Handover Document

Project: Oxygen Consumption Prediction Project
Creator: Zhiyong Wu
Version: V3
Creation Date: 18th Dec 2022

Table of Contents

1 Introduction

In this report, I establish an oxygen consumption prediction system using Python. The purpose of the project is to monitor the oxygen consumption of players using our VR game platform. The prediction can be used to monitor the health status of players and help them to improve their cycling abilities both in games and in real-world scenarios.

1.1 Experimental environments

To simplify the model establishment and maintenance process, we use Jupyter Notebook as the IDE, which is Python-based script programming environment. It can show the results step by step.
To allow forthcoming programmers to follow easily, we use one of the most basic and easy-to-handle data structure, *numpy array*, to store values, while using *matplotlib.pyplot* to plot the diagrams including line charts, bar plots, scatter plots, etc. We use a popular machine learning package called *sklearn* to deploy the machine learning models.

```
In [3]: # packages

        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split


        from sklearn.metrics import explained_variance_score
        from sklearn.metrics import median_absolute_error
        from sklearn.metrics import mean_absolute_error
        from sklearn.metrics import mean_squared_error
        from sklearn.metrics import r2_score

        # do not display in scientific notation
        np.set_printoptions(suppress=True)
```

Fig. 1: Packages used for experiments

1.2 Datasets

We use two datasets in this project. One is a real-world dataset called *Kaggle* while the other one is a collected dataset called *data_3* in this project.

- Kaggle dataset (in .csv format): 70% for training and 30% for testing
- data_3 dataset (in .xls format): 100% for prediction

The examples of raw datasets are screenshotted as follows.

	A	B	C	D	E	F	G	H
1	time	Power	Oxygen	Cadence	HR	RF	Participant	Method
2	1	0	318.4	0	75.6	20.1	1	0
3	2	0	356.1667	0	75.66667	19.75	1	0
4	3	0	403.2857	0	75.71429	19.42857	1	0
5	4	0	456.25	0	75.75	19.125	1	0
6	5	0	478.9259	0	75.74074	18.96296	1	0
7	6	0	480.4	0	75.7	18.9	1	0
8	7	0	480	0	75.7	18.6	1	0
9	8	0	482.7167	0	75.69167	18.375	1	0
10	9	0	488.55	0	75.675	18.225	1	0

Fig. 2: Kaggle Dataset

	A	B	C	D	E	F	G
1	step	enhanced_speed	cadence	grade	enhanced_altitude	max_heart_rate	power
2	0	26.1936	66	0	9.6	0.7162	2.075
3	1	24.858	66	0	9.6	0.7162	2.0125
4	2	25.6644	66	0	9.6	0.7162	1.9125
5	3	25.7004	66	0	9.6	0.7162	1.8375
6	4	25.7868	66	0	9.6	0.7215	1.825
7	5	25.6896	66	0	9.8	0.7215	1.8
8	6	25.7076	66	0	9.8	0.7268	1.6875
9	7	25.5744	66	0	9.8	0.7268	1.5625
10	8	24.9048	66	0	9.8	0.7268	1.525

Fig. 3 data_3 Dataset

1.3 Attributes and Target

The target is the Oxygen consumption as described above. Based on the team discussion, the chosen attributes are power, cadence, and heart rate.

- Power: Attribute
- Cadence: Attribute
- Heart rate: Attribute
- Oxygen: Target

2 Kaggle Dataset Visualization

In this section, we look into the distributions and correlations of the three attributes and the target. This will help following students or engineers to better understand the project.

2.1 Distributions of Attributes and Target

We show screenshots of the code first and then the corresponding histograms.

Apparently, the distributions vary a lot.

- Power: Highly left skewed
- Cadence: Highly centralized but with a considerable number of records on the left
- Heart rate: Almost normalized
- Oxygen: A bit left skewed

```
In [5]: # 1-2 visualize data -- histograms


fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(7,7))
fig.suptitle('Histogram for Each Attribute')

axes[0][0].hist(kaggle_selected[:,[0]])
axes[0][0].set_title("Power")

axes[0][1].hist(kaggle_selected[:,[1]])
axes[0][1].set_title("Oxygen")

axes[1][0].hist(kaggle_selected[:,[2]])
axes[1][0].set_title("Cadence")

axes[1][1].hist(kaggle_selected[:,[3]])
axes[1][1].set_title("Heart Rate")

plt.show()
plt.close()
```

Fig. 4: Code to plot histograms

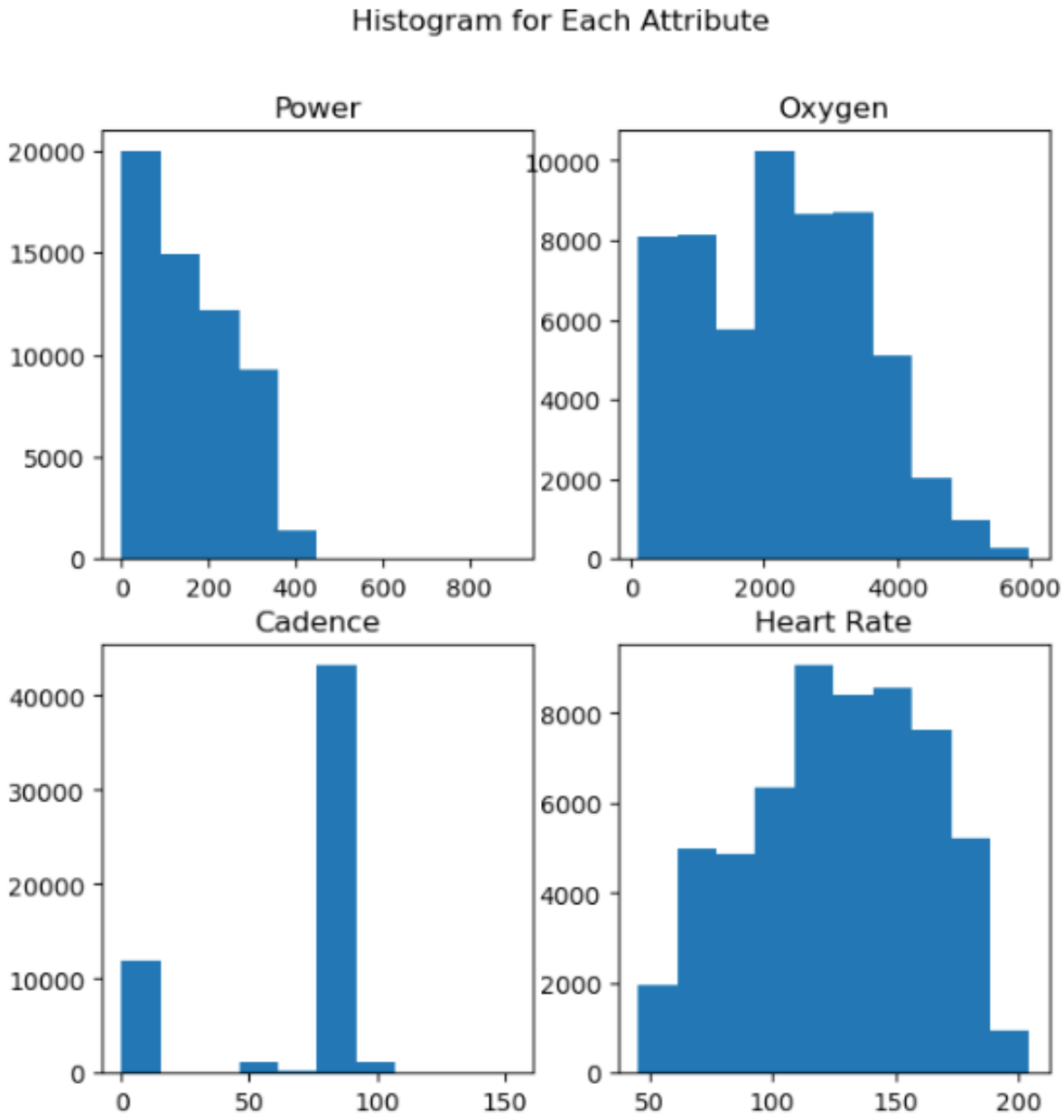

Fig. 5: Histograms to show the distributions

2.2 Line Charts

The coder and line charts to show the trends of each attribute and target are as follows.

```
In [6]: # 1-3 visualize data -- line charts


        # define x
        x = np.arange(0,len(kaggle_selected[:,[0]]))


        fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(7,7))
        fig.suptitle('Line Charts for Each Attribute')



        axes[0][0].plot(x,kaggle_selected[:,[0]])
        axes[0][0].set_title("Power")

        axes[0][1].plot(x,kaggle_selected[:,[1]])
        axes[0][1].set_title("Oxygen")

        axes[1][0].plot(x,kaggle_selected[:,[2]])
        axes[1][0].set_title("Cadence")

        axes[1][1].plot(x,kaggle_selected[:,[3]])
        axes[1][1].set_title("Heart Rate")

        plt.show()
        plt.close()
```

Fig. 6: Code for line charts

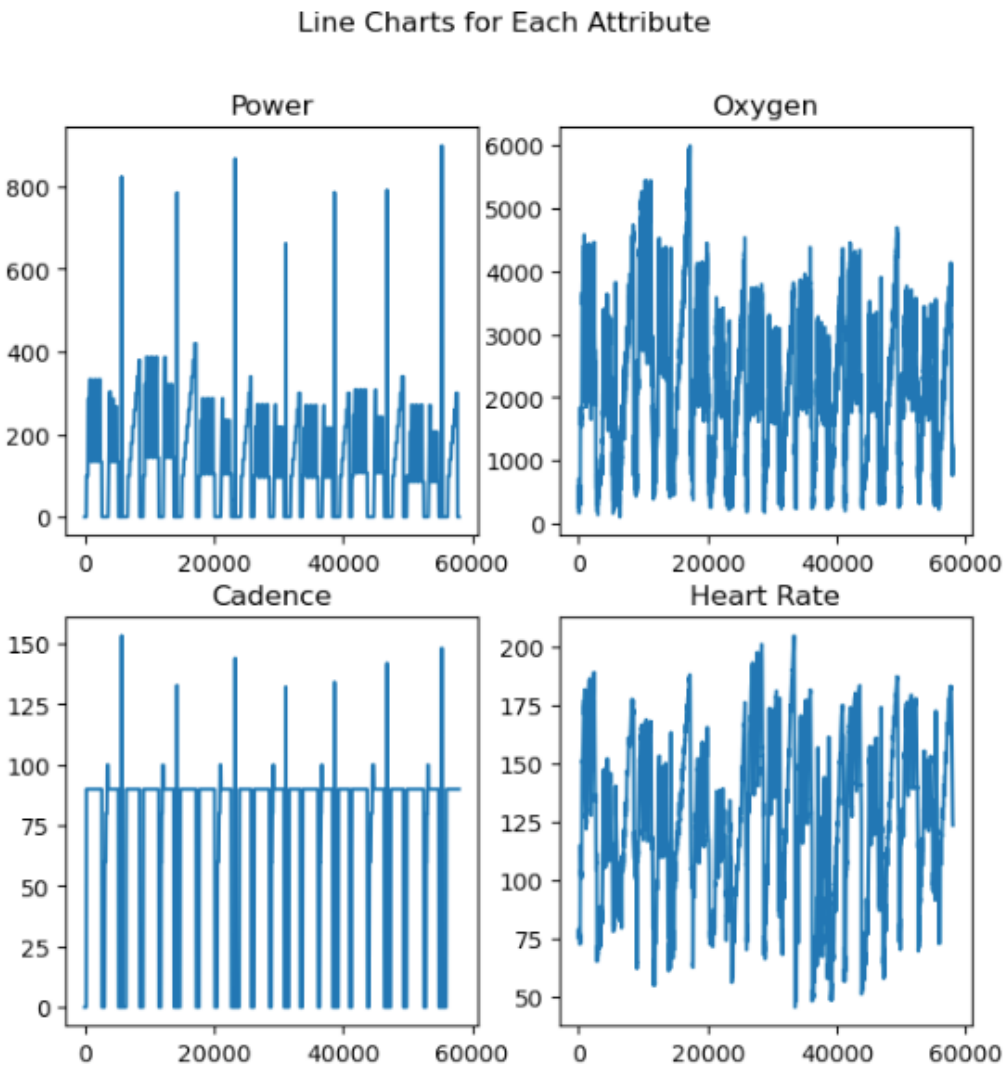

Fig. 7 Line charts for each attribute and target

The line charts show a fluctuation of each attribute. But for all attributes and the target, we have an observation that there are steady periods between all fluctuations. Based on our experience, this will help with the performance improvement for the machine learning models.

2.3 Correlations via scatter plots

We further investigate the correlations between all attributes and Oxygen. This will help with the analysis of the final machine learning model for prediction. The code and plots are as Fig. 8 and Fig. 9.

```
In [7]: # 1-4 visualize data -- scatter charts - correlations


        fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(7,7))
        fig.suptitle('Scatter Plots of Each Attribute v.s. Oxygen')


        axes[0][0].scatter(kaggle_selected[:,[0]],kaggle_selected[:,[1]])
        axes[0][0].set_title("Power v.s. Oxygen")

        axes[0][1].scatter(kaggle_selected[:,[1]],kaggle_selected[:,[1]])
        axes[0][1].set_title("Oxygen v.s. Oxygen")

        axes[1][0].scatter(kaggle_selected[:,[2]],kaggle_selected[:,[1]])
        axes[1][0].set_title("Cadence v.s. Oxygen")

        axes[1][1].scatter(kaggle_selected[:,[3]],kaggle_selected[:,[1]])
        axes[1][1].set_title("Heart Rate v.s. Oxygen")

        plt.show()
        plt.close()
```

Fig. 8: Code for scatter plots

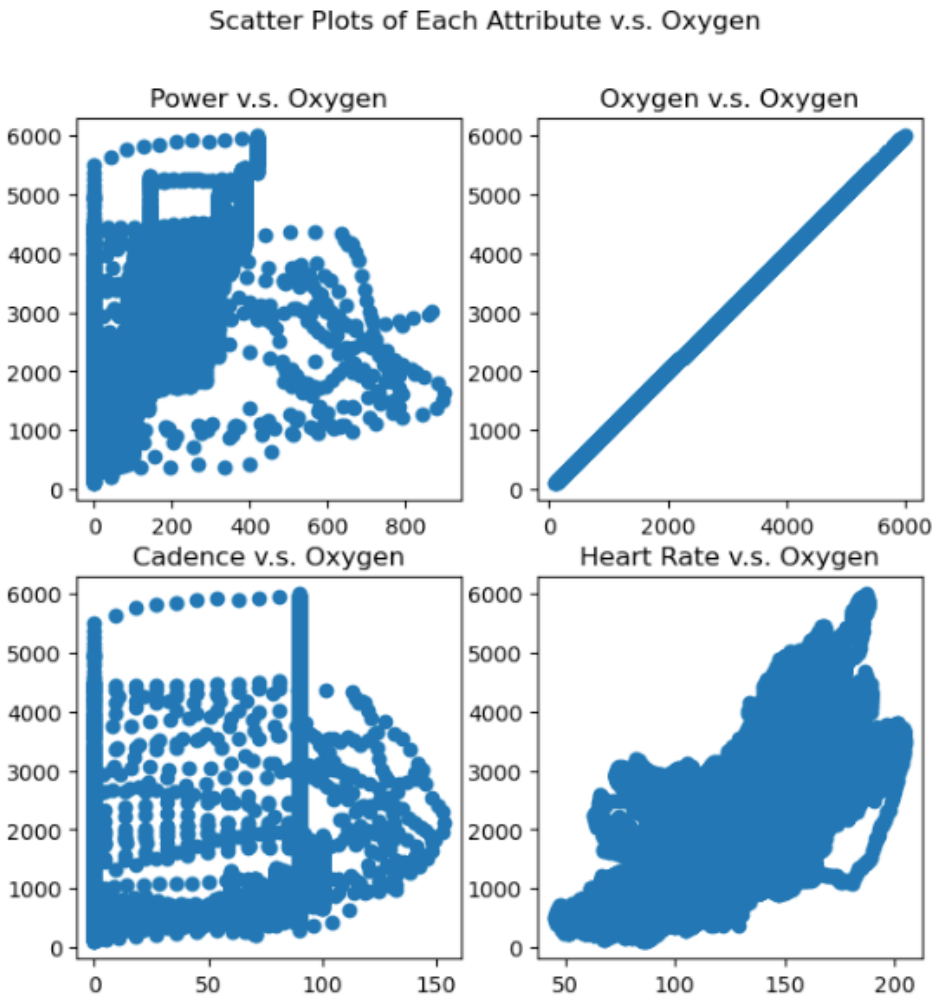

Fig. 9: Scatter plots

From Fig. 9, we can tell that there are unique correlations between attributes and the target Oxygen.

- Power v.s. Oxygen:
- Cadence v.s. Oxygen:
- Heart Rate v.s. Oxygen:

3 Deploy 10 Regression Machine Learning Models

In this section, we show the code and results of 10 machine learning models. Since the data is not for classification but for regression, we choose 10 most popular regression machine learning models.

- Linear Regression
- Support Vector Machine
- Ridge Regression
- Least absolute shrinkage and selection operator (Lasso)
- Multi-Layer Perceptron (MLP)
- Decision Tree
- Extra Tree

- Random Forest
- AdaBoost
- Gradient Boosting

Among all the machine learning models, there are basic ones like linear regression, tree-based ones like decision tree, neural network based ones like MLP. We deploy all of them and derive results in this section. The evolution of the machine learning models will be provided in the next section.

We first split the Kaggle dataset into training (70%) and testing (30%) subsets.

```
In [8]: # 1-5 pre-processing data for machine learning

        # power, cadence, heart rate as the attributes and oxygen as the target
        x = kaggle_selected[:,[0,2,3]]
        y = kaggle_selected[:,[1]]

        # split the dataset for training and testing (training 70%, testing 20%)
        x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3)


        # print the length of the whole set, training set, and test set
        print(len(x),len(y))
        print(len(x_train),len(x_test),len(y_train),len(y_test))


        57983 57983
        40588 17395 40588 17395
```

Fig. 10: Dataset preparation

- Total length of dataset: 57983
- Total length of training dataset: 40588
- Total length of dataset: 17359

Besides, we use five matrixes to measure the performance of the machine learning models.

- explained variance score
- R2 score
- mean absolute error
- mean square error
- median absolute error

For the first two matrixes, explained variance score and R2 score, the higher the better while the best is 1. For the last three matrixes, mean absolute error, mean square error, and median absolute error, the lower the better while 0 is the best.

3.1 Linear Regression

Linear regression is the most basic model we use as the baseline model. We did not expect very good performance since based on the correlation diagrams, we can see there aren't any linear correlations between attributes and the target.

```
In [11]: # 1-6 establishing a machine learning model  --- the most basic one --- linear regression

         from sklearn.model_selection import cross_val_score
         from sklearn.linear_model import LinearRegression as LR

         reg = LR().fit(x_train, y_train)
         y_predict = reg.predict(x_test)


         ## Evaluation Matrix:
         print("")
         print("The evalution of Linear Regression is as follows.")

         # Explained variance score (1 for the best)
         print("Explained variance score: ",explained_variance_score(y_test, y_predict))

         # R2 score (1 for the best)
         print("R2 score: ", r2_score(y_test, y_predict))

         # Mean absolute error
         print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

         # Mean squared error
         print("Mean squared error: ",mean_squared_error(y_test, y_predict))


         # Median absolute error (the smaller, the better)
         print("Median absolute error: ", median_absolute_error(y_test, y_predict))


         # store the evaluation matrix values


         scores[0][0] = explained_variance_score(y_test, y_predict)
         scores[0][1] = r2_score(y_test, y_predict)
         scores[0][2] = mean_absolute_error(y_test, y_predict)
         scores[0][3] = mean_squared_error(y_test, y_predict)
         scores[0][4] = median_absolute_error(y_test, y_predict)


         The evalution of Linear Regression is as follows.
         Explained variance score:  0.8790847891776439
         R2 score:  0.879050638108291
         Mean absolute error:  298.6938342854376
         Mean squared error:  188329.60356875215
         Median absolute error:  218.0147861118262
```

Fig. 11: Code for Linear Regression

The evaluation of Linear Regression is as follows.

- Explained variance score: 0.8817123778999642
- R2 score: 0.8817122677808992
- Mean absolute error: 295.51529321350614
- Mean squared error: 183373.1092282643
- Median absolute error: 215.66437414520942

3.2 Support Vector Machine

The deployment and results of Support Vector Machine are as follows.

In [57]:
```
# 1-6 establishing a machine learning model  -- support vector machine

from sklearn.svm import SVR

clf = SVR()
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of Support Vector Machine is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))


# store the evaluation matrix values

scores[1][0] = explained_variance_score(y_test, y_predict)
scores[1][1] = r2_score(y_test, y_predict)
scores[1][2] = mean_absolute_error(y_test, y_predict)
scores[1][3] = mean_squared_error(y_test, y_predict)
scores[1][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of Support Vector Machine is as follows.
Explained variance score:  0.8889528969200657
R2 score:  0.886518359441424
Mean absolute error:  265.5864900836913
Mean squared error:  175922.56508059194
Median absolute error:  169.63183767642613
```

Fig. 12: Code for SVM

The evaluation of Support Vector Machine is as follows.

- Explained variance score: 0.8889528969200657
- R2 score: 0.886518359441424
- Mean absolute error: 265.5864900836913
- Mean squared error: 175922.56508059194
- Median absolute error: 169.63183767642613

3.3 Ridge Regression

The deployment and results of Ridge Regression are as follows.

In [67]:
```
# 1-6 establishing a machine learning model  -- Ridge Regression

from sklearn.linear_model import Ridge

clf = Ridge()
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of Ridge Regression is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))

# store the evaluation matrix values

scores[2][0] = explained_variance_score(y_test, y_predict)
scores[2][1] = r2_score(y_test, y_predict)
scores[2][2] = mean_absolute_error(y_test, y_predict)
scores[2][3] = mean_squared_error(y_test, y_predict)
scores[2][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of Ridge Regression is as follows.
Explained variance score:  0.8817123779829996
R2 score:  0.8817122678639555
Mean absolute error:  295.5152905599042
Mean squared error:  183373.1090995079
Median absolute error:  215.66437379545414
```

Fig. 13: Code for Ridge Regression

The evaluation of Ridge Regression is as follows.

- Explained variance score: 0.8817123779829996
- R2 score: 0.8817122678639555
- Mean absolute error: 295.5152905599042
- Mean squared error: 183373.1090995079
- Median absolute error: 215.66437379545414

3.4 Least absolute shrinkage and selection operator (Lasso)

The deployment and results of Lasso are as follows.

```
In [58]: # 1-6 establishing a machine learning model  -- Lasso (Least absolute shrinkage and selection operator)

from sklearn.linear_model import Lasso
clf = Lasso()
rf = clf.fit (x_train, y_train.ravel())
y_predic = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of Lasso is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))

# store the evaluation matrix values

scores[3][0] = explained_variance_score(y_test, y_predict)
scores[3][1] = r2_score(y_test, y_predict)
scores[3][2] = mean_absolute_error(y_test, y_predict)
scores[3][3] = mean_squared_error(y_test, y_predict)
scores[3][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of Lasso is as follows.
Explained variance score:  0.8889528969200657
R2 score:  0.886518359441424
Mean absolute error:  265.5864900836913
Mean squared error:  175922.56508059194
Median absolute error:  169.63183767642613
```

Fig. 14: Code for Lasso

The evaluation of Lasso is as follows.

- Explained variance score: 0.8889528969200657
- R2 score: 0.886518359441424
- Mean absolute error: 265.5864900836913
- Mean squared error: 175922.56508059194
- Median absolute error: 169.63183767642613

3.5 Multi-Layer Perceptron (MLP)

The deployment and results of MLP are as follows.

```
In [59]: # 1-6 establishing a machine learning model  -- Multi-Layer Perceptron (MLP)

from sklearn.neural_network import MLPRegressor
clf = MLPRegressor(max_iter=1000)
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of MLP is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))



# store the evaluation matrix values

scores[4][0] = explained_variance_score(y_test, y_predict)
scores[4][1] = r2_score(y_test, y_predict)
scores[4][2] = mean_absolute_error(y_test, y_predict)
scores[4][3] = mean_squared_error(y_test, y_predict)
scores[4][4] = median_absolute_error(y_test, y_predict)


The evalution of MLP is as follows.
Explained variance score:  0.9209992539453974
R2 score:  0.9209948499023212
Mean absolute error:  251.20386626922385
Mean squared error:  122476.09914122349
Median absolute error:  195.3279246175748
```

Fig. 15: Code for MLP

The evaluation of MLP is as follows.

- Explained variance score: 0.9209992539453974
- R2 score: 0.9209948499023212
- Mean absolute error: 251.20386626922385
- Mean squared error: 122476.09914122349
- Median absolute error: 195.3279246175748

3.6 Decision Tree

The deployment and results of Decision Tree are as follows.

```
In [61]: # 1-6 establishing a machine learning model  -- Decision Tree


from sklearn.tree import DecisionTreeRegressor
clf = DecisionTreeRegressor()
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of Decision Tree is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))



# store the evaluation matrix values

scores[5][0] = explained_variance_score(y_test, y_predict)
scores[5][1] = r2_score(y_test, y_predict)
scores[5][2] = mean_absolute_error(y_test, y_predict)
scores[5][3] = mean_squared_error(y_test, y_predict)
scores[5][4] = median_absolute_error(y_test, y_predict)


The evalution of Decision Tree is as follows.
Explained variance score:  0.9355290132379899
R2 score:  0.9355286624125085
Mean absolute error:  185.77155176418535
Mean squared error:  99945.3570352102
Median absolute error:  103.316666666667
```

Fig. 16: Code for Decision Tree

The evaluation of Decision Tree is as follows.

- Explained variance score: 0.9355290132379899
- R2 score: 0.9355286624125085
- Mean absolute error: 185.77155176418535
- Mean squared error: 99945.3570352102
- Median absolute error: 103.316666666667

3.7 Extra Tree

The deployment and results of Extra Tree are as follows.

In [62]:
```
# 1-6 establishing a machine learning model  -- Extra Tree


from sklearn.tree import ExtraTreeRegressor
clf = ExtraTreeRegressor()
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)


## Evaluation Matrix:
print("")
print("The evalution of ExtraTreeRegressor is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))



# store the evaluation matrix values

scores[6][0] = explained_variance_score(y_test, y_predict)
scores[6][1] = r2_score(y_test, y_predict)
scores[6][2] = mean_absolute_error(y_test, y_predict)
scores[6][3] = mean_squared_error(y_test, y_predict)
scores[6][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of ExtraTreeRegressor is as follows.
Explained variance score:  0.9361064372270421
R2 score:  0.9361055340412346
Mean absolute error:  186.43046321560124
Mean squared error:  99051.07372957435
Median absolute error:  103.79999999999973
```

Fig. 17: Code for Extra Tree

The evaluation of ExtraTreeRegressor is as follows.

- Explained variance score: 0.9361064372270421
- R2 score: 0.9361055340412346
- Mean absolute error: 186.43046321560124
- Mean squared error: 99051.07372957435
- Median absolute error: 103.79999999999973

3.8 Random Forest

The deployment and results of Random Forest are as follows.

In [100]:
```
# 1-6 establishing a machine learning model  -- Random Forest


from sklearn.ensemble import RandomForestRegressor
clf = RandomForestRegressor()
random_forest = clf.fit (x_train, y_train.ravel())
y_predict = random_forest.predict(x_test)

## Evaluation Matrix:
print("")
print("The evalution of RandomForest is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))

# store the evaluation matrix values

scores[7][0] = explained_variance_score(y_test, y_predict)
scores[7][1] = r2_score(y_test, y_predict)
scores[7][2] = mean_absolute_error(y_test, y_predict)
scores[7][3] = mean_squared_error(y_test, y_predict)
scores[7][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of RandomForest is as follows.
Explained variance score:  0.9517013229892095
R2 score:  0.951701258239
Mean absolute error:  168.2790711110451
Mean squared error:  74874.125003281
Median absolute error:  101.05835000000002
```

Fig. 18: Code for Random Forest

The evaluation of RandomForest is as follows.

- Explained variance score: 0.9517013229892095
- R2 score: 0.951701258239
- Mean absolute error: 168.2790711110451
- Mean squared error: 74874.125003281
- Median absolute error: 101.05835000000002

3.9 AdaBoost

The deployment and results of AdaBoost are as follows.

```
In [64]: # 1-6 establishing a machine learning model  -- Adaboost


from sklearn.ensemble import AdaBoostRegressor
clf = AdaBoostRegressor()
rf = clf.fit (x_train, y_train.ravel())
y_predict = rf.predict(x_test)

## Evaluation Matrix:
print("")
print("The evalution of Adaboost is as follows.")

# Explained variance score (1 for the best)
print("Explained variance score: ",explained_variance_score(y_test, y_predict))

# R2 score (1 for the best)
print("R2 score: ", r2_score(y_test, y_predict))

# Mean absolute error
print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

# Mean squared error
print("Mean squared error: ",mean_squared_error(y_test, y_predict))


# Median absolute error (the smaller, the better)
print("Median absolute error: ",median_absolute_error(y_test, y_predict))

# store the evaluation matrix values

scores[8][0] = explained_variance_score(y_test, y_predict)
scores[8][1] = r2_score(y_test, y_predict)
scores[8][2] = mean_absolute_error(y_test, y_predict)
scores[8][3] = mean_squared_error(y_test, y_predict)
scores[8][4] = median_absolute_error(y_test, y_predict)
```

```
The evalution of Adaboost is as follows.
Explained variance score:  0.8899422377636226
R2 score:  0.8887588228539545
Mean absolute error:  314.9882687069276
Mean squared error:  172449.33303564155
Median absolute error:  250.99863942877278
```

Fig. 19: Code for Adaboost

The evaluation of Adaboost is as follows.

- Explained variance score: 0.8899422377636226
- R2 score: 0.8887588228539545
- Mean absolute error: 314.9882687069276
- Mean squared error: 172449.33303564155
- Median absolute error: 250.99863942877278

3.10 Gradient Boosting

The deployment and results of Gradient Boosting are as follows.

```
In [65]: # 1-6 establishing a machine learning model  -- Gradient Boosting

         from sklearn.ensemble import GradientBoostingRegressor
         clf = GradientBoostingRegressor()
         rf = clf.fit (x_train, y_train.ravel())
         y_predict = rf.predict(x_test)

         ## Evaluation Matrix:
         print("")
         print("The evalution of Gradient Boosting is as follows.")

         # Explained variance score (1 for the best)
         print("Explained variance score: ",explained_variance_score(y_test, y_predict))

         # R2 score (1 for the best)
         print("R2 score: ", r2_score(y_test, y_predict))

         # Mean absolute error
         print("Mean absolute error: ",mean_absolute_error(y_test, y_predict))

         # Mean squared error
         print("Mean squared error: ",mean_squared_error(y_test, y_predict))


         # Median absolute error (the smaller, the better)
         print("Median absolute error: ",median_absolute_error(y_test, y_predict))

         # store the evaluation matrix values

         scores[9][0] = explained_variance_score(y_test, y_predict)
         scores[9][1] = r2_score(y_test, y_predict)
         scores[9][2] = mean_absolute_error(y_test, y_predict)
         scores[9][3] = mean_squared_error(y_test, y_predict)
         scores[9][4] = median_absolute_error(y_test, y_predict)

         The evalution of Gradient Boosting is as follows.
         Explained variance score:  0.9460882993293025
         R2 score:  0.946088085775963
         Mean absolute error:  199.16373319950796
         Mean squared error:  83575.82946469558
         Median absolute error:  143.12377630169794
```

Fig. 20: Code for Gradient Boosting

The evaluation of Gradient Boosting is as follows.

- Explained variance score: 0.9460882993293025
- R2 score: 0.946088085775963
- Mean absolute error: 199.16373319950796
- Mean squared error: 83575.82946469558
- Median absolute error: 143.12377630169794

4 Evaluation of All Machine Learning Models

In this section, we evaluate the performances of all machine learning models and select the best one for prediction.

4.1 Evaluation Matrixes

As mentioned above, we will use five matrixes to evaluate.

- explained variance score
- R2 score
- mean absolute error
- mean square error
- median absolute error

4.2 Criterion

The evaluation criterion is as follows.

- For the first two matrixes, explained variance score and R2 score, the higher the better while the best is 1.
- For the last three matrixes, mean absolute error, mean square error, and median absolute error, the lower the better while 0 is the best.

4.3 Explained variance score

The code screenshot and the bar plot for explained variance score are as follow.

```
In [70]: # 1-7 evaluate the perforahnces of all machine learning models -- explained_variance_score

         x = np.arange(1,11,1)

         plt.bar(x,scores[:,0])
         plt.ylim([0.8,1])
         plt.title("Explained Variance Score")
```

Fig. 21: Code for Explained Variance Score

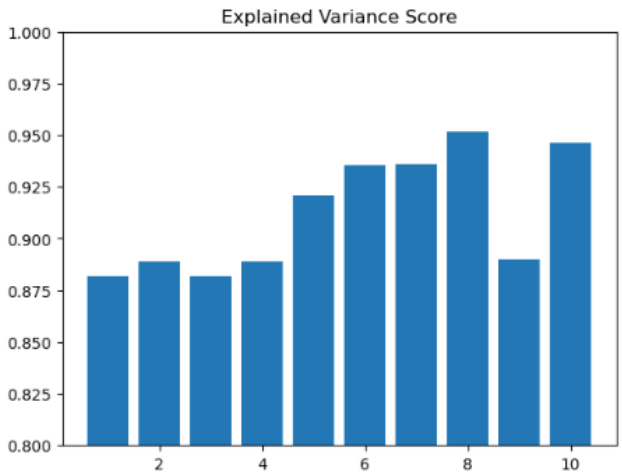

Fig. 22: Bar Plot for Explained Variance Score

From Fig. 22, it can be seen that all models achieve a score over 0.85. Among all of them, the explained variance score of Random Forest (the 8th one) is the highest, which is over 0.95. Compared to the highest value 1, 0.95 is reasonably good. Therefore, Random Forest performs the best for explained variance score.

4.4 R2 score

The code screenshot and the bar plot for R2 score are as follow.

```
In [72]: # 1-7 evaluate the perforahnces of all machine learning models -- explained_variance_score

         x = np.arange(1,11,1)

         plt.bar(x,scores[:,1])
         plt.ylim([0.8,1])
         plt.title("R2 Score")
```

Fig. 23: Code for R2 Score

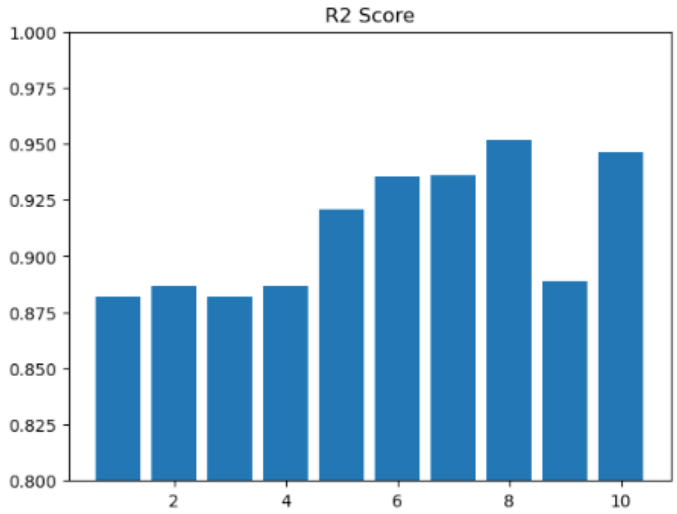

Fig. 24: Bar Plot for R2 Score

From Fig. 24, it can be seen that all models achieve a score over 0.85. Among all of them, the R2 score of Random Forest (the 8th one) is the highest, which is over 0.95. Compared to the highest value 1, 0.95 is reasonably good. Therefore, Random Forest performs the best for R2 score.

4.5 Mean absolute error

The code screenshot and the bar plot for Mean Absolute Error are as follow.

```
In [73]: # 1-7 evaluate the perforahnces of all machine learning models -- mean_absolute_error

         x = np.arange(1,11,1)

         plt.bar(x,scores[:,2])
         plt.title("Mean Abosulte Error")
```

Fig. 25: Code for Mean Absolute Error

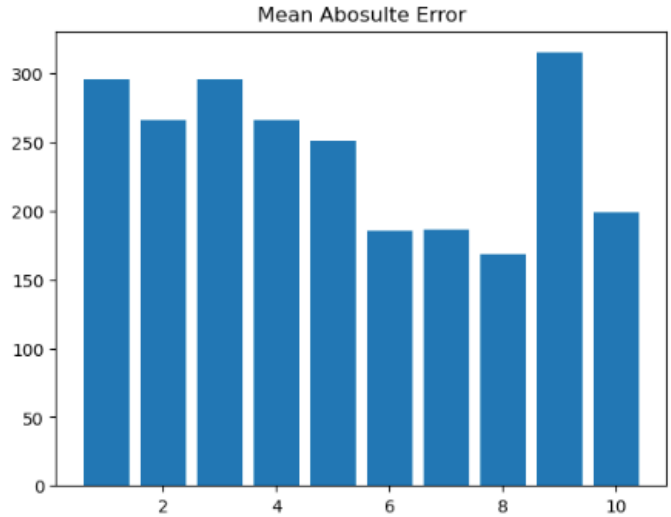

Fig. 26: Bar Plot for Mean Absolute Error

From Fig. 26, it can be seen that the highest error is over 300 and all errors are over 150. Among all of them, the Mean Absolute Error of Random Forest (the 8th one) is the lowest, which is a bit over 150. Therefore, Random Forest performs the best for Mean Absolute Error.

4.6 Mean square error

The code screenshot and the bar plot for Mean Square Error are as follow.

```
In [75]: # 1-7 evaluate the perforahnces of all machine learning models -- mean_squared_error

         x = np.arange(1,11,1)

         plt.bar(x,scores[:,3])
         plt.title("Mean Square Error")
```

Fig. 27: Code for Mean Square Error

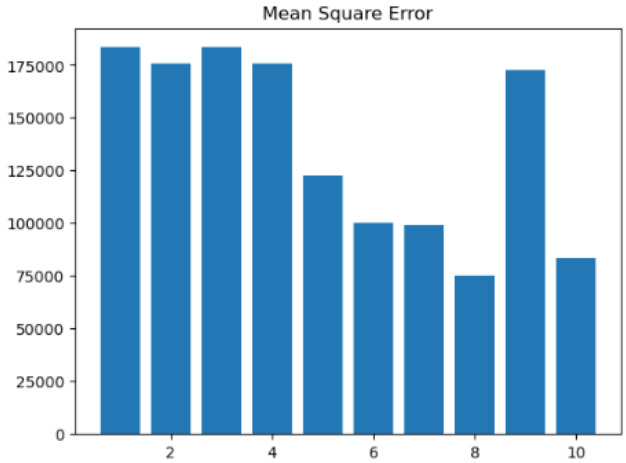

Fig. 28: Bar Plot for Mean Square Error

From Fig. 28, it can be seen that the highest error is over 175,000 and all errors are over 50,000. Among all of them, the Mean Square Error of Random Forest (the 8th one) is the lowest, which is a under 75,000. Therefore, Random Forest performs the best for Mean Square Error.

4.7 Median absolute error

The code screenshot and the bar plot for Median Absolute Error are as follow.

```
In [77]: # 1-7 evaluate the perforahnces of all machine learning models -- median_absolute_error

         x = np.arange(1,11,1)

         plt.bar(x,scores[:,3])
         plt.title("Median Abosulte Error")
```

Fig. 29: Code for Median Absolute Error

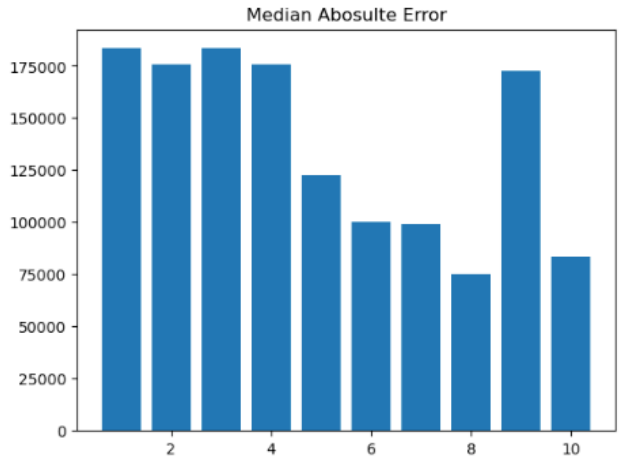

Fig. 30: Bar Plot for Median Absolute Error

From Fig. 30, it can be seen that the highest error is over 175,000 and all errors are over 50,000. Among all of them, the Median Absolute Error of Random Forest (the 8th one) is the lowest, which is a under 75,000. Therefore, Random Forest performs the best for Median Absolute Error.

4.8 Discussion on the performance

We test ten regression machine learning models in this series of experiments and using five matrixes to evaluate them. Ten regression machine learning models include Linear Regression, Support Vector Machine, Ridge Regression, Least absolute shrinkage and selection operator (Lasso), Multi-Layer Perceptron (MLP), Decision Tree, Extra Tree, Random Forest, AdaBoost, and Gradient Boosting. All the models are trained over 70% and tested over 30% of the whole dataset. For the first two matrixes, explained variance score and R2 score, the higher the better while the best is 1. For the last three matrixes, mean absolute error, mean square error, and median absolute error, the lower the better while 0 is the best. Superisingly, Random Forest has the highers scores and least errors for all five matrixes, which means it is the best fit for the dataset. Therefore, we choose it for stage 2 test on the other dataset.

5 Prediction on New Dataset

In this section, we predict the oxygen consumption of the new dataset. The example of the new dataset is shown as follows.

	A	B	C	D	E	F	G
1	step	enhanced_speed	cadence	grade	enhanced_altitude	max_heart_rate	power
2	0	26.1936	66	0	9.6	0.7162	2.075
3	1	24.858	66	0	9.6	0.7162	2.0125
4	2	25.6644	66	0	9.6	0.7162	1.9125
5	3	25.7004	66	0	9.6	0.7162	1.8375
6	4	25.7868	66	0	9.6	0.7215	1.825
7	5	25.6896	66	0	9.8	0.7215	1.8
8	6	25.7076	66	0	9.8	0.7268	1.6875
9	7	25.5744	66	0	9.8	0.7268	1.5625
10	8	24.9048	66	0	9.8	0.7268	1.525

Fig. 31 data_3 Dataset

5.1 Data preparation

In this part, we extract the three columns as Kaggle dataset.

```
In [14]: # 2-1 Load the new test data
         import xlrd

         # load the new dataset in xlsx form

         new_data = xlrd.open_workbook('data_3.xls')
         table = new_data.sheet_by_name('in')

         # extract power values
         power = table.col_values(6)
         power.pop(0)

         # extract cadence values
         cadence = table.col_values(2)
         cadence.pop(0)

         # extract heart rate values
         heart_rate = table.col_values(5)
         heart_rate.pop(0)

         # combine three columns to a new numpy array
         new_input_data = np.vstack((np.array(power),np.array(cadence),np.array(heart_rate)))
         new_input_data = new_input_data.transpose()

         new_input_data

Out[14]: array([[ 2.075 , 66.    ,  0.7162],
                [ 2.0125, 66.    ,  0.7162],
                [ 1.9125, 66.    ,  0.7162],
                ...,
                [ 1.975 , 80.    ,  0.8058],
                [ 1.9875, 80.    ,  0.8058],
                [ 2.1125, 80.    ,  0.8005]])
```

Fig. 32 Load New Dataset

5.2 Data Prediction

We use the pre-trained Random Forest model to predict the values.

```
In [106]: # 2-2 using the pre-trained random forest model to train the new data
          new_predict = random_forest.predict(new_input_data)
          new_predict

Out[106]: array([462.09483333, 462.09483333, 462.09483333, ..., 606.64675   ,
                 606.64675   , 606.64675   ])
```

Fig. 33 Predict on new data

5.3 Data Visualization

We visualize the data using line charts as follows.

```
In [110]: plt.plot(new_input_data[:,0],new_predict)
Out[110]: [<matplotlib.lines.Line2D at 0x7fe2cad60d30>]
```

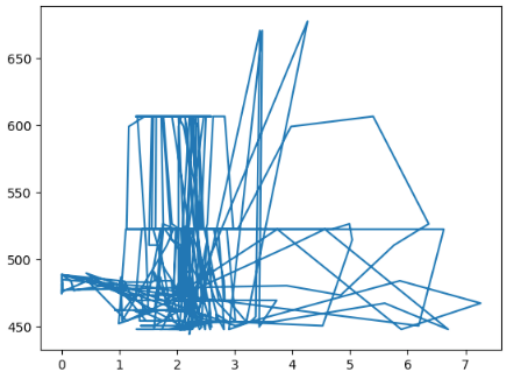

Fig. 34 Correlations between Power and Predicted Oxygen Consumption

```
In [111]: plt.plot(new_input_data[:,1],new_predict)
Out[111]: [<matplotlib.lines.Line2D at 0x7fe2d9914370>]
```

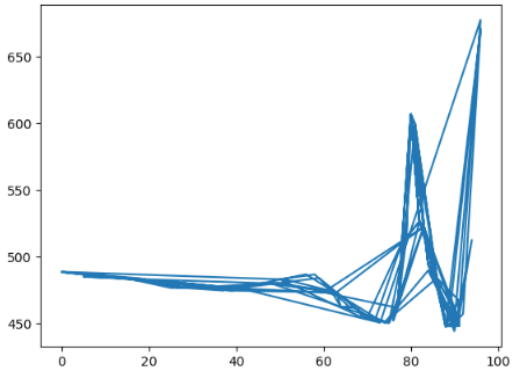

Fig. 35 Correlations between Cadence and Predicted Oxygen Consumption

```
In [112]: plt.plot(new_input_data[:,2],new_predict)
Out[112]: [<matplotlib.lines.Line2D at 0x7fe2d99e9be0>]
```

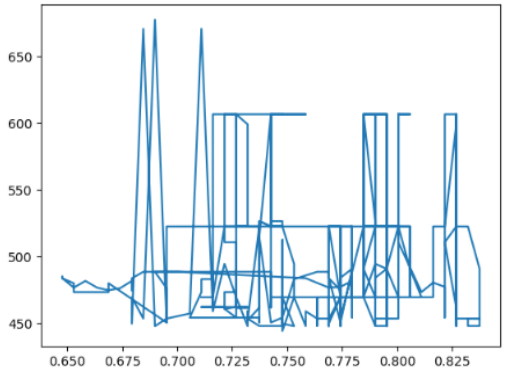

Fig. 36 Correlations between Heart Rate and Predicted Oxygen Consumption

From the visualize results, we can see the correlations somehow aligns with the correlations of Kaggle dataset. But due to the limited number of the prediction dataset, the similarity is not as good as expected.

6 Summary and Conclusion

We summarize the project with limitations and further plans

6.1 Limitations

The limitations of the project are summarized as follows.

- **Limited new dataset size**: the prediction dataset size is not large enough to show same correlations and distributions compared to the Kaggle dataset
- **Lack of the Oxygen column in the new dataset**: the new dataset does not have an Oxygen column as the benchmark to evaluate the prediction results.
- **Not trying complex deep learning models**: the limited dataset size and simple structure may cause that deep learning cannot release its potential.

6.2 Further plans

For further plans, there are some promising ones.

- **Considering more attributes**: more attributes may further improve the performance from the terms of accuracy
- **Using another dataset to predict:** using another dataset with Oxygen column will help with the evaluation of prediction results.
- **Trying deep learning models:** deep learning models may help with the performance enhancement if we have more data and more attributes involved.

To sum up, the project comprehensively evaluates ten popular machine learning models when prediction the oxygen consumption using Kaggle dataset. The best one is identified as Random Forest, which is measured by five matrixes. Then, the pre-trained model is used for prediction on the new dataset. Based on the results and analysis, we provide limitations and further plans for the project.