Data Science Project on Daily Step Counts

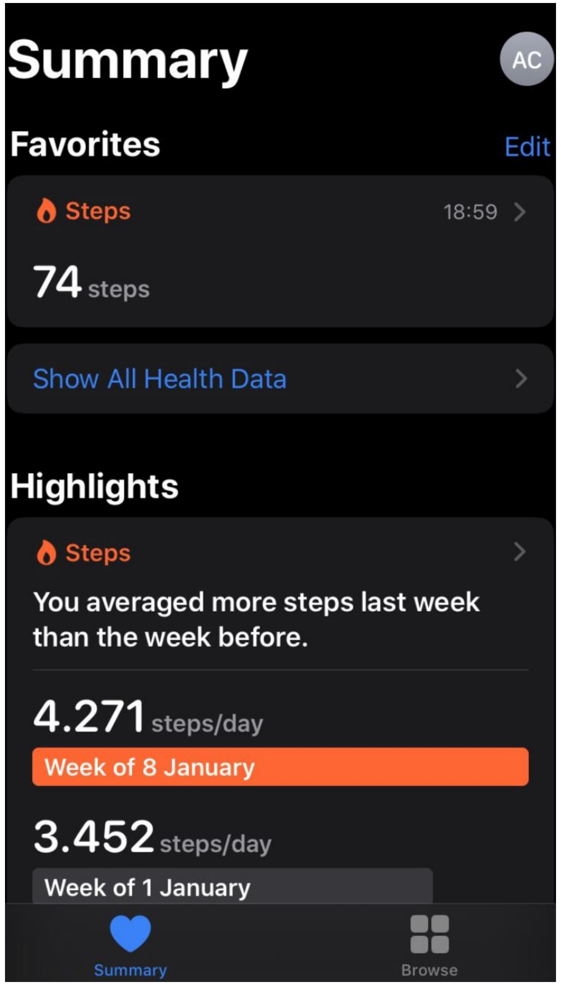
Motivation

- Interesting dataset
- Insights about myself
- A lot of data to work with
- Pattern analysis
- Using machine learning to predict future data

Data Source

```
endDate="2018-06-03 19:53:34 +0300" value="85"/>
endDate="2018-06-03 20:03:34 +0300" value="1033"/>
endDate="2018-06-03 20:13:00 +0300" value="249"/>
endDate="2018-06-03 20:20:14 +0300" value="253"/>
endDate="2018-06-03 21:18:52 +0300" value="20"/>
```

- Apple's "Health" app keeps track of daily steps
- There are over 5 years of data about me approximately 2000 days
- All data from the app can easily be exported into a xml file
- Data inside xml file is in a very easy format to read, allowing parsing and working on the data easy.



Data analysis

- Data is parsed into a pandas data frame
- First to work on data and later for machine learning data needs to be binned / maped
- Steps divided into 8 bins
- All columns are maped

index	Day of the Week	Day of the Month	Month	Year	TotalDistanceWalkingRunning	TotalStepCount
0	Friday	1	April	2022	0.17224	255
1	Friday	1	December	2023	5.338128	7271
2	Friday	1	January	2021	0.55074	763
3	Friday	1	July	2022	4.80259	6988
4	Friday	1	March	2019	0.91722	1363
5	Friday	1	May	2020	0.74135	1157
6	Friday	1	November	2019	1.80836	2684
7	Friday	1	October	2021	0.11799	175
8	Friday	1	September	2023	0.13235	186
9	Friday	2	April	2021	0.23316	340
10	Friday	2	August	2019	3.72394	5891

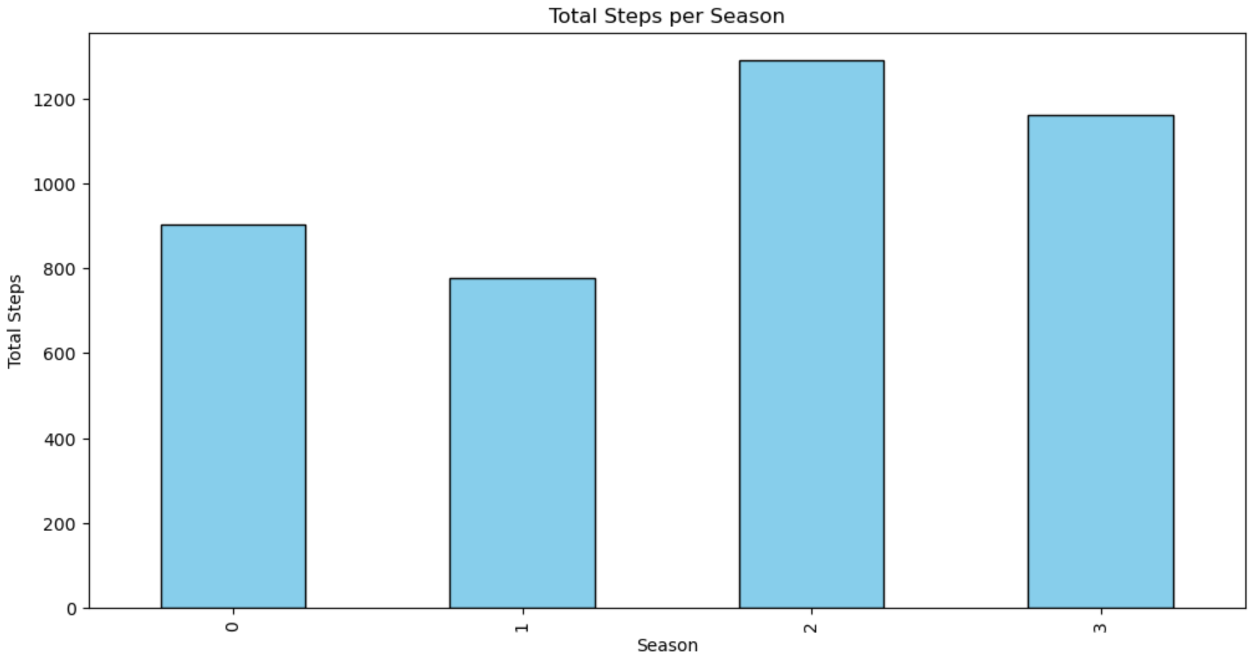
```python
day_map = {'Sunday': 6, 'Saturday': 5, 'Friday': 4, 'Thursday': 3, 'Wednesday': 2, 'Tuesday': 1, 'Monday': 0}
month_map = {'December': 11, 'November': 10, 'October': 9, 'September': 8, 'August': 7, 'July': 6, 'June': 5, '
year_map = {2024: 6, 2023: 5, 2022: 4, 2021: 3, 2020: 2, 2019: 1, 2018: 0}
grouped_df['Day of the Week'] = grouped_df['Day of the Week'].map(day_map)
grouped_df['Month'] = grouped_df['Month'].map(month_map)
grouped_df['Year'] = grouped_df['Year'].map(year_map)

def map_steps(value):
    if value < 100:
        return 0
    elif 100 <= value < 1000:
        return 1
    elif 1000 <= value < 2000:
        return 2
    elif 2000 <= value < 3000:
        return 3
    elif 3000 <= value < 4000:
        return 4
    elif 4000 <= value < 7000:
        return 5
    elif 7000 <= value < 12000:
        return 6
    elif value >= 12000:
        return 7


grouped_df['TotalStepCount'] = grouped_df['TotalStepCount'].apply(map_steps)
```

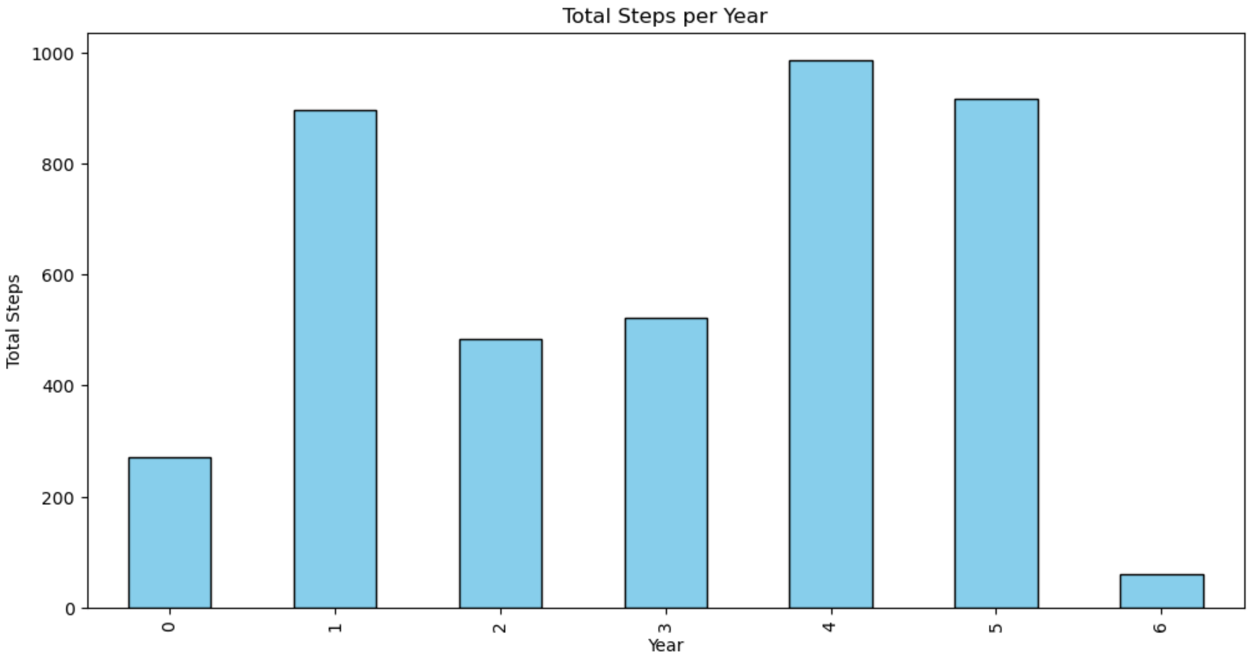
Data analysis

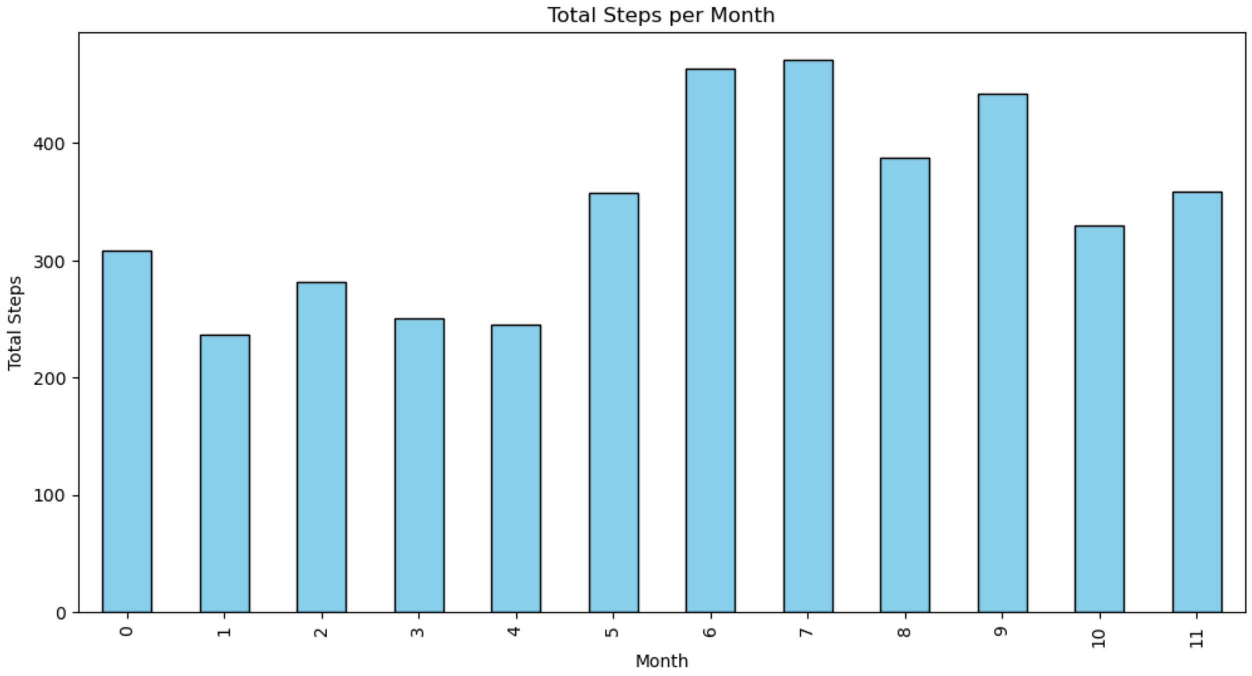
- Adding new features helps to better understand data with creating new features from existing ones
- For example: different seasons might have an impact on the data, we can create this from Month data
- Helps machine learning to realize patterns

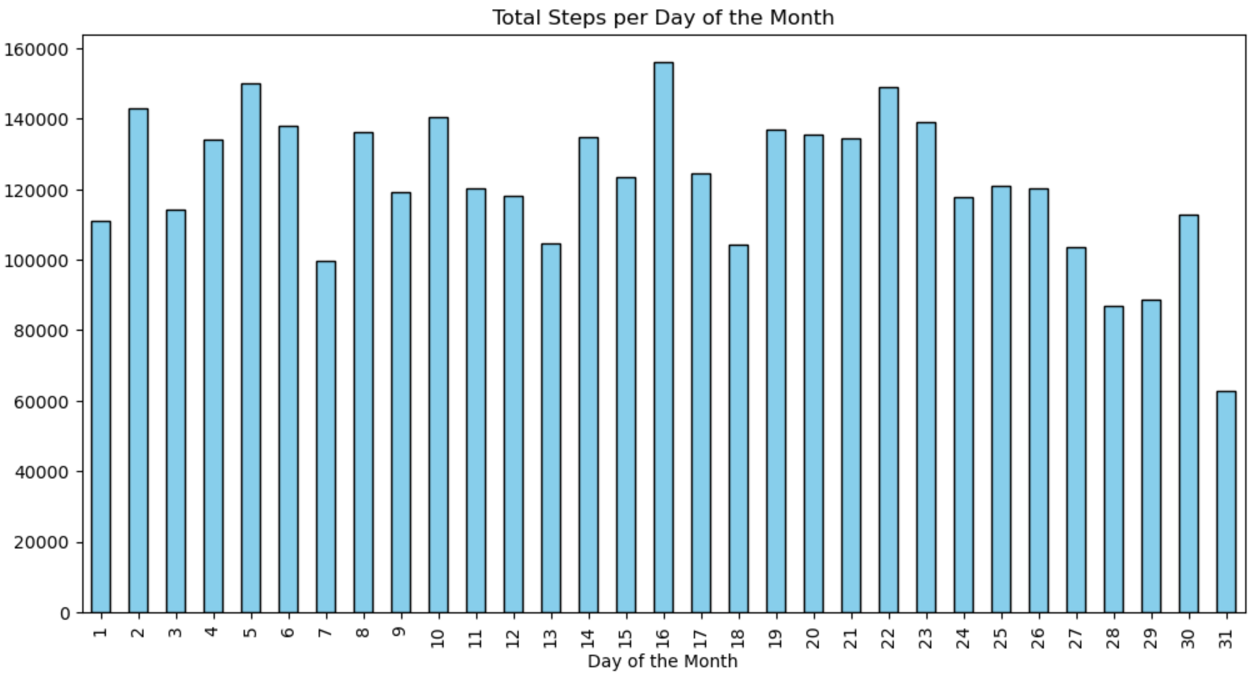
```python
# Weekday/Weekend Column
grouped_df['Weekday_Weekend'] = grouped_df['Day of the Week'].apply(lambda x: 0 if x <= 4 else 1)

# Season Column
def classify_season(month):
    if month in [11, 0, 1]:  # Winter
        return 0
    elif month in [2, 3, 4]:  # Spring
        return 1
    elif month in [5, 6, 7]:  # Summer
        return 2
    elif month in [8, 9, 10]:  # Fall
        return 3

grouped_df['Season'] = grouped_df['Month'].apply(classify_season)

# Last Year Column
grouped_df['Is_Last_Year'] = grouped_df['Year'].apply(lambda x: 1 if x == 5 else 0)
```

index	Day of...	Day of...	Month	Year	TotalD...	TotalS...	Weekday_Weekend	Season	Is_Last_Year
0	4	1	3	4	0.17224	1	0	1	0
1	4	1	11	5	5.338128	6	0	0	1
2	4	1	0	3	0.55074	1	0	0	0
3	4	1	6	4	4.80259	5	0	2	0
4	4	1	2	1	0.91722	2	0	1	0
5	4	1	4	2	0.74135	2	0	1	0
6	4	1	10	1	1.80836	3	0	3	0
7	4	1	9	3	0.11799	1	0	3	0
8	4	1	8	5	0.13235	1	0	3	1
9	4	2	3	3	0.23316	1	0	1	0
10	4	2	7	1	3.72394	5	0	2	0


Total Steps per Season
Total Steps
Season
0
200
400
600
800
1000
1200
0
1
2
3

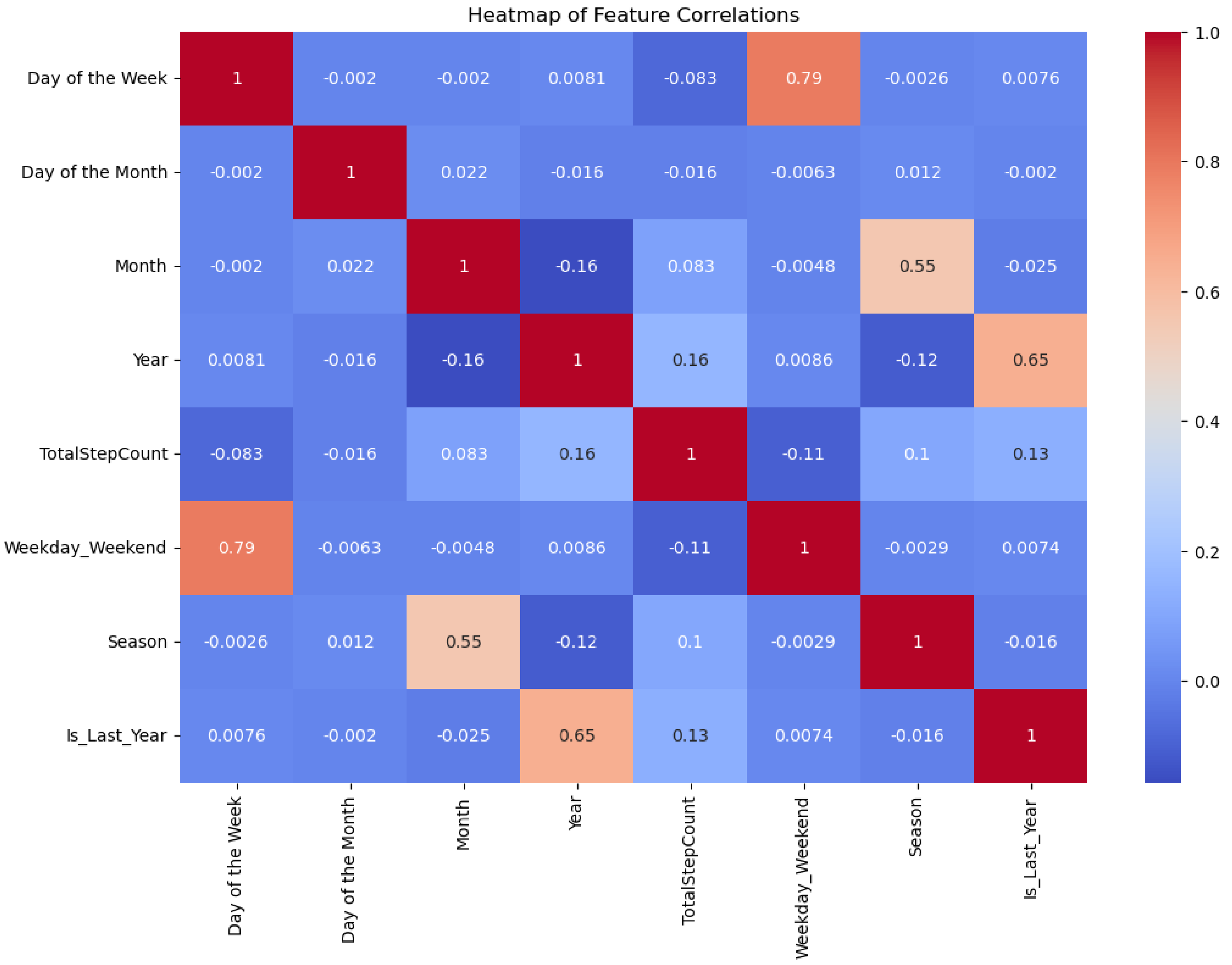

Total Steps per Year
Total Steps
Year
0
200
400
600
800
1000
0
1
2
3
4
5
6

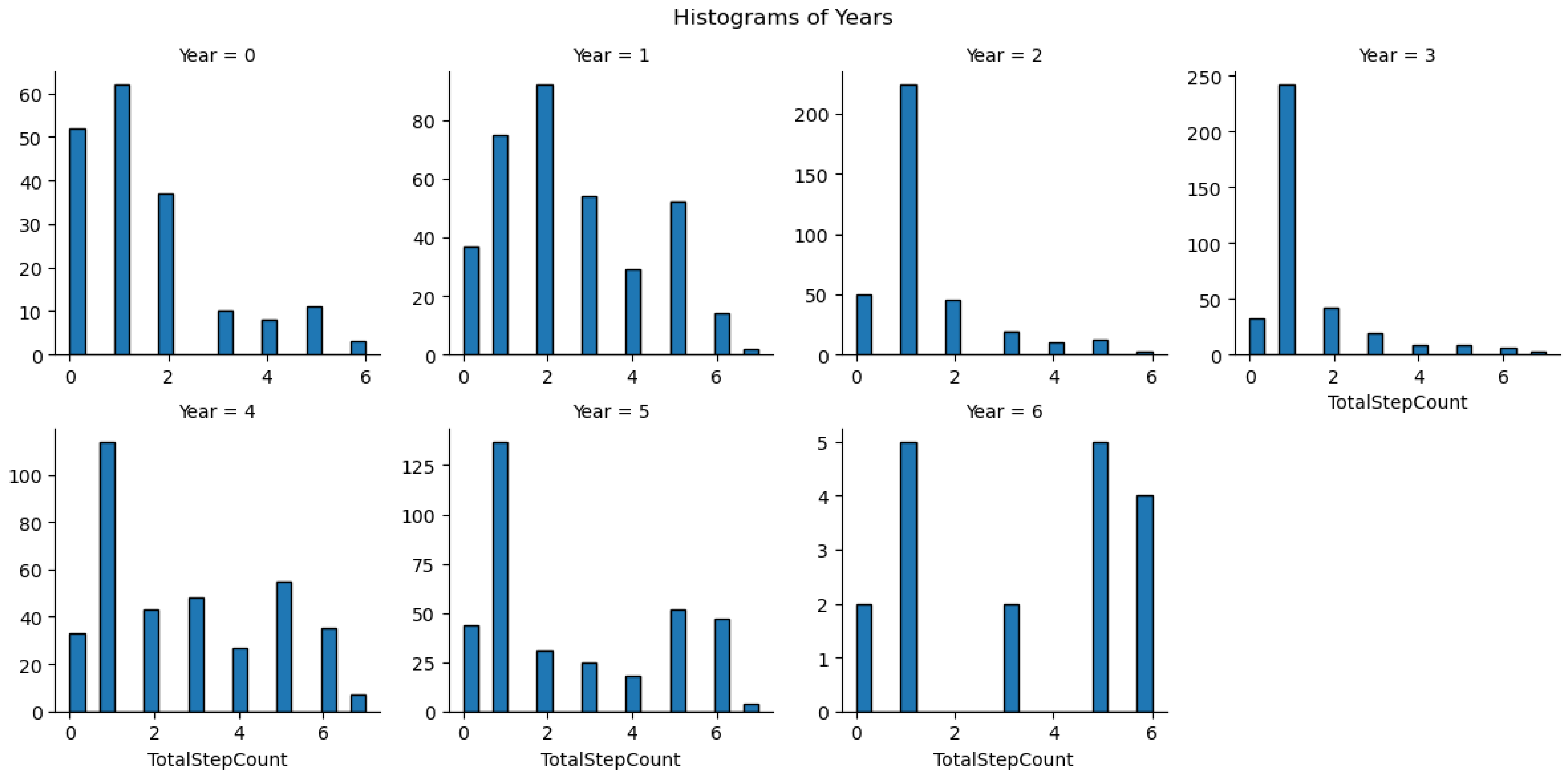

Total Steps per Month
Total Steps
Month
0
100
200
300
400
0
1
2
3
4
5
6
7
8
9
10
11

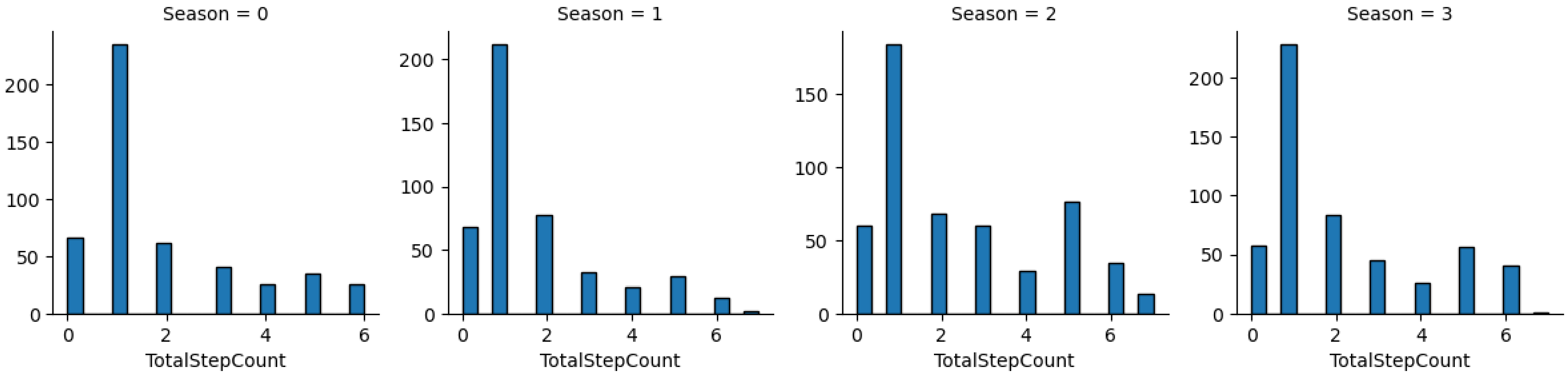

Total Steps per Day of the Month
Day of the Month
0
20000
40000
60000
80000
100000
120000
140000
160000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

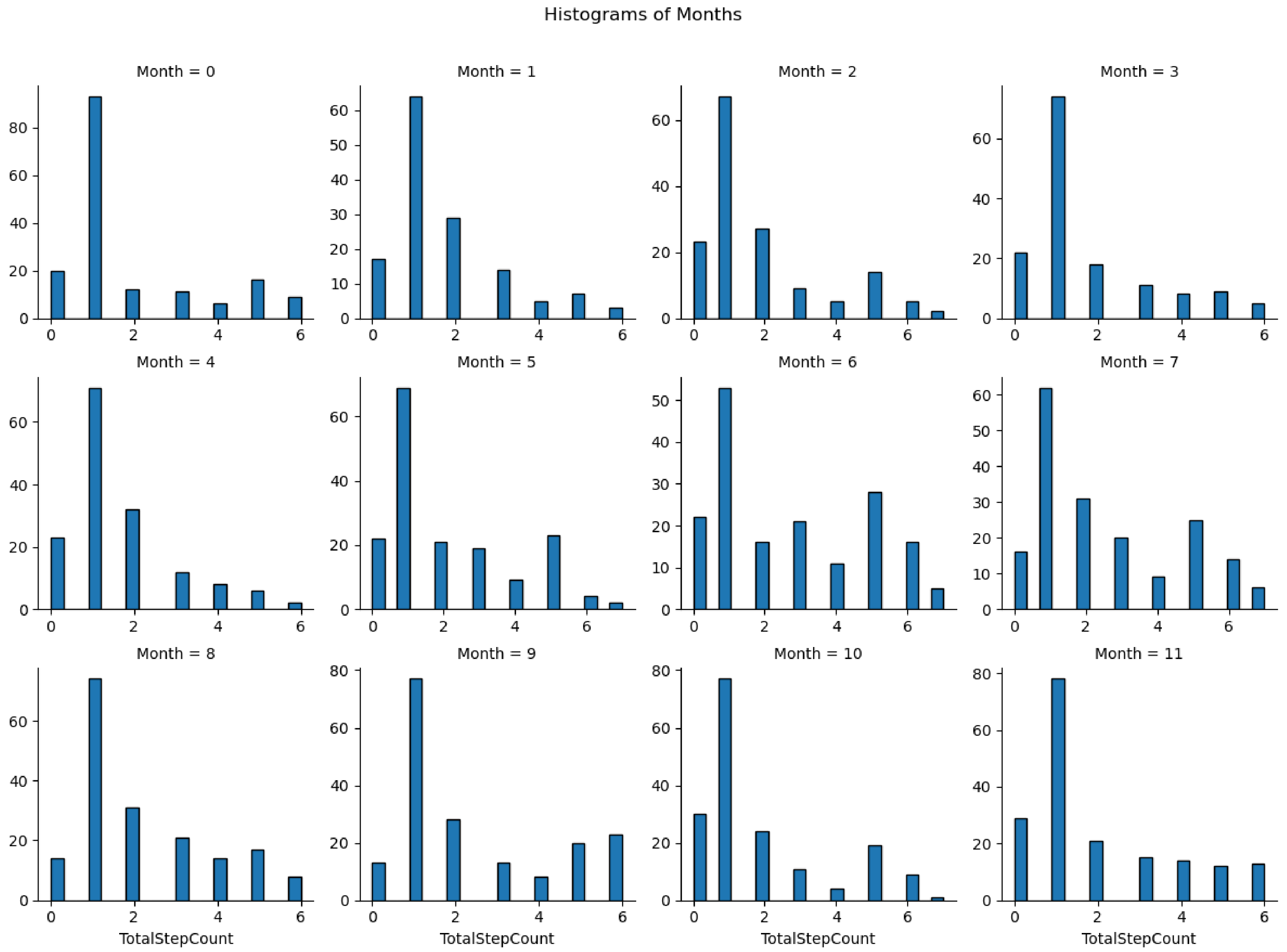
Data analysis

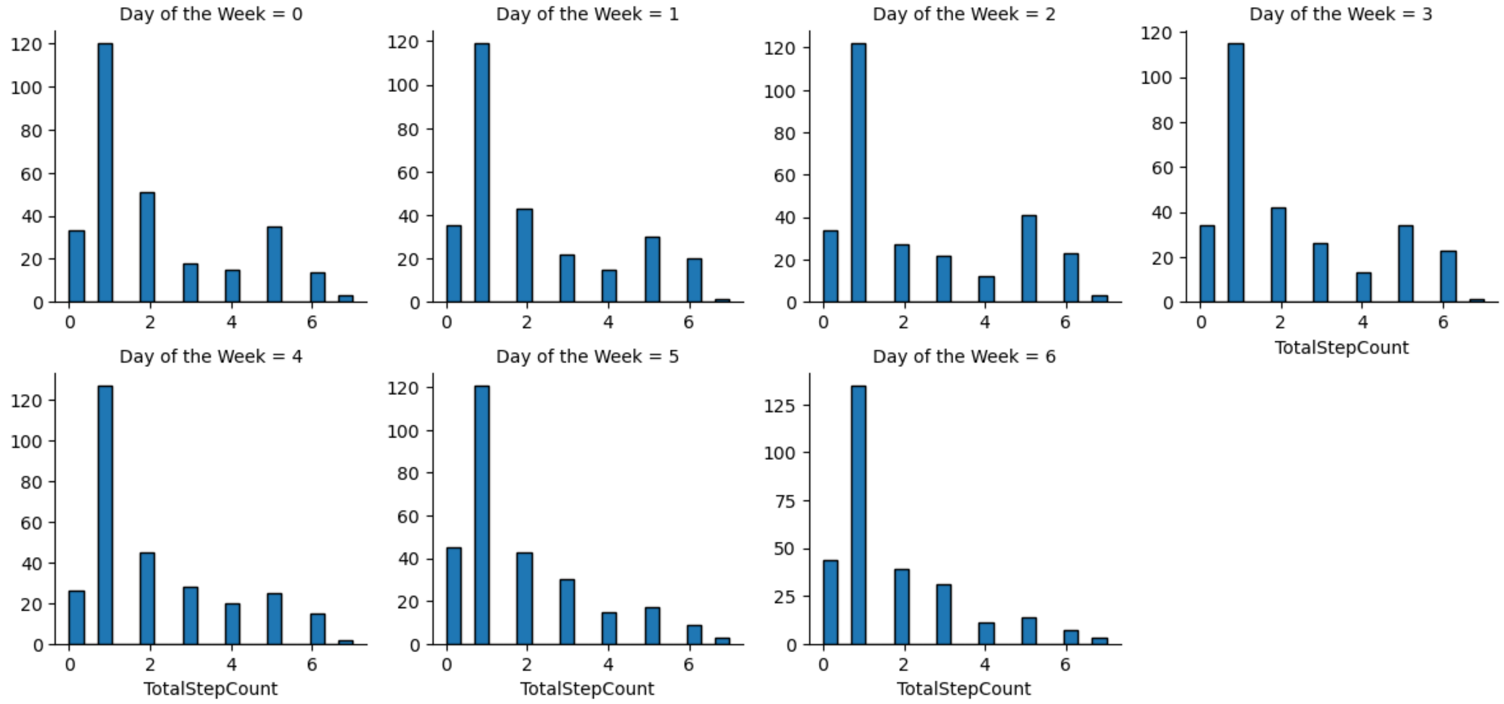
- Correlation heatmap shows no significant correlation between specific features (expected)
- Correlation between derived features is expected but irrelevant

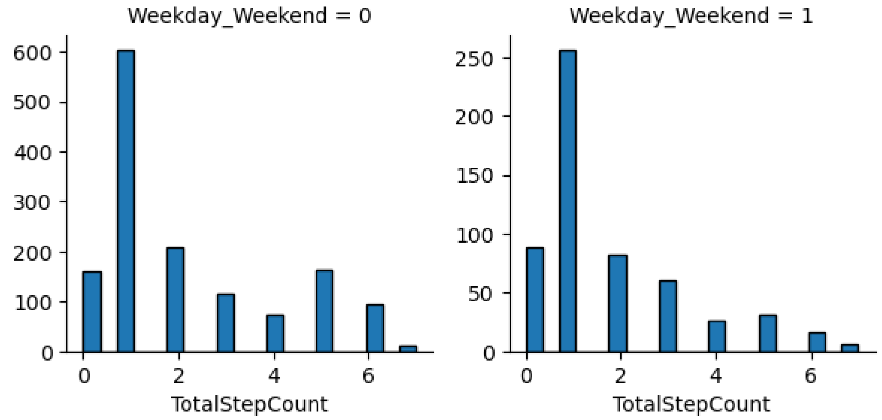



Histograms of Years
Year = 0
Year = 1
Year = 2
Year = 3
Year = 4
Year = 5
Year = 6
TotalStepCount
TotalStepCount
TotalStepCount
TotalStepCount


Season = 0
Season = 1
Season = 2
Season = 3
TotalStepCount
TotalStepCount
TotalStepCount
TotalStepCount

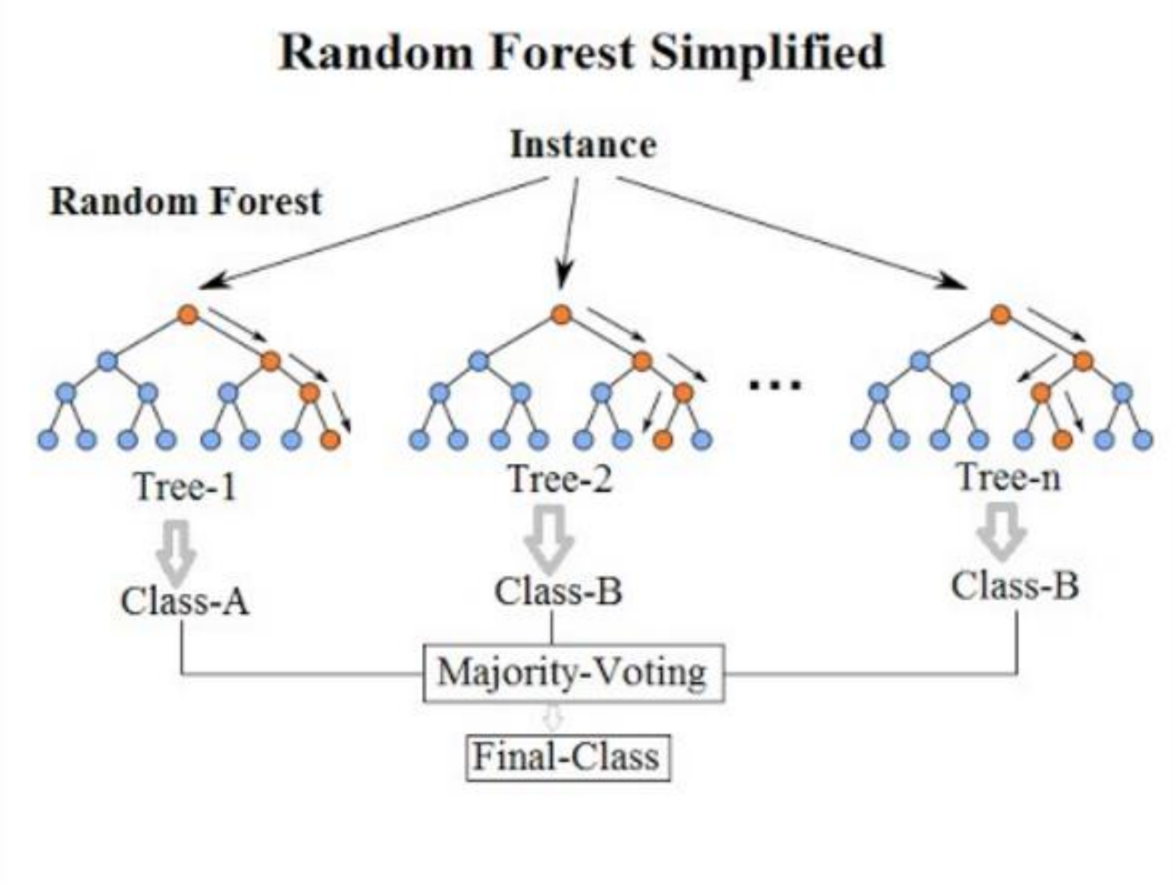

Histograms of Months
Month = 0
Month = 1
Month = 2
Month = 3
Month = 4
Month = 5
Month = 6
Month = 7
Month = 8
Month = 9
Month = 10
Month = 11
TotalStepCount


Day of the Week = 0
Day of the Week = 1
Day of the Week = 2
Day of the Week = 3
Day of the Week = 4
Day of the Week = 5
Day of the Week = 6
TotalStepCount
TotalStepCount
TotalStepCount
TotalStepCount


Weekday_Weekend = 0
Weekday_Weekend = 1
TotalStepCount
TotalStepCount

Machine Learning Model

- Using Random Forests
- Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time



Training

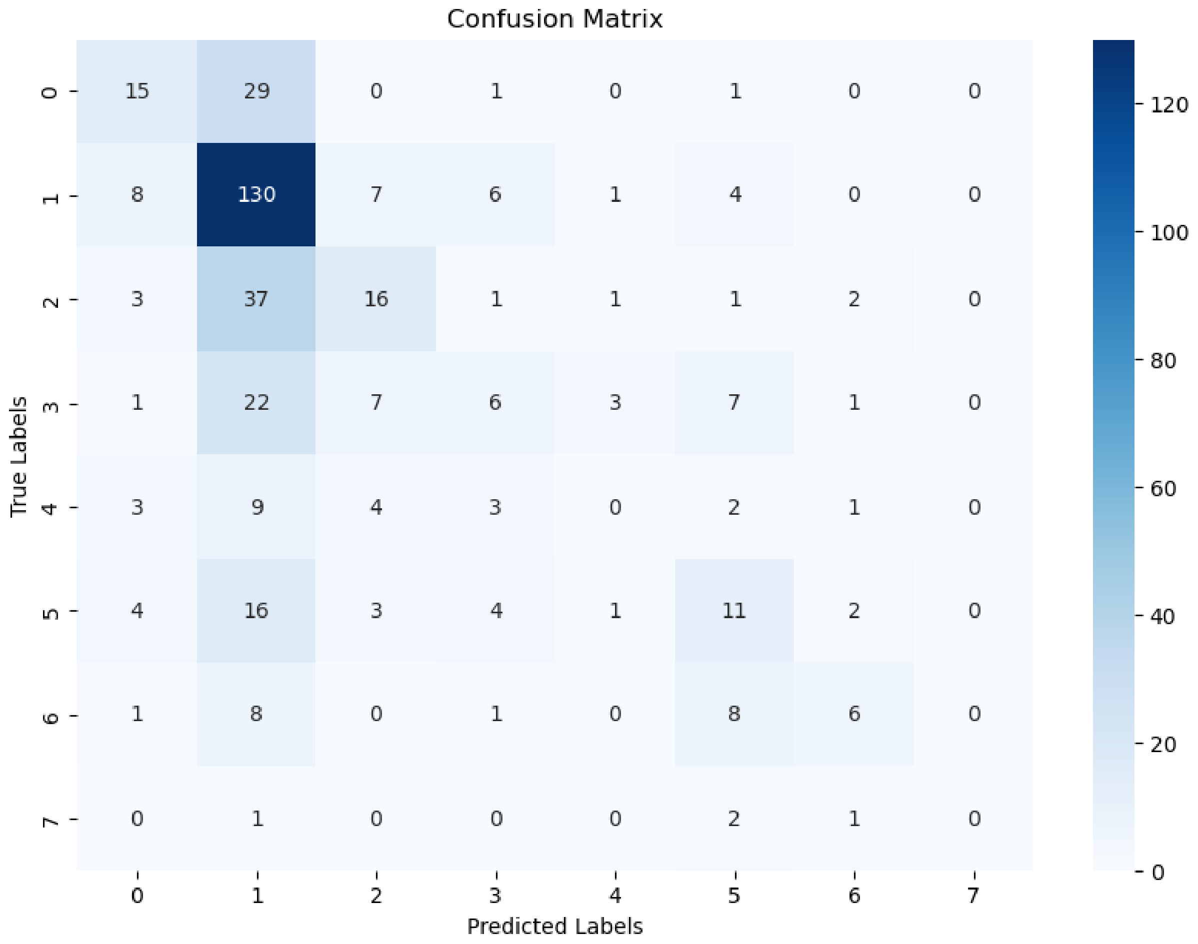
- %80 (1600 days) of the data set is used for training and the rest %20 (400 days) for testing
- Data set is shuffled before splitting
- Split between X and Y columns Y being the Total Step Count and X being rest of the features

```
# Initialize the RandomForestClassifier
rf = RandomForestClassifier(random_state=1)

# Define the parameter grid for RandomForest
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

# Use TimeSeriesSplit for cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Setup RandomizedSearchCV for the RandomForestClassifier
random_search = RandomizedSearchCV(rf, param_grid, n_iter=10, cv=tscv, scorin

# Fit the model
random_search.fit(X_train, y_train)

# Predict on the test set
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)
```

Machine learning results

	Feature	Importance
1	Day of the Month	0.323528
3	Year	0.207863
2	Month	0.193148
0	Day of the Week	0.160556
5	Season	0.066094
4	Weekday_Weekend	0.026833
6	Is_Last_Year	0.021978

- Overall %45 accuracy (decent)
- There seems to be a pattern between steps and date, we can predict step counts with some accuracy
- Heavily biased towards “1” bin
- Imbalanced data, most of step count is between 100 – 1000 steps per day
- Can we improve?

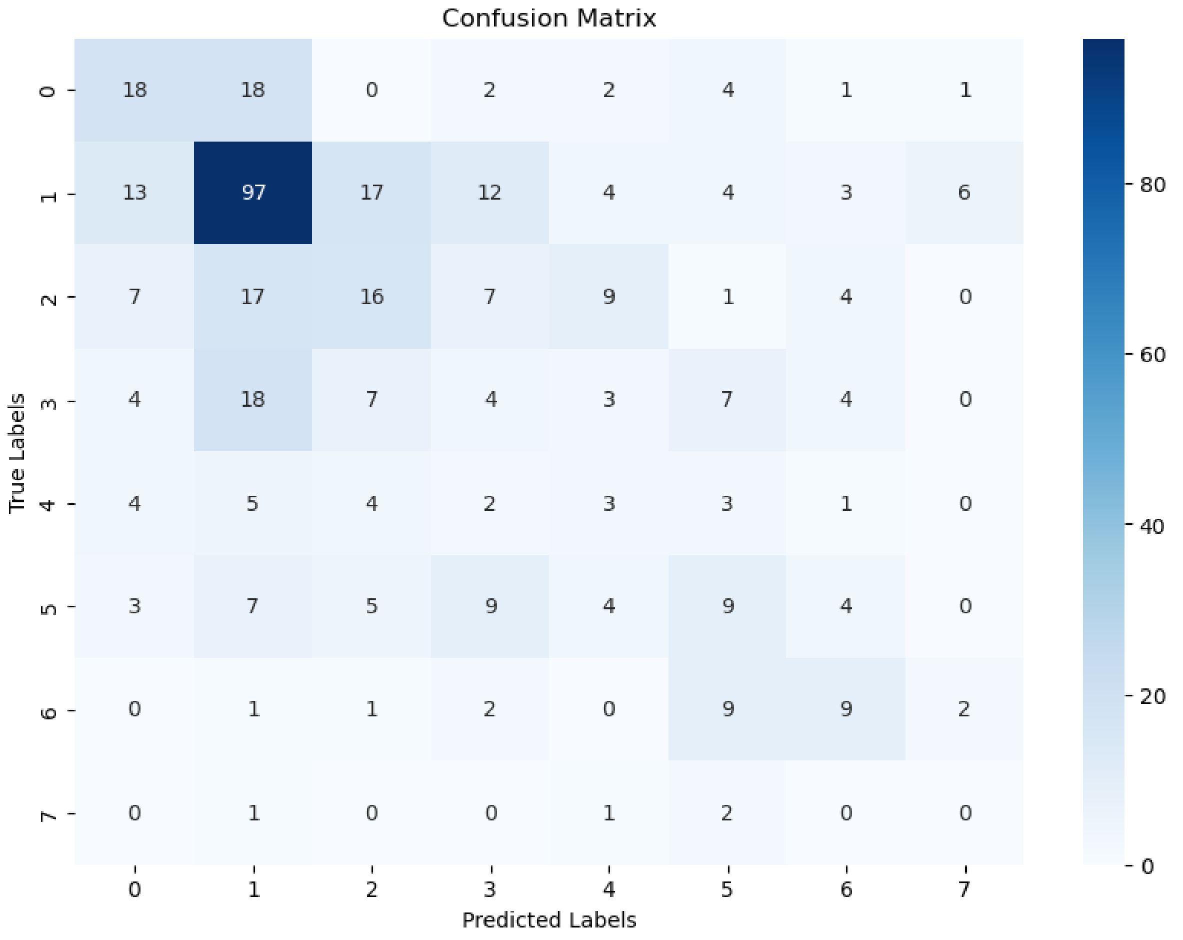



```
Accuracy: 0.45885286783042395, Precision: 0.40646318514148944, Recall: 0.45885286783042395, F1 Score: 0.4090751911007629
```

Weight balancing

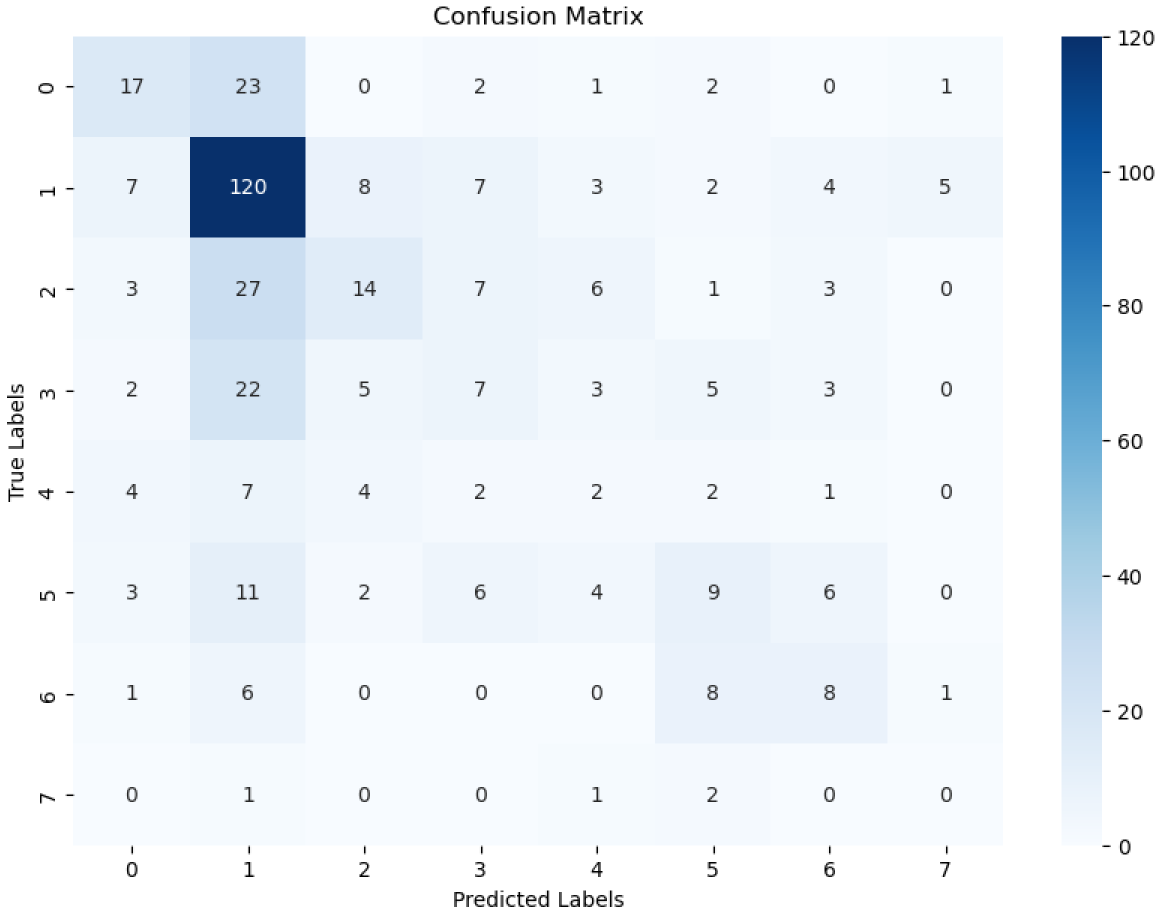
- We can give more weight to underrepresented classes
- Reduces the bias towards one bin
- Better accuracy for predicting bins other than “1”
- Overall reduced accuracy

```
Feature Importances:
            Feature  Importance
1  Day of the Month    0.389357
2             Month    0.179112
0   Day of the Week    0.167183
3              Year    0.161243
5            Season    0.060420
4   Weekday_Weekend    0.025255
6      Is_Last_Year    0.017429
```




```
Accuracy: 0.38902743142144636, Precision: 0.38389354761931 49, Recall: 0.38902743142144636, F1 Score: 0.3856127031040368
```

Finding a balanced Weight



Feature Importances:

	Feature	Importance
1	Day of the Month	0.365586
2	Month	0.182960
3	Year	0.171261
0	Day of the Week	0.168932
5	Season	0.065999
4	Weekday_Weekend	0.026003
6	Is_Last_Year	0.019259

Accuracy: 0.44139650872817954, Precision: 0.4131604318009274, Recall: 0.44139650872817954, F1 Score: 0.4139820239540894

Findings: What did we learn?

- there seems to be a pattern between steps I take everyday depending on the date
- Different seasons, weekdays, months and years have all different effects on my step count
- Although with not very high accuracy it is possible to make predictions on my step count based solely on date. Which is not bad considering we only know the date

Limitations and future work: What could be done better?

- Better predictions can be made provided with more data other just dates. For example; exam schedules, holidays
- Data can be filtered from exceptions to make better predictions. For example; one time when broke my foot had significant impact on my step count for 2 months and disrupts the overall pattern
- Machine learning can be improved with testing variety of different models and making comparisons.
- I could increase my overall step count