import pandas as pd  
##%matplotlib inline  
import matplotlib  
import numpy as np  
import matplotlib.pyplot as plt  
from plotly import graph\_objects  
import streamlit as st

## My Info

st.title('Computer Science Capstone — C964')  
st.header('Student – Mark Nefzger')  
st.header('Student ID: 001411596')

2023-10-06 15:04:13.438   
 Warning: to view this Streamlit app on a browser, run it with the following  
 command:  
  
 streamlit run [ARGUMENTS]

DeltaGenerator()

## Version Info

pd\_ver = pd.\_\_version\_\_  
st.write("Pandas version: ", pd\_ver)

np\_ver = np.\_\_version\_\_  
st.write("Numpy version: ", np\_ver)

st\_ver = st.\_\_version\_\_  
st.write("Streamlit version: ", st\_ver)

plt\_ver = matplotlib.\_\_version\_\_  
st.write("Matplotlib version: ", plt\_ver)

## Import data

# Import Data  
health\_data = pd.read\_csv("New Data/oura\_2019-01-01\_2023-09-09\_trends\_Original.csv")

## View data

st.header('Imported Data')

DeltaGenerator()

health\_data

date Sleep Score ... HRV Balance Score Recovery Index Score  
0 2021-01-13 83.0 ... NaN 99.0  
1 2021-01-14 76.0 ... NaN 69.0  
2 2021-01-15 78.0 ... NaN 96.0  
3 2021-01-16 69.0 ... NaN 51.0  
4 2021-01-17 78.0 ... NaN 50.0  
.. ... ... ... ... ...  
965 2023-07-22 NaN ... NaN NaN  
966 2023-07-29 NaN ... NaN NaN  
967 2023-07-31 NaN ... NaN NaN  
968 2023-08-07 NaN ... NaN NaN  
969 2023-09-05 NaN ... NaN NaN  
  
[970 rows x 54 columns]

## Describe Data

# health\_data.dtypes

## Set up dataframe

df = pd.DataFrame(health\_data)

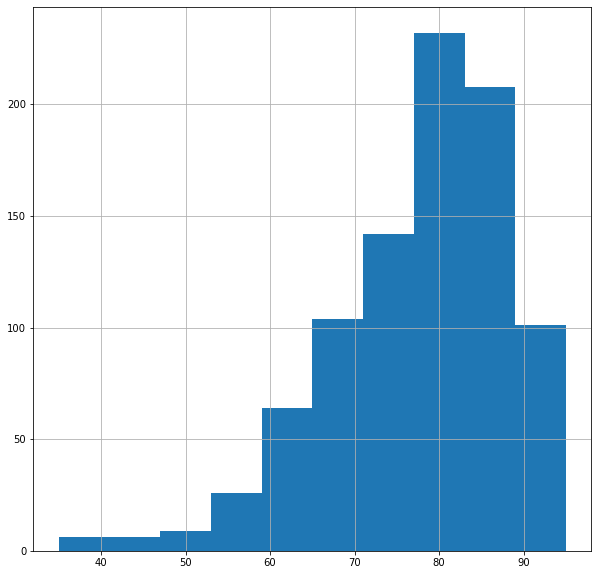
## Convert Sleep Duration and Rest Time to hours

df["Total Sleep Duration"] = df["Total Sleep Duration"] / 3600  
df["Rest Time"] = df["Rest Time"] / 3600

pd.crosstab(df["Total Sleep Duration"] > 7, df["Readiness Score"] >85)

Readiness Score False True   
Total Sleep Duration   
False 493 47  
True 279 151

plt.show(df["Readiness Score"].hist(figsize=(10, 10)))



## Manipulating Data

df.dropna(inplace=True)

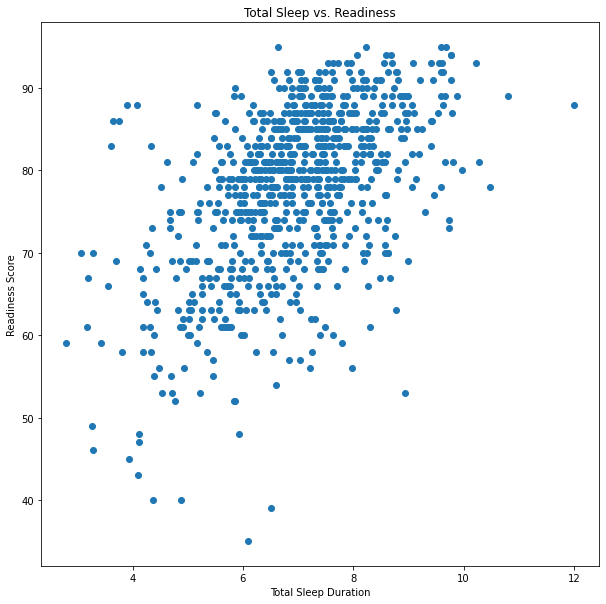
# Randomize data 1 = 100%  
df.sample(frac=1)

date Sleep Score ... HRV Balance Score Recovery Index Score  
178 2021-07-16 57.0 ... 90.0 44.0  
667 2022-12-25 94.0 ... 87.0 49.0  
624 2022-11-07 83.0 ... 80.0 100.0  
310 2021-12-05 71.0 ... 54.0 62.0  
301 2021-11-26 90.0 ... 76.0 78.0  
.. ... ... ... ... ...  
586 2022-09-19 79.0 ... 46.0 95.0  
469 2022-05-15 89.0 ... 48.0 100.0  
520 2022-07-08 93.0 ... 75.0 40.0  
859 2023-08-02 89.0 ... 41.0 68.0  
328 2021-12-23 77.0 ... 37.0 100.0  
  
[820 rows x 54 columns]

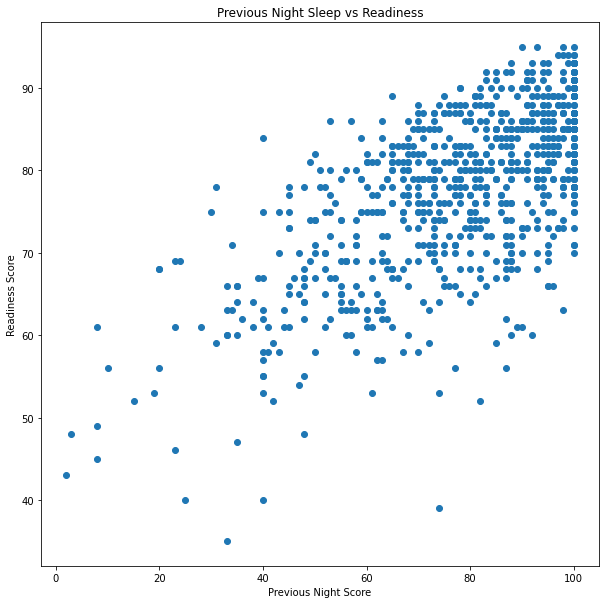
# Reset index if necessary  
# df.reset\_index(drop=True, inplace=True)

## Matplotlib

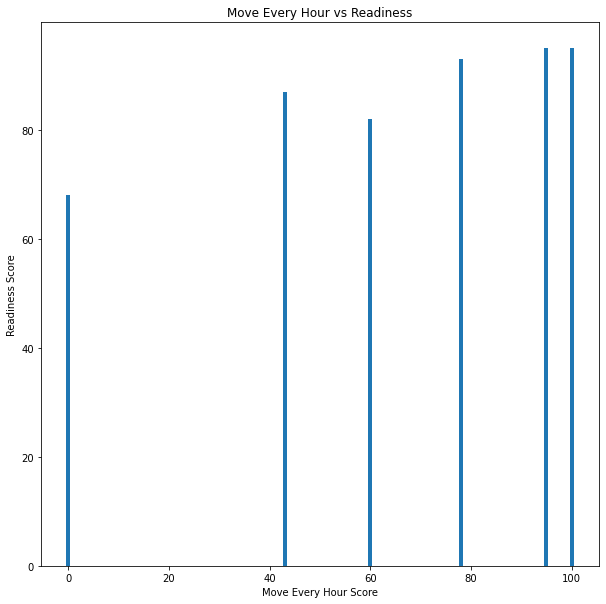
# 1. Prepare data  
x = df["Total Sleep Duration"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Total Sleep vs. Readiness",   
 xlabel="Total Sleep Duration",  
 ylabel="Readiness Score")  
  
# 5. Save and show  
fig.savefig("Figures/Figure\_1.png")



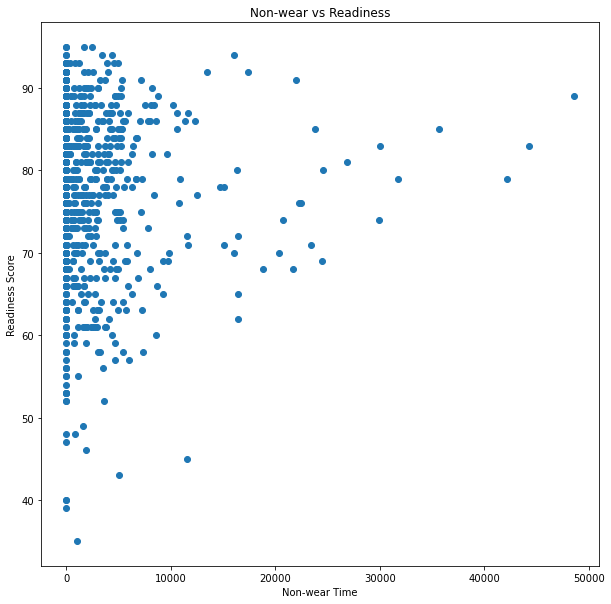
# 1. Prepare data  
x = df["Previous Night Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
  
# 4. Customize plot  
ax.set(title="Previous Night Sleep vs Readiness",   
 xlabel="Previous Night Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_2.png")



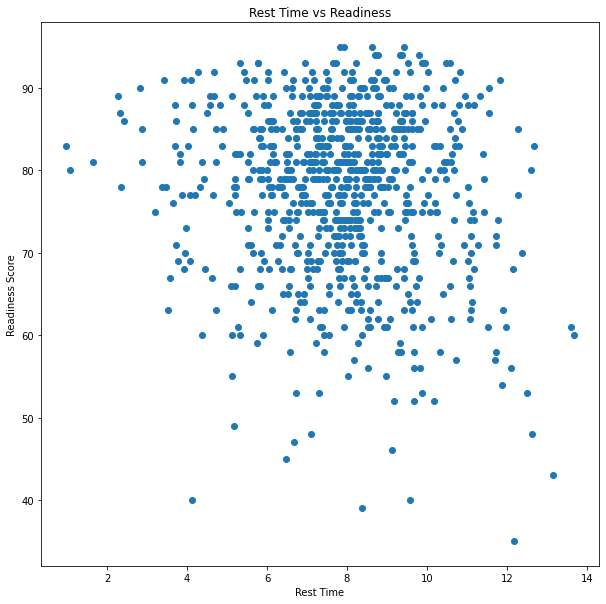
# 1. Prepare data  
x = df["Move Every Hour Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.bar(x,y)  
  
# 4. Customize plot  
ax.set(title="Move Every Hour vs Readiness",   
 xlabel="Move Every Hour Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_3.png")



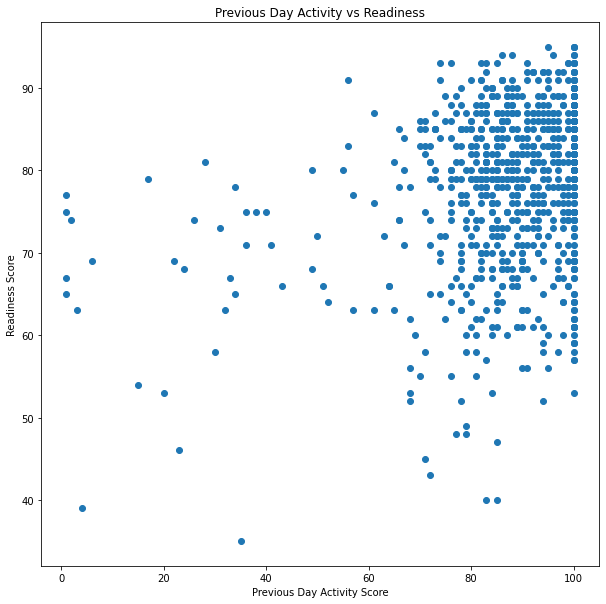
# 1. Prepare data  
x = df["Non-wear Time"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Non-wear vs Readiness",   
 xlabel="Non-wear Time",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_4.png")



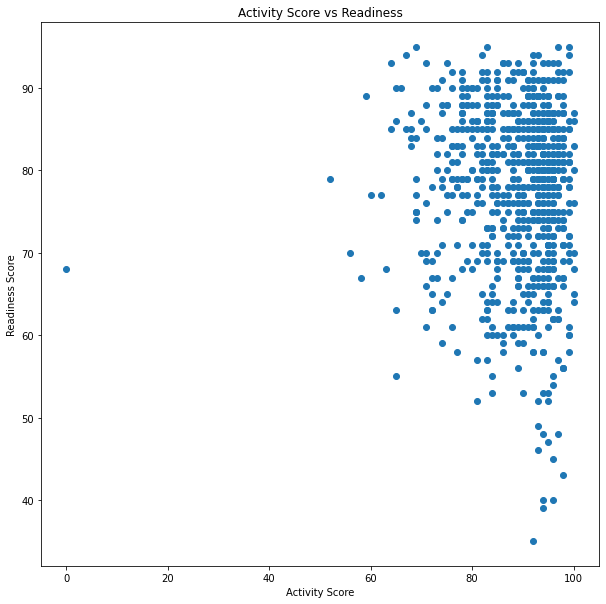
# 1. Prepare data  
x = df["Rest Time"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Rest Time vs Readiness",   
 xlabel="Rest Time",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_5.png")



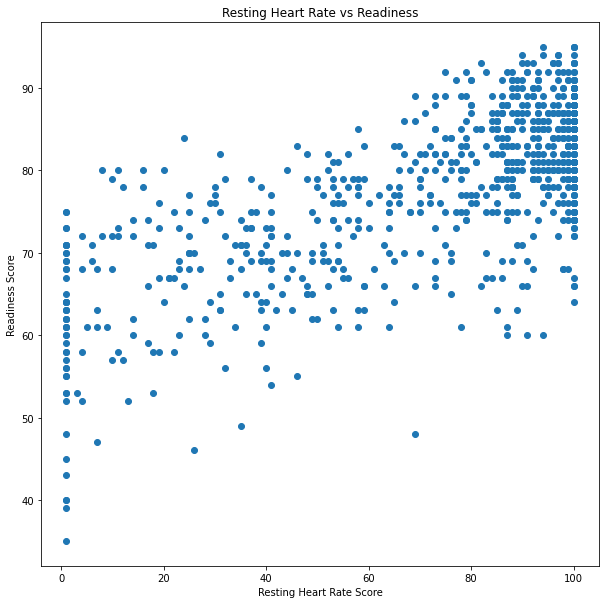
# 1. Prepare data  
x = df["Previous Day Activity Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Previous Day Activity vs Readiness",   
 xlabel="Previous Day Activity Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_6.png")



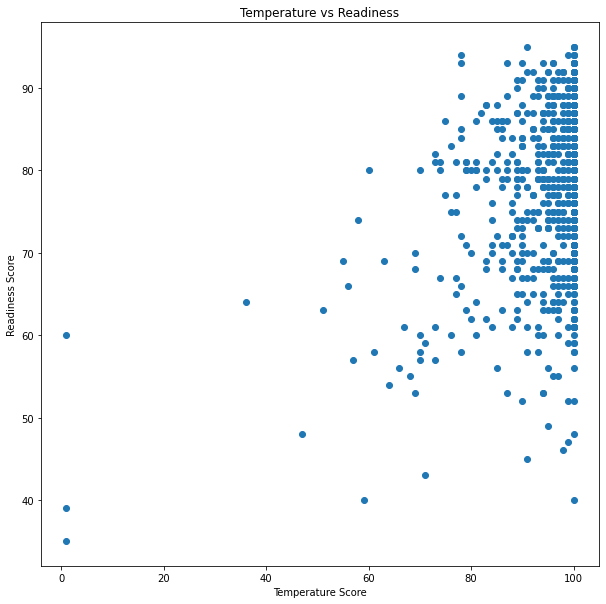
# 1. Prepare data  
x = df["Activity Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Activity Score vs Readiness",   
 xlabel="Activity Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_7.png")



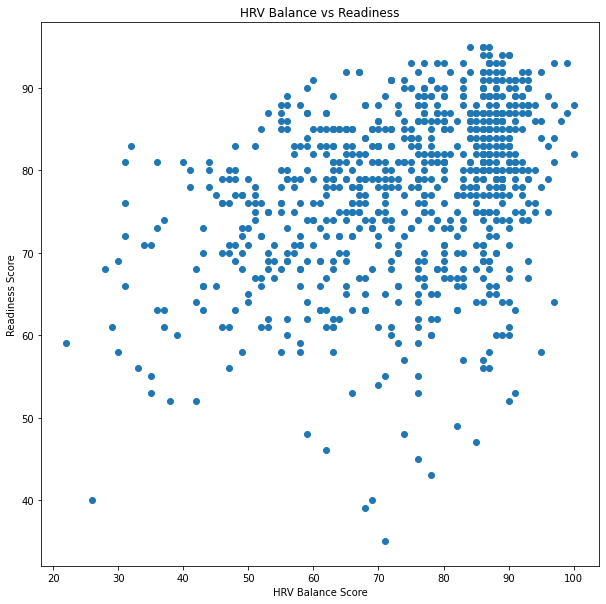
# 1. Prepare data  
x = df["Resting Heart Rate Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Resting Heart Rate vs Readiness",   
 xlabel="Resting Heart Rate Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_8.png")



# 1. Prepare data  
x = df["Temperature Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="Temperature vs Readiness",   
 xlabel="Temperature Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_9.png")



# 1. Prepare data  
x = df["HRV Balance Score"]  
y = df["Readiness Score"]  
  
# 2. Setup plot  
fig, ax = plt.subplots(figsize=(10, 10))  
  
# 3. Plot data  
ax.scatter(x,y)  
  
# 4. Customize plot  
ax.set(title="HRV Balance vs Readiness",   
 xlabel="HRV Balance Score",  
 ylabel="Readiness Score")  
  
# 5. Save and show   
fig.savefig("Figures/Figure\_10.png")



st.header('Relevent Data')

DeltaGenerator()

st.image('Figures/Figure\_1.png')  
st.image('Figures/Figure\_7.png')  
st.image('Figures/Figure\_8.png')  
st.image('Figures/Figure\_9.png')  
st.image('Figures/Figure\_10.png')

DeltaGenerator()

## Remove data columns that are lagging data fields or not necessary

df.drop(df.columns[[0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,46,47,48,49,53]], axis=1, inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>  
Index: 820 entries, 7 to 895  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Total Sleep Duration 820 non-null float64  
 1 Activity Score 820 non-null int64   
 2 Readiness Score 820 non-null float64  
 3 Temperature Score 820 non-null float64  
 4 Resting Heart Rate Score 820 non-null float64  
 5 HRV Balance Score 820 non-null float64  
dtypes: float64(5), int64(1)  
memory usage: 44.8 KB

## Describe Relevent Data

st.header('Describe Relevent Data')  
st.write(df.describe())  
df.describe()

Total Sleep Duration ... HRV Balance Score  
count 820.000000 ... 820.000000  
mean 6.934004 ... 74.113415  
std 1.324972 ... 15.302503  
min 2.800000 ... 22.000000  
25% 6.162500 ... 64.000000  
50% 6.966667 ... 77.000000  
75% 7.741667 ... 87.000000  
max 11.991667 ... 100.000000  
  
[8 rows x 6 columns]

## Algorithm/Estimator

# Import algorithm/estimator  
  
# Instantiate and fit the model (on the training set)  
# Try RandomForest estimator  
  
  
from sklearn.ensemble import RandomForestClassifier  
model = RandomForestClassifier()  
  
# Setup random seed  
np.random.seed(42)  
  
# Create the data  
df.dropna(inplace=True)  
X = df.drop("Readiness Score", axis=1)  
y = df["Readiness Score"] #target  
  
# Split into train and test sets  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

model.get\_params()

{'bootstrap': True,  
 'ccp\_alpha': 0.0,  
 'class\_weight': None,  
 'criterion': 'gini',  
 'max\_depth': None,  
 'max\_features': 'sqrt',  
 'max\_leaf\_nodes': None,  
 'max\_samples': None,  
 'min\_impurity\_decrease': 0.0,  
 'min\_samples\_leaf': 1,  
 'min\_samples\_split': 2,  
 'min\_weight\_fraction\_leaf': 0.0,  
 'n\_estimators': 100,  
 'n\_jobs': None,  
 'oob\_score': False,  
 'random\_state': None,  
 'verbose': 0,  
 'warm\_start': False}

model.fit(X\_train, y\_train);

y\_preds = model.predict(X\_test)

model.score(X\_test, y\_test)

0.06707317073170732

# Try Ridge Regression  
  
from sklearn.linear\_model import Ridge  
  
model = Ridge()  
model.fit(X\_train, y\_train)  
  
# Check the score of the model (on the test set)  
model.score(X\_test, y\_test)

0.7083864563369104

model.get\_params()

{'alpha': 1.0,  
 'copy\_X': True,  
 'fit\_intercept': True,  
 'max\_iter': None,  
 'positive': False,  
 'random\_state': None,  
 'solver': 'auto',  
 'tol': 0.0001}

from sklearn import linear\_model  
model = linear\_model.LassoLars(alpha=1.0)  
model.fit(X\_train, y\_train)  
  
# Check the score of the model (on the test set)  
model.score(X\_test, y\_test)

0.722420736903518

## Make Predictions Using Machine Language Model

## Pick Typical Day (9/17/2023)

test\_data = pd.read\_csv("New Data/oura\_2023-09-17\_2023-09-17\_trends.csv")

test\_data.drop(test\_data.columns[[0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,46,47,48,49,53]], axis=1, inplace=True)

## Remove Readiness Score  
test\_data.drop(test\_data.columns[2], axis=1, inplace=True)  
test\_data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1 entries, 0 to 0  
Data columns (total 5 columns):  
 # Column Non-Null Count Dtype  
--- ------ -------------- -----  
 0 Total Sleep Duration 1 non-null int64  
 1 Activity Score 1 non-null int64  
 2 Temperature Score 1 non-null int64  
 3 Resting Heart Rate Score 1 non-null int64  
 4 HRV Balance Score 1 non-null int64  
dtypes: int64(5)  
memory usage: 172.0 bytes

## Convert Total Sleep Duration to hours  
test\_data["Total Sleep Duration"] = test\_data["Total Sleep Duration"] / 3600

st.header('Data for a Typical Day (9/17/2023)')  
st.write(test\_data)

value = st.slider(  
 'Select a estimated sleep',  
 4.0, 12.0, 8.0)  
st.write('Estimated Sleep:', value)  
test\_data["Total Sleep Duration"] = value

## Prediction:

st.header('Based on your estimated sleep, your readiness score for 9/18/23 is prediced to be: ')  
Prediction = str(model.predict(test\_data))  
font\_size = 50  
  
html\_str = f"""  
<style>  
p.a {{  
 font: bold {font\_size}px Courier;  
}}  
</style>  
<p class="a">{Prediction}</p>  
"""  
  
st.markdown(html\_str, unsafe\_allow\_html=True)  
#value=model.predict(test\_data)

DeltaGenerator()

!jupyter nbconvert --to script Capstone.ipynb  
with open('Capstone.py', 'r') as f:  
 lines = f.readlines()  
with open('Capstone.py', 'w') as f:  
 for line in lines:  
 if 'nbconvert --to script' in line:  
 break  
 else:  
 f.write(line)

[NbConvertApp] Converting notebook Capstone.ipynb to script  
[NbConvertApp] Writing 8655 bytes to Capstone.py