Lineup Optimization

Table of contents

1 Import Packages & Data

```python
import pandas as pd
import numpy as np
import itertools
import random
import warnings

warnings.filterwarnings('ignore')

uconn_data = pd.read_csv('data/in-game-trackman/final_fall_stats.csv')
```

2 Data Cleaning & Necessary Information

```python
uconn_data['Name'] = uconn_data['First'] + ' ' + uconn_data['Last']
uconn_data.drop(columns = ['First', 'Last'], inplace = True)

req_data = uconn_data[['Name', 'Prob1B', 'Prob2B', 'Prob3B', 'Prob4B',
                                                              'ProbOut']]
```

2.1 TODO 1: Set Player Positions

```python
# players per position
catcher = ['Matt Garbowski', 'Connor Lane', 'Mike Oates', 'Gabriel Tirado']
first_b = ['Maddix Dalena', 'Grant MacArthur', 'Gabriel Tirado']
second_b = ['Ryan Daniels', 'Jack LaRose', 'Bryan Padilla', 'Rob Rispoli']
third_b = ['Tyler Minick', 'Bryan Padilla', 'Jack LaRose']
ss = ['Anthony Belisario', 'Bryan Padilla', 'Rob Rispoli']
left = ['Sam Biller', 'Carter Groen', 'Drew Kron', 'Beau Root',
                                                'Aidan Dougherty']
right = ['Sam Biller', 'Aidan Dougherty', 'Carter Groen', 'Caleb Shpur',
                                                                'Drew Kron']
center = ['Sam Biller', 'Drew Kron', 'Beau Root', 'Caleb Shpur']
dh = ['Maddix Dalena', 'Ryan Daniels', 'Matt Garbowski', 'Carter Groen',
                                                            'Tyler Minick']
```

2.2 TODO 2: Establish Definite Starters

```python
# definite starters
starters = ['Matt Garbowski', 'Maddix Dalena', 'Bryan Padilla',
            'Ryan Daniels', 'Tyler Minick', 'Caleb Shpur', 'Sam Biller',
                                                            'Rob Rispoli']
```

2.3 TODO 3: Limit Player Positions in Lineup

```python
# possible lineup positions
first_names = ['Matt Garbowski', 'Bryan Padilla', 'Tyler Minick',
                'Drew Kron', 'Beau Root', 'Caleb Shpur', 'Sam Biller',
                                                      'Rob Rispoli']
second_names = ['Matt Garbowski', 'Bryan Padilla', 'Tyler Minick',
                    'Drew Kron', 'Beau Root', 'Sam Biller', 'Rob Rispoli']
third_names = ['Matt Garbowski', 'Maddix Dalena', 'Grant MacArthur',
                'Jack LaRose', 'Bryan Padilla', 'Tyler Minick', 'Drew Kron',
                    'Beau Root', 'Sam Biller', 'Rob Rispoli', 'Ryan Daniels',
                                            'Connor Lane', 'Gabriel Tirado']
fourth_names = ['Matt Garbowski', 'Maddix Dalena', 'Grant MacArthur',
                'Jack LaRose', 'Bryan Padilla', 'Tyler Minick', 'Drew Kron',
                    'Beau Root', 'Sam Biller', 'Rob Rispoli', 'Ryan Daniels',
                                            'Connor Lane', 'Gabriel Tirado']
fifth_names = ['Matt Garbowski', 'Maddix Dalena', 'Beau Root', 'Caleb Shpur',
                'Bryan Padilla', 'Tyler Minick', 'Ryan Daniels', 'Connor Lane',
                    'Gabriel Tirado', 'Sam Biller', 'Rob Rispoli', 'Mike Oates',
                                    'Grant MacArthur', 'Jack LaRose', 'Drew Kron']
sixth_names = ['Matt Garbowski', 'Maddix Dalena', 'Beau Root', 'Caleb Shpur',
                            'Tyler Minick', 'Ryan Daniels', 'Anthony Belisario',
                                    'Connor Lane', 'Gabriel Tirado', 'Sam Biller',
                                    'Rob Rispoli', 'Mike Oates', 'Grant MacArthur',
                                                        'Jack LaRose', 'Drew Kron']
seventh_names = ['Caleb Shpur', 'Ryan Daniels', 'Anthony Belisario',
                        'Connor Lane', 'Gabriel Tirado', 'Sam Biller',
                            'Carter Groen', 'Rob Rispoli', 'Mike Oates',
                            'Grant MacArthur', 'Jack LaRose','Drew Kron']
eighth_names = ['Ryan Daniels', 'Anthony Belisario', 'Connor Lane',
                'Gabriel Tirado', 'Sam Biller', 'Carter Groen', 'Rob Rispoli',
                    'Mike Oates', 'Grant MacArthur', 'Jack LaRose', 'Drew Kron']
ninth_names = ['Ryan Daniels', 'Anthony Belisario', 'Connor Lane',
                    'Gabriel Tirado', 'Carter Groen', 'Mike Oates',
                        'Grant MacArthur', 'Jack LaRose', 'Drew Kron']
```

2.4 Probabilities & Simulation Preparation

```
# probability assumptions
prob_2ndtohome_single = 0.60
prob_1sttohome_double = 0.45
prob_2ndto3rd_double = 0.10

num_games = 100
```

3 Monte-Carlo Simulation

```python
# table for mapping player names to numbers
lookup_table = pd.DataFrame({'Number_for_Player': range(1, 20),
                                        'Player_Name': req_data['Name']})
```

```python
c_nums = np.empty(len(catcher))
first_nums = np.empty(len(first_b))
second_nums = np.empty(len(second_b))
third_nums = np.empty(len(third_b))
short_nums = np.empty(len(ss))
left_nums = np.empty(len(left))
center_nums = np.empty(len(center))
right_nums = np.empty(len(right))
dh_nums = np.empty(len(dh))

def fill_lookup(pos, lst):
    for i in range(len(lst)):
        try:
            lst[i] = lookup_table['Player_Name'].tolist().index(pos[i]) + 1
        except ValueError:
            lst[i] = np.nan

fill_lookup(catcher, c_nums)
fill_lookup(first_b, first_nums)
fill_lookup(second_b, second_nums)
fill_lookup(third_b, third_nums)
fill_lookup(ss, short_nums)
fill_lookup(left, left_nums)
fill_lookup(center, center_nums)
fill_lookup(right, right_nums)
fill_lookup(dh, dh_nums)
```

```python
comb_grid = pd.DataFrame(
    itertools.product(c_nums, first_nums, second_nums, third_nums,
                        short_nums, left_nums, right_nums, center_nums,
                                                        dh_nums),
    columns = ['C_Numbers', 'First_Numbers', 'Second_Numbers',
                'Short_Numbers', 'Third_Numbers', 'LF_Numbers',
                        'RF_Numbers', 'CF_Numbers', 'DH_Numbers']
)
```

```
comb_grid = comb_grid.values
cols = ['C_Numbers', 'First_Numbers', 'Second_Numbers', 'Short_Numbers',
                'Third_Numbers', 'LF_Numbers', 'RF_Numbers', 'CF_Numbers',
                                                'DH_Numbers']

comb_grid = pd.DataFrame(comb_grid, columns = cols)

comb_grid['TestingColumn'] = 'tmp'

comb_grid['TestingColumn'] = comb_grid.iloc[:, :9].apply(lambda row: len(set(row)),
                                                            axis = 1)

comb_grid = comb_grid[comb_grid['TestingColumn'] == 9]
```

```
starters = [lookup_table['Player_Name'].tolist().index(s) + 1 for s in starters]

comb_grid['start_test'] = comb_grid.iloc[:, :9].apply(
    lambda row: all(num in row.values for num in starters), axis=1
)

comb_grid = comb_grid[comb_grid['start_test'] == True]

comb_grid.drop(columns = ['start_test', 'TestingColumn'], inplace = True)

comb_grid.columns = ['C', '1B', '2B', 'SS', '3B', 'LF', 'RF', 'CF', 'DH']

comb_grid_text = comb_grid.copy()

for col in comb_grid_text.columns:
    comb_grid_text[col] = comb_grid_text[col].apply(
        lambda num: lookup_table.iloc[int(num) - 1, 1]
    )
```

```
first_perm = list(itertools.permutations(comb_grid.iloc[0, :].astype(int)))

permutations_expanded = pd.DataFrame(first_perm)

for i in range(1, comb_grid.shape[0]):
    row_perm = list(itertools.permutations(comb_grid.iloc[i, :].astype(int)))
    permutations_expanded = pd.concat(
        [permutations_expanded, pd.DataFrame(row_perm)], ignore_index = True
    )

permutations_expanded.columns = ['first', 'second', 'third', 'fourth',
                                    'fifth', 'sixth', 'seventh', 'eighth',
                                                                'ninth']

permutations_expanded = permutations_expanded.drop_duplicates()
```

```
def map_to_indices(names, lookup_table):
    return lookup_table.loc[lookup_table['Player_Name'].isin(names),
                                                'Number_for_Player'].tolist()

first_opt = map_to_indices(first_names, lookup_table)
second_opt = map_to_indices(second_names, lookup_table)
third_opt = map_to_indices(third_names, lookup_table)
fourth_opt = map_to_indices(fourth_names, lookup_table)
fifth_opt = map_to_indices(fifth_names, lookup_table)
sixth_opt = map_to_indices(sixth_names, lookup_table)
seventh_opt = map_to_indices(seventh_names, lookup_table)
eighth_opt = map_to_indices(eighth_names, lookup_table)
ninth_opt = map_to_indices(ninth_names, lookup_table)

permutations_condensed = permutations_expanded[
    (permutations_expanded['first'].isin(first_opt)) &
    (permutations_expanded['second'].isin(second_opt)) &
    (permutations_expanded['third'].isin(third_opt)) &
    (permutations_expanded['fourth'].isin(fourth_opt)) &
    (permutations_expanded['fifth'].isin(fifth_opt)) &
    (permutations_expanded['sixth'].isin(sixth_opt)) &
    (permutations_expanded['seventh'].isin(seventh_opt)) &
    (permutations_expanded['eighth'].isin(eighth_opt)) &
    (permutations_expanded['ninth'].isin(ninth_opt))
].copy()

permutations_condensed['mean_runs'] = 0
permutations_condensed['sd_runs'] = 0

permutations_condensed = permutations_condensed.reset_index()
permutations_condensed = permutations_condensed.drop(columns = ['index'])
```

```python
np.random.seed(777)

def update_cols(prev_row, hit, prob_wts = None):
    '''Move runners and update score.'''
    r1 = prev_row[9]
    r2 = prev_row[10]
    r3 = prev_row[11]
    runs = prev_row[12]

    # single
    if hit == 1:
        # no runners
        if r1 == 0: return [1, 0, 0, runs]
        # 1b
        elif r1 == 1 and r2 == 0:
            return [2, 1, 0, runs]
        # 2b
        elif r1 == 2 and r2 == 0:
            r1 = random.choices([3, 1], weights = prob_wts, k = 1)[0]
            return [3, 1, 0, runs] if r1 == 3 else [1, 0, 0, 1 + runs]
        # 1b & 2b
        elif r1 == 2 and r2 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 1, runs] if r1 == 3 else [2, 1, 0, 1 + runs]
        # 3b
        elif r1 == 3 and r2 == 0: return [1, 0, 0, 1 + runs]
        # 1b & 3b
        elif r1 == 3 and r2 == 1: return [2, 1, 0, 1 + runs]
        # 2b & 3b
        elif r1 == 3 and r2 == 2 and r3 == 0:
            r1 = random.choices([3, 1], weights = prob_wts, k = 1)[0]
            return [3, 1, 0, 1 + runs] if r1 == 3 else [1, 0, 0, 2 + runs]
        # bases loaded
        elif r3 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 1, 1 + runs] if r1 == 3 else [2, 1, 0, 2 + runs]
        else: return [0, 0, 0, 0]

    # double
    elif hit == 2:
        # no runners
        if r1 == 0: return [2, 0, 0, runs]
        # 1b -- homebydouble
        elif r1 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 0, runs] if r1 == 3 else [2, 0, 0, 1 + runs]
        # 2b -- 2ndto3rd
        elif r1 == 2 and r2 == 0:
            r1 = random.choices([2, 3], weights = prob_wts, k = 1)[0]
            return [2, 0, 0, 1 + runs] if r1 == 2 else [3, 2, 0, runs]
        # 1b & 2b -- homebydouble
        elif r1 == 2 and r2 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 0, 1 + runs] if r1 == 3 else [2, 0, 0, 2 + runs]
        # 3b
        elif r1 == 3 and r2 == 0: return [2, 0, 0, 1 + runs]
        # 1b & 3b -- homebydouble
        elif r1 == 3 and r2 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 0, 1 + runs] if r1 == 3 else [2, 0, 0, 2 + runs]
        # 2b & 3b -- 2ndto3rd
        elif r1 == 3 and r2 == 2 and r1 == 0:
            r1 = random.choices([2, 3], weights = prob_wts, k = 1)[0]
            return [2, 0, 0, 2 + runs] if r1 == 2 else [3, 2, 0, 1 + runs]
        # bases loaded -- homebydouble
        elif r3 == 1:
            r1 = random.choices([3, 2], weights = prob_wts, k = 1)[0]
            return [3, 2, 0, 2 + runs] if r1 == 3 else [2, 0, 0, 3 + runs]
        else: return [0, 0, 0, 0]

    # triple
    elif hit == 3:
        # no runners
        if r1 == 0: return [3, 0, 0, runs]
        # 1b or 2b or 3b
        elif r1 != 0 and r2 == 0: return [3, 0, 0, 1 + runs]
        # 1b & 2b or 1b & 3b or 2b & 3b
        elif r1 != 0 and r2 != 0 and r3 == 0: return [3, 0, 0, 2 + runs]
        # bases loaded
        elif r3 == 1: return [3, 0, 0, 3 + runs]
        else: return [0, 0, 0, 0]

    # homerun
    elif hit == 4:
        # no runners
        if r1 == 0: return [0, 0, 0, 1 + runs]
        # 1b, 2b, 3b
        elif r1 != 0 and r2 == 0: return [0, 0, 0, 2 + runs]
        # 1b & 2b, 1b & 3b, 2b & 3b
        elif r1 != 0 and r2 != 0 and r3 == 0: return [0, 0, 0, 3 + runs]
        # bases loaded
        elif r3 == 1: return [0, 0, 0, 4 + runs]
        else: return [0, 0, 0, 0]

def process_event(cur_row, prev_row):
    '''Update game settings and score based on event.'''
    prev_row = np.nan_to_num(prev_row, nan = 0)

    # last out of game -- nothing left to update
    if prev_row[7] == 9 and prev_row[8] == 3:
        cur_row[7] = 9
        cur_row[8] = 3
        cur_row[12] = prev_row[12]

    # less than 3 outs
    if prev_row[8] != 3:
        cur_row[7:9] = prev_row[7:9]
        cur_row[12] = prev_row[12]

    # no hit
    if cur_row[6] == 0:
        if prev_row[8] != 3:
            cur_row[8] = 1 + prev_row[8]
            cur_row[9:] = prev_row[9:]
        else:
            cur_row[8] = 1
            cur_row[12] = prev_row[12]

    # single
    elif cur_row[6] == 1:
        settings = update_cols(prev_row, 1, [1 - prob_2ndtohome_single,
                                              prob_2ndtohome_single])

    # double
    elif cur_row[6] == 2:
        if prev_row[9] == 0 or prev_row[9] == 1 or prev_row[11] == 1:
            probs = [1 - prob_1sttohome_double, prob_1sttohome_double]
        else:
            probs = [1 - prob_2ndto3rd_double, prob_2ndto3rd_double]
        settings = update_cols(prev_row, 2, probs)

    # triple
    elif cur_row[6] == 3:
        settings = update_cols(prev_row, 3)

    # homerun
    elif cur_row[6] == 4:
        settings = update_cols(prev_row, 4)

    if cur_row[6] != 0:
        cur_row[9], cur_row[10], cur_row[11], cur_row[12] = settings

    cur_row = np.nan_to_num(cur_row, nan = 0)

    return cur_row
```

```
np.random.seed(777)

# map batter names to integers
batter_names = req_data.iloc[:, 0].unique()
batter_map = dict(zip(lookup_table['Player_Name'],
                        lookup_table['Number_for_Player']))

num_permutations = len(permutations_condensed)

# initialize storage for run counts and game data
run_total = np.zeros(num_games)
skeleton_arr = np.full((81, 13), np.nan, dtype = float)

# iterate through lineups
for i in range(num_permutations):
    # get possible lineup
    indices = permutations_condensed.iloc[i, :9].values - 1

    # get list of batters and encode names as integers
    batter_vec = req_data.iloc[indices, 0].values
    batter_vec = np.vectorize(lambda name: batter_map[name])(batter_vec)
```

```
# get probabilities associated with lineup
prob_vecs = req_data.iloc[indices, 1:6].values

# combine batter and probability vectors into array of length 81
skeleton_arr[:, :6] = np.tile(np.column_stack([batter_vec, prob_vecs]),
                                                                (9, 1))

# normalize data so probabilities add to one
prob_cols = skeleton_arr[:, 1:6]
prob_sums = prob_cols.sum(axis = 1, keepdims = True)
skeleton_arr[:, 1:6] = prob_cols / prob_sums

# iterate through games
for j in range(num_games):
    skeleton_arr_tmp = skeleton_arr.copy()

    # assign a random result (out, 1b, 2b, 3b, hr) to each row
    skeleton_arr_tmp[:, 6] = np.array([
        np.random.choice([0, 1, 2, 3, 4], size = 1, p = row)[0]
        for row in skeleton_arr_tmp[:, 1:6]
    ])

    # initial states
    skeleton_arr_tmp[:, 7] = 1  # inning
    skeleton_arr_tmp[:, 12] = 0  # runs
    skeleton_arr_tmp[:, 8] = 0  # outs

    # first outcome of game
    first_res = skeleton_arr_tmp[0, 6]

    # adjust outs and runs according to first outcome
    skeleton_arr_tmp[0, 8] = 1 if first_res == 0 else 0
    skeleton_arr_tmp[0, 12] = 1 if first_res == 4 else 0

    # update runner placements according to first outcome
    skeleton_arr_tmp[0, 9:12] = 0
    skeleton_arr_tmp[0, 9] = 1 if first_res == 1 else 0
    skeleton_arr_tmp[0, 10] = 1 if first_res == 2 else 0
    skeleton_arr_tmp[0, 11] = 1 if first_res == 3 else 0

    # iterate through each at bat
    for k in range(1, len(skeleton_arr_tmp)):
        # store previous and current rows
        prev_row = skeleton_arr_tmp[k - 1, :]
        cur_row = skeleton_arr_tmp[k, :]

        # end of game -- stop
        if (prev_row[7] == 9 and prev_row[8] == 3):
            skeleton_arr_tmp[k, 7] = 9
            skeleton_arr_tmp[k, 8] = 3
            skeleton_arr_tmp[k, 12] = prev_row[12]

        # process next outcome of game
        else:
            skeleton_arr_tmp[k, :] = process_event(cur_row, prev_row)

    # get run total for each game and lineup
    run_total[j] = skeleton_arr_tmp[80, 12]

# calculate mean and standard deviation of all run totals
orderopt_mean = np.mean(run_total)
orderopt_sd = np.std(run_total)

permutations_condensed.loc[i, 'mean_runs'] = orderopt_mean
permutations_condensed.loc[i, 'sd_runs'] = orderopt_sd
```

4 Results

```
text_results = permutations_condensed

# convert integers back to names
for i in range(9):
    for j in range(len(text_results)):
        numeric_value = int(text_results.iloc[j, i]) - 1
        text_results.iloc[j, i] = lookup_table.iloc[numeric_value, 1]

# sort lineups by most runs created
text_results = text_results.sort_values(by = 'mean_runs')

# get top three best lineups
top_3 = text_results.iloc[:3, :-2]
```

```
top_3.iloc[:, :5]
```

	first	second	third	fourth	fifth
2665	Caleb Shpur	Tyler Minick	Sam Biller	Matt Garbowski	Bryan Padilla
251	Matt Garbowski	Rob Rispoli	Bryan Padilla	Maddix Dalena	Tyler Minick
1233	Bryan Padilla	Tyler Minick	Sam Biller	Maddix Dalena	Matt Garbowski

```
top_3.iloc[:, 5:]
```

	sixth	seventh	eighth	ninth
2665	Maddix Dalena	Rob Rispoli	Ryan Daniels	Carter Groen
251	Ryan Daniels	Caleb Shpur	Sam Biller	Carter Groen
1233	Caleb Shpur	Rob Rispoli	Carter Groen	Ryan Daniels

5 Additional Notes

1. The current runtime is about 20-25 minutes for the number of players currently used and the number of preset starters. This could change depending on how many starters you manually set (currently has 8), but overall this shouldn't exceed an hour.
2. The most optimal lineups may not include players in the exact positions you want. To set more strict restrictions, you can edit the nine list in the first TODO section. These state the players that can be chosen for each position and are currently pretty generous in terms of providing multiple players for each position. Obviously, you may have a single player in mind for a certain position, so you can limit those lists if needed.
3. If players are added or removed from the roster, they need to also be added/removed from multiple places in the TODO sections. In TODO 1, they need to be removed from all positions lists they're included in or added to any lists for positions they may play. For TODO 2, removed players must be removed and added players should be added if they have a definite starting position. Lastly, for TODO 3, removed players must be removed fromm all lists they're included in and new players must be added to at least one list.
4. A quick explanation of the third TODO, as it's not as intuitive as the other two: the third TODO is there to limit permutations and remove lineups that are most definitely not optimal. The lists first through ninth include all players that can be placed in that lineup position. For example, Bryan Padilla is a senior and clearly one of UConn's better hitters, so I've removed him from the bottom half of the lineup, as even without proper testing, it's clear that wouldn't be an appropriate position for him. If you'd really like

to see all lineup combinations, you can put the entire roster in each of the nine lists. I discourage this though, as it will make the running time of the code exponentially larger. Feel free to play around with smaller changes if you'd like.

5. You may run into issues with importing the data since I have it saved with a specific name and in a specific folder. In the last line of the first code block, the data is currently imported as 'data/in-game-trackman/final_fall_stats.csv'. For me, this is inside a data folder containing another folder title in-game-trackman, and the file itself is 'final_fall_stats.csv'. You can rename anything you'd like, and remove everything but 'final_fall_stats.csv' if you don't have the data stored in a folder.

:::