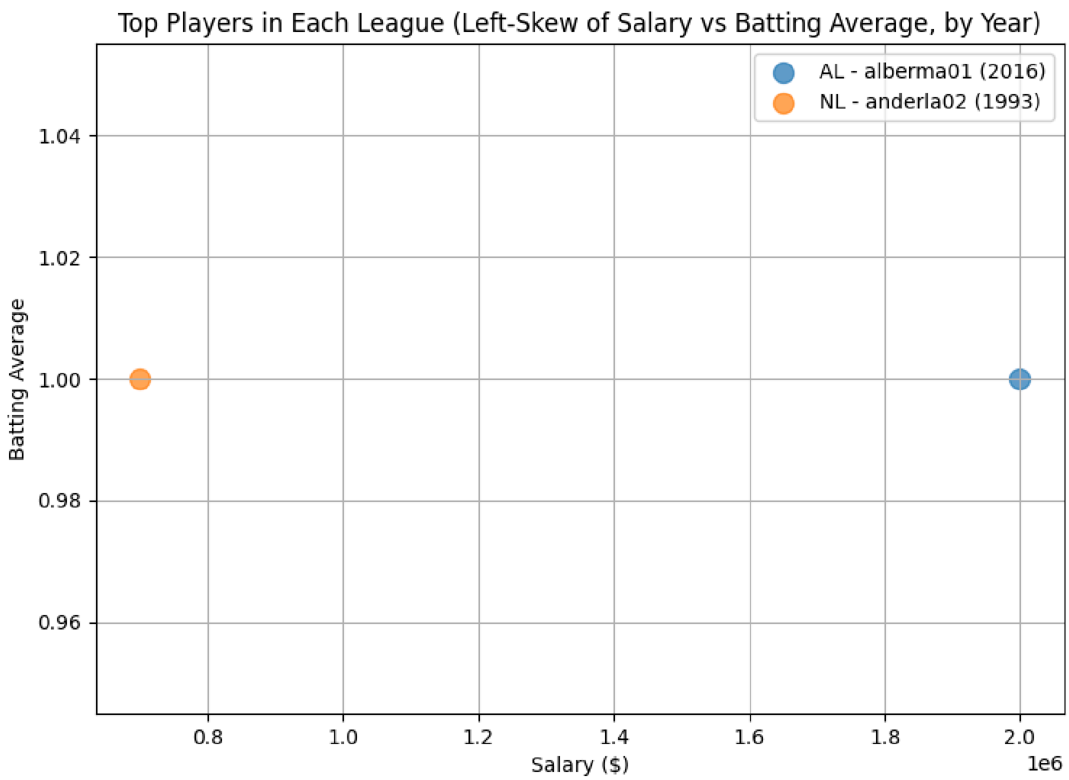
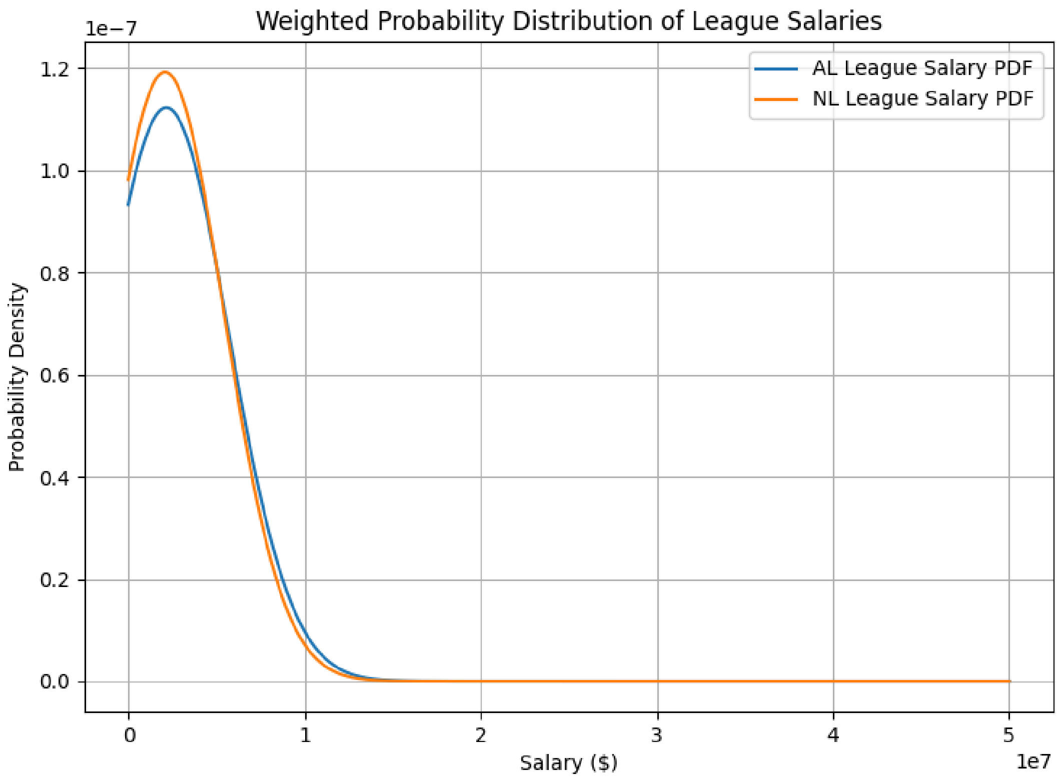
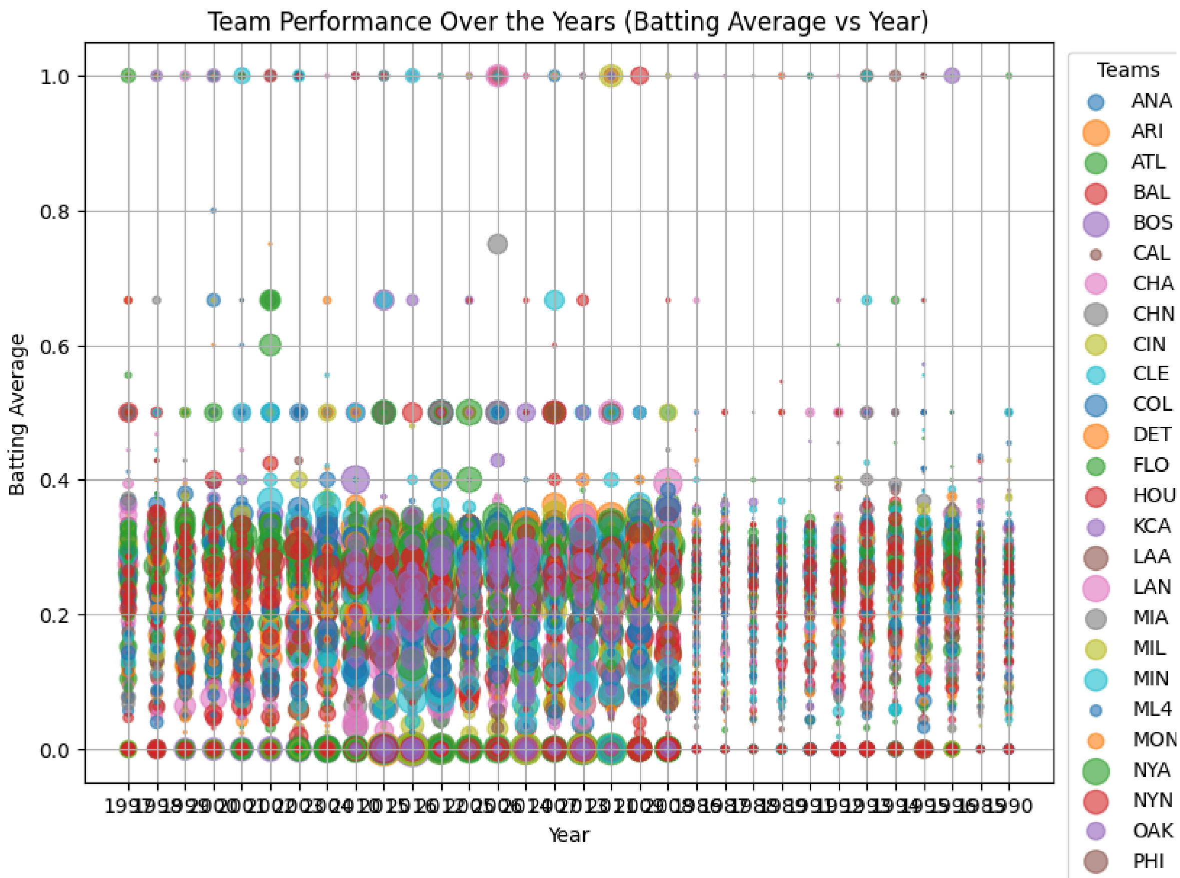
In Class discussion: *Code who's the highest paid player of the data sheet based on the factors of league total money within the league and their highest paid players of all time based on the player's batting average through their career.*





Weighted Probability Distribution of League Salaries
1e−7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Probability Density
AL League Salary PDF
NL League Salary PDF
0
1
2
3
4
5
1e7
Salary ($)




```
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
##Coversation
#https://chatgpt.com/share/66ee2d9d-91c8-8002-a05d-5d9f2d1f7422
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Load Batting and Salaries data using csv module
batting_data = []
salaries_data = []

with open('Batting - Batting.csv.csv', newline='') as batting_file:
    reader = csv.DictReader(batting_file)
```

```
    for row in reader:
        batting_data.append(row)

with open('Salaries - Salaries.csv', newline='') as salaries_file:
    reader = csv.DictReader(salaries_file)
    for row in reader:
        salaries_data.append(row)

# Convert to numpy arrays for easier manipulation
batting_data = np.array(batting_data)
salaries_data = np.array(salaries_data)

# Extract relevant data (playerID, batting average, salary, lgID as
league, and yearID as year)
batting_player_ids = np.array([row['playerID'] for row in batting_data])
salary_player_ids = np.array([row['playerID'] for row in salaries_data])

batting_years = np.array([row['yearID'] for row in batting_data])
salary_years = np.array([row['yearID'] for row in salaries_data])

# Ensure alignment by matching playerID and yearID
aligned_indices = []
for i, (player, year) in enumerate(zip(batting_player_ids,
batting_years)):
    match_idx = np.where((salary_player_ids == player) & (salary_years ==
year))[0]
    if len(match_idx) > 0:
        aligned_indices.append((i, match_idx[0]))

# Filter the data based on aligned indices
batting_player_ids = np.array([batting_player_ids[i] for i, j in
aligned_indices])
batting_avg = np.array([float(batting_data[i]['H']) /
float(batting_data[i]['AB']) if float(batting_data[i]['AB']) > 0 else 0
for i, j in aligned_indices])
salary = np.array([float(salaries_data[j]['salary']) for i, j in
aligned_indices])
league = np.array([batting_data[i]['lgID'] for i, j in aligned_indices])
year = np.array([batting_data[i]['yearID'] for i, j in aligned_indices])
team = np.array([batting_data[i]['teamID'] for i, j in aligned_indices])
```

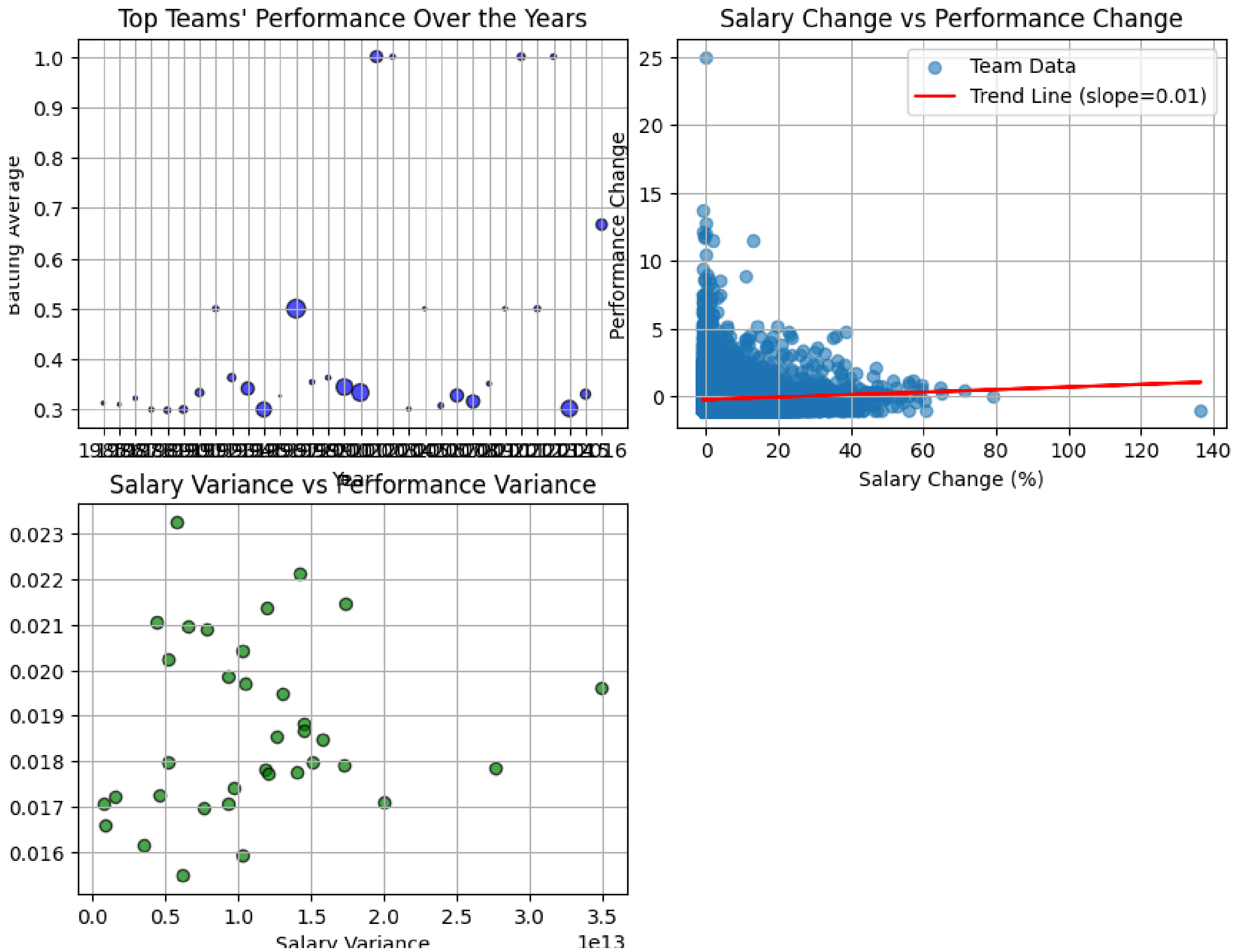
```
# Step 1: Perform two-way ANOVA using league and batting average vs salary
# Organize data for ANOVA
anova_data = []
for lg in np.unique(league):
    for i, l in enumerate(league):
        if l == lg:
            anova_data.append([salary[i], batting_avg[i], lg])

anova_data = np.array(anova_data)

# Prepare groups for two-way ANOVA
salary_groups = []
batting_avg_groups = []

for lg in np.unique(league):
    salary_groups.append(anova_data[anova_data[:, 2] == lg][:,
0].astype(float))
    batting_avg_groups.append(anova_data[anova_data[:, 2] == lg][:,
1].astype(float))

# Perform two-way ANOVA on salary and batting average
f_salary, p_salary = stats.f_oneway(*salary_groups)
f_batting_avg, p_batting_avg = stats.f_oneway(*batting_avg_groups)

print(f"ANOVA Results: Salary F-value = {f_salary}, p-value = {p_salary}")
print(f"ANOVA Results: Batting Avg F-value = {f_batting_avg}, p-value =
{p_batting_avg}")

# Step 2: Calculate Weighted PDF for League Salary Distribution
unique_leagues = np.unique(league)
league_mean_salaries = {lg: np.mean([salary[i] for i, l in
enumerate(league) if l == lg]) for lg in unique_leagues}
league_std_salaries = {lg: np.std([salary[i] for i, l in enumerate(league)
if l == lg]) for lg in unique_leagues}

# Simulate a PDF for the leagues' salary distribution based on batting
average and salary
x = np.linspace(0, 50000000, 1000)  # Simulated salary range
pdfs = {}
```

```
for lg in unique_leagues:
    mean_salary = league_mean_salaries[lg]
    std_salary = league_std_salaries[lg]

    # PDF centered around the league mean salary
    pdfs[lg] = (1 / (std_salary * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x
- mean_salary) / std_salary) ** 2)

# Plot the PDFs for each league
plt.figure(figsize=(12, 6))
for lg in unique_leagues:
    plt.plot(x, pdfs[lg], label=f'{lg} League Salary PDF')

plt.title('Weighted Probability Distribution of League Salaries')
plt.xlabel('Salary ($)')
plt.ylabel('Probability Density')
plt.legend()
plt.grid(True)
plt.show()

# Step 3: Compare the Left-Skew of Top Players from Each League Based on
Batting Average
# Get the top players based on batting average for each league and the
earliest year
top_players_by_league = {}
for lg in unique_leagues:
    league_players = [(batting_player_ids[i], batting_avg[i], salary[i],
year[i]) for i, l in enumerate(league) if l == lg]
    top_player = max(league_players, key=lambda x: x[1])  # Top player by
batting average
    top_players_by_league[lg] = top_player

# Left-skewed plot for the top players based on batting average and salary
plt.figure(figsize=(10, 6))
for lg, player in top_players_by_league.items():
    plt.scatter(player[2], player[1], label=f'{lg} - {player[0]}
({player[3]})', s=100, alpha=0.7)
```

```python
plt.title('Top Players in Each League (Left-Skew of Salary vs Batting
Average, by Year)')
plt.xlabel('Salary ($)')
plt.ylabel('Batting Average')
plt.legend()
plt.grid(True)
plt.show()

# Step 4: Scatter Plot for Each Team's Performance by Year
plt.figure(figsize=(12, 8))

# Unique teams for color differentiation
unique_teams = np.unique(team)

# Scatter plot by team, year, and performance
for t in unique_teams:
    team_indices = np.where(team == t)
    plt.scatter(year[team_indices], batting_avg[team_indices],
s=salary[team_indices] / 100000, label=t, alpha=0.6)

plt.title('Team Performance Over the Years (Batting Average vs Year)')
plt.xlabel('Year')
plt.ylabel('Batting Average')
plt.legend(loc='upper right', bbox_to_anchor=(1.15, 1.0), title='Teams')
plt.grid(True)
plt.show()
#Samuel-Akira Masters
```

ANOVA Results: Salary F-value = 3.090000549746806, p-value = 0.07878566153432548
ANOVA Results: Batting Avg F-value = 373.221050785537, p-value = 1.2791132962946878e-82

Based on the batting average P value more investigation is needed to see if the teams change in salary causes a better player performance the following year. Show best fit trend following the highest yielding number of top player of a team throughout the years




```python
import csv
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
from matplotlib import gridspec

# Load Batting and Salaries data using csv module
#https://chatgpt.com/share/66ee2d9d-91c8-8002-a05d-5d9f2d1f7422
batting_data = []
salaries_data = []

with open('Batting - Batting.csv.csv', newline='') as batting_file:
    reader = csv.DictReader(batting_file)
    for row in reader:
        batting_data.append(row)

with open('Salaries - Salaries.csv', newline='') as salaries_file:
```

```python
    reader = csv.DictReader(salaries_file)
    for row in reader:
        salaries_data.append(row)

# Convert to numpy arrays for easier manipulation
batting_data = np.array(batting_data)
salaries_data = np.array(salaries_data)

# Extract relevant data (playerID, batting average, salary, lgID as
league, and yearID as year)
batting_player_ids = np.array([row['playerID'] for row in batting_data])
salary_player_ids = np.array([row['playerID'] for row in salaries_data])

batting_years = np.array([row['yearID'] for row in batting_data])
salary_years = np.array([row['yearID'] for row in salaries_data])

# Ensure alignment by matching playerID and yearID
aligned_indices = []
for i, (player, year) in enumerate(zip(batting_player_ids,
batting_years)):
    match_idx = np.where((salary_player_ids == player) & (salary_years ==
year))[0]
    if len(match_idx) > 0:
        aligned_indices.append((i, match_idx[0]))

# Filter the data based on aligned indices
batting_player_ids = np.array([batting_player_ids[i] for i, j in
aligned_indices])
batting_avg = np.array([float(batting_data[i]['H']) /
float(batting_data[i]['AB']) if float(batting_data[i]['AB']) > 0 else 0
for i, j in aligned_indices])
salary = np.array([float(salaries_data[j]['salary']) for i, j in
aligned_indices])
league = np.array([batting_data[i]['lgID'] for i, j in aligned_indices])
year = np.array([batting_data[i]['yearID'] for i, j in aligned_indices])
team = np.array([batting_data[i]['teamID'] for i, j in aligned_indices])

# Step 1: Calculate Year-Over-Year Salary Change for Each Team
unique_teams = np.unique(team)
team_salary_changes = []
```

```
team_performance_changes = []

for t in unique_teams:
    team_indices = np.where(team == t)
    team_years = year[team_indices].astype(int)
    team_salaries = salary[team_indices]
    team_batting_avg = batting_avg[team_indices]

    # Sort data by year to calculate year-over-year change
    sorted_indices = np.argsort(team_years)
    team_years_sorted = team_years[sorted_indices]
    team_salaries_sorted = team_salaries[sorted_indices]
    team_batting_avg_sorted = team_batting_avg[sorted_indices]

    # Filter out zero values (to avoid division by zero) AFTER sorting
    nonzero_salary_indices = team_salaries_sorted[:-1] != 0
    nonzero_batting_avg_indices = team_batting_avg_sorted[:-1] != 0

    # Align indices where both salary and batting average are non-zero
    valid_indices = nonzero_salary_indices & nonzero_batting_avg_indices

    # Calculate salary change and performance change only for valid
(non-zero) entries
    salary_change = np.diff(team_salaries_sorted)[valid_indices] /
team_salaries_sorted[:-1][valid_indices]
    performance_change = np.diff(team_batting_avg_sorted)[valid_indices] /
team_batting_avg_sorted[:-1][valid_indices]

    # Avoid NaN or infinite values
    salary_change = np.nan_to_num(salary_change, nan=0, posinf=0,
neginf=0)
    performance_change = np.nan_to_num(performance_change, nan=0,
posinf=0, neginf=0)

    team_salary_changes.append(salary_change)
    team_performance_changes.append(performance_change)

# Flatten the salary and performance changes
salary_changes_flat = np.concatenate(team_salary_changes)
performance_changes_flat = np.concatenate(team_performance_changes)
```

```
# Ensure both arrays have the same length by filtering out invalid entries
valid_indices = (salary_changes_flat != 0) & (performance_changes_flat !=
0)
salary_changes_flat = salary_changes_flat[valid_indices]
performance_changes_flat = performance_changes_flat[valid_indices]

# Step 2: Fit a Best-Fit Trend Line for Salary vs Performance Change
slope, intercept, r_value, p_value, std_err =
stats.linregress(salary_changes_flat, performance_changes_flat)

# Step 3: Choose the best team per year based on highest batting average
and salary
top_players_by_year = {}
for y in np.unique(year):
    yearly_indices = np.where(year == y)
    yearly_teams = team[yearly_indices]
    for t in np.unique(yearly_teams):
        team_indices = np.where((year == y) & (team == t))
        best_player_idx =
team_indices[0][np.argmax(batting_avg[team_indices])]
        top_players_by_year[y] = {
            'team': team[best_player_idx],
            'playerID': batting_player_ids[best_player_idx],
            'batting_avg': batting_avg[best_player_idx],
            'salary': salary[best_player_idx],
            'year': year[best_player_idx]
        }

# Choose the best player each year
best_team_by_year = {}
for y in np.unique(year):
    year_players = [v for k, v in top_players_by_year.items() if v['year']
== y]
    if year_players:
        best_player = max(year_players, key=lambda x: (x['batting_avg'],
x['salary']))
        best_team_by_year[y] = best_player

# Prepare the data for scatter plot
```

```
best_years = []
best_batting_avg = []
best_salaries = []
for y, data in best_team_by_year.items():
    best_years.append(data['year'])
    best_batting_avg.append(data['batting_avg'])
    best_salaries.append(data['salary'])

# Convert lists to numpy arrays for plotting
best_years = np.array(best_years)
best_batting_avg = np.array(best_batting_avg)
best_salaries = np.array(best_salaries)

# Step 4: Analyze Salary vs Performance Variance for each team
team_variances = {}
for t in np.unique(team):
    team_indices = np.where(team == t)
    team_salary_variance = np.var(salary[team_indices])
    team_batting_avg_variance = np.var(batting_avg[team_indices])
    team_variances[t] = {'salary_variance': team_salary_variance,
'batting_avg_variance': team_batting_avg_variance}

# Prepare variance data for plot
salary_variances = []
batting_avg_variances = []
for t, variances in team_variances.items():
    salary_variances.append(variances['salary_variance'])
    batting_avg_variances.append(variances['batting_avg_variance'])

# Convert to numpy arrays for plotting
salary_variances = np.array(salary_variances)
batting_avg_variances = np.array(batting_avg_variances)

# Step 5: Set up subplots
fig = plt.figure(figsize=(15, 10))
gs = gridspec.GridSpec(2, 2, width_ratios=[1, 1], height_ratios=[1, 1])

# Plot 1: Top Teams' Performance Over the Years
ax0 = plt.subplot(gs[0])
```

```
ax0.scatter(best_years, best_batting_avg, s=best_salaries / 100000,
c='blue', edgecolor='black', alpha=0.7)
ax0.set_title("Top Teams' Performance Over the Years")
ax0.set_xlabel('Year')
ax0.set_ylabel('Batting Average')
ax0.grid(True)

# Plot 2: Salary Change vs Performance Change with Trend Line
ax1 = plt.subplot(gs[1])
ax1.scatter(salary_changes_flat, performance_changes_flat, alpha=0.6,
label='Team Data')
ax1.plot(salary_changes_flat, intercept + slope * salary_changes_flat,
'r', label=f'Trend Line (slope={slope:.2f})')
ax1.set_title('Salary Change vs Performance Change')
ax1.set_xlabel('Salary Change (%)')
ax1.set_ylabel('Performance Change')
ax1.legend()
ax1.grid(True)

# Plot 3: Salary Variance vs Performance Variance
ax2 = plt.subplot(gs[2])
ax2.scatter(salary_variances, batting_avg_variances, color='green',
edgecolor='black', alpha=0.7)
ax2.set_title('Salary Variance vs Performance Variance')
ax2.set_xlabel('Salary Variance')
ax2.set_ylabel('Performance Variance')
ax2.grid(True)

# Adjust layout
plt.tight_layout()

# Save the figure as an SVG file
plt.savefig('team_performance_analysis.svg', format='svg')

# Show the figure
plt.show()
# Sammy Masters
```

Under null hypothesis that pay increases team performance shows a general trend upwards
Ever so slightly with a tolerance of 10%