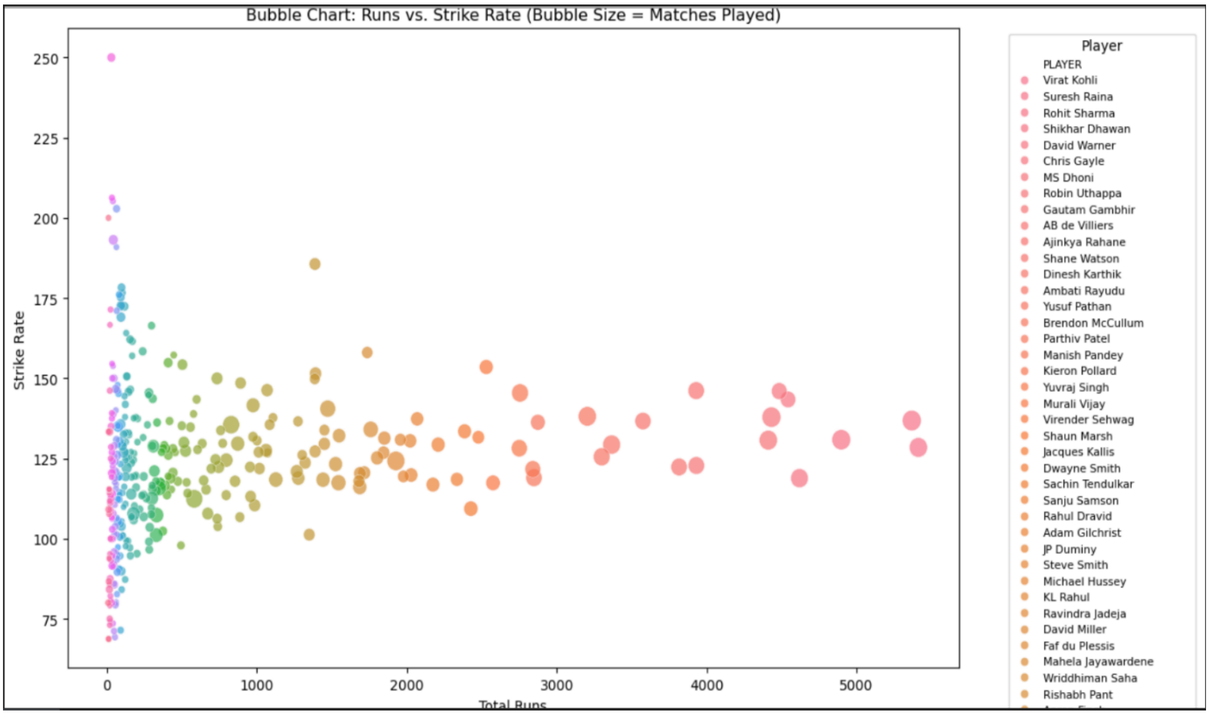
```
# Visualization: Proportion of Runs Scored by Top 5 Batsmen
def plot_runs_pie_chart(data, top_n=5):
    top_batsmen = data.nlargest(top_n, 'Runs')
    plt.figure(figsize=(8, 8))
    plt.pie(top_batsmen['Runs'], labels=top_batsmen['PLAYER'], autopct='%1.1f%%', startangle=140, colors=sns.color_palette('pastel'))
    plt.title(f'Proportion of Runs Scored by Top {top_n} Batsmen')
    plt.show()

plot_runs_pie_chart(batting_all_time)
```



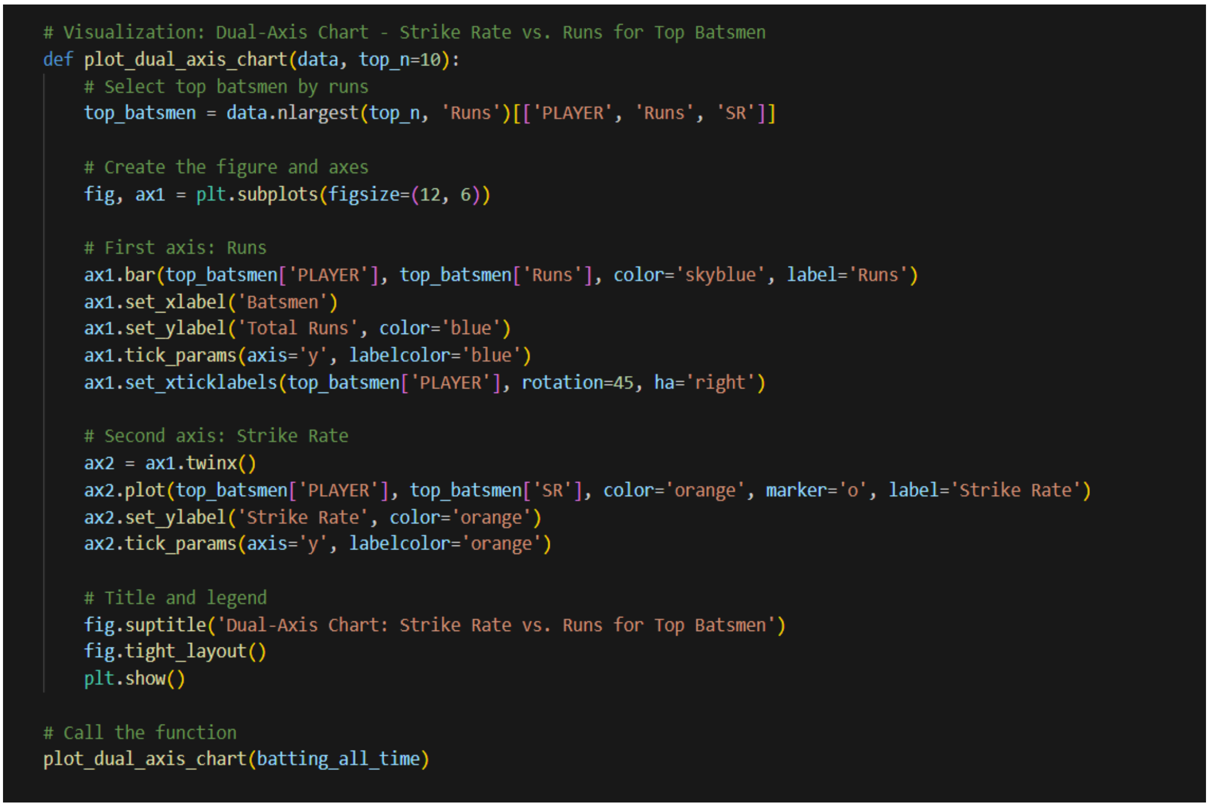
Runs vs Strike Rate with Bubble size for Matches:



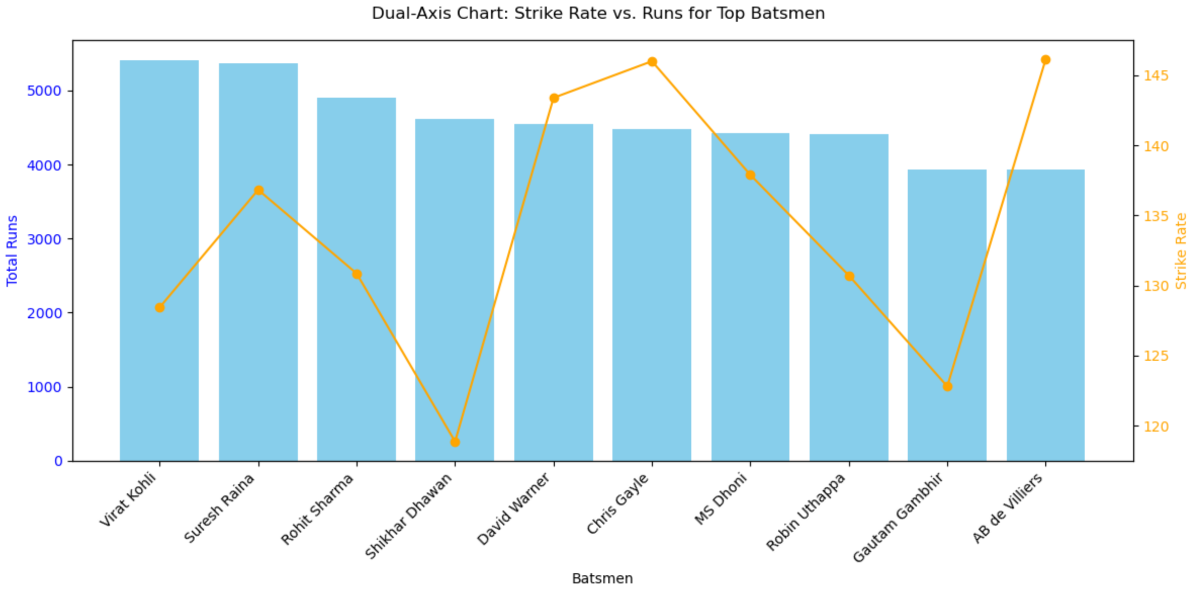
```python
# Visualization: Bubble Chart - Runs vs. Strike Rate with Bubble Size for Matches
def plot_bubble_chart(data):
    plt.figure(figsize=(12, 8))
    sns.scatterplot(
        x='Runs',
        y='SR',
        size='Mat',  # Bubble size based on matches played
        sizes=(20, 200),  # Minimum and maximum bubble sizes
        hue='PLAYER',
        data=data,
        alpha=0.7
    )
    plt.title('Bubble Chart: Runs vs. Strike Rate (Bubble Size = Matches Played)')
    plt.xlabel('Total Runs')
    plt.ylabel('Strike Rate')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', title='Player', fontsize=8)
    plt.show()

plot_bubble_chart(batting_all_time)
```

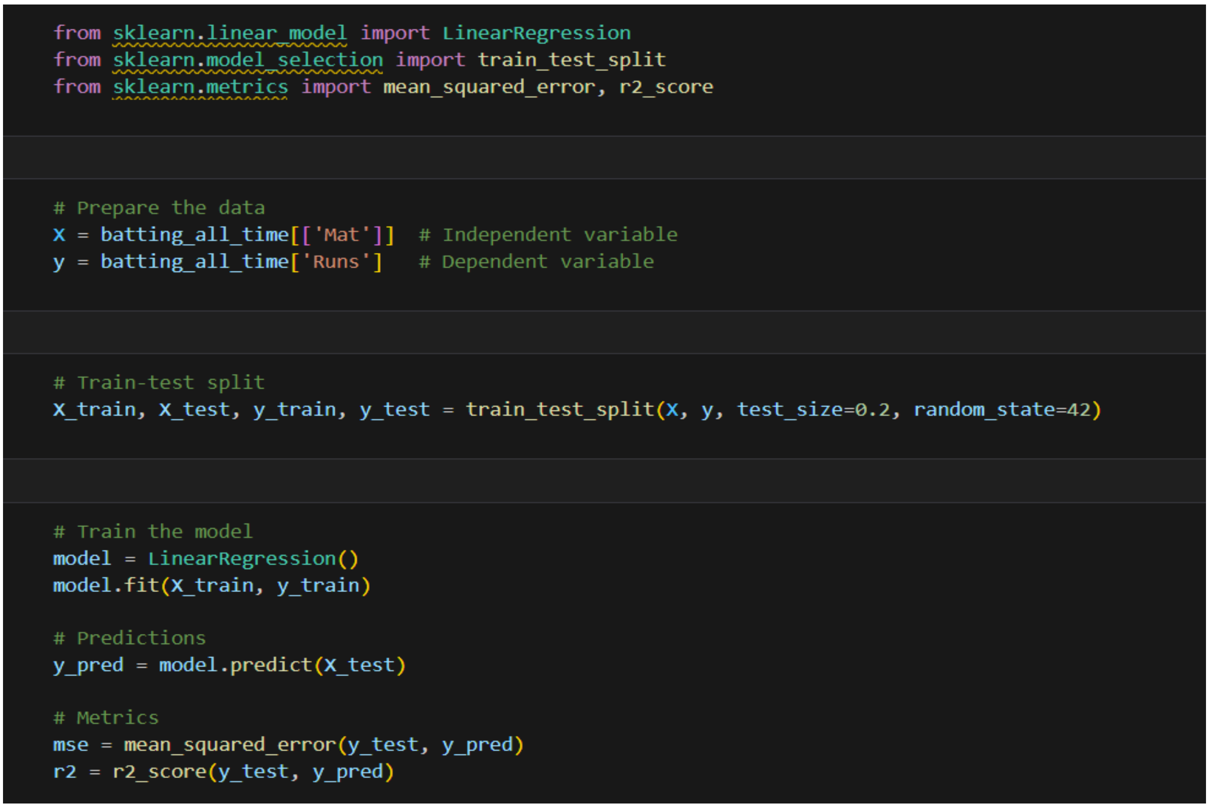


Strike rate vs Runs for Top Batsmen:



```
# Visualization: Dual-Axis Chart - Strike Rate vs. Runs for Top Batsmen
def plot_dual_axis_chart(data, top_n=10):
    # Select top batsmen by runs
    top_batsmen = data.nlargest(top_n, 'Runs')[['PLAYER', 'Runs', 'SR']]

    # Create the figure and axes
    fig, ax1 = plt.subplots(figsize=(12, 6))

    # First axis: Runs
    ax1.bar(top_batsmen['PLAYER'], top_batsmen['Runs'], color='skyblue', label='Runs')
    ax1.set_xlabel('Batsmen')
    ax1.set_ylabel('Total Runs', color='blue')
    ax1.tick_params(axis='y', labelcolor='blue')
    ax1.set_xticklabels(top_batsmen['PLAYER'], rotation=45, ha='right')

    # Second axis: Strike Rate
    ax2 = ax1.twinx()
    ax2.plot(top_batsmen['PLAYER'], top_batsmen['SR'], color='orange', marker='o', label='Strike Rate')
    ax2.set_ylabel('Strike Rate', color='orange')
    ax2.tick_params(axis='y', labelcolor='orange')

    # Title and legend
    fig.suptitle('Dual-Axis Chart: Strike Rate vs. Runs for Top Batsmen')
    fig.tight_layout()
    plt.show()

# Call the function
plot_dual_axis_chart(batting_all_time)
```



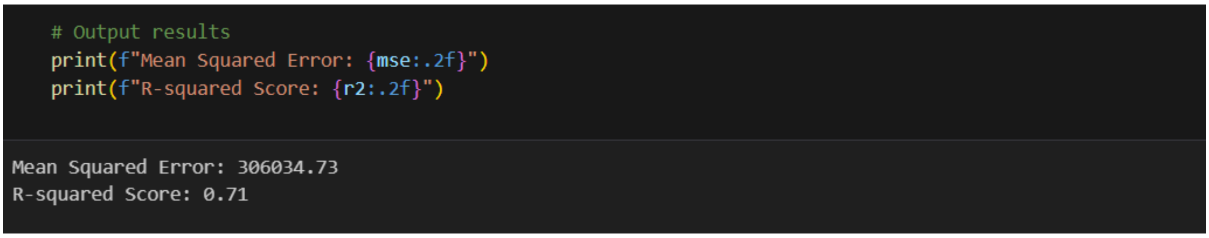


```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Prepare the data
X = batting_all_time[['Mat']]  # Independent variable
y = batting_all_time['Runs']   # Dependent variable
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

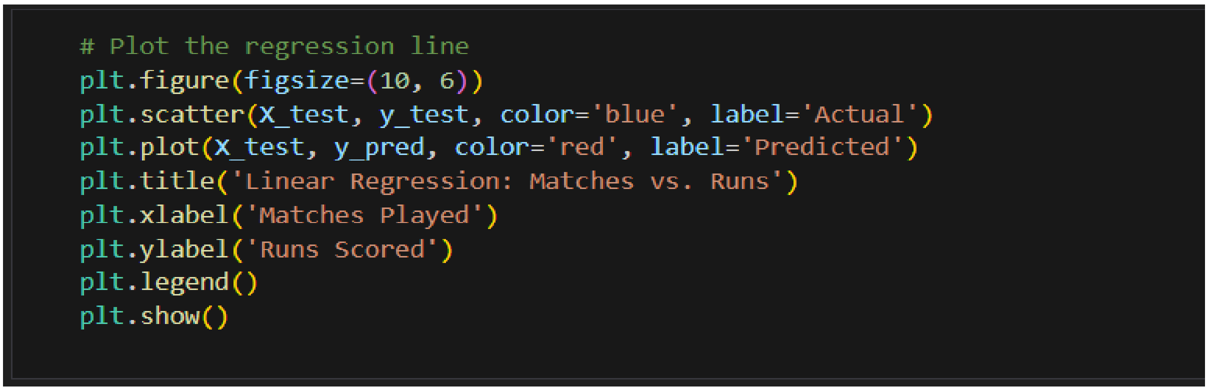
```
# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```



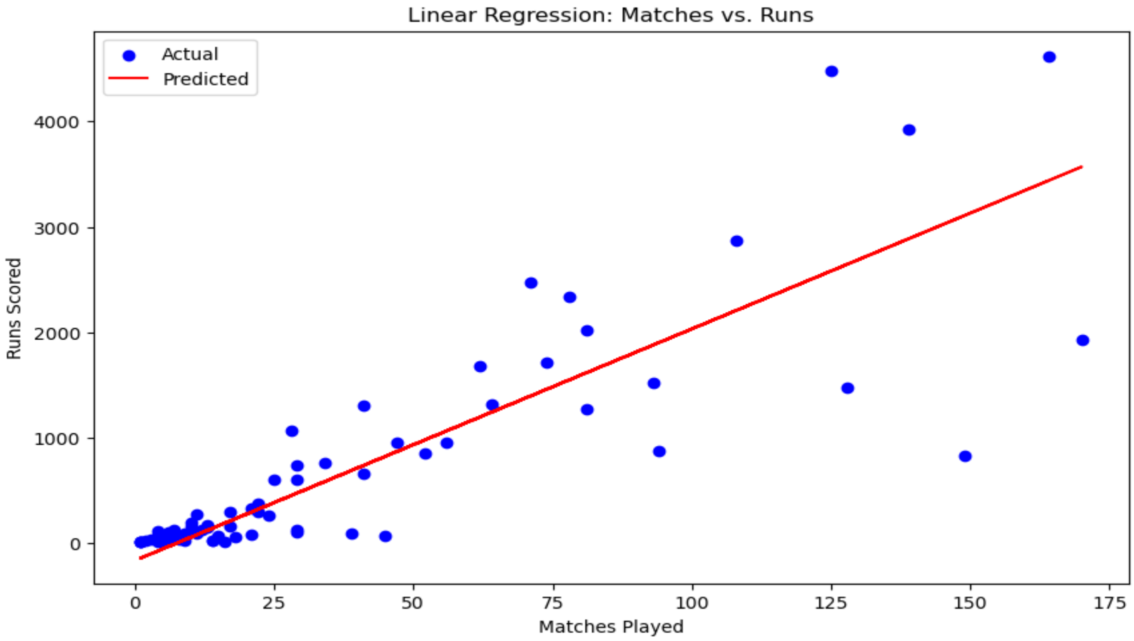
```
# Output results
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
```

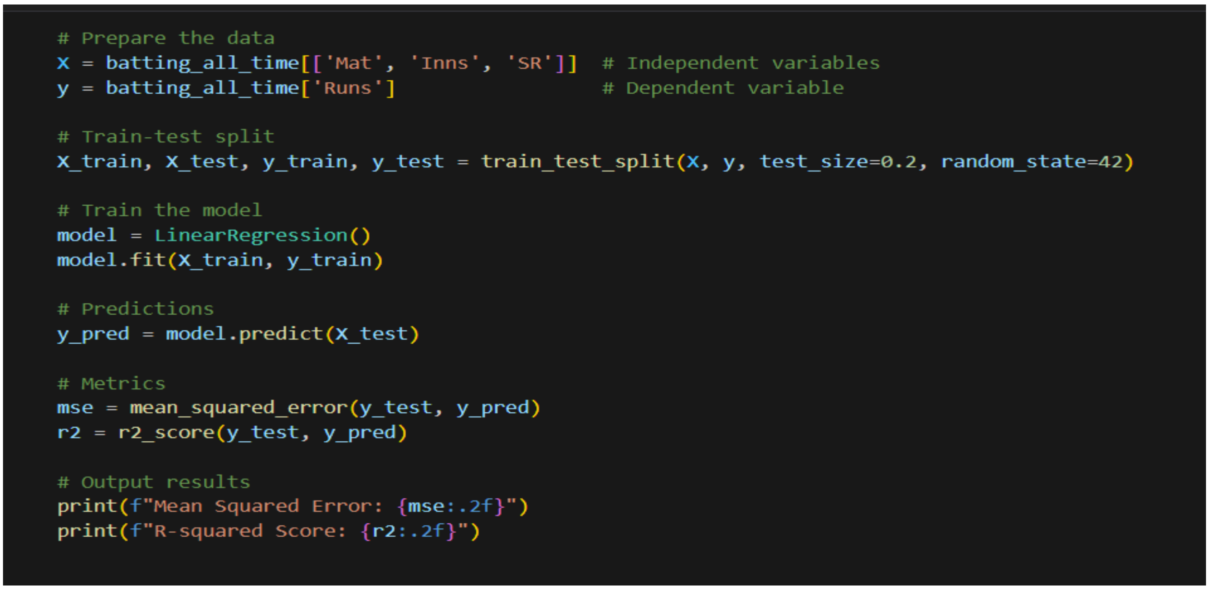
```
Mean Squared Error: 306034.73
R-squared Score: 0.71
```

Matches vs Runs:

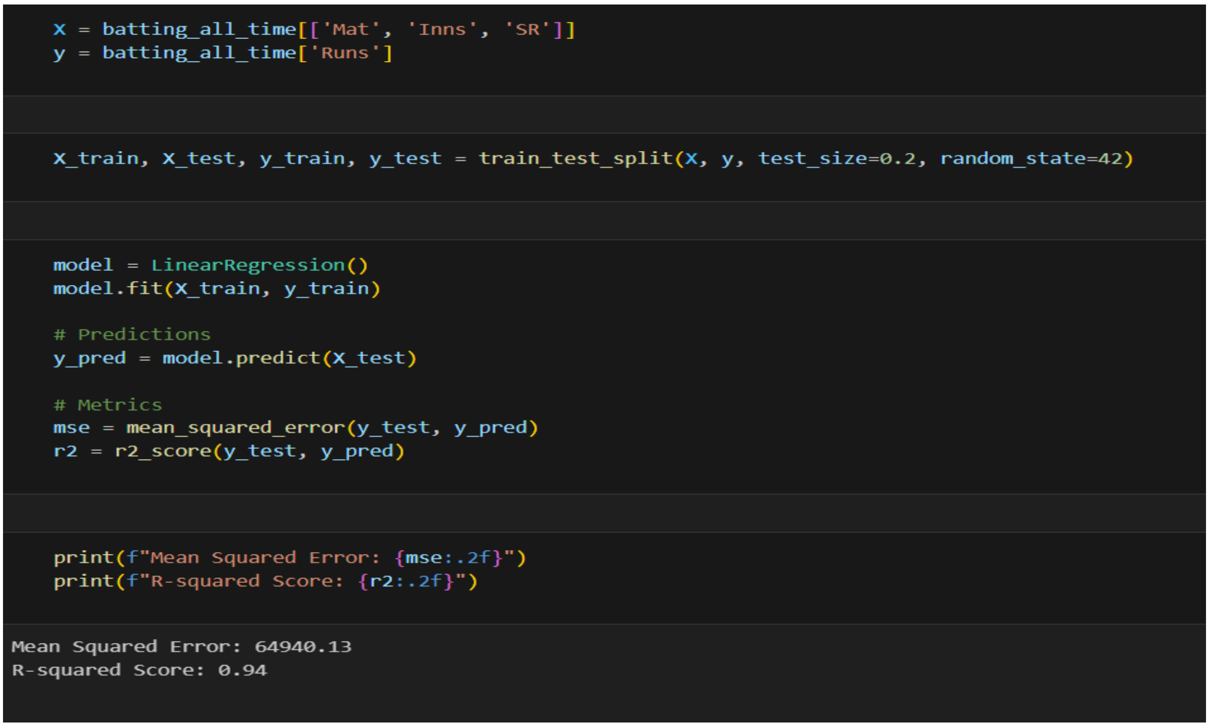


```python
# Plot the regression line
plt.figure(figsize=(10, 6))
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', label='Predicted')
plt.title('Linear Regression: Matches vs. Runs')
plt.xlabel('Matches Played')
plt.ylabel('Runs Scored')
plt.legend()
plt.show()
```





```python
# Prepare the data
X = batting_all_time[['Mat', 'Inns', 'SR']]  # Independent variables
y = batting_all_time['Runs']                  # Dependent variable

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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

# Predictions
y_pred = model.predict(X_test)

# Metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Output results
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
```



```
X = batting_all_time[['Mat', 'Inns', 'SR']]
y = batting_all_time['Runs']
```

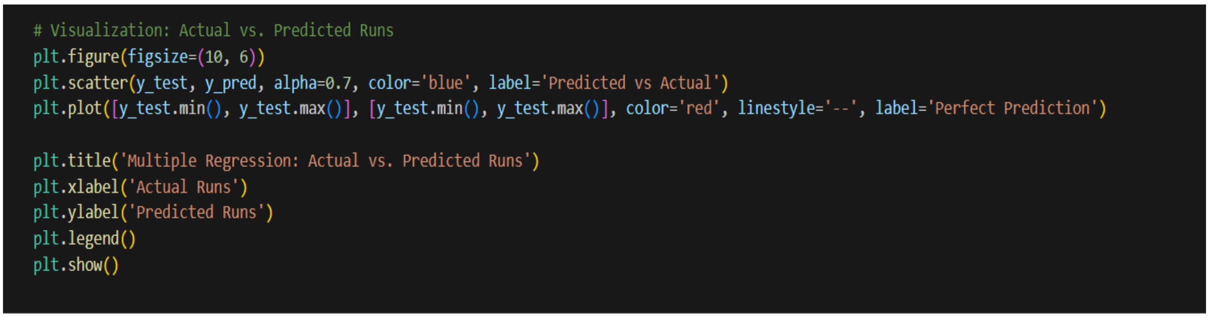
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

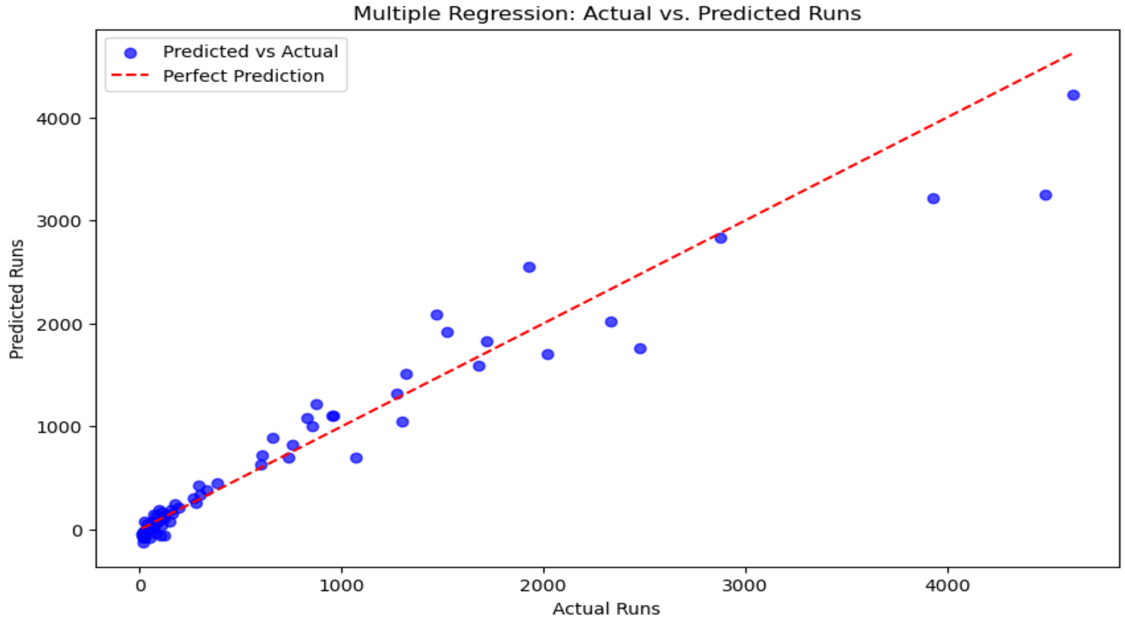
```
print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared Score: {r2:.2f}")
```

```
Mean Squared Error: 64940.13
R-squared Score: 0.94
```

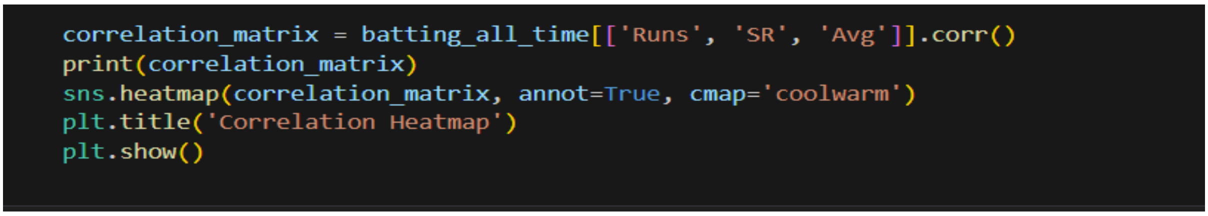
Actual vs Predicted Runs:



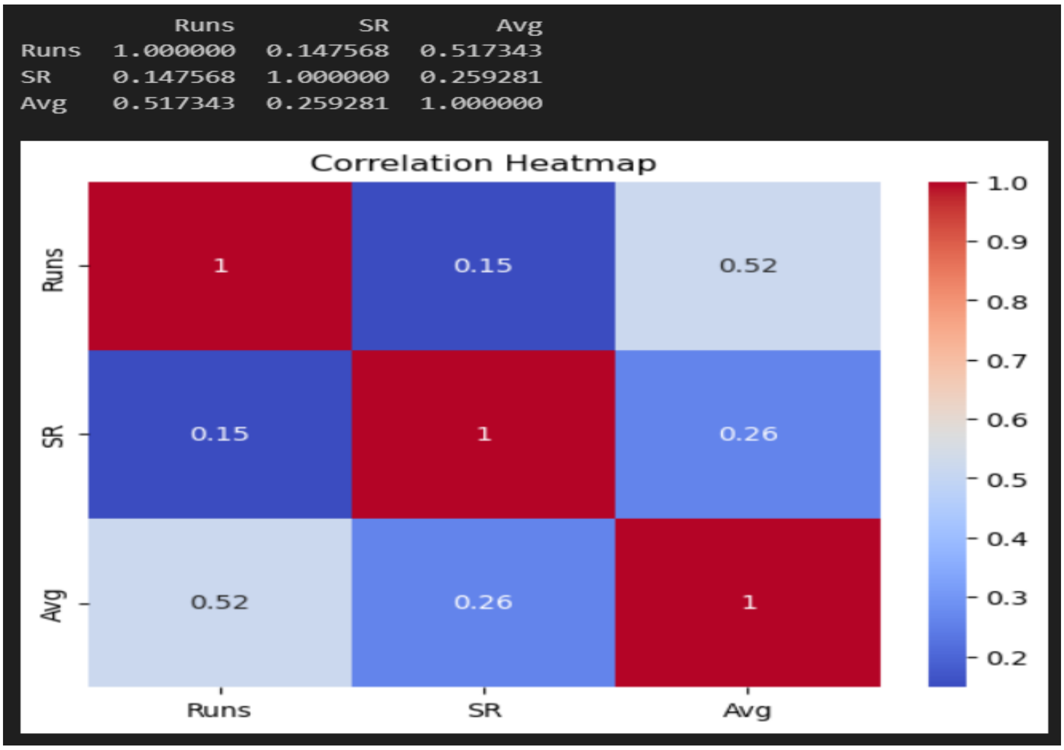
```
# Visualization: Actual vs. Predicted Runs
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actual')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', label='Perfect Prediction')

plt.title('Multiple Regression: Actual vs. Predicted Runs')
plt.xlabel('Actual Runs')
plt.ylabel('Predicted Runs')
plt.legend()
plt.show()
```



Corelation Heatmap:

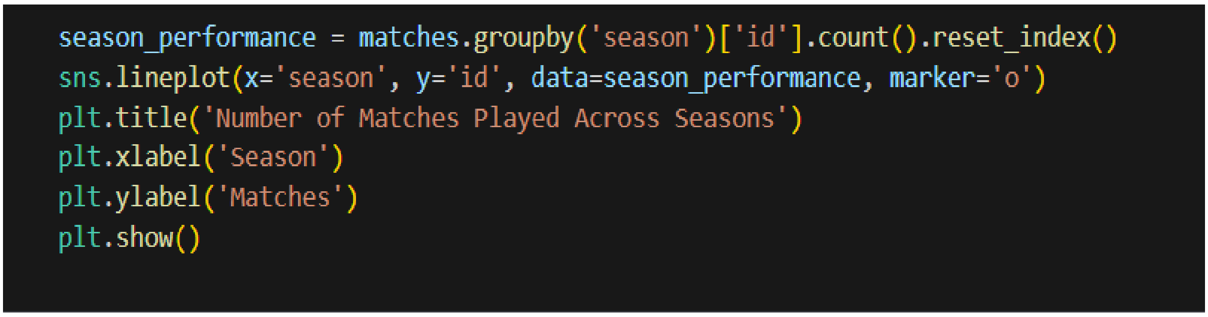


```
correlation_matrix = batting_all_time[['Runs', 'SR', 'Avg']].corr()
print(correlation_matrix)
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

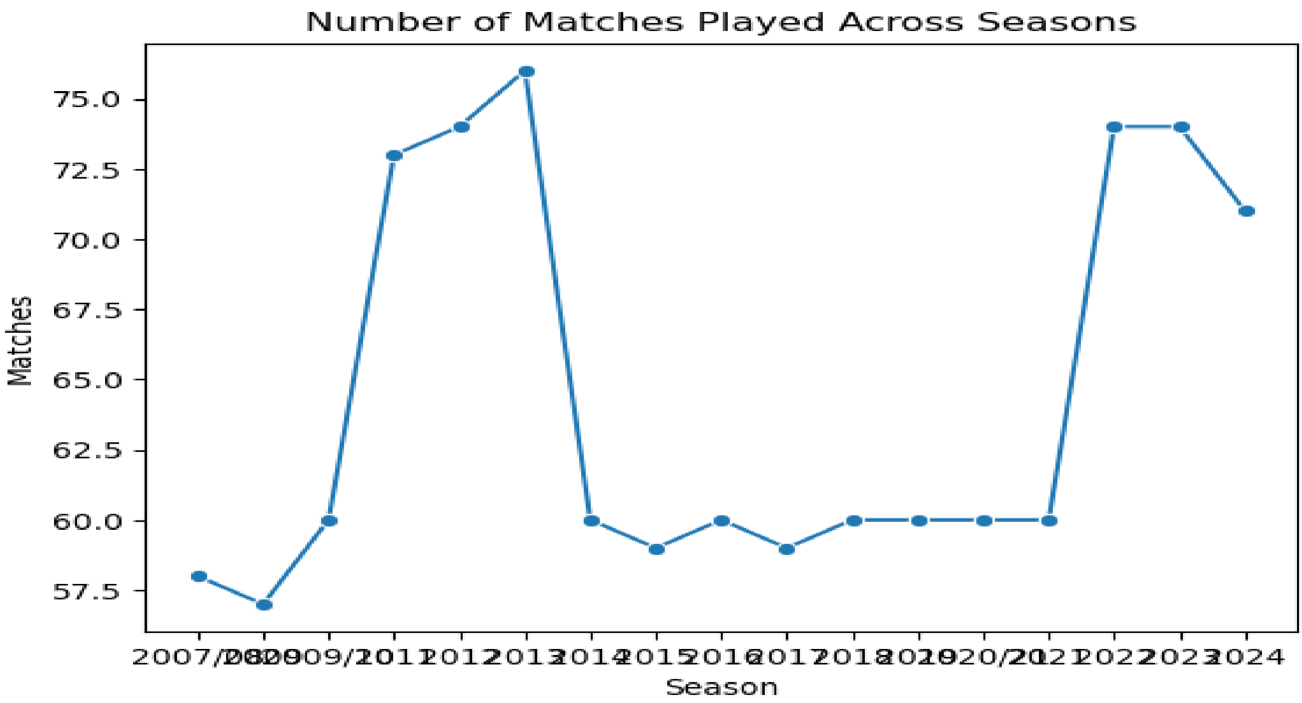


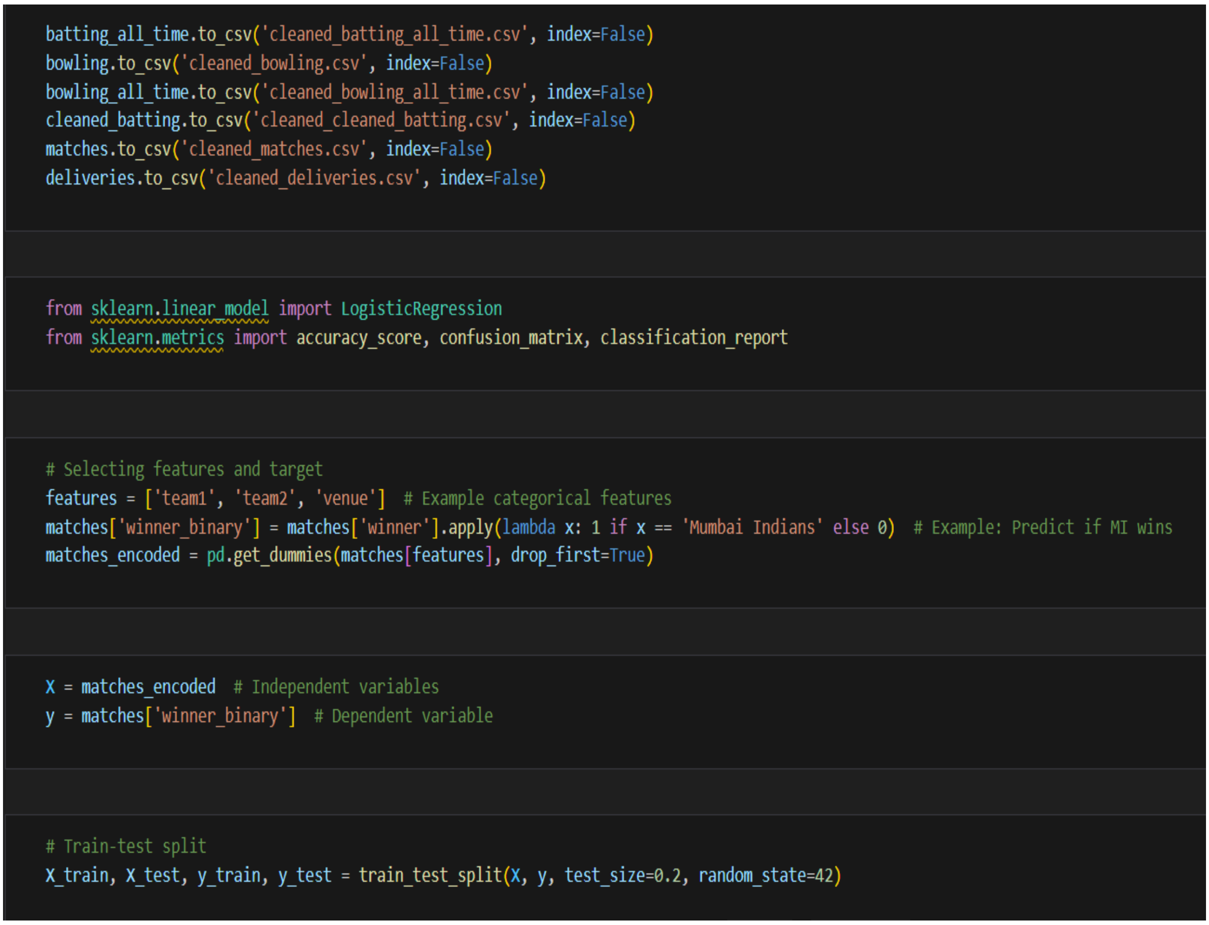
```
          Runs        SR       Avg
Runs  1.000000  0.147568  0.517343
SR    0.147568  1.000000  0.259281
Avg   0.517343  0.259281  1.000000
```

Number of matches played across seasons:



```
season_performance = matches.groupby('season')['id'].count().reset_index()
sns.lineplot(x='season', y='id', data=season_performance, marker='o')
plt.title('Number of Matches Played Across Seasons')
plt.xlabel('Season')
plt.ylabel('Matches')
plt.show()
```





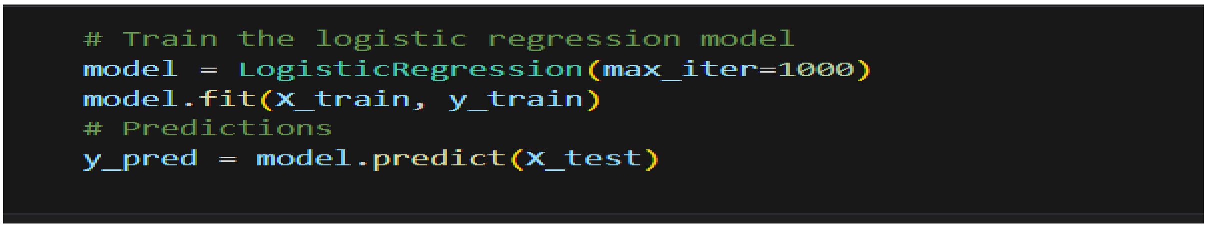
```
batting_all_time.to_csv('cleaned_batting_all_time.csv', index=False)
bowling.to_csv('cleaned_bowling.csv', index=False)
bowling_all_time.to_csv('cleaned_bowling_all_time.csv', index=False)
cleaned_batting.to_csv('cleaned_cleaned_batting.csv', index=False)
matches.to_csv('cleaned_matches.csv', index=False)
deliveries.to_csv('cleaned_deliveries.csv', index=False)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

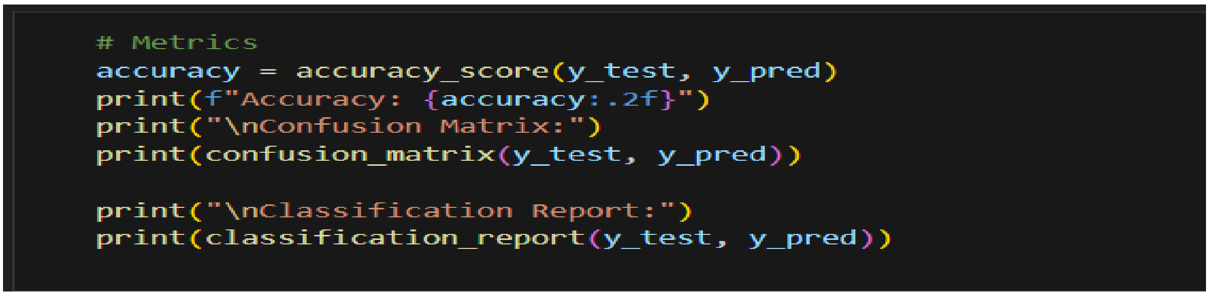
```
# Selecting features and target
features = ['team1', 'team2', 'venue']  # Example categorical features
matches['winner_binary'] = matches['winner'].apply(lambda x: 1 if x == 'Mumbai Indians' else 0)  # Example: Predict if MI wins
matches_encoded = pd.get_dummies(matches[features], drop_first=True)
```

```
X = matches_encoded  # Independent variables
y = matches['winner_binary']  # Dependent variable
```

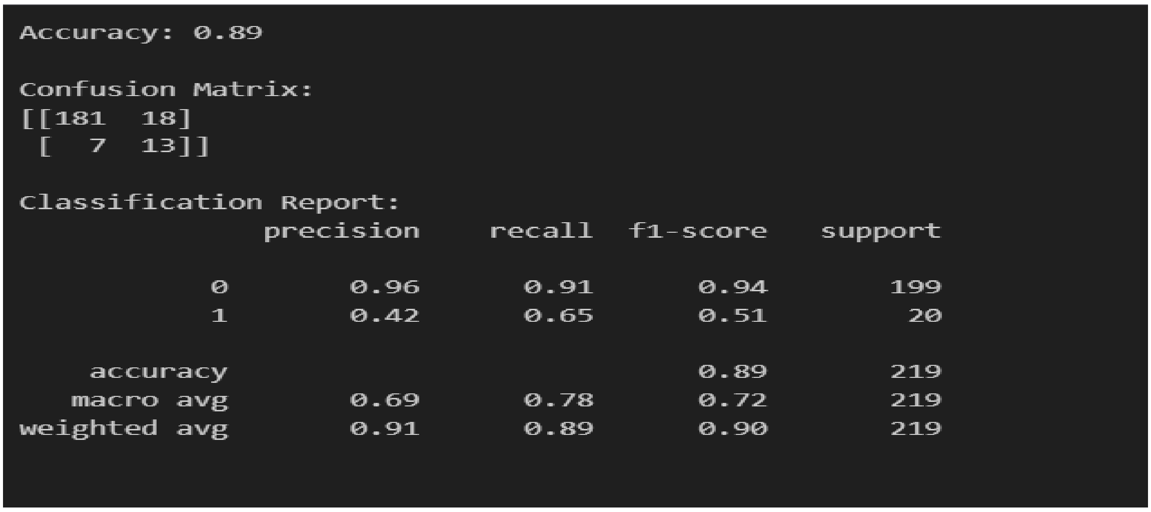
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



```python
# Train the logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
# Predictions
y_pred = model.predict(X_test)
```

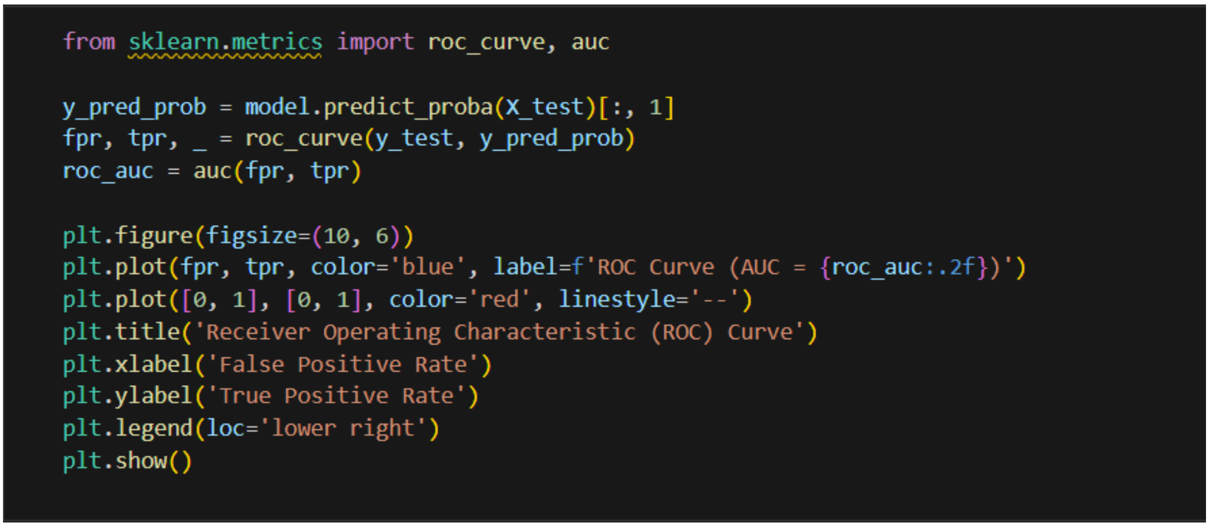


```python
# Metrics
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

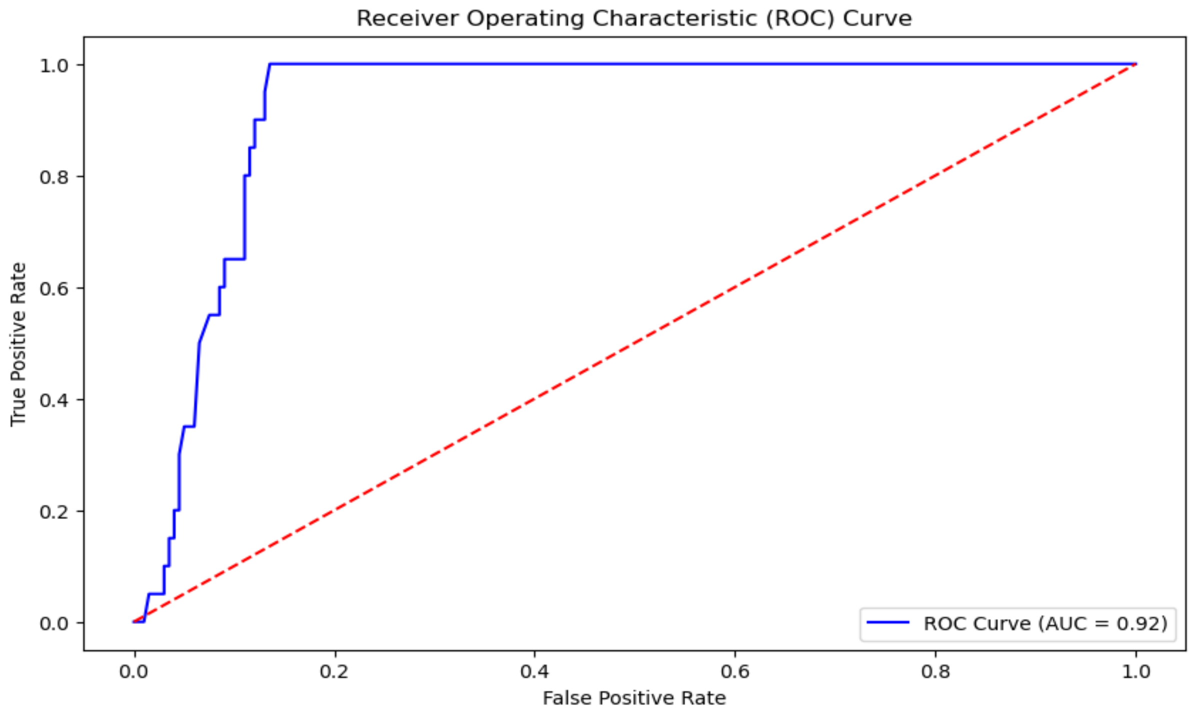


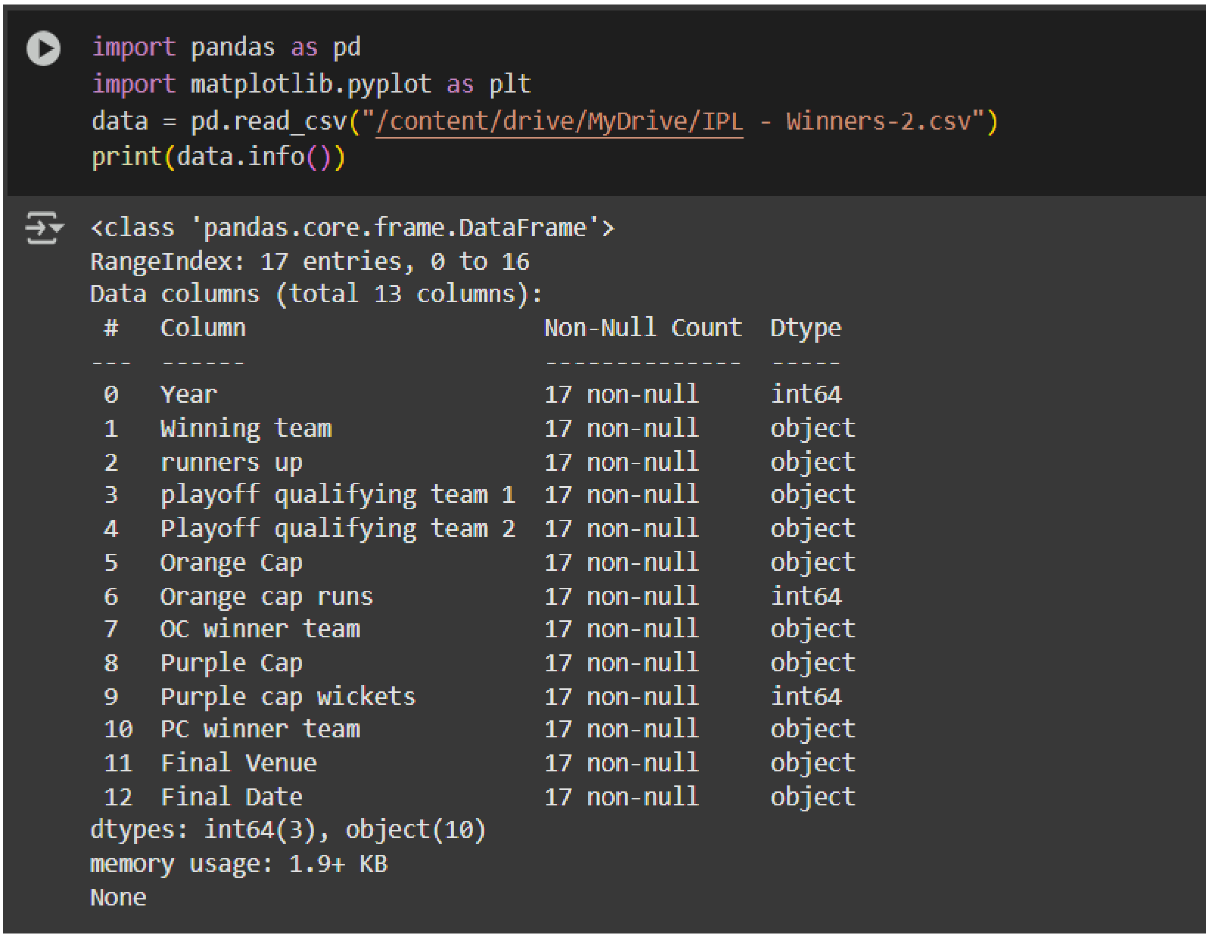
```
Accuracy: 0.89

Confusion Matrix:
[[181  18]
 [  7  13]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.91      0.94       199
           1       0.42      0.65      0.51        20

    accuracy                           0.89       219
   macro avg       0.69      0.78      0.72       219
weighted avg       0.91      0.89      0.90       219
```

Roc Curve:



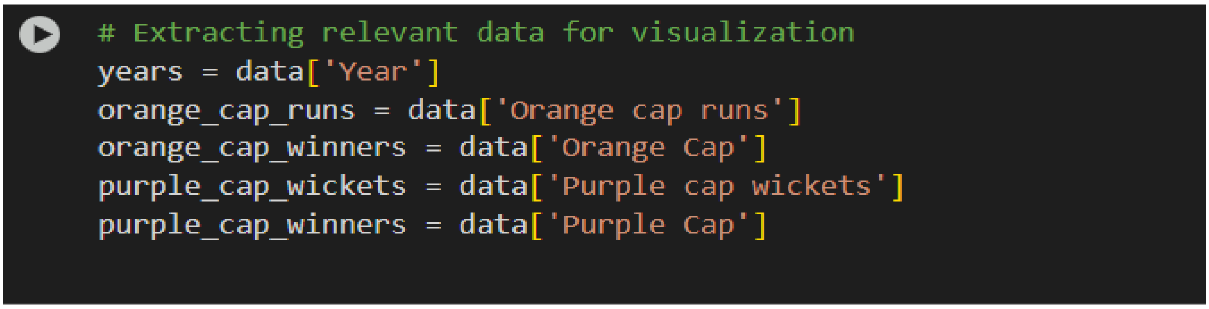
```python
from sklearn.metrics import roc_curve, auc

y_pred_prob = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='lower right')
plt.show()
```





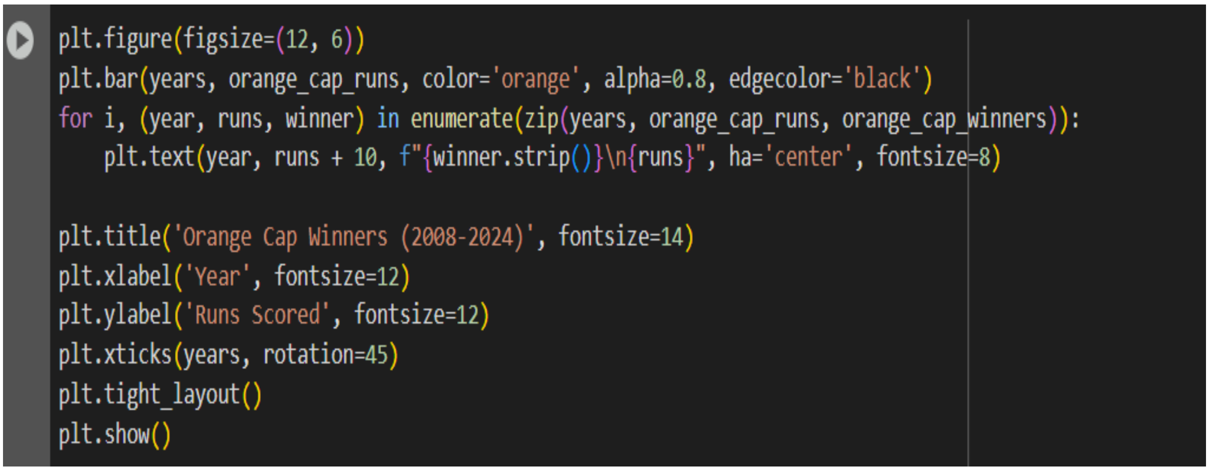
```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("/content/drive/MyDrive/IPL - Winners-2.csv")
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   Year                       17 non-null     int64 
 1   Winning team               17 non-null     object
 2   runners up                 17 non-null     object
 3   playoff qualifying team 1  17 non-null     object
 4   Playoff qualifying team 2  17 non-null     object
 5   Orange Cap                 17 non-null     object
 6   Orange cap runs            17 non-null     int64 
 7   OC winner team             17 non-null     object
 8   Purple Cap                 17 non-null     object
 9   Purple cap wickets         17 non-null     int64 
 10  PC winner team             17 non-null     object
 11  Final Venue                17 non-null     object
 12  Final Date                 17 non-null     object
dtypes: int64(3), object(10)
memory usage: 1.9+ KB
None
```

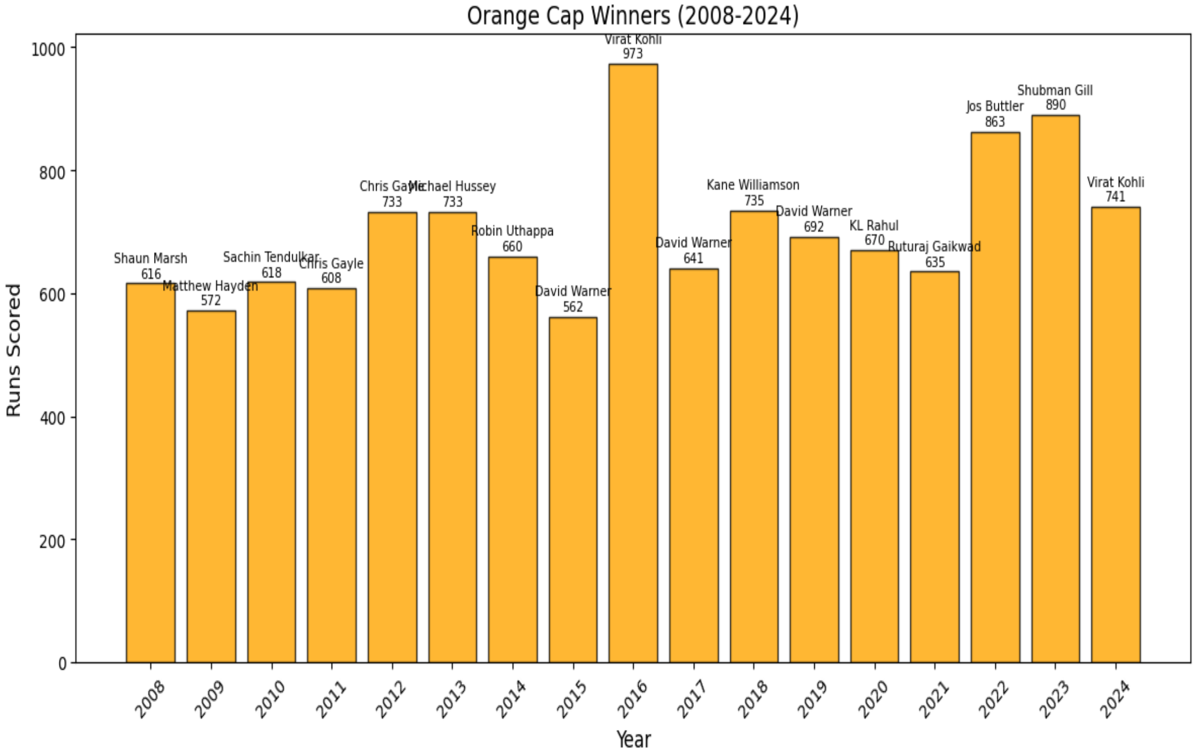


```
# Extracting relevant data for visualization
years = data['Year']
orange_cap_runs = data['Orange cap runs']
orange_cap_winners = data['Orange Cap']
purple_cap_wickets = data['Purple cap wickets']
purple_cap_winners = data['Purple Cap']
```

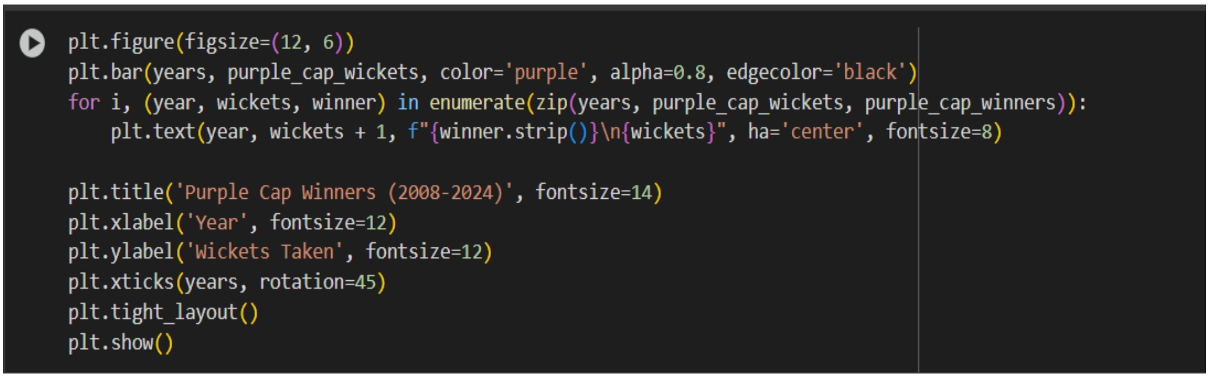
Orange cap Winners:



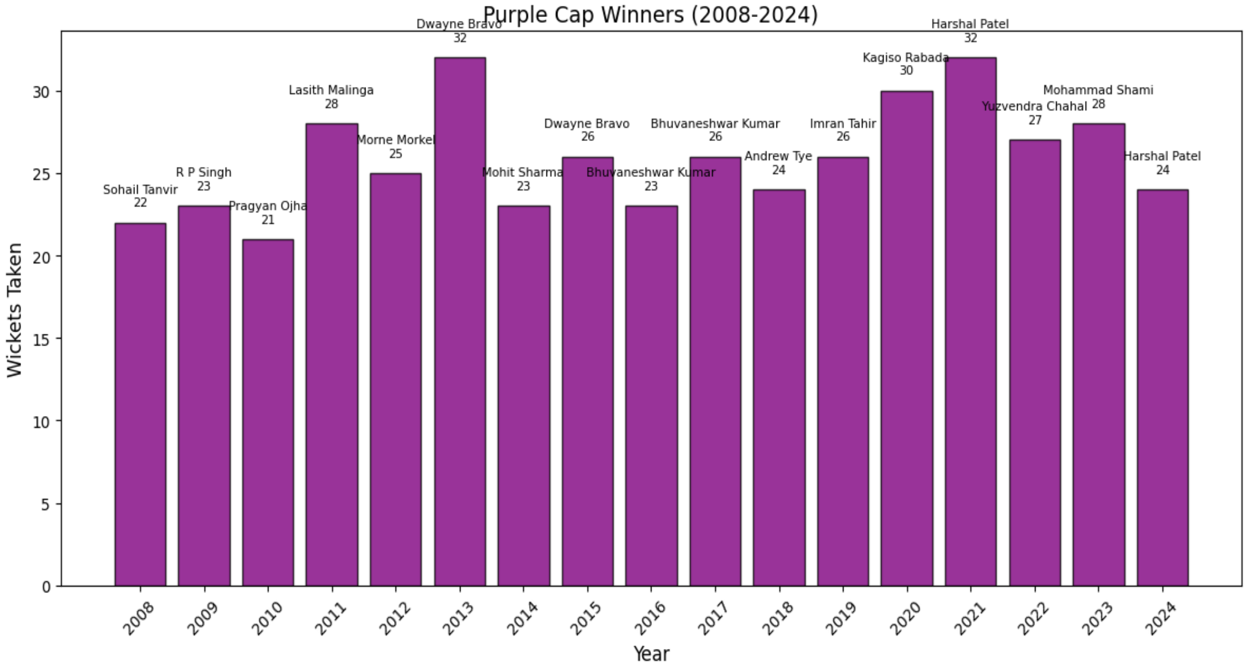
```
plt.figure(figsize=(12, 6))
plt.bar(years, orange_cap_runs, color='orange', alpha=0.8, edgecolor='black')
for i, (year, runs, winner) in enumerate(zip(years, orange_cap_runs, orange_cap_winners)):
    plt.text(year, runs + 10, f"{winner.strip()}\n{runs}", ha='center', fontsize=8)

plt.title('Orange Cap Winners (2008-2024)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Runs Scored', fontsize=12)
plt.xticks(years, rotation=45)
plt.tight_layout()
plt.show()
```



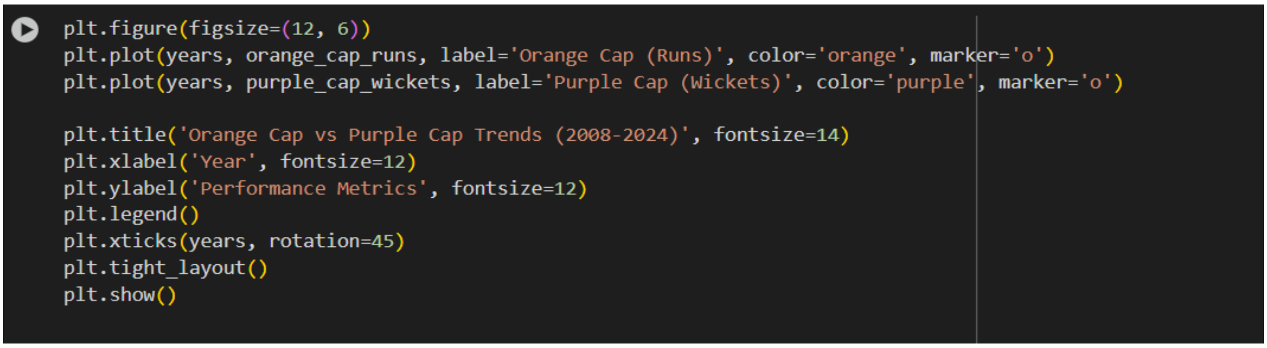
Purple Caps Winners:



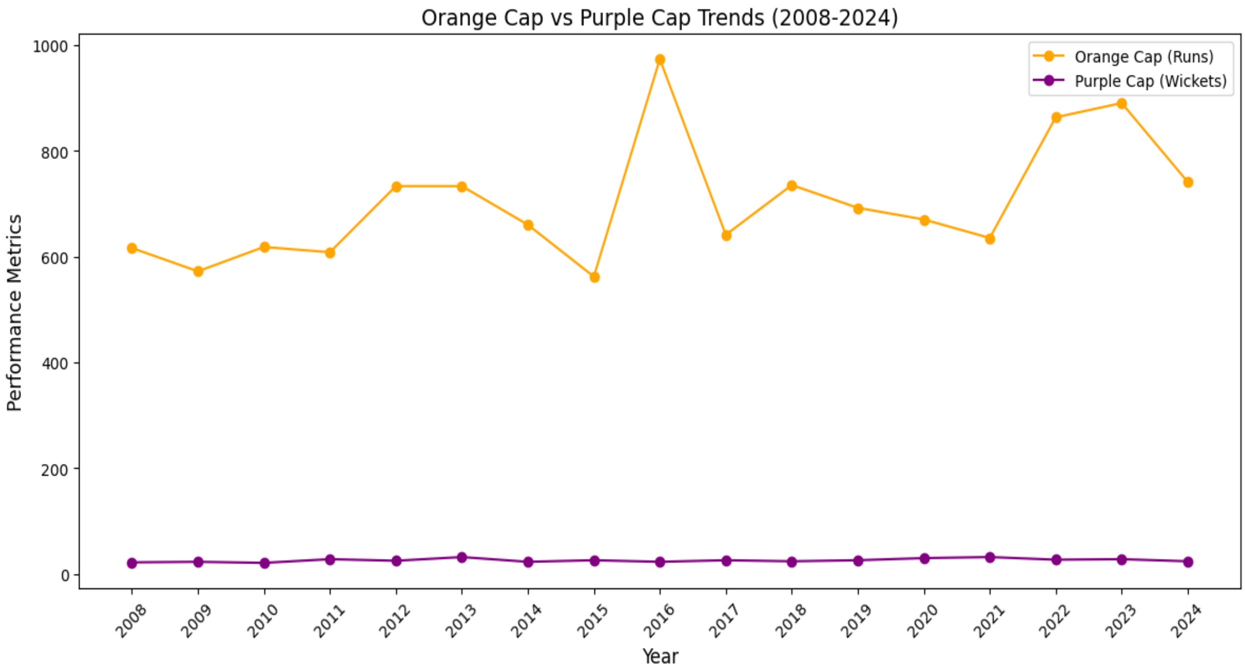
```
plt.figure(figsize=(12, 6))
plt.bar(years, purple_cap_wickets, color='purple', alpha=0.8, edgecolor='black')
for i, (year, wickets, winner) in enumerate(zip(years, purple_cap_wickets, purple_cap_winners)):
    plt.text(year, wickets + 1, f"{winner.strip()}\n{wickets}", ha='center', fontsize=8)

plt.title('Purple Cap Winners (2008-2024)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Wickets Taken', fontsize=12)
plt.xticks(years, rotation=45)
plt.tight_layout()
plt.show()
```



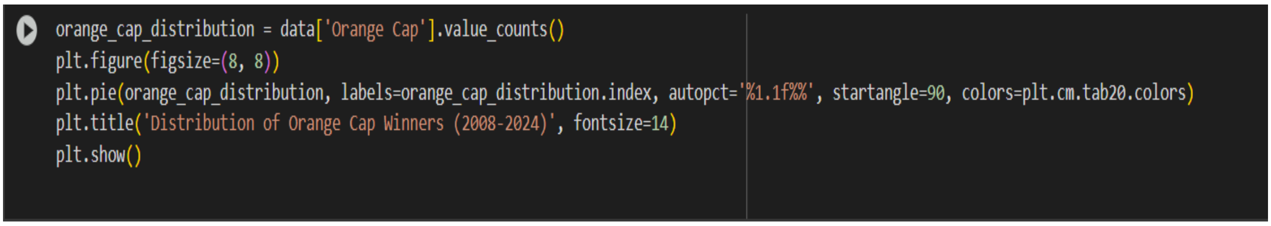
Orange cap vs Purple cap:



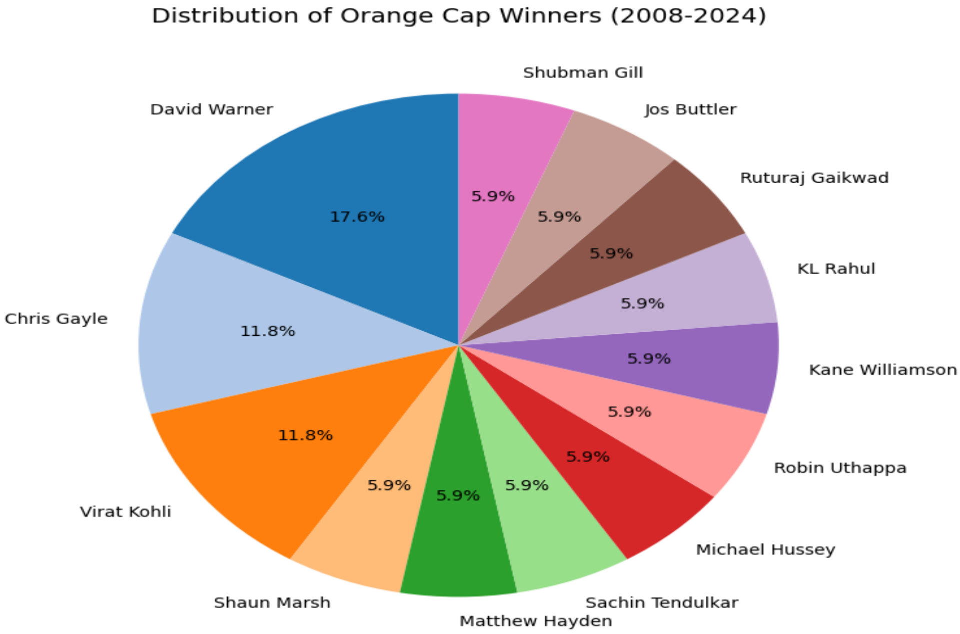
```
plt.figure(figsize=(12, 6))
plt.plot(years, orange_cap_runs, label='Orange Cap (Runs)', color='orange', marker='o')
plt.plot(years, purple_cap_wickets, label='Purple Cap (Wickets)', color='purple', marker='o')

plt.title('Orange Cap vs Purple Cap Trends (2008-2024)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Performance Metrics', fontsize=12)
plt.legend()
plt.xticks(years, rotation=45)
plt.tight_layout()
plt.show()
```



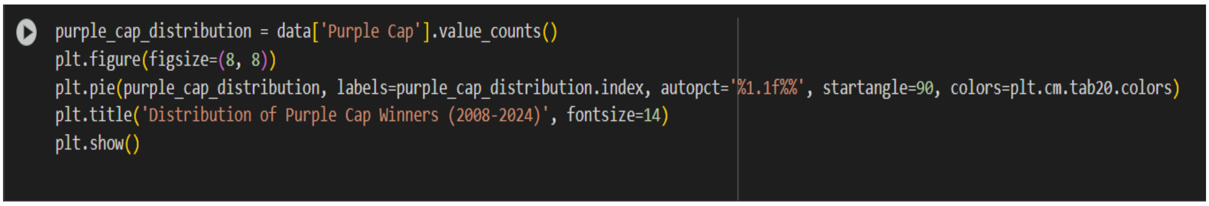
Distribution of Orange cap Winners:



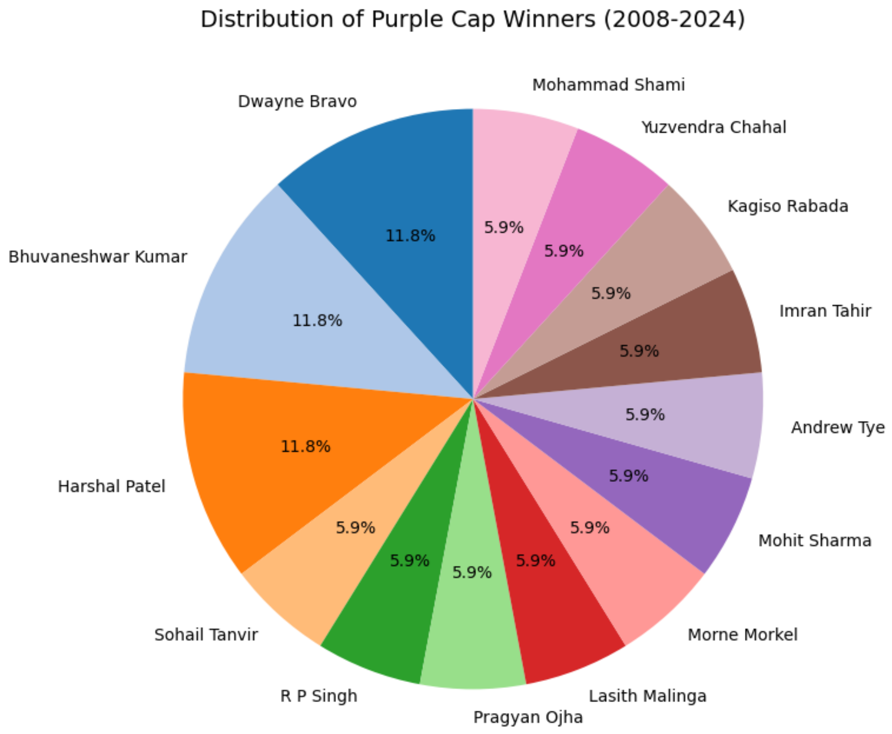
```
orange_cap_distribution = data['Orange Cap'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(orange_cap_distribution, labels=orange_cap_distribution.index, autopct='%1.1f%%', startangle=90, colors=plt.cm.tab20.colors)
plt.title('Distribution of Orange Cap Winners (2008-2024)', fontsize=14)
plt.show()
```



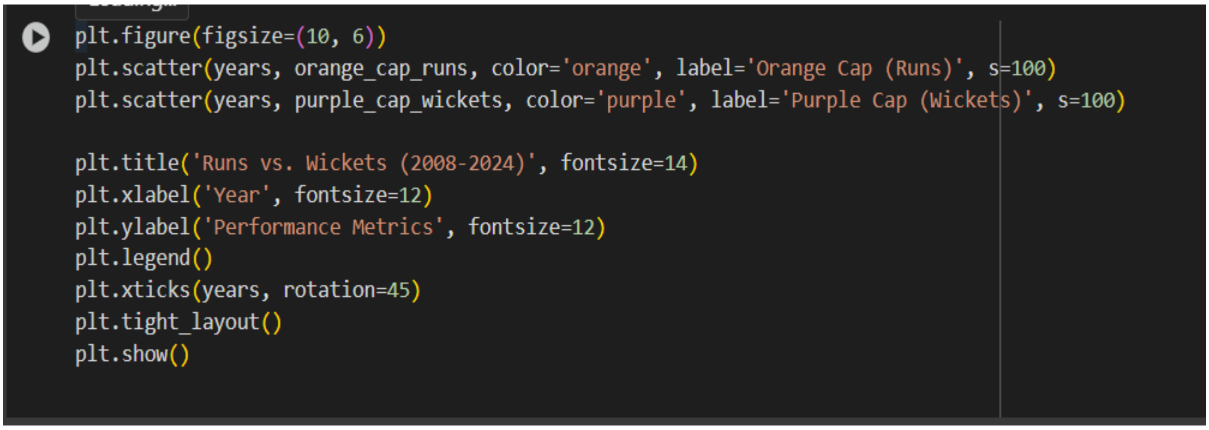
Distribution of Purple cap Winners:



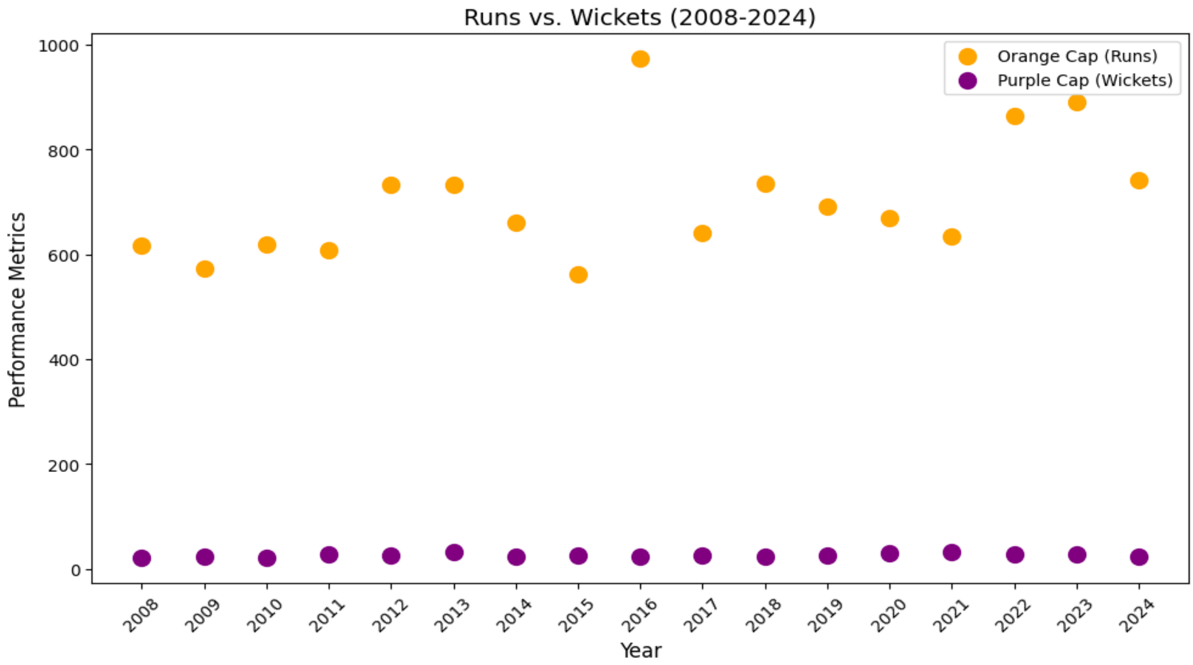
```
purple_cap_distribution = data['Purple Cap'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(purple_cap_distribution, labels=purple_cap_distribution.index, autopct='%1.1f%%', startangle=90, colors=plt.cm.tab20.colors)
plt.title('Distribution of Purple Cap Winners (2008-2024)', fontsize=14)
plt.show()
```



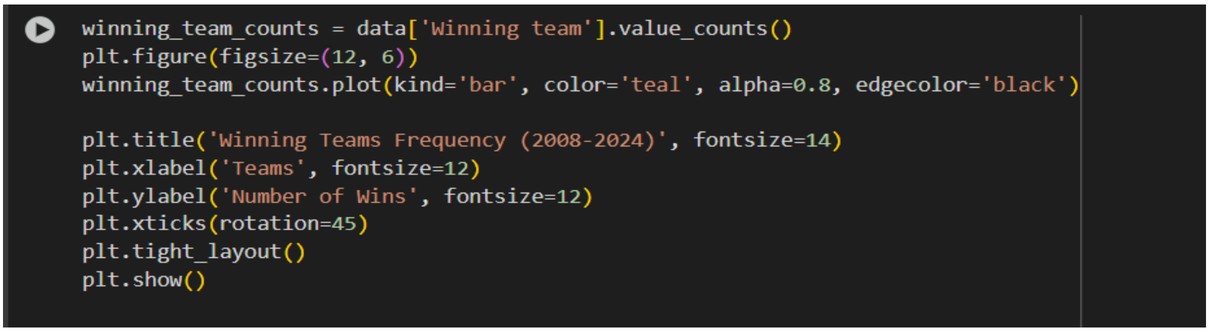
Runs vs Wickets:



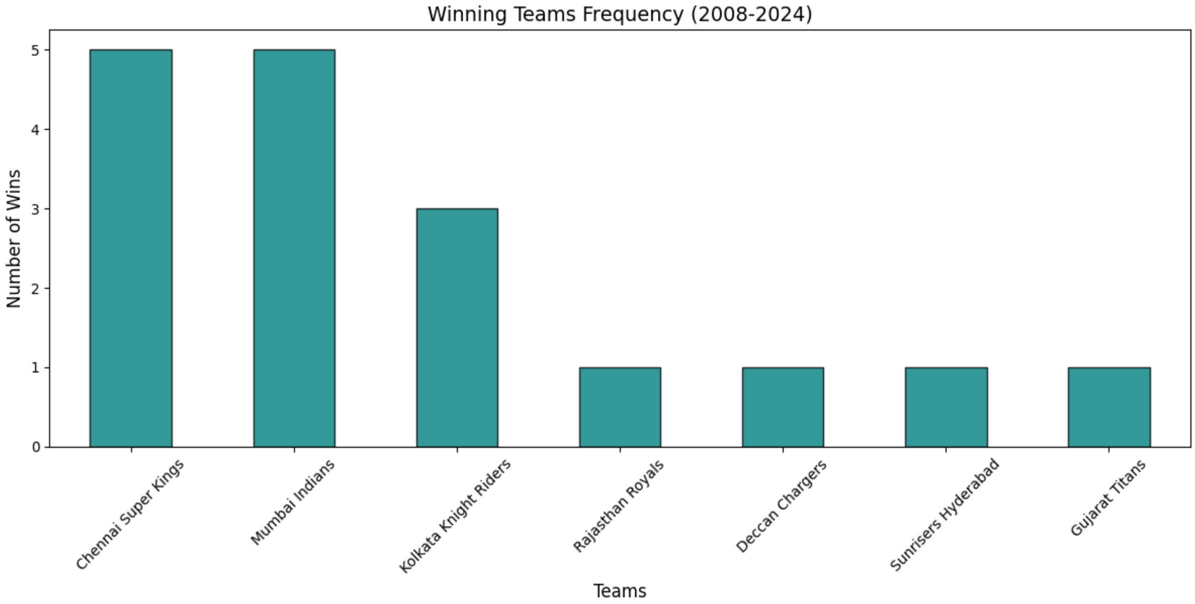
```
plt.figure(figsize=(10, 6))
plt.scatter(years, orange_cap_runs, color='orange', label='Orange Cap (Runs)', s=100)
plt.scatter(years, purple_cap_wickets, color='purple', label='Purple Cap (Wickets)', s=100)

plt.title('Runs vs. Wickets (2008-2024)', fontsize=14)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Performance Metrics', fontsize=12)
plt.legend()
plt.xticks(years, rotation=45)
plt.tight_layout()
plt.show()
```



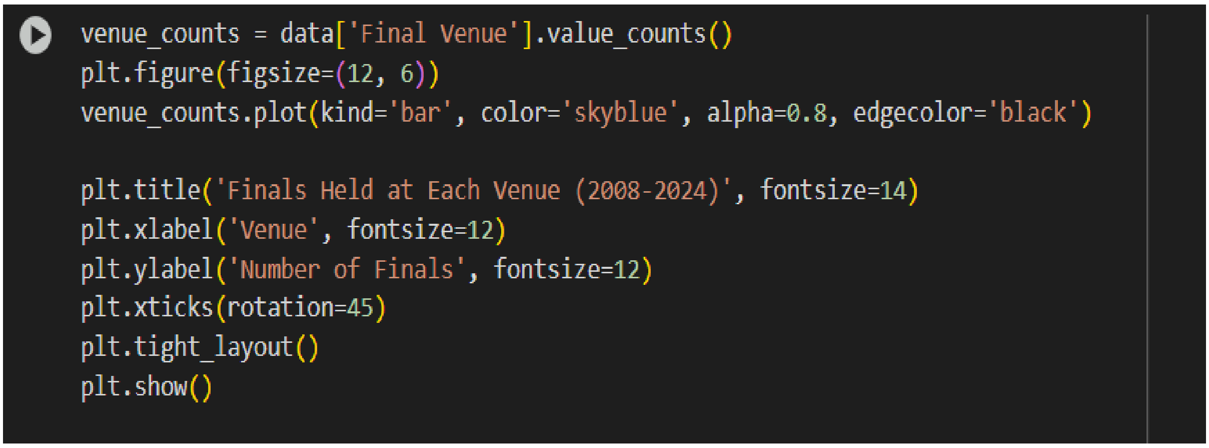
Frequency of Winning Teams:



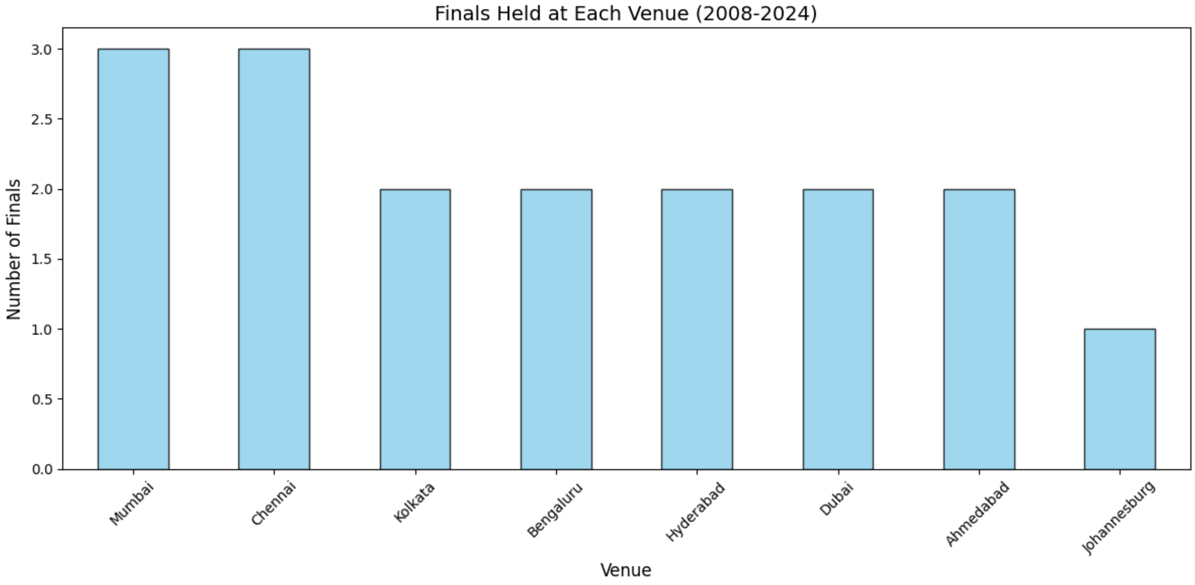
```
winning_team_counts = data['Winning team'].value_counts()
plt.figure(figsize=(12, 6))
winning_team_counts.plot(kind='bar', color='teal', alpha=0.8, edgecolor='black')

plt.title('Winning Teams Frequency (2008-2024)', fontsize=14)
plt.xlabel('Teams', fontsize=12)
plt.ylabel('Number of Wins', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



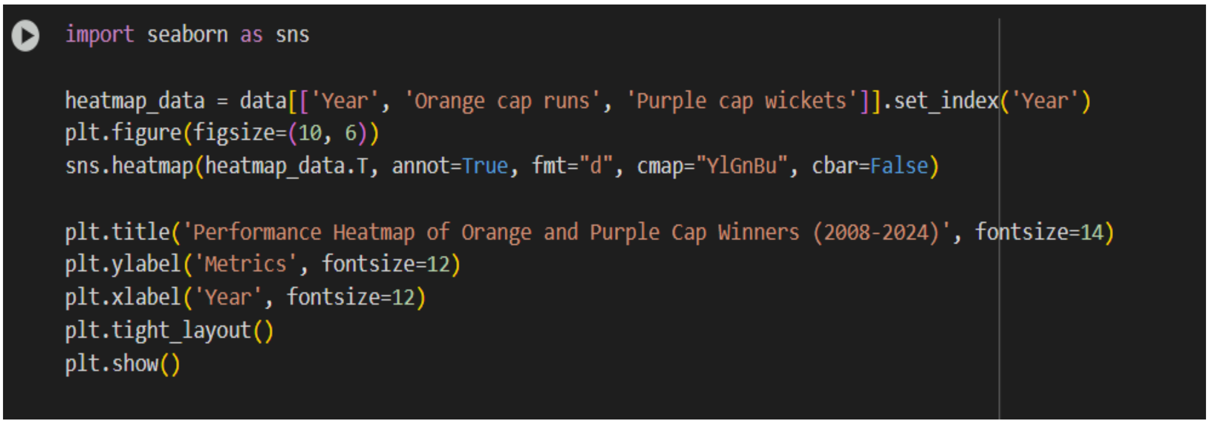
Finals Held Venues:



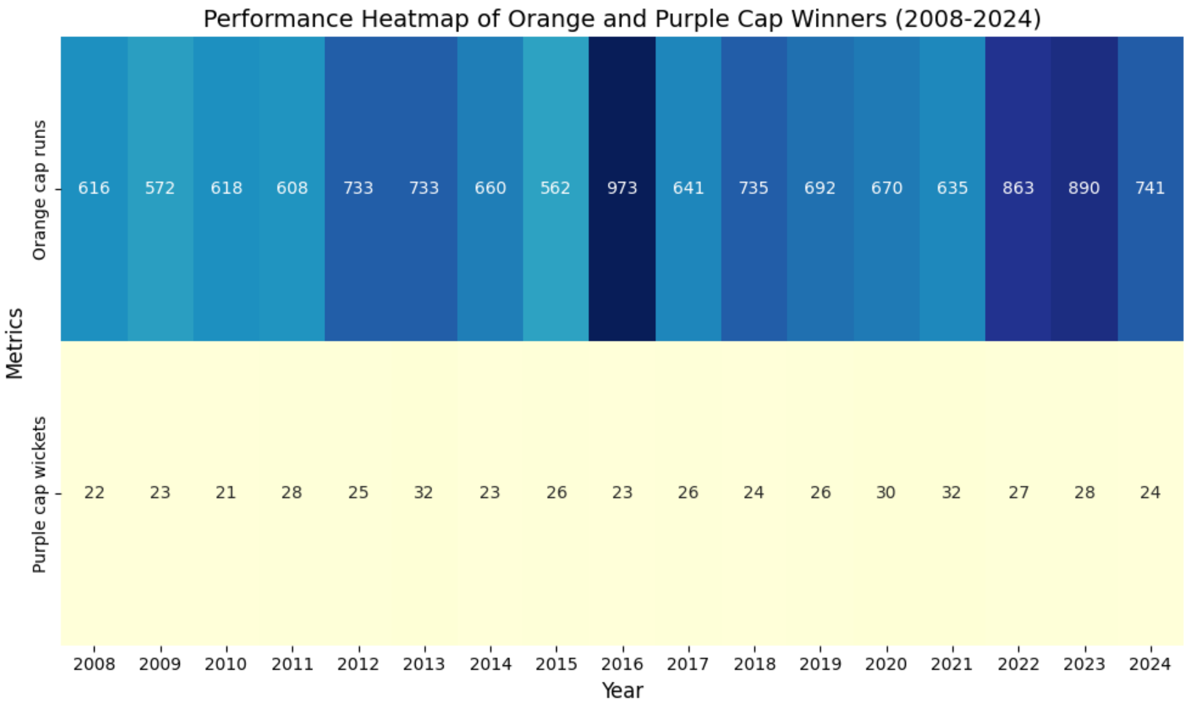
```
venue_counts = data['Final Venue'].value_counts()
plt.figure(figsize=(12, 6))
venue_counts.plot(kind='bar', color='skyblue', alpha=0.8, edgecolor='black')

plt.title('Finals Held at Each Venue (2008-2024)', fontsize=14)
plt.xlabel('Venue', fontsize=12)
plt.ylabel('Number of Finals', fontsize=12)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Heatmap of orange cap winners and purple cap Winners:

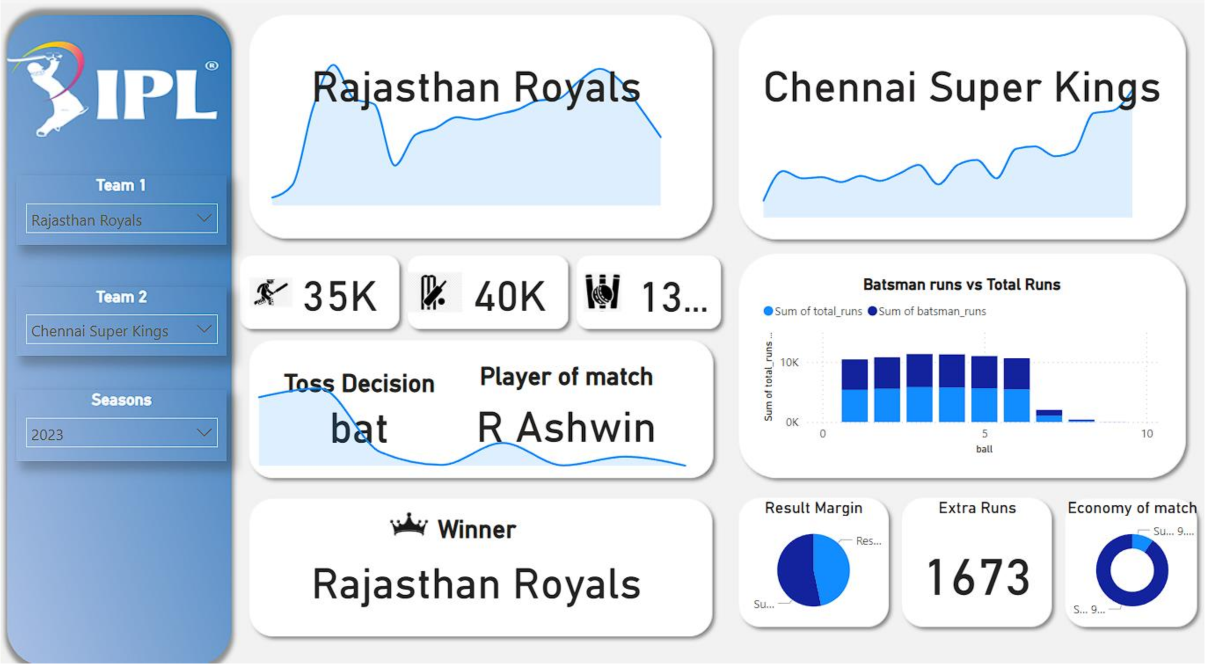


```
import seaborn as sns

heatmap_data = data[['Year', 'Orange cap runs', 'Purple cap wickets']].set_index('Year')
plt.figure(figsize=(10, 6))
sns.heatmap(heatmap_data.T, annot=True, fmt="d", cmap="YlGnBu", cbar=False)

plt.title('Performance Heatmap of Orange and Purple Cap Winners (2008-2024)', fontsize=14)
plt.ylabel('Metrics', fontsize=12)
plt.xlabel('Year', fontsize=12)
plt.tight_layout()
plt.show()
```



Performance Heatmap of Orange and Purple Cap Winners (2008-2024)

Power BI Report :



Team 1: Rajasthan Royals

Team 2: Chennai Super Kings

Seasons: 2023

1. Match Analysis:

- **Performance Trends:**
  The line charts for both teams (Rajasthan Royals and Chennai Super Kings) allow users to track performance trends, such as runs scored over time. Analysts can identify critical moments, scoring peaks, or slumps that impacted the match outcome.

- **Batsman vs. Team Contribution:**
  The bar chart comparing batsman runs to total team runs highlights individual contributions, enabling deeper insights into the batting strategy. For instance:

  - Did the team rely on one player for most of the runs?
  - Was the scoring evenly distributed among batsmen?

2. Player and Team Recognition:

- **Player of the Match Highlight:**
  Featuring R. Ashwin as the Player of the Match celebrates individual brilliance. It showcases his contribution as a game-changer, whether through batting, bowling, or fielding, making this information prominent for fans and commentators.

- **Winning Team Focus:**
  Highlighting the Rajasthan Royals as the winner puts the emphasis on the victorious team, making the dashboard celebratory and engaging for its fans. This can also be used for promotional or marketing purposes by the team or sponsors.

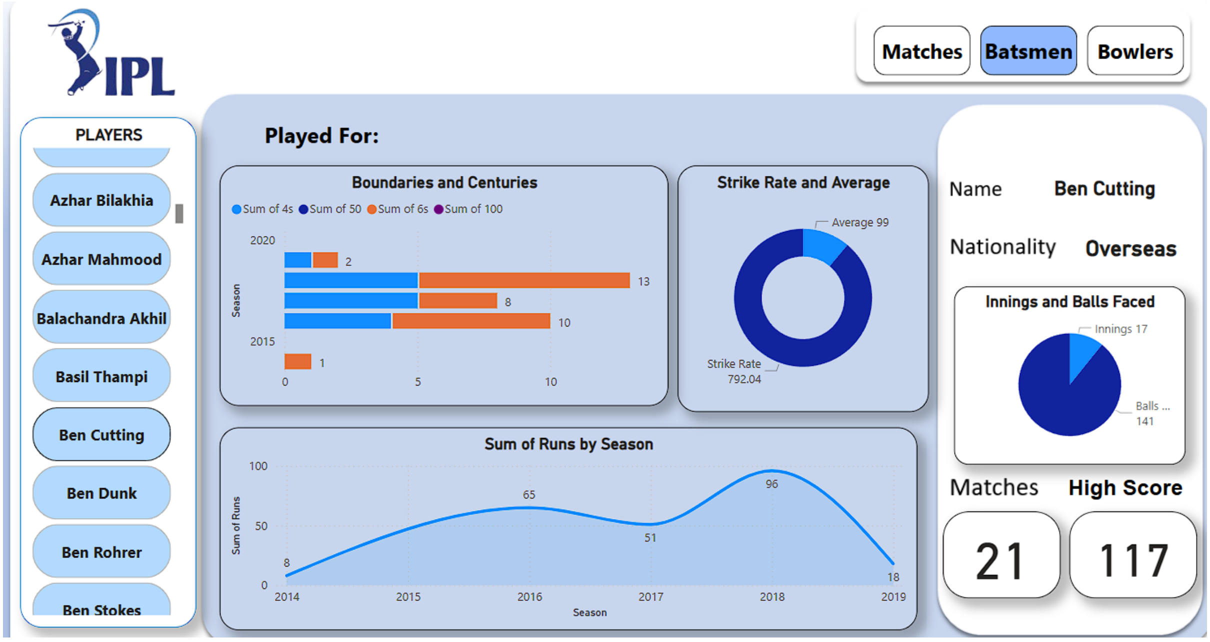
3. Coaching and Strategy Development:

- **Result Margin and Economy Rate:**
  By visualizing the result margin (e.g., runs or wickets) and the economy rate, this dashboard gives coaches and analysts a clearer picture of where the game was won or lost. For example:
  - Was the bowling economy a decisive factor?
  - How significant was the winning margin in reflecting the teams' dominance?
- **Insights for Future Matches:**
  Historical data (e.g., toss decisions, batting performance trends) helps teams refine their strategies for upcoming matches. For instance, if batting first consistently leads to victories, teams might adjust their approach during toss decisions.

4. Interactive Comparison Across Teams and Seasons:

- The dropdown filters (for teams and seasons) make this dashboard highly interactive, allowing users to:
  - Compare performances between different teams.
  - Analyze trends across multiple seasons.
  - Identify patterns in toss decisions, result margins, or key contributors.

  This transforms the dashboard into a dynamic tool, rather than a static report, making it suitable for a wide range of users, from casual fans to professional analysts.



Top Left Section: Player Selection

- **Player List:**
  A scrollable list of IPL players is provided. Users can select any player to view their detailed performance stats. Currently, the dashboard is displaying data for **Ben Cutting.**

Middle Section: Performance Overview

1. **Boundaries and Centuries (Bar Chart):**
   - This chart shows the player's performance in terms of:
     - **4s (boundaries):** Blue bars
     - **6s (sixes):** Orange bars
     - **Half-centuries (50s):** Purple bars
     - **Centuries (100s):** Pink bars (none for Ben Cutting)
   - Data is segmented by season, indicating consistency or improvement over the years.
   - Example: In **2020**, Ben Cutting hit 13 sixes and 2 fours, showcasing his power-hitting ability.
2. **Strike Rate and Average (Donut Chart):**
   - **Strike Rate:** An exceptionally high strike rate of 792.04, reflecting his effectiveness as a finisher or power-hitter.
   - **Average:** A batting average of 99, indicating strong performance when batting.

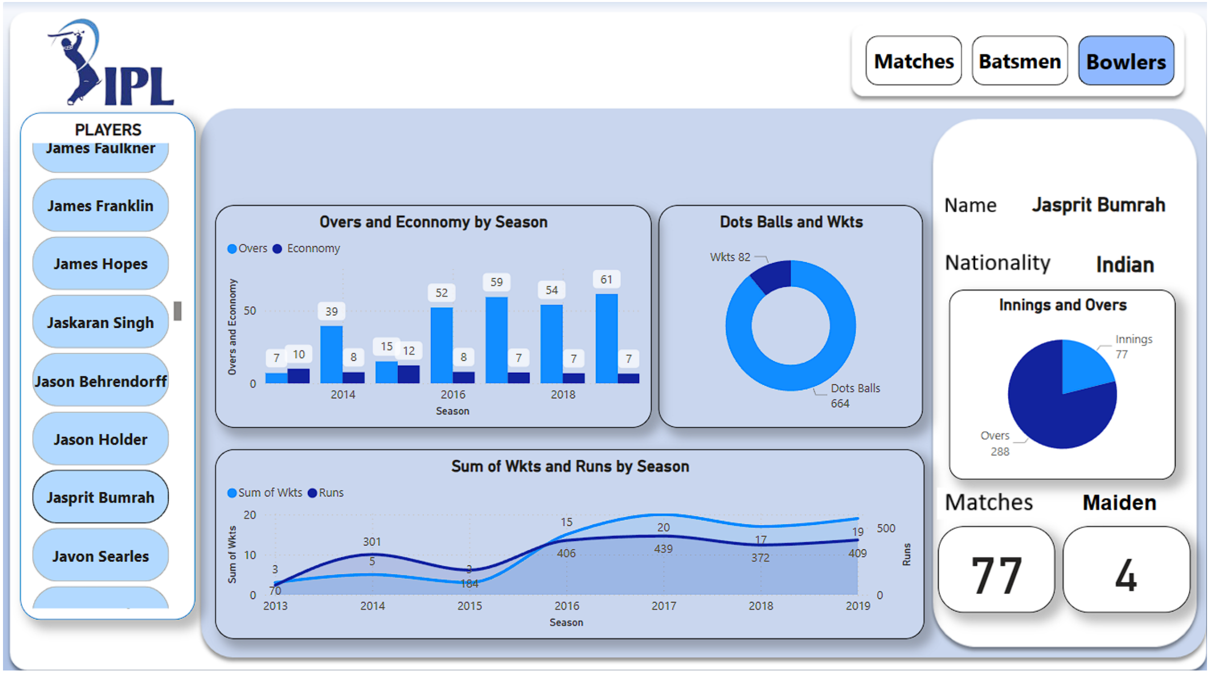
Sum of Runs by Season (Line Chart):

- This chart tracks the total runs scored by Ben Cutting across seasons.

- Example: His performance peaked in **2018** with 96 runs, then declined in 2019.

Right Section: Key Player Details

1. **Name and Nationality:**
   - Name: **Ben Cutting**
   - Nationality: **Overseas** player, likely from Australia.
2. **Innings and Balls Faced (Pie Chart):**
   - Visual representation of:
     - **Innings Played:** 17
     - **Balls Faced:** 141
   - This gives insight into his contributions and opportunities to bat.
3. **Matches and High Score:**
   - **Matches Played:** 21, showcasing his limited appearances in the IPL.
   - **High Score:** 117, indicating his capability to play impactful innings.



Name Jasprit Bumrah

Nationality Indian

Matches 77

Maiden 4

Top Left Section: Player Selection

- **Player List:**
  A scrollable list of bowlers is shown. Users can select any player to view their performance. Currently, the dashboard displays data for **Jasprit Bumrah**.

Middle Section: Performance Overview

1. **Overers and Economy by Season (Bar Chart):**
    - **Overs Bowled:** Blue bars represent the total overs bowled by Bumrah each season.
    - **Economy Rate:** Numbers at the top of each bar reflect the economy rate (runs conceded per over).
    - Key Insights:
        - Bumrah's economy has remained consistent around 7 runs per over in recent seasons, showcasing his control and discipline.
        - The number of overs bowled has steadily increased, reflecting his growing importance as a primary bowler.
2. **Dots Balls and Wickets (Donut Chart):**
    - **Dot Balls (664):** The proportion of balls that didn't concede runs, emphasizing his ability to build pressure.
    - **Wickets (82):** The number of wickets taken across his IPL career, highlighting his wicket-taking ability.
3. **Sum of Wickets and Runs by Season (Line Chart):**
    - **Wickets Taken:** Blue line tracks the number of wickets taken per season.
    - **Runs Conceded:** Grey line shows the total runs conceded per season.
    - Example: In **2018**, Bumrah took 17 wickets while conceding 372 runs.

Right Section: Key Player Details

1. **Name and Nationality:**
    - Name: **Jasprit Bumrah**
    - Nationality: **Indian** player, making him a key domestic talent.
2. **Innings and Overs (Pie Chart):**
    - **Innings Played:** 77 matches, showcasing his experience in the IPL.
    - **Overs Bowled:** 288, indicating his role as a frontline bowler.
3. **Matches and Maidens:**
    - **Matches Played:** 77, reflecting his consistent participation.
    - **Maidens Bowled:** 4, indicating his ability to deliver pressure-filled overs.

Features :

1. **Team Dashboard:**
   - Focuses on team performance, match results, and head-to-head comparisons (e.g., Rajasthan Royals vs. Chennai Super Kings in 2023).
   - Enables users to analyze team strategies, such as toss decisions and player contributions, while visualizing match outcomes.
2. **Batsman Dashboard:**
   - Highlights individual batting performances, including runs scored, boundaries hit, strike rates, and contributions across seasons.
   - Allows users to identify consistent performers or match-winners in the batting lineup.
3. **Bowler Dashboard:**
   - Provides an in-depth view of bowling performances, including overs bowled, economy rates, dot balls, and wickets taken.
   - Helps users assess the efficiency and impact of bowlers across seasons.

Who Can Benefit?

1. **Cricket Enthusiasts:**
   - Gain a detailed understanding of their favorite teams and players.
   - Deepen their knowledge of IPL statistics and trends.
2. **Team Management and Coaches:**
   - Use historical data for strategic planning and player evaluation.
3. **Fantasy League Players:**
   - Make informed decisions for their fantasy teams by analyzing player stats.
4. **Broadcasters and Commentators:**
   - Enhance storytelling during matches by referencing visually appealing and insightful data.

Conclusion :

The IPL dashboards collectively serve as a powerful analytical tool designed to simplify complex cricket data and present it in an interactive, visually appealing format. By focusing on teams, batsmen, and bowlers, the dashboards provide a **holistic view of IPL performances** across seasons, catering to diverse audiences, including fans, analysts, team management, fantasy league players, and broadcasters.