In [225]:

```
import sqlalchemy
from sqlalchemy import create_engine
import numpy as np
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = "svg"
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
```

Charlie Morris - Cosc 61 Final Project

In [226]:

```
#Connect to MySQL database called 'baseball'
con_string = 'mysql+pymysql://root:C&bDanHa1@127.0.0.1:3306/baseball'
engine = create_engine(con_string)
```

In [227]:

```
query2 = """
    SHOW TABLES
"""

df2_read_sql = pd.read_sql(query2, engine)
df2_read_sql
```

Out[227]:

	Tables_in_baseball
0	award_types
1	awards
2	batting_stats
3	countries
4	errors
5	franchises
6	hall_of_fame
7	pitching_stats
8	players
9	positions
10	salaries
11	team_stats
12	teams
13	world_series

Logistic Regression: Probability of Making HOF Vesus Career Pitching SOs

In [228]:

```
#Grab the data for pitchers and their career strikeouts
#Look only at pitchers with minimum 201 strikeouts

query = """
    SELECT playerID, SUM(SO) AS Career_SO
    FROM pitching_stats
    WHERE playerID IN
        (SELECT playerID
        FROM players
        WHERE '1900-12-31' < finalGame AND finalGame < '2008-12-31')
    GROUP BY playerID
    HAVING Career_SO > 200;
"""

df_CareerSOs = pd.read_sql(query, engine)
df_CareerSOs
```

Out[228]:

	playerID	Career_SO
0	aasedo01	641.0
1	abbotgl01	484.0
2	abbotji01	888.0
3	abbotpa01	496.0
4	abernte02	765.0
...	...	...
1935	zambrvi01	529.0
1936	zimmeje02	213.0
1937	zoldasa01	207.0
1938	zuberbi01	383.0
1939	zuverge01	223.0

1940 rows × 2 columns

In [229]:

```
#Grab data on HOF players
query = """
    SELECT *
    FROM hall_of_fame;
"""

df_HOF = pd.read_sql(query, engine)
df_HOF
```

Out[229]:

	playerID	year
0	aaronha01	1982
1	alexape01	1938
2	alomaro01	2011
3	alstowa01	1983
4	andersp01	2000
...	...	...
282	wynnea01	1972
283	yastrca01	1989
284	youngcy01	1937
285	youngro01	1972
286	yountro01	1999

287 rows × 2 columns

In [230]:

```
#Merge the 2 data frames
#1 means made HOF, 0 means didn't make HOF
full_df = pd.merge(df_CareerSOs, df_HOF, on = "playerID", how = "left")
full_df['HOF_Status'] = full_df['year'].notnull().astype(int)
full_df
```

Out[230]:

	playerID	Career_SO	year	HOF_Status
0	aasedo01	641.0	NaN	0
1	abbotgl01	484.0	NaN	0
2	abbotji01	888.0	NaN	0
3	abbotpa01	496.0	NaN	0
4	abernte02	765.0	NaN	0
...	...	...	...	...
1935	zambrvi01	529.0	NaN	0
1936	zimmeje02	213.0	NaN	0
1937	zoldasa01	207.0	NaN	0
1938	zuberbi01	383.0	NaN	0
1939	zuverge01	223.0	NaN	0

1940 rows × 4 columns

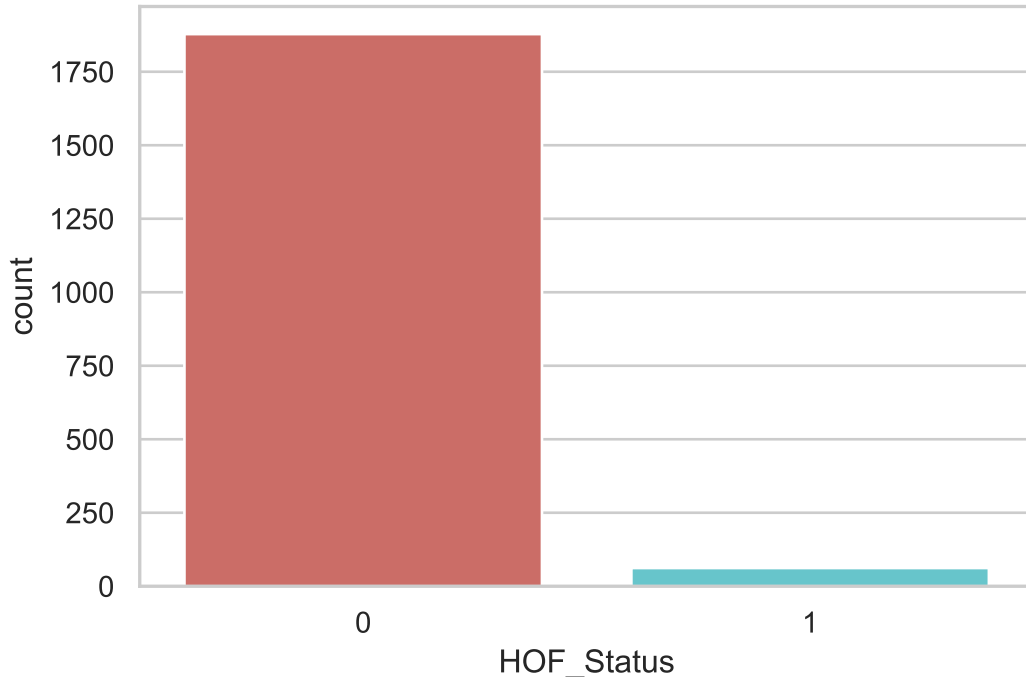
In [231]:

```
#Count how many pitchers in the HOF
full_df['HOF_Status'].value_counts()
```

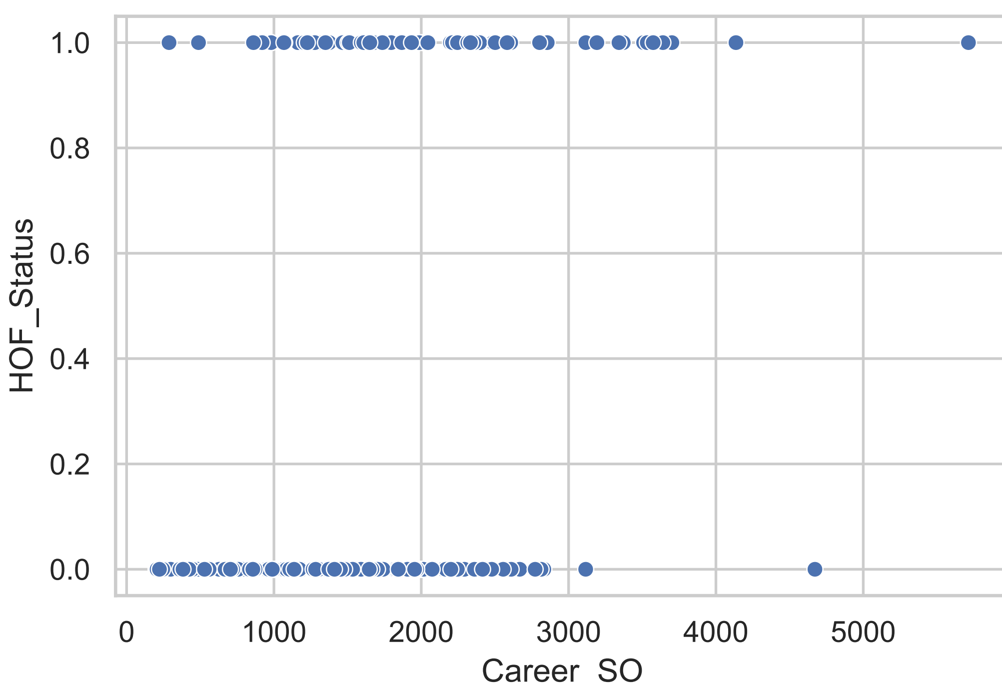
Out[231]:

```
0    1878
1      62
Name: HOF_Status, dtype: int64
```

In [232]:
```
#Plot HOF players count and non-HOF players count
sns.countplot(x='HOF_Status', data = full_df, palette='hls')
plt.show()
```



In [233]:
```
#Scatterplot of data
sns.scatterplot(x="Career_SO", y = "HOF_Status", data = full_df)
plt.show()
```



In [234]:

```
#Mean values of each group
full_df.groupby('HOF_Status').mean()['Career_SO']
```

Out[234]:

```
HOF_Status
0     630.900958
1    2090.032258
Name: Career_SO, dtype: float64
```

In [235]:

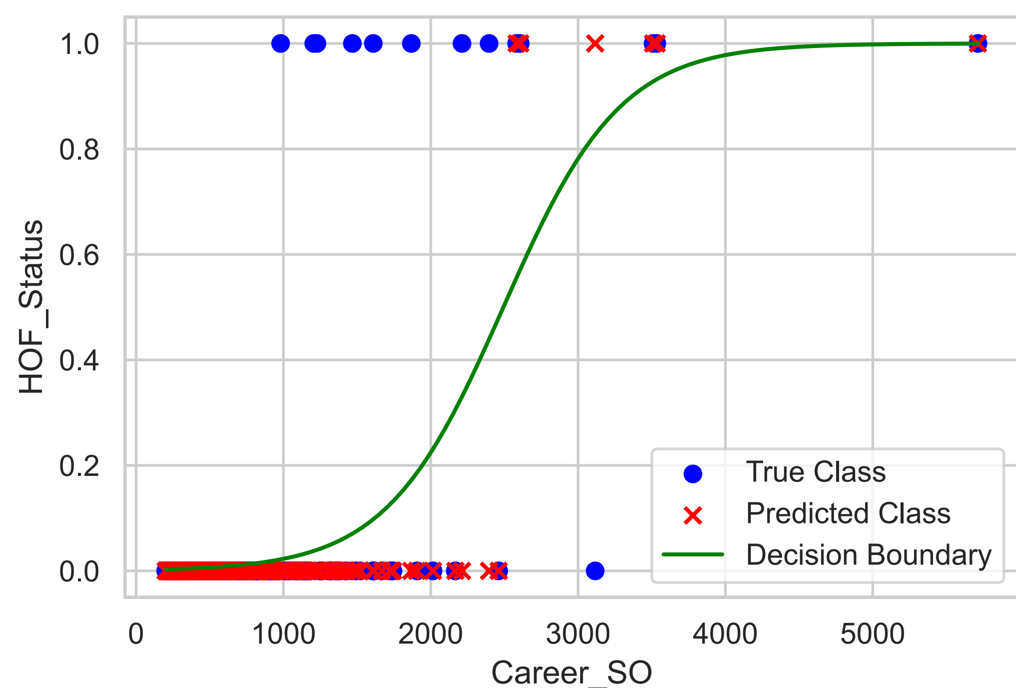
```
#Run the logistic regression
X = full_df[['Career_SO']]
y = full_df['HOF_Status']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.98
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       375
           1       0.83      0.38      0.53        13

    accuracy                           0.98       388
   macro avg       0.91      0.69      0.76       388
weighted avg       0.97      0.98      0.97       388
```

In [236]:

```python
# Plot the data points and decision boundary
plt.scatter(X_test, y_test, color='blue', label='True Class')
plt.scatter(X_test, y_pred, color='red', marker='x', label='Predicted C
plt.xlabel('Career_SO')
plt.ylabel('HOF_Status')
plt.legend()

# Create a range of values for the x-axis
x_range = np.linspace(X.min(), X.max(), num=100)
# Calculate the corresponding y-values using the logistic regression mo
y_range = model.predict_proba(x_range.reshape(-1, 1))[:, 1]

# Plot the decision boundary
plt.plot(x_range, y_range, color='green', label='Decision Boundary')
plt.legend()

plt.show()
```

```
/Users/charliemorris/opt/anaconda3/lib/python3.9/site-packages/sklearn/ba
se.py:450: UserWarning: X does not have valid feature names, but Logistic
Regression was fitted with feature names
  warnings.warn(
```



Logistic Regression: Probability of Making HOF Vesus Career Batting Average

In [237]:
```
#Grab the data for batters and their career batting averages
#Look only at batters with minimum 1500 at bats

query = """
    SELECT playerID, ROUND(SUM(H)/SUM(AB) * 1000) AS Career_BA
    FROM batting_stats
    WHERE playerID IN
        (SELECT playerID
        FROM players
        WHERE '1930-12-31' < finalGame AND finalGame < '2008-12-31')
    GROUP BY playerID
    HAVING SUM(AB) > 2000;
"""

df_CareerBA = pd.read_sql(query, engine)
df_CareerBA
```

Out[237]:

	playerID	Career_BA
0	aaronha01	305.0
1	abbotku01	256.0
2	adairje01	254.0
3	adamsbo03	269.0
4	adamsbu01	266.0
...	...	...
1448	zarilal01	276.0
1449	zeileto01	265.0
1450	zernigu01	265.0
1451	zimmedo01	235.0
1452	ziskri01	287.0

1453 rows × 2 columns

In [238]:

```
#Grab data on HOF players
query = """
    SELECT *
    FROM hall_of_fame;
"""

df_HOF = pd.read_sql(query, engine)
df_HOF
```

Out[238]:

	playerID	year
0	aaronha01	1982
1	alexape01	1938
2	alomaro01	2011
3	alstowa01	1983
4	andersp01	2000
...	...	...
282	wynnea01	1972
283	yastrca01	1989
284	youngcy01	1937
285	youngro01	1972
286	yountro01	1999

287 rows × 2 columns

In [239]:
```
#Merge the 2 data frames
#1 means made HOF, 0 means didn't make HOF
full_df = pd.merge(df_CareerBA, df_HOF, on = "playerID", how = "left")
full_df['HOF_Status'] = full_df['year'].notnull().astype(int)
full_df
```

Out[239]:

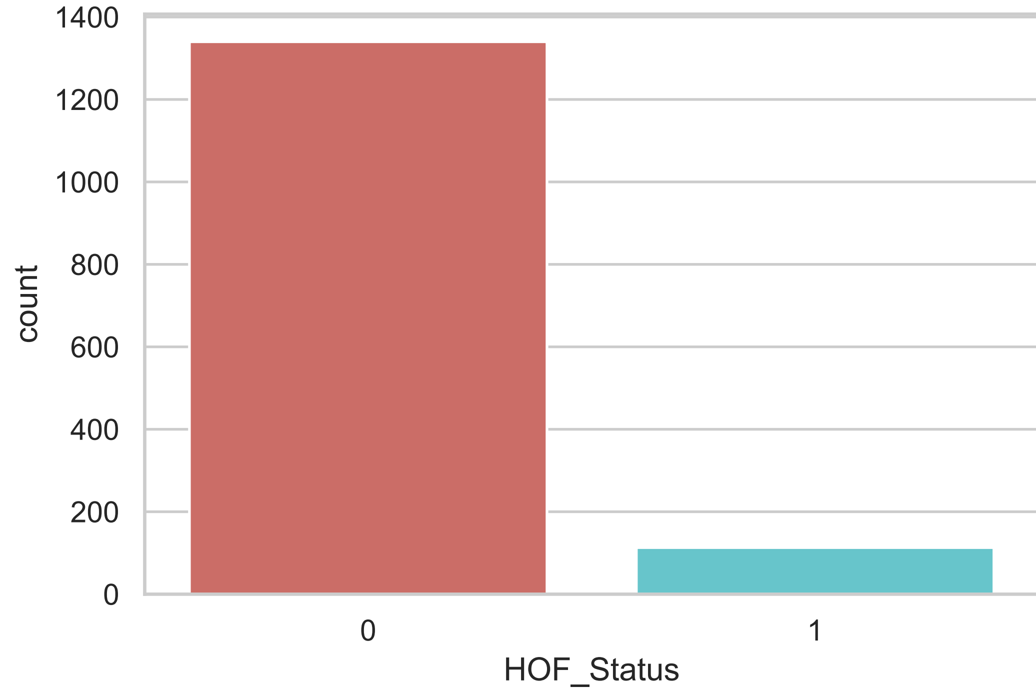
	playerID	Career_BA	year	HOF_Status
0	aaronha01	305.0	1982.0	1
1	abbotku01	256.0	NaN	0
2	adairje01	254.0	NaN	0
3	adamsbo03	269.0	NaN	0
4	adamsbu01	266.0	NaN	0
...	...	...	...	...
1448	zarilal01	276.0	NaN	0
1449	zeileto01	265.0	NaN	0
1450	zernigu01	265.0	NaN	0
1451	zimmedo01	235.0	NaN	0
1452	ziskri01	287.0	NaN	0

1453 rows × 4 columns

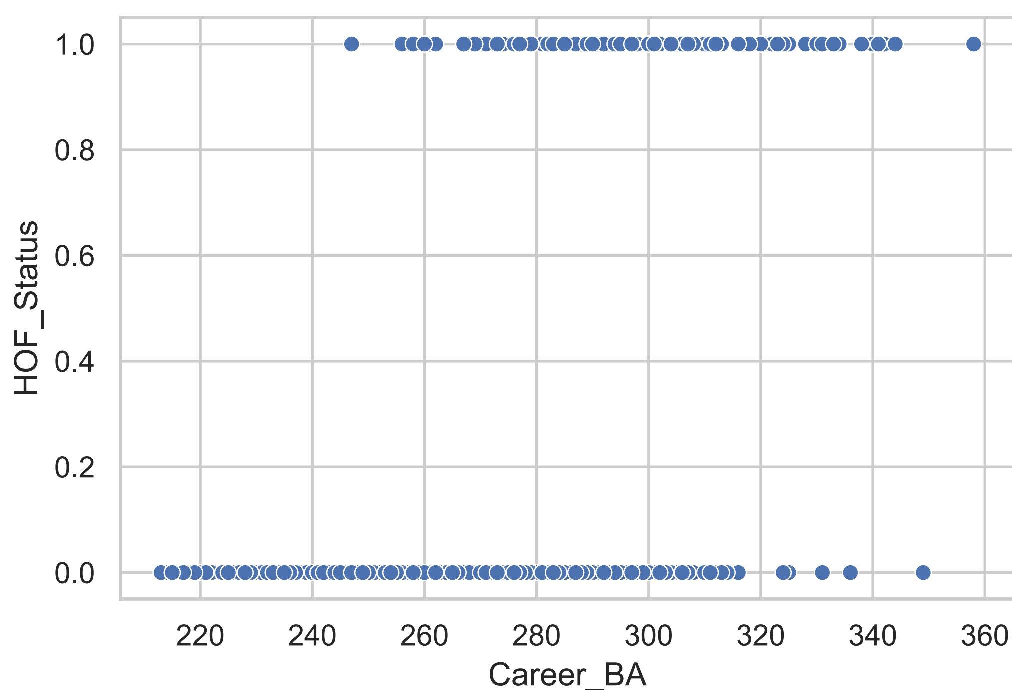
In [240]:
```
#Count how many batters in the HOF
full_df['HOF_Status'].value_counts()
```

Out[240]:
```
0    1340
1     113
Name: HOF_Status, dtype: int64
```

In [241]:
```
#Plot HOF players count and non-HOF players count
sns.countplot(x='HOF_Status', data = full_df, palette='hls')
plt.show()
```



In [242]:
```
#Scatterplot of data
sns.scatterplot(x="Career_BA", y = "HOF_Status", data = full_df)
plt.show()
```



In [243]:
```
#Mean values of each group
full_df.groupby('HOF_Status').mean()['Career_BA']
```

Out[243]:
```
HOF_Status
0    266.487313
1    298.415929
Name: Career_BA, dtype: float64
```

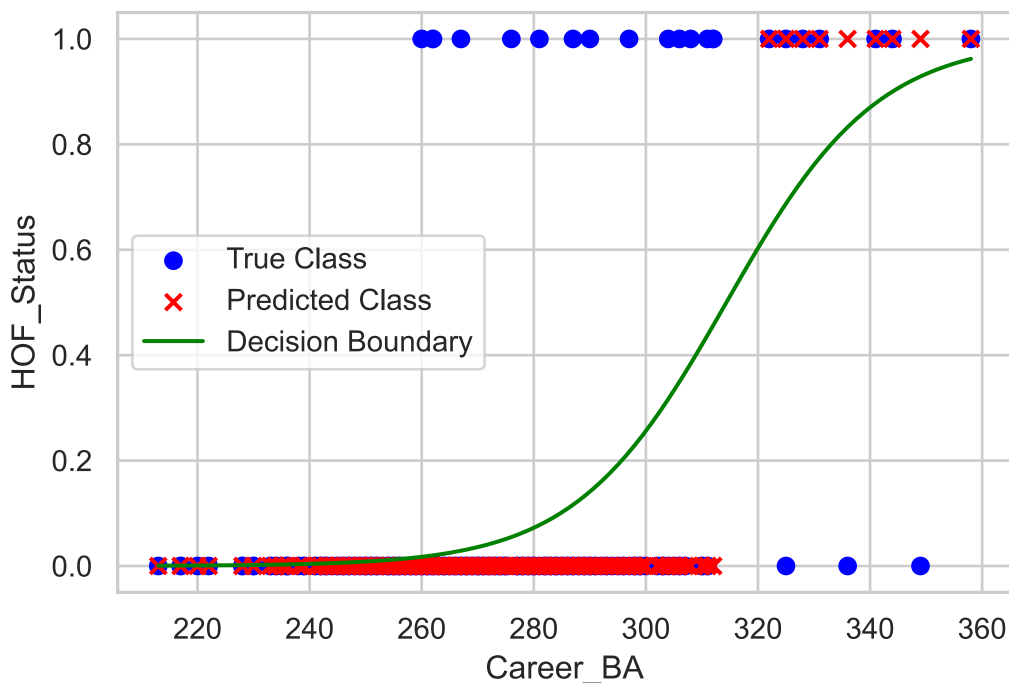
In [244]:
```
#Run the logistic regression
X = full_df[['Career_BA']]
y = full_df['HOF_Status']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2

# Initialize the logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model's accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.94
              precision    recall  f1-score   support

           0       0.95      0.99      0.97       269
           1       0.70      0.32      0.44        22

    accuracy                           0.94       291
   macro avg       0.82      0.65      0.70       291
weighted avg       0.93      0.94      0.93       291
```

In [245]:

```
# Plot the data points and decision boundary
plt.scatter(X_test, y_test, color='blue', label='True Class')
plt.scatter(X_test, y_pred, color='red', marker='x', label='Predicted C
plt.xlabel('Career_BA')
plt.ylabel('HOF_Status')
plt.legend()

# Create a range of values for the x-axis
x_range = np.linspace(X.min(), X.max(), num=100)
# Calculate the corresponding y-values using the logistic regression mo
y_range = model.predict_proba(x_range.reshape(-1, 1))[:, 1]

# Plot the decision boundary
plt.plot(x_range, y_range, color='green', label='Decision Boundary')
plt.legend()

plt.show()
```

```
/Users/charliemorris/opt/anaconda3/lib/python3.9/site-packages/sklearn/ba
se.py:450: UserWarning: X does not have valid feature names, but Logistic
Regression was fitted with feature names
  warnings.warn(
```



Prime Age for Baseball

In [246]:

```
#Look at the baseball players with career HR totals of at least 275
#Each row is their age during a season and their HR totals
query = """
    SELECT year - birthYear AS Age, HR
    FROM players AS p
    INNER JOIN batting_stats AS bs
    ON p.playerID = bs.playerID
    WHERE p.playerID IN
        (SELECT playerID
        FROM batting_stats
        GROUP BY playerID
        HAVING SUM(HR) > 275);
"""

df = pd.read_sql(query, engine)
df
```

Out[246]:

	Age	HR
0	20	13
1	21	27
2	22	26
3	23	44
4	24	30
...	...	...
3078	32	18
3079	33	17
3080	34	6
3081	34	15
3082	35	0

3083 rows × 2 columns

In [247]:
```
counts = df['Age'].value_counts().sort_index()
counts_df = counts.reset_index()
counts_df.columns = ['Age', 'Count']
counts_df
```

Out[247]:

	Age	Count
0	17	1
1	18	3
2	19	12
3	20	28
4	21	72
5	22	108
6	23	143
7	24	174
8	25	167
9	26	166
10	27	172
11	28	174
12	29	172
13	30	174
14	31	173
15	32	174
16	33	165
17	34	178
18	35	175
19	36	160
20	37	133
21	38	117
22	39	82
23	40	69
24	41	43
25	42	29
26	43	10
27	44	6
28	45	2
29	46	1

In [248]:
```
medianHRs = df.groupby('Age').median()["HR"]
medianHRs_df = medianHRs.reset_index()
medianHRs_df.columns = ['Age', 'MedianHRs']
medianHRs_df
```

Out[248]:

	Age	MedianHRs
0	17	0.0
1	18	0.0
2	19	0.5
3	20	4.5
4	21	3.0
5	22	10.0
6	23	16.0
7	24	18.0
8	25	23.0
9	26	26.0
10	27	27.5
11	28	28.5
12	29	29.0
13	30	26.0
14	31	28.0
15	32	26.0
16	33	23.0
17	34	21.0
18	35	19.0
19	36	18.0
20	37	16.0
21	38	11.0
22	39	14.5
23	40	12.0
24	41	9.0
25	42	4.0
26	43	9.0
27	44	3.5
28	45	2.5
29	46	1.0

In [249]:
```
primeHRs = pd.merge(counts_df, medianHRs_df, on = "Age", how = "inner")
primeHRs
```

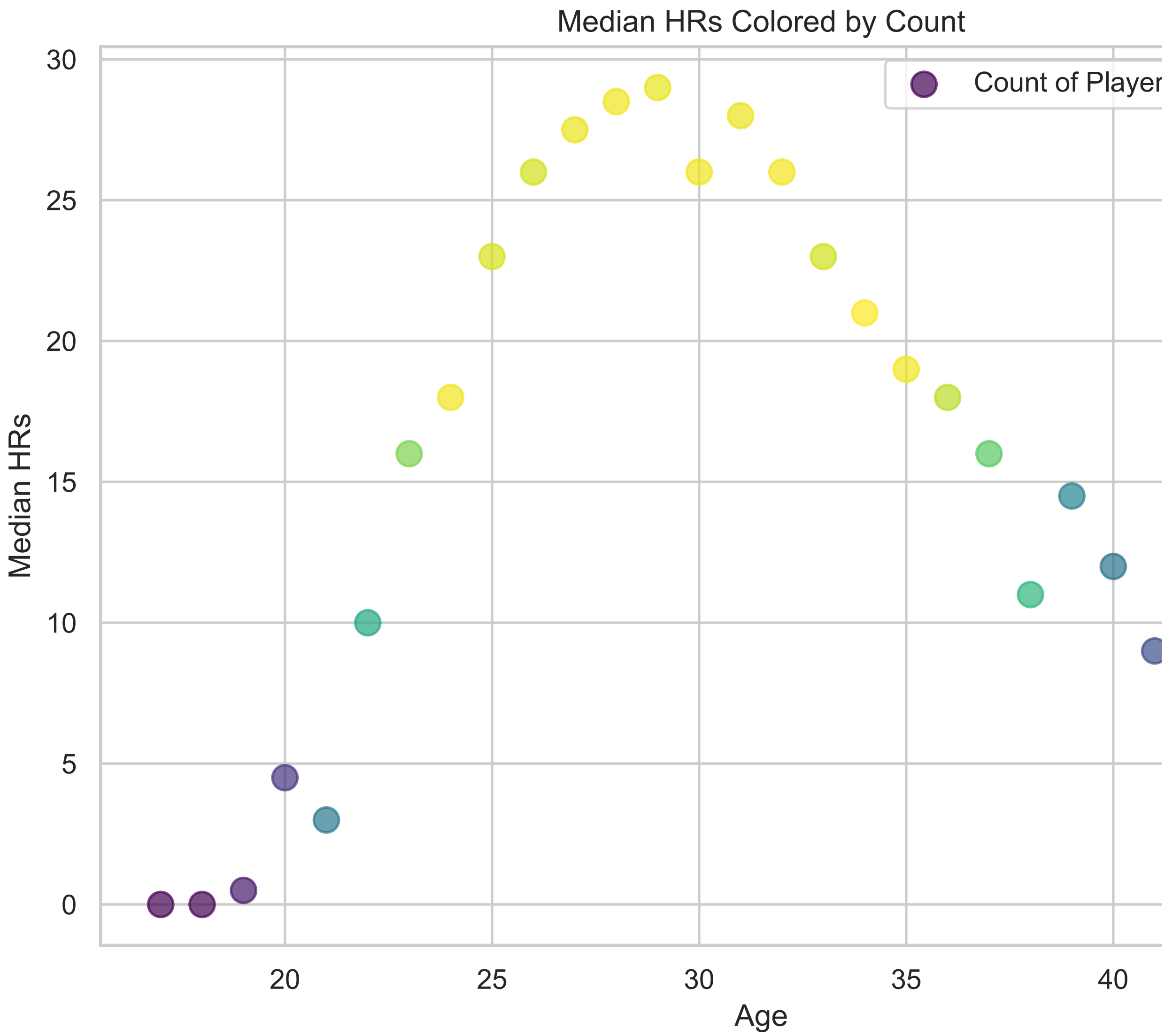
Out[249]:

	Age	Count	MedianHRs
0	17	1	0.0
1	18	3	0.0
2	19	12	0.5
3	20	28	4.5
4	21	72	3.0
5	22	108	10.0
6	23	143	16.0
7	24	174	18.0
8	25	167	23.0
9	26	166	26.0
10	27	172	27.5
11	28	174	28.5
12	29	172	29.0
13	30	174	26.0
14	31	173	28.0
15	32	174	26.0
16	33	165	23.0
17	34	178	21.0
18	35	175	19.0
19	36	160	18.0
20	37	133	16.0
21	38	117	11.0
22	39	82	14.5
23	40	69	12.0
24	41	43	9.0
25	42	29	4.0
26	43	10	9.0
27	44	6	3.5
28	45	2	2.5
29	46	1	1.0

In [250]:

```
# Set up the figure and axes for plotting
plt.figure(figsize=(10, 6))

# Create a scatter plot for median HRs with varying colors based on cou
scatter = plt.scatter(
    x=primeHRs['Age'],
    y=primeHRs['MedianHRs'],
    c=primeHRs['Count'],  # Use count for coloring
    cmap='viridis',  # Choose a colormap
    s=100,  # Marker size
    alpha=0.7,  # Transparency
    label='Count of Players in the Calculation'
)

# Add colorbar
cbar = plt.colorbar(scatter)
cbar.set_label('Count')

# Add labels and title
plt.xlabel('Age')
plt.ylabel('Median HRs')
plt.title('Median HRs Colored by Count')
plt.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

```
/var/folders/cj/jcn0rhn52g397xd7gy7ndwnc0000gn/T/ipykernel_51952/14756100
78.py:16: MatplotlibDeprecationWarning: Auto-removal of grids by pcolor()
and pcolormesh() is deprecated since 3.5 and will be removed two minor re
leases later; please call grid(False) first.
  cbar = plt.colorbar(scatter)
```


Median HRs Colored by Count
Count of Player
30
25
20
15
10
5
0
Median HRs
20
25
30
35
40
Age

Linear Regression on Team Winning

In [251]:
```
ecting predictors for linear regression
e that I scale stats based on games played
y = """
SELECT W * (G/162) AS "Wins", HR * (G/162) AS "HRs", BB * (G/162) AS "BBs",
FROM team_stats
WHERE G >= 155;


pd.read_sql(query, engine)

```

Out[251]:

	Wins	HRs	BBs	Es	BA	SBs	ERA
0	84.0000	161.0000	617.0000	126.0000	272.0327	126.0000	4.52
1	85.0000	147.0000	510.0000	106.0000	271.7584	93.0000	4.49
2	70.0000	158.0000	511.0000	106.0000	255.5515	71.0000	4.79
3	82.0000	236.0000	608.0000	134.0000	279.6731	93.0000	5.00
4	75.0000	158.0000	494.0000	103.0000	260.6738	116.0000	4.20
...	...	...	...	...	...	...	...
1733	86.0000	148.0000	630.0000	140.0000	250.5967	52.0000	3.49
1734	70.0000	138.0000	635.0000	116.0000	238.4615	72.0000	3.80
1735	61.8333	84.4074	564.3519	138.3889	230.4348	66.7407	3.70
1736	48.7963	34.4444	354.0123	422.9012	270.6867	188.4877	4.52
1737	51.6667	44.9691	334.8765	385.5864	271.8798	168.3951	4.93

1738 rows × 7 columns

In [252]:
```
#Linear regression with 4 features
data = df[['BBs','Es', 'BA', 'ERA']]
x = data.to_numpy() # convert to numpy array
X = sm.add_constant(x) # add a column of all 1s
y = df[["Wins"]].to_numpy()
model = sm.OLS(y,X) #run OLS
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:                      y   R-squared:
0.741
Model:                            OLS   Adj. R-squared:
0.740
Method:                 Least Squares   F-statistic:
1237.
Date:                Wed, 23 Aug 2023   Prob (F-statistic):
0.00
Time:                        20:05:10   Log-Likelihood:                    -5
645.3
No. Observations:                1738   AIC:                             1.13
0e+04
Df Residuals:                    1733   BIC:                             1.13
3e+04
Df Model:                           4
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const         10.8395      2.928      3.702      0.000       5.097       1
6.582
x1             0.0501      0.002     24.629      0.000       0.046
0.054
x2            -0.1097      0.004    -30.430      0.000      -0.117       -
0.103
x3             0.4414      0.011     38.548      0.000       0.419
0.464
x4           -14.5919      0.258    -56.631      0.000     -15.097      -1
4.086
==========================================================================
=====
Omnibus:                       47.057   Durbin-Watson:
1.343
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                   8
8.628
Skew:                           0.183   Prob(JB):                         5.6
8e-20
Kurtosis:                       4.044   Cond. No.                         1.1
7e+04
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.17e+04. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [253]:

```
#Linear regression with 6 features
data = df[['HRs','SBs','ERA']]
x = data.to_numpy() # convert to numpy array
X = sm.add_constant(x) # add a column of all 1s
y = df[["Wins"]].to_numpy()
model = sm.OLS(y,X) #run OLS
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:                      y   R-squared:
0.576
Model:                            OLS   Adj. R-squared:
0.575
Method:                 Least Squares   F-statistic:
785.6
Date:                Wed, 23 Aug 2023   Prob (F-statistic):          1.63
e-322
Time:                        20:05:10   Log-Likelihood:                -6
072.2
No. Observations:                1738   AIC:                         1.21
5e+04
Df Residuals:                    1734   BIC:                         1.21
7e+04
Df Model:                           3
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const        108.0434      1.355     79.751      0.000     105.386      11
0.701
x1             0.1686      0.004     39.649      0.000       0.160
0.177
x2             0.0371      0.004      8.912      0.000       0.029
0.045
x3           -13.7936      0.338    -40.848      0.000     -14.456      -1
3.131
==========================================================================
=====
Omnibus:                        0.746   Durbin-Watson:
1.216
Prob(Omnibus):                  0.689   Jarque-Bera (JB):
0.704
Skew:                          -0.048   Prob(JB):
0.703
Kurtosis:                       3.021   Cond. No.                      1.2
2e+03
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.22e+03. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

Autoregression on Wins

In [254]:
```
#Scale wins for prev year and curr year to 162
query = """
    SELECT t1.W * (t1.G / 162) AS "PrevWins", t2.W * (t2.G / 162) AS "C
    FROM team_stats AS t1
    INNER JOIN team_stats AS t2
    ON t1.teamID = t2.teamID AND t1.year = t2.year - 1
    WHERE t1.G >= 155 AND t2.G >= 155;
"""

df = pd.read_sql(query, engine)
df
```

Out[254]:

	PrevWins	CurrWins
0	84.0000	85.0000
1	85.0000	70.0000
2	70.0000	82.0000
3	82.0000	75.0000
4	75.0000	99.0000
...	...	...
1434	75.5309	64.5988
1435	64.5988	86.0000
1436	86.0000	70.0000
1437	70.0000	61.8333
1438	48.7963	51.6667

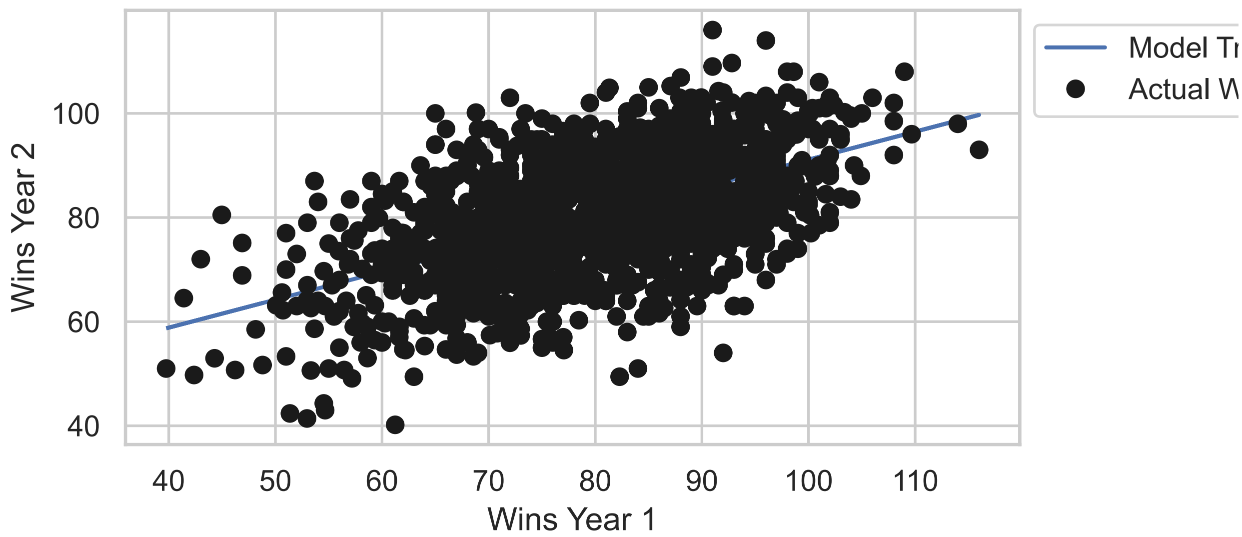
1439 rows × 2 columns

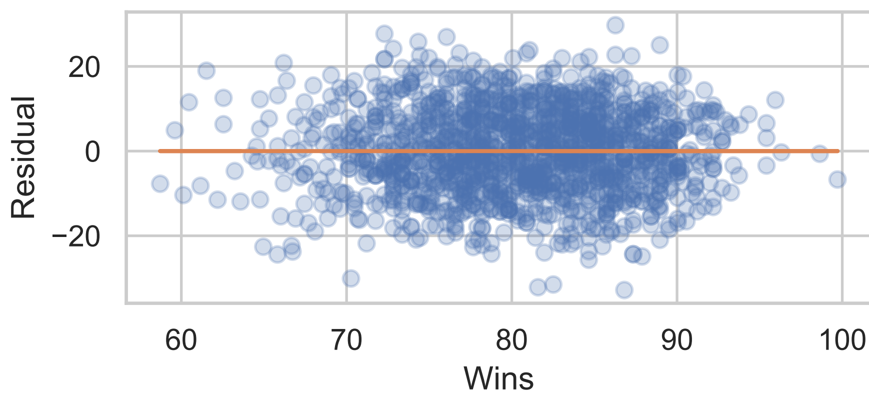
In [255]:

```
#Run linear regression
x = df["PrevWins"].to_numpy()
y = df["CurrWins"].to_numpy()
X = sm.add_constant(x) # add a column of all 1s
model = sm.OLS(y,X) #run OLS
results = model.fit()
print(results.summary())
bhat, ahat = results.params #Grab values
sigma_eps_hat = np.sqrt(results.mse_resid)

# Graph it
fig,ax = plt.subplots(figsize=(6,3))
ax.plot(x,ahat*x + bhat, '-', label="Model Trendline") # graph line of
ax.plot(x,y,"ko",  label="Actual Wins")
ax.set_xlabel("Wins Year 1")
ax.set_ylabel("Wins Year 2")
ax.legend(loc="upper left", bbox_to_anchor=(1, 1))

#Graph residuals
r = results.resid
y_mean_pred = y - r
fig,ax = plt.subplots(figsize=(5,2))
ax.plot(y_mean_pred,r,"o", alpha = 0.25)  # Plot the residuals
ax.plot(y_mean_pred,np.zeros(len(y)),"-") # Plot the reference line of
ax.set_xlabel("Wins")
ax.set_ylabel("Residual")
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:                      y   R-squared:
0.293
Model:                            OLS   Adj. R-squared:
0.292
Method:                 Least Squares   F-statistic:
594.5
Date:                Wed, 23 Aug 2023   Prob (F-statistic):           3.60
e-110
Time:                        20:05:10   Log-Likelihood:                 -5
346.5
No. Observations:                1439   AIC:                          1.07
0e+04
Df Residuals:                    1437   BIC:                          1.07
1e+04
Df Model:                           1
Covariance Type:            nonrobust
==========================================================================
=====
                 coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------
-----
const         37.3423      1.788     20.882      0.000      33.834       4
0.850
x1             0.5377      0.022     24.382      0.000       0.494
0.581
==========================================================================
=====
Omnibus:                        6.742   Durbin-Watson:
2.130
Prob(Omnibus):                  0.034   Jarque-Bera (JB):
5.738
Skew:                          -0.082   Prob(JB):
0.0567
Kurtosis:                       2.737   Cond. No.
553.
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
```

Out[255]:
```
Text(0, 0.5, 'Residual')
```


Model Tr
Actual W
Wins Year 2
Wins Year 1
100
80
60
40
40
50
60
70
80
90
100
110


Residual
Wins
20
0
−20
60
70
80
90
100