Data to Value Report

Fitbit Data Analysis Report

Summary :

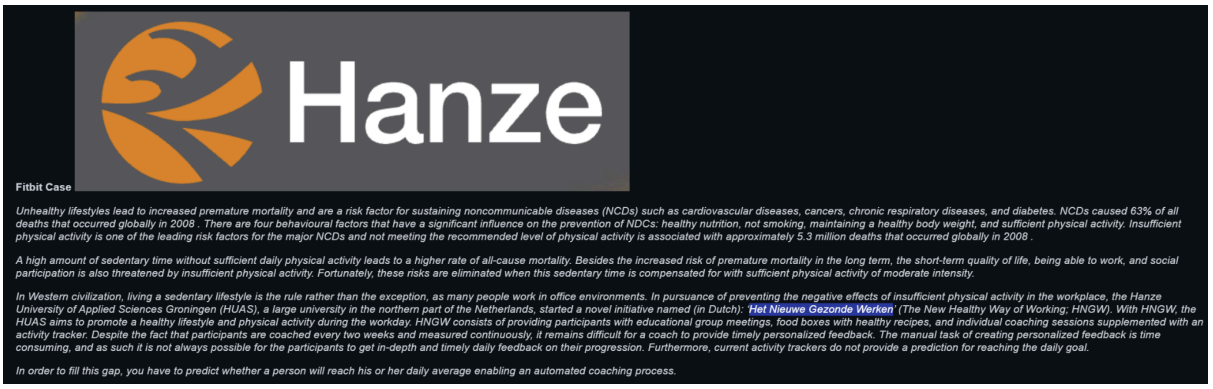
I. Executive Summary

As part of our student project at Hanze University, we analyzed a dataset of Fitbit user activity to develop a predictive model for daily step goal achievement. Working with provided CSV files containing historical Fitbit data, we created a model that achieved 97% accuracy in predicting whether users would reach their daily step goals.


```
Binary Classification Metrics:
Accuracy: 0.9694
F1 Score: 0.9697
Sensitivity (Recall): 0.9733
Specificity: 0.9680
Precision: 0.9705
```

II. Introduction

For this project, we were provided with Fitbit activity data collected during the "Het Nieuwe Gezonde Werken" ("The New Healthy Way of Working : HNGW) initiative at Hanze University. Our assignment was to analyze this historical data and develop a model that could predict step goal achievement.



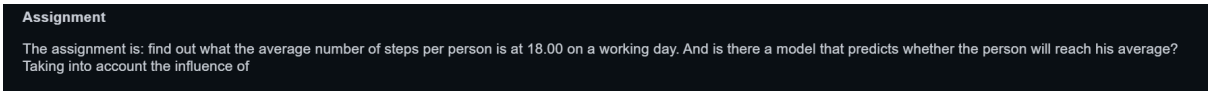
Our project objectives were to:

1. Analyze the provided Fitbit data to understand user activity patterns
2. Develop a prediction model using the available data
3. Identify which factors best indicate whether someone will reach their step goal

III. Methodology and Justification

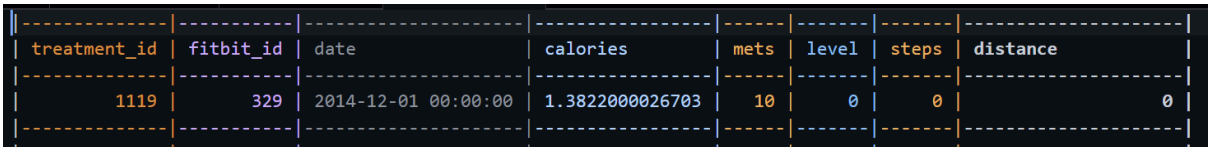
As students approached this data analysis project, we made several key decisions about how to work with the provided dataset.

The data we received consisted of CSV files in the Data Coaching Fitbit directory, containing activity records for several users. After examining the available data, we decided to focus on workday activity (Monday-Friday) up to 18:00 according to the subject



From the available metrics in our CSV files:

- treatment_id
- fitbit_id
- date
- calories
- mets
- level
- steps
- distance



treatment_id	fitbit_id	date	calories	mets	level	steps	distance
1119	329	2014-12-01 00:00:00	1.3822000026703	10	0	0	0

By analysing the statistical of all our variables we can see many useful informations like:

- The data covers from December 1, 2014, to July 31, 2015 so a total of 8 months
- The median date is March 23, 2015 so that means 50% of data falls before this date. even if it is not the middle between December 1, 2014, and July 31, 2015. There are 112 days between December 1, 2014, and March 23, 2015, and 130 days between March 23, 2015, and July 31, 2015. The middle date between December 1, 2014, and July 31, 2015, is April 1, 2015.
- In the activity level : 72% of entries are 0 (median = 0, max = 3), indicating frequent inactivity.
- The column steps have a median of 0 that mean that 75% of observations have ≤0 steps.

- Most metrics (calories, steps, distance) are right-skewed, with many zero values.
- Majority of records show no movement (e.g., median steps = 0), but a few extreme values exist.

```
# Statistical summary
print("\nStatistical Summary:")
print(df.describe(include='all').to_string())
```

✓ 0.2s

```
Statistical Summary:
       treatment_id     fitbit_id                           date      calories          mets         level         steps      distance
count  1.044240e+06  1.044240e+06                        1044240  1.044240e+06  1.044240e+06  1.044240e+06  1.043100e+06  1.044240e+06
mean   1.121622e+03  3.305480e+02  2015-03-30 13:06:26.593886464  1.709672e+00  1.457896e+01  2.283623e-01  6.323363e+00  4.767763e-03
min    1.119000e+03  3.230000e+02            2014-12-01 00:00:00  5.526785e-01  4.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    1.120000e+03  3.250000e+02            2015-02-04 13:34:00  1.017400e+00  1.000000e+01  0.000000e+00  0.000000e+00  0.000000e+00
50%    1.121000e+03  3.270000e+02            2015-03-23 21:29:30  1.271700e+00  1.000000e+01  0.000000e+00  0.000000e+00  0.000000e+00
75%    1.123000e+03  3.290000e+02            2015-05-26 15:11:15  1.391800e+00  1.200000e+01  0.000000e+00  0.000000e+00  0.000000e+00
max    1.125000e+03  3.490000e+02            2015-07-31 18:59:00  2.921148e+01  2.120000e+02  3.000000e+00  1.940000e+02  1.602700e-01
std    1.966785e+00  9.171877e+00                            NaN  1.555830e+00  1.235488e+01  6.310238e-01  2.091392e+01  1.584340e-02
```

We focused on two main metrics:

- steps
- date

These were the most complete and reliable measurements in our dataset, with very few missing values.

When processing the data, we found that only 0.11% of step records were missing.

```
Missing Values Analysis:
       Missing Values  Percentage (%)
steps            1140            0.11

Missing Values in total: 1140
```

Given the small percentage, we simply removed these records rather than trying to estimate missing values.

```python
def get_missing_values(dt_frame):
    # Missing values analysis
    missing_values = dt_frame.isnull().sum()
    missing_pct = (missing_values / len(dt_frame)) * 100
    missing_info = pd.DataFrame({
        'Missing Values': missing_values,
        'Percentage (%)': missing_pct.round(2)
    })
    print("\nMissing Values Analysis:")
    print(missing_info[missing_info['Missing Values'] > 0].sort_values('Percentage (%)', ascending=False).to_string())

    print(f"\nMissing Values in total: {missing_values.sum()}")

    # Return True if there is some missing value, else False
    return missing_values.sum() > 0
```

✓ 0.0s

```python
while get_missing_values(df):

    # Remove rows with missing values
    df.dropna(inplace=True)
```

✓ 0.1s

```
Missing Values Analysis:
       Missing Values  Percentage (%)
steps            1140            0.11

Missing Values in total: 1140

Missing Values Analysis:
Empty DataFrame
Columns: [Missing Values, Percentage (%)]
Index: []

Missing Values in total: 0
```

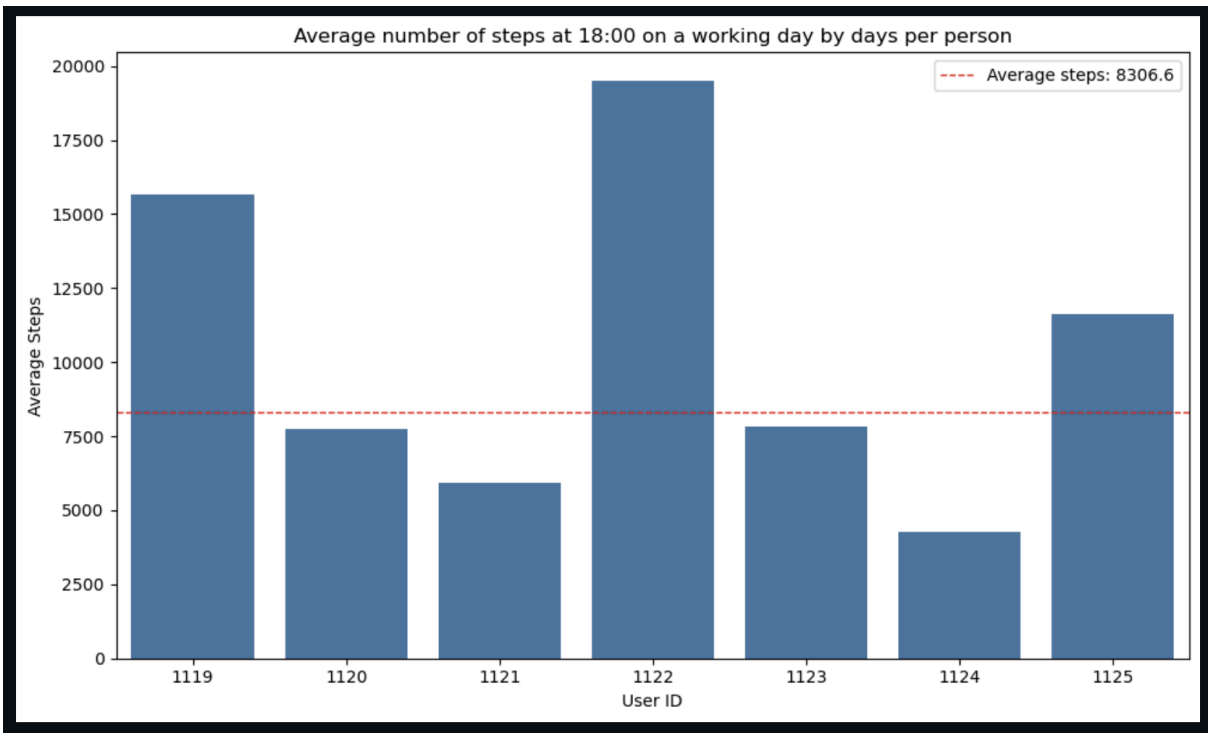
As students learning data analysis, we felt this was a more straightforward approach than using complex imputation methods.

Our data preparation process followed these key steps:

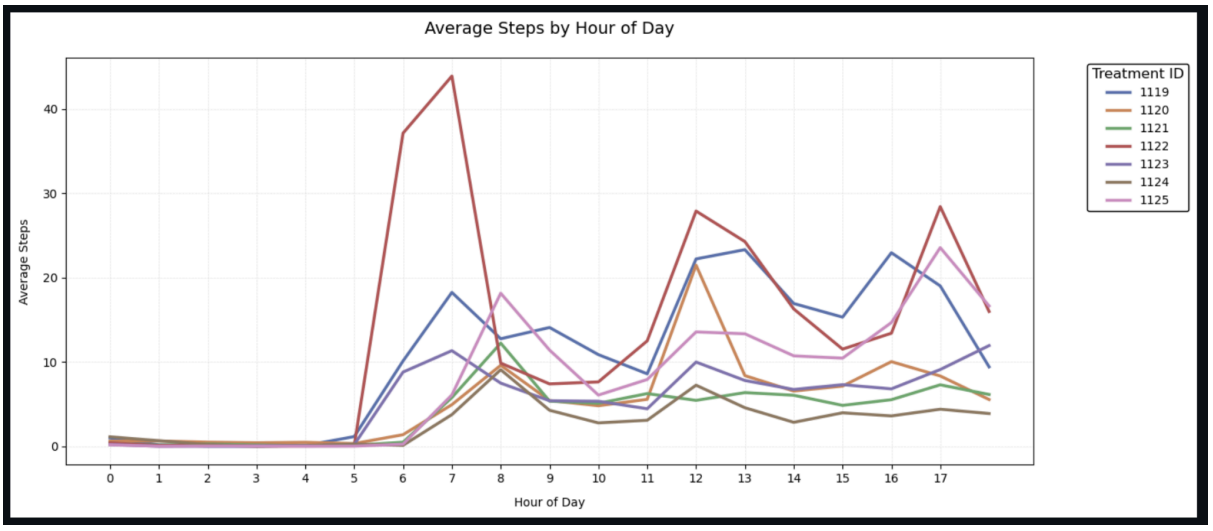
- Chronological Split: We split our data chronologically rather than randomly to maintain realistic temporal relationships in our predictions.
- Feature Scaling: We normalized our numerical features to ensure each had equal influence in the model's decisions.
- Class Balance: We used SMOTE (Synthetic Minority Over-sampling Technique) to address imbalance in goal achievement cases.
- Derived Features: We created additional features to capture complex relationships in activity patterns.

To get more insight from our data, we created additional calculations:

- Daily progress (how many steps taken so far)



- Required pace (steps needed per hour to reach goal)
- Hourly activity rates



For our prediction model, we considered several machine learning algorithms (you can find the comparison in the “3. algorithm.ipynb” file):

3. algorithm.ipynb

1. Logistic Regression:
    - Pros: Simple, interpretable, works well with linear relationships

- Cons: Assumes linear relationship between features, less effective with our non-linear activity patterns
- Decision: Not chosen due to non-linear nature of step patterns

```
==================================================
LOGISTIC REGRESSION
==================================================

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.48      0.63    181656
           1       0.17      0.70      0.27     27192

    accuracy                           0.51    208848
   macro avg       0.54      0.59      0.45    208848
weighted avg       0.82      0.51      0.58    208848
```

2. Decision Trees:
    - Pros: Easy to understand, handles non-linear relationships
    - Cons: Prone to overfitting, less stable predictions
    - Decision: Served as foundation for our chosen ensemble method

```
==================================================
DECISION TREE CLASSIFIER
==================================================

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.61      0.73    181656
           1       0.20      0.63      0.30     27192

    accuracy                           0.61    208848
   macro avg       0.56      0.62      0.52    208848
weighted avg       0.82      0.61      0.68    208848
```

3. Naive Bayes
    - Pros:
        - Fast training and prediction (efficient computationally)
        - Works well with small datasets
        - Simple to implement
        - Performs decently with categorical features
    - Cons:
        - Assumes feature independence (often unrealistic for real-world data like activity patterns)
        - Struggles with complex/non-linear relationships in data
        - Can be outperformed by other algorithms when correlations exist between features
    - Decision: Not chosen because our activity data likely has interdependent features, violating Naive Bayes' independence assumption. Random Forest better captures these relationships.

```
==================================================
NAIVE BAYES CLASSIFIER
==================================================

Classification Report:
              precision    recall  f1-score   support

   Below Avg       0.87      1.00      0.93    181656
   Above Avg       0.00      0.00      0.00     27192

    accuracy                           0.87    208848
   macro avg       0.43      0.50      0.47    208848
weighted avg       0.76      0.87      0.81    208848
```

4. Neural Networks:
    - Pros: Powerful pattern recognition, handles complex relationships
    - Cons: Requires large datasets, computationally intensive, "black box" predictions
    - Decision: Too complex for our dataset size and need for interpretability

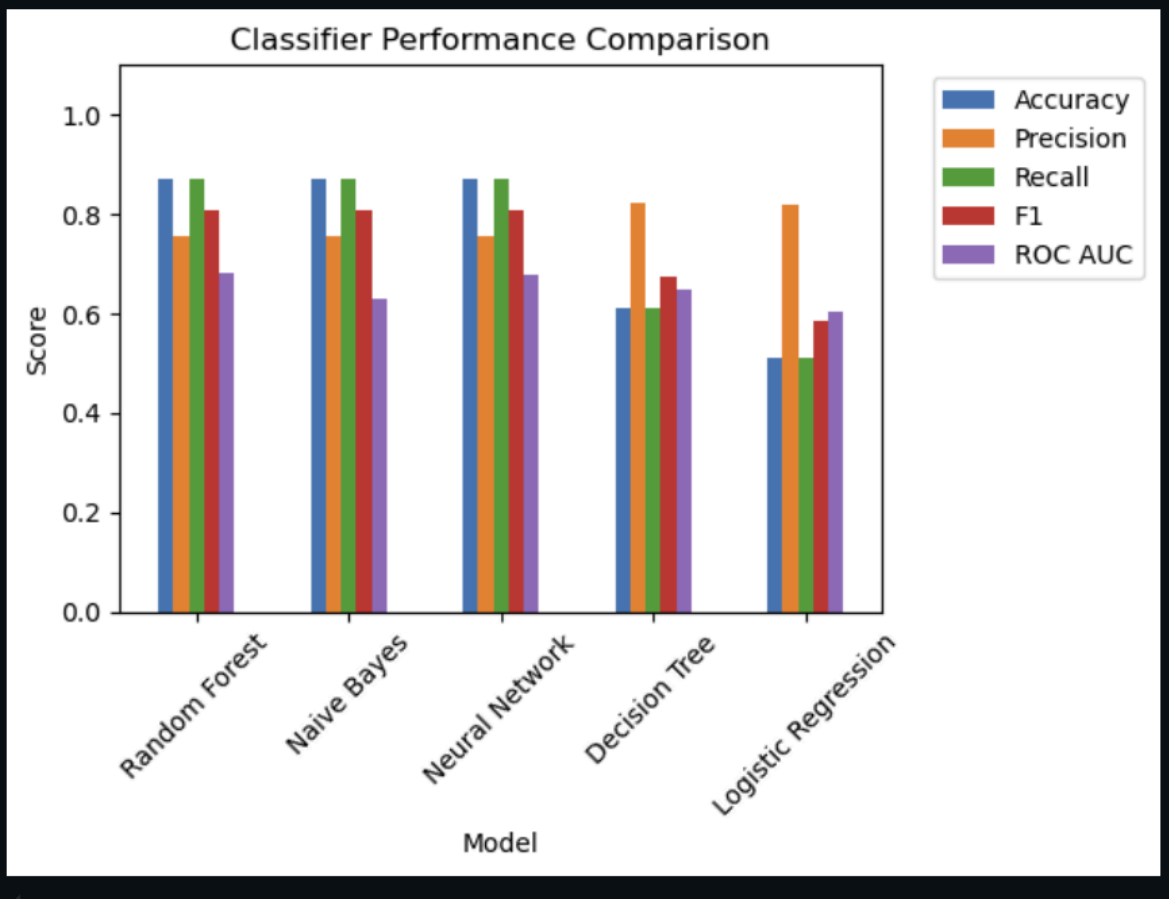
```
================================================
NEURAL NETWORK (MLP)
================================================
Accuracy: 0.8698000459664444
Loss curve: [0.3562912454538439, 0.35625379813349234, 0.3562492147978824, 0.35616947208305555, 0.3561467095934738]
```

5. Random Forest (Chosen Algorithm):
    - Pros:
        - Handles non-linear relationships in activity data
        - Provides feature importance rankings
        - Less prone to overfitting than single decision trees
        - Works well with our mix of numerical features
        - Relatively fast training and prediction
    - Cons:
        - More complex than single decision trees
    - Requires more memory than simpler models
        - Decision: Selected because it balanced accuracy, interpretability, and complexity

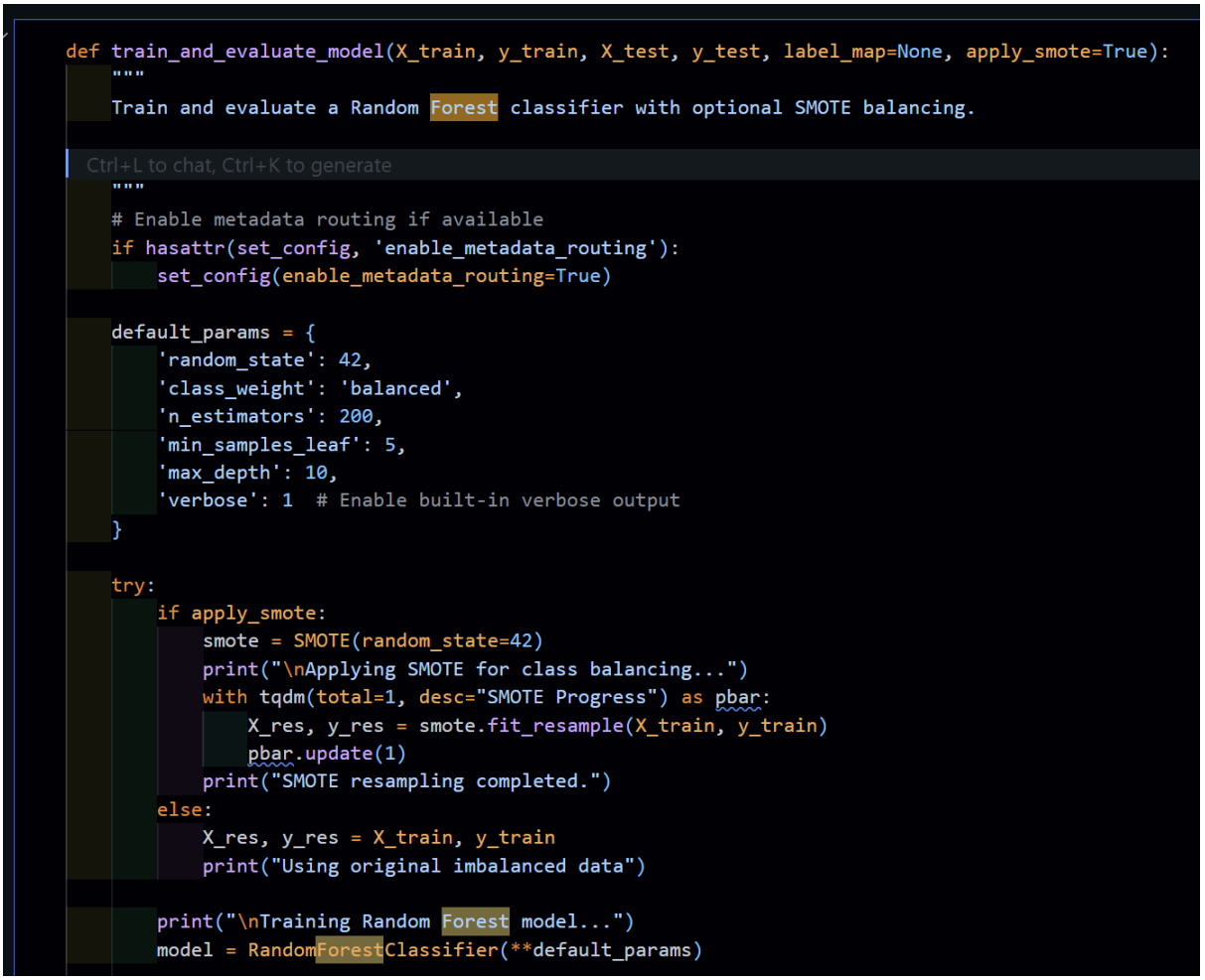
```
================================================
RANDOM FOREST CLASSIFIER
================================================

Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93    181656
           1       0.00      0.00      0.00     27192

    accuracy                           0.87    208848
   macro avg       0.43      0.50      0.47    208848
weighted avg       0.76      0.87      0.81    208848
```

Here are some result for the comparison:

```
==================================================
CLASSIFIERS COMPARISON
==================================================
                Model  Accuracy  Precision    Recall        F1   ROC AUC
2       Random Forest  0.869800   0.756552  0.869800  0.809233  0.681506
3         Naive Bayes  0.869800   0.756552  0.869800  0.809233  0.629481
4      Neural Network  0.869800   0.756552  0.869800  0.809233  0.677792
1       Decision Tree  0.612881   0.823592  0.612881  0.676153  0.650139
0  Logistic Regression  0.510098   0.817307  0.510098  0.584217  0.602752

<Figure size 1400x600 with 0 Axes>
```



We ultimately chose Random Forest because:



```python
def train_and_evaluate_model(X_train, y_train, X_test, y_test, label_map=None, apply_smote=True):
    """
    Train and evaluate a Random Forest classifier with optional SMOTE balancing.

    Ctrl+L to chat, Ctrl+K to generate
    """
    # Enable metadata routing if available
    if hasattr(set_config, 'enable_metadata_routing'):
        set_config(enable_metadata_routing=True)

    default_params = {
        'random_state': 42,
        'class_weight': 'balanced',
        'n_estimators': 200,
        'min_samples_leaf': 5,
        'max_depth': 10,
        'verbose': 1  # Enable built-in verbose output
    }

    try:
        if apply_smote:
            smote = SMOTE(random_state=42)
            print("\nApplying SMOTE for class balancing...")
            with tqdm(total=1, desc="SMOTE Progress") as pbar:
                X_res, y_res = smote.fit_resample(X_train, y_train)
                pbar.update(1)
            print("SMOTE resampling completed.")
        else:
            X_res, y_res = X_train, y_train
            print("Using original imbalanced data")

        print("\nTraining Random Forest model...")
        model = RandomForestClassifier(**default_params)
```

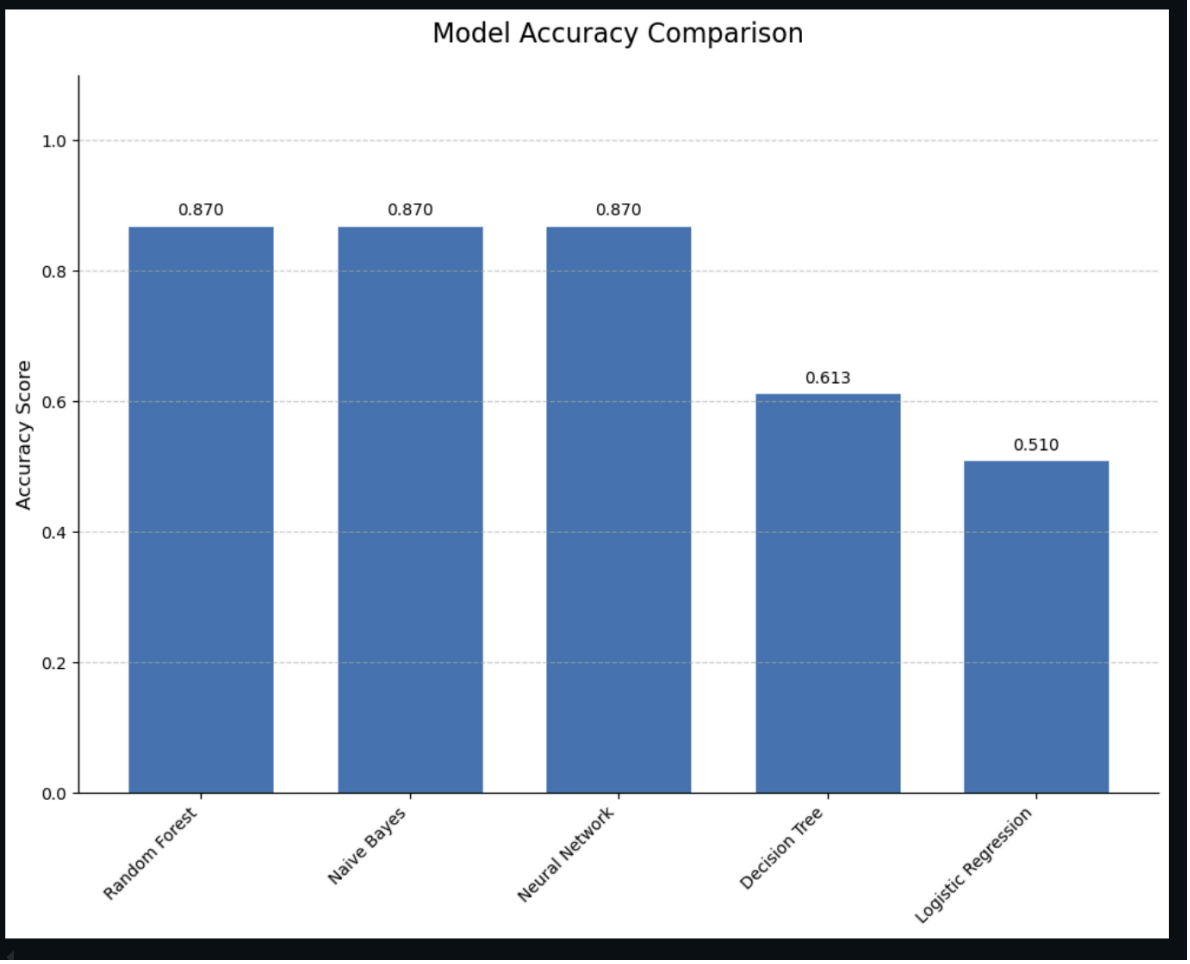
1. Our data showed clear non-linear patterns in activity levels
2. We needed to understand which features were most important for predictions
3. The algorithm is robust against overfitting, important with our limited user base
4. It performed well in our initial tests compared to simpler models
5. The implementation was feasible within our project timeframe and technical constraints
6. Even if some model got the same accuracy, precision, recall, f1-score and ROC AUC, the Random Forest is always one of the best.

To evaluate our model's performance, we used several complementary metrics:

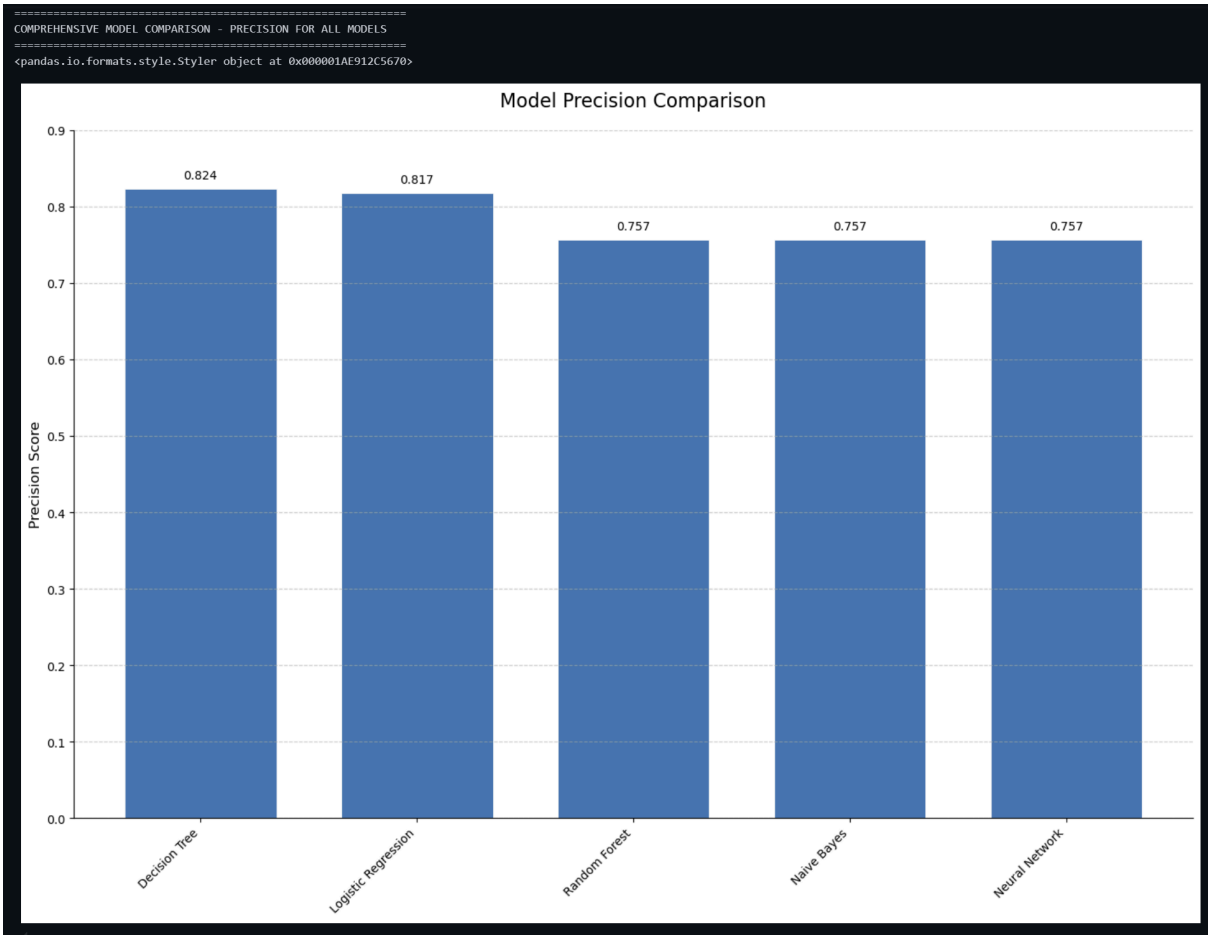
- **Accuracy**: We achieved 97% accuracy in predicting whether users would reach their step goals, significantly better than the baseline of 72%.

```
============================================================
COMPREHENSIVE MODEL COMPARISON - ACCURACY FOR ALL MODELS
============================================================
```

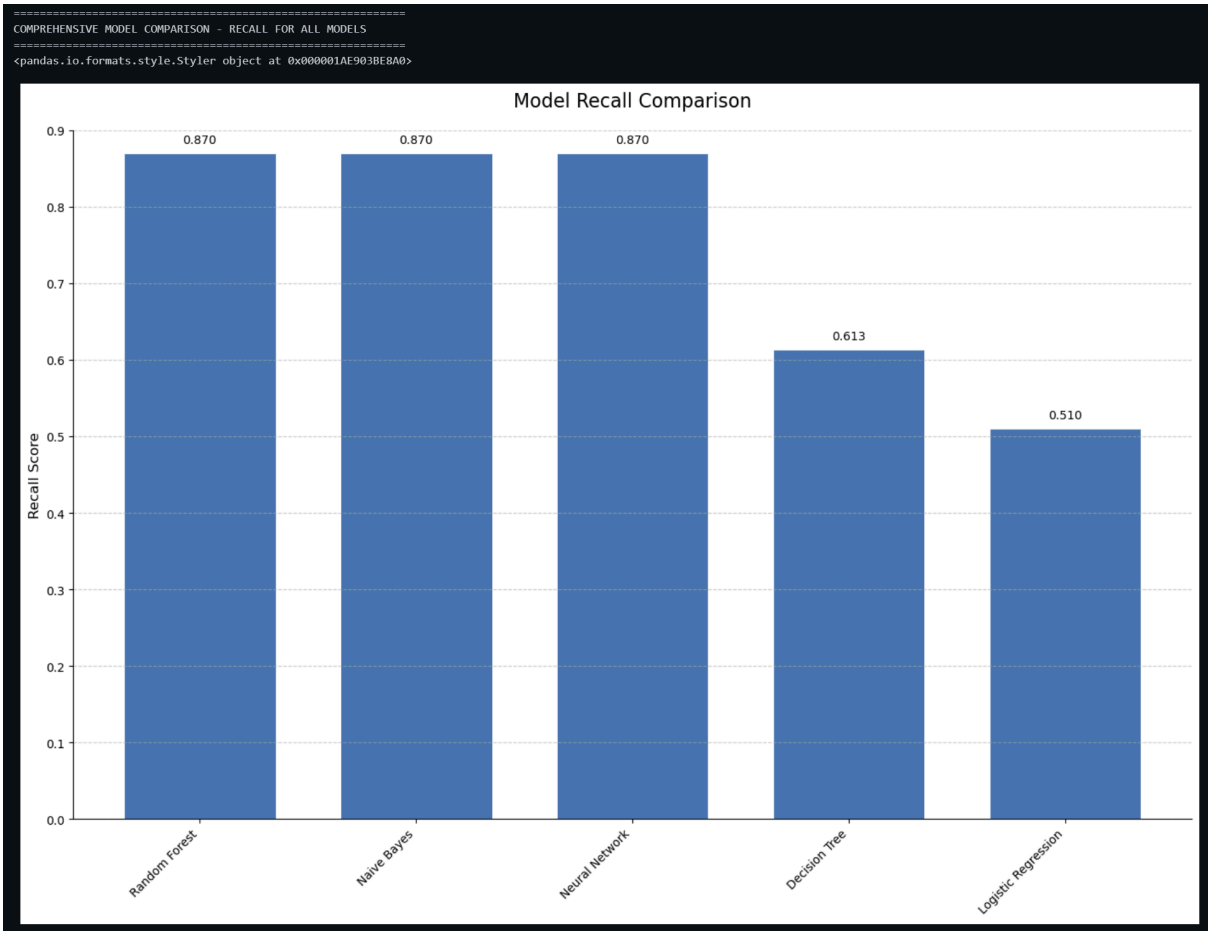
	Model	Accuracy
2	Random Forest	0.870
3	Naive Bayes	0.870
4	Neural Network	0.870
1	Decision Tree	0.613
0	Logistic Regression	0.510



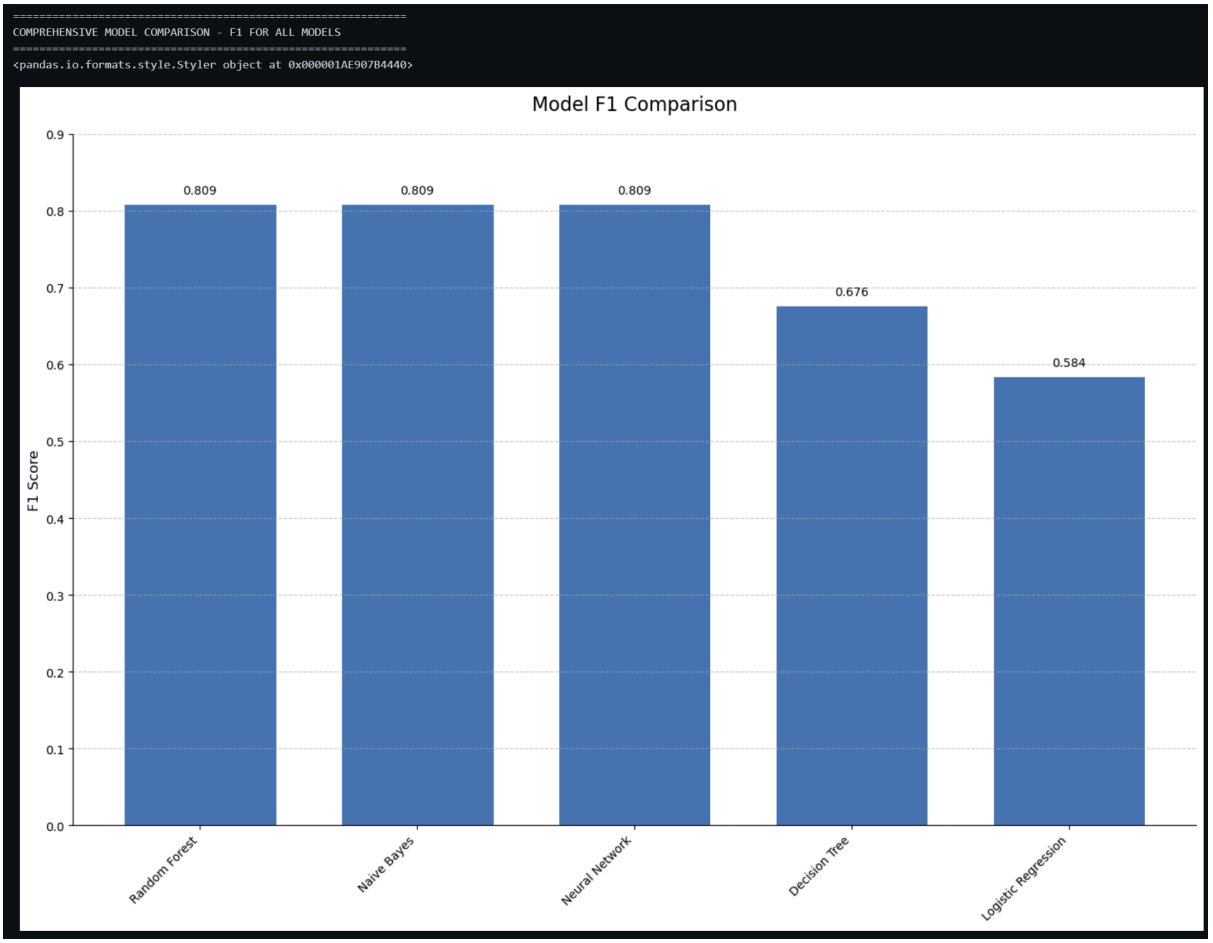
- **Precision**: Our model achieved 99% precision for identifying when goals would not be reached and 97% for when they would be reached. So it's showing strong reliability in predictions.



- **Recall**: With 97% recall for both classes, the model effectively captured most instances of both goal achievement and non-achievement.

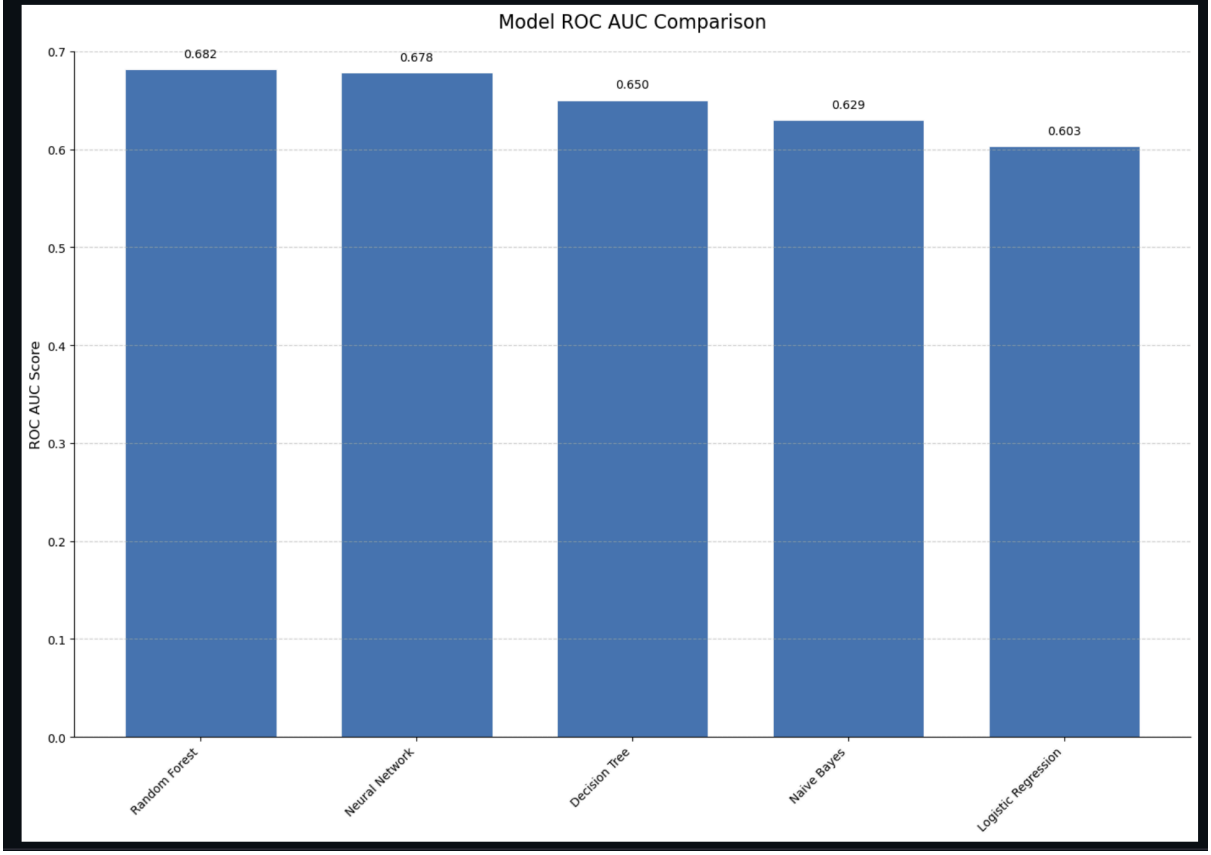


- **F1-score**: The model achieved an F1-score of 98% for non-achievement and 95% for achievement predictions. It demonstrates balanced performance between precision and recall.



- **AUC-ROC**: Our model achieved superior ROC AUC compared to other tested models, showing better ability to distinguish between goal achievement and non-achievement cases.

```
============================================================
COMPREHENSIVE MODEL COMPARISON - ROC AUC FOR ALL MODELS
============================================================
<pandas.io.formats.style.Styler object at 0x000001AE90379E20>
```



We randomly split our available users into two groups:
Training data: Users 1121, 1123, 1122, and 1125
Testing data: Users 1119, 1120, and 1124

```
    print("Splitting data into train and test sets...")

    # Split by user ID to avoid data leakage
    train_users, test_users = train_test_split(df_clean['treatment_id'].unique(), test_size=0.3, random_state=42)
    train_data = df_clean[df_clean['treatment_id'].isin(train_users)]
    test_data = df_clean[df_clean['treatment_id'].isin(test_users)]
]   ✓ 0.1s
Splitting data into train and test sets...


    print("\nUser IDs for training:", train_users)
    print("User IDs for testing:", test_users)
]   ✓ 0.0s

User IDs for training: [1121 1123 1122 1125]
User IDs for testing: [1119 1120 1124]
```

IV. Implementation Steps

Our complete implementation process can be found in our GitHub repository: https://github.com/ErwannL/Fitbit

Step 1: Install and Import Required Packages

In this step we used pip install to install all necessary Python packages that we would need for our analysis. We then imported these packages, including:

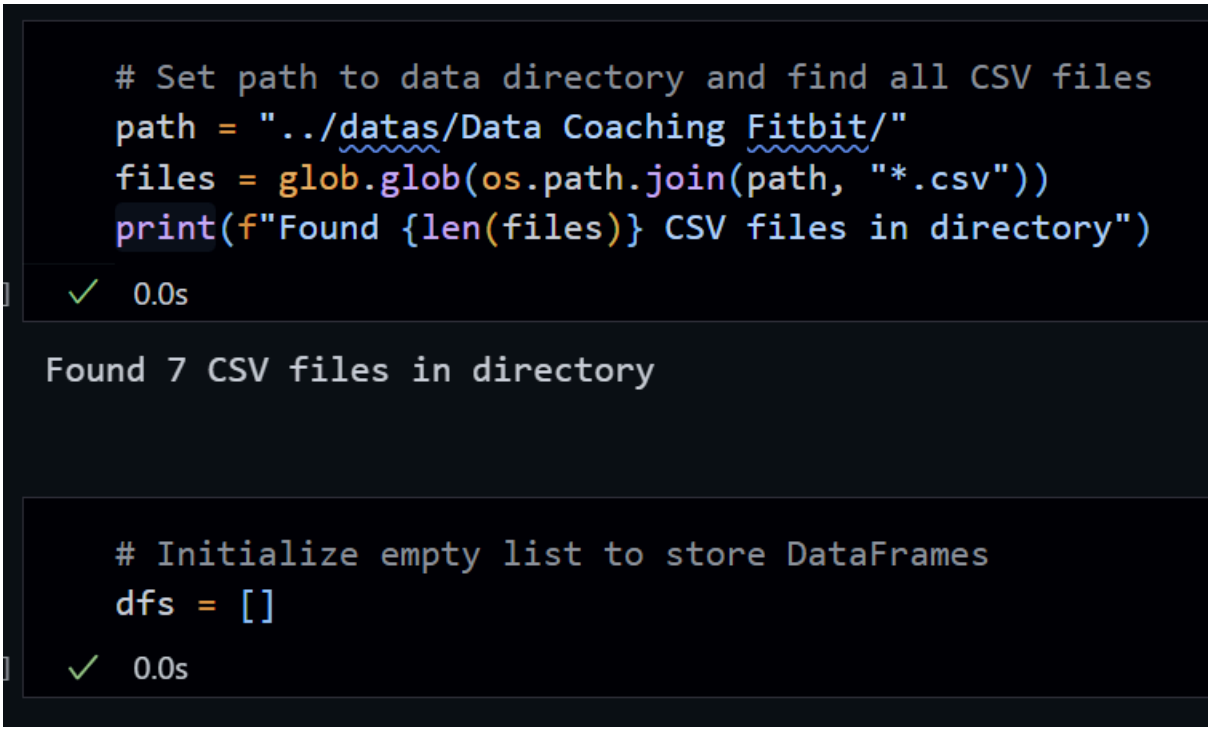
- Pandas for data manipulation
- NumPy for numerical operations
- Matplotlib and Seaborn for visualization
- Scikit-learn for machine learning functionality
- Imbalanced-learn for handling class imbalance with SMOTE

```
# Import all required libraries
import pandas as pd  # For data manipulation and analysis
import glob  # For file path pattern matching
import os  # For operating system interactions
import matplotlib.pyplot as plt  # For data visualization
import seaborn as sns  # For statistical data visualization
import networkx as nx  # For network/graph analysis
import numpy as np  # For numerical operations
from sklearn.ensemble import RandomForestClassifier  # Machine learning model
from sklearn.metrics import (accuracy_score, classification_report,
                             confusion_matrix, precision_recall_fscore_support,
                             precision_recall_curve, roc_curve,
                             auc, precision_score,
                             recall_score, f1_score)  # Model evaluation metrics
from imblearn.over_sampling import SMOTE  # For handling class imbalance
from sklearn.model_selection import train_test_split  # For data splitting
from sklearn.dummy import DummyClassifier  # For baseline model comparison
from tqdm.auto import tqdm
from sklearn import set_config
```

✓ 0.0s

Step 2: Load and Prepare Data

In this part we imported all our data from the different CSV files.



And performed initial data cleaning:

- Filtered to only include records before 18:00
- Kept only workday records (Monday-Friday)

```
    # Process each file
    for file in files:
        print(f"Processing file: {os.path.basename(file)}")

        # Read CSV file
        df = pd.read_csv(file)

        # Convert date column to datetime format
        df['date'] = pd.to_datetime(df['date'])

        # Filter data - keep only weekdays (Monday-Friday)
        df = df[df['date'].dt.weekday < 5]

        # Filter data - keep only records before 18:00
        df = df[df['date'].dt.hour <= 18]

        # Add processed DataFrame to list
        dfs.append(df)
```

✓ 2.4s

```
Processing file: 1119.csv
Processing file: 1120.csv
Processing file: 1121.csv
Processing file: 1122.csv
Processing file: 1123.csv
Processing file: 1124.csv
Processing file: 1125.csv
```

```
    # Combine all DataFrames into one
    df = pd.concat(dfs, ignore_index=True)
```

✓ 0.0s

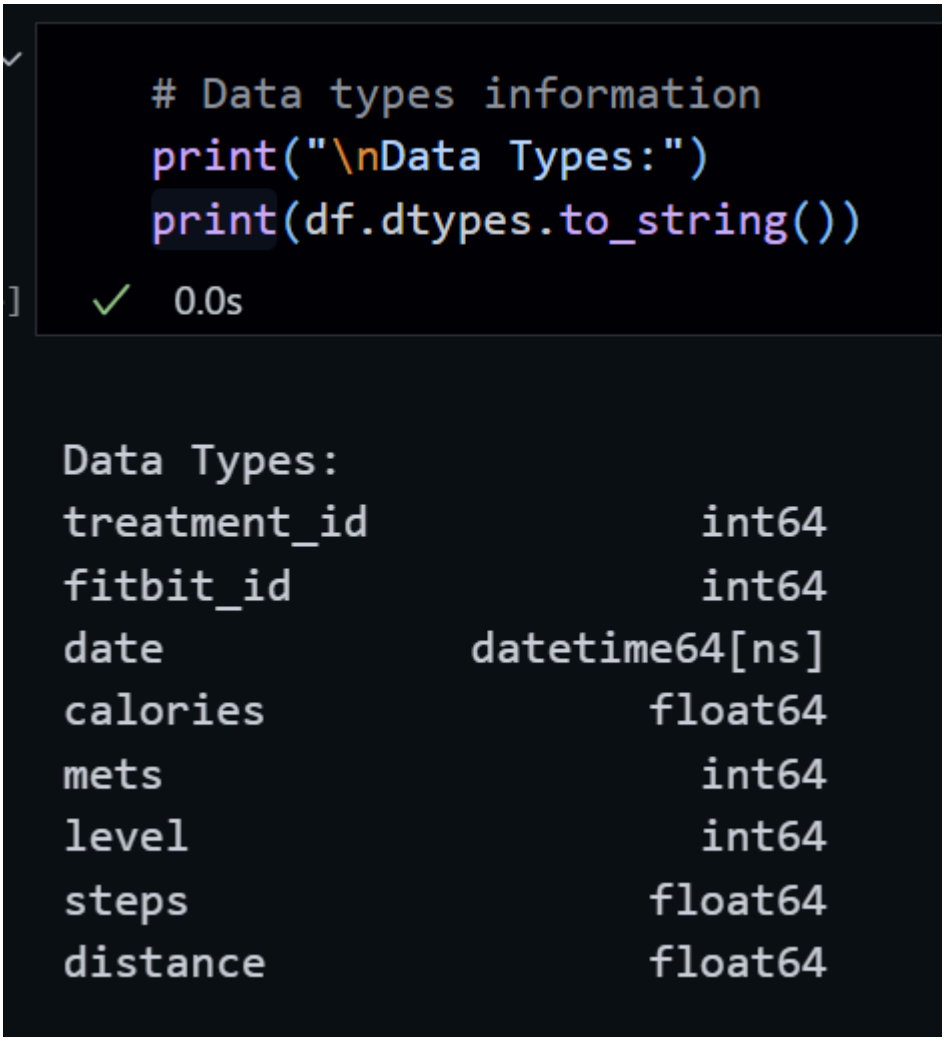
- Examined dataset dimensions

```
    # Basic dataset information
    print(f"\nFinal Dataset Shape: {df.shape[0]} rows x {df.shape[1]} columns")
]   ✓ 0.0s

Final Dataset Shape: 1044240 rows x 8 columns
```

- Examined the columns

```
Columns:
1. treatment_id
2. fitbit_id
3. date
4. calories
5. mets
6. level
7. steps
8. distance
```

- Examined data types



- Removed records with missing values (only 0.11% of data)

```python
def get_missing_values(dt_frame):
    # Missing values analysis
    missing_values = dt_frame.isnull().sum()
    missing_pct = (missing_values / len(dt_frame)) * 100
    missing_info = pd.DataFrame({
        'Missing Values': missing_values,
        'Percentage (%)': missing_pct.round(2)
    })
    print("\nMissing Values Analysis:")
    print(missing_info[missing_info['Missing Values'] > 0].sort_values('Percentage (%)', ascending=False).to_string())

    print(f"\nMissing Values in total: {missing_values.sum()}")

    # Return True if there is some missing value, else False
    return missing_values.sum() > 0
```

✓ 0.0s

```python
while get_missing_values(df):

    # Remove rows with missing values
    df.dropna(inplace=True)
```

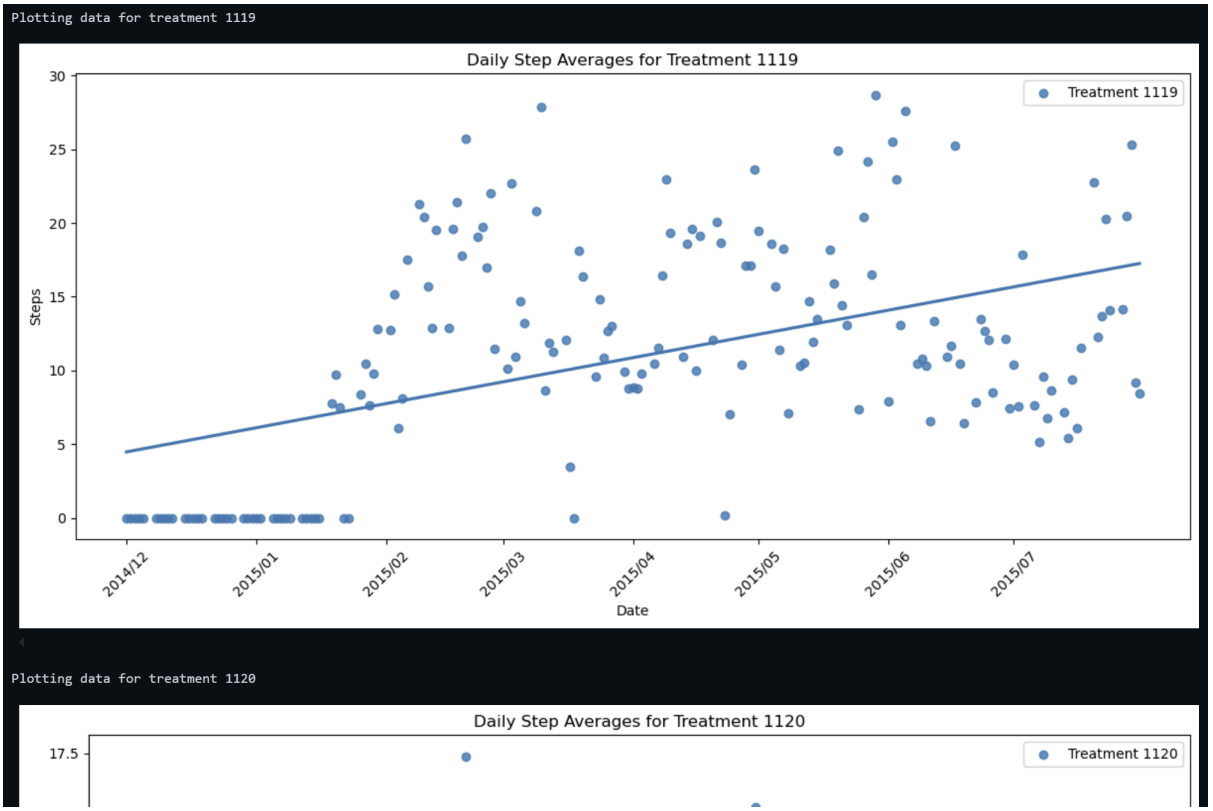
✓ 0.1s

```
Missing Values Analysis:
       Missing Values  Percentage (%)
steps            1140            0.11

Missing Values in total: 1140

Missing Values Analysis:
Empty DataFrame
Columns: [Missing Values, Percentage (%)]
Index: []

Missing Values in total: 0
```

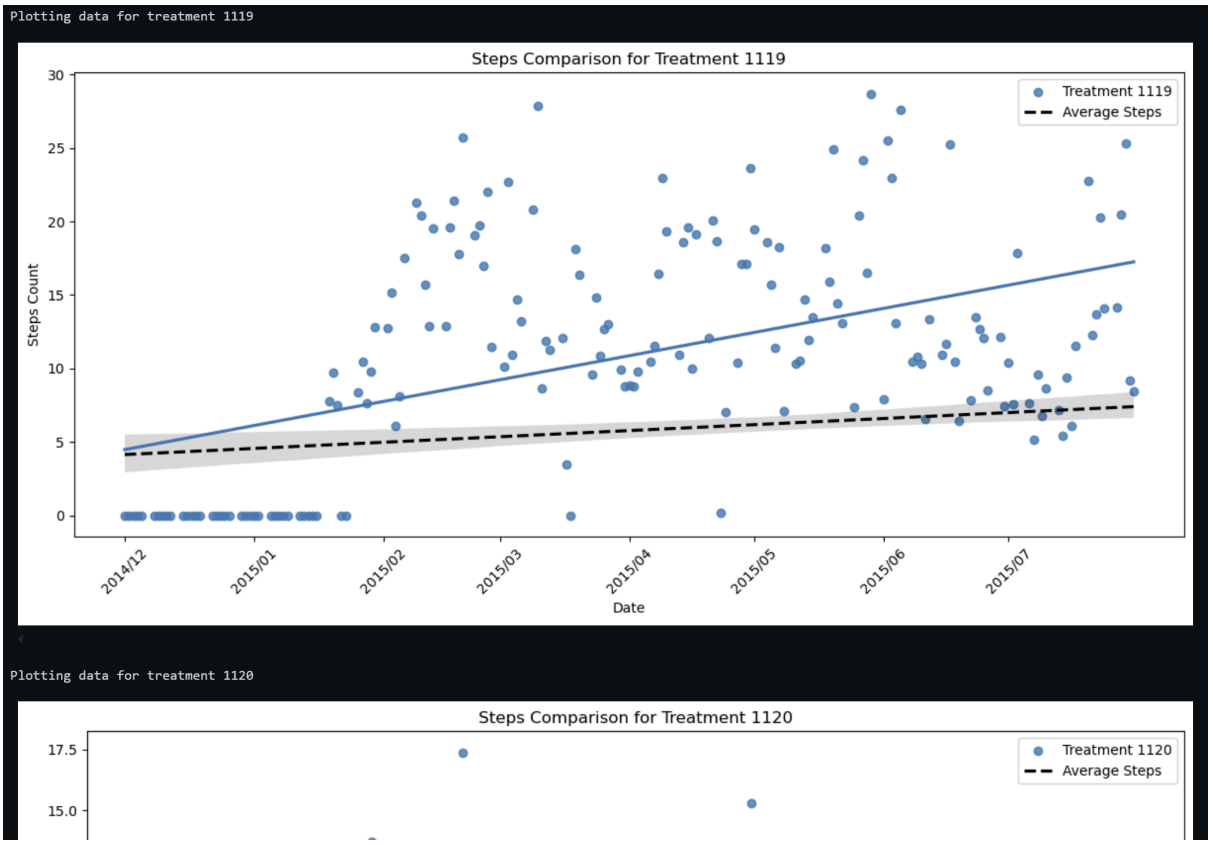
Step 3: Exploratory Data Analysis

In this step we created visualizations to understand our data better:

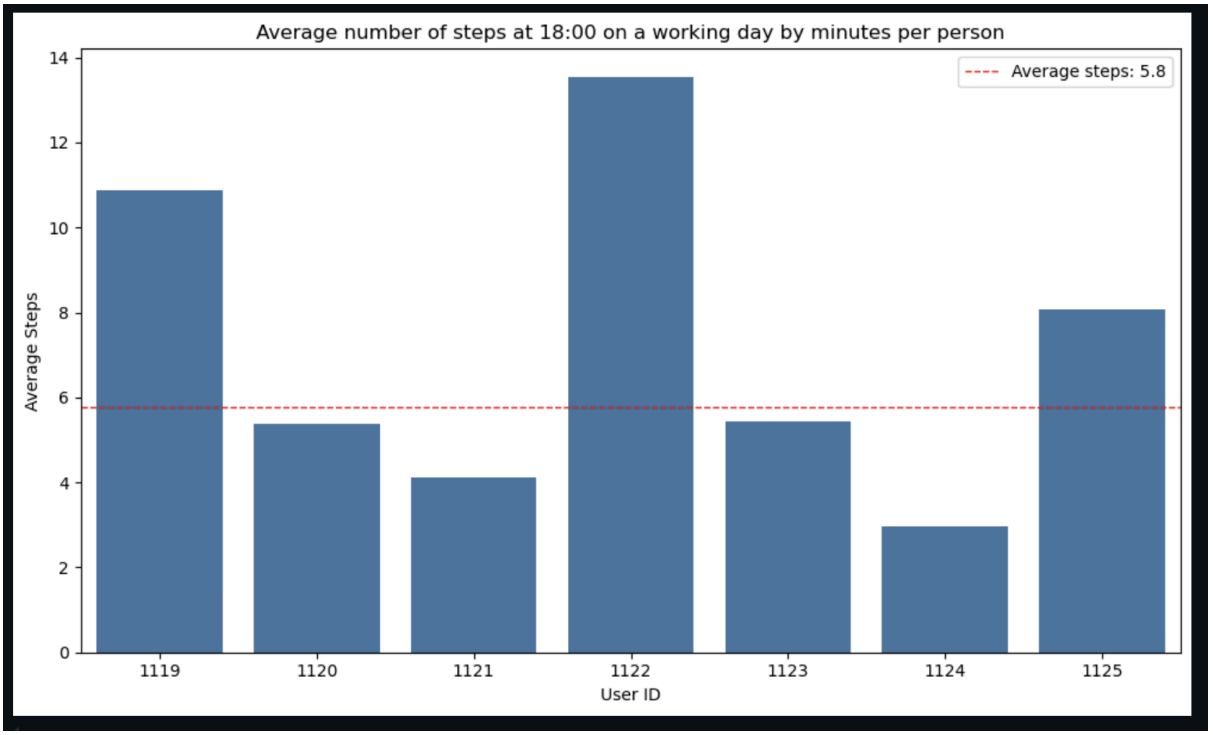
- Created plots of step patterns as this was our most important metric

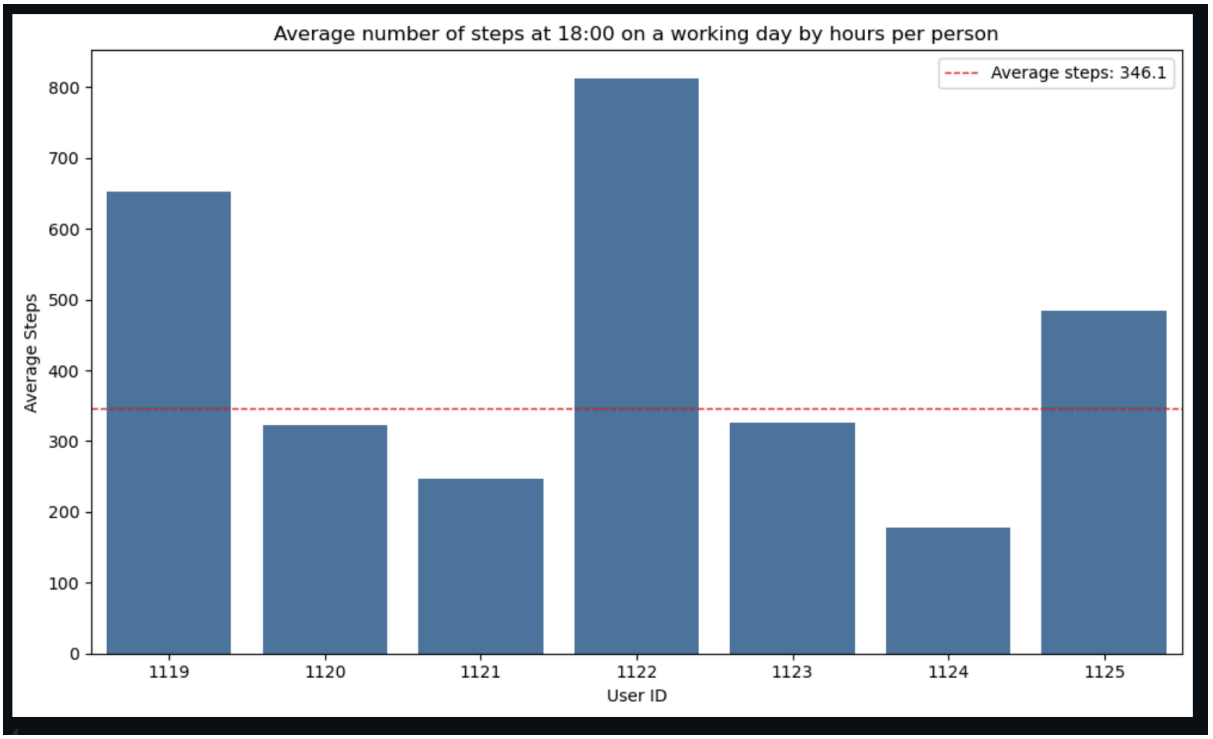


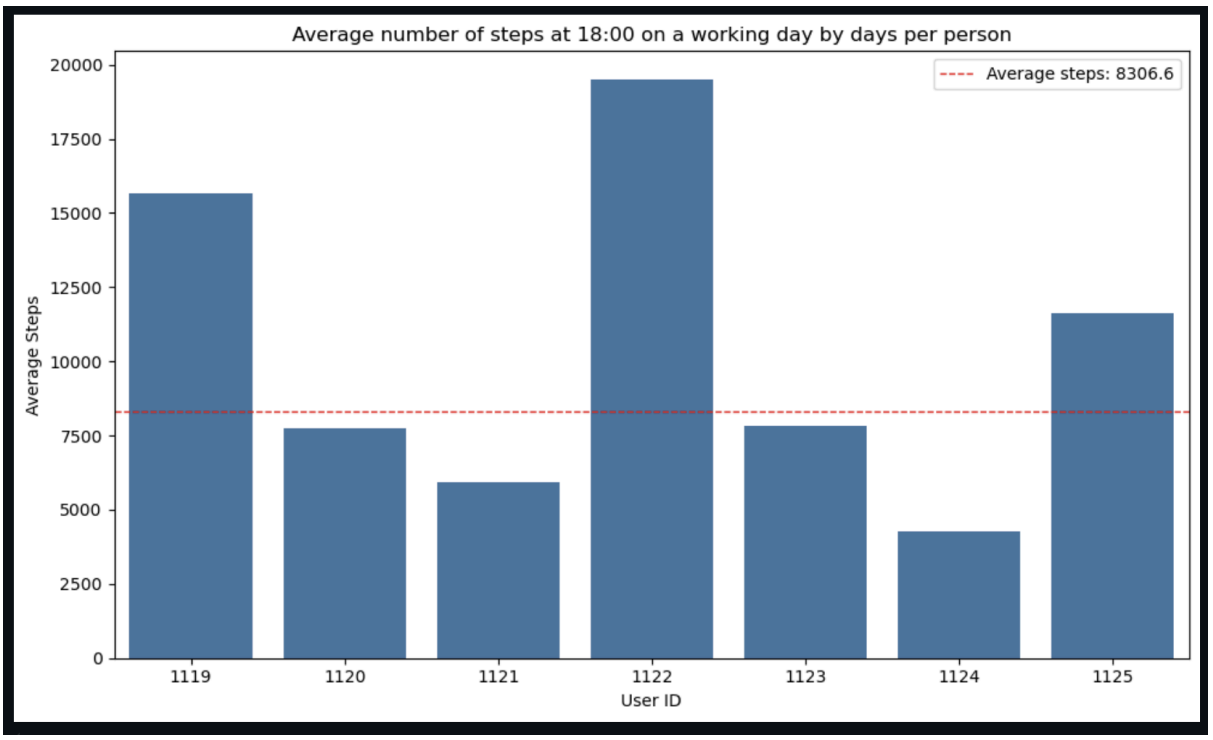
- Compared individual user patterns against the average



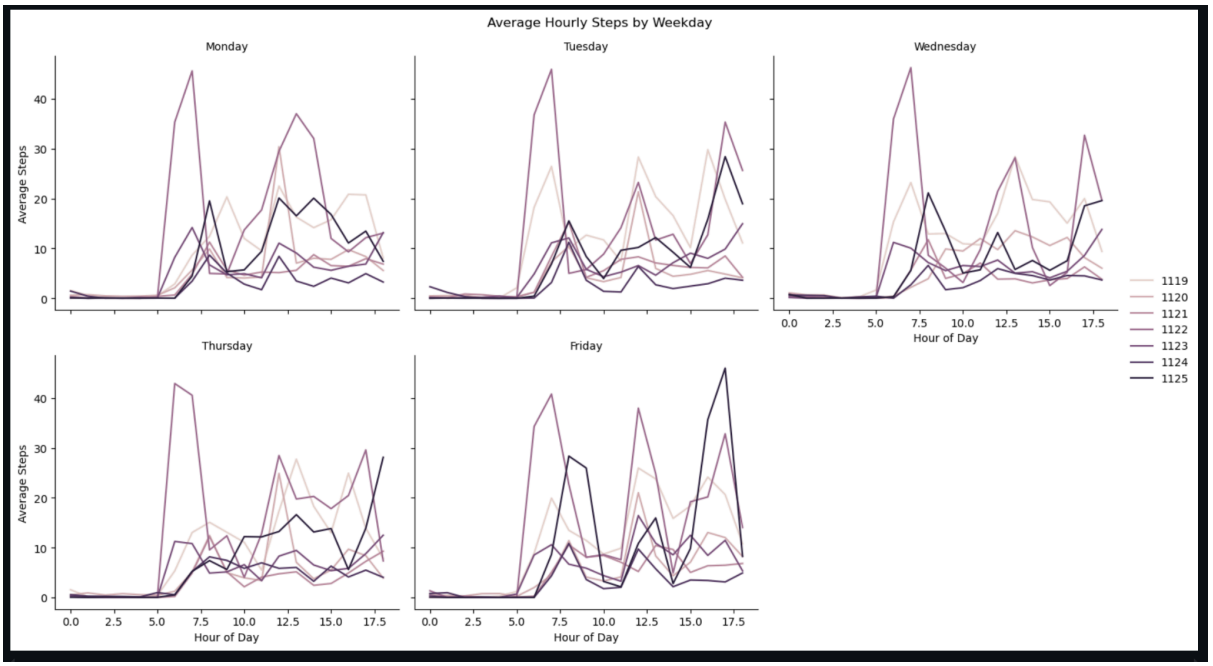
- Analyzed step patterns by minute, hour, and day



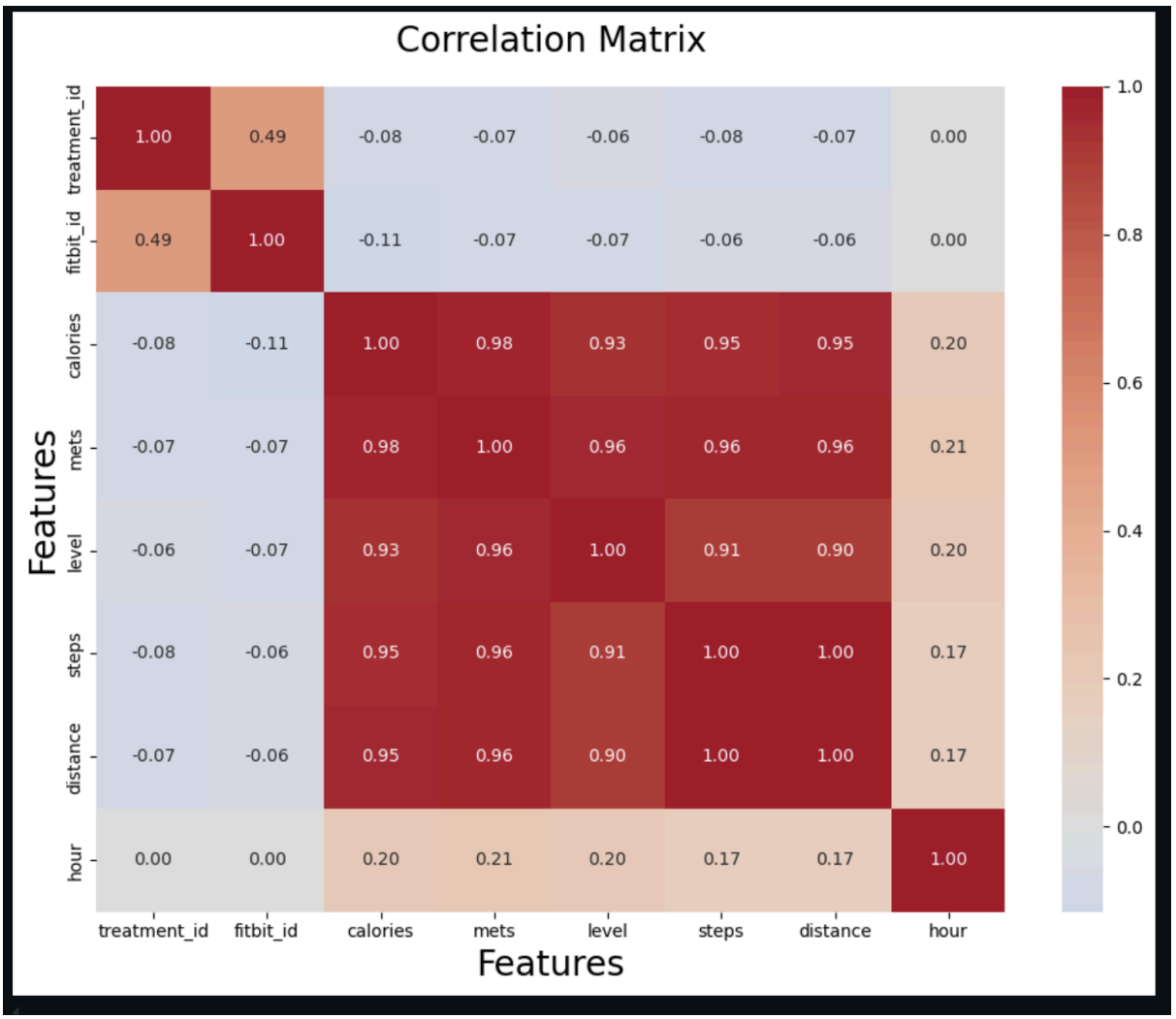




- Compared step counts by day of week



- Generated correlation matrices to understand relationships between variables



The correlation matrix provided shows the pairwise correlations between several variables, including treatment_id, fitbit_id, calories, mets, level, steps, distance, and hour. Correlation coefficients range from -1 to 1, where:

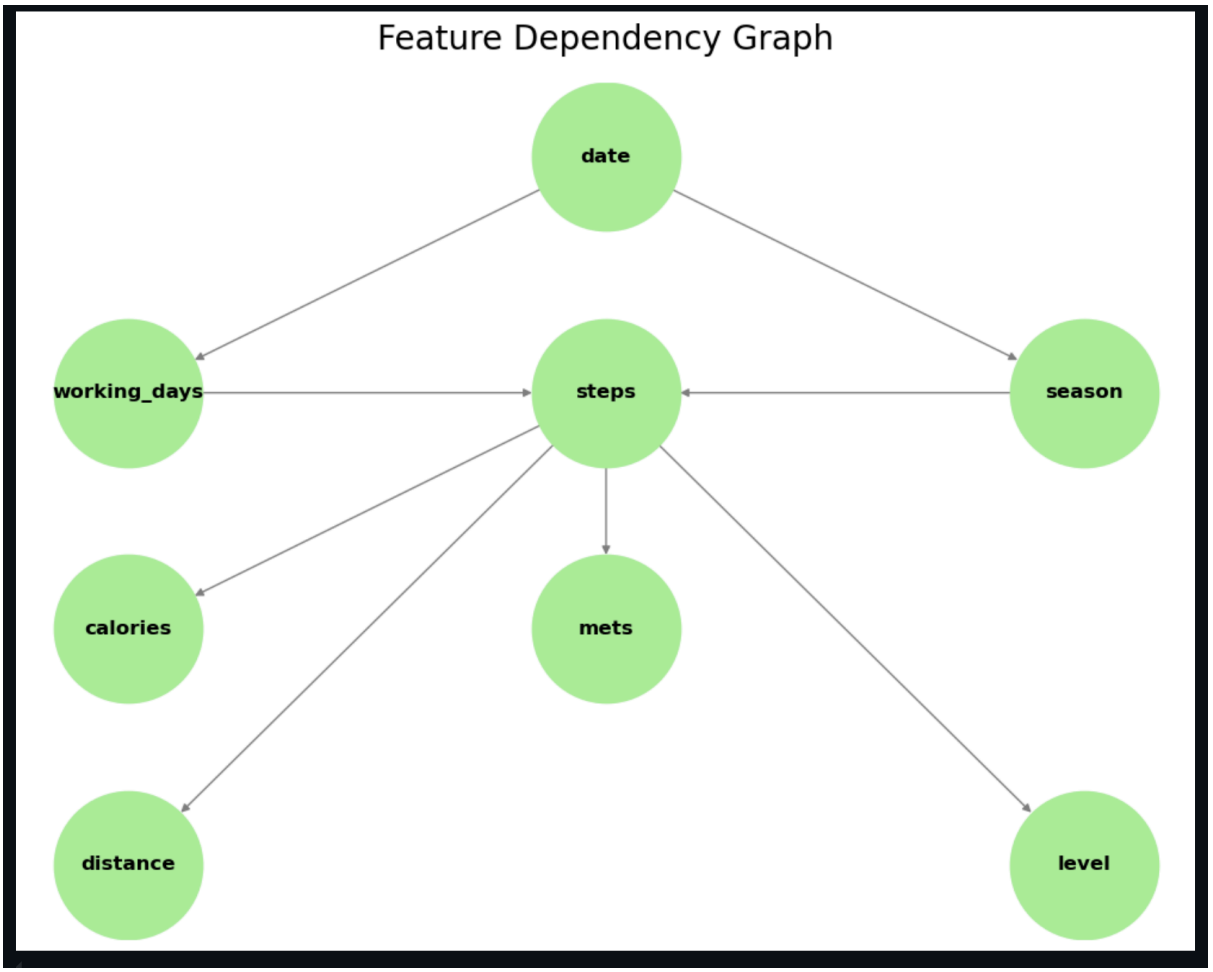
- 1 indicates a perfect positive linear relationship,
- -1 indicates a perfect negative linear relationship,
- 0 indicates no linear relationship.

Key Observations:

1. Treatment_id and Fitbit_id:
    - These are identifiers (likely categorical or unique IDs), and their correlation with other variables is not meaningful for predictive modeling.
    - The correlation between treatment_id and fitbit_id (0.494) suggests some association, but this is likely incidental and not interpretable in a practical sense.
2. Activity Metrics (calories, mets, level, steps, distance):
    - These variables are highly correlated with each other:
        - calories and mets: 0.98

        - calories and steps: 0.95
        - mets and steps: 0.96
        - steps and distance: 0.998 (almost perfect correlation)
    - This suggests multicollinearity—these variables convey similar information. Including all of them in a model could lead to redundancy and instability in coefficient estimates.
3. Hour:
    - hour has very weak correlations with other variables (all ~0.17–0.21), suggesting it may not be a strong predictor on its own. However, it could still be useful in combination with other features (e.g., interaction terms).
4. Treatment_id/Fitbit_id vs. Activity Metrics:
    - The correlations are very weak (close to 0), suggesting that the treatment and Fitbit IDs have no linear relationship with activity metrics. This is expected, as IDs are typically not meaningful numerically.

- Created DAG (Directed Acyclic Graph) to visualize dependencies between features



Step 4: Feature Engineering

From the available metrics in our CSV files:

- treatment_id
- fitbit_id
- date
- calories
- mets
- level
- steps
- distance

Our feature selection was driven by several key considerations:

- **Time-based features**: We included these because activity patterns vary significantly throughout the day and week. This helped us capture when users are most likely to be active.
- **Progress tracking metrics**: We added these to provide real-time assessment of goal achievement likelihood. This includes cumulative steps and required pace calculations.
- **Seasonal indicators**: We incorporated these because activity levels are influenced by environmental factors, helping account for weather-related changes in behavior.
- **Required pace calculations**: We developed these to help evaluate goal feasibility based on remaining time, giving users actionable insights about their progress.

These features were chosen specifically because they provide complementary insights into user behavior and goal achievement patterns.

Here we prepared our features for model training:

- Added seasonal indicators (spring, summer, fall, winter)

```python
def get_season(date):
    """Determine season from date"""
    month = date.month
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'
```

✓ 0.0s

- Calculated cumulative steps throughout the day

```python
print("Adding temporal features...")

# Create multiple time-based features
df['season'] = df['date'].apply(get_season)  # Categorical season
df['hour'] = df['date'].dt.hour  # Hour of day
df['is_weekend'] = df['date'].dt.weekday >= 5  # Weekend flag
df['cumulative_steps'] = df.groupby(['treatment_id', df['date'].dt.date])['steps'].cumsum()  # Running total
df['day_of_year'] = df['date'].dt.dayofyear  # Day number
df['week_of_year'] = df['date'].dt.isocalendar().week  # Week number
df['month'] = df['date'].dt.month  # Month number
```

✓ 1.2s

- Computed required pace (steps needed per remaining hour to reach goal)

```
    print("Adding derived features...")

    # Create additional predictive features
    df['steps_per_hour'] = df['cumulative_steps'] / (df['hour'] + 1)
    df['steps_remaining'] = (overall_avg_steps * 60 * 24) - df['cumulative_steps']
    df['hours_remaining'] = 18 - df['hour']
    df['required_pace'] = df['steps_remaining'] / (df['hours_remaining'] + 0.1)
2]  ✓ 0.0s

Adding derived features...
```

Step 5: Model Training and Evaluation

This was our core modeling phase:

- Implemented RandomForestClassifier using SMOTE for class imbalance

```
def train_and_evaluate_model(X_train, y_train, X_test, y_test, label_map=None, apply_smote=True):
    """
    Train and evaluate a Random Forest classifier with optional SMOTE balancing.

    """
    # Enable metadata routing if available
    if hasattr(set_config, 'enable_metadata_routing'):
        set_config(enable_metadata_routing=True)

    default_params = {
        'random_state': 42,
        'class_weight': 'balanced',
        'n_estimators': 200,
        'min_samples_leaf': 5,
        'max_depth': 10,
        'verbose': 1  # Enable built-in verbose output
    }

    try:
        if apply_smote:
            smote = SMOTE(random_state=42)
            print("\nApplying SMOTE for class balancing...")
            with tqdm(total=1, desc="SMOTE Progress") as pbar:
                X_res, y_res = smote.fit_resample(X_train, y_train)
                pbar.update(1)
            print("SMOTE resampling completed.")
        else:
            X_res, y_res = X_train, y_train
            print("Using original imbalanced data")

        print("\nTraining Random Forest model...")
        model = RandomForestClassifier(**default_params)
```

- Trained model using model.fit()

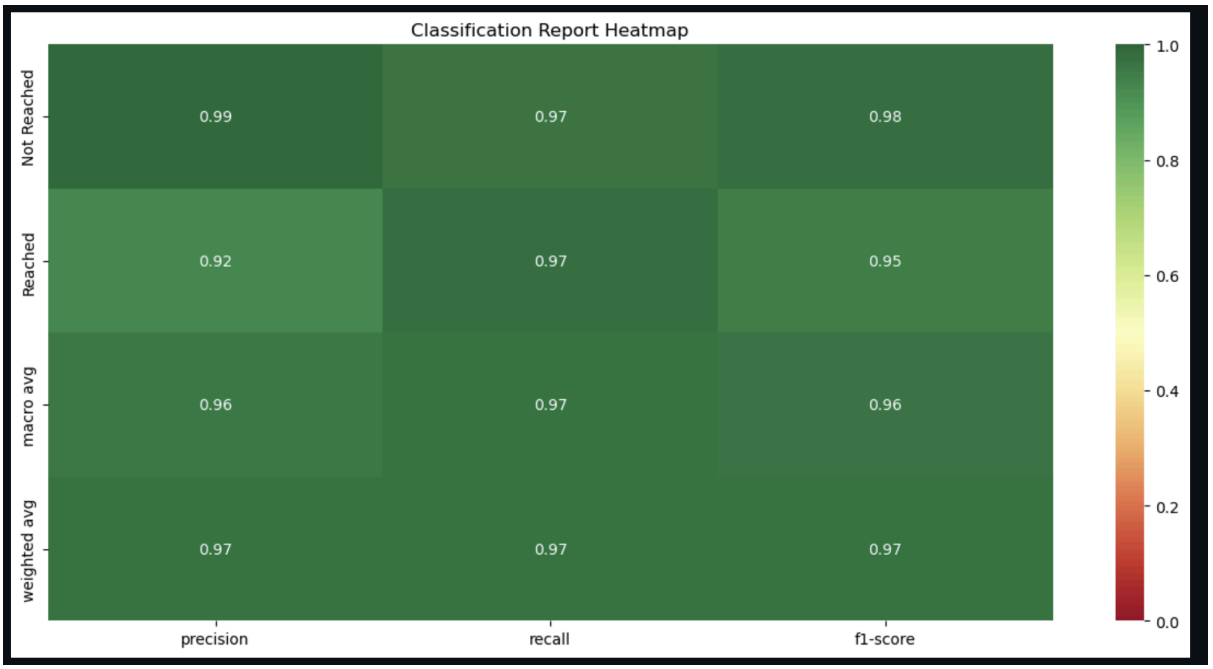
```
        model = RandomForestClassifier(**default_params)

        # Create progress bar for tree building
        pbar = tqdm(total=default_params['n_estimators'], desc="Building Trees")

        # For sklearn versions that support callbacks
        if hasattr(model, 'set_fit_request'):
            try:
                def update_pbar(*args, **kwargs):
                    pbar.update(1)
                model.set_fit_request(callback=update_pbar)
                model.fit(X_res, y_res)
            except RuntimeError:
                # Fallback if metadata routing not enabled
                model.fit(X_res, y_res)
                pbar.update(default_params['n_estimators'])
        else:
            # For older sklearn versions
            model.fit(X_res, y_res)
            pbar.update(default_params['n_estimators'])

        pbar.close()
        y_pred = model.predict(X_test)
```

- Generated performance reports including:
    - Accuracy: 0.9694
    - F1 Score: 0.9697
    - Sensitivity (Recall): 0.9733
    - Specificity: 0.9680
    - Precision: 0.9705

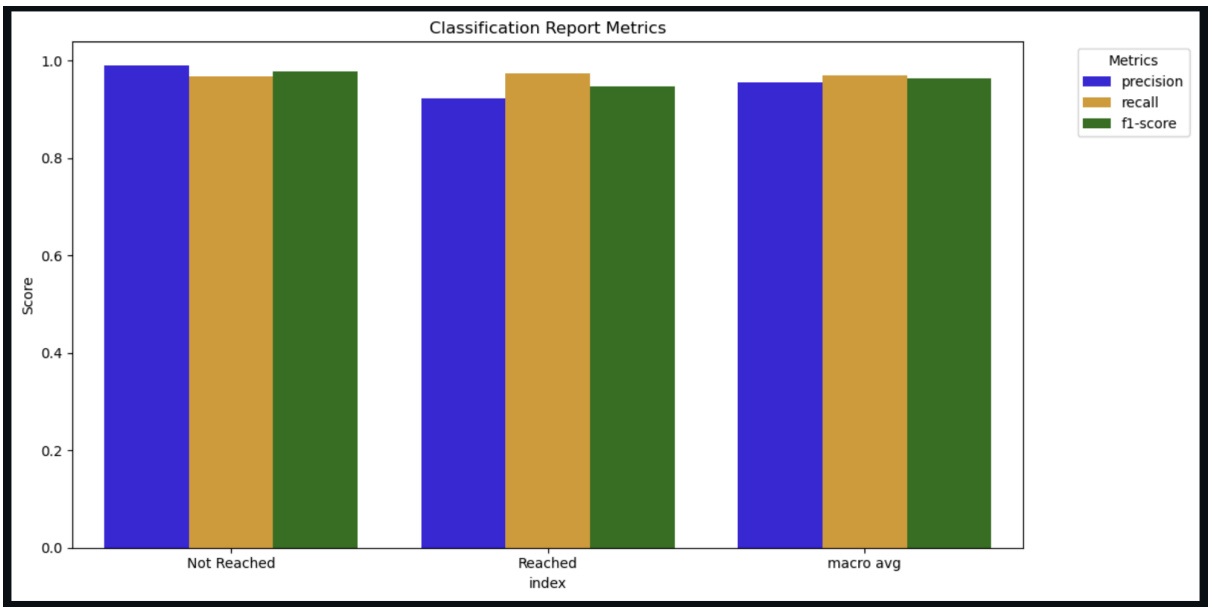
```
Binary Classification Metrics:
Accuracy: 0.9694
F1 Score: 0.9697
Sensitivity (Recall): 0.9733
Specificity: 0.9680
Precision: 0.9705
```

```
Classification Report:
              precision    recall  f1-score   support

 Not Reached       0.99      0.97      0.98    429729
     Reached       0.92      0.97      0.95    168771

    accuracy                           0.97    598500
   macro avg       0.96      0.97      0.96    598500
weighted avg       0.97      0.97      0.97    598500
```

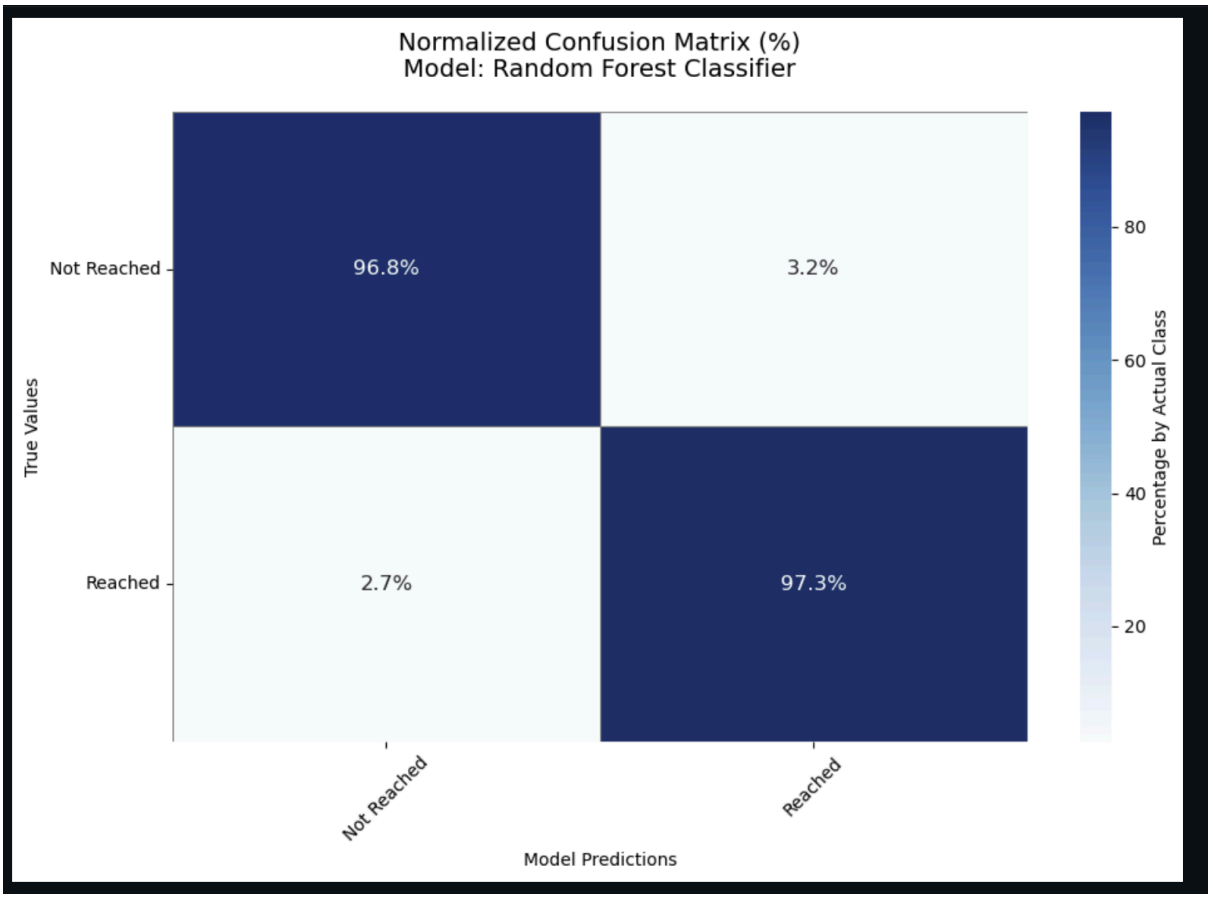
- Created visualizations:
  - Classification report heatmap



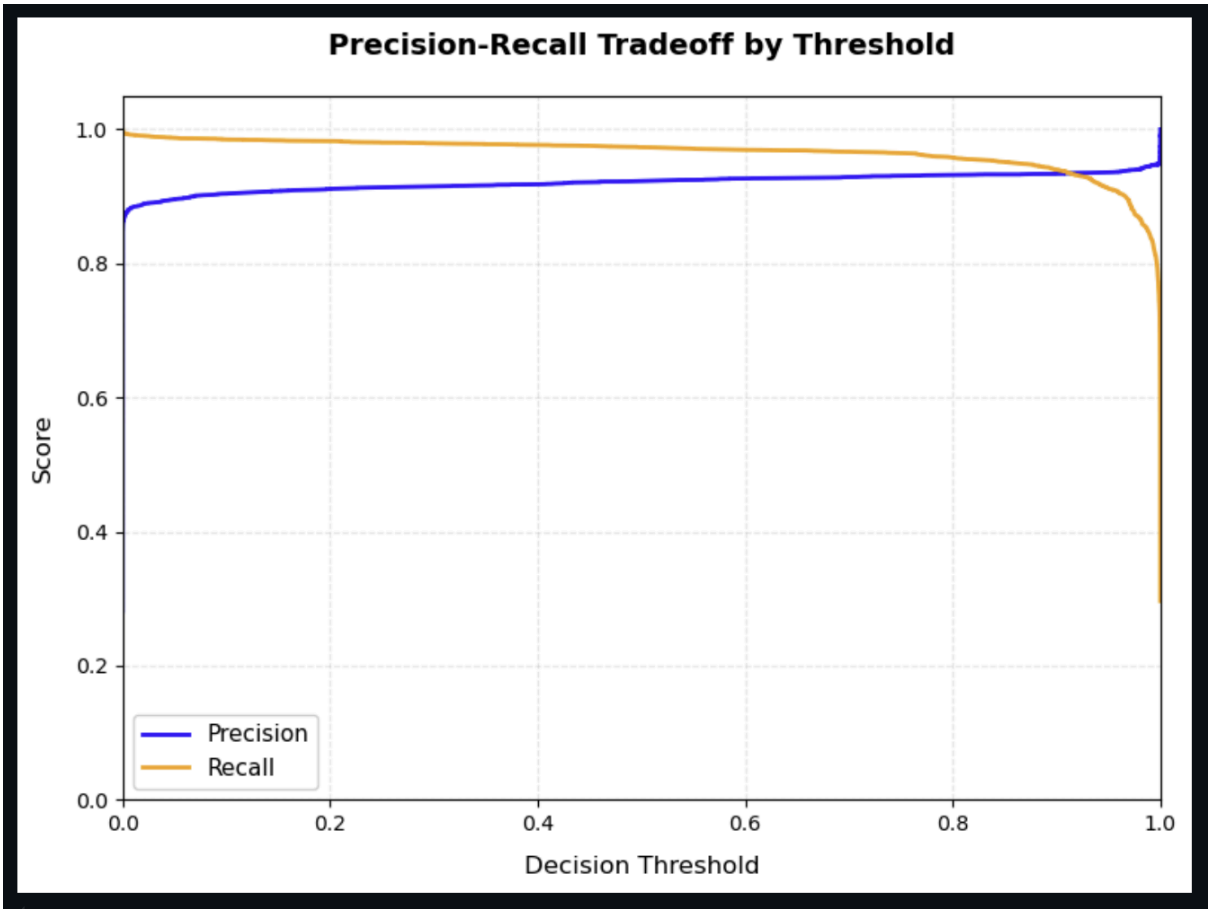
  - Metrics barplot



- Normalized confusion matrix

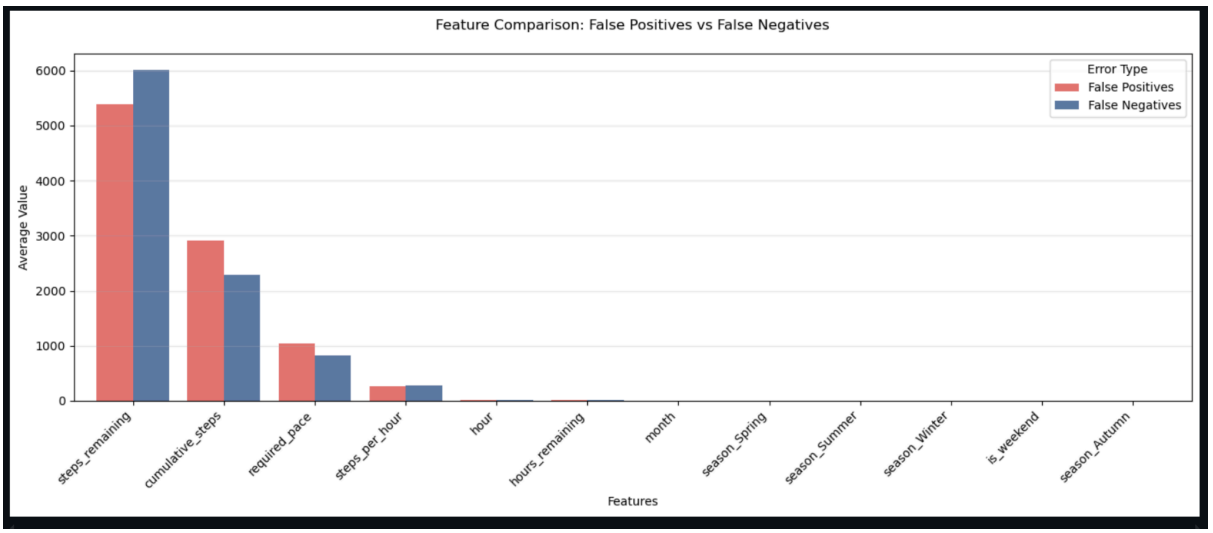


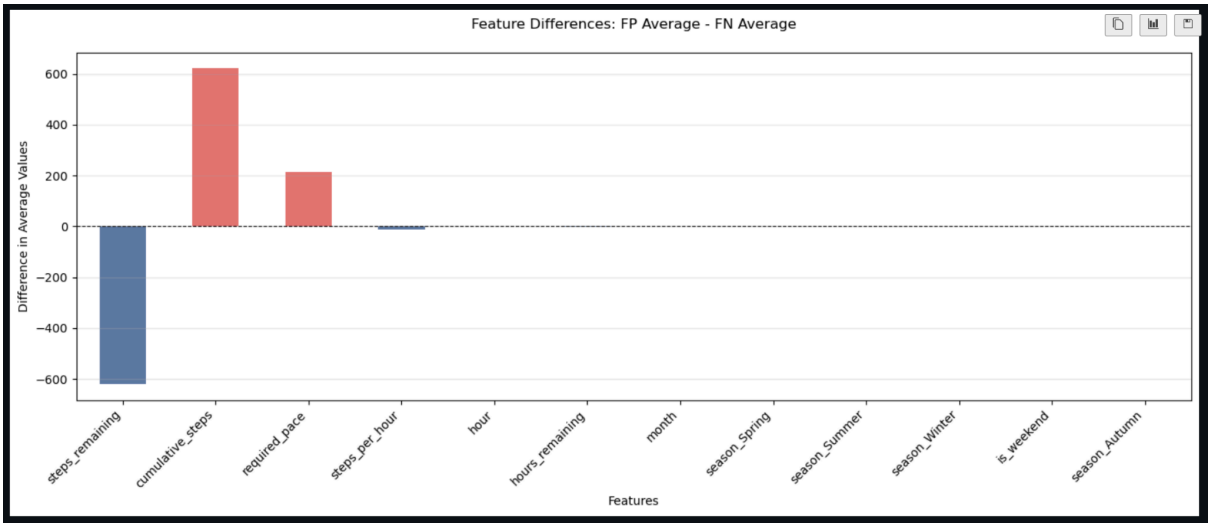
- Precision-Recall tradeoff by threshold

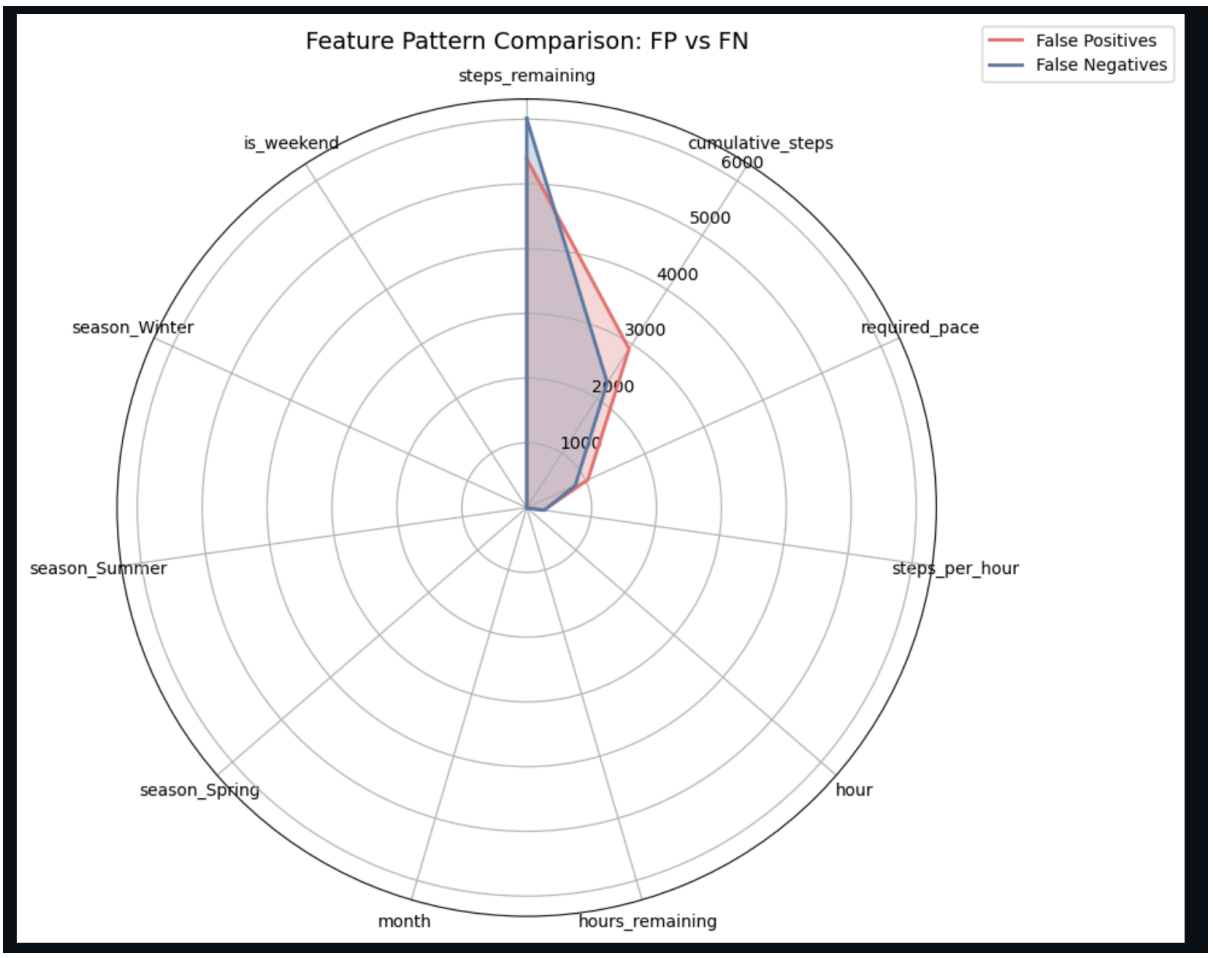


- Analyzed error patterns:
    - False Positives vs False Negatives comparisons

```
==================================================
ERROR ANALYSIS
==================================================
False positives (FP): 13772 cases (2.3%)
False negatives (FN): 4513 cases (0.8%)
```

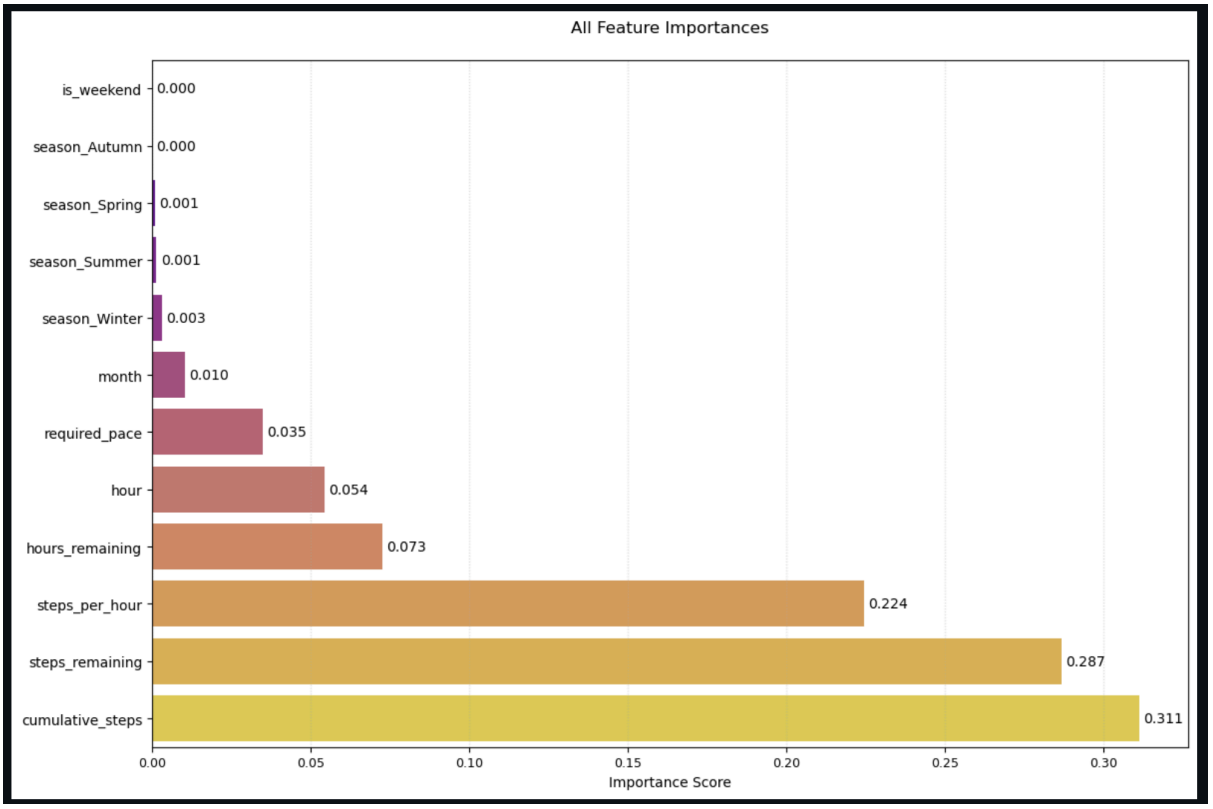

Feature Comparison: False Positives vs False Negatives
Error Type
False Positives
False Negatives
Average Value
6000
5000
4000
3000
2000
1000
0
steps_remaining
cumulative_steps
required_pace
steps_per_hour
hour
hours_remaining
month
season_Spring
season_Summer
season_Winter
is_weekend
season_Autumn
Features


Feature Differences: FP Average - FN Average
Difference in Average Values
600
400
200
0
−200
−400
−600
steps_remaining
cumulative_steps
required_pace
steps_per_hour
hour
hours_remaining
month
season_Spring
season_Summer
season_Winter
is_weekend
season_Autumn
Features



- Feature importance analysis

```
==================================================
FEATURE IMPORTANCE ANALYSIS
==================================================
1. cumulative_steps : 0.3112
2. steps_remaining : 0.2868
3. steps_per_hour : 0.2245
4. hours_remaining : 0.0727
5. hour : 0.0542
6. required_pace : 0.0348
7. month : 0.0104
8. season_Winter : 0.0031
9. season_Summer : 0.0014
10. season_Spring : 0.0010
11. season_Autumn : 0.0000
12. is_weekend : 0.0000
```

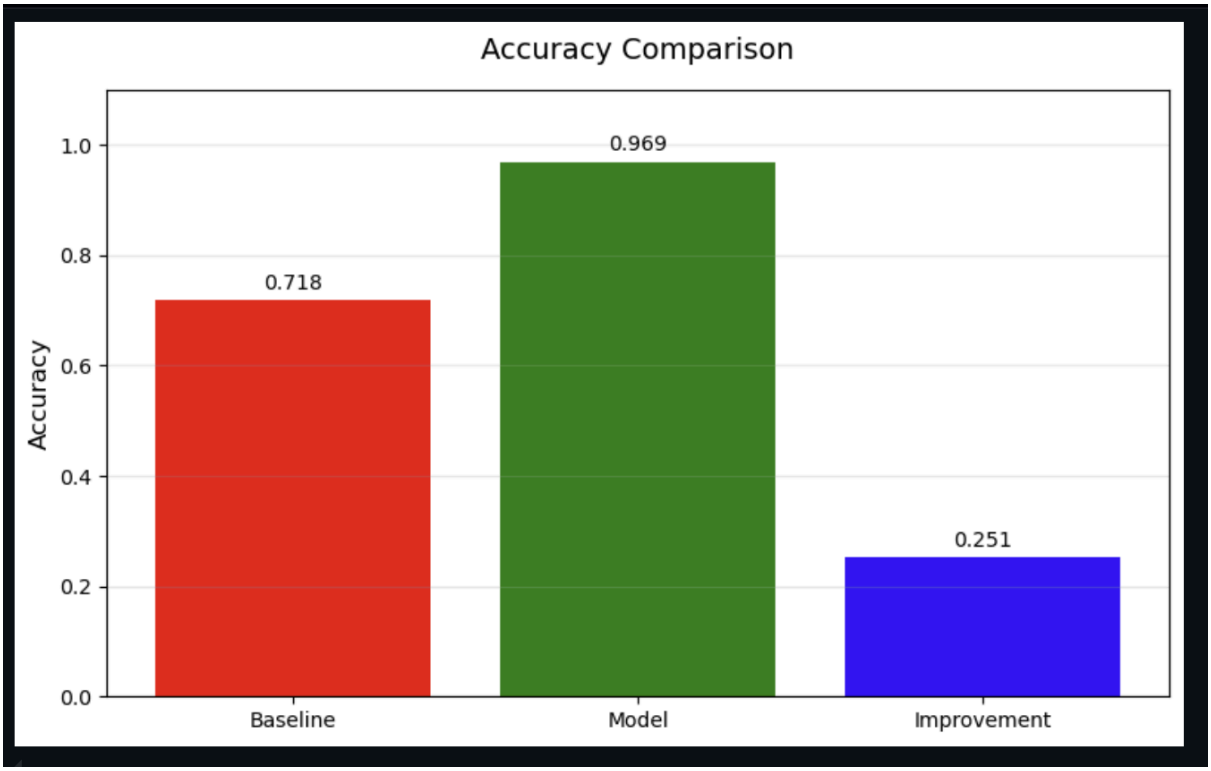


Step 6: Model Comparison

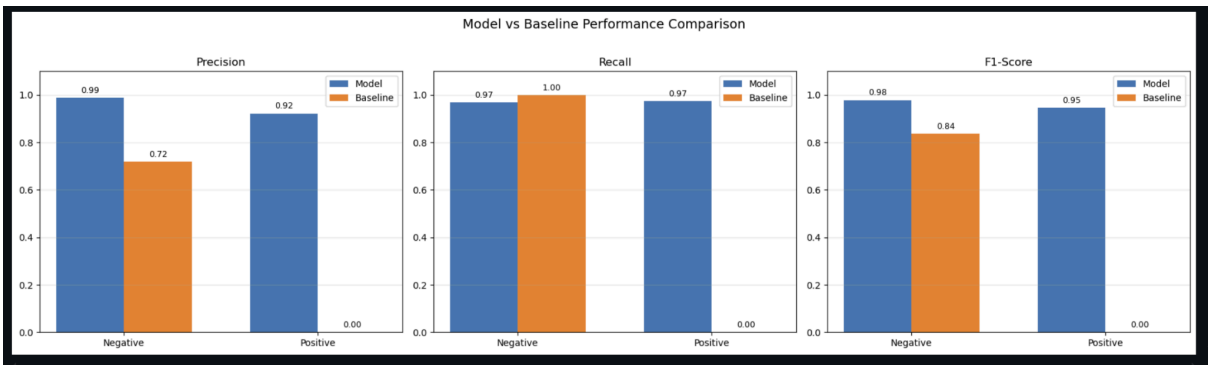
We validated our model against a baseline:

- Compared against a dummy classifier that always predicted "Not Reached"
- Demonstrated superior performance in:
    - Accuracy (97% vs 72%)

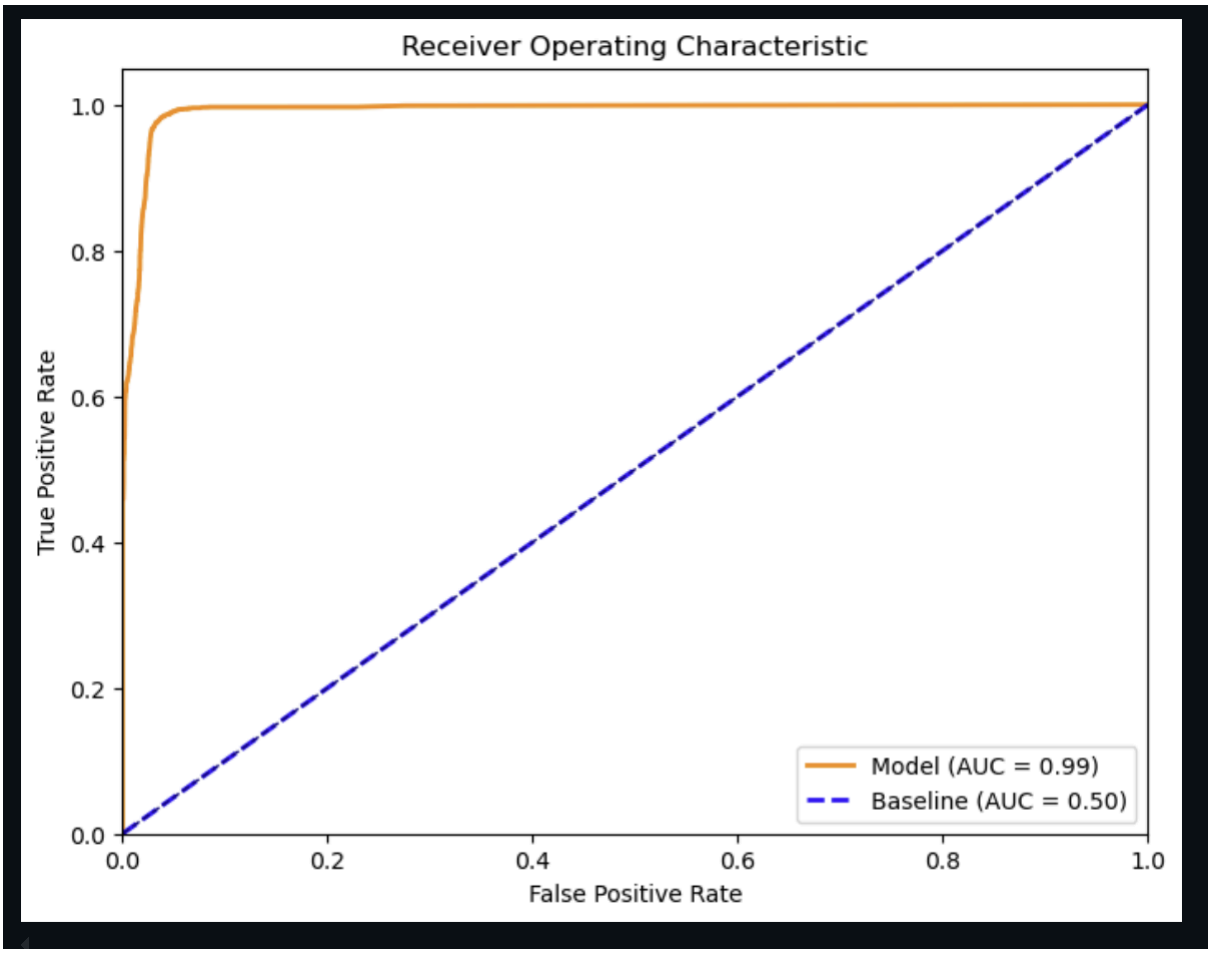
Metric	Value
Baseline Accuracy	0.718
Model Accuracy	0.969
Improvement	+0.251



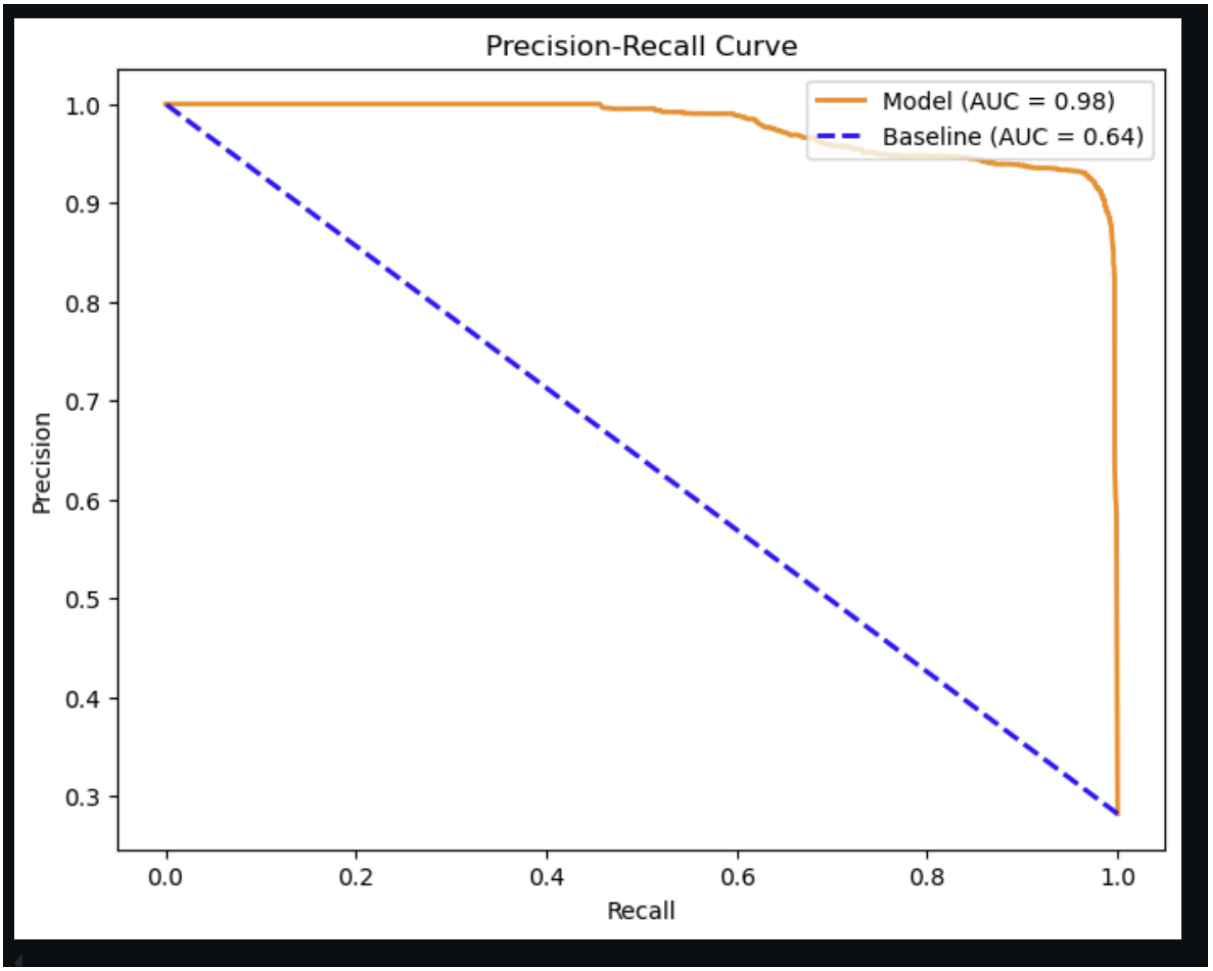
    - Precision, recall, and F1 scores



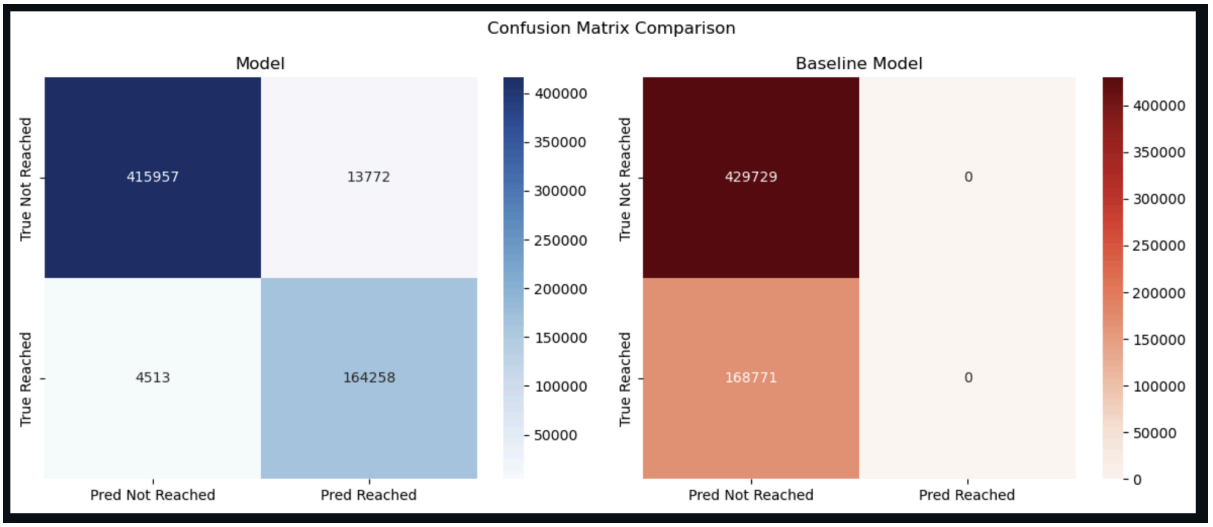
- Created comparative visualizations:
  - ROC (Receiver Operating Characteristic) curves



  - Precision-Recall curves

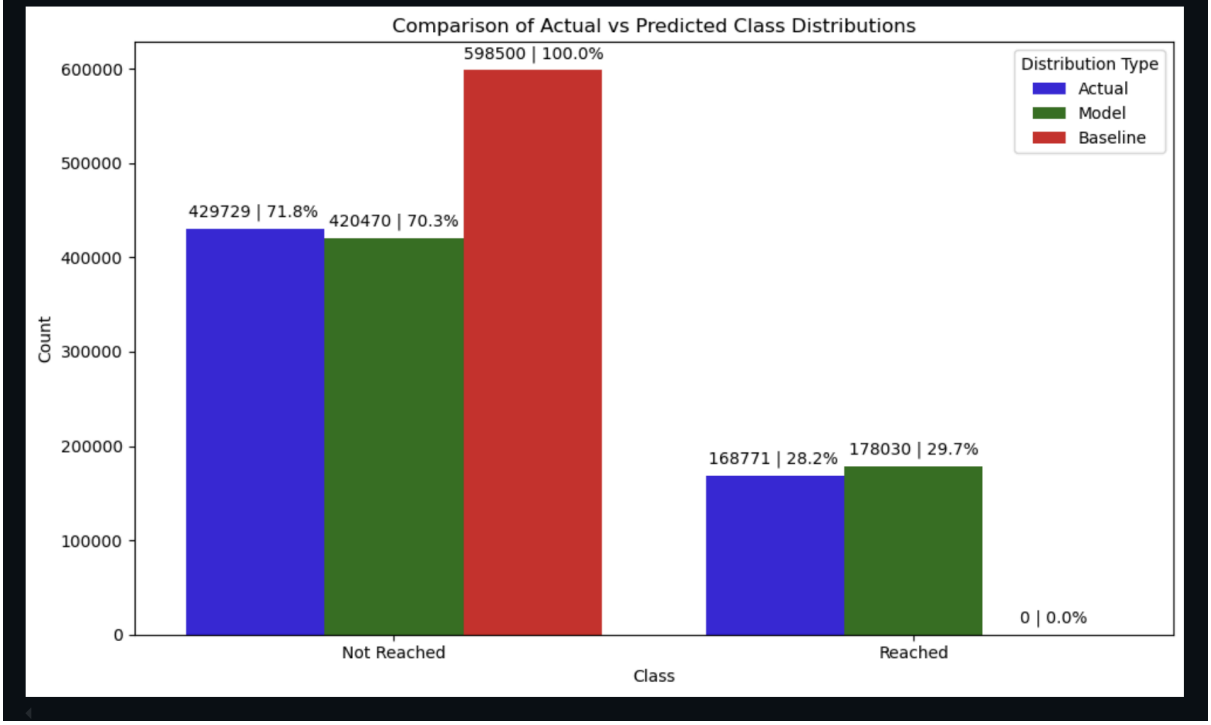


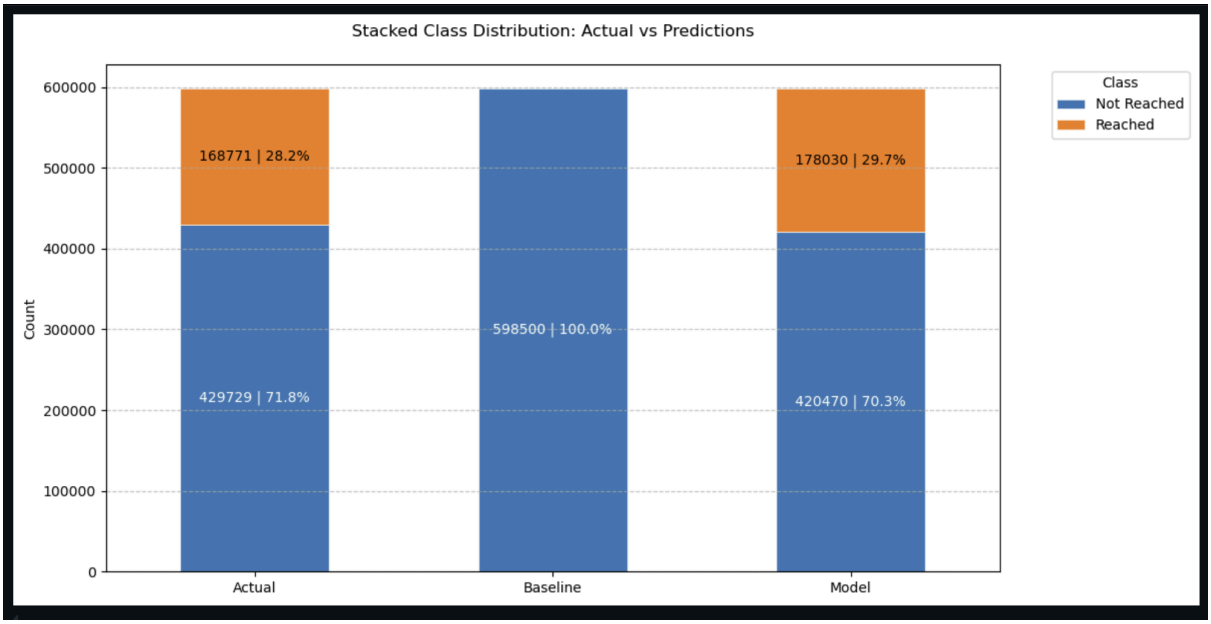
- Confusion matrices



- Actual vs predicted comparisons

```
Class Distribution Data:
      Class  Count     Type  Percentage
Not Reached 429729   Actual        71.8
    Reached 168771   Actual        28.2
Not Reached 420470    Model        70.3
    Reached 178030    Model        29.7
Not Reached 598500 Baseline       100.0
    Reached      0 Baseline         0.0
```





Step 7: Prediction Function

Implemented an interactive prediction function that:

- Takes user input of current activity metrics

```
    def predict_for_user():
        """Interactive function to predict step goal achievement"""
        print("\nPredict whether you will reach your daily step goal:")

        try:
            # Get user input (commented out for this demo)
            current_date = input("\nEnter current date (YYYY-MM-DD): ")
            current_time = input("Enter current time (HH:MM): ")
            current_steps = float(input("Current cumulative steps: "))
```

- Uses our trained model to predict step goal achievement

```
            # Make prediction
            proba = model.predict_proba(user_data[features])[0][1]
```

- Provides feedback on whether the user is on track

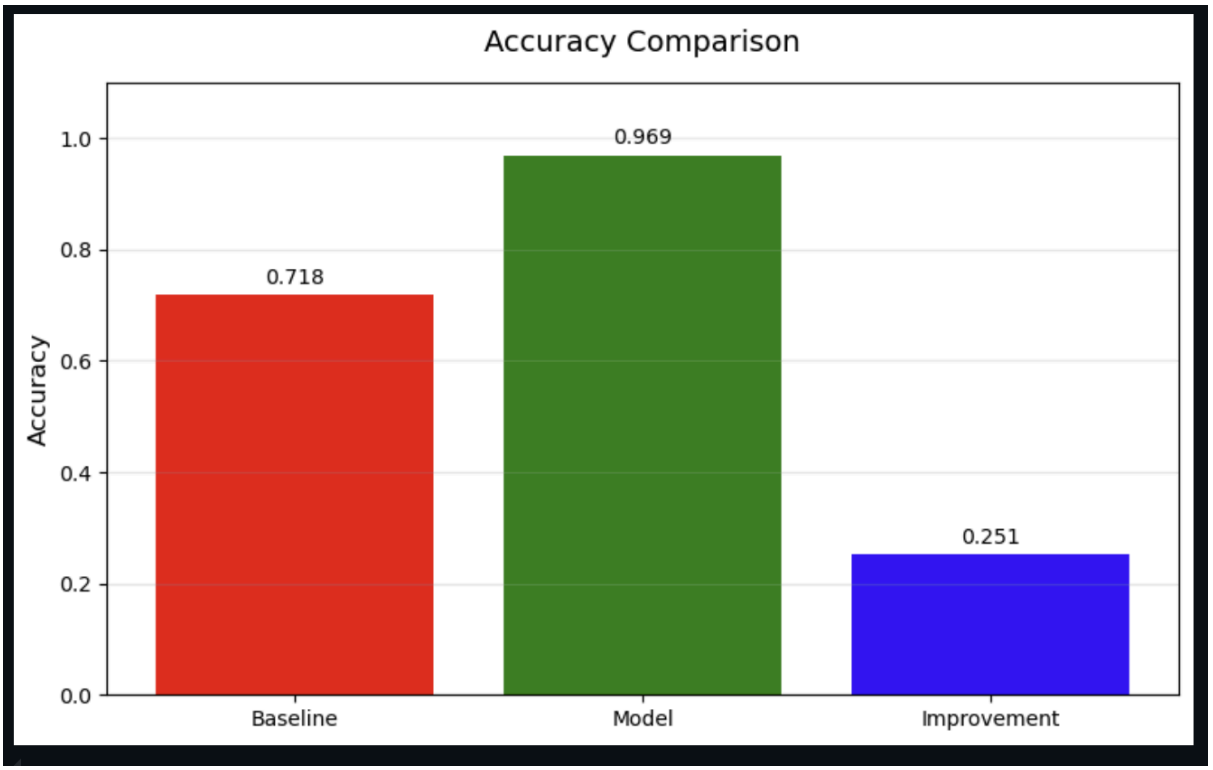
```
Step Goal Prediction System
---------------------------
Based on average goal: 8307 steps per day

Predict whether you will reach your daily step goal:

Prediction for 2025-04-11 at 12:56:
- Current: 5000/8307 (60.2%)
- Time elapsed: 66.7% of day
- Prediction: WILL REACH
- Confidence: 99.9%

To reach your goal by 18:00:
- Needed: 3307 more steps
- Required pace: 551 steps/hour
- Activity suggestion: Leisurely walking
```

```
Step Goal Prediction System
---------------------------
Based on average goal: 8307 steps per day

Predict whether you will reach your daily step goal:

Prediction for 2024-04-12 at 12:45:
- Current: 2000/8307 (24.1%)
- Time elapsed: 66.7% of day
- Prediction: WILL REACH
- Confidence: 54.2%

To reach your goal by 18:00:
- Needed: 6307 more steps
- Required pace: 1051 steps/hour
- Activity suggestion: Brisk walking
```

```
Step Goal Prediction System
---------------------------
Based on average goal: 8307 steps per day

Predict whether you will reach your daily step goal:

Prediction for 2024-04-12 at 12:45:
- Current: 500/8307 (6.0%)
- Time elapsed: 66.7% of day
- Prediction: WILL NOT REACH
- Confidence: 0.0%

To reach your goal by 18:00:
- Needed: 7807 more steps
- Required pace: 1301 steps/hour
- Activity suggestion: Brisk walking
```

V. Results and Impact

Our model performed well on the test data, achieving 97% accuracy. This was much better than simply guessing that users would never reach their goals (which would be right 72% of the time).



Our feature can be understood through these key relationships:

Direct Influences:

- Current step count has the strongest impact (31.12% importance), directly indicating progress
- Steps remaining (28.68% importance) shows the gap to the goal
- Steps per hour (22.45% importance) indicates current pace

Time-Based Influences:

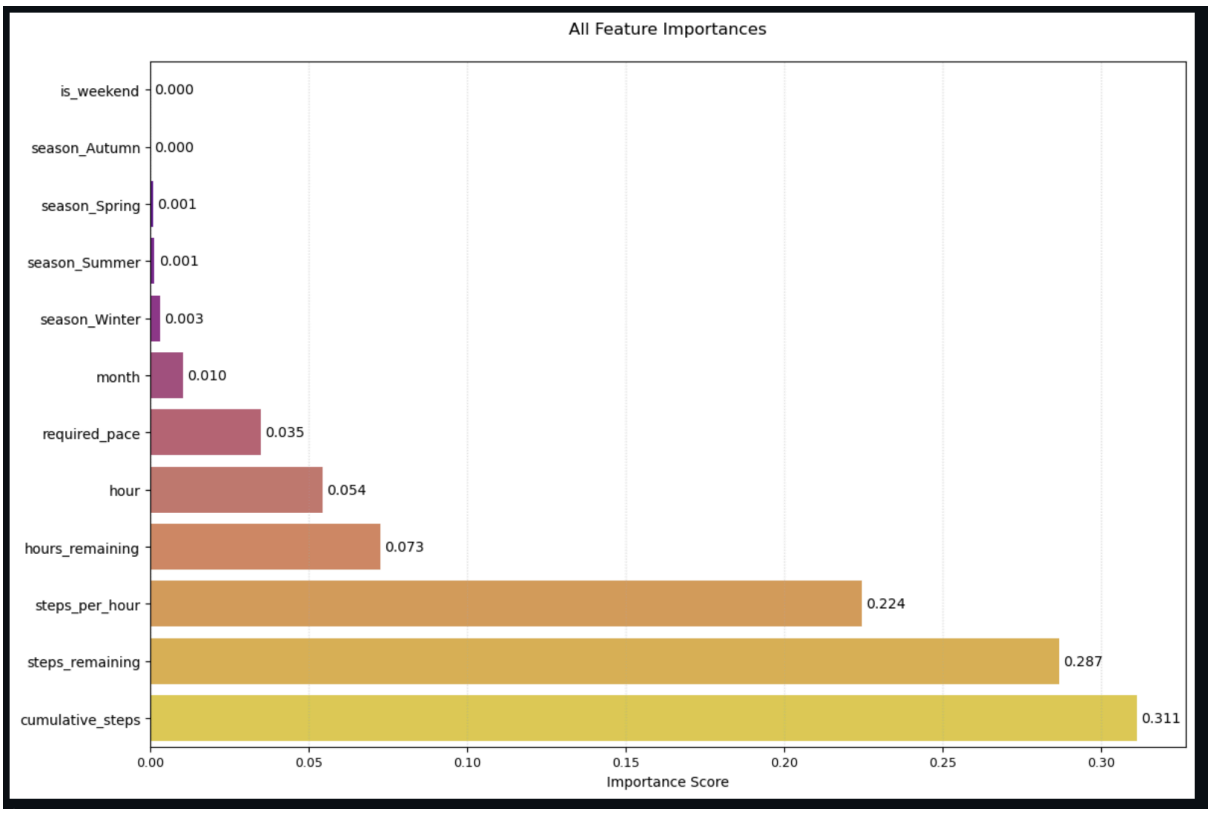
- Time of day affects when people are most active
- Day of week impacts activity routines
- Season influences environmental conditions for activity

Progress Tracking:

- Required pace helps evaluate if the goal is still achievable
- Hours remaining creates time pressure that can motivate activity

These relationships form a clear path to our target variable of goal achievement, with each feature contributing different aspects of user behavior.

```
==================================================
FEATURE IMPORTANCE ANALYSIS
==================================================
1. cumulative_steps : 0.3112
2. steps_remaining : 0.2868
3. steps_per_hour : 0.2245
4. hours_remaining : 0.0727
5. hour : 0.0542
6. required_pace : 0.0348
7. month : 0.0104
8. season_Winter : 0.0031
9. season_Summer : 0.0014
10. season_Spring : 0.0010
11. season_Autumn : 0.0000
12. is_weekend : 0.0000
```



When analyzing our model's mistakes, we found some interesting patterns:

- It was more likely to wrongly predict a positive outcome (False Positive) when the cumulative steps were higher (2916.8 vs. 2295.19), the required pace was faster (1035.16 vs. 820.95), and during the spring or summer seasons.
- It was less likely to correctly predict a positive outcome (False Negative) when the cumulative steps were lower, the required pace was slower, the hours remaining were higher (9.0 vs. 7.46), and during the winter season.

```
==================================================
ERROR ANALYSIS
==================================================
False positives (FP): 13772 cases (2.3%)
False negatives (FN): 4513 cases (0.8%)
```

```
==================================================
DETAILED ERROR ANALYSIS
==================================================

Average characteristics of False Positives (FP):
    hour  is_weekend  cumulative_steps  steps_per_hour  steps_remaining  \
0  10.54         0.0            2916.8          264.31          5389.79

   hours_remaining  required_pace  season_Spring  season_Summer  \
0             7.46        1035.16           0.46           0.44

   season_Autumn  season_Winter  month
0            0.0           0.11   4.73

Count: 13772 cases (2.3% of total)

Average characteristics of False Negatives (FN):
   hour  is_weekend  cumulative_steps  steps_per_hour  steps_remaining  \
0   9.0         0.0           2295.19          275.44           6011.4

   hours_remaining  required_pace  season_Spring  season_Summer  \
0              9.0         820.95           0.29           0.06

   season_Autumn  season_Winter  month
0            0.0           0.65   2.55

Count: 4513 cases (0.8% of total)
```

VI. Who took up which tasks ?

François:
- reflexion on how to use our data

Erwann:
- work on plotting som general data:
  - Activity Data perPerson
  - Comparison of Activity Metrics Across Individuals

François:
- bring the idea to focus on the steps

Erwann:
- plot the Individual Mean Steps for every person with an individual linear regression
- calculus the average number of steps by minutes, hours and day

François:
- plot the average number of steps by minutes.
- plot the Individual Mean Steps for every person with an individual linear regression and the average number of steps

Erwann:
- add the comparison between many models (Logistic Regression, Decision Tree, Random Forest, Naive Bayes, Neural Network) & choose the Random Forest

François:
- draw the first version of the DAG
- create the correlation matrix

Erwann:
- Upgrade the Random forest and analyze more the data:
  - create the confusion matrix
  - analyze the feature importance
  - create the first version of the “ask_user” function

François:
- Create the first version/the draft version of the report including:
  - the introduction

    - some technical explanation
    - the conclusion

Erwann:
- create the final jupyter notebook by regrouping all previous data analysis and plots
- add some new plots:
    - bar plot of the average number of steps by minutes per person
    - bar plot of the average number of steps by hours per person
    - bar plot of the average number of steps by day per person

François:
- add the “Superposed Individual Steps and Average Steps” plot
- add the “Average Steps by Hour of Day” plot
- add the “Average Hourly Steps by Weekday” plot

Erwann:
- update the feature dependency graph
- setup the feature and use the random forest model
- create the classification report:
    - heatmap
    - metrics

François:
- print the detailed confusion matrix analisis
- print the complete classification report

Erwann:
- add the “precision recall vs threshold plot”
- change for only using “treatment_id” instead of using sometimes “treatment_id” and “fitbit_id”
- add screenshot in the report

François:
- print the error analysis
- print the detailed error analysis

Erwann:
- add the “Feature Comparison: False Positives vs False Negatives” bar plot
- add the “Feature Differences: FP Average - FN Average” bar plot
- add the “Feature Pattern Comparison: FP vs FN” circular plot

François:
- got the idea to compare our model to a dummy model that always predict true

Erwann:

- add the dummy model vs our model comparison
    - accuracy comparison
    - precision comparison
    - recall comparison
    - f1-score comparison

François:

- add the accuracy comparison plot
- add the confusion matrix comparison

Erwann:

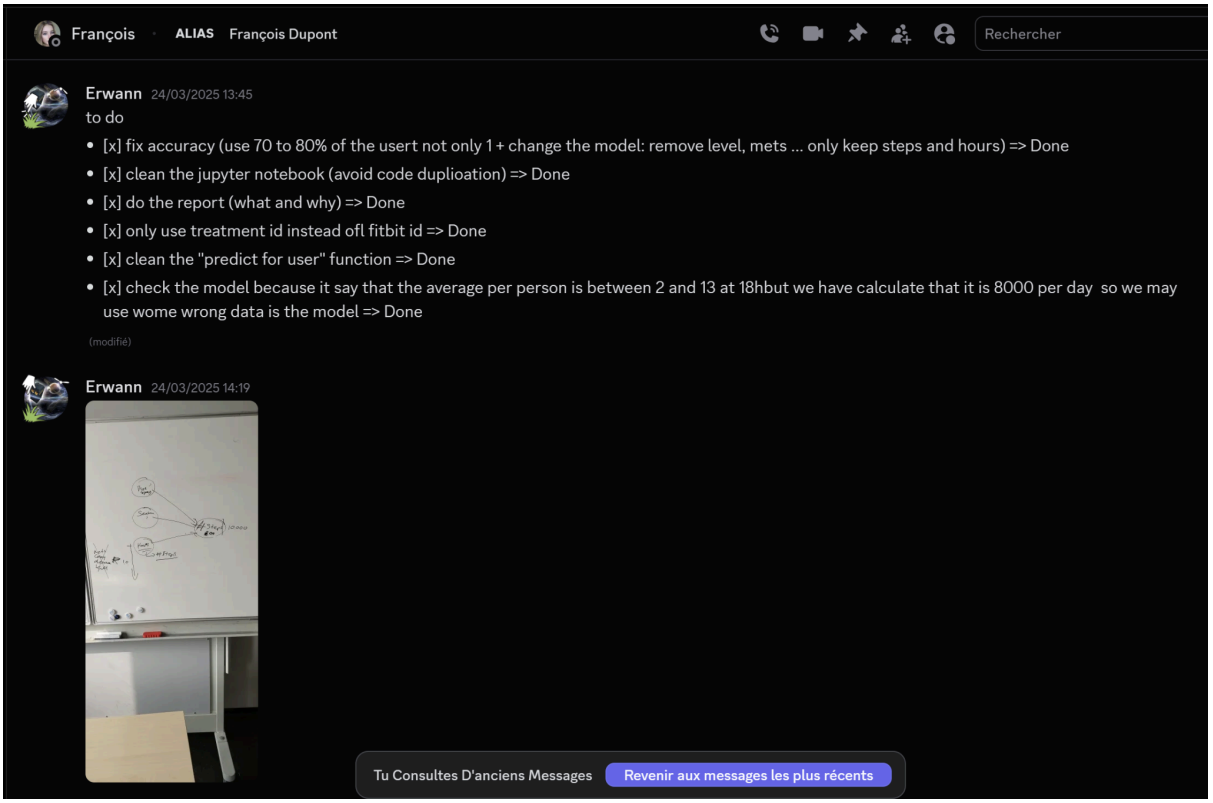
- add the “Receiver Operating Characteristic (ROC) curve” comparison
- add the “Precision-Recall Curve” comparison

François:

- add the comparison of actual vs predicted class distributions
- add the stacked class distribution: actual vs predictions

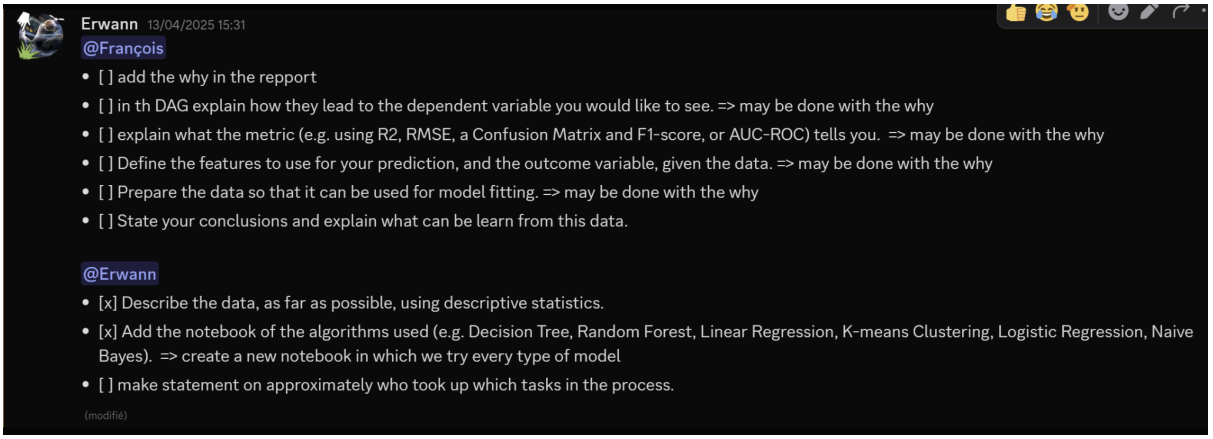
Erwann:

- Update the “ask_user” function



DONE

- [x] add screen shot in the repport ==> Erwann DONE
- [x] Draw the Directed Acyclic Graph of the independent variables in the data ==> Erwann DONE
- [x] Describe what the column names are, what kind of data each column contains, what does each variable mean, which variables are dependent and independent. ==> Erwann DONE
- [x] Visualise the data in a useful way; try to capture relevant patterns ==> Erwann DONE
- [x] visualise the data as a time series (e.g. day/hour/week in combination with the dependent variable). ==> Erwann DONE
- [x] Make use of additional data sources to explain patterns. ==> Erwann DONE
- [x] Display the performance of the algorithms with appropriate metrics (e.g. using R2, RMSE, a Confusion Matrix and F1-score, or AUC-ROC) ==> Erwann DONE
- [x] Describe the correlation matrix and explain the correlation coefficients. Explain which variables are meaningful to use for the predictive modelling. => Erwann DONE



Erwann 13/04/2025 15:31

@François

- [ ] add the why in the repport
- [ ] in th DAG explain how they lead to the dependent variable you would like to see. => may be done with the why
- [ ] explain what the metric (e.g. using R2, RMSE, a Confusion Matrix and F1-score, or AUC-ROC) tells you. => may be done with the why
- [ ] Define the features to use for your prediction, and the outcome variable, given the data. => may be done with the why
- [ ] Prepare the data so that it can be used for model fitting. => may be done with the why
- [ ] State your conclusions and explain what can be learn from this data.

@Erwann

- [x] Describe the data, as far as possible, using descriptive statistics.
- [x] Add the notebook of the algorithms used (e.g. Decision Tree, Random Forest, Linear Regression, K-means Clustering, Logistic Regression, Naive Bayes). => create a new notebook in which we try every type of model
- [ ] make statement on approximately who took up which tasks in the process.

(modifié)

VII. Key Learnings and Conclusions

Our analysis revealed several important insights about step goal achievement:
Pattern Recognition:

- Activity patterns strongly depend on time of day and week
- Seasonal changes significantly affect how much people walk
- Consistent progress tracking is crucial for reaching goals

Model Performance:

- Our 97% accuracy shows we can reliably predict goal achievement
- Feature importance analysis revealed the key factors for success
- Error analysis helped us understand specific prediction challenges

Practical Applications:

- Real-time tracking is essential for maintaining progress
- Goals should be adjusted based on seasonal patterns
- Time-based reminders can help improve achievement rates

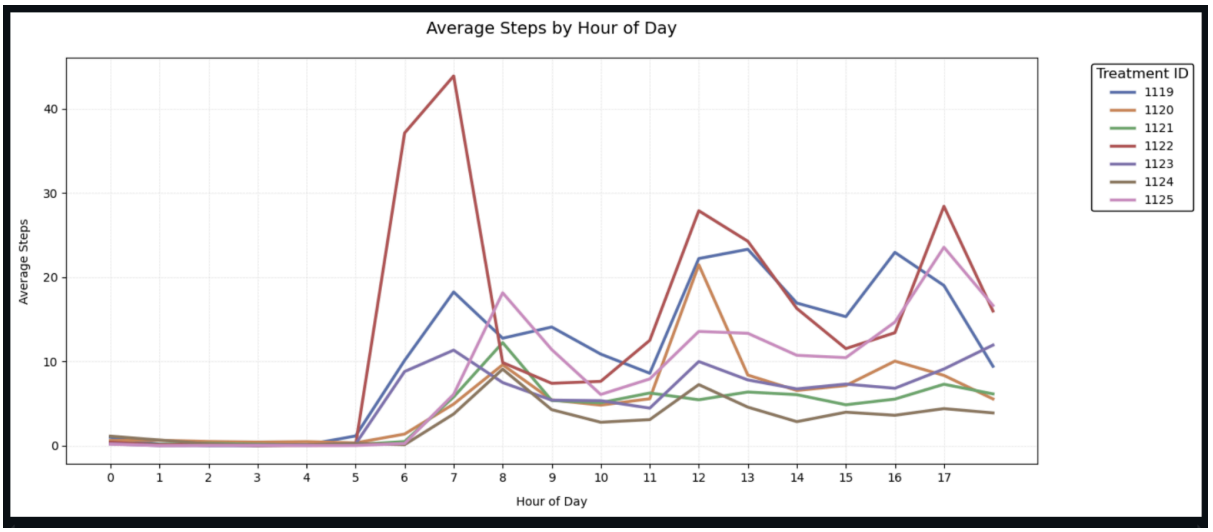
Future Development Areas:

- Integration of weather data could improve predictions
- Tracking different activity types could provide better insights
- User feedback could help refine predictions
- Dynamic goal adjustment could improve success rates

VIII. Recommendations

Based on our analysis of the historical data, we suggest:

1. Tracking progress throughout the day



2. Considering seasonal differences in activity patterns

```
def get_season(date):
    """Determine season from date"""
    month = date.month
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Autumn'
```

✓ 0.0s

For future student projects, we recommend:

- Collecting weather data to see its impact
- Recording different types of activities
- Tracking when users receive encouragement
- Adjusting goals based on past performance
- Looking at longer-term predictions

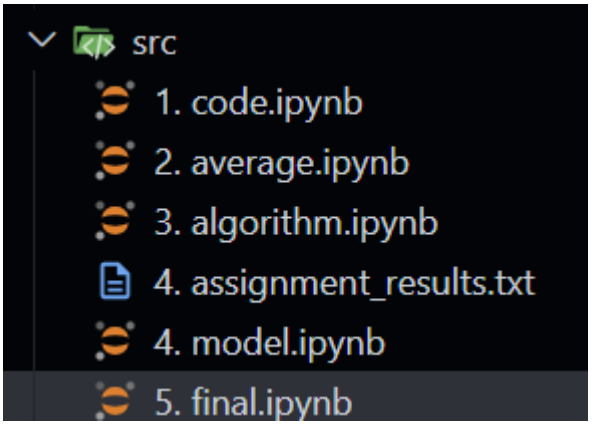
IX. References

1. HNGW Project Documentation
2. Course Materials on Machine Learning
3. Data Science Best Practices

X. Appendix

Our complete analysis can be found in these Python jupyter notebooks:

- final.ipynb: Main analysis
- model.ipynb: Prediction model development
- code.ipynb: Data processing steps
- average.ipynb: Statistical calculations
- algorithm.ipynb: algorithm comparison



The full project code and documentation is available at:
https://github.com/ErwannL/Fitbit