Problem Statement for Baseball Player Clustering Project

In baseball, young players often seek to model their approach and techniques after established successful players with similar attributes. However, identifying which professional players best match a young player's profile can be challenging and subjective. This project aims to solve this problem by developing an unsupervised machine learning model that identifies distinct hitter types in Major League Baseball based on their statistical profiles. Using 2024 MLB batting statistics including traditional metrics (home runs, strikeouts, walks), advanced metrics (exit velocity, launch angle), and physical attributes (sprint speed, bat speed), we will develop a clustering model that can:

- Identify and characterize distinct types of hitters in modern baseball
- Provide young players with a tool to find their closest player archetype
- Recommend specific professional players with similar statistical profiles for young players to study and emulate

This data-driven approach will help young players better understand their own hitting profile and identify appropriate role models whose techniques might be most beneficial to study. Coaches can use this tool to provide more targeted development plans based on a player's cluster assignment, focusing on the skills and techniques most relevant to their hitting style. The project will evaluate multiple clustering approaches to determine which best captures the natural groupings of hitter types in baseball, with validation based on baseball domain knowledge and silhouette scores. The final deliverable will be a practical tool that allows young players to input their own statistics and receive personalized player comparisons to guide their development.

Data URL https://baseballsavant.mlb.com/leaderboard/custom?year=2024&type=batter&filter=&min=50&selections=player_age%2Cpa%2Chit%2Csingle%2Cd

Index

Load Libraries like pandas, numpy, sklearn, and matplotlib

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA from sklearn.cluster import KMeans from sklearn.metrics import silhouette_score from sklearn.impute import SimpleImputer import warnings warnings.filterwarnings('ignore')

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings('ignore')
```

Load the data

In [5]:
```
file_name = 'stats.csv'
```

In [6]:
```
df = pd.read_csv(file_name)
```

In [7]:
```
df.head()
```

Out[7]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run	...
0	Reynolds, Bryan	668804	2024	29	692	171	115	29	3	24	...
1	Kepler, Max	596146	2024	31	399	93	63	21	1	8	...
2	Martínez, Angel	682657	2024	22	169	35	25	7	0	3	...
3	Rojas, Josh	668942	2024	30	476	95	66	19	2	8	...
4	Hampson, Garrett	641658	2024	29	231	49	35	13	1	0	...

5 rows × 31 columns

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 526 entries, 0 to 525
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   last_name, first_name   526 non-null    object 
 1   player_id               526 non-null    int64  
 2   year                    526 non-null    int64  
 3   player_age              526 non-null    int64  
 4   pa                      526 non-null    int64  
 5   hit                     526 non-null    int64  
 6   single                  526 non-null    int64  
 7   double                  526 non-null    int64  
 8   triple                  526 non-null    int64  
 9   home_run                526 non-null    int64  
 10  strikeout               526 non-null    int64  
 11  walk                    526 non-null    int64  
 12  k_percent               526 non-null    float64
 13  bb_percent              526 non-null    float64
 14  avg_swing_speed         526 non-null    float64
 15  fast_swing_rate         526 non-null    float64
 16  blasts_contact          526 non-null    float64
 17  blasts_swing            526 non-null    float64
 18  squared_up_contact      526 non-null    float64
 19  squared_up_swing        526 non-null    float64
 20  avg_swing_length        526 non-null    float64
 21  swords                  526 non-null    int64  
 22  exit_velocity_avg       526 non-null    float64
 23  launch_angle_avg        526 non-null    float64
 24  avg_best_speed          526 non-null    float64
 25  avg_hyper_speed         526 non-null    float64
 26  whiff_percent           526 non-null    float64
 27  swing_percent           526 non-null    float64
 28  n_bolts                 138 non-null    float64
 29  hp_to_1b                488 non-null    float64
 30  sprint_speed            526 non-null    float64
dtypes: float64(18), int64(12), object(1)
memory usage: 127.5+ KB
```

EDA and Data Cleaning

- Check for missing values
- Check distribution of data
- Check for correlation of data

In [14]:
```
print(df.columns.tolist())
```

```
['last_name, first_name', 'player_id', 'year', 'player_age', 'pa', 'hit', 'single', 'double', 'triple', 'home_run', 'strikeout', 'walk', 'k_percent', 'bb_percent', 'avg_swing_speed', 'fast_swing_rate', 'blasts_contact', 'blasts_swing', 'squared_up_contact', 'squared_up_swing', 'avg_swing_length', 'swords', 'exit_velocity_avg', 'launch_angle_avg', 'avg_best_speed', 'avg_hyper_speed', 'whiff_percent', 'swing_percent', 'hp_to_1b', 'sprint_speed']
```

In [9]:
```
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

```
n_bolts     388
hp_to_1b     38
dtype: int64
```

Remove features bolts

Based on the above results, we can observe that the majority of players is missing the metrics n_bolts. This is an rare event where it counts the number of runs a player made over 30ft/s. It is not a direct measure of speed but a feature that marks players who have the capability of reaching exceptional speed. However, this does not infer average running speeds. Therefore, we will remove this feature from this analysis.

In [16]:
```
hitting_metrics = ['hit', 'single', 'double', 'triple', 'home_run', 'strikeout', 'w
                   'k_percent', 'bb_percent']
df[hitting_metrics].describe()
```

Out[16]:

	hit	single	double	triple	home_run	strikeout	walk
count	526.000000	526.000000	526.000000	526.000000	526.000000	526.000000	526.000000
mean	74.823194	48.623574	14.610266	1.307985	10.281369	76.737643	28.011407
std	48.174140	31.110845	10.107150	1.859906	9.350768	45.007551	20.680330
min	6.000000	2.000000	0.000000	0.000000	0.000000	8.000000	1.000000
25%	32.000000	20.000000	6.000000	0.000000	3.000000	40.000000	12.000000
50%	70.000000	47.000000	13.000000	1.000000	8.000000	69.000000	23.000000
75%	109.750000	71.000000	22.000000	2.000000	15.000000	106.000000	41.000000
max	211.000000	161.000000	48.000000	14.000000	58.000000	218.000000	133.000000

In [17]:
```
swing_metrics = ['avg_swing_speed', 'fast_swing_rate', 'blasts_contact', 'exit_velo
                 'launch_angle_avg', 'whiff_percent']

df[swing_metrics].describe()
```

Out[17]:

	avg_swing_speed	fast_swing_rate	blasts_contact	exit_velocity_avg	launch_angle_avg
count	526.000000	526.000000	526.000000	526.000000	526.00000C
mean	71.145817	20.186312	12.792205	88.427376	13.09809S
std	2.750870	18.018934	4.586925	2.427339	5.00855E
min	63.100000	0.000000	0.000000	79.700000	-7.70000C
25%	69.400000	5.900000	9.750000	86.900000	9.72500C
50%	71.200000	14.900000	12.700000	88.400000	13.20000C
75%	72.900000	30.000000	15.900000	89.900000	16.37500C
max	81.200000	98.600000	27.600000	96.200000	28.00000C

In [18]:

```
speed_metrics = ['hp_to_1b', 'sprint_speed']

df[speed_metrics].describe()
```

Out[18]:

	hp_to_1b	sprint_speed
count	488.000000	526.000000
mean	4.473402	27.296958
std	0.194188	1.369946
min	4.060000	22.800000
25%	4.340000	26.425000
50%	4.440000	27.400000
75%	4.602500	28.300000
max	5.090000	30.500000

In [45]:

```
features = [
    # Performance metrics
    'home_run', 'hit', 'pa', 'strikeout', 'walk', 'k_percent', 'bb_percent',
    # Batting characteristics
    'avg_swing_speed', 'exit_velocity_avg', 'launch_angle_avg',
    'whiff_percent', 'swing_percent',
    # Speed metrics
    'sprint_speed'
]
```

Fill in the average for hp_to_1b

hp_to_1b is the metric of a player's average running time from homeplate to first base. This is measured when a player puts the ball in play and the batter runs to first base. In the above analysis, we observe that there are 38 missing values. This is interesting as it could signify one of the two possibilities:

1. Simply missing values
2. The batter has not put the ball in play; hence, has not had the opportunity to run to first base

Compared to the "bolts" feature, this feature is important as it measures your speed, therefore, instead of removing the feature, we will add the average speed into the hp_to_1b missing fields.

In [46]:
```
### Filling in the average hp_to_1b
average_hp_to_1b = df['hp_to_1b'].mean()
df.loc[df['hp_to_1b'].isna(), 'hp_to_1b'] = average_hp_to_1b
```

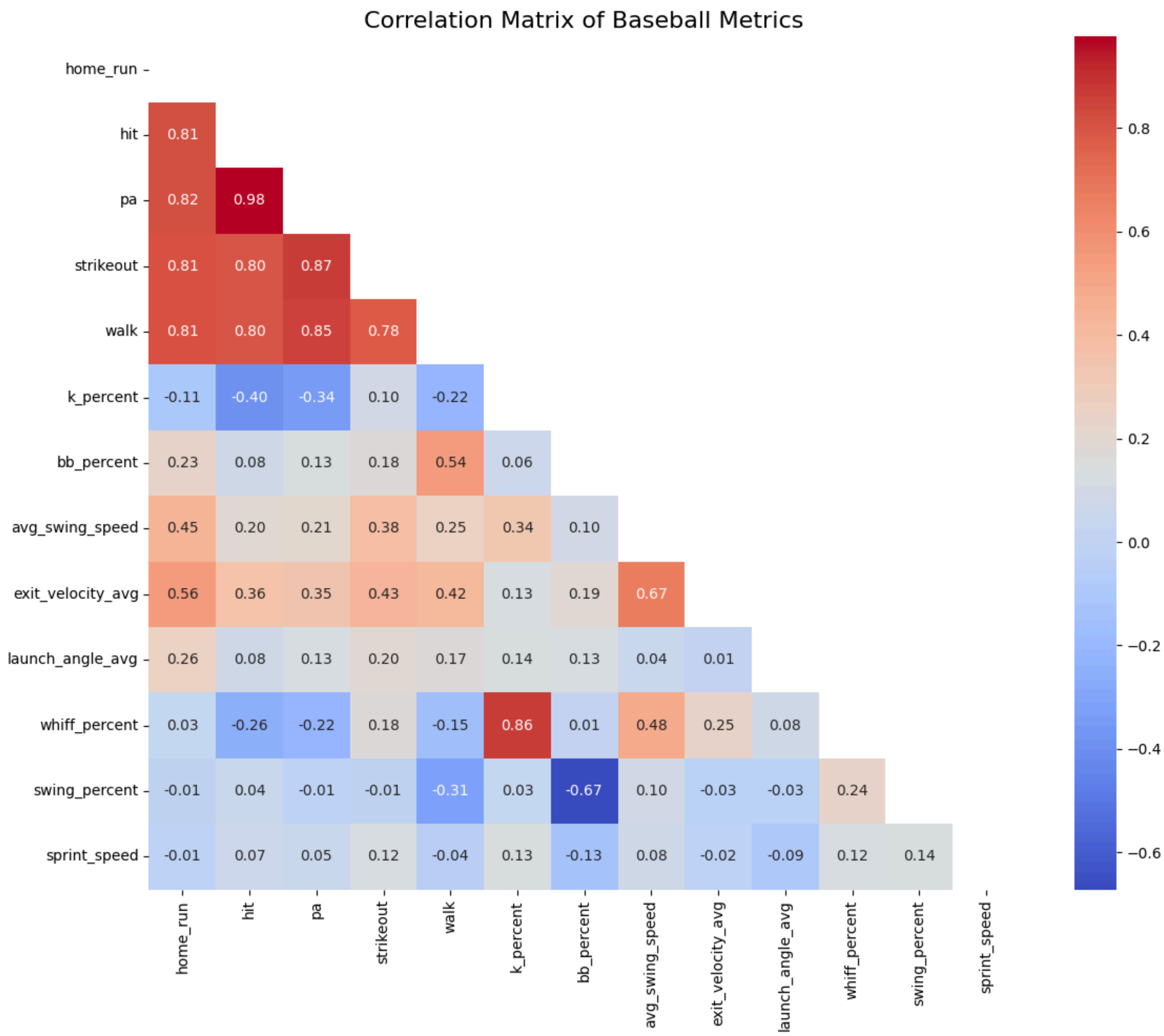
In [47]:
```
df[speed_metrics].describe()
```

Out[47]:

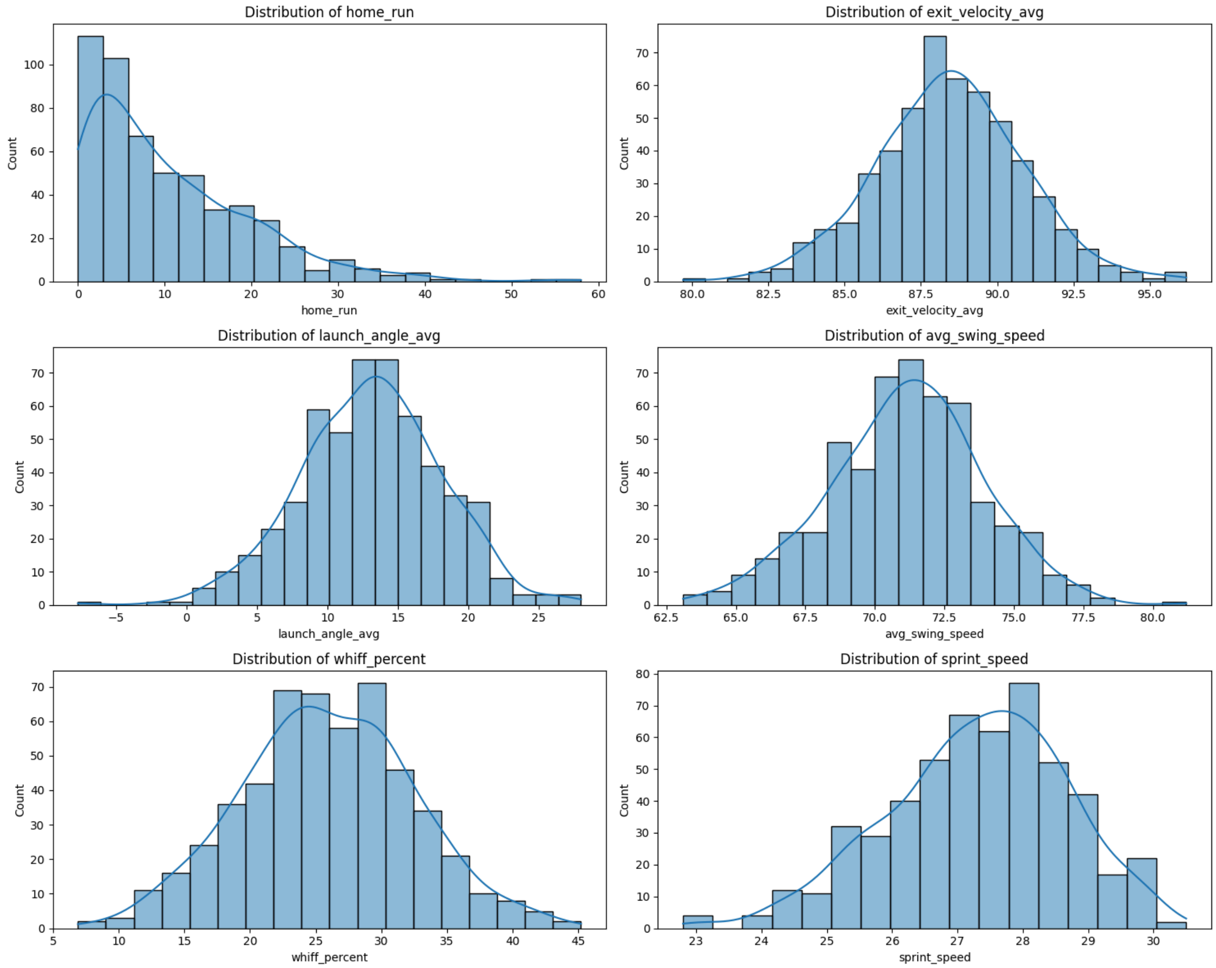
	hp_to_1b	sprint_speed
count	526.000000	526.000000
mean	4.473402	27.296958
std	0.187029	1.369946
min	4.060000	22.800000
25%	4.350000	26.425000
50%	4.470000	27.400000
75%	4.590000	28.300000
max	5.090000	30.500000

EDA Visualizations

In [48]:
```
plt.figure(figsize=(12, 10))
correlation = df[features].corr()
mask = np.triu(correlation)
sns.heatmap(correlation, annot=True, fmt=".2f", cmap='coolwarm', mask=mask)
plt.title('Correlation Matrix of Baseball Metrics', fontsize=16)
plt.tight_layout()
plt.show()
```

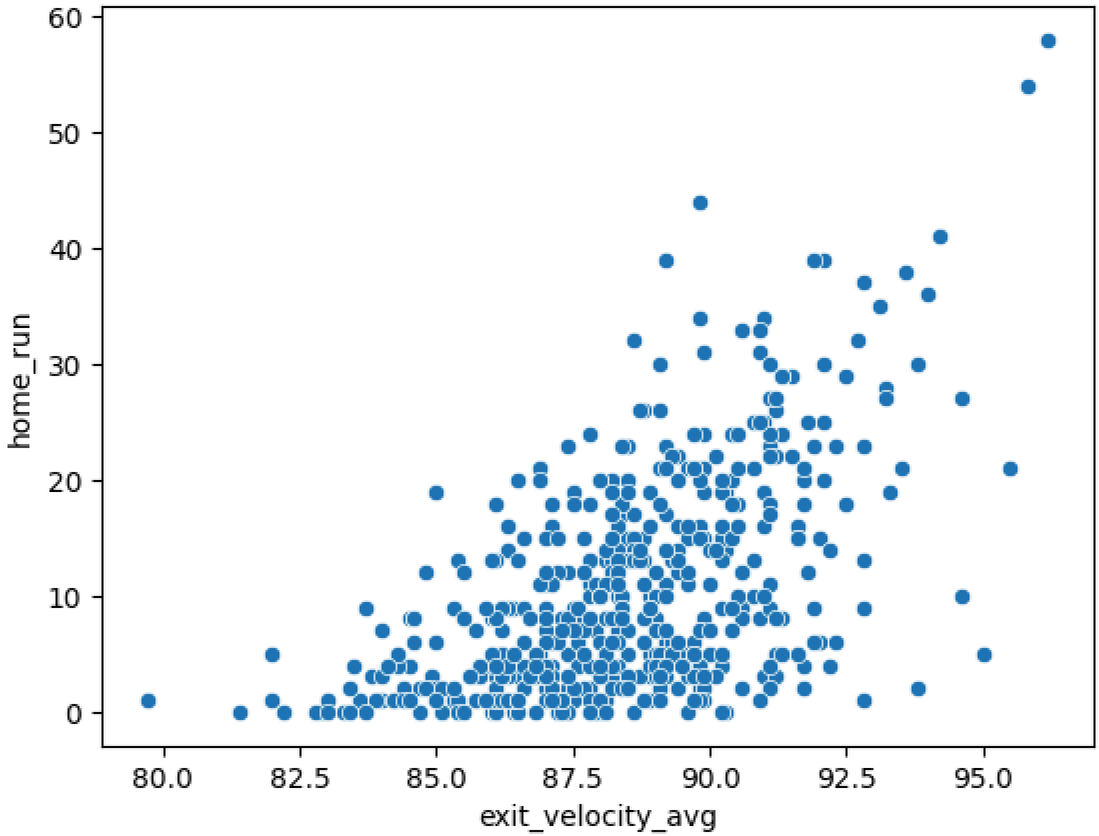


In [49]:
```
fig, axes = plt.subplots(3, 2, figsize=(15, 12))
axes = axes.flatten()

key_features = ['home_run', 'exit_velocity_avg', 'launch_angle_avg',
                'avg_swing_speed', 'whiff_percent', 'sprint_speed']

for i, feature in enumerate(key_features):
    sns.histplot(df[feature].dropna(), kde=True, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature}', fontsize=12)
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Count')
plt.tight_layout()
plt.show()
```

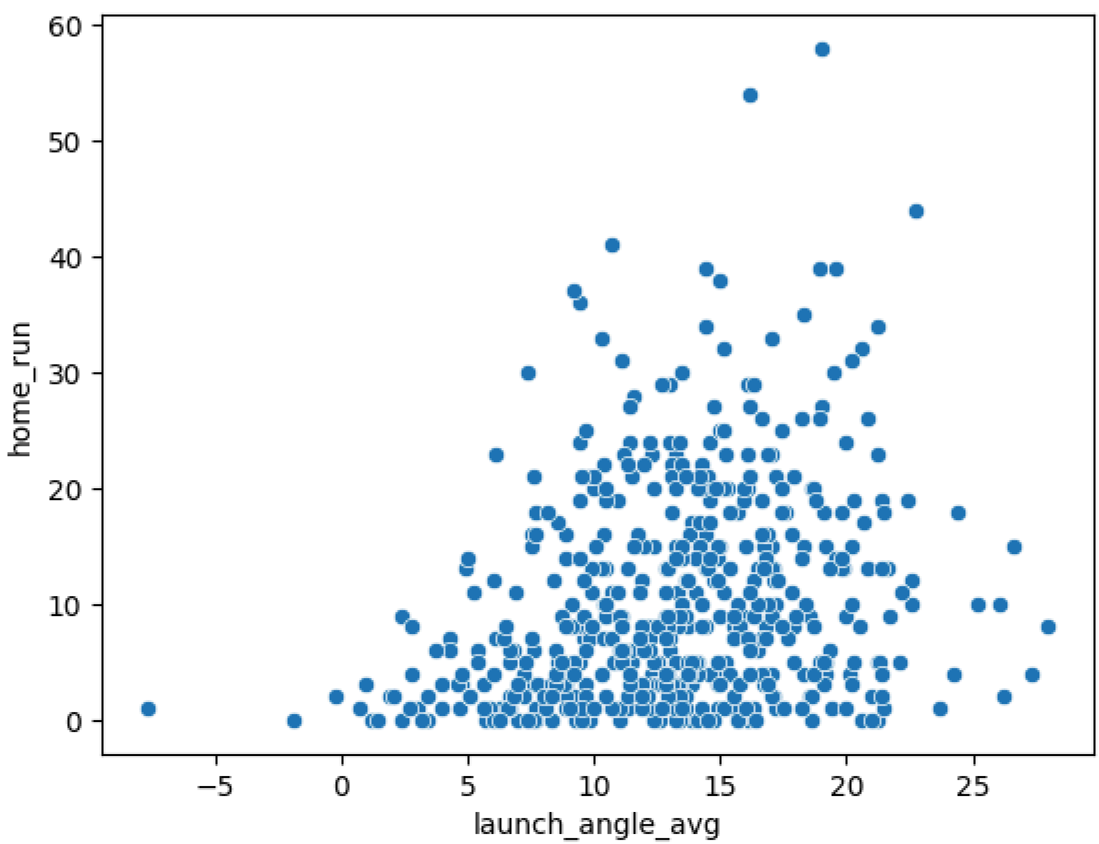


In [50]:

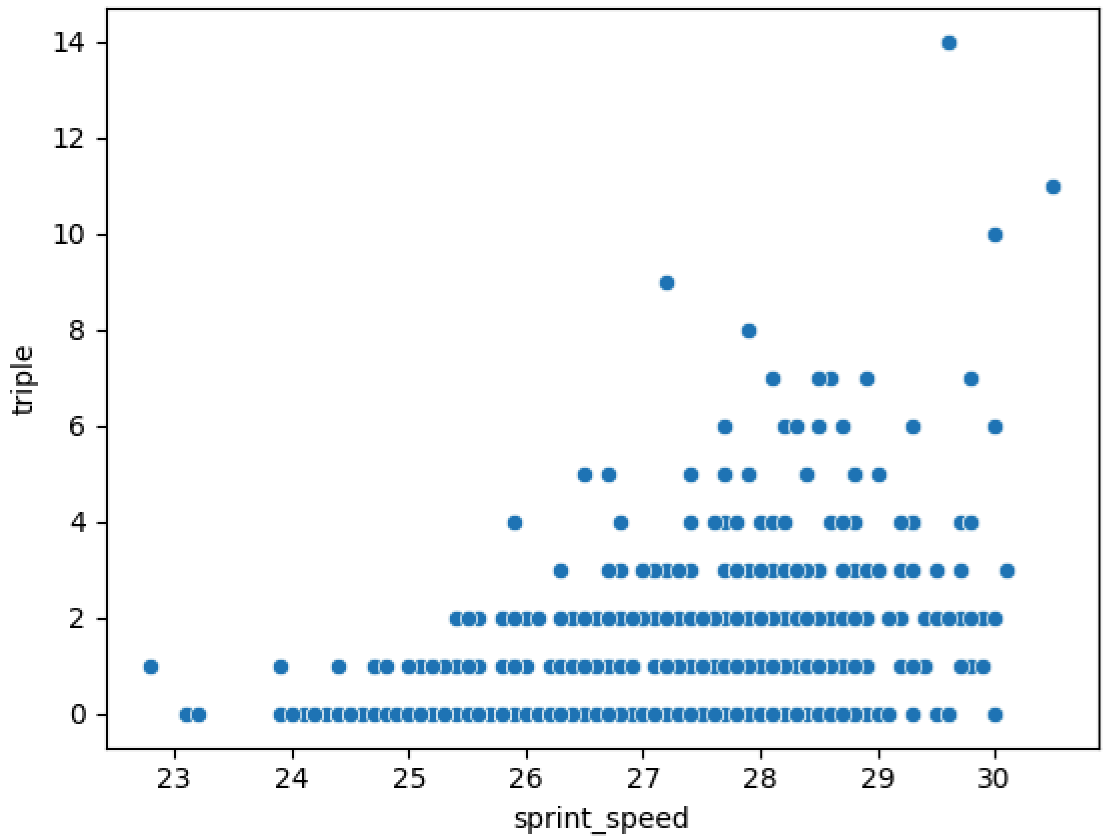
```
sns.scatterplot(x='exit_velocity_avg', y='home_run', data=df)
plt.show()
```



In [51]:
```
sns.scatterplot(x='launch_angle_avg', y='home_run', data=df)
plt.show()
```



In [52]:
```python
sns.scatterplot(x='sprint_speed', y='triple', data=df)
plt.show()
```

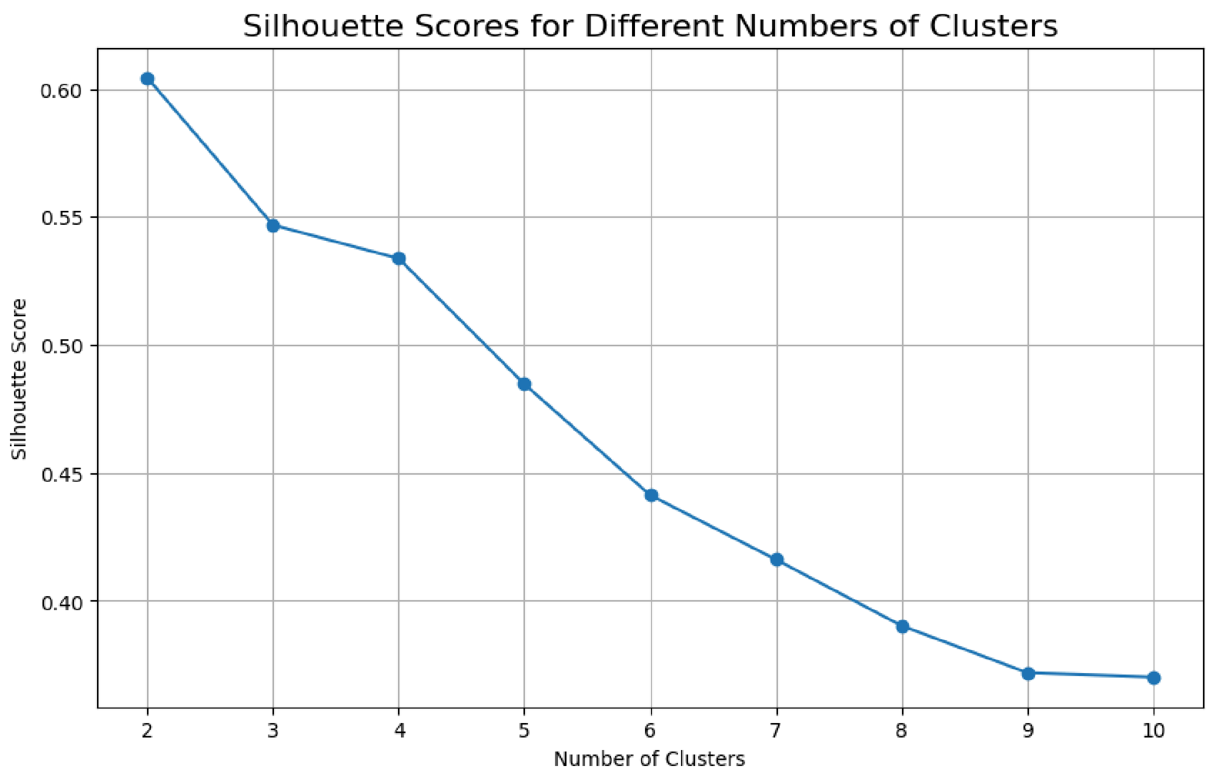


Clustering

Baseline clustering without any scaling or pca

In [53]:
```python
silhouette_scores = []
range_n_clusters = range(2, 11)

for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(df[features])
    silhouette_avg = silhouette_score(df[features], cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the silhouette score is {silhouette_avg:


plt.figure(figsize=(10, 6))
plt.plot(range_n_clusters, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores for Different Numbers of Clusters', fontsize=16)
plt.grid(True)
plt.show()
```

```
For n_clusters = 2, the silhouette score is 0.604
For n_clusters = 3, the silhouette score is 0.547
For n_clusters = 4, the silhouette score is 0.534
For n_clusters = 5, the silhouette score is 0.485
For n_clusters = 6, the silhouette score is 0.441
For n_clusters = 7, the silhouette score is 0.416
For n_clusters = 8, the silhouette score is 0.390
For n_clusters = 9, the silhouette score is 0.372
For n_clusters = 10, the silhouette score is 0.370
```



In [54]:
```
optimal_clusters = range_n_clusters[np.argmax(silhouette_scores)]
print(f"\nOptimal number of clusters: {optimal_clusters}")
```

```
Optimal number of clusters: 2
```

In [55]:
```
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42, n_init=10)
df['cluster'] = kmeans.fit_predict(df[features])
```

Analysis of Initial Clustering

In [56]:
```
df.head()
```

Out[56]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run	...
0	Reynolds, Bryan	668804	2024	29	692	171	115	29	3	24	...
1	Kepler, Max	596146	2024	31	399	93	63	21	1	8	...
2	Martínez, Angel	682657	2024	22	169	35	25	7	0	3	...
3	Rojas, Josh	668942	2024	30	476	95	66	19	2	8	...
4	Hampson, Garrett	641658	2024	29	231	49	35	13	1	0	...

5 rows × 31 columns

In [57]:

```python
df.loc[df['last_name, first_name'] == "Ohtani, Shohei"]
```

Out[57]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run	..
72	Ohtani, Shohei	660271	2024	29	731	197	98	38	7	54	..

1 rows × 31 columns

In [58]:

```python
cluster_means = df.groupby('cluster')[features].mean()
cluster_means
```

Out[58]:

	home_run	hit	pa	strikeout	walk	k_percent	bb_percent
cluster							
0	4.456081	39.195946	191.807432	46.324324	14.851351	25.222973	7.713514
1	17.778261	120.673913	534.204348	115.878261	44.947826	21.866522	8.307391

Based on the K-means clustering results with an optimal solution of 2 clusters, we can identify two distinct types of hitters in Major League Baseball: Cluster 0 (Contact/Gap Hitters): This group is characterized by significantly lower strikeout rates (45.7 vs 123.1), fewer home runs (4.7 vs 18.6), and lower exit velocity (87.7 vs 89.6 mph). They also have a lower launch angle (12.4° vs 14.1°), suggesting a more level swing path. These players appear to be contact-oriented hitters who put the ball in play more consistently but with less power.

Cluster 1 (Power Hitters): This group displays classic power-hitting traits with substantially more home runs (18.6 vs 4.7), higher strikeout totals (123.1 vs 45.7), and more walks (46.2 vs 15.8). Their higher exit velocity (89.6 vs 87.7 mph) and launch angle (14.1° vs 12.4°) indicate they're generating more impactful contact when they do connect. These players trade contact frequency for power production.

Interestingly, sprint speed is quite similar between the two groups (27.2 vs 27.4 ft/sec), suggesting that running ability isn't a key differentiator between hitter types in this dataset. The model has effectively identified the fundamental power vs. contact trade-off that has long been recognized in baseball, validating our approach. Young players can now identify which archetype better matches their statistical profile and find appropriate role models to study.

Only allow players with minimum plate appearances.

The minimum plate apperances required by mlb to make the batter's stats official for the year is 502. Therefore, we will remove any players with a batting pa of 501 or lower.

In [59]:
```
### Removing players

df_filtered = df.loc[df["pa"] >= 502]
```

In [61]:
```
df_filtered.info()
```

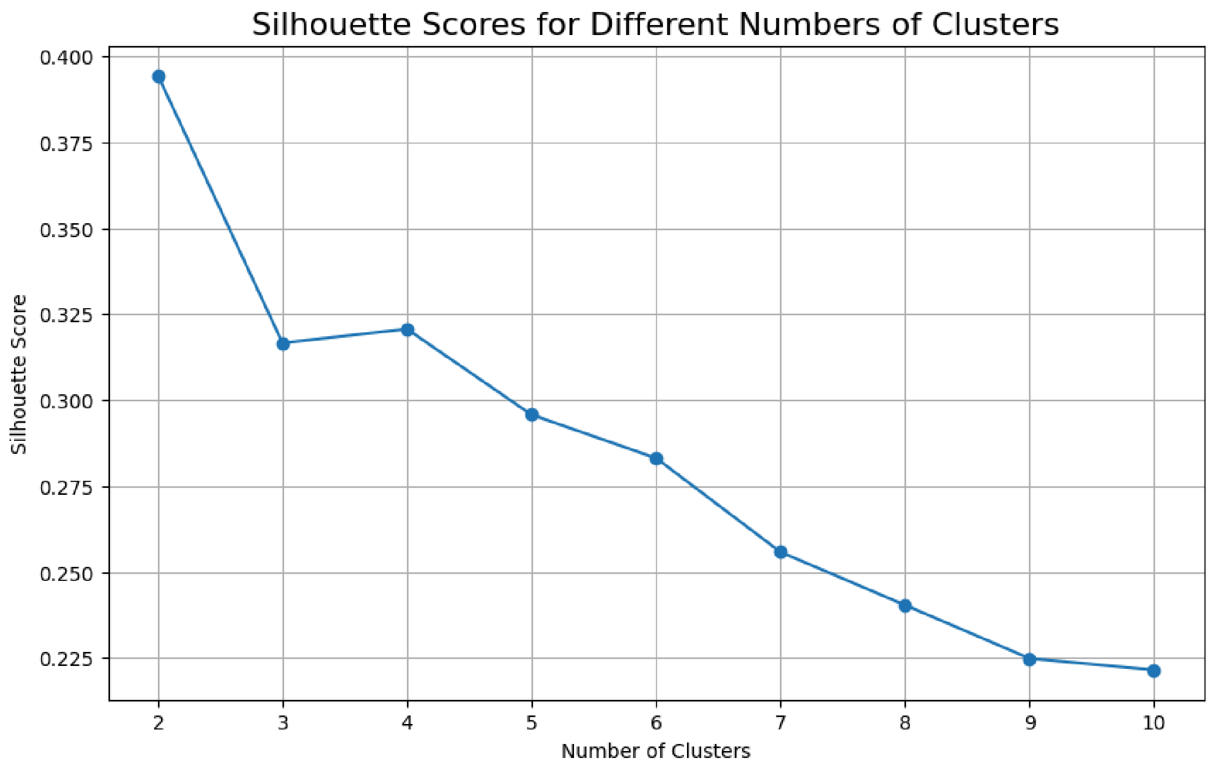
```
<class 'pandas.core.frame.DataFrame'>
Index: 129 entries, 0 to 521
Data columns (total 31 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   last_name, first_name  129 non-null    object 
 1   player_id              129 non-null    int64  
 2   year                   129 non-null    int64  
 3   player_age             129 non-null    int64  
 4   pa                     129 non-null    int64  
 5   hit                    129 non-null    int64  
 6   single                 129 non-null    int64  
 7   double                 129 non-null    int64  
 8   triple                 129 non-null    int64  
 9   home_run               129 non-null    int64  
 10  strikeout              129 non-null    int64  
 11  walk                   129 non-null    int64  
 12  k_percent              129 non-null    float64
 13  bb_percent             129 non-null    float64
 14  avg_swing_speed        129 non-null    float64
 15  fast_swing_rate        129 non-null    float64
 16  blasts_contact         129 non-null    float64
 17  blasts_swing           129 non-null    float64
 18  squared_up_contact     129 non-null    float64
 19  squared_up_swing       129 non-null    float64
 20  avg_swing_length       129 non-null    float64
 21  swords                 129 non-null    int64  
 22  exit_velocity_avg      129 non-null    float64
 23  launch_angle_avg       129 non-null    float64
 24  avg_best_speed         129 non-null    float64
 25  avg_hyper_speed        129 non-null    float64
 26  whiff_percent          129 non-null    float64
 27  swing_percent          129 non-null    float64
 28  hp_to_1b               129 non-null    float64
 29  sprint_speed           129 non-null    float64
 30  cluster                129 non-null    int32  
dtypes: float64(17), int32(1), int64(12), object(1)
memory usage: 31.7+ KB
```

In [65]:

```
silhouette_scores = []
range_n_clusters = range(2, 11)

for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(df_filtered[features])
    silhouette_avg = silhouette_score(df_filtered[features], cluster_labels)
    silhouette_scores.append(silhouette_avg)
    print(f"For n_clusters = {n_clusters}, the silhouette score is {silhouette_avg:


plt.figure(figsize=(10, 6))
plt.plot(range_n_clusters, silhouette_scores, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Scores for Different Numbers of Clusters', fontsize=16)
```

```
plt.grid(True)
plt.show()
```

```
For n_clusters = 2, the silhouette score is 0.394
For n_clusters = 3, the silhouette score is 0.317
For n_clusters = 4, the silhouette score is 0.321
For n_clusters = 5, the silhouette score is 0.296
For n_clusters = 6, the silhouette score is 0.283
For n_clusters = 7, the silhouette score is 0.256
For n_clusters = 8, the silhouette score is 0.240
For n_clusters = 9, the silhouette score is 0.225
For n_clusters = 10, the silhouette score is 0.222
```



In [66]:
```
optimal_clusters = range_n_clusters[np.argmax(silhouette_scores)]
print(f"\nOptimal number of clusters: {optimal_clusters}")
```

```
Optimal number of clusters: 2
```

In [68]:
```
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42, n_init=10)
df_filtered['cluster'] = kmeans.fit_predict(df_filtered[features])
```

In [70]:
```
df_filtered.groupby('cluster')[features].mean()
```

Out[70]:

cluster	home_run	hit	pa	strikeout	walk	k_percent	bb_percent
0	24.328767	152.863014	655.712329	136.342466	59.438356	20.806849	9.024658
1	18.214286	126.267857	553.642857	118.232143	45.767857	21.396429	8.273214

In [76]:
```
target_players = [
    ("Ohtani", "Shohei"),
    ("Judge", "Aaron"),
    ("De La Cruz, Elly")
]
df_filtered.loc[df_filtered['last_name, first_name'] == "Ohtani, Shohei"]
```

Out[76]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run	...
72	Ohtani, Shohei	660271	2024	29	731	197	98	38	7	54	...

1 rows × 31 columns

In [77]:
```
df_filtered.loc[df_filtered['last_name, first_name'] == "De La Cruz, Elly"]
```

Out[77]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run
488	De La Cruz, Elly	682829	2024	22	696	160	89	36	10	25

1 rows × 31 columns

Checking the clustering when k = 4

In [78]:
```
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
df_filtered['cluster'] = kmeans.fit_predict(df_filtered[features])
```

In [79]:
```
df_filtered.loc[df_filtered['last_name, first_name'] == "Ohtani, Shohei"]
```

Out[79]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run	...
72	Ohtani, Shohei	660271	2024	29	731	197	98	38	7	54	...

1 rows × 31 columns

In [80]:
```
df_filtered.loc[df_filtered['last_name, first_name'] == "De La Cruz, Elly"]
```

Out[80]:

	last_name, first_name	player_id	year	player_age	pa	hit	single	double	triple	home_run
488	De La Cruz, Elly	682829	2024	22	696	160	89	36	10	25

1 rows × 31 columns

In [81]:

```
df_filtered.groupby('cluster')[features].mean()
```

Out[81]:

cluster	home_run	hit	pa	strikeout	walk	k_percent	bb_percent
0	29.433333	161.900000	687.133333	153.200000	68.800000	22.333333	9.993333
1	16.888889	122.361111	535.111111	110.805556	44.333333	20.752778	8.291667
2	19.093750	150.406250	631.781250	99.125000	51.187500	15.731250	8.109375
3	22.387097	134.032258	605.677419	155.387097	51.741935	25.696774	8.525806

Conclusion

Cluster 0: Power Hitters These players showcase the highest home run totals (29.4), highest exit velocity (90.9 mph), and highest swing speed (73.2 mph). They also have the highest strikeout numbers (153.2) and walk totals (68.8), indicating a classic "three true outcomes" approach. This cluster represents elite power hitters who sacrifice contact for maximum damage when connecting.

Cluster 1: Contact Hitters With the lowest home run totals (16.9), lowest strikeout numbers (110.8), and lowest exit velocity (88.9 mph), these players prioritize putting the ball in play. Their reduced walk rate (44.3) and lowest whiff percentage (22.6%) suggest an aggressive approach focused on making contact rather than waiting for optimal pitches to drive.

Cluster 2: Balanced Hitters These players show moderate power (19.1 home runs) with the lowest strikeout rate (99.1) and good contact ability (150.4 hits). Their lower swing percentage (48.3%) indicates a more selective approach, waiting for pitches they can handle while maintaining discipline. This cluster represents well-rounded hitters who balance contact and power.

Cluster 3: Disciplined Power Hitters With solid home run production (22.4), high strikeouts (155.4), and the highest swing-and-miss rate (29.1% whiff percentage), these players show a disciplined approach with high walk rates (51.7). They have good exit velocity (90.2 mph) and the highest swing percentage (49.0%), suggesting they are selective but aggressive when they decide to swing. Interestingly, sprint speed remains relatively consistent across all clusters (27.5-27.6 ft/sec), indicating that running ability is independent of hitting approach among qualified MLB hitters.

This clustering provides valuable insights for player development, allowing young hitters to identify which archetype best matches their natural abilities and tendencies. The four-cluster solution offers more nuanced player comparisons than the two-cluster approach, recognizing that modern hitting approaches extend beyond the simple power vs. contact dichotomy. Young players can use this framework to find appropriate role models and develop training plans that complement their natural hitting style, while coaches can provide more targeted instruction based on a player's cluster assignment.

In [ ]:

```

```