Import libraries,dataset.Perform some of the operations

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv("ODI.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1540 entries, 0 to 1539
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   player_name              1540 non-null   object
 1   role                     1540 non-null   object
 2   total_runs               1540 non-null   int64 
 3   strike_rate              1540 non-null   object
 4   total_balls_faced        1540 non-null   int64 
 5   total_wickets_taken      1540 non-null   int64 
 6   total_runs_conceded      1540 non-null   int64 
 7   total_overs_bowled       1540 non-null   int64 
 8   total_matches_played     1540 non-null   int64 
 9   matches_played_as_batter 1540 non-null   int64 
 10  matches_played_as_bowler 1540 non-null   int64 
 11  matches_won              1540 non-null   int64 
 12  matches_lost             1540 non-null   int64 
 13  player_of_match_awards   1540 non-null   int64 
 14  team                     1540 non-null   object
 15  average                  1540 non-null   object
 16  percentage               1540 non-null   object
dtypes: int64(11), object(6)
memory usage: 204.7+ KB
```

```
df.head(2)
```

	player_name	role	total_runs	strike_rate	total_balls_faced	total_wickets_taken	total_runs_conceded	total_overs_bowled
0	V Kohli	Batter	13784	9.170.381.212.161.530	15031	7	681	671
1	KC Sangakkara	Batter	11618	7.939.046.057.127.230	14634	0	0	0

Next steps: Generate code with df View recommended plots New interactive sheet

```
df.duplicated()
```

	0
0	False
1	False
2	False
3	False
4	False
...	...
1535	False
1536	False
1537	False
1538	False
1539	False

1540 rows × 1 columns

dtype: bool

df.describe(),df.isnull().sum(), df.duplicated()

```
num_col='matches_won'
```

```
mean_value = df[num_col].mean()
median_value = df[num_col].median()
mode_value = df[num_col].mode()[0]
std_dev = df[num_col].std()
variance = df[num_col].var()
data_range = df[num_col].max() - df[num_col].min()
```

```
print(f"Mean: {mean_value}")
print(f"Median: {median_value}")
print(f"Mode: {mode_value}")
print(f"Standard Deviation: {std_dev}")
print(f"Variance: {variance}")
print(f"Range: {data_range}")
```

```
Mean: 148.4525974025974
Median: 137.5
Mode: 27
Standard Deviation: 118.57929723750318
Variance: 14061.04973334013
Range: 381
```

```
Q1 = df[num_col].quantile(0.25)
Q3 = df[num_col].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df[num_col] < lower_bound) | (df[num_col] > upper_bound)]
print("\nOutliers detected:")
print(outliers[num_col ])
```

```

Outliers detected:
Series([], Name: matches_won, dtype: int64)
```

```
df['strike_rate'] = pd.to_numeric(df['strike_rate'],errors='coerce')
df['average'] = pd.to_numeric(df['average'], errors='coerce')
df['percentage'] = pd.to_numeric(df['percentage'], errors='coerce')
```

```
df[['total_runs', 'strike_rate']].corr()
```

	total_runs	strike_rate
total_runs	1.000000	0.123998
strike_rate	0.123998	1.000000

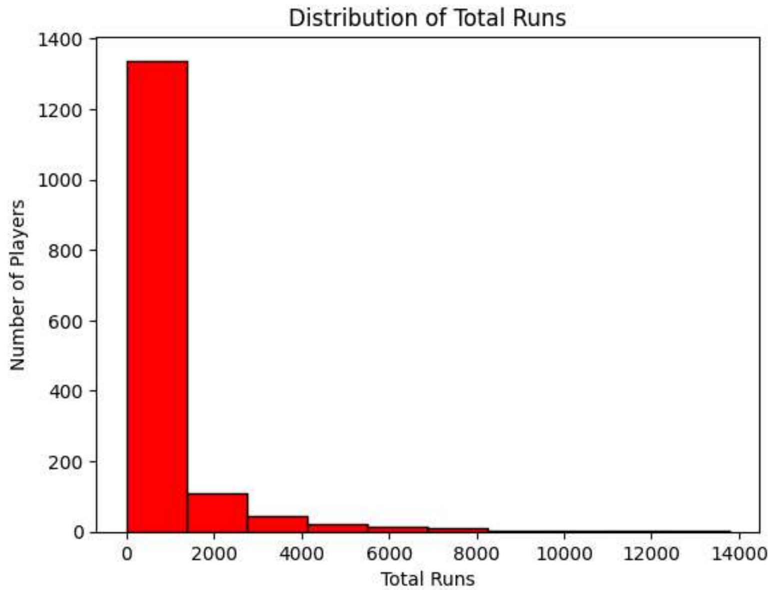
```
df[['total_runs', 'strike_rate']].cov()
```

	total_runs	strike_rate
total_runs	1.739826e+06	1503.375650
strike_rate	1.503376e+03	8489.196576

Data Visualization

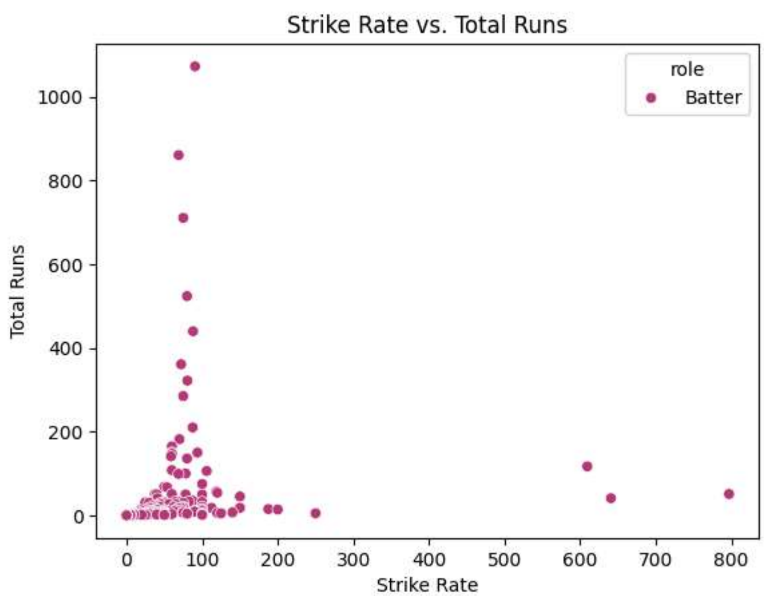
1.Histogram

```
plt.figure()
plt.hist(df['total_runs'], bins=10, color='red', edgecolor='black')
plt.xlabel("Total Runs")
plt.ylabel("Number of Players")
plt.title("Distribution of Total Runs")
plt.show()
```



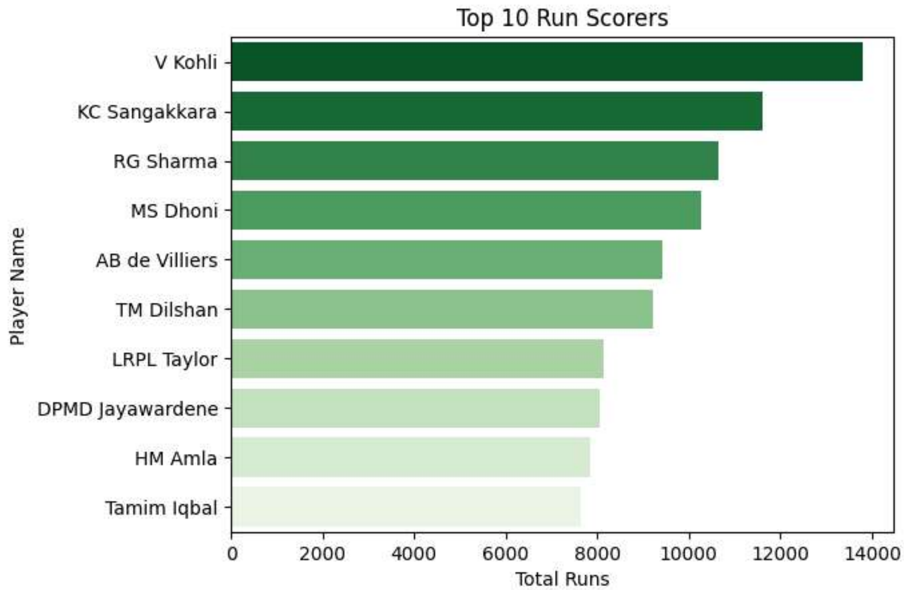
2.Scatter plot

```
plt.figure()
sns.scatterplot(x='strike_rate', y='total_runs', hue='role', data=df, palette="magma")
plt.xlabel("Strike Rate")
plt.ylabel("Total Runs")
plt.title("Strike Rate vs. Total Runs")
plt.show()
```



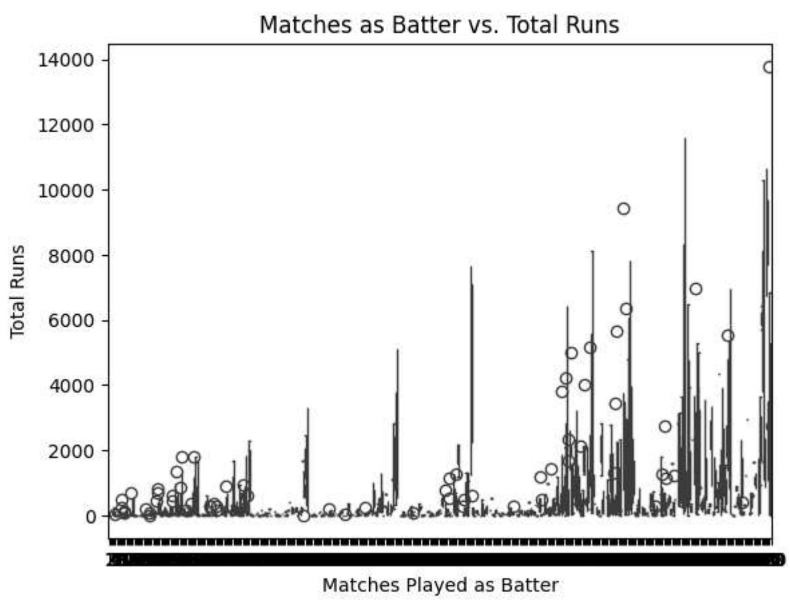
3.Bar chart

```
top_scorers = df.nlargest(10, 'total_runs')
plt.figure()
sns.barplot(x='total_runs', y='player_name', data=top_scorers, palette="Greens_r")
plt.xlabel("Total Runs")
plt.ylabel("Player Name")
plt.title("Top 10 Run Scorers")
plt.show()
```



4.Boxplot

```
plt.figure()
sns.boxplot(x='matches_played_as_batter', y='total_runs', data=df)
plt.xlabel("Matches Played as Batter")
plt.ylabel("Total Runs")
plt.title("Matches as Batter vs. Total Runs")
plt.show()
```



Algorithms to perform-Unsupervised

1.PCA

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA


numeric_df = df.select_dtypes(include=['int64'])


numeric_df.isnull()
```

	total_runs	total_balls_faced	total_wickets_taken	total_runs_conceded	total_overs_bowled	total_matches_played	matches_played_
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
1535	False	False	False	False	False	False	
1536	False	False	False	False	False	False	
1537	False	False	False	False	False	False	
1538	False	False	False	False	False	False	
1539	False	False	False	False	False	False	

1540 rows × 11 columns

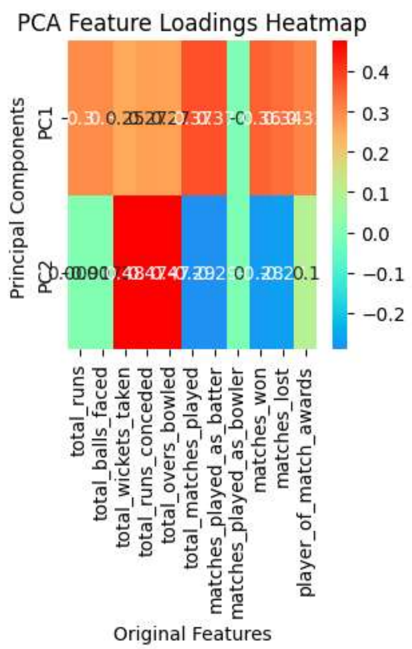
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(numeric_df)


pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_data)
pca_df = pd.DataFrame(pca_components, columns=['PC1', 'PC2'])
pca_df
```

	PC1	PC2
0	11.893420	0.094485
1	8.922220	-0.914877
2	8.713684	-0.564040
3	7.960842	-1.388427
4	7.251446	-0.311747
...	...	...
1535	-2.282195	0.692034
1536	-2.571364	0.806800
1537	-2.815851	0.997893
1538	-2.787238	0.975547
1539	-2.840626	1.017152

1540 rows × 2 columns

Next steps: Generate code with pca_df View recommended plots New interactive sheet

```
loading= pd.DataFrame(pca.components_, columns=numeric_df.columns, index=['PC1', 'PC2'])
plt.figure(figsize=(3,3))
sns.heatmap(loading, annot=True, cmap='rainbow', center=0)
plt.title('PCA Feature Loadings Heatmap')
plt.xlabel('Original Features')
plt.ylabel('Principal Components')
plt.show()
```



2.K-Means Cluster

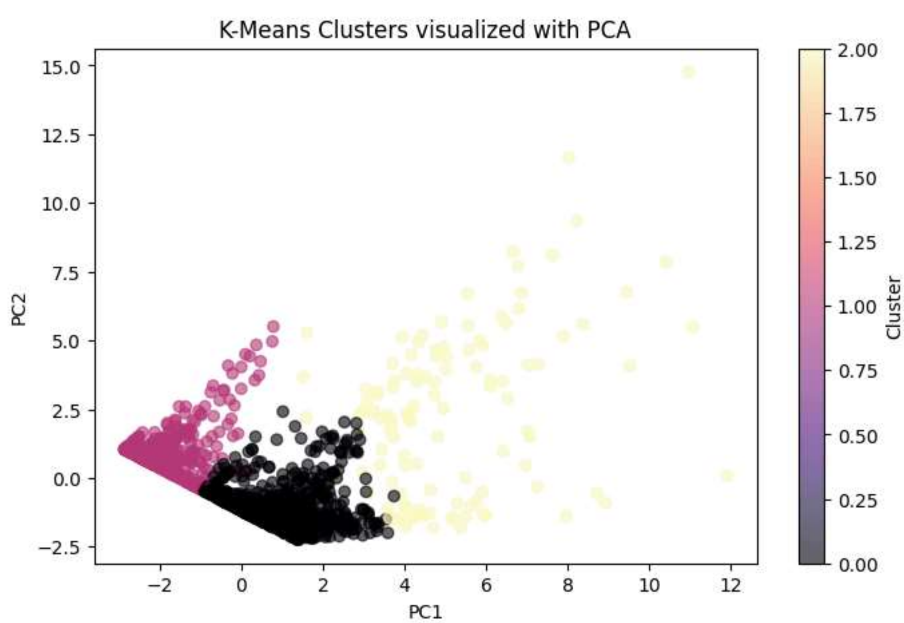
```
from sklearn.cluster import KMeans
```

```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(scaled_data)
```

```
KMeans
KMeans(n_clusters=3, random_state=42)
```

```
df['cluster'] = kmeans.labels_
```

```
plt.figure(figsize=(8,5))
plt.scatter(pca_components[:,0], pca_components[:,1], c=df['cluster'], cmap='magma', alpha=0.6)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('K-Means Clusters visualized with PCA')
plt.colorbar(label='Cluster')
plt.show()
```



Algorithms to perform-Supervised

1.Linear Regression

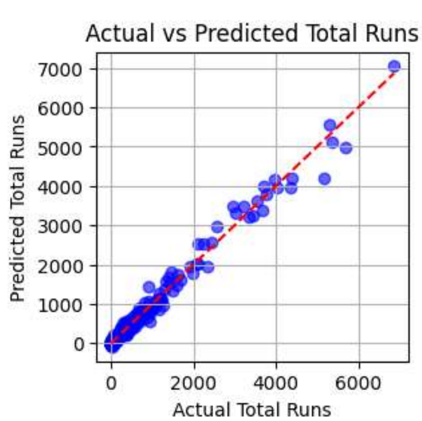
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score


X = numeric_df.drop(columns=['total_runs'])
y = numeric_df['total_runs']


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


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


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

```
▾ LinearRegression
LinearRegression()
```

```
y_pred = model.predict(X_test_scaled)


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: 16955.27
R-squared Score: 0.98
```

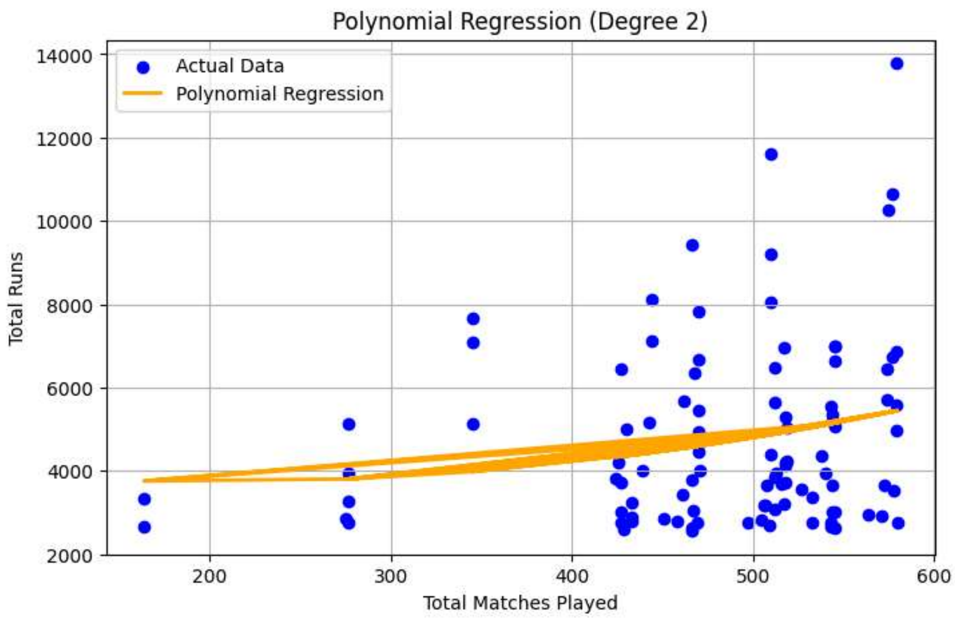
```
plt.figure(figsize=(3,3))
plt.scatter(y_test, y_pred, alpha=0.6, color='blue')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--')
plt.xlabel('Actual Total Runs')
plt.ylabel('Predicted Total Runs')
plt.title('Actual vs Predicted Total Runs')
plt.grid(True)
plt.show()
```



2.Polnomial regression

```
from sklearn.preprocessing import PolynomialFeatures


subset = df[['total_matches_played', 'total_runs']].head(100)

X = subset[['total_matches_played']]
```

```
y = subset['total_runs']

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)


model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)


plt.figure(figsize=(8, 5))
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, y_pred, color='orange', label='Polynomial Regression', linewidth=2)
plt.title('Polynomial Regression (Degree 2)')
plt.xlabel('Total Matches Played')
plt.ylabel('Total Runs')
plt.legend()
plt.grid(True)
plt.show()

# Print metrics
print("MSE:", mean_squared_error(y, y_pred))
print("R² Score:", r2_score(y, y_pred))
```

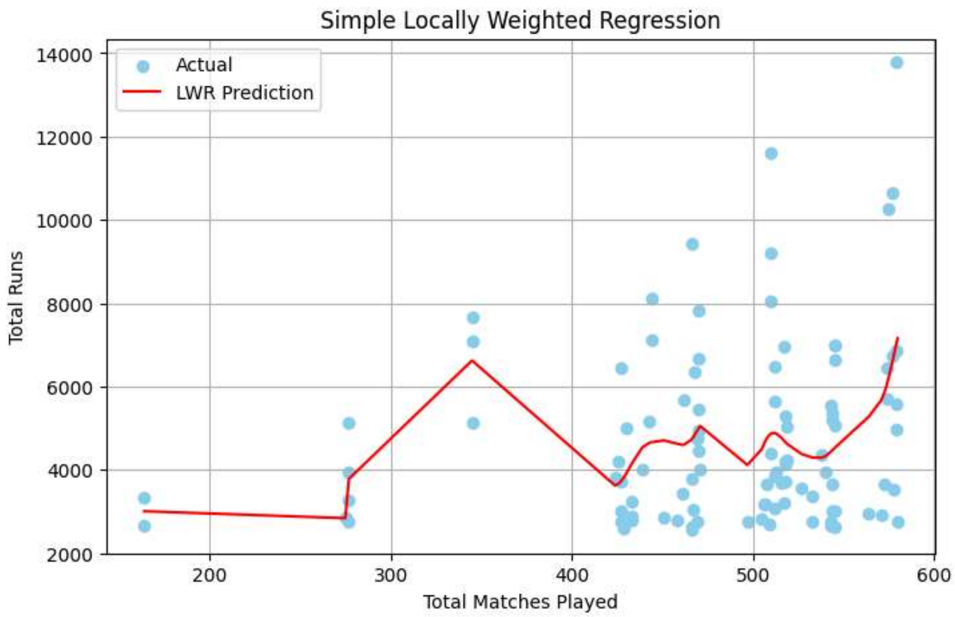



```
MSE: 4523139.886022998
R² Score: 0.04251056912120732
```

3.Locally Weighted regression

```
subset = df[['total_matches_played', 'total_runs']].head(100)
X = subset['total_matches_played'].values
y = subset['total_runs'].values


sorted_indices = X.argsort()
X = X[sorted_indices]
y = y[sorted_indices]


def weights(x, x0, tau):
    return np.exp(-(x - x0) ** 2 / (2 * tau ** 2))

def predict(x0, X, y, tau):
    w = weights(X, x0, tau)
    W = np.diag(w)
    X_design = np.c_[np.ones_like(X), X]
    theta = np.linalg.pinv(X_design.T @ W @ X_design) @ X_design.T @ W @ y
    return np.dot([1, x0], theta)
```

```
tau = 10  # Bandwidth
y_pred = [predict(x0, X, y, tau) for x0 in X]


plt.figure(figsize=(8, 5))
plt.scatter(X, y, label='Actual', color='skyblue')
plt.plot(X, y_pred, color='red', label='LWR Prediction')
plt.xlabel('Total Matches Played')
plt.ylabel('Total Runs')
plt.title('Simple Locally Weighted Regression')
plt.legend()
plt.grid(True)
plt.show()
```

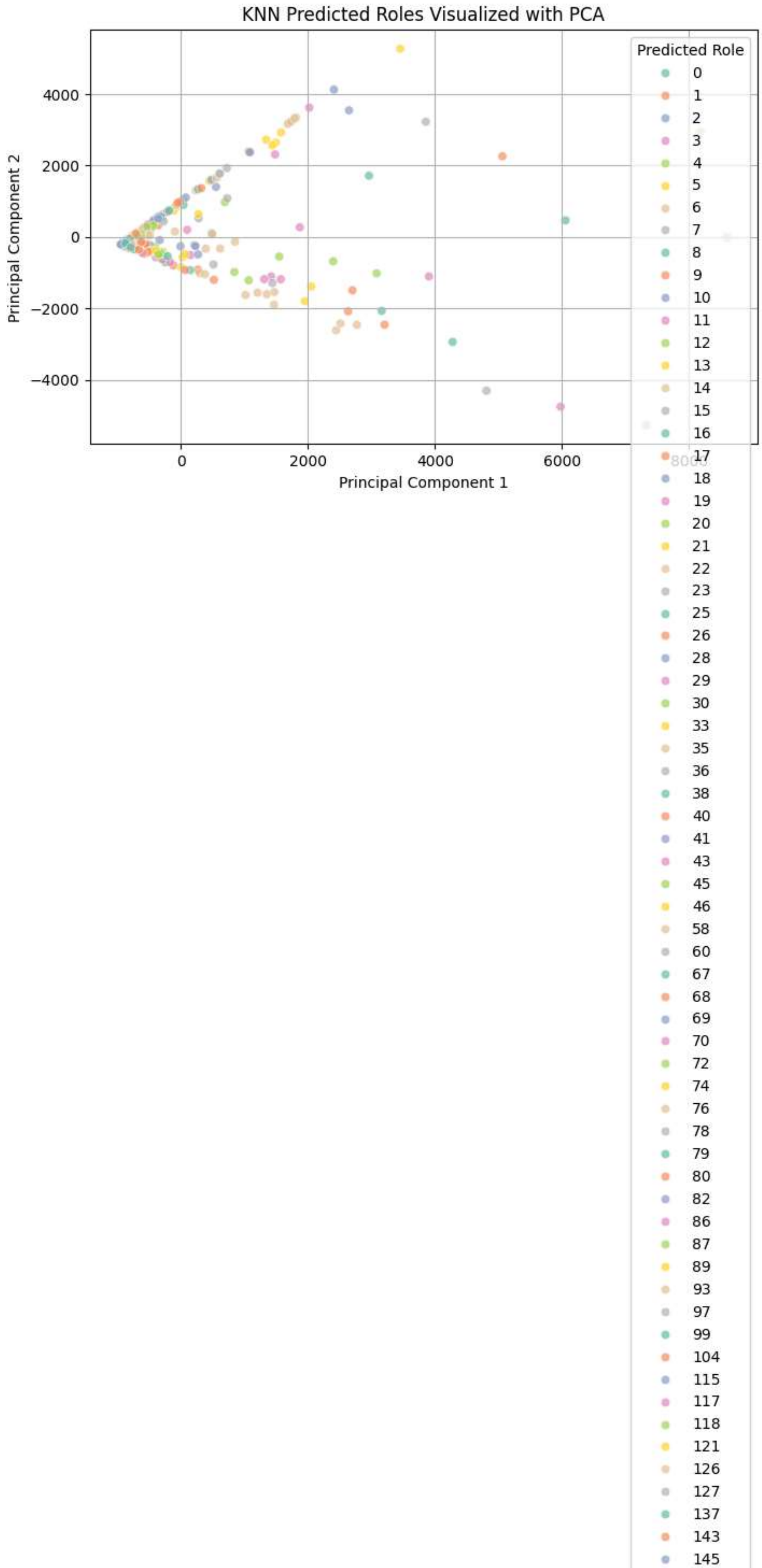


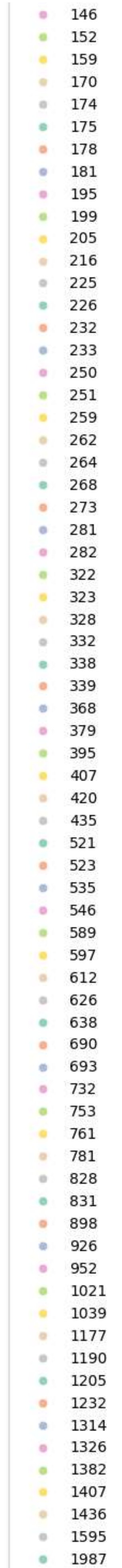
4.KNN

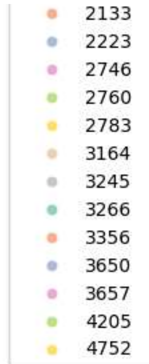
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns


features = ['total_runs', 'total_balls_faced', 'total_wickets_taken',
            'total_runs_conceded', 'total_overs_bowled', 'total_matches_played',
            'matches_played_as_batter', 'matches_played_as_bowler',
            'matches_won', 'matches_lost', 'player_of_match_awards']

X = df[features]
y = df['role']


knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)


print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
        2328       0.00      0.00      0.00         1
        2438       0.00      0.00      0.00         1
        2555       0.00      0.00      0.00         1
        2746       0.00      0.00      0.00         0
        2760       0.00      0.00      0.00         0
        2783       0.00      0.00      0.00         0
        2949       0.00      0.00      0.00         1
        3007       0.00      0.00      0.00         1
        3164       0.00      0.00      0.00         0
        3204       0.00      0.00      0.00         1
        3245       0.00      0.00      0.00         0
        3266       0.00      0.00      0.00         0
        3338       0.00      0.00      0.00         1
        3356       0.00      0.00      0.00         0
        3434       0.00      0.00      0.00         1
        3555       0.00      0.00      0.00         1
        3650       0.00      0.00      0.00         0
        3657       0.00      0.00      0.00         0
        3671       0.00      0.00      0.00         1
        3717       0.00      0.00      0.00         1
        3766       0.00      0.00      0.00         1
        3955       0.00      0.00      0.00         1
        4019       0.00      0.00      0.00         1
        4205       0.00      0.00      0.00         0
        4355       0.00      0.00      0.00         1
        4402       0.00      0.00      0.00         1
        4752       0.00      0.00      0.00         0
        5157       0.00      0.00      0.00         1
        5289       0.00      0.00      0.00         1
        5357       0.00      0.00      0.00         1
        5692       0.00      0.00      0.00         1
        6868       0.00      0.00      0.00         1

    accuracy                           0.02       308
   macro avg       0.00      0.00      0.00       308
weighted avg       0.02      0.02      0.01       308

Confusion Matrix:
 [[3 0 2 ... 0 0 0]
 [0 1 2 ... 0 0 0]
 [1 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
import seaborn as sns


features = ['total_runs', 'total_balls_faced', 'total_wickets_taken',
            'total_runs_conceded', 'total_overs_bowled', 'total_matches_played',
            'matches_played_as_batter', 'matches_played_as_bowler',
            'matches_won', 'matches_lost', 'player_of_match_awards']

X = df[features]
y = df['role']


knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)


pca = PCA(n_components=2)
X_test_pca = pca.fit_transform(X_test)


plt.figure(figsize=(8, 5))
sns.scatterplot(x=X_test_pca[:, 0], y=X_test_pca[:, 1], hue=y_pred, palette='Set2', alpha=0.7)
plt.title('KNN Predicted Roles Visualized with PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Predicted Role')
plt.grid(True)
plt.show()
```


KNN Predicted Roles Visualized with PCA
Principal Component 1
Principal Component 2
4000
2000
0
−2000
−4000
0
2000
4000
6000
Predicted Role
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
25
26
28
29
30
33
35
36
38
40
41
43
45
46
58
60
67
68
69
70
72
74
76
78
79
80
82
86
87
89
93
97
99
104
115
117
118
121
126
127
137
143
145


146
152
159
170
174
175
178
181
195
199
205
216
225
226
232
233
250
251
259
262
264
268
273
281
282
322
323
328
332
338
339
368
379
395
407
420
435
521
523
535
546
589
597
612
626
638
690
693
732
753
761
781
828
831
898
926
952
1021
1039
1177
1190
1205
1232
1314
1326
1382
1407
1436
1595
1987


2133
2223
2746
2760
2783
3164
3245
3266
3356
3650
3657
4205
4752