2022 Wellness Tracking

Objectives

The objective of this project is to record and analyze the relationships between input variables and target variables throughout the 2022 Midland Rockhound season in the Texas League of AA baseball to create actionable ideas on how to increase performance. This project also strives to incorporate mental variables as well as physical ones, in the hopes of observing the impacts of mental health and training in sport as well as physical.

Methods

This project will involve various data cleaning and manipulation methods, which will be more specifically explained upon implementation. However, the focus of this project is on machine learning implementations, and analyzing feature impact on target variables.

Imports and Data Cleaning

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Importing CSV data

filename = 'Data/michael_wellness_data.csv'
def read_csv(filename):
    df = pd.read_csv(filename)
    return df

data = read_csv(filename)
column_headers = data.columns
```

In [3]:
```
drop_headers = ['Thought of the Day', 'Game Notes']

data = data.drop(drop_headers, axis=1)
data = data.dropna(axis=0, thresh=3)
column_headers = data.columns
num_entries = data.shape[0]
data
```

Out[3]:

	Date	Day of Week	Whoop Recovery	HRV	Hours of Sleep	Body Weight	Body Fat %	Body Water %	Hydration Score (urine)	Sick Feeling	...	Lift?	Protein Shake?	Fun Rating	Family Love Rating	Tired Rating	Strain	Alcohol (# of Drinks)	CBD (mg)	Melatonin	Current BA
0	4/14/2022	2	99.0	61.0	8.95	174.5	NaN	NaN	7.0	2.0	...	0.0	0.0	7.0	7.0	5.0	15.7	0.0	0.0	0.0	NaN
1	4/15/2022	3	81.0	51.0	9.05	NaN	NaN	NaN	7.0	1.0	...	1.0	0.0	2.0	6.0	8.0	17.5	0.0	0.0	0.0	NaN
2	4/16/2022	4	82.0	51.0	8.33	NaN	NaN	NaN	5.0	1.0	...	0.0	0.0	4.0	6.0	6.0	13.9	2.0	0.0	0.0	NaN
3	4/17/2022	5	76.0	51.0	7.32	NaN	NaN	NaN	5.0	2.0	...	0.0	0.0	8.0	8.0	3.0	5.5	1.0	0.0	0.0	NaN
4	4/18/2022	6	77.0	51.0	6.92	NaN	NaN	NaN	4.0	1.0	...	0.0	0.0	7.0	8.0	4.0	5.2	0.0	0.0	0.0	NaN
5	4/19/2022	0	85.0	52.0	8.72	NaN	NaN	NaN	4.0	1.0	...	1.0	1.0	7.0	7.0	3.0	8.7	0.0	0.0	0.0	NaN
6	4/20/2022	1	52.0	42.0	8.53	NaN	NaN	NaN	5.0	1.0	...	0.0	1.0	6.0	8.0	4.0	14.8	0.0	0.0	0.0	NaN
7	4/21/2022	2	83.0	52.0	8.30	NaN	NaN	NaN	4.0	1.0	...	0.0	2.0	3.0	6.0	7.0	15.3	0.0	0.0	0.0	NaN
8	4/22/2022	3	60.0	46.0	9.08	179.5	NaN	NaN	6.0	1.0	...	1.0	1.0	9.0	7.0	3.0	9.6	0.0	0.0	0.0	NaN
9	4/23/2022	4	73.0	49.0	8.48	179.4	NaN	NaN	4.0	2.0	...	0.0	1.0	7.0	6.0	3.0	14.5	0.0	0.0	0.0	NaN
10	4/24/2022	5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.0	1.0	6.0	6.0	6.0	5.5	0.0	0.0	0.0	NaN
11	4/25/2022	6	63.0	43.0	6.28	176.0	NaN	NaN	4.0	2.0	...	0.0	0.0	9.0	8.0	5.0	9.9	2.0	0.0	0.0	NaN
12	4/26/2022	0	51.0	43.0	7.73	174.0	NaN	NaN	3.0	1.0	...	1.0	0.0	7.0	7.0	7.0	17.5	0.0	0.0	0.0	NaN
13	4/27/2022	1	94.0	53.0	8.67	NaN	NaN	NaN	3.0	1.0	...	0.0	1.0	8.0	7.0	7.0	14.7	0.0	0.0	0.0	NaN
14	4/28/2022	2	95.0	61.0	8.30	174.0	NaN	NaN	5.0	1.0	...	0.0	1.0	9.0	9.0	3.0	15.0	0.0	0.0	0.0	NaN
15	4/29/2022	3	70.0	52.0	6.43	NaN	NaN	NaN	5.0	1.0	...	1.0	1.0	3.0	8.0	4.0	11.6	0.0	0.0	0.0	NaN
16	4/30/2022	4	67.0	49.0	7.48	NaN	NaN	NaN	6.0	1.0	...	0.0	1.0	7.0	7.0	6.0	15.7	0.0	0.0	0.0	NaN
17	5/1/2022	5	72.0	53.0	7.30	NaN	NaN	NaN	5.0	1.0	...	0.0	0.0	3.0	7.0	9.0	15.6	0.0	0.0	0.0	NaN
18	5/2/2022	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.0	0.0	9.0	8.0	5.0	6.0	2.0	0.0	0.0	NaN
19	5/3/2022	0	77.0	52.0	8.50	NaN	NaN	NaN	4.0	3.0	...	0.0	0.0	6.0	7.0	3.0	7.1	0.0	0.0	0.0	NaN
20	5/4/2022	1	31.0	38.0	8.37	NaN	NaN	NaN	5.0	9.0	...	1.0	0.0	4.0	7.0	6.0	14.3	0.0	0.0	0.0	NaN
21	5/5/2022	2	42.0	44.0	7.97	NaN	NaN	NaN	6.0	7.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

22 rows × 29 columns

In [4]:
```
target_variables = ['Whoop Recovery', 'HRV', 'Body Weight', 'Current BA']

morning_variables = ['Day of Week', 'Whoop Recovery', 'HRV', 'Hours of Sleep',
                     'Body Weight', 'Body Fat %', 'Body Water %', 'Hydration Score (urine)',
                     'Sick Feeling', 'Soreness/Fatigue Score', 'Fitness Rating',
                     'Positivity Score (1-10)', 'Confidence Score', 'Date']
night_variables = ['Caffeine in mg',
                   'Water Before Coffee?', 'Breakfast Before Coffee?',
                   'Devo Quality (1-10)', 'Game?', 'Lift?', 'Protein Shake?', 'Fun Rating',
                   'Family Love Rating', 'Tired Rating', 'Strain', 'Alcohol (# of Drinks)',
                   'CBD (mg)', 'Melatonin', 'Current BA']
```

In [5]:
```
print('Enter desired target value according to the keys below:')
dict_targets = { i : target_variables[i - 1] for i in range(1, len(target_variables) + 1) }
print(dict_targets)
desired_target = input()
if desired_target == '':
    desired_target = 1
target = dict_targets[int(desired_target)]
print(f'Selected Target Variable: {target}')
```

```
Enter desired target value according to the keys below:
{1: 'Whoop Recovery', 2: 'HRV', 3: 'Body Weight', 4: 'Current BA'}

Selected Target Variable: Whoop Recovery
```

In [6]:
```
def update_df(df, morning, night, target):
    if target in night:
        night.remove(target)
    df['Date'] = pd.to_datetime(df['Date'])
    curr_day = df.drop(night, axis=1)
    prev_day = df.drop(morning, axis=1)
    prev_day = prev_day.shift(periods=1)
    return pd.concat([curr_day, prev_day], axis=1, join='inner')

adjusted_df = update_df(data, morning=morning_variables, night=night_variables, target=target)
```

In [7]:
```
def drop_nans(df, threshold=0.5):
    df_length = df.shape[0]
    df = df.dropna(thresh=threshold * df_length, axis=1).dropna(how='any', axis=0)
    return df

adjusted_df = drop_nans(adjusted_df)
X = adjusted_df.drop([target, 'Date'], axis=1)
y = adjusted_df[target]
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X.iloc[0:10,:]
```

Out[7]:

	Day of Week	HRV	Hours of Sleep	Hydration Score (urine)	Sick Feeling	Soreness/Fatigue Score	Fitness Rating	Positivity Score (1-10)	Confidence Score	Caffeine in mg	...	Game?	Lift?	Protein Shake?	Fun Rating	Family Love Rating	Tired Rating	Strain	Alcohol (# of Drinks)	CBD (mg)	Melatonin
1	3	51.0	9.05	7.0	1.0	3.0	6.0	8.0	8.0	300.0	...	1.0	0.0	0.0	7.0	7.0	5.0	15.7	0.0	0.0	0.0
2	4	51.0	8.33	5.0	1.0	3.0	5.0	2.0	3.0	250.0	...	1.0	1.0	0.0	2.0	6.0	8.0	17.5	0.0	0.0	0.0
3	5	51.0	7.32	5.0	2.0	2.0	4.0	5.0	8.0	250.0	...	1.0	0.0	0.0	4.0	6.0	6.0	13.9	2.0	0.0	0.0
4	6	51.0	6.92	4.0	1.0	1.0	5.0	7.0	7.0	250.0	...	0.0	0.0	0.0	8.0	8.0	3.0	5.5	1.0	0.0	0.0
5	0	52.0	8.72	4.0	1.0	2.0	4.0	7.0	6.0	250.0	...	0.0	0.0	0.0	7.0	8.0	4.0	5.2	0.0	0.0	0.0
6	1	42.0	8.53	5.0	1.0	3.0	4.0	6.0	7.0	200.0	...	0.0	1.0	1.0	7.0	7.0	3.0	8.7	0.0	0.0	0.0
7	2	52.0	8.30	4.0	1.0	3.0	5.0	7.0	6.0	100.0	...	1.0	0.0	1.0	6.0	8.0	4.0	14.8	0.0	0.0	0.0
8	3	46.0	9.08	6.0	1.0	4.0	6.0	7.0	7.0	250.0	...	1.0	0.0	2.0	3.0	6.0	7.0	15.3	0.0	0.0	0.0
9	4	49.0	8.48	4.0	2.0	4.0	6.0	6.0	6.0	250.0	...	0.0	1.0	1.0	9.0	7.0	3.0	9.6	0.0	0.0	0.0
12	0	43.0	7.73	3.0	1.0	2.0	5.0	7.0	7.0	200.0	...	0.0	0.0	0.0	9.0	8.0	5.0	9.9	2.0	0.0	0.0

10 rows × 23 columns

In [8]:
```
y[0:10]
```

Out[8]:
```
1     81.0
2     82.0
3     76.0
4     77.0
5     85.0
6     52.0
7     83.0
8     60.0
9     73.0
12    51.0
Name: Whoop Recovery, dtype: float64
```

In [9]:
```
print(f'Total Training Size: {X_train.shape[0]}')
print(f'Total Testing Size: {X_test.shape[0]}')
print(f'Total Observations: {y.shape[0]}')
```

```
Total Training Size: 12
Total Testing Size: 5
Total Observations: 17
```

Random Forest Regression

Reasoning and Methodology

Random Forest Regression is a robust machine learning algorithm which uses random sampling and bootstrapping from decision trees to make continuous predictions on target variables. I chose this as the first model to be implemented because of its capability to accomodate both continuous and discrete variable. Since the decision trees themselves also support this input flexibility, Random Forest Regression is an excellent candidate for this data.

Implementation

In [10]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
```

In [11]:
```
rfr = RandomForestRegressor()

rfr.fit(X_train, y_train)

score = rfr.score(X_train, y_train)
print("R-squared:", round(score, 2))
```

```
R-squared: 0.92
```

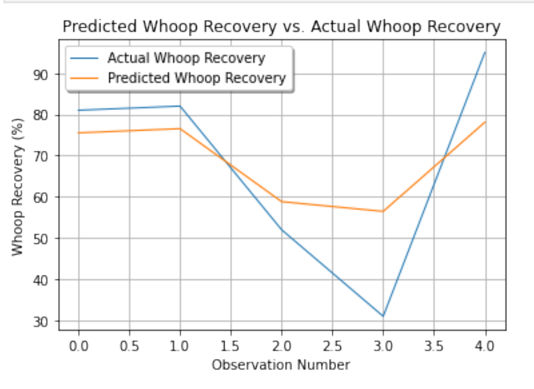
In [12]:
```
y_pred = rfr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print("MSE: ", round(mse, 2))
print("RMSE: ", round(mse**(1/2.0), 2))
```

```
MSE:  208.21
RMSE:  14.43
```

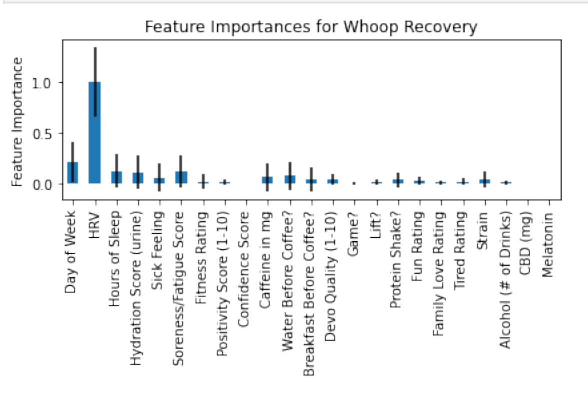
In [13]:
```
importances = rfr.feature_importances_
x_column_headers = X_test.columns
std = np.std([tree.feature_importances_ for tree in rfr.estimators_], axis=0)
normalized_importances = (importances - importances.min()) / (importances.max() - importances.min())
```

In [14]:
```
x_ax = range(len(y_test))
plt.plot(x_ax, y_test, linewidth=1, label=f"Actual {y.name}")
plt.plot(x_ax, y_pred, linewidth=1.1, label=f"Predicted {y.name}")
plt.title(f"Predicted {y.name} vs. Actual {y.name}")
plt.xlabel('Observation Number')
units = {'Whoop Recovery': '%', 'HRV': 'bpm', 'Current BA': '%', 'Body Weight': 'lbs'}
unit = units[target]
plt.ylabel(f'{y.name} ({unit})')
plt.legend(loc='best',fancybox=True, shadow=True)
plt.grid(True)
plt.show()
```



In [15]:
```
forest_importances = pd.Series(normalized_importances, index=x_column_headers)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title(f"Feature Importances for {target}")
ax.set_ylabel("Feature Importance")
fig.tight_layout()
```



Analysis and Impact

Linear Regression

Reasoning and Methodology

Though Random Forest Regression seems to be a reasonable and scalable fit for the data, it is worth implementing Linear Regression because of it's simplicity. While it is certainly simple to implement, it is also more easily understandable to peopel with less experience with data and statistics. Linear regression also allows flexibility in inputting both continous and discrete variables by normalizing discrete variables to become continuous. This model also outputs clear regression coefficients, which are extremely readable and applicable to this project.

Implementation

In [16]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

In [17]:
```
regr = LinearRegression()
regr.fit(X_train, y_train)

y_pred = regr.predict(X_test)
r2_score2 = r2_score(y_pred, y_test)
print(f'R-squared: {round(r2_score2, 2)}')
```

```
R-squared: 0.91
```

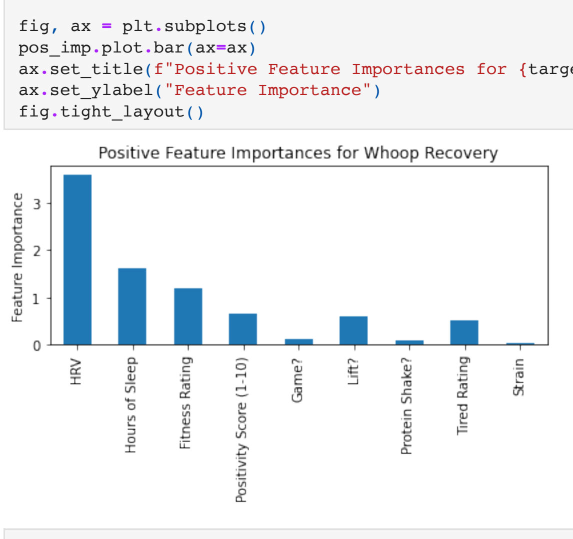
In [18]:
```
feature_titles = X_train.columns
reg_coef = regr.coef_
```

In [19]:
```
pos_features_with_coef = {}
neg_features_with_coef = {}
for i in range(len(feature_titles)):
    if reg_coef[i] > 0:
        pos_features_with_coef[feature_titles[i]] = round(reg_coef[i], 2)
    elif reg_coef[i] < 0:
        neg_features_with_coef[feature_titles[i]] = round(-1*reg_coef[i], 2)
    print(f'{feature_titles[i]}: {round(reg_coef[i], 2)}')
```

```
Day of Week: -0.64
HRV: 3.59
Hours of Sleep: 1.62
Hydration Score (urine): -0.9
Sick Feeling: -0.59
Soreness/Fatigue Score: -2.2
Fitness Rating: 1.18
Positivity Score (1-10): 0.66
Confidence Score: -0.5
Caffeine in mg: -0.02
Water Before Coffee?: -0.83
Breakfast Before Coffee?: -0.83
Devo Quality (1-10): -0.99
Game?: 0.11
Lift?: 0.61
Protein Shake?: 0.08
Fun Rating: -0.12
Family Love Rating: -0.63
Tired Rating: 0.51
Strain: 0.03
Alcohol (# of Drinks): -1.22
CBD (mg): 0.0
Melatonin: 0.0
```

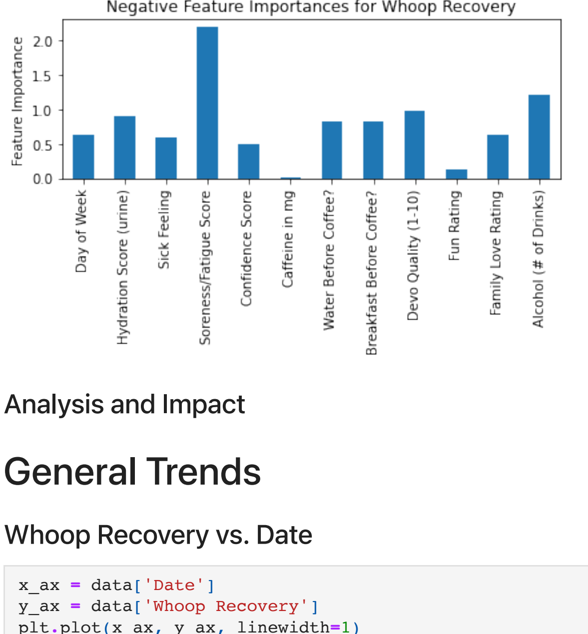
In [20]:
```
pos_imp = pd.Series(pos_features_with_coef.values(), index=pos_features_with_coef.keys())

fig, ax = plt.subplots()
pos_imp.plot.bar(ax=ax)
ax.set_title(f"Positive Feature Importances for {target}")
ax.set_ylabel("Feature Importance")
fig.tight_layout()
```



In [21]:
```
neg_imp = pd.Series(neg_features_with_coef.values(), index=neg_features_with_coef.keys())

fig, ax = plt.subplots()
neg_imp.plot.bar(ax=ax)
ax.set_title(f"Negative Feature Importances for {target}")
ax.set_ylabel("Feature Importance")
fig.tight_layout()
```

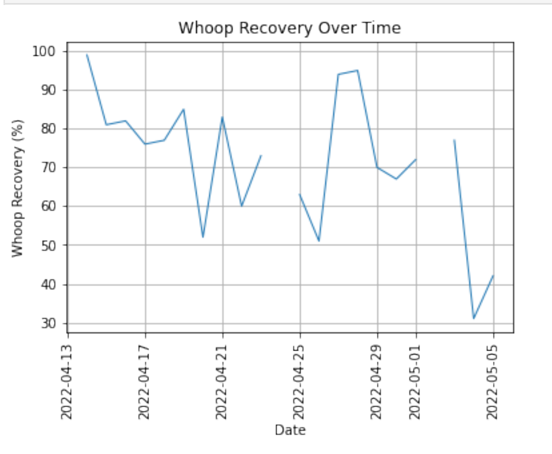


Analysis and Impact

General Trends

Whoop Recovery vs. Date

In [22]:
```
x_ax = data['Date']
y_ax = data['Whoop Recovery']
plt.plot(x_ax, y_ax, linewidth=1)
plt.title(f"Whoop Recovery Over Time")
plt.xlabel('Date')
unit = '%'
plt.ylabel(f'Whoop Recovery ({unit})')
plt.grid(True)
plt.xticks(rotation=90)
plt.show()
```



In [ ]:

In [ ]:

In [ ]: