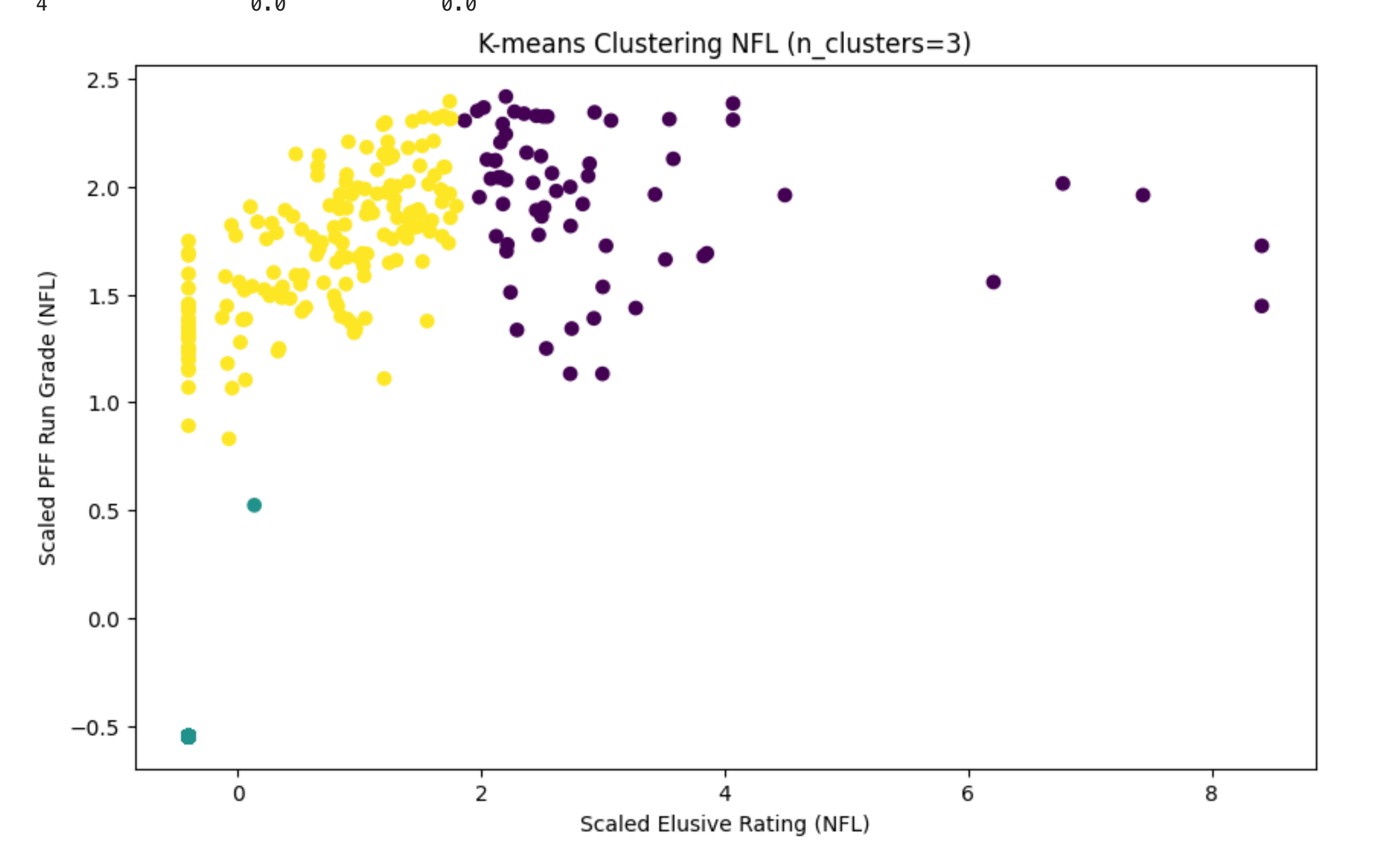
```
In [1]: import pandas as pd
        import numpy as np
        import altair as alt
        import random
```

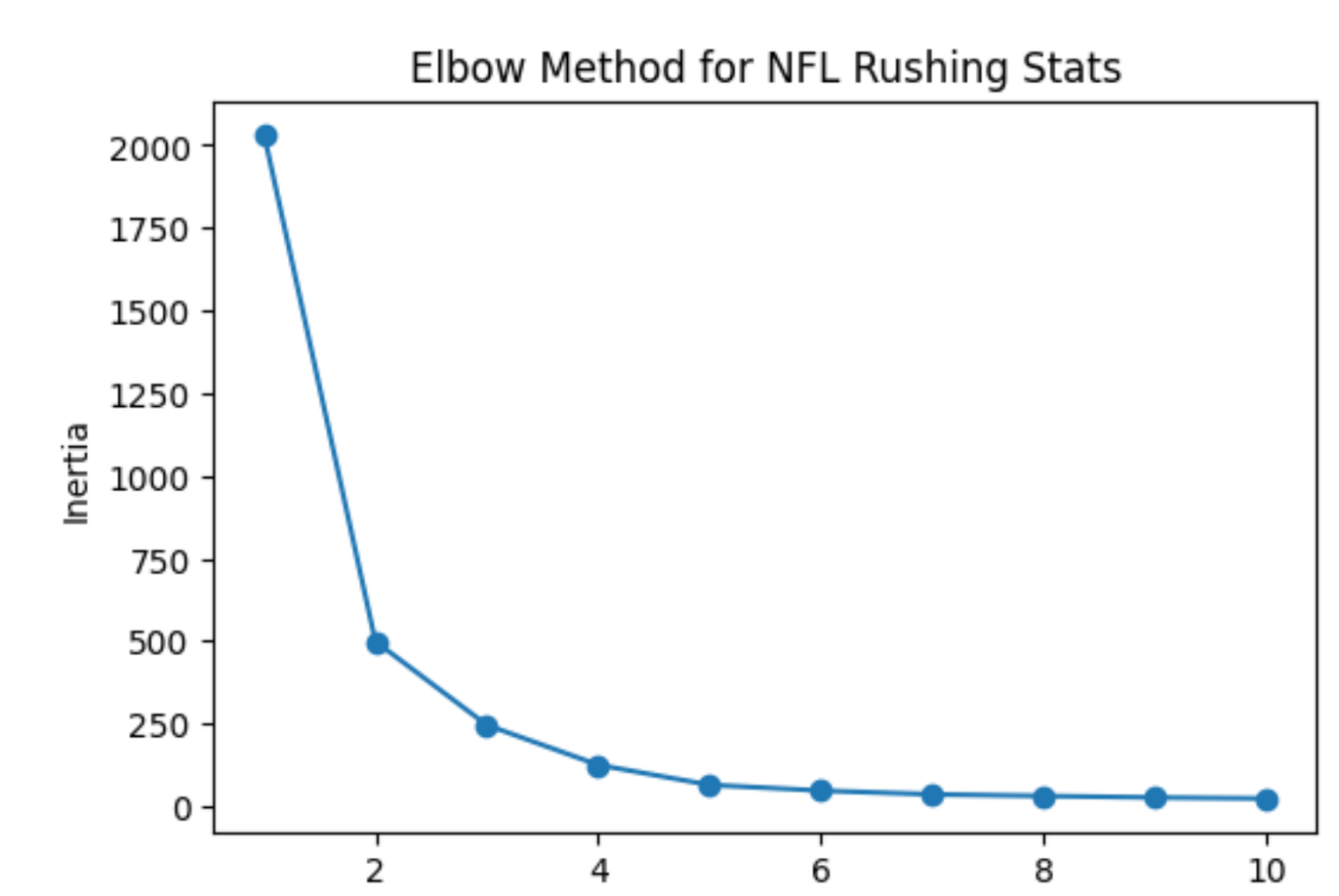
Dictionary

- `#` : Jersey Number
- `POS` : Season position
- `#G` : Number of games in which the player appeared
- `SNP` : Snaps lined up on the field on run plays
- `ATT` : Designed Rushing Attempts
- `YDS` : Rushing Yards
- `YPA` : Rushing Yards per Attempt
- `TD` : Rushing Touchdowns
- `FUM` : Fumbles
- `OFF` : PFF Grade for Offense
- `RUN` : PFF Grade for Rushing
- `FUM` : PFF Grade for HandsFumble
- `RBLK` : PFF Grade for Run Blocking
- `YCO` : Yards After Contact
- `YCO/A` : Yards After Contact per Attempt
- `MTF` : Missed Tackles Forced after a Rush
- `LNG` : Longest
- `10+` : Explosive Runs - runs over 10 yards
- `ZONE` : Designed Rushing Attempts
- `GAP` : Designed Rushing Attempts
- `SCR` : Scrambles - undesigned runs by the QB
- `SYDS` : Yardage accumulated on scrambles
- `DYDS` : Yardage accumulated on designed runs
- `D15+` : Designed Rushing Attempts more than 15 yards
- `BAY` : Rushing yardage on designed attempts more than 15 yards
- `BAY%` : Breakaway Percentage
- `1st` : First Downs
- `PEN` : Total (Declined+Offset): Total and (declined or offsetting) penalties
- `RECV` : PFF Grade for Pass Routes
- `PBLK` : PFF Grade for Pass Blocking
- `TGT` : Receiving Targets
- `REC` : Receptions
- `YDS` : Receiving Yards
- `RSNP` : Snaps where running a receiving route
- `Y/RR` : Yards per Route Run
- `DRP` : Drops - on-target passes dropped by the receiver
- `ELU` : Elusive Rating - A PFF Signature stat measuring success and impact of a runner with the ball independently of the blocking

```
In [2]: df_filtered = pd.read_excel('PFFRBModelData(2014-2024)-Filtered.xlsx')
        len(df_filtered.columns)
Out[2]: 94

In [3]: df_filtered.size
Out[3]: 95598

In [4]: import matplotlib.pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.datasets import make_blobs # Optional, for generating sample data
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import silhouette_score

In [5]: def clusterRushing_NFL(num_clusters=3):
            features = ['elusive_rating_NFL', 'grades_run_NFL']
            df = df_filtered.dropna(subset=features).copy()  # Only use rows with NFL data

            print(f"Rows before dropna: {len(df_filtered)}")
            print(f"Rows after dropna: {len(df)}")
            print(df[features].head())

            if df.empty:
                print("No NFL data available for clustering. Check your merge and data sources.")
                return

            try:
                scaler = StandardScaler()
                X_scaled = scaler.fit_transform(df[features])

                kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
                df['cluster'] = kmeans.fit_predict(X_scaled)

                # Elbow method
                inertia_values = []
                k_values = range(1, 11)
                for k in k_values:
                    kmeans_elbow = KMeans(n_clusters=k, random_state=42, n_init=10)
                    kmeans_elbow.fit(X_scaled)
                    inertia_values.append(kmeans_elbow.inertia_)

                plt.figure(figsize=(10, 6))
                plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['cluster'], cmap='viridis')
                plt.title(f'K-means Clustering NFL (n_clusters={num_clusters})')
                plt.xlabel('Scaled Elusive Rating (NFL)')
                plt.ylabel('Scaled PFF Run Grade (NFL)')
                plt.show()

                plt.figure(figsize=(6, 4))
                plt.plot(k_values, inertia_values, marker='o')
                plt.xlabel('Number of Clusters (k)')
                plt.ylabel('Inertia')
                plt.title('Elbow Method for NFL Rushing Stats')
                plt.show()

                print(f"Inertia: {kmeans.inertia_}")
                score = silhouette_score(X_scaled, df['cluster'])
                print(f"Silhouette Score: {score}")

                cluster_means = df.groupby('cluster')[features].mean()
                print("\nCluster Means:")
                print(cluster_means)

            except Exception as e:
                print(f"An error occurred: {e}")

        # Example usage:
        clusterRushing_NFL(num_clusters=3)
```

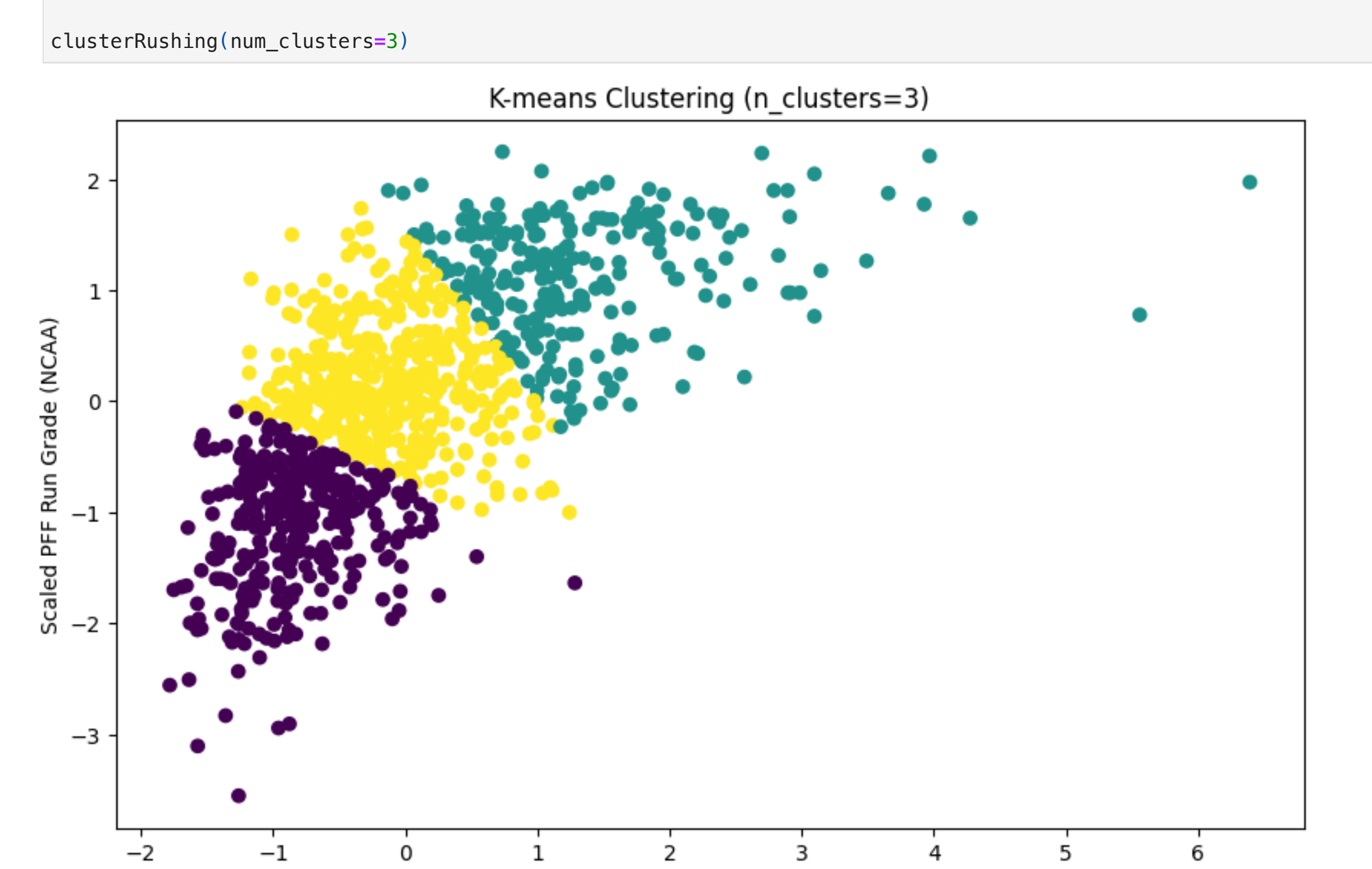
```
Rows before dropna: 1017
Rows after dropna: 1017
   elusive_rating_NFL  grades_run_NFL
0                44.2            70.0
1                73.0            92.8
2                98.7            76.0
3                 0.0             0.0
4                 0.0             0.0
```

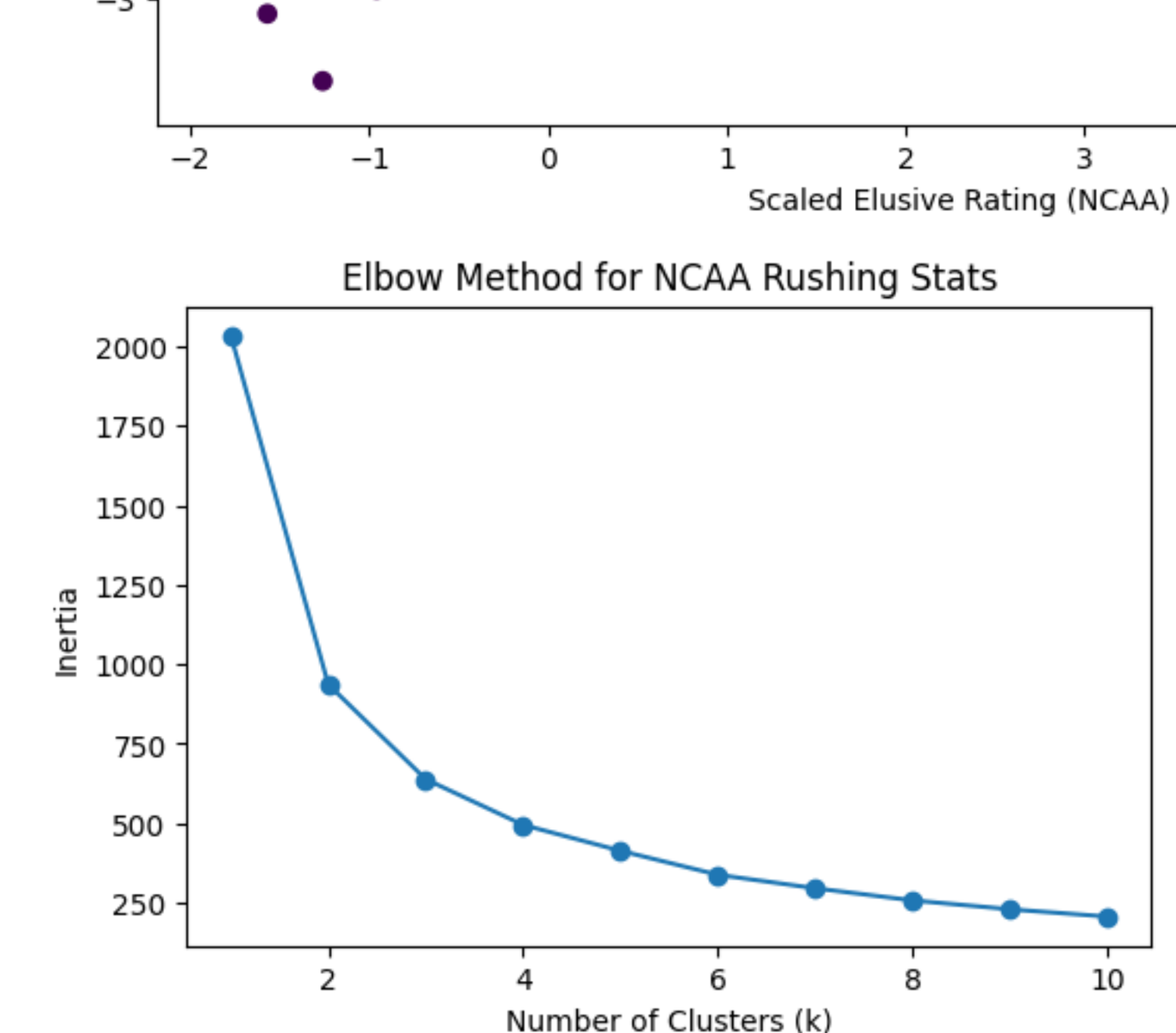





```
Inertia: 245.62309150905077
Silhouette Score: 0.8746368576210871

Cluster Means:
         elusive_rating_NFL  grades_run_NFL
cluster
0                116.992188       78.550000
1                  0.023711        0.043557
2                 37.087006       71.033898
```

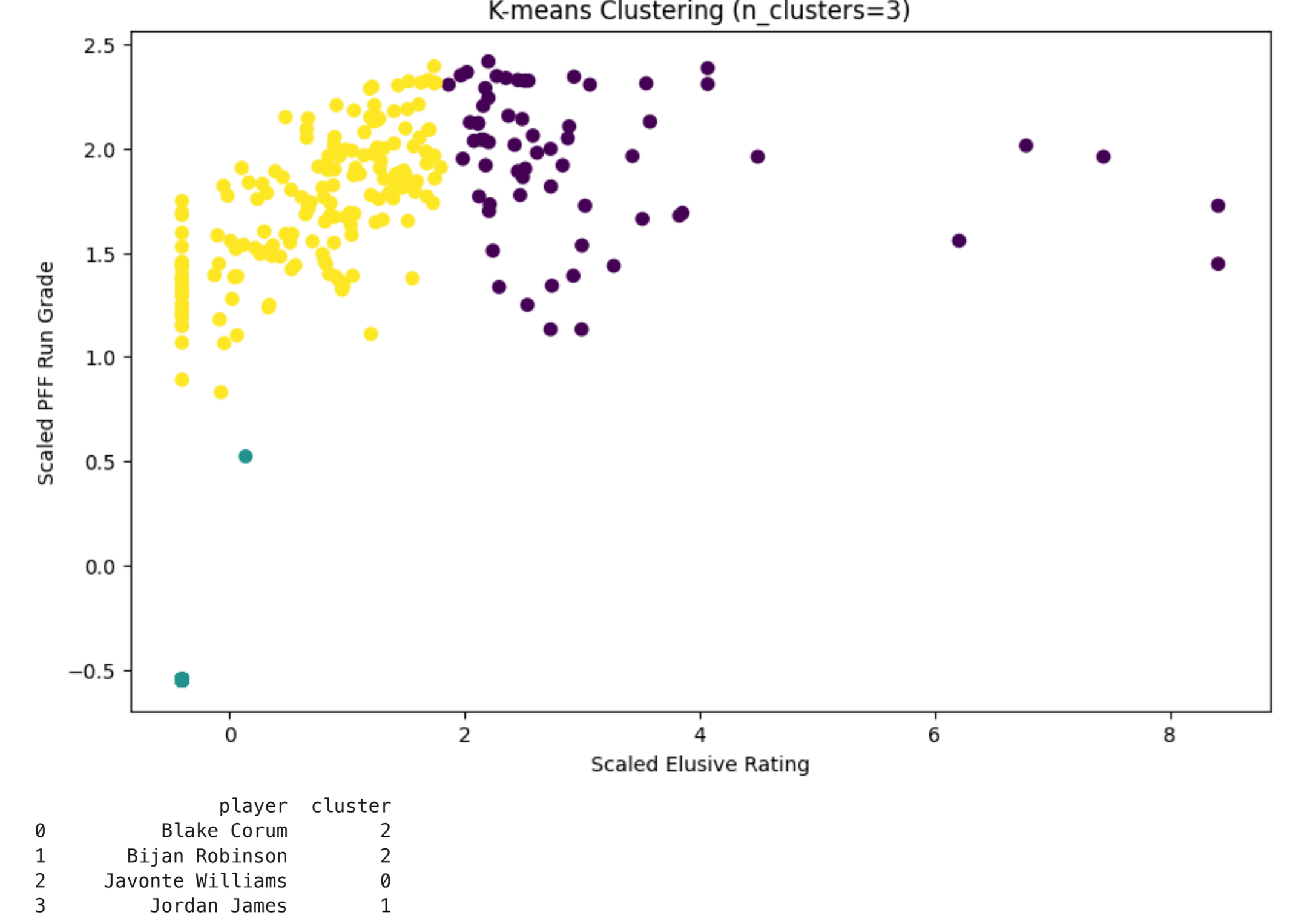
```
In [6]: def clusterRushing(num_clusters=3):
            features = ['elusive_rating_NCAA', 'grades_run_NCAA']
            df = df_filtered.dropna(subset=features).copy()  # Only use rows with NCAA data

            try:
                scaler = StandardScaler()
                X_scaled = scaler.fit_transform(df[features])

                kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
                df['cluster'] = kmeans.fit_predict(X_scaled)

                # Elbow method
                inertia_values = []
                k_values = range(1, 11)
                for k in k_values:
                    kmeans_elbow = KMeans(n_clusters=k, random_state=42, n_init=10)
                    kmeans_elbow.fit(X_scaled)
                    inertia_values.append(kmeans_elbow.inertia_)

                plt.figure(figsize=(10, 6))
                plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['cluster'], cmap='viridis')
                plt.title(f'K-means Clustering (n_clusters={num_clusters})')
                plt.xlabel('Scaled Elusive Rating (NCAA)')
                plt.ylabel('Scaled PFF Run Grade (NCAA)')
                plt.show()

                plt.figure(figsize=(6, 4))
                plt.plot(k_values, inertia_values, marker='o')
                plt.xlabel('Number of Clusters (k)')
                plt.ylabel('Inertia')
                plt.title('Elbow Method for NCAA Rushing Stats')
                plt.show()

                print(f"Inertia: {kmeans.inertia_}")
                score = silhouette_score(X_scaled, df['cluster'])
                print(f"Silhouette Score: {score}")

                cluster_means = df.groupby('cluster')[features].mean()
                print("\nCluster Means:")
                print(cluster_means)

            except Exception as e:
                print(f"An error occurred: {e}")

        clusterRushing(num_clusters=3)
```



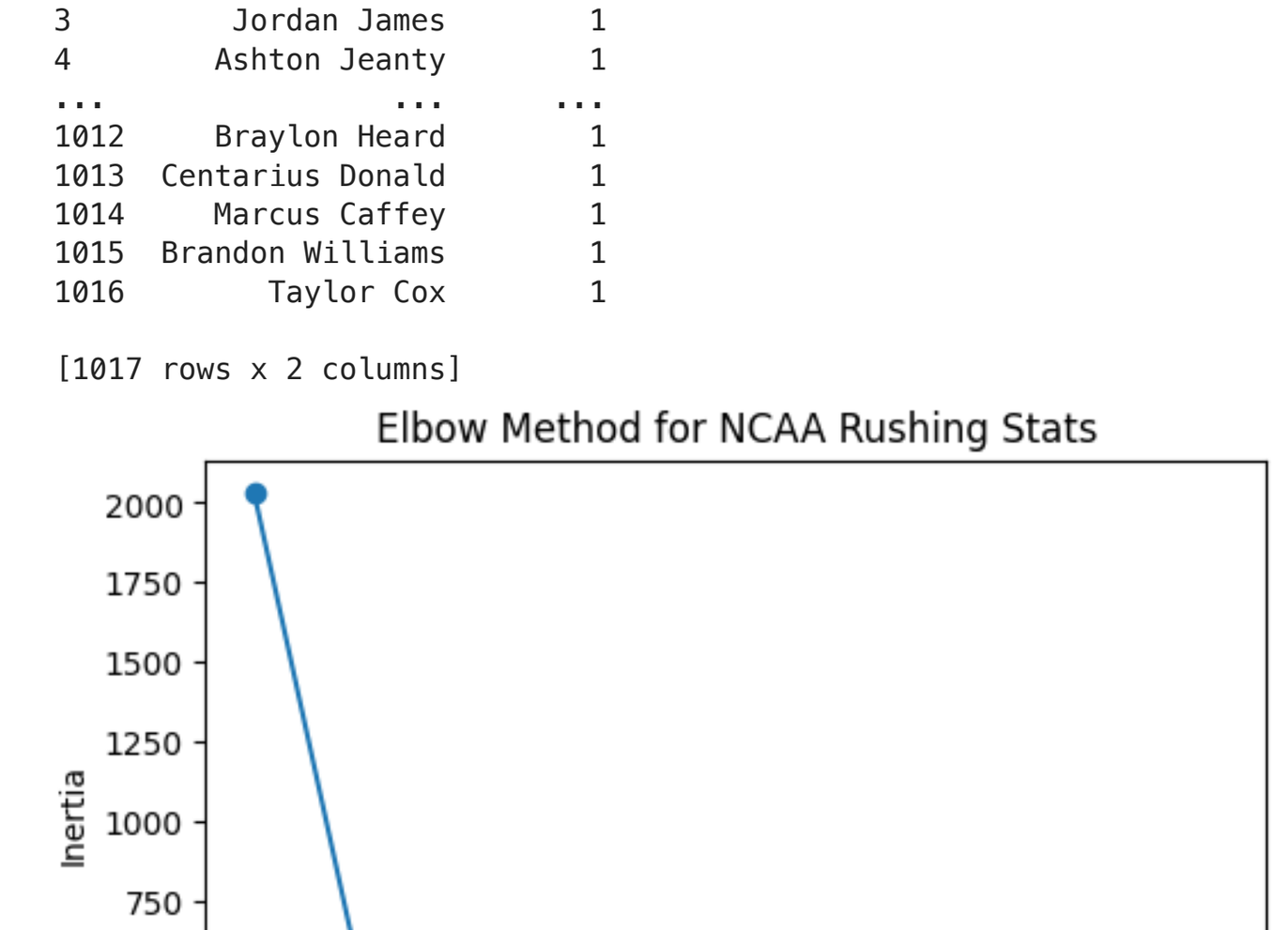



```
Inertia: 638.7389673205305
Silhouette Score: 0.3830403563568278

Cluster Means:
         elusive_rating_NCAA  grades_run_NCAA
cluster
0                  40.443789        69.200000
1                 118.860800        87.164400
2                  65.526067        79.573708
```

```
In [7]: def clusterRushing(num_clusters=3):
          features = ['elusive_rating_NFL', 'grades_run_NFL']
          df = df_filtered.fillna(0).copy() # Use .copy() to avoid SettingWithCopyWarning later

          try:
            X = df[features]
            scaler = StandardScaler()
            X_scaled = scaler.fit_transform(X) # Corrected: Removed redundant lines

            # Perform the actual clustering with the chosen num_clusters
            kmeans_final = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
            df['cluster'] = kmeans_final.fit_predict(X_scaled)

            # Evaluate Model using Elbow method
            inertia_values = []
            k_values = range(1, 11)
            for k in k_values:
              kmeans_elbow = KMeans(n_clusters=k, random_state=42, n_init=10) # Added n_init
              kmeans_elbow.fit(X_scaled)
              inertia_values.append(kmeans_elbow.inertia_)


            plt.figure(figsize=(10, 6))
            plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['cluster'], cmap='viridis')
            plt.title(f'K-means Clustering (n_clusters={num_clusters})')
            plt.xlabel('Scaled Elusive Rating')
            plt.ylabel('Scaled PFF Run Grade')
            plt.show()

            print(df[['player', 'cluster']])

            plt.figure(figsize=(6, 4))
            plt.plot(k_values, inertia_values, marker='o')
            plt.xlabel('Number of Clusters (k)')
            plt.ylabel('Inertia')
            plt.title('Elbow Method for NCAA Rushing Stats') # Corrected title
            plt.show()

            # Print Inertia for the chosen num_clusters
            print(f"Inertia (for n_clusters={num_clusters}): {kmeans_final.inertia_}")

            # Print Silhouette Score for the chosen num_clusters
            # Ensure there's more than one cluster to calculate silhouette score
            if num_clusters > 1:
                score = silhouette_score(X_scaled, df['cluster']) # Use df['cluster']
                print(f"Silhouette Score (for n_clusters={num_clusters}): {score}")
            else:
                print("Silhouette Score not applicable for 1 cluster.")


            cluster_means = df.groupby('cluster')[features].mean()
            print("\nCluster Means:")
            print(cluster_means)

          except Exception as e:
            print(f"An error occurred: {e}")


        clusterRushing(num_clusters=3)
```

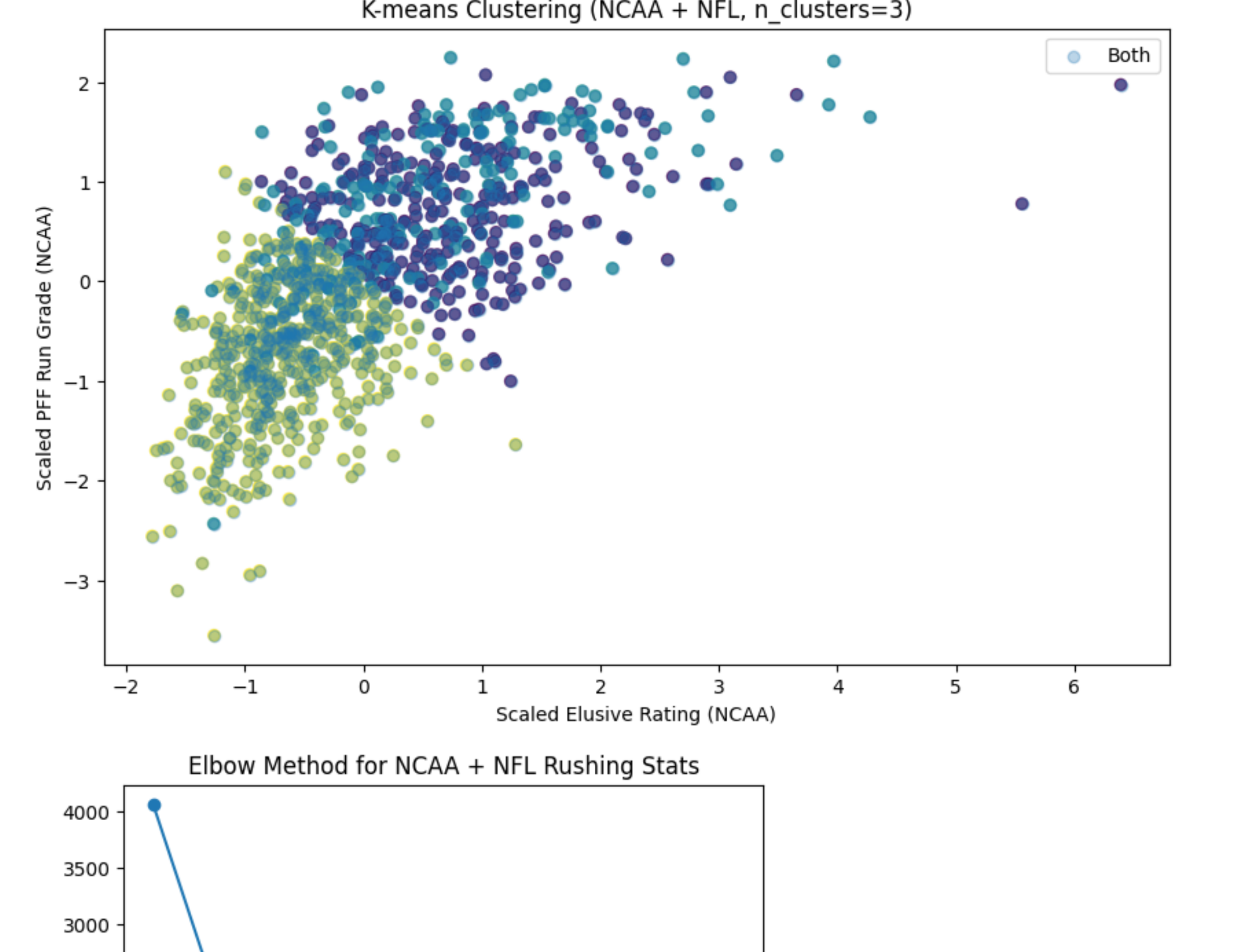


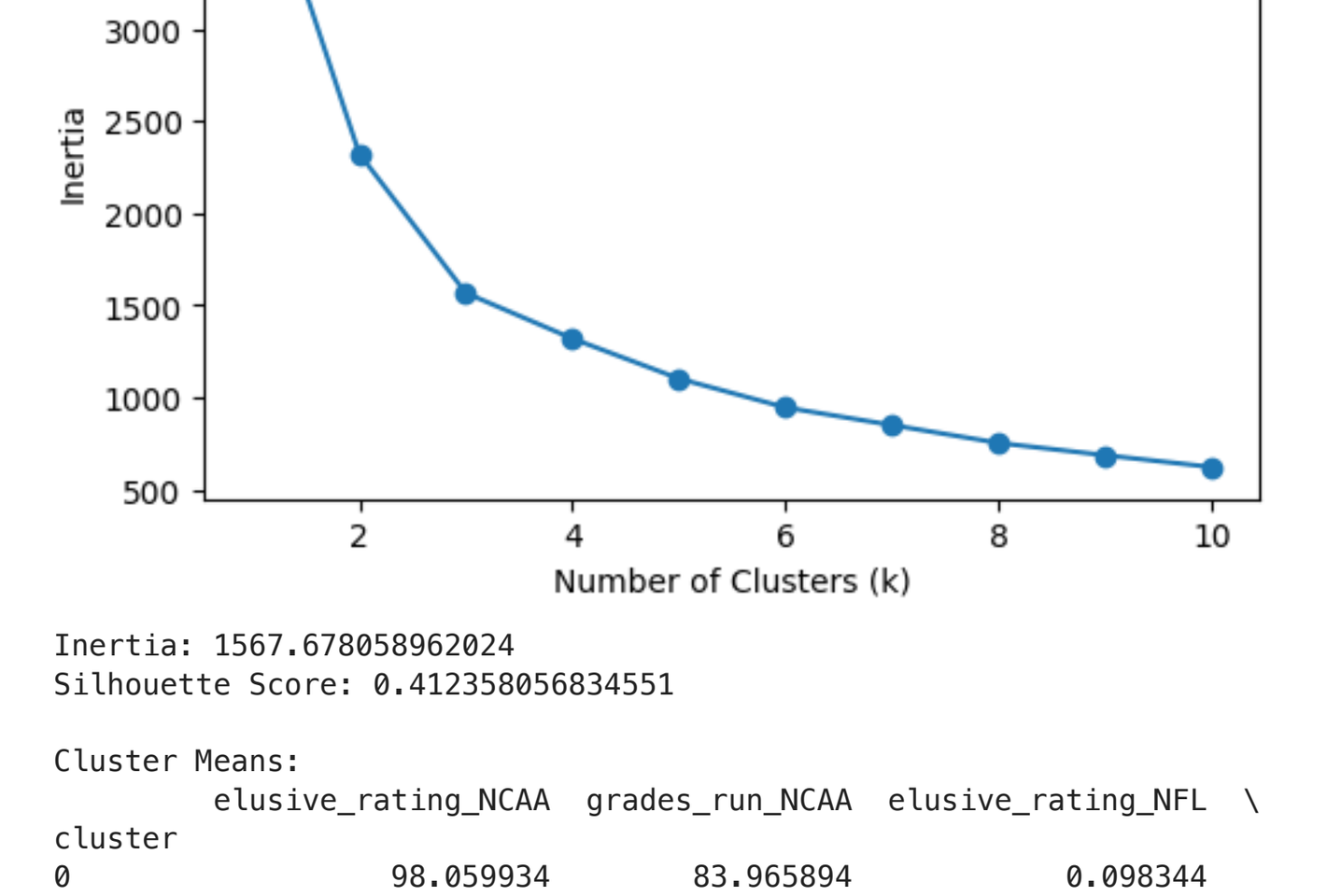

```
                player  cluster
0          Blake Corum        2
1       Bijan Robinson        2
2     Javonte Williams        0
3          Jordan James        1
4         Ashton Jeanty        1
...                ...      ...
1012     Braylon Heard        1
1013  Centarius Donald        1
1014      Marcus Caffey        1
1015  Brandon Williams        1
1016        Taylor Cox        1

[1017 rows x 2 columns]
```




```
Inertia (for n_clusters=3): 245.62309150905077
Silhouette Score (for n_clusters=3): 0.8746368576210871

Cluster Means:
         elusive_rating_NFL  grades_run_NFL
cluster
0                116.992188       78.550000
1                  0.023711        0.043557
2                 37.087006       71.033898
```

```
In [8]: def clusterRushing_all_no_pca(num_clusters=3):
            features = [
                'elusive_rating_NCAA', 'grades_run_NCAA',
                'elusive_rating_NFL', 'grades_run_NFL'
            ]
            # Keep rows with at least one non-null feature
            df = df_filtered.dropna(subset=features, how='all').copy()
            # Fill missing values with 0 (or use another imputation if you prefer)
            df[features] = df[features].fillna(0)

            # Add a column to indicate if the row is NCAA only, NFL only, or both
            def label_row(row):
                if pd.notna(row['elusive_rating_NCAA']) and pd.notna(row['elusive_rating_NFL']):
                    return 'Both'
                elif pd.notna(row['elusive_rating_NCAA']):
                    return 'NCAA only'
                elif pd.notna(row['elusive_rating_NFL']):
                    return 'NFL only'
                else:
                    return 'None'
            df['level'] = df.apply(label_row, axis=1)

            scaler = StandardScaler()
            X_scaled = scaler.fit_transform(df[features])

            kmeans = KMeans(n_clusters=num_clusters, random_state=42, n_init=10)
            df['cluster'] = kmeans.fit_predict(X_scaled)

            # Elbow method
            inertia_values = []
            k_values = range(1, 11)
            for k in k_values:
                kmeans_elbow = KMeans(n_clusters=k, random_state=42, n_init=10)
                kmeans_elbow.fit(X_scaled)
                inertia_values.append(kmeans_elbow.inertia_)

            # Plot clusters using the first two features (for visualization)
            plt.figure(figsize=(10, 6))
            scatter = plt.scatter(
                X_scaled[:, 0], X_scaled[:, 1],
                c=df['cluster'], cmap='viridis', alpha=0.7
            )
            for level in df['level'].unique():
                idx = df['level'] == level
                plt.scatter(
                    X_scaled[idx, 0], X_scaled[idx, 1],
                    label=level, alpha=0.3, marker='o'
                )
            plt.title(f'K-means Clustering (NCAA + NFL, n_clusters={num_clusters})')
            plt.xlabel('Scaled Elusive Rating (NCAA)')
            plt.ylabel('Scaled PFF Run Grade (NCAA)')
            plt.legend()
            plt.show()

            plt.figure(figsize=(6, 4))
            plt.plot(k_values, inertia_values, marker='o')
            plt.xlabel('Number of Clusters (k)')
            plt.ylabel('Inertia')
            plt.title('Elbow Method for NCAA + NFL Rushing Stats')
            plt.show()

            print(f"Inertia: {kmeans.inertia_}")
            from sklearn.metrics import silhouette_score
            score = silhouette_score(X_scaled, df['cluster'])
            print(f"Silhouette Score: {score}")

            cluster_means = df.groupby('cluster')[features].mean()
            print("\nCluster Means:")
            print(cluster_means)

        # Example usage:
        clusterRushing_all_no_pca(num_clusters=3)
```






```
Inertia: 1567.678058962024
Silhouette Score: 0.412358056834551

Cluster Means:
         elusive_rating_NCAA  grades_run_NCAA  elusive_rating_NFL  \
cluster
0                  98.059934        83.965894            0.098344
1                  90.894472        84.370352           68.809045
2                  46.889729        72.357364            0.673643

         grades_run_NFL
cluster
0              2.357616
1             75.979899
2              3.492248

In [ ]:
```