USOPC Performance Data Analyst Coding Exercise

Abigail Snyder

In [397…
```
import pandas as pd
import numpy as np
import seaborn as sns
import math
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree
from sklearn.metrics import mean_squared_error, r2_score
```

Data Loading & EDA

In [398…
```
wellness = pd.read_excel("Wellness Load and Results Data.xlsx", sheet_name=0)
```

In [399…
```
wellness.head()
```

Out[399]:

	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress	Tra Ho
0	2023-05-05	Athlete 1	m	47	30	31	0	8.50	61	39	N
1	2023-05-05	Athlete 2	m	27	54	47	68	7.00	33	59	N
2	2023-05-05	Athlete 3	m	35	36	90	65	7.25	94	32	N
3	2023-05-05	Athlete 4	m	55	1	84	58	7.00	64	0	N
4	2023-05-05	Athlete 5	m	50	67	74	80	6.25	73	40	N

In [400…
```
wellness.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 688 entries, 0 to 687
Data columns (total 12 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Date                            688 non-null    datetime64[ns]
 1   Athlete                         688 non-null    object
 2   Gender                          688 non-null    object
 3   Fatigue                         688 non-null    int64
 4   Soreness                        688 non-null    int64
 5   Motivation                      688 non-null    int64
 6   Resting HR                      688 non-null    int64
 7   Sleep Hours                     688 non-null    float64
 8   Sleep Quality                   688 non-null    int64
 9   Stress                          688 non-null    int64
 10  Travel Hours                    107 non-null    float64
 11  Sport Specific Training Volume  346 non-null    object
dtypes: datetime64[ns](1), float64(2), int64(6), object(3)
memory usage: 64.6+ KB
```

In [401…
```
wellness.dtypes
```

Out[401]:
```
Date                              datetime64[ns]
Athlete                                   object
Gender                                    object
Fatigue                                    int64
Soreness                                   int64
Motivation                                 int64
Resting HR                                 int64
Sleep Hours                              float64
Sleep Quality                              int64
Stress                                     int64
Travel Hours                             float64
Sport Specific Training Volume            object
dtype: object
```

In [402…
```
wellness.describe()
```

Out[402]:

	Date	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	
count	688	688.000000	688.000000	688.000000	688.000000	688.000000	68
mean	2023-07-02 20:26:30.697674496	35.997093	31.453488	62.681686	56.443314	7.522892	6
min	2023-05-05 00:00:00	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2023-05-30 00:00:00	19.000000	12.000000	48.000000	56.000000	7.000000	5
50%	2023-06-26 12:00:00	34.000000	31.000000	66.000000	60.000000	7.750000	6
75%	2023-08-05 06:00:00	51.000000	48.000000	86.000000	63.000000	8.250000	7
max	2023-09-20 00:00:00	100.000000	100.000000	100.000000	90.000000	13.000000	10
std	NaN	22.643236	22.685384	28.535054	16.122339	1.490790	2

In [403…
```
wellness.isnull().sum()
```

Out[403]:
```
Date                              0
Athlete                           0
Gender                            0
Fatigue                           0
Soreness                          0
Motivation                        0
Resting HR                        0
Sleep Hours                       0
Sleep Quality                     0
Stress                            0
Travel Hours                    581
Sport Specific Training Volume  342
dtype: int64
```

In [404…
```
results = pd.read_excel("Wellness Load and Results Data.xlsx", sheet_name=1)
```

In [405…
```
results.head()
```

Out[405]:

	Date	Athlete	Event	Time: Athlete	Time: Best	Rank: Athlete	Time: Athlete Heat 1	Time: Best Heat 1	Split Time: Athlete Heat 1	Split Rank: Athlete Heat 1	Time: Athlete Heat 2
0	2023-05-19	Athlete 2	Men's	342.36	334.17	10	171.48	166.32	14.70	3	170.88
1	2023-05-19	Athlete 4	Men's	355.50	334.17	16	179.97	166.32	14.67	2	175.53
2	2023-05-19	Athlete 7	Men's	340.11	334.17	3	170.01	166.32	15.03	10	170.10
3	2023-05-20	Athlete 2	Men's	340.08	337.86	3	169.80	169.08	14.94	2	170.28
4	2023-05-20	Athlete 4	Men's	351.51	337.86	14	174.24	169.08	14.97	3	177.27

In [406…
```
results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        77 non-null     datetime64[ns]
 1   Athlete                     77 non-null     object
 2   Event                       77 non-null     object
 3   Time: Athlete               65 non-null     float64
 4   Time: Best                  77 non-null     float64
 5   Rank: Athlete               77 non-null     int64
 6   Time: Athlete Heat 1        77 non-null     float64
 7   Time: Best Heat 1           77 non-null     float64
 8   Split Time: Athlete Heat 1  77 non-null     float64
 9   Split Rank: Athlete Heat 1  77 non-null     int64
 10  Time: Athlete Heat 2        63 non-null     float64
 11  Time: Best Heat 2           63 non-null     float64
 12  Split Time: Athlete Heat 2  63 non-null     float64
 13  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 8.5+ KB
```

In [407…

```
results.dtypes
```

Out[407]:

```
Date                          datetime64[ns]
Athlete                               object
Event                                 object
Time: Athlete                        float64
Time: Best                           float64
Rank: Athlete                          int64
Time: Athlete Heat 1                 float64
Time: Best Heat 1                    float64
Split Time: Athlete Heat 1           float64
Split Rank: Athlete Heat 1             int64
Time: Athlete Heat 2                 float64
Time: Best Heat 2                    float64
Split Time: Athlete Heat 2           float64
Split Rank: Athlete Heat 2           float64
dtype: object
```

In [408…

```
results.describe()
```

Out[408]:

	Date	Time: Athlete	Time: Best	Rank: Athlete	Time: Athlete Heat 1	Time: Best Heat 1	At H
count	77	65.000000	77.000000	77.000000	77.000000	77.000000	77.00
mean	2023-07-13 13:05:27.272727296	327.451385	322.274416	17.142857	167.005714	162.456623	15.14
min	2023-05-19 00:00:00	162.060000	159.360000	2.000000	98.190000	96.360000	12.93
25%	2023-06-11 00:00:00	319.680000	309.360000	12.000000	160.260000	154.920000	14.55
50%	2023-07-14 00:00:00	334.560000	326.070000	16.000000	167.070000	162.210000	15.00
75%	2023-08-12 00:00:00	350.010000	337.860000	23.000000	174.690000	169.080000	15.57
max	2023-09-08 00:00:00	422.490000	462.210000	42.000000	212.220000	209.430000	19.23
std	NaN	48.473188	46.548975	8.690155	18.875922	18.046627	0.98

In [409…
```
results.isnull().sum()
```

Out[409]:
```
Date                         0
Athlete                      0
Event                        0
Time: Athlete               12
Time: Best                   0
Rank: Athlete                0
Time: Athlete Heat 1         0
Time: Best Heat 1            0
Split Time: Athlete Heat 1   0
Split Rank: Athlete Heat 1   0
Time: Athlete Heat 2        14
Time: Best Heat 2           14
Split Time: Athlete Heat 2  14
Split Rank: Athlete Heat 2  14
dtype: int64
```

For a specific athlete, what data is observed?

In [410…
```
athlete_2 = wellness[wellness['Athlete'] == 'Athlete 2']
```

In [411…
```
athlete_2
```

Out[411]:

	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress
1	2023-05-05	Athlete 2	m	27	54	47	68	7.00	33	59
7	2023-05-06	Athlete 2	m	42	57	48	62	8.00	69	43
14	2023-05-07	Athlete 2	m	38	28	42	67	8.00	67	18
20	2023-05-08	Athlete 2	m	14	0	66	66	7.00	37	27
26	2023-05-09	Athlete 2	m	40	10	49	62	8.00	52	48
34	2023-05-10	Athlete 2	m	61	41	69	66	7.25	25	64
40	2023-05-11	Athlete 2	m	83	100	16	67	7.25	51	51
47	2023-05-12	Athlete 2	m	78	100	14	68	8.50	100	68
53	2023-05-13	Athlete 2	m	51	84	21	62	7.75	51	12
60	2023-05-14	Athlete 2	m	61	66	36	60	8.25	87	34
74	2023-05-16	Athlete 2	m	46	57	49	65	5.75	7	47
79	2023-05-17	Athlete 2	m	53	61	69	62	8.00	55	29
97	2023-05-20	Athlete 2	m	59	57	33	67	7.50	41	73
110	2023-05-22	Athlete 2	m	45	64	61	65	7.25	53	62
118	2023-05-23	Athlete 2	m	79	47	0	69	3.00	0	64
130	2023-05-25	Athlete 2	m	18	0	82	63	8.50	100	0
138	2023-05-26	Athlete 2	m	47	62	46	67	8.00	100	61
153	2023-05-28	Athlete 2	m	60	66	15	65	8.00	76	7

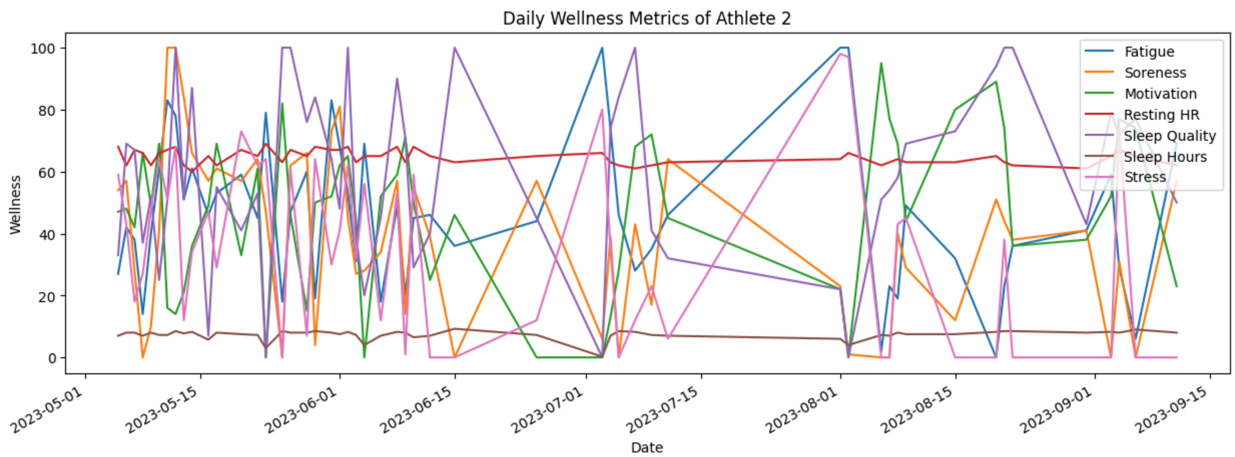
	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress
161	2023-05-29	Athlete 2	m	19	4	50	68	8.50	84	64
175	2023-05-31	Athlete 2	m	83	73	52	67	8.00	64	30
182	2023-06-01	Athlete 2	m	68	81	62	67	7.50	48	41
188	2023-06-02	Athlete 2	m	53	45	65	68	8.25	100	62
196	2023-06-03	Athlete 2	m	31	27	44	63	7.25	38	32
204	2023-06-04	Athlete 2	m	69	28	0	65	4.00	20	56
217	2023-06-06	Athlete 2	m	18	34	52	65	7.00	45	12
230	2023-06-08	Athlete 2	m	49	57	59	68	8.25	90	53
237	2023-06-09	Athlete 2	m	20	14	71	63	8.00	71	1
245	2023-06-10	Athlete 2	m	45	55	50	68	6.50	29	59
259	2023-06-12	Athlete 2	m	46	39	25	65	7.00	40	0
277	2023-06-15	Athlete 2	m	36	0	46	63	9.25	100	0
337	2023-06-25	Athlete 2	m	44	57	0	65	7.25	45	12
367	2023-07-03	Athlete 2	m	100	6	0	66	0.25	0	80
371	2023-07-04	Athlete 2	m	74	39	13	63	7.00	74	35
376	2023-07-05	Athlete 2	m	46	0	28	62	8.50	84	0
388	2023-07-07	Athlete 2	m	28	43	68	61	8.25	100	12
398	2023-07-09	Athlete 2	m	35	17	72	62	7.25	41	23

	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress
411	2023-07-11	Athlete 2	m	46	64	45	63	7.00	32	6
497	2023-08-01	Athlete 2	m	100	23	22	64	6.00	22	98
502	2023-08-02	Athlete 2	m	100	1	0	66	4.00	0	97
516	2023-08-06	Athlete 2	m	2	0	95	62	7.25	51	0
522	2023-08-07	Athlete 2	m	23	0	77	63	7.00	54	0
527	2023-08-08	Athlete 2	m	19	40	69	64	8.00	58	43
532	2023-08-09	Athlete 2	m	49	29	44	63	7.50	69	45
561	2023-08-15	Athlete 2	m	32	12	80	63	7.50	73	0
583	2023-08-20	Athlete 2	m	0	51	89	65	8.25	94	0
587	2023-08-21	Athlete 2	m	23	44	74	63	8.50	100	38
590	2023-08-22	Athlete 2	m	36	38	36	62	8.50	100	0
619	2023-08-31	Athlete 2	m	41	41	38	61	8.00	43	0
628	2023-09-03	Athlete 2	m	59	0	52	65	8.25	79	0
633	2023-09-04	Athlete 2	m	29	31	77	67	8.00	72	74
640	2023-09-06	Athlete 2	m	6	0	74	65	9.00	77	0
657	2023-09-11	Athlete 2	m	69	57	23	62	8.00	50	0

In [412…

```
# Plot the wellness metrics
athlete_2.plot(x='Date', y=['Fatigue', 'Soreness', 'Motivation', 'Resting HR',

# Set the x-axis label
```

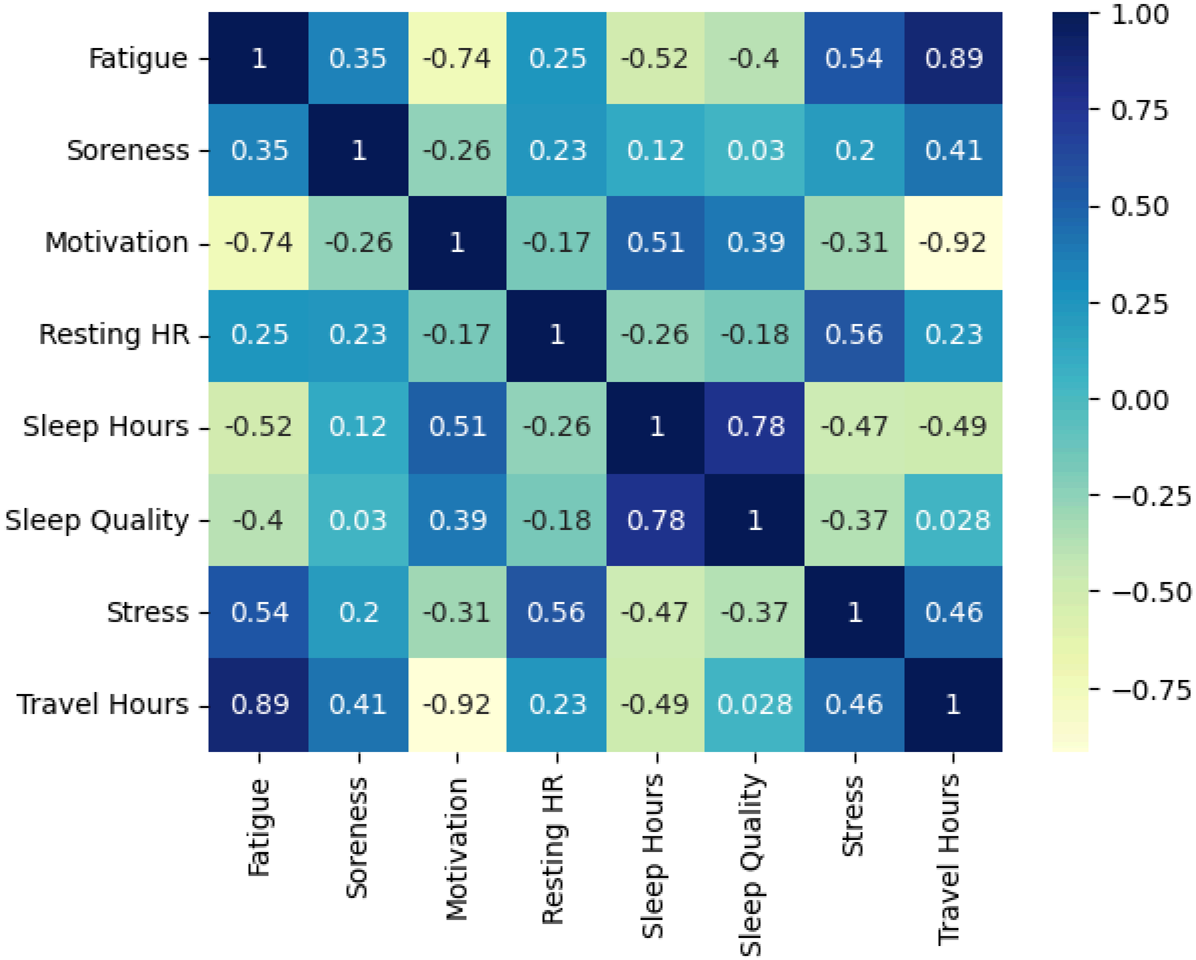
```python
plt.xlabel('Date')

# Set the y-axis label
plt.ylabel('Wellness')

# Set the title of the plot
plt.title('Daily Wellness Metrics of Athlete 2')

# Display the plot
plt.show()
```



In [413…

```python
sns.heatmap(athlete_2.corr(numeric_only=True), cmap="YlGnBu", annot=True)
```

Out[413]: <Axes: >



Join both dataframes to keep all data for all events & athletes (outer join)

In [414…
```
data = wellness.merge(results, on=['Date', 'Athlete'], how='outer')
```

In [415…
```
data.head()
```

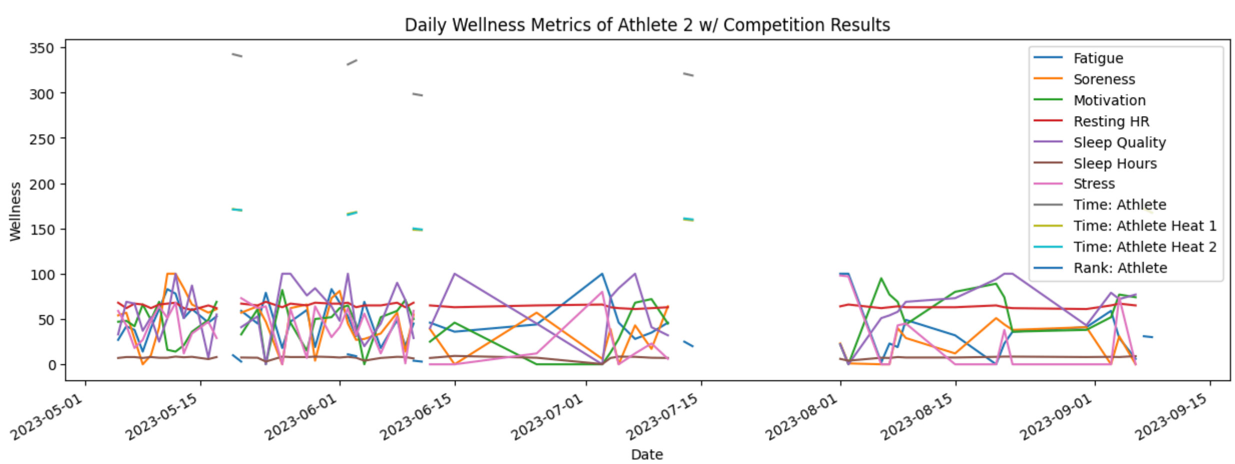
Out[415]:

	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress	...
0	2023-05-05	Athlete 1	m	47.0	30.0	31.0	0.0	8.50	61.0	39.0	...
1	2023-05-05	Athlete 2	m	27.0	54.0	47.0	68.0	7.00	33.0	59.0	...
2	2023-05-05	Athlete 3	m	35.0	36.0	90.0	65.0	7.25	94.0	32.0	...
3	2023-05-05	Athlete 4	m	55.0	1.0	84.0	58.0	7.00	64.0	0.0	...
4	2023-05-05	Athlete 5	m	50.0	67.0	74.0	80.0	6.25	73.0	40.0	...

5 rows × 24 columns

In [418…

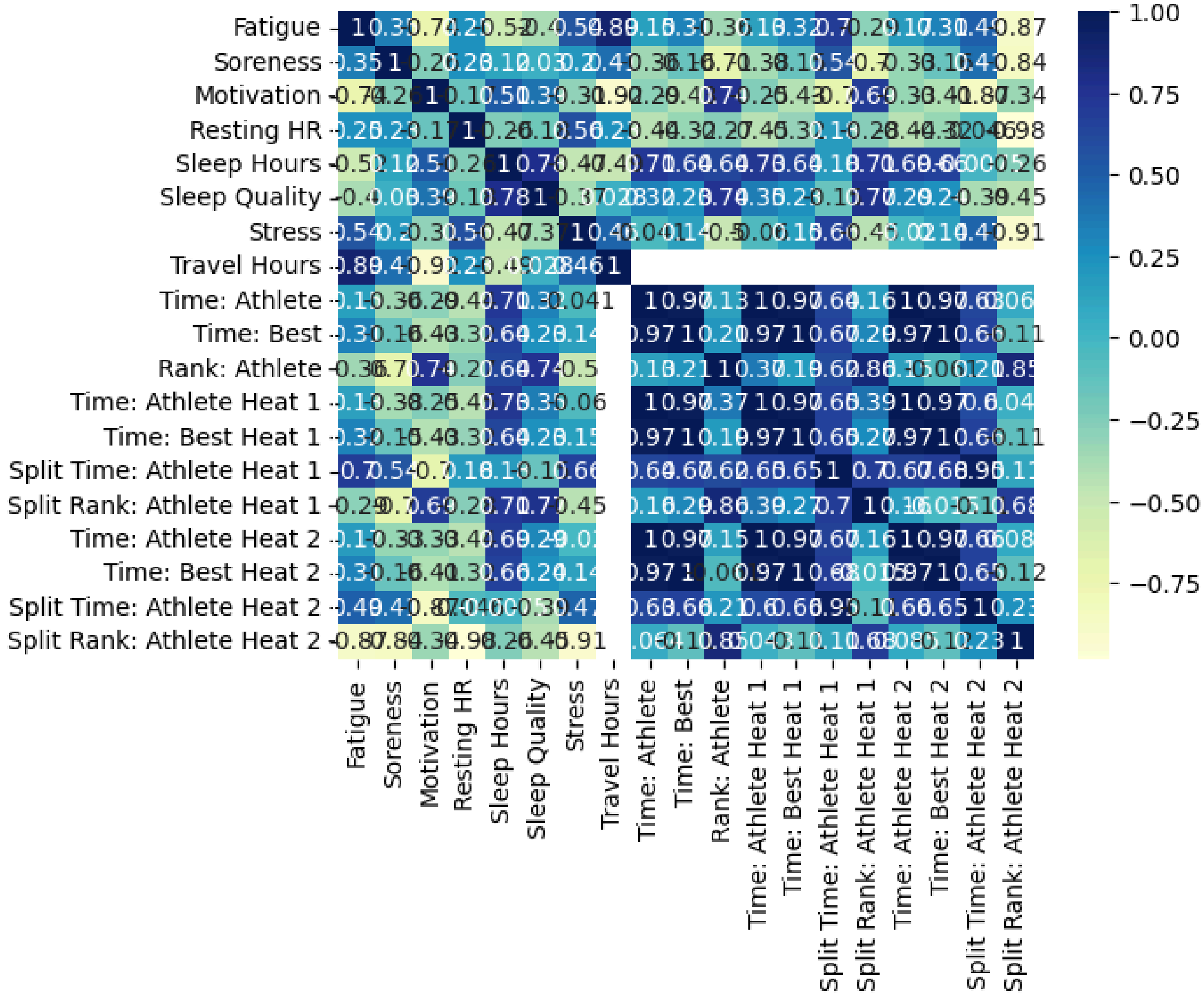
```
athlete_2 = data[data['Athlete'] == 'Athlete 2']

# Plot the wellness metrics
athlete_2.plot(x='Date', y=['Fatigue', 'Soreness', 'Motivation', 'Resting HR',

# Set the x-axis label
plt.xlabel('Date')

# Set the y-axis label
plt.ylabel('Wellness')

# Set the title of the plot
plt.title('Daily Wellness Metrics of Athlete 2 w/ Competition Results')

# Display the plot
plt.show()
```



In [419…

```
sns.heatmap(athlete_2.corr(numeric_only=True), cmap="YlGnBu", annot=True)
```

Out[419]: <Axes: >



With varying terrain/length/course conditions, as well as the potential variance of other variables outside of the athlete's individual wellness and performance (what competitors are there, their performance, etc.), the confidence of any conclusions drawn from the result data is limited.

For example, any correlation between wellness factors and improved (reduced) times may be incorrect, as the duration/distance of the course may be more a factor in athlete times across events than are the wellness metrics. If we choose to use ranking instead, improvements in ranking may also be more a factor of what other athletes competed and their resulting performances than a representation of "Athlete 2's" wellness metrics.

However, with these limitations in mind...what wellness metrics could potentially be used to predict an athlete's performance (by rank or time) in an event?

For Athlete 2:

In [420…

```
corr_matrix = athlete_2.corr(numeric_only=True)
corr_matrix['Rank: Athlete'].sort_values(ascending=False)
```

Out[420]:
```
Rank: Athlete                   1.000000
Split Rank: Athlete Heat 1      0.863119
Split Rank: Athlete Heat 2      0.850120
Sleep Quality                   0.737782
Motivation                      0.737459
Sleep Hours                     0.642794
Split Time: Athlete Heat 1      0.619287
Time: Athlete Heat 1            0.369432
Time: Best                      0.209201
Split Time: Athlete Heat 2      0.208182
Time: Best Heat 1               0.192826
Time: Athlete Heat 2            0.146348
Time: Athlete                   0.132303
Time: Best Heat 2              -0.060683
Resting HR                     -0.271919
Fatigue                        -0.356326
Stress                         -0.498296
Soreness                       -0.711283
Travel Hours                         NaN
Name: Rank: Athlete, dtype: float64
```

In [421…
```
corr_matrix['Time: Athlete'].sort_values(ascending=False)
```

Out[421]:
```
Time: Athlete                   1.000000
Time: Athlete Heat 1            0.999270
Time: Athlete Heat 2            0.999156
Time: Best Heat 2               0.974696
Time: Best                      0.973636
Time: Best Heat 1               0.971549
Sleep Hours                     0.710950
Split Time: Athlete Heat 1      0.643608
Split Time: Athlete Heat 2      0.630583
Sleep Quality                   0.320172
Split Rank: Athlete Heat 1      0.161461
Fatigue                         0.150024
Rank: Athlete                   0.132303
Split Rank: Athlete Heat 2      0.063560
Stress                         -0.041005
Motivation                     -0.288759
Soreness                       -0.357235
Resting HR                     -0.443148
Travel Hours                         NaN
Name: Time: Athlete, dtype: float64
```

For all athletes:

In [422…
```
corr_matrix = data.corr(numeric_only=True)
corr_matrix['Rank: Athlete'].sort_values(ascending=False)
```

Out[422]:
```
Rank: Athlete                  1.000000
Split Rank: Athlete Heat 1     0.592624
Split Rank: Athlete Heat 2     0.583739
Stress                         0.395331
Motivation                     0.335443
Sleep Quality                  0.324928
Split Time: Athlete Heat 2     0.119892
Time: Athlete Heat 1           0.088981
Time: Best                     0.083130
Time: Athlete                  0.057400
Split Time: Athlete Heat 1     0.040761
Time: Athlete Heat 2           0.039888
Sleep Hours                    0.034895
Time: Best Heat 1              0.011373
Resting HR                    -0.001620
Time: Best Heat 2             -0.005163
Soreness                      -0.329040
Fatigue                       -0.562172
Travel Hours                        NaN
Name: Rank: Athlete, dtype: float64
```

In [423…
```
corr_matrix['Time: Athlete'].sort_values(ascending=False)
```

Out[423]:
```
Time: Athlete                  1.000000
Time: Athlete Heat 2           0.998536
Time: Best                     0.996301
Time: Best Heat 2              0.993916
Time: Athlete Heat 1           0.807309
Time: Best Heat 1              0.795826
Split Time: Athlete Heat 2     0.595801
Split Time: Athlete Heat 1     0.395425
Stress                         0.220491
Motivation                     0.181881
Split Rank: Athlete Heat 1     0.122185
Sleep Quality                  0.102630
Rank: Athlete                  0.057400
Resting HR                     0.024302
Split Rank: Athlete Heat 2     0.018976
Fatigue                       -0.059377
Sleep Hours                   -0.182302
Soreness                      -0.289720
Travel Hours                        NaN
Name: Time: Athlete, dtype: float64
```

As a cyclist, I also know that some training platforms (Training Peaks) have a metric (TSS) based on power, that, when exponentially calculated over time, can be used to measure chronic training load (CTL), acute training load (ATL), and traiing stress balance (TSB), also sometimes referred to as form. While this dataset does not include TSS, I thought perhaps a variation of these metrics could be calculated to help measure the cumulative training load of athletes leading up to competition.

Though fatigue is self-reported here by the athlete, my thought was to use the fatigue metric, exponentially weighted, with the starting values used by Training Peaks in calculating their CTL from TSS (42).

In [424…
```
def calc_ctl(Fatigue:list, start_ctl, exponent):
    ctl = [start_ctl]
```

```
    for i in range(len(Fatigue)):
        ctl_value = Fatigue[i] * (1 - math.exp(-1 / exponent)) + ctl[-1] * matl
        ctl.append(ctl_value)
    return ctl[:-1]

def calculate_metrics_for_dataframe(data, start_ctl, ctl_exponent, start_atl, a
    metrics_by_athlete = {}
    for athlete in data['Athlete'].unique():
        athlete_df = data[data['Athlete'] == athlete].copy()
        Fatigue_values = athlete_df['Fatigue'].tolist()
        ctl_values = calc_ctl(Fatigue_values, start_ctl, ctl_exponent)
        atl_values = calc_ctl(Fatigue_values, start_atl, atl_exponent)
        tsb_values = [ctl - atl for ctl, atl in zip(ctl_values, atl_values)]
        metrics_by_athlete[athlete] = {'CTL': ctl_values, 'ATL': atl_values, '
        athlete_df['CTL'] = ctl_values
        athlete_df['ATL'] = atl_values
        athlete_df['TSB'] = tsb_values
        data.loc[athlete_df.index, ['CTL', 'ATL', 'TSB']] = athlete_df[['CTL',
    return metrics_by_athlete, data

start_ctl = 103
ctl_exponent = 42
start_atl = 50
atl_exponent = 7

metrics_by_athlete, df_with_metrics = calculate_metrics_for_dataframe(data, sta

df_with_metrics.head()
```

Out[424]:

	Date	Athlete	Gender	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	Sleep Quality	Stress	...
0	2023-05-05	Athlete 1	m	47.0	30.0	31.0	0.0	8.50	61.0	39.0	...
1	2023-05-05	Athlete 2	m	27.0	54.0	47.0	68.0	7.00	33.0	59.0	...
2	2023-05-05	Athlete 3	m	35.0	36.0	90.0	65.0	7.25	94.0	32.0	...
3	2023-05-05	Athlete 4	m	55.0	1.0	84.0	58.0	7.00	64.0	0.0	...
4	2023-05-05	Athlete 5	m	50.0	67.0	74.0	80.0	6.25	73.0	40.0	...

5 rows × 27 columns

In [425…

```
corr_matrix = data.corr(numeric_only=True)
corr_matrix['Rank: Athlete'].sort_values(ascending=False)
```

Out[425]:
```
Rank: Athlete                    1.000000
Split Rank: Athlete Heat 1       0.592624
Split Rank: Athlete Heat 2       0.583739
Stress                           0.395331
Motivation                       0.335443
Sleep Quality                    0.324928
Split Time: Athlete Heat 2       0.119892
Time: Athlete Heat 1             0.088981
Time: Best                       0.083130
Time: Athlete                    0.057400
Split Time: Athlete Heat 1       0.040761
Time: Athlete Heat 2             0.039888
Sleep Hours                      0.034895
Time: Best Heat 1                0.011373
Resting HR                      -0.001620
Time: Best Heat 2               -0.005163
Soreness                        -0.329040
TSB                             -0.401635
ATL                             -0.439617
Fatigue                         -0.562172
CTL                             -0.612215
Travel Hours                          NaN
Name: Rank: Athlete, dtype: float64
```

In [426…
```
corr_matrix['Time: Athlete'].sort_values(ascending=False)
```

Out[426]:
```
Time: Athlete                    1.000000
Time: Athlete Heat 2             0.998536
Time: Best                       0.996301
Time: Best Heat 2                0.993916
Time: Athlete Heat 1             0.807309
Time: Best Heat 1                0.795826
Split Time: Athlete Heat 2       0.595801
Split Time: Athlete Heat 1       0.395425
Stress                           0.220491
Motivation                       0.181881
Split Rank: Athlete Heat 1       0.122185
Sleep Quality                    0.102630
Rank: Athlete                    0.057400
TSB                              0.029144
Resting HR                       0.024302
Split Rank: Athlete Heat 2       0.018976
CTL                             -0.047689
Fatigue                         -0.059377
ATL                             -0.101475
Sleep Hours                     -0.182302
Soreness                        -0.289720
Travel Hours                          NaN
Name: Time: Athlete, dtype: float64
```

Now, I wanted to attempt to predict both Time and Rank based on these wellness metrics.

In order to do this, I aggregated the mean of the previous 7 day's fitness metrics (Stress, Motivation, Sleep Quality, Resting HR, Fatigue, Sleep Hours, Soreness, ATL, and CTL), as well as the prior day's TSB for each athlete's event date.

In [427…
```
data['Event_binary'] = 0

# Set Event_binary to 1 where 'Time: Athlete' is not NaN
```

```
# Also set the 'Athlete' column to the corresponding athlete for those rows
athletes_competing = data.loc[data['Rank: Athlete'].notna(), 'Athlete'].unique
for athlete in athletes_competing:
    data.loc[(data['Athlete'] == athlete) & (data['Rank: Athlete'].notna()), 'I
```

In [428…

```
data.describe()
```

Out[428]:

	Date	Fatigue	Soreness	Motivation	Resting HR	Sleep Hours	
count	716	688.000000	688.000000	688.000000	688.000000	688.000000	68
mean	2023-07-03 23:15:45.251396864	35.997093	31.453488	62.681686	56.443314	7.522892	6
min	2023-05-05 00:00:00	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2023-05-31 00:00:00	19.000000	12.000000	48.000000	56.000000	7.000000	5
50%	2023-06-29 00:00:00	34.000000	31.000000	66.000000	60.000000	7.750000	6
75%	2023-08-06 00:00:00	51.000000	48.000000	86.000000	63.000000	8.250000	7
max	2023-09-20 00:00:00	100.000000	100.000000	100.000000	90.000000	13.000000	10
std	NaN	22.643236	22.685384	28.535054	16.122339	1.490790	2

8 rows × 24 columns

In [429…

```
data.to_csv('data.csv')
```

In [430…

```
event_dates = data[data['Event_binary'] == 1]['Date']
```

In [431…

```
# Initialize an empty list to store aggregated wellness data
aggregated_wellness_data = []

# Define the subset of columns to aggregate
columns_to_aggregate = ['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Restin
                        'CTL', 'Fatigue', 'ATL', 'Sleep Hours', 'Soreness']

# Iterate over each unique athlete
for athlete in data['Athlete'].unique():
    # Iterate over each event date where 'Event_binary' is 1 for this athlete
    for event_date in data[(data['Athlete'] == athlete) & (data['Event_binary']
        # Filter wellness data for the previous 7 days leading up to the event
        filtered_wellness_data = data[(data['Athlete'] == athlete) &
                                      (data['Date'] >= event_date - pd.Timedel
                                      (data['Date'] <= event_date - pd.Timedel

        # Aggregate the subset of columns for this athlete and event date
        aggregated_data = filtered_wellness_data[columns_to_aggregate].mean().
        aggregated_data['Athlete'] = athlete
        aggregated_data['Date'] = event_date
```

```
        # Append the aggregated data to the list
        aggregated_wellness_data.append(aggregated_data)

# Concatenate the aggregated wellness data into a single DataFrame
aggregated_wellness_data = pd.concat(aggregated_wellness_data).reset_index(drop
```

In [432…

```
aggregated_wellness_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Stress         69 non-null     float64
 1   Motivation     69 non-null     float64
 2   Sleep Quality  69 non-null     float64
 3   TSB            69 non-null     float64
 4   Resting HR     69 non-null     float64
 5   CTL            69 non-null     float64
 6   Fatigue        69 non-null     float64
 7   ATL            69 non-null     float64
 8   Sleep Hours    69 non-null     float64
 9   Soreness       69 non-null     float64
 10  Athlete        77 non-null     object
 11  Date           77 non-null     datetime64[ns]
dtypes: datetime64[ns](1), float64(10), object(1)
memory usage: 7.3+ KB
```

In [433…

```
results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        77 non-null     datetime64[ns]
 1   Athlete                     77 non-null     object
 2   Event                       77 non-null     object
 3   Time: Athlete               65 non-null     float64
 4   Time: Best                  77 non-null     float64
 5   Rank: Athlete               77 non-null     int64
 6   Time: Athlete Heat 1        77 non-null     float64
 7   Time: Best Heat 1           77 non-null     float64
 8   Split Time: Athlete Heat 1  77 non-null     float64
 9   Split Rank: Athlete Heat 1  77 non-null     int64
 10  Time: Athlete Heat 2        63 non-null     float64
 11  Time: Best Heat 2           63 non-null     float64
 12  Split Time: Athlete Heat 2  63 non-null     float64
 13  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 8.5+ KB
```

In [434…

```
merged_data = pd.merge(aggregated_wellness_data, results, on=['Athlete', 'Date
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 24 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Stress                       69 non-null     float64
 1   Motivation                   69 non-null     float64
 2   Sleep Quality                69 non-null     float64
 3   TSB                          69 non-null     float64
 4   Resting HR                   69 non-null     float64
 5   CTL                          69 non-null     float64
 6   Fatigue                      69 non-null     float64
 7   ATL                          69 non-null     float64
 8   Sleep Hours                  69 non-null     float64
 9   Soreness                     69 non-null     float64
 10  Athlete                      77 non-null     object
 11  Date                         77 non-null     datetime64[ns]
 12  Event                        77 non-null     object
 13  Time: Athlete                65 non-null     float64
 14  Time: Best                   77 non-null     float64
 15  Rank: Athlete                77 non-null     int64
 16  Time: Athlete Heat 1         77 non-null     float64
 17  Time: Best Heat 1            77 non-null     float64
 18  Split Time: Athlete Heat 1   77 non-null     float64
 19  Split Rank: Athlete Heat 1   77 non-null     int64
 20  Time: Athlete Heat 2         63 non-null     float64
 21  Time: Best Heat 2            63 non-null     float64
 22  Split Time: Athlete Heat 2   63 non-null     float64
 23  Split Rank: Athlete Heat 2   63 non-null     float64
dtypes: datetime64[ns](1), float64(19), int64(2), object(2)
memory usage: 14.6+ KB
```

In [435…

```
merged_data.head()
```

Out[435]:

	Stress	Motivation	Sleep Quality	TSB	Resting HR	CTL	Fatigue	ATL
0	17.200000	45.000000	48.600000	26.019460	12.000000	77.110130	50.000000	51.090670
1	36.166667	45.333333	46.833333	20.011457	10.666667	69.387753	46.833333	49.376296
2	33.166667	44.166667	51.833333	32.094611	29.333333	83.225232	50.833333	51.130621
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 24 columns

In [436…

```
# Create a boolean mask indicating where NaN values exist in merged_data
nan_mask = merged_data.isna().any(axis=1)

# Filter merged_data to show only rows with NaN values
rows_with_nan = merged_data[nan_mask]
rows_with_nan[['Athlete', 'Date']]
```

Out[436]:

	Athlete	Date
3	Athlete 1	2023-07-14
4	Athlete 1	2023-07-28
7	Athlete 1	2023-09-07
16	Athlete 2	2023-09-07
17	Athlete 2	2023-09-08
24	Athlete 3	2023-08-04
25	Athlete 3	2023-08-18
26	Athlete 3	2023-09-08
34	Athlete 4	2023-07-14
37	Athlete 4	2023-09-07
38	Athlete 4	2023-09-08
42	Athlete 5	2023-07-14
44	Athlete 5	2023-07-28
45	Athlete 5	2023-08-04
46	Athlete 5	2023-08-18
52	Athlete 6	2023-07-14
53	Athlete 6	2023-08-11
60	Athlete 8	2023-08-04
61	Athlete 8	2023-08-18
76	Athlete 7	2023-09-08

In [437…

```
merged_data_cleaned = merged_data.dropna()
```

In [438…

```
merged_data_cleaned.to_csv('7_day_agg.csv')
```

In [439…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Rest
y = merged_data_cleaned['Rank: Athlete']
```

In [440…

```
# Add constant to the model
X = sm.add_constant(X)

# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Rank: Athlete   R-squared:                       0.260
Model:                            OLS   Adj. R-squared:                  0.118
Method:                 Least Squares   F-statistic:                     1.833
Date:                Thu, 21 Mar 2024   Prob (F-statistic):             0.0870
Time:                        19:39:51   Log-Likelihood:                -178.42
No. Observations:                  57   AIC:                             376.8
Df Residuals:                      47   BIC:                             397.3
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const            17.5748     11.239      1.564      0.125      -5.034      40.
184
Stress            0.0162      0.073      0.223      0.825      -0.130       0.
163
Motivation        0.0075      0.084      0.089      0.929      -0.162       0.
177
Sleep Quality     0.0599      0.114      0.526      0.601      -0.169       0.
289
TSB              -0.1284      0.072     -1.778      0.082      -0.274       0.
017
Resting HR        0.0556      0.065      0.860      0.394      -0.075       0.
186
CTL              -0.0414      0.048     -0.870      0.389      -0.137       0.
054
Fatigue          -0.1912      0.105     -1.814      0.076      -0.403       0.
021
ATL               0.0870      0.094      0.923      0.361      -0.103       0.
276
Sleep Hours      -0.0197      1.312     -0.015      0.988      -2.660       2.
621
Soreness         -0.0262      0.075     -0.349      0.729      -0.177       0.
125
==============================================================================
Omnibus:                        1.957   Durbin-Watson:                   2.098
Prob(Omnibus):                  0.376   Jarque-Bera (JB):                1.918
Skew:                          -0.410   Prob(JB):                        0.383
Kurtosis:                       2.633   Cond. No.                     1.25e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 7.77e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.26, which indicate that only approximtely 26% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 1.833 with a p-value of 0.08, which is over the standard significance level of 0.05, which would indicate that the model as a whole is not statistically significant. In addition, none of the individual coefficients are statistically significant when controlling for the other variables.

In [441…
```
def plot_feature_importance(importance,names,model_type):

    #Create arrays from feature importance and feature names
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    #Create a DataFrame using a Dictionary
    data={'feature_names':feature_names,'feature_importance':feature_importanc
    fi_df = pd.DataFrame(data)

    #Sort the DataFrame in order decreasing feature importance
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    #Define size of bar plot
    plt.figure(figsize=(10,8))

    #Plot Searborn bar chart
    sns.barplot(x=fi_df['feature_importance'], y=fi_df['feature_names'])

    #Add chart labels
    plt.title(model_type + 'Feature Importance')
    plt.xlabel('Feature Importance')
    plt.ylabel('Features')
```

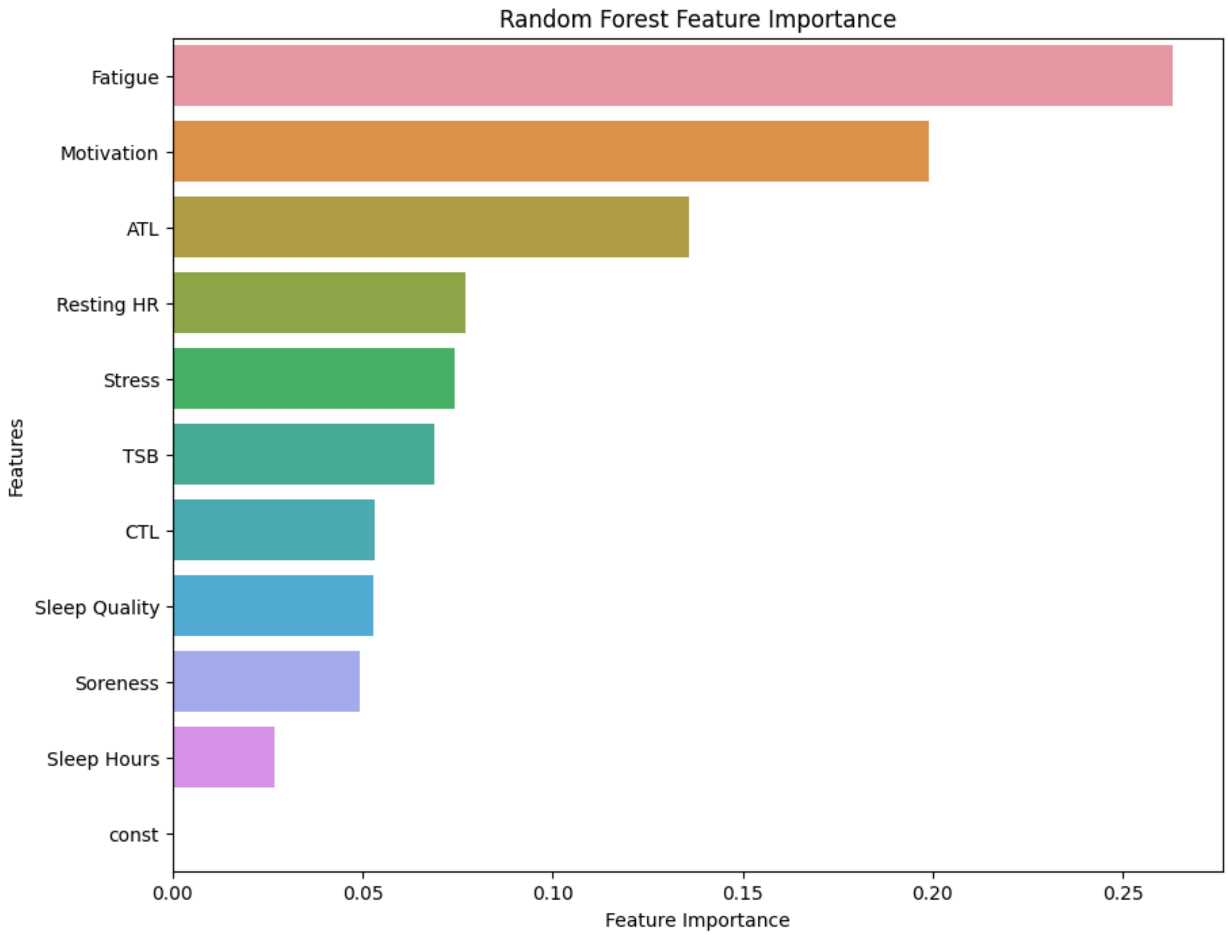
In [442…
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: 0.048086737035364036
Mean Squared Error: 10.004196731194025
R-squared: 0.7583983589304608
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

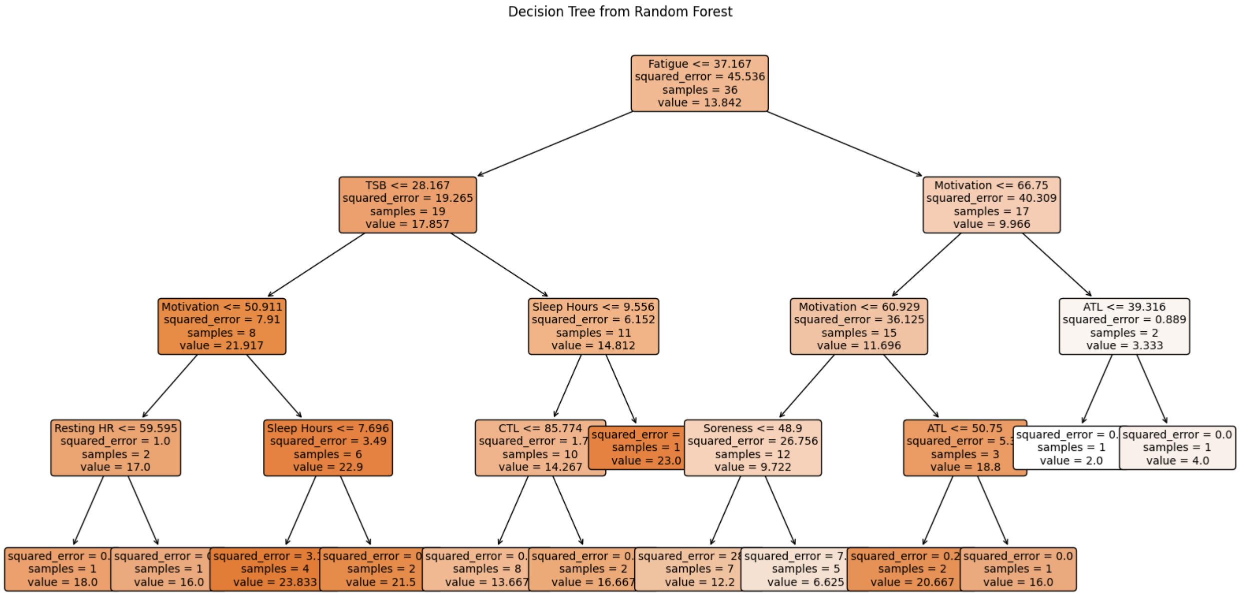
In [443…
```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Fores
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.py:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



In [444…

```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



The random forest model, when compared to the multiple regression model, produces a higher R-squared, indicating that approximately 75% of the variance in Athlete Rank can be explained by the model. The Out-of-Bag (OOB) score is a measure of prediction accuracy, with the low score of 0.05 indicating that the model is only slightly better than simply predicting the mean of the target value.

That being said, the random forest model does indicate that fatigue, motivation, and ATL are the three most important features for predicting Athlete Rank.

In [445…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Rest
y = merged_data_cleaned['Time: Athlete']
```

In [446…

```
# Add constant to the model
X = sm.add_constant(X)

# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Time: Athlete   R-squared:                       0.204
Model:                            OLS   Adj. R-squared:                  0.051
Method:                 Least Squares   F-statistic:                     1.336
Date:                Thu, 21 Mar 2024   Prob (F-statistic):              0.245
Time:                        19:40:03   Log-Likelihood:                -283.62
No. Observations:                  57   AIC:                             587.2
Df Residuals:                      47   BIC:                             607.7
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const           398.8362     71.155      5.605      0.000     255.691     541.
981
Stress            0.5616      0.462      1.217      0.230      -0.367       1.
490
Motivation       -0.9906      0.533     -1.857      0.070      -2.064       0.
082
Sleep Quality     1.8410      0.721      2.553      0.014       0.390       3.
292
TSB              -0.1029      0.457     -0.225      0.823      -1.023       0.
817
Resting HR       -0.0182      0.409     -0.044      0.965      -0.842       0.
806
CTL              -0.5945      0.301     -1.973      0.054      -1.201       0.
012
Fatigue           0.7702      0.668      1.154      0.254      -0.573       2.
113
ATL              -0.4915      0.596     -0.824      0.414      -1.691       0.
708
Sleep Hours     -11.4363      8.310     -1.376      0.175     -28.153       5.
281
Soreness         -0.3597      0.476     -0.756      0.454      -1.317       0.
598
==============================================================================
Omnibus:                       27.141   Durbin-Watson:                   2.422
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               47.948
Skew:                          -1.590   Prob(JB):                     3.87e-11
Kurtosis:                       6.175   Cond. No.                     1.25e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 7.77e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.20, which indicate that only 20% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 1.336 with a p-value of 0.245, which is over the standard significance level of 0.05, which would indicate that the model as a whole is not statistically significant.

Of all of the individual variables, Sleep Quality and CTL are the only ones that are less than or close to the standard significance level of 0.05 when controlling for the other variables; all of the other coefficients are not statistically significant.

In [447…

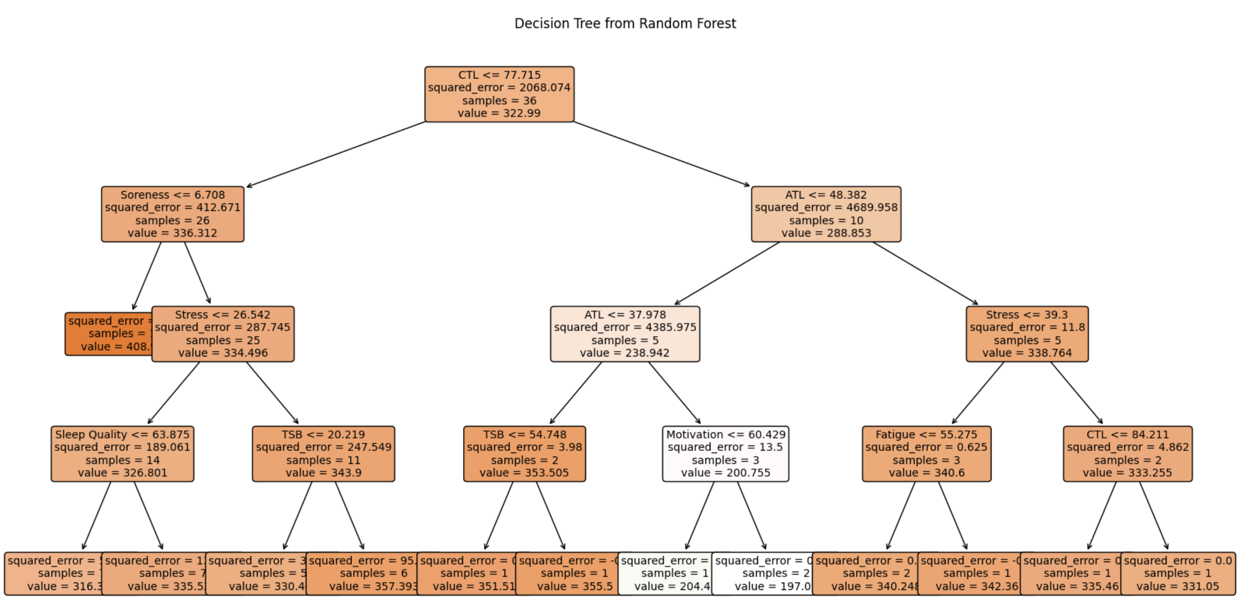
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: -0.4148757403134187
Mean Squared Error: 295.5928054068103
R-squared: 0.8084050614913577
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

In [448…

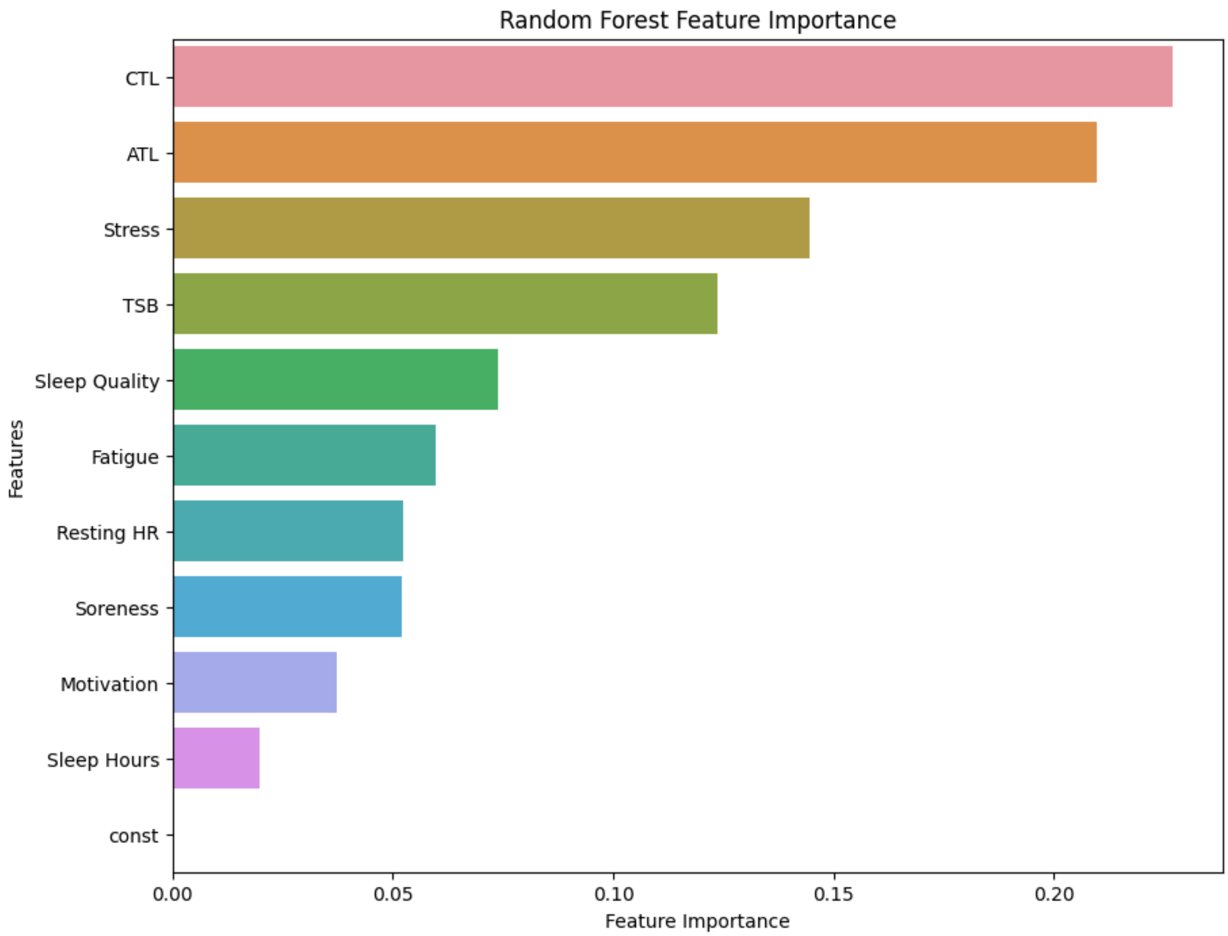
```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



In [449…

```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Fores
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Here, the random forest's R-squared value indicates that approximately 80% of the variance in Athlete Rank can be explained by the model. The Out-of-Bag (OOB) score of -.41 would indicate that simply predicting the mean of the target value would be more accurate than the model.

That being said, this random forest model does indicate that CTL, ATL, and Stress are the three most important features for predicting Athlete Time.

Attempting same process, but with 30 day aggregation

In [450…

```
# Initialize an empty list to store aggregated wellness data
aggregated_wellness_data = []
```

```
# Define the subset of columns to aggregate
columns_to_aggregate = ['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Restin
                        'CTL', 'Fatigue', 'ATL', 'Sleep Hours', 'Soreness']

# Iterate over each unique athlete
for athlete in data['Athlete'].unique():
    # Iterate over each event date where 'Event_binary' is 1 for this athlete
    for event_date in data[(data['Athlete'] == athlete) & (data['Event_binary']
        # Filter wellness data for the previous 30 days leading up to the event
        filtered_wellness_data = data[(data['Athlete'] == athlete) &
                                      (data['Date'] >= event_date - pd.Timedel
                                      (data['Date'] <= event_date - pd.Timedel

        # Aggregate the subset of columns for this athlete and event date
        aggregated_data = filtered_wellness_data[columns_to_aggregate].mean().
        aggregated_data['Athlete'] = athlete
        aggregated_data['Date'] = event_date

        # Append the aggregated data to the list
        aggregated_wellness_data.append(aggregated_data)

# Concatenate the aggregated wellness data into a single DataFrame
aggregated_wellness_data = pd.concat(aggregated_wellness_data).reset_index(drop
```

In [451…

```
aggregated_wellness_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Stress         76 non-null     float64
 1   Motivation     76 non-null     float64
 2   Sleep Quality  76 non-null     float64
 3   TSB            76 non-null     float64
 4   Resting HR     76 non-null     float64
 5   CTL            76 non-null     float64
 6   Fatigue        76 non-null     float64
 7   ATL            76 non-null     float64
 8   Sleep Hours    76 non-null     float64
 9   Soreness       76 non-null     float64
 10  Athlete        77 non-null     object
 11  Date           77 non-null     datetime64[ns]
dtypes: datetime64[ns](1), float64(10), object(1)
memory usage: 7.3+ KB
```

In [452…

```
results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 14 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Date                      77 non-null     datetime64[ns]
 1   Athlete                   77 non-null     object
 2   Event                     77 non-null     object
 3   Time: Athlete             65 non-null     float64
 4   Time: Best                77 non-null     float64
 5   Rank: Athlete             77 non-null     int64
 6   Time: Athlete Heat 1      77 non-null     float64
 7   Time: Best Heat 1         77 non-null     float64
 8   Split Time: Athlete Heat 1  77 non-null     float64
 9   Split Rank: Athlete Heat 1  77 non-null     int64
 10  Time: Athlete Heat 2      63 non-null     float64
 11  Time: Best Heat 2         63 non-null     float64
 12  Split Time: Athlete Heat 2  63 non-null     float64
 13  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 8.5+ KB
```

In [453…

```
merged_data = pd.merge(aggregated_wellness_data, results, on=['Athlete', 'Date
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Stress                    76 non-null     float64
 1   Motivation                76 non-null     float64
 2   Sleep Quality             76 non-null     float64
 3   TSB                       76 non-null     float64
 4   Resting HR                76 non-null     float64
 5   CTL                       76 non-null     float64
 6   Fatigue                   76 non-null     float64
 7   ATL                       76 non-null     float64
 8   Sleep Hours               76 non-null     float64
 9   Soreness                  76 non-null     float64
 10  Athlete                   77 non-null     object
 11  Date                      77 non-null     datetime64[ns]
 12  Event                     77 non-null     object
 13  Time: Athlete             65 non-null     float64
 14  Time: Best                77 non-null     float64
 15  Rank: Athlete             77 non-null     int64
 16  Time: Athlete Heat 1      77 non-null     float64
 17  Time: Best Heat 1         77 non-null     float64
 18  Split Time: Athlete Heat 1  77 non-null     float64
 19  Split Rank: Athlete Heat 1  77 non-null     int64
 20  Time: Athlete Heat 2      63 non-null     float64
 21  Time: Best Heat 2         63 non-null     float64
 22  Split Time: Athlete Heat 2  63 non-null     float64
 23  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(19), int64(2), object(2)
memory usage: 14.6+ KB
```

In [454…

```
merged_data.head()
```

Out[454]:

	Stress	Motivation	Sleep Quality	TSB	Resting HR	CTL	Fatigue	ATL
0	20.285714	45.238095	50.380952	34.304753	19.523810	84.532665	52.142857	50.227912
1	30.416667	41.750000	46.500000	21.446969	10.333333	70.976628	47.416667	49.529659
2	20.250000	46.500000	53.450000	40.219088	17.500000	89.182098	50.200000	48.963010
3	18.857143	45.142857	46.714286	27.100900	16.571429	76.374259	48.857143	49.273359
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 24 columns

In [455…

```
# Create a boolean mask indicating where NaN values exist in merged_data
nan_mask = merged_data.isna().any(axis=1)

# Filter merged_data to show only rows with NaN values
rows_with_nan = merged_data[nan_mask]
rows_with_nan[['Athlete', 'Date']]
```

Out[455]:

	Athlete	Date
4	Athlete 1	2023-07-28
16	Athlete 2	2023-09-07
17	Athlete 2	2023-09-08
34	Athlete 4	2023-07-14
37	Athlete 4	2023-09-07
38	Athlete 4	2023-09-08
42	Athlete 5	2023-07-14
44	Athlete 5	2023-07-28
45	Athlete 5	2023-08-04
46	Athlete 5	2023-08-18
52	Athlete 6	2023-07-14
53	Athlete 6	2023-08-11
60	Athlete 8	2023-08-04
61	Athlete 8	2023-08-18
76	Athlete 7	2023-09-08

In [456…

```
merged_data_cleaned = merged_data.dropna()
```

In [457…

```
merged_data_cleaned.to_csv('30_day_agg.csv')
```

In [458…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Resti
y = merged_data_cleaned['Rank: Athlete']
```

In [459…

```
# Add constant to the model
X = sm.add_constant(X)

# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Rank: Athlete   R-squared:                       0.298
Model:                             OLS   Adj. R-squared:                  0.176
Method:                  Least Squares   F-statistic:                     2.451
Date:                 Thu, 21 Mar 2024   Prob (F-statistic):             0.0207
Time:                         19:40:14   Log-Likelihood:                -194.16
No. Observations:                   62   AIC:                             408.3
Df Residuals:                       52   BIC:                             429.6
Df Model:                            9
Covariance Type:            nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const            14.4368     12.717      1.135      0.261     -11.081       39.
955
Stress           -0.0746      0.116     -0.644      0.522      -0.307        0.
158
Motivation        0.0836      0.096      0.866      0.390      -0.110        0.
277
Sleep Quality    -0.2311      0.177     -1.308      0.197      -0.585        0.
123
TSB              -0.0669      0.135     -0.495      0.623      -0.338        0.
204
Resting HR        0.1749      0.075      2.332      0.024       0.024        0.
325
CTL              -0.0875      0.087     -1.005      0.320      -0.262        0.
087
Fatigue           0.1792      0.264      0.678      0.501      -0.351        0.
710
ATL              -0.0206      0.211     -0.098      0.923      -0.445        0.
403
Sleep Hours       1.7151      1.847      0.929      0.357      -1.990        5.
421
Soreness         -0.2660      0.127     -2.100      0.041      -0.520       -0.
012
==============================================================================
Omnibus:                         5.513   Durbin-Watson:                   1.829
Prob(Omnibus):                   0.064   Jarque-Bera (JB):                4.554
Skew:                           -0.600   Prob(JB):                        0.103
Kurtosis:                        3.568   Cond. No.                     2.50e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 2.15e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.29, which indicate that only 29% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 2.451 with a p-value of 0.02, which is less than the standard significance level of 0.05, which would indicate that the model as a whole is statistically significant.

Interestingly, only Resting HR and Soreness, individually, when controlling for other variables, are statistically significant when predicting Athlete Rank. None of the individual coefficients are statistically significant when controlling for the other variables.

In [460…

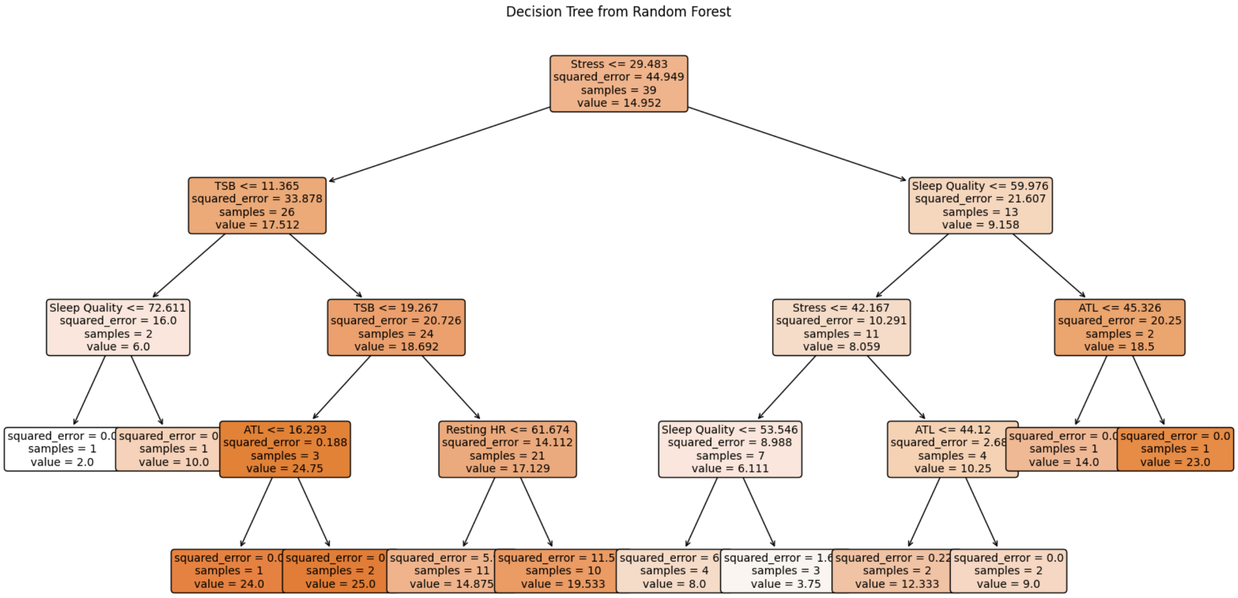
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: -0.1644741520725419
Mean Squared Error: 11.331981115077719
R-squared: 0.741144904882584
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

In [461…

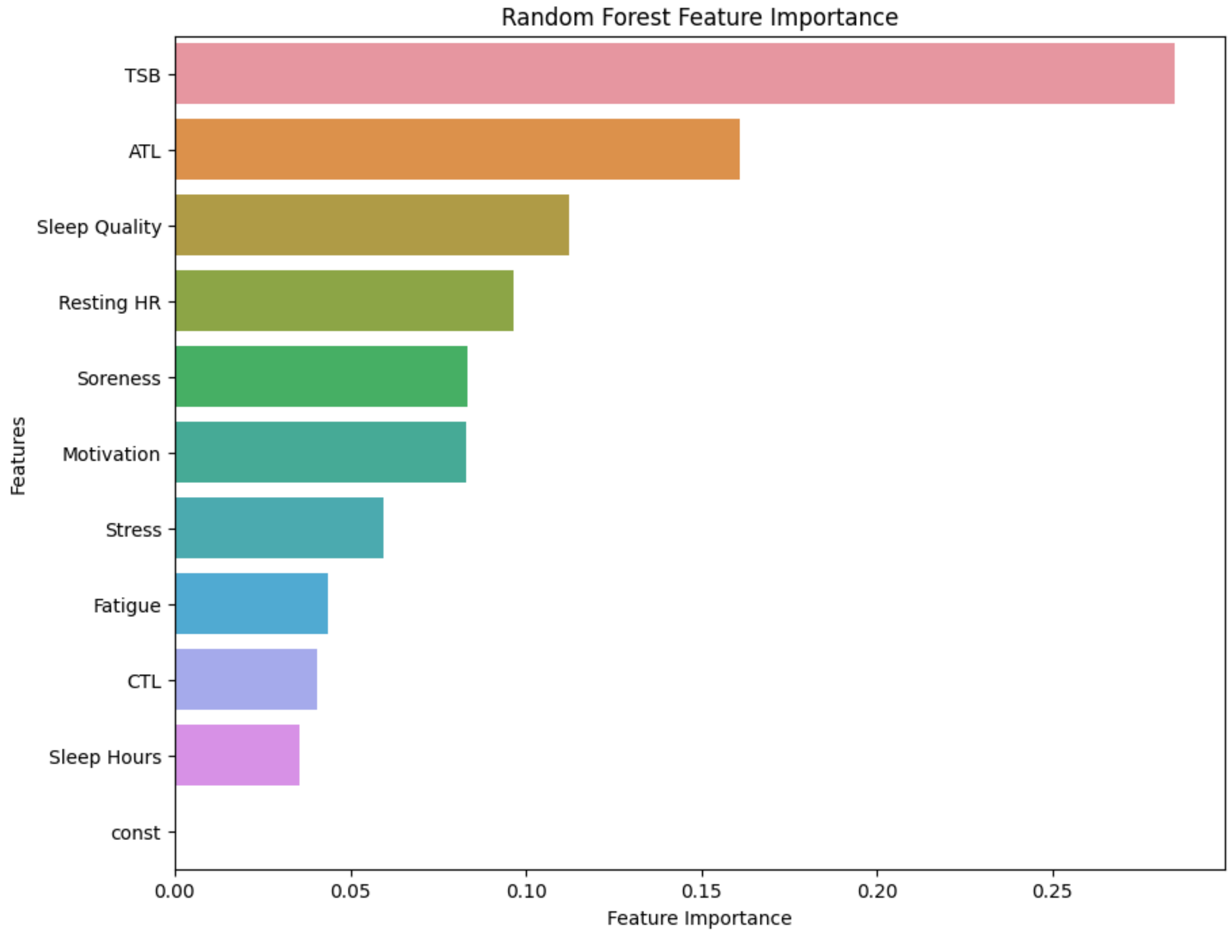
```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



In [462…

```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Fores
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Here, the random forest's R-squared value indicates that approximately 74% of the variance in Athlete Rank can be explained by the model. The Out-of-Bag (OOB) score of -.16 would indicate that simply predicting the mean of the target value would be more accurate than the model. However, this random forest model does have a much lower mean squared error (MSE) than either of the prior models.

This random forest model indicates that TSB is by far the most important feature for predicting Athlete Rank, followed by ATL and Sleep Quality.

In [463…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Rest
y = merged_data_cleaned['Time: Athlete']
```

In [464…

```
# Add constant to the model
X = sm.add_constant(X)

# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Time: Athlete   R-squared:                       0.097
Model:                            OLS   Adj. R-squared:                 -0.059
Method:                 Least Squares   F-statistic:                    0.6196
Date:                Thu, 21 Mar 2024   Prob (F-statistic):              0.775
Time:                        19:40:20   Log-Likelihood:                -312.10
No. Observations:                  62   AIC:                             644.2
Df Residuals:                      52   BIC:                             665.5
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const           368.9848     85.216      4.330      0.000     197.987     539.
983
Stress           -0.9916      0.776     -1.277      0.207      -2.549       0.
566
Motivation        0.0862      0.646      0.133      0.894      -1.211       1.
383
Sleep Quality    -1.4409      1.183     -1.218      0.229      -3.816       0.
934
TSB               0.0252      0.906      0.028      0.978      -1.792       1.
843
Resting HR        0.0625      0.503      0.124      0.902      -0.946       1.
071
CTL              -0.3506      0.583     -0.601      0.550      -1.521       0.
820
Fatigue           0.8566      1.771      0.484      0.631      -2.697       4.
410
ATL              -0.3758      1.416     -0.265      0.792      -3.217       2.
465
Sleep Hours      12.1886     12.374      0.985      0.329     -12.642      37.
019
Soreness         -0.4528      0.849     -0.533      0.596      -2.156       1.
251
==============================================================================
Omnibus:                       24.783   Durbin-Watson:                   2.308
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               52.337
Skew:                          -1.244   Prob(JB):                     4.32e-12
Kurtosis:                       6.751   Cond. No.                     2.50e+16
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 2.15e-27. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.097, which indicate that only approximately 10% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 0.6196 with a p-value of 0.775, which is over the standard significance level of 0.05, which would indicate that the model as a whole is not statistically significant.

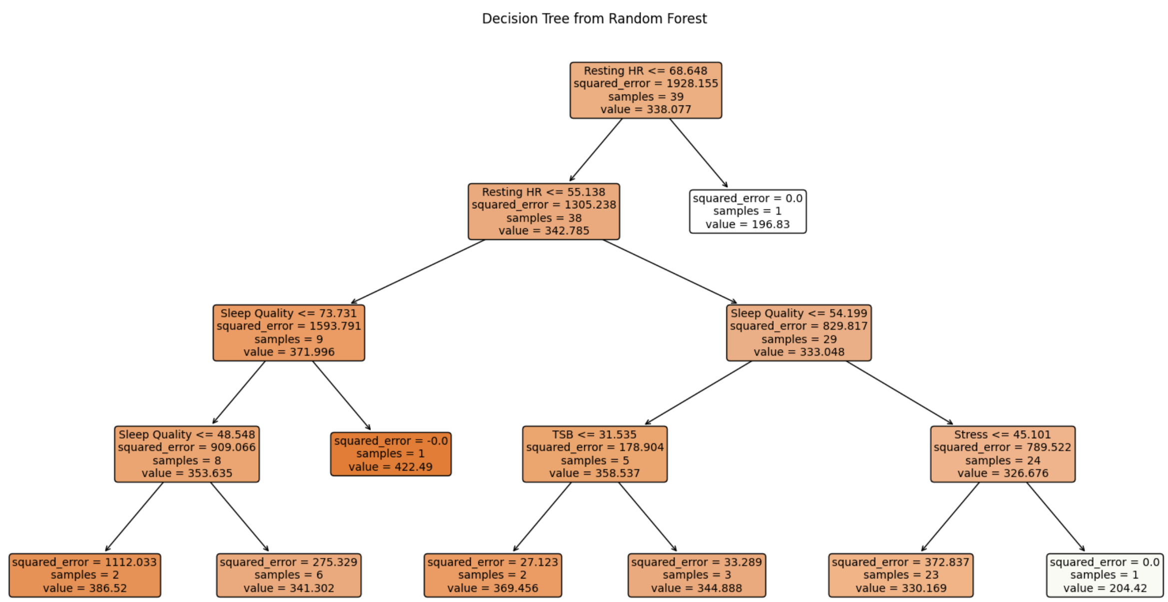
None of the coefficients are statistically significant.

In [465…
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: -0.4778548265598479
Mean Squared Error: 402.795356669819
R-squared: 0.7364280633806937
```

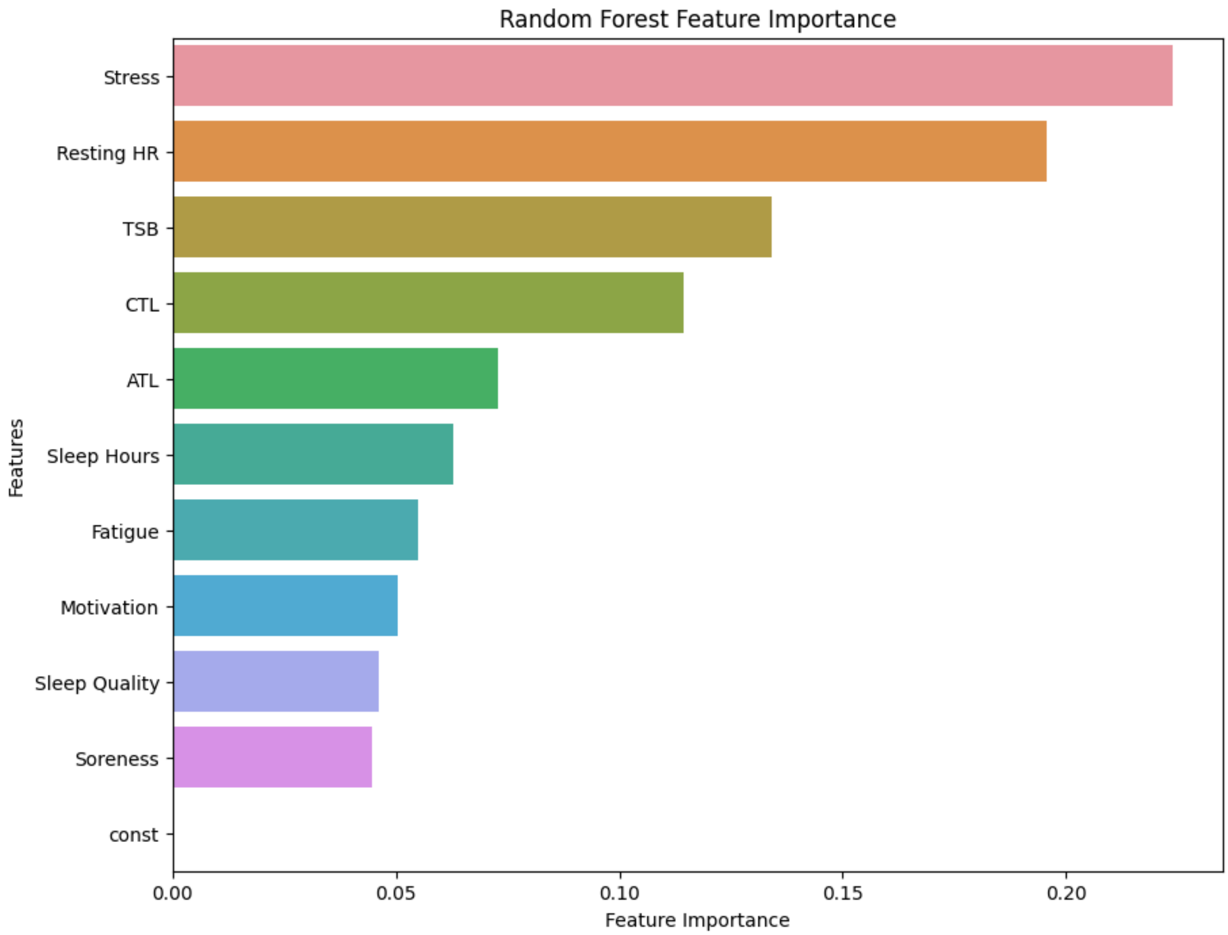
```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

In [466…
```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



In [467…
```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Fores
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Here, the random forest's R-squared value indicates that approximately 74% of the variance in Athlete Rank can be explained by the model (which is nearly identical to the 7-day aggregation data when predicting Athlete Time). The Out-of-Bag (OOB) score of -.47 would indicate, however, that simply predicting the mean of the target value would be more accurate than the model.

That being said, this random forest model does indicate that Stress, Resting Heart Rate, and TSB are the three most important features for predicting Athlete Rank.

Attempting same process, but with no date aggregation

In [468…

```python
# Initialize an empty list to store aggregated wellness data
aggregated_wellness_data = []
```

```
# Define the subset of columns to aggregate
columns_to_aggregate = ['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Restin
                        'CTL', 'Fatigue', 'ATL', 'Sleep Hours', 'Soreness']

# Iterate over each unique athlete
for athlete in data['Athlete'].unique():
    # Iterate over each event date where 'Event_binary' is 1 for this athlete
    for event_date in data[(data['Athlete'] == athlete) & (data['Event_binary']
        # Filter wellness data for the previous 7 days leading up to the event
        filtered_wellness_data = data[(data['Athlete'] == athlete)]

        # Aggregate the subset of columns for this athlete and event date
        aggregated_data = filtered_wellness_data[columns_to_aggregate].mean().
        aggregated_data['Athlete'] = athlete
        aggregated_data['Date'] = event_date

        # Append the aggregated data to the list
        aggregated_wellness_data.append(aggregated_data)

# Concatenate the aggregated wellness data into a single DataFrame
aggregated_wellness_data = pd.concat(aggregated_wellness_data).reset_index(drop
```

In [469…

```
aggregated_wellness_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 12 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Stress         77 non-null     float64
 1   Motivation     77 non-null     float64
 2   Sleep Quality  77 non-null     float64
 3   TSB            77 non-null     float64
 4   Resting HR     77 non-null     float64
 5   CTL            77 non-null     float64
 6   Fatigue        77 non-null     float64
 7   ATL            77 non-null     float64
 8   Sleep Hours    77 non-null     float64
 9   Soreness       77 non-null     float64
 10  Athlete        77 non-null     object
 11  Date           77 non-null     datetime64[ns]
dtypes: datetime64[ns](1), float64(10), object(1)
memory usage: 7.3+ KB
```

In [470…

```
results.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Date                        77 non-null     datetime64[ns]
 1   Athlete                     77 non-null     object
 2   Event                       77 non-null     object
 3   Time: Athlete               65 non-null     float64
 4   Time: Best                  77 non-null     float64
 5   Rank: Athlete               77 non-null     int64
 6   Time: Athlete Heat 1        77 non-null     float64
 7   Time: Best Heat 1           77 non-null     float64
 8   Split Time: Athlete Heat 1  77 non-null     float64
 9   Split Rank: Athlete Heat 1  77 non-null     int64
 10  Time: Athlete Heat 2        63 non-null     float64
 11  Time: Best Heat 2           63 non-null     float64
 12  Split Time: Athlete Heat 2  63 non-null     float64
 13  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(9), int64(2), object(2)
memory usage: 8.5+ KB
```

In [471…

```
merged_data = pd.merge(aggregated_wellness_data, results, on=['Athlete', 'Date
merged_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 77 entries, 0 to 76
Data columns (total 24 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Stress                      77 non-null     float64
 1   Motivation                  77 non-null     float64
 2   Sleep Quality               77 non-null     float64
 3   TSB                         77 non-null     float64
 4   Resting HR                  77 non-null     float64
 5   CTL                         77 non-null     float64
 6   Fatigue                     77 non-null     float64
 7   ATL                         77 non-null     float64
 8   Sleep Hours                 77 non-null     float64
 9   Soreness                    77 non-null     float64
 10  Athlete                     77 non-null     object
 11  Date                        77 non-null     datetime64[ns]
 12  Event                       77 non-null     object
 13  Time: Athlete               65 non-null     float64
 14  Time: Best                  77 non-null     float64
 15  Rank: Athlete               77 non-null     int64
 16  Time: Athlete Heat 1        77 non-null     float64
 17  Time: Best Heat 1           77 non-null     float64
 18  Split Time: Athlete Heat 1  77 non-null     float64
 19  Split Rank: Athlete Heat 1  77 non-null     int64
 20  Time: Athlete Heat 2        63 non-null     float64
 21  Time: Best Heat 2           63 non-null     float64
 22  Split Time: Athlete Heat 2  63 non-null     float64
 23  Split Rank: Athlete Heat 2  63 non-null     float64
dtypes: datetime64[ns](1), float64(19), int64(2), object(2)
memory usage: 14.6+ KB
```

In [472…

```
merged_data.head()
```

Out[472]:

	Stress	Motivation	Sleep Quality	TSB	Resting HR	CTL	Fatigue	ATL	
0	25.020408	43.469388	52.55102	31.736531	14.653061	80.963027	48.836735	49.226495	7
1	25.020408	43.469388	52.55102	31.736531	14.653061	80.963027	48.836735	49.226495	7
2	25.020408	43.469388	52.55102	31.736531	14.653061	80.963027	48.836735	49.226495	7
3	25.020408	43.469388	52.55102	31.736531	14.653061	80.963027	48.836735	49.226495	7
4	25.020408	43.469388	52.55102	31.736531	14.653061	80.963027	48.836735	49.226495	7

5 rows × 24 columns

In [473…

```
# Create a boolean mask indicating where NaN values exist in merged_data
nan_mask = merged_data.isna().any(axis=1)

# Filter merged_data to show only rows with NaN values
rows_with_nan = merged_data[nan_mask]
rows_with_nan[['Athlete', 'Date']]
```

Out[473]:

	Athlete	Date
16	Athlete 2	2023-09-07
17	Athlete 2	2023-09-08
34	Athlete 4	2023-07-14
37	Athlete 4	2023-09-07
38	Athlete 4	2023-09-08
42	Athlete 5	2023-07-14
44	Athlete 5	2023-07-28
45	Athlete 5	2023-08-04
46	Athlete 5	2023-08-18
52	Athlete 6	2023-07-14
53	Athlete 6	2023-08-11
60	Athlete 8	2023-08-04
61	Athlete 8	2023-08-18
76	Athlete 7	2023-09-08

In [474…

```
merged_data_cleaned = merged_data.dropna()
```

In [475…

```
merged_data_cleaned.to_csv('no_date_agg.csv')
```

In [476…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Resti
y = merged_data_cleaned['Rank: Athlete']
```

In [477…

```
# Add constant to the model
X = sm.add_constant(X)
```

```
# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           Rank: Athlete   R-squared:                       0.205
Model:                             OLS   Adj. R-squared:                  0.104
Method:                  Least Squares   F-statistic:                     2.023
Date:                 Thu, 21 Mar 2024   Prob (F-statistic):             0.0684
Time:                         19:40:26   Log-Likelihood:                -200.72
No. Observations:                   63   AIC:                             417.4
Df Residuals:                       55   BIC:                             434.6
Df Model:                            7
Covariance Type:             nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const            -1.8397      0.773     -2.380      0.021      -3.389      -0.
290
Stress           -0.2487      0.263     -0.945      0.349      -0.776       0.
279
Motivation        0.3738      0.253      1.480      0.145      -0.132       0.
880
Sleep Quality    -1.7596      0.949     -1.854      0.069      -3.662       0.
143
TSB              -5.0200      2.133     -2.353      0.022      -9.295      -0.
745
Resting HR        0.0672      0.103      0.653      0.516      -0.139       0.
274
CTL               3.8105      1.605      2.374      0.021       0.594       7.
027
Fatigue         -11.7269      4.849     -2.418      0.019     -21.445      -2.
008
ATL               8.8305      3.735      2.364      0.022       1.345      16.
316
Sleep Hours      15.9857      6.689      2.390      0.020       2.581      29.
390
Soreness         -0.9623      0.411     -2.339      0.023      -1.787      -0.
138
==============================================================================
Omnibus:                         0.034   Durbin-Watson:                   1.941
Prob(Omnibus):                   0.983   Jarque-Bera (JB):                0.207
Skew:                            0.002   Prob(JB):                        0.902
Kurtosis:                        2.719   Cond. No.                     1.03e+18
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 1.23e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.205, which indicate that only approximately 21% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 2.023 with a p-value of 0.068, which is greater than the standard significance level of 0.05, which would indicate that the model as a whole is not statistically significant.

However, TSB, CTL, Fatigue, ATL, Sleep Hours, and Soreness, individually, when controlling for other variables, are statistically significant when predicting Athlete Rank.

In [478…

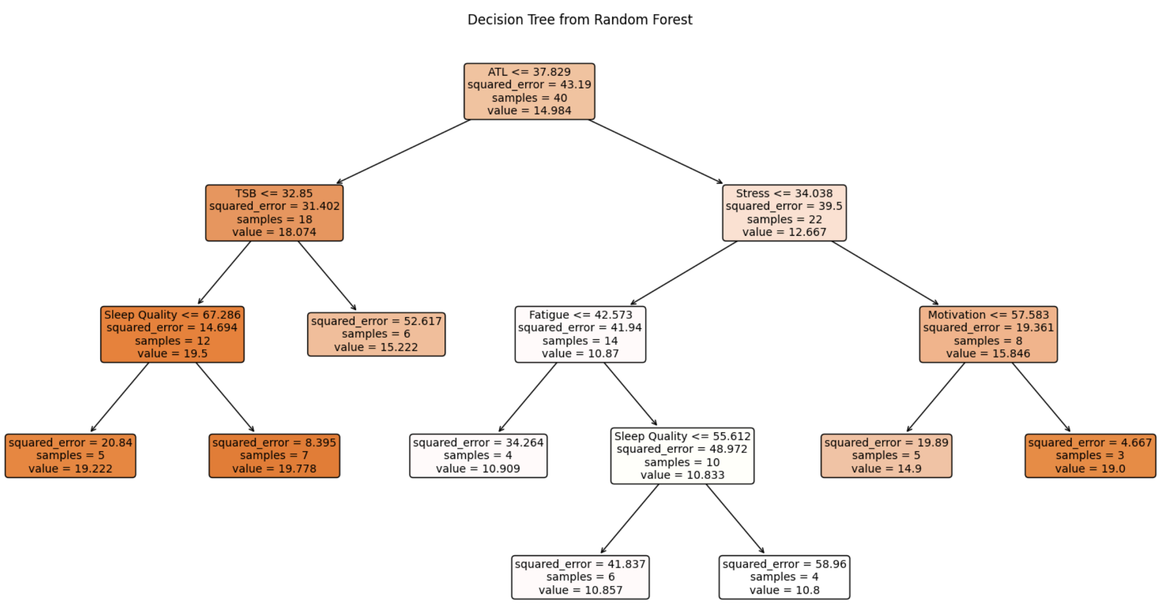
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: -0.15291612927295062
Mean Squared Error: 34.80776376256519
R-squared: 0.1921029322837089
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

In [479…

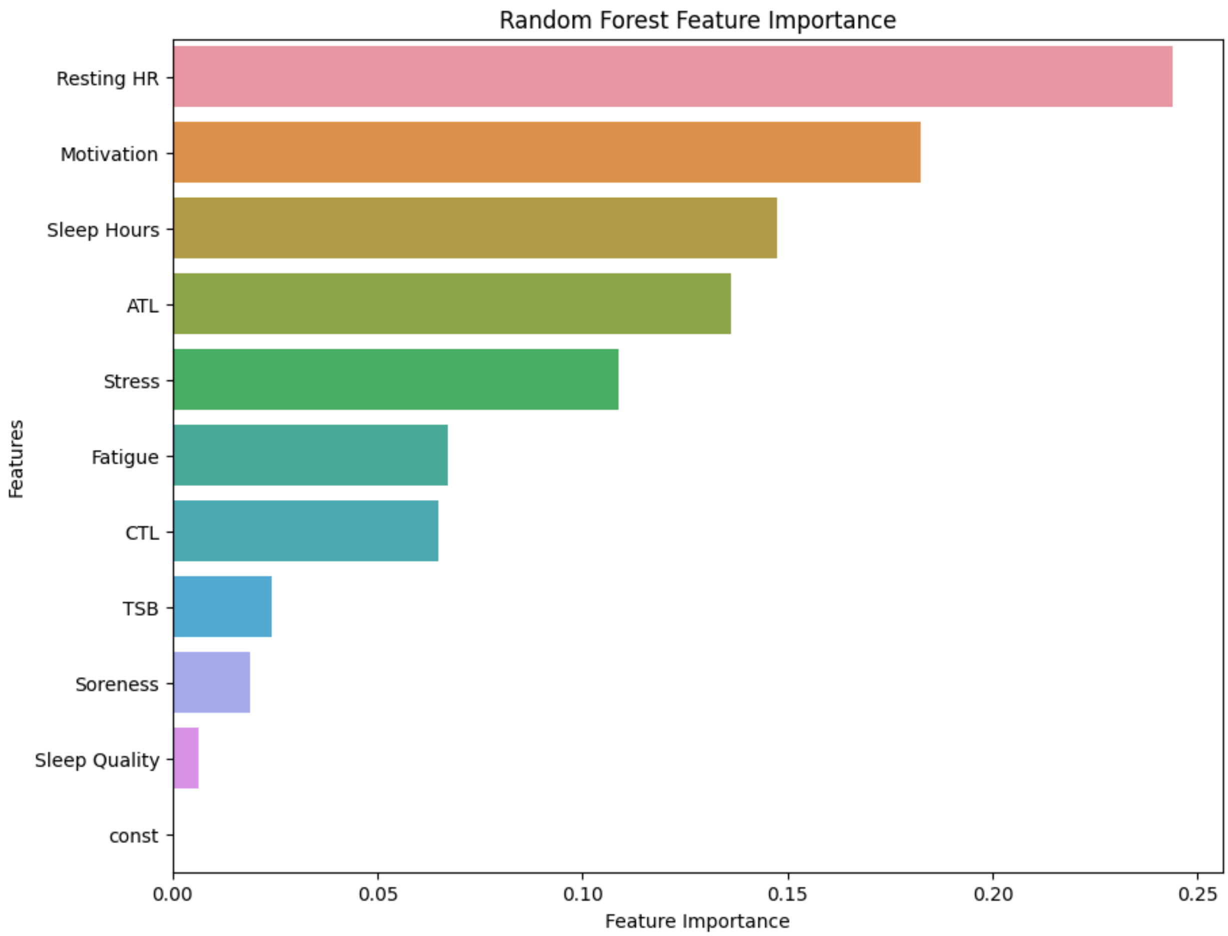
```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



In [480…

```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Forest
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Here, the random forest's R-squared value indicates that only approximately 19% of the variance in Athlete Rank can be explained by the model. The Out-of-Bag (OOB) score of -.15 would indicate that simply predicting the mean of the target value would be more accurate than the model. By all metrics, either the 7- or 30-day aggregations of the data are more accurate when predicting for Athlete Rank than not aggregating the data at all.

That being said, this random forest model does indicate that Resting Heart Rate, Motivation, and Sleep Hours are the three most important features for predicting Athlete Rank.

In [481…

```
X = merged_data_cleaned[['Stress', 'Motivation', 'Sleep Quality', 'TSB', 'Resti
y = merged_data_cleaned['Time: Athlete']
```

In [482…

```
# Add constant to the model
X = sm.add_constant(X)

# Reset the indices of X and y to ensure alignment
X.reset_index(drop=True, inplace=True)
y.reset_index(drop=True, inplace=True)

# Fit the regression model
model = sm.OLS(y, X).fit()

# Print summary statistics
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          Time: Athlete   R-squared:                       0.085
Model:                            OLS   Adj. R-squared:                 -0.031
Method:                 Least Squares   F-statistic:                    0.7314
Date:                Thu, 21 Mar 2024   Prob (F-statistic):              0.646
Time:                        19:40:30   Log-Likelihood:                -317.19
No. Observations:                  63   AIC:                             650.4
Df Residuals:                      55   BIC:                             667.5
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
===
                    coef    std err          t      P>|t|      [0.025      0.9
75]
------------------------------------------------------------------------------
---
const             1.1653      4.911      0.237      0.813      -8.676      11.
007
Stress            3.3721      1.671      2.017      0.049       0.022       6.
722
Motivation       -0.1742      1.604     -0.109      0.914      -3.390       3.
041
Sleep Quality     6.2476      6.029      1.036      0.305      -5.835      18.
330
TSB               2.6272     13.551      0.194      0.847     -24.529      29.
783
Resting HR       -1.4700      0.654     -2.248      0.029      -2.780      -0.
160
CTL              -2.0277     10.196     -0.199      0.843     -22.461      18.
405
Fatigue           7.0089     30.803      0.228      0.821     -54.723      68.
740
ATL              -4.6549     23.726     -0.196      0.845     -52.202      42.
893
Sleep Hours      -9.1917     42.487     -0.216      0.830     -94.337      75.
953
Soreness         -0.2977      2.614     -0.114      0.910      -5.536       4.
940
==============================================================================
Omnibus:                       25.616   Durbin-Watson:                   2.271
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               48.798
Skew:                          -1.347   Prob(JB):                     2.53e-11
Kurtosis:                       6.367   Cond. No.                     1.03e+18
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The smallest eigenvalue is 1.23e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Here, the R-squared is 0.085, which indicate that only approximately 9% of the variability in Athlete Rank can be explained by the 7-day aggregation of daily wellness metrics.

The F-statistic is 0.7314 with a p-value of 0.646, which is over the standard significance level of 0.05, which would indicate that the model as a whole is not statistically significant.

Stress and Resting HR, individually, when controlling for other variables, are statistically significant when predicting Athlete Rank.

In [483…

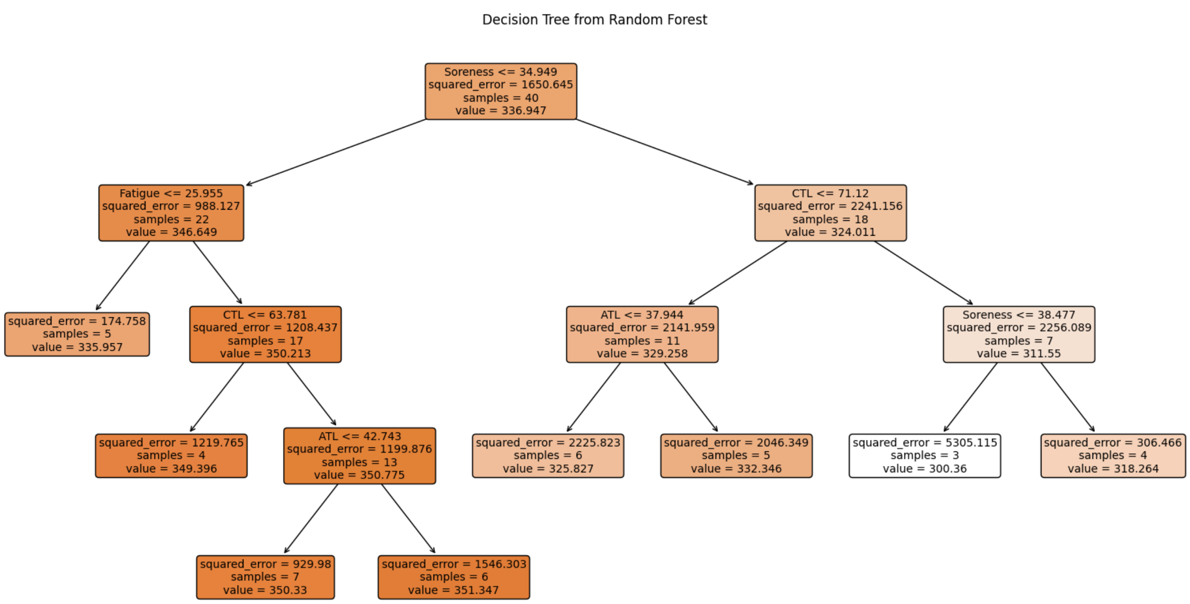
```
# Fitting Random Forest Regression to the dataset
regressor = RandomForestRegressor(n_estimators=10, max_depth=4, random_state=0

regressor.fit(X, y)

oob_score = regressor.oob_score_
print(f'Out-of-Bag Score: {oob_score}')

predictions = regressor.predict(X)

mse = mean_squared_error(y, predictions)
print(f'Mean Squared Error: {mse}')

r2 = r2_score(y, predictions)
print(f'R-squared: {r2}')
```

```
Out-of-Bag Score: -0.3901814566871973
Mean Squared Error: 1405.5433303944112
R-squared: 0.06983880854397273
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/sklearn/utils/vali
dation.py:623: FutureWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

In [484…

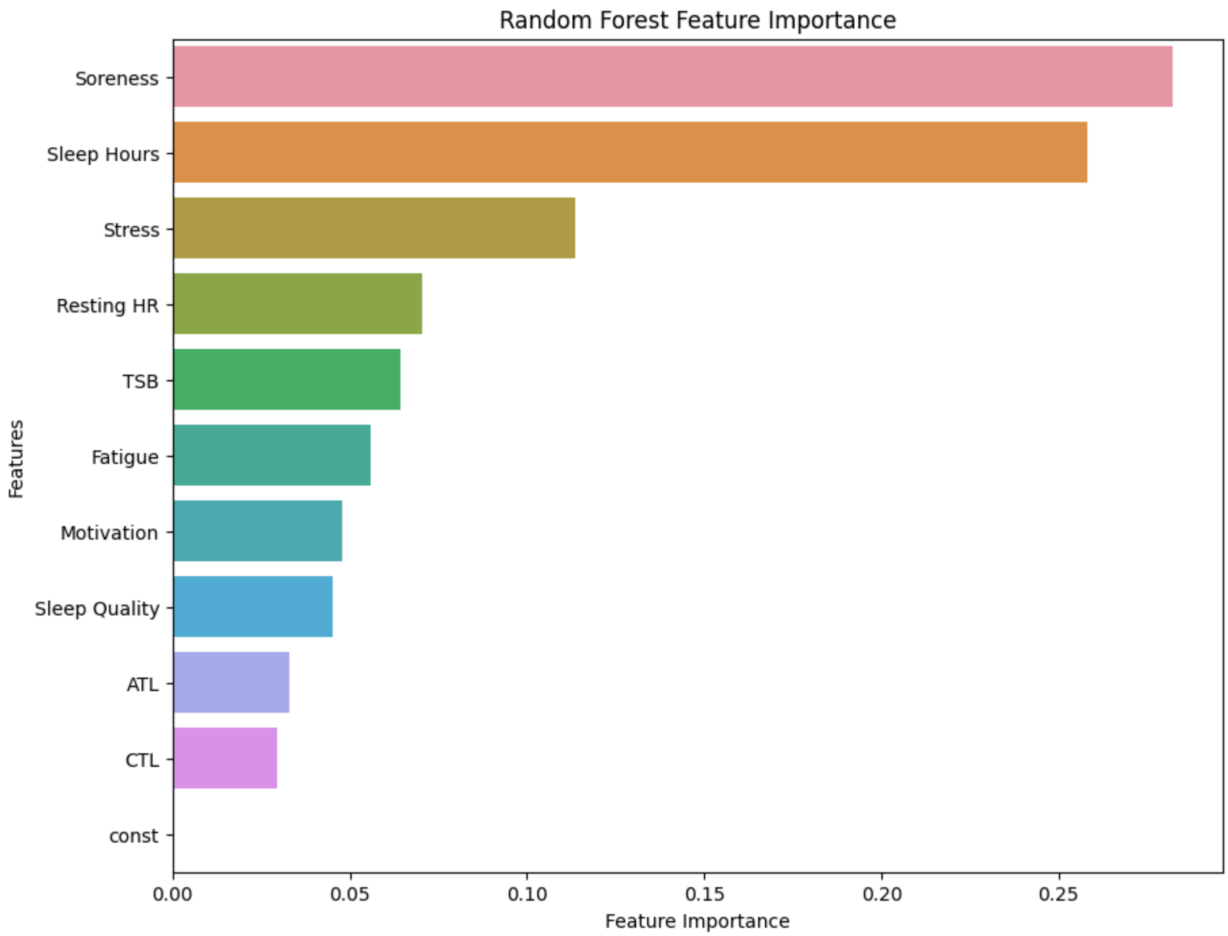
```
tree_to_plot = regressor.estimators_[0]

plt.figure(figsize=(20, 10))
plot_tree(tree_to_plot, feature_names=X.columns.tolist(), filled=True, rounded=
plt.title("Decision Tree from Random Forest")
plt.show()
```



In [485…

```
plot_feature_importance(regressor.feature_importances_,X.columns,'Random Fores
```

```
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
/Users/abigailsnyder/anaconda3/lib/python3.10/site-packages/seaborn/_oldcore.p
y:1498: FutureWarning: is_categorical_dtype is deprecated and will be removed
in a future version. Use isinstance(dtype, CategoricalDtype) instead
  if pd.api.types.is_categorical_dtype(vector):
```



Here, the random forest's R-squared value indicates that only approximately 6% of the variance in Athlete Rank can be explained by the model. The Out-of-Bag (OOB) score of -.39 would indicate that simply predicting the mean of the target value would be more accurate than the model. The Mean Squared Error (MSE) of this model is also significantly larger than previous models, indicating, by all counts, that this model is less accurate when predicting Athlete Time than the models using data aggregated over time.

That being said, this random forest model does indicate that Soreness, Sleep Hours, and Stress are the three most important features for predicting Athlete Time.

Conclusions

Both multiple regression models would indicate that the 7-day aggregation of wellness data is not statistically significant when trying to predict event performance (either by rank or time). The 30-day aggregation of wellness data is only statistically significant when predicting athlete time in an event (which, due to the potential variance in event distance may be in itself an unreliable outcome). Without aggregating the data, the model is statistically significant when predicting rank, but still only explains a small amount of the variance in rank.

The random forest models seem to perform significantly better when using 7- or 30- day aggregated data in comparison to the non-date-aggregated data. The best performing model (as measured by MSE) was the random forest model using 7-day aggregated data to predict athlete rank. This model indicated that fatigue, motivation, and acute training load (ATL) are the three most important features when predicting athlete rank.

This can lead to several conclusions:

1. For the athlete (and coach), the numbers are not the final say. If wellness numbers seem to be poor prior to an event, the athlete can still enter the event with confidence--knowing that wellness metrics do not consistently correlate with performance outcomes.
2. For researchers (and athletes and coaches), more data should be gathered, especially as regards the mental state of athletes going into events. There may be a more definitive correlation between mental state and performance than between wellness metrics and performance.

In [ ]:

```

```