# Marathon

Aidan Carrier

```
"""
AUTHOR: Aidan Carrier
DATE: May 13, 2025
CLASS: ISTA 421 - Machine Learning
DESCRIPTION: This file contains all of the code for a machine learning model
developed for marathon runners. This file includes every step of the process,
from the problem statement to conclusions.
"""
```

```
'\nAUTHOR: Aidan Carrier\nDATE: May 13, 2025\nCLASS: ISTA 421 - Machine Learning\nDESCRIPTIO
```

## 1. Problem Formulation and Significance

### Problem Statement:

The Boston Marathon is one of the most prestigious marathons in the world, but marathon runners as a whole may fail to realize the significance of using machine learning methods to improve their times.

### Significance:

Using data from a previous year, we can identify patterns and predictors of better times that could be used in helping future runners achieve faster runs.

### Dataset Description:

The data was collected from 2015, and contains 26,598 rows and 25 columns. The columns include split times every 5K, demographics such as age and gender, and overall/gender finish time. All 26,598 rows represent a runner who finished the race.

## 2. Exploratory Data Analysis

### Data Exploration

```
import pandas as pd
import numpy as np
```

```
# Read in data, sometimes "-" is used instead of NA
marathon = pd.read_csv("marathon_results_2015.csv", na_values=["-"])
```

I noticed that a lot of columns didn't use NA values, but instead used "-" to show empty values.

```
print("Shape: ", marathon.shape)
#print("Data Types for each column: ",marathon.dtypes)
#print("NA counts: \n", marathon.isnull().sum())
```

```
Shape:  (26598, 25)
```

```
def time_to_sec(time):
  if pd.isna(time):
    return np.nan
  else:
    time = str(time)
    lst = time.split(":")
    return int(lst[0])*3600 + int(lst[1])*60 + int(lst[2])

time_cols = ['5K', '10K', '15K', '20K', 'Half', '25K',
             '30K', '35K', '40K', 'Official Time']

for col in time_cols:
  marathon[col] = marathon[col].apply(time_to_sec)
```

The values of the split times were in H:MM:SS format, and have to be converted to numeric values to be used in models.

```
marathon = marathon.drop(columns=['Unnamed: 0', 'Bib', 'Name', 'City', 'State', 'Country', '
```

Drop columns with either too many NA values, or not relevant to the problem.

```
marathon.describe().T.round(1)
```

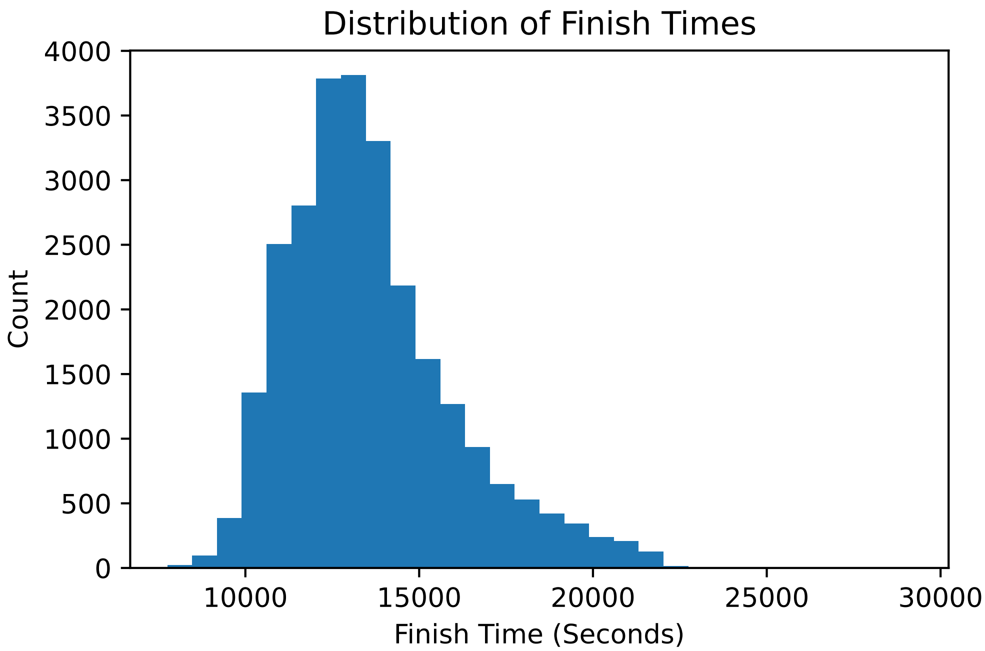
| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| Age | 26598.0 | 42.1 | 11.3 | 18.0 | 33.0 | 42.0 | 50.0 | 82.0 |
| 5K | 26446.0 | 1532.1 | 242.4 | 883.0 | 1356.0 | 1498.0 | 1658.0 | 3183.0 |
| 10K | 26567.0 | 3045.6 | 478.8 | 1783.0 | 2702.0 | 2979.0 | 3299.0 | 6436.0 |
| 15K | 26580.0 | 4575.8 | 728.7 | 2697.0 | 4057.0 | 4469.0 | 4958.0 | 9705.0 |
| 20K | 26569.0 | 6139.5 | 1001.0 | 3628.0 | 5434.0 | 5985.0 | 6649.0 | 13027.0 |
| Half | 26570.0 | 6478.3 | 1057.5 | 3841.0 | 5733.0 | 6315.0 | 7017.0 | 13701.0 |
| 25K | 26567.0 | 7729.4 | 1294.7 | 4565.0 | 6823.0 | 7521.0 | 8374.0 | 16566.0 |
| 30K | 26559.0 | 9396.8 | 1622.3 | 5519.0 | 8262.0 | 9125.0 | 10190.0 | 20624.0 |
| 35K | 26547.0 | 11119.7 | 1961.9 | 6479.0 | 9754.0 | 10791.0 | 12074.5 | 24691.0 |
| 40K | 26542.0 | 12834.6 | 2290.9 | 7359.0 | 11255.0 | 12448.0 | 13956.8 | 27688.0 |
| Official Time | 26598.0 | 13585.6 | 2428.1 | 7757.0 | 11918.0 | 13180.0 | 14785.0 | 29161.0 |

This shows the summary statistics of the numeric data. This can be used to see the means, standard deviations, and distributions of data such as min/max and percentiles.

### Data Visualization

```
import matplotlib.pyplot as plt

plt.figure()
plt.hist(marathon["Official Time"], bins=30)
plt.xlabel("Finish Time (Seconds)")
plt.ylabel("Count")
plt.title("Distribution of Finish Times")
plt.show()
```




Here is a plot of the distribution of finish times. It is interesting to see how this is right skewed.

### Handling of Missing/Imbalanced Data

```
print("NA counts: \n", marathon.isnull().sum())
```

```
NA counts:
 Age              0
M/F              0
5K             152
10K             31
15K             18
20K             29
Half            28
25K             31
30K             39
35K             51
40K             56
Official Time    0
dtype: int64
```

```
marathon = marathon.dropna(subset=time_cols)
```

With over 25000 observations, we can simply remove the split times with incomplete values, without it causing too much harm to the model. It is important to note that I am dropping the whole row when removing the NA values, even if a runner only had one incomplete data point.

```
#Check
#print("NA counts: \n", marathon.isnull().sum())
```

## 3. Model Selection, Application, and Evaluation

### Justify Model Choice

Decision Trees: By classifying runners into brackets (Under 3 hours and over 5 hours) based on splits and demographic data, we can create a blueprint that future runners can use to achieve a goal. While a majority of the runners will fall into the between 3-5 hours category, they can use this model (or change the brackets) to see what they can do to improve.

### Method Applications

```
def time_bracket(sec):
   if sec < (3*3600):
     return 'Under 3'
   elif sec > (5*3600):
      return 'Over 5'
   else:
      return 'Between 3-5'

marathon['Bracket'] = marathon['Official Time'].apply(time_bracket)
```

```
# If/else statement assigning 1 to Male, 0 to Female
marathon["M/F"] =np.where(marathon["M/F"] == 'M', 1, 0)
```

```
# Select features
features = ['Age', 'M/F', '5K', '15K', 'Half', '25K', '35K']
x= marathon[features]
y= marathon['Bracket']
```

```python
#Imports related to tree methods
import sklearn.model_selection as skm
from sklearn.tree import (DecisionTreeClassifier as DTC,
                          plot_tree, export_text)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier


# Get training and test data, using 20% test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1000)
```

### Decision Tree

```python
#Fit decision tree
tree = DTC(max_depth=4, random_state=1000)
tree.fit(x_train, y_train)
```

```
DecisionTreeClassifier(max_depth=4, random_state=1000)
```

### Random Forest

```python
#Fit random forest
rf = RandomForestClassifier(n_estimators=100,max_depth=10, random_state=1000)
rf.fit(x_train, y_train)
```

```
RandomForestClassifier(max_depth=10, random_state=1000)
```

## Model Evaluation

```python
#Validate the decision tree using cross validation
validation = skm.ShuffleSplit(n_splits=1, test_size=200, random_state=0)

results = skm.cross_validate(tree, x, y, cv=validation)

print("Test Score:",results['test_score'])
```

```
Test Score: [0.975]
```

```
#Identical code as before, but using the random forest model
validation = skm.ShuffleSplit(n_splits=1, test_size=200, random_state=0)

results = skm.cross_validate(rf, x, y, cv=validation)

print("Test Score:",results['test_score'])
```

```
Test Score: [0.985]
```

The decision tree is more intuitive, but slightly less accurate than the random forest.

## 4. Results, Conclusions, and Real-World Implications

### Results Presentation

```
#Print the tree, showing decision boundaries
print(export_text(tree, feature_names=features, show_weights=True))
```

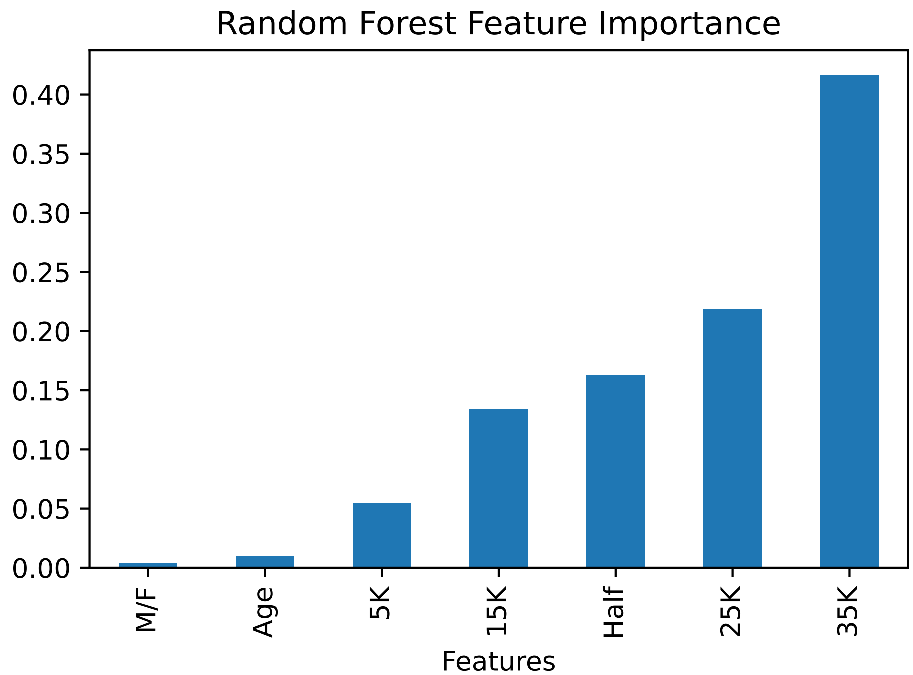
```
|--- 35K <= 8949.50
|   |--- 35K <= 8798.00
|   |   |--- 35K <= 8654.50
|   |   |   |--- 35K <= 8462.50
|   |   |   |   |--- weights: [2.00, 0.00, 854.00] class: Under 3
|   |   |   |--- 35K >  8462.50
|   |   |   |   |--- weights: [10.00, 0.00, 417.00] class: Under 3
|   |   |--- 35K >  8654.50
|   |   |   |--- 25K <= 5998.00
|   |   |   |   |--- weights: [15.00, 0.00, 2.00] class: Between 3-5
|   |   |   |--- 25K >  5998.00
|   |   |   |   |--- weights: [26.00, 0.00, 355.00] class: Under 3
|   |--- 35K >  8798.00
|   |   |--- Half <= 5220.50
|   |   |   |--- 35K <= 8840.50
|   |   |   |   |--- weights: [26.00, 0.00, 28.00] class: Under 3
|   |   |   |--- 35K >  8840.50
|   |   |   |   |--- weights: [63.00, 0.00, 9.00] class: Between 3-5
```

```
|   |   |--- Half >  5220.50
|   |   |   |--- 35K <= 8887.50
|   |   |   |   |--- weights: [32.00, 0.00, 238.00] class: Under 3
|   |   |   |--- 35K >  8887.50
|   |   |   |   |--- weights: [93.00, 0.00, 144.00] class: Under 3
|--- 35K >  8949.50
|   |--- 35K <= 14786.00
|   |   |--- 35K <= 14567.50
|   |   |   |--- 35K <= 9010.50
|   |   |   |   |--- weights: [186.00, 0.00, 43.00] class: Between 3-5
|   |   |   |--- 35K >  9010.50
|   |   |   |   |--- weights: [17037.00, 38.00, 11.00] class: Between 3-5
|   |   |--- 35K >  14567.50
|   |   |   |--- 25K <= 9965.50
|   |   |   |   |--- weights: [15.00, 38.00, 0.00] class: Over 5
|   |   |   |--- 25K >  9965.50
|   |   |   |   |--- weights: [93.00, 24.00, 0.00] class: Between 3-5
|   |--- 35K >  14786.00
|   |   |--- 35K <= 15076.50
|   |   |   |--- 25K <= 10330.50
|   |   |   |   |--- weights: [13.00, 108.00, 0.00] class: Over 5
|   |   |   |--- 25K >  10330.50
|   |   |   |   |--- weights: [35.00, 27.00, 0.00] class: Between 3-5
|   |   |--- 35K >  15076.50
|   |   |   |--- 35K <= 15274.00
|   |   |   |   |--- weights: [4.00, 118.00, 0.00] class: Over 5
|   |   |   |--- 35K >  15274.00
|   |   |   |   |--- weights: [1.00, 933.00, 0.00] class: Over 5
```

Above is the decision tree, classifying runners by their splits. Because of the high correlation to time, the later the split means it is a more significant predictor of finish time. Because of this, I also decided to fit a random forest do be able to see if the other factors could also be significant.

```
# Get important features and sort
important = pd.Series(rf.feature_importances_, index=features).sort_values()

plt.figure()
important.plot(kind='bar')
plt.title("Random Forest Feature Importance")
plt.xlabel("Features")
plt.show()
```




This plot above shows how important each feature is to the random forest. Surprisingly, age and sex do not appear significant at all. When looking at the accuracy scores in the previous section, it was seen that both types of trees have high accuracy scores, with random forest having a slightly higher accuracy.

## Conclusions

Both the decision tree and random forest had high accuracy scores, indicating that both models could predict which bracket a runner would fall into given splits and demographic data. The decision tree is able to be used as a guideline for future runners to see which split times they need to get to achieve a goal. The random forest could also be used if a runner wants a more accurate model, with less interpretability. When looking at the features used in models, it was interesting to see that age and sex were insignificant predictors, and the later splits were very significant.

## Workforce/Graduate School Preparation

Looking into my future, I am happy I did this project. This project strengthened my skills in data collection, cleaning, analysis, and machine learning models that will be important to my future. Focusing on one question for this project allowed for deeper thought into what models could be used, and what is necessary to answer that question. I have also used qmd files to strengthen my communication skills by having text outside of code chunks, and used sections for clear documentation and organization.