Sprint 06

Optimizing Goodlife

Presented by: Group 5

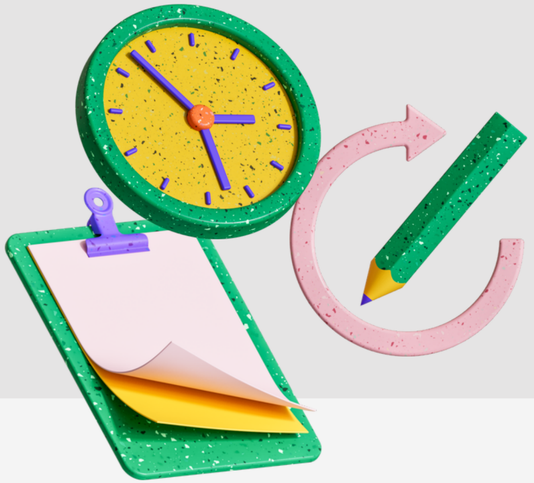
Challenges

- High demand for fitness classes but marked by low attendance rates.
- Importance of accurate attendance predictions to improve space utilization.
- Balancing member satisfaction with operational efficiency.

Objectives And Goals

- Develop predictive optimization models for attendance and facility utilization.
- Enable dynamic allocation of class spaces to maximize participation.
- Leverage insights to improve customer satisfaction and increase revenue.

Data Preprocessing



EDA

- Checking for null values
- Checked for datatypes
- Checked for duplicates
- Checked for irregular values

Data Cleaning

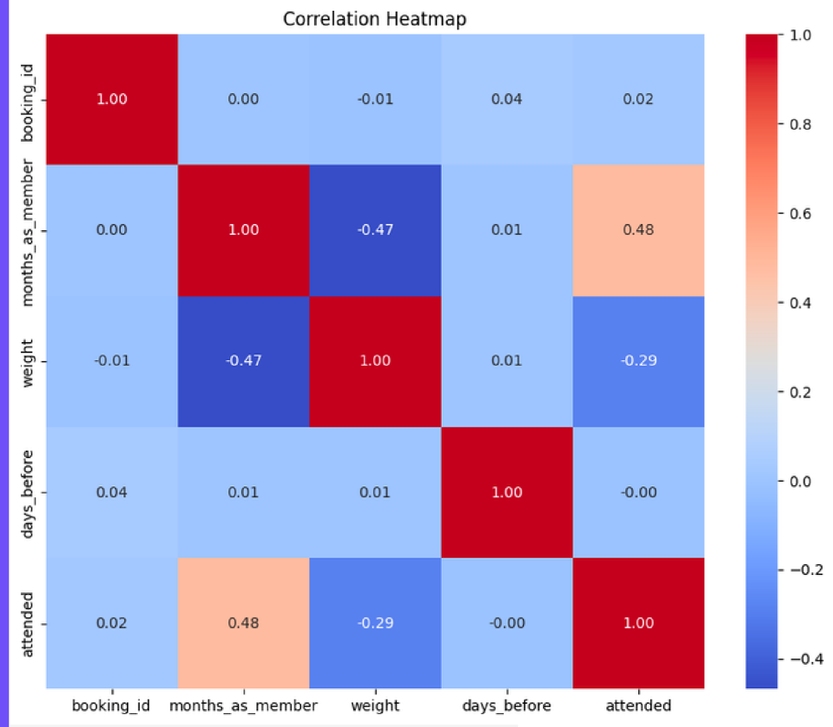
- Removed null values from ‘weight’
- Changed datatype from object to numeric for ‘days_before’
- Removed duplicates from ‘day_of_week’
- Removed 13 ‘-’ values from ‘category’

*Performed the same set of EDA on problem 2 but much data cleaning wasn’t necessary

Exploratory Data Analysis

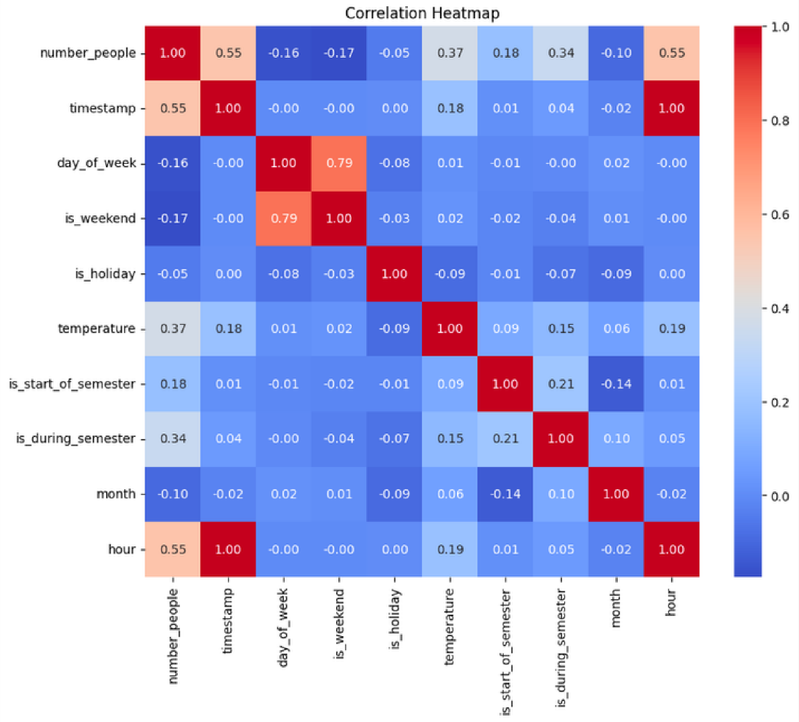
Correlation Matrix used to figure out the significance between features

- Slightly less correlation found between features in problem 1
- Approx 80% of correlation found between 2 values in problem 2



Problem 1

Problem 2



Models Used

XGBoost - Problem 1 & 2

It Is suitable for its robustness in handling imbalanced GoodLife Fitness datasets, where attendance numbers vary greatly.

It effectively processes diverse data without the need for preliminary scaling, accommodating the wide range of features from member weights to booking times.

Also, its capability to handle non-normalized data means we can directly use our 'date' column by converting it to a numerical format without complex preprocessing.

Sprint 06

Problem 1

Optmizing Group Fitness Class Utilization

Group 5

Problem 1

PuLP Optimization Library

Customizable Optimization

The library is extremely helpful in tailoring the model to your specific needs such as allocating members, their charaacteristics and class capacities.

Complex constraints

PuLP handles capacity constraints for different class times and predictive constraints as well. Traditional time series models such as ARIMA, might not be able to handle such constraints as they are focued on predicting future based values

Easy Adaptation to Change

The model can adapt dynamicaly to conditions that change such as input data. For example, if a new member data become available, or class capacitites change, the model can very easily adapt to these conditions

Problem 1

PuLP Functions

LpVariable

Function creates a binary decision variable for each row with the help of a loop

LpProblem

Creates an optimization problem to maximize the total number of booked spaces

LpSum

Uses the function to max the sum of all decision variables

Problem 1

Code Analysis

Function used to perform optimization using PuLP

Briefly elaborate on the observation

Creating binary decision variables

```
def optimize_classes(df):
    # Creating a binary decision variables and map booking IDs to decision variables
    book_space_vars = {}
    for i, row in df.iterrows():
        book_space_vars[row['booking_id']] = LpVariable(f"book_space_{row['booking_id']}", cat='Binary')
```

Creating an optimization problem

```
# Creating an optimization problem
prob = LpProblem("Class_Optimization", LpMaximize)

# Using the function to max out all the decision variables by sum
prob += lpSum(book_space_vars.values())
```

Making sure to fit in with Capacity

```
# Capacity constraint
class_capacity = {'AM': 25, 'PM': 15}  # Assuming capacity based on time
for time, group in df.groupby('time'):
    prob += lpSum(book_space_vars[row['booking_id']] for _, row in group.iterrows()) <= class_capacity[time]

# Predictive constraint
for _, row in df.iterrows():
    if row['attended'] == 0:
        prob += book_space_vars[row['booking_id']] == 1
```

Solving the problem and printing the results

```
# Solve problem
prob.solve()

# Extract results  Option-J to start new chat
results = [booking_id for booking_id, var in book_space_vars.items() if var.value() == 1]

return results
```

Source: *Add your references here.*

Problem 1

Result

PuLP model optimization that’s being done

Numbers indicate booking ids

Possible Attendances: 262

Booking IDs to open additional spaces for: [9, 11, 16, 29, 30, 31, 36, 37, 39, 54, 55, 78, 79, 88, 92, 93, 94, 97, 109, 132, 144, 166, 172, 176, 189, 208, 262, 271, 286, 287, 288, 290, 292, 294, 295, 296, 297, 299, 302, 304, 307, 313, 323, 326, 332, 334, 335, 343, 344, 345, 346, 352, 357, 364, 370, 372, 374, 380, 382, 389, 404, 406, 419, 421, 424, 428, 433, 435, 436, 447, 449, 452, 454, 459, 461, 463, 466, 467, 474, 476, 480, 484, 486, 487, 489, 491, 500, 515, 519, 522, 530, 531, 532, 533, 534, 536, 537, 541, 543, 545, 546, 548, 551, 552, 557, 561, 564, 567, 568, 571, 572, 573, 574, 580, 581, 584, 585, 591, 594, 596, 598, 606, 608, 610, 611, 613, 616, 618, 619, 620, 621, 626, 632, 633, 634, 635, 642, 644, 652, 661, 663, 665, 667, 668, 669, 673, 676, 678, 685, 686, 690, 691, 694, 695, 698, 701, 704, 708, 709, 715, 718, 724, 725, 726, 728, 730, 733, 739, 741, 745, 747, 749, 765, 767, 772, 773, 774, 784, 787, 789, 795, 798, 800, 802, 803, 807, 810, 811, 814, 818, 819, 821, 823, 832, 833, 835, 846, 847, 848, 849, 852, 853, 856, 870, 871, 872, 879, 889, 893, 897, 904, 914, 921, 922, 926, 927, 933, 937, 940, 948, 952, 956, 957, 958, 959, 961, 970, 972, 978, 980, 982, 986, 988, 992, 1033, 1062, 1103, 1115, 1135, 1202, 1228, 1241, 1242, 1251, 1254, 1257, 1261, 1294, 1314, 1339, 1342, 1344, 1350, 1353, 1355, 1399, 1403, 1415, 1441, 1442, 1474, 1481]

Sprint 06

Problem 2

Optimizing Gym Equipment Based on Utilization

Group 5

Problem 2

Results

```
# Evaluate the model on the testing set
y_pred = model_2.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = mse**0.5
print("Root Mean Squared Error:", rmse)
```

RMSE (Root Mean Squared Error)
6.2

Predictions
Predicted value: 79
Actual Value: 79

```
# Predictions for new data
# Assuming new_data contains similar features as the training data
test_data = X_test.iloc[[5]]
actual_result = y_test.iloc[5]
prediction = model_2.predict(test_data)
prediction = round(prediction[0])
print("Predicted gym crowding for new data:", prediction, "\nActual gym crowding for new data:", actual_result)
  Option-J to start new chat
```

```
Predicted gym crowding for new data: 79
Actual gym crowding for new data: 79
```

Thank You