Research Draft – Oxygen Prediction Algo Documentation

Redback Operations

Prototype Team - Shashvat Joshi

Abstract

Artificial Intelligence, Machine Learning, Deep Learning are disrupting almost every industry. AI/Data science is also revolutionizing the healthcare, wellness, fitness industry. Over the decades we have been acknowledging the fact "Health is wealth". However, the Covid pandemic taught us the universal truth a hard way. We recognized the importance of every aspect of a healthy lifestyle, awareness, and increased focus of science and technology in healthcare.

The role of Data analysis, Data Science, AI is expanding fast in sports, health & fitness programs.

One such use case is the study of oxygen uptake in (VO2), Both muscle and pulmonary VO(2) rise in an exponential fashion after the onset of exercise. It can take 2-4 min, or even longer at higher work rates to reach the steady state. Slow VO(2) kinetics increase the so-called O(2) deficit and obligate a greater contribution from anaerobic mechanisms of ATP production (involving the breakdown of muscle high energy phosphates and lactate production from glycogen) to meet the ATP requirement of the exercise task.

Measurement of oxygen uptake is expensive and not everyone can afford to have such expensive tools. The purpose of this research is to study this new algorithm that can predict the oxygen uptake for cyclists using an alternative method. Racing cyclists need personalized optimized fitness training. The proposed algorithm is focused to achieve a discrete-event non-linear model where one of the fitness attributes - "power" is used in a different way. Power is treated as input which will control the body to respond accordingly; rather than as the output which is a normal technique used in most of the algorithms. Heart rate is treated as output. Dynamic Heart rate is the response of the body during exercise and is explained with the time constant in this method. The research has also presented a comparative study between the established predictive algorithmic techniques and this new way of using the Power and Hear Rate as fitness attributes. Let's explore how can we benefit from this new algo research in the next section.

Introduction

Fitness plays a vital role to measure the ability of an athlete. An athlete performs at his/her best in a race as per their fitness given other external conditions in the event and their preparation and talent. Fitness training of athletes or any sportsperson is designed to unlock this maximum potential. Training is a complex process where the fitness of the body along with the freshness of the body is carefully balanced to get the best result. Training is the procedure of unlocking one athlete's maximum potential, aiming for peak performance in a race. Training is a complex art where firstly the overall body's ability to perform and secondly the athlete's freshness on the day of the event is both carefully balanced; thus, this procedure can be referred to as fitness optimization. If we want to measure the fitness process quantitatively, it becomes difficult due to the variety of parameters affecting the cardiovascular and skeletomuscular control mechanisms. The efficiency of supplying the muscles with the required oxygen, both for aerobic respiration as well as for the breakdown of anaerobic wastes also depend on various body parameters which are difficult to control. We want to avoid fatigue on the day of the event which is caused due to body reaching the maximum capacity. Reaching

maximum body capacity without fatigue is obtained by proper fitness training. Here exactly we use the various parameters and tune them so they can not affect the performance of the body. The key parameters are increased plasma volume, increased lactate threshold, increased muscle capillarization, increased VO2 max, increased muscle high energy phosphate stores, increased stroke volume, and several others. The usual method tries to tune these parameters using different types of training, it also controls the process to best adapt the athlete's body to the demanding needs of the race.

The established method is that oxygen supply is the input and power is the output. But assessing the current fitness of the athlete, even though it can certainly be related, is not the same as predicting his fitness on the day of the event. Hence new study is proposing a model which can take into account both the capacity as well as the fatigue of the athlete on that day.

The most important model that is currently widely used in the cycling industry and is found built-in in as well, such as

Performance Manager of Hunter Allen and Andrew Coggan talks about two well-known cycling power software TrainingPeaks and CyclingPeaks, where post-ride analysis is easily done, is used widely in the cycling industry. Its concept lies in the basic principle of training: breakdown of 1 the organism due to exercise, is followed by rest (recovery), which results in an improvement in performance (adaptation). An imbalance in the training load and recovery time can result in symptoms of fatigue. If the imbalance between training and rest persists, the athlete may develop serious symptoms of fatigue that will negatively affect his ability to sustain a high training volume and causes poor performance in the race.

Hence recovery must match the breakdown to retain the freshness of the athlete on a racing day.

The main advantage of the Performance Manager is that it captures the empirical knowledge of coaches regarding training, overtraining, and fatigue. The current purely physiological or semi-quantitative empirical techniques, taken so far, despite the respective advantages they offer, require a tool to bridge the gap between the two worlds: the endogenous physiological one, with the exogenous systemic, more empirical, one. This is exactly what we are trying to explore algorithmic systems approach whose parameters do not exclusively depend upon either ability or freshness, taking into account both endogenous as well as exogenous information.

Research Details

As per the current research, the Power is input, and Heart rate is the output response of the system

Power vs Heart Rate

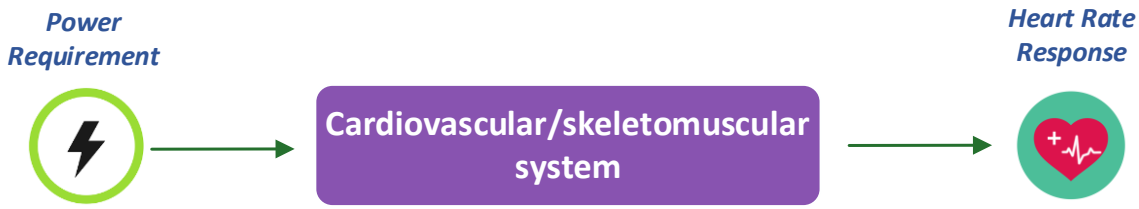


Fig. 1: *Power and heart rate relationship in a new method.—*

The relationship between steady-state power and heart rate under carefully controlled conditions is proven to be a good indicator of fitness. However, we cannot ignore the time dependency. The time constant is less dependent on the type of training being undertaken (intervals, sustained efforts, long rides, etc) than the absolute value of heart rate required for a given power. We will develop the model to explore these dynamic dependency factors. This approach looks promising to capture the current state of the body by looking closely at how the latter responds to the varying stress the athlete exerts on it.

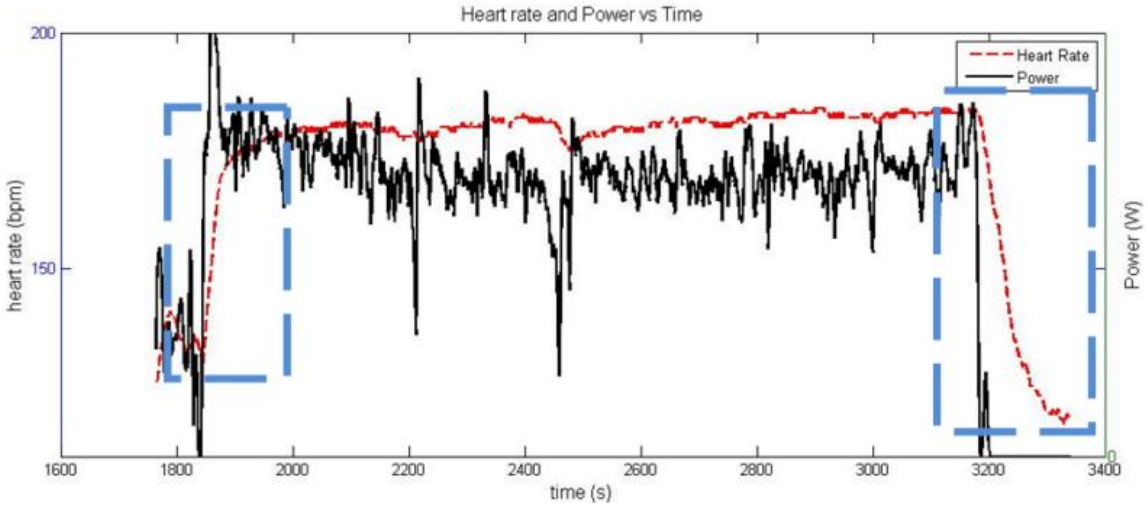


Fig. 2: *Power and heart rate dynamic relationship. The asymmetric property of the heart rate (the fact that heart rate increase is faster than heart rate decrease) is also observed.*

As per Fig-2 the time constant related to the increase of heart rate referred to later as tc1, is shorter than the corresponding constant for a heart rate decrease, tc2. This illustrates the asymmetric property of the heart rate. There is a lag introduced between the power step input and the heart rate response. This lag applies both to step increases and step decreases in power, yet probably it has a different size.

Explore Training data & Model

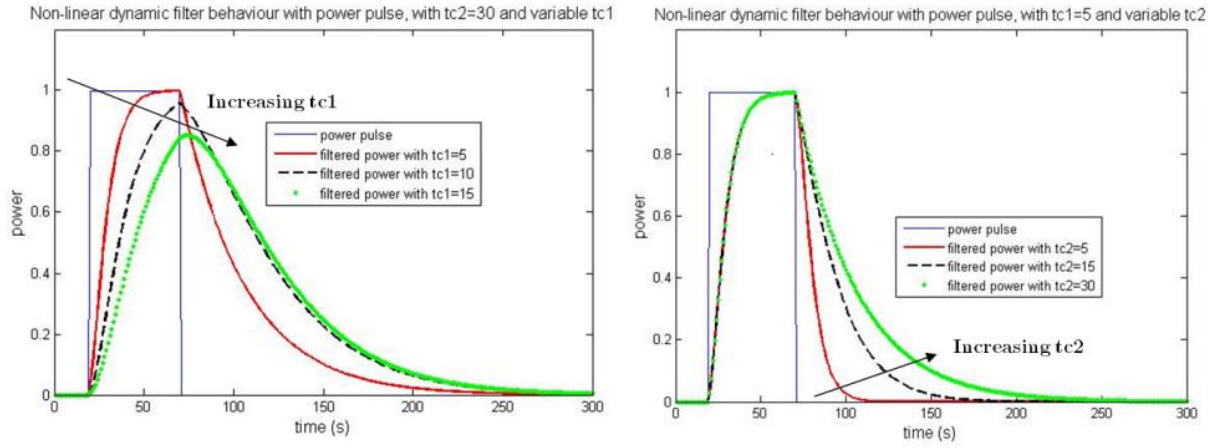


***Fig 3**: tc1 and tc2 effect on the non-linear dynamic discrete-event filter. The outputs behavior is sensitive to tc1 and tc2, as required. Response agrees relatively well with heart-rate physiology.*

Due to the nonlinear response of heart rate, the continuous state systems approach was not applicable here. Several models were tested with various nonlinear combinations of Infinite Impulse Response filters without proper results. Hence a bottom-up approach is taken. The system is thought to be a second-order mechanical system with non-linearities and responding to external forces. This can be obtained by setting different conditions for selecting the value of the time constant T such that T=tc1, when power (pow) > predicted heart rate (phr), and T=tc2 when pow < phr. In other words, there is an addition of discontinuity in the time constant component. This is a simple example of simulation for continuous state systems using discrete event system specifications.

The new model is

$$T^2(\ddot{phr}) + 2T(\dot{phr}) + (phr) = pow, \begin{cases} T = tc_1, pow > phr \\ T = tc_2, pow < phr \end{cases} \quad (1)$$

The new non-linear discrete event model equation was subsequently discretized and was turned in a form that could be used to obtain the predicted heart rate at each instant (eq 2)

$$phr(a) = \frac{pow(a) + 2Tphr(a-1) + T^2(2phr(a-1) - phr(a-2))}{(1+T)^2} \quad (2)$$

Math behinds the model

The testing process involved plotting of response of the model under an extensive filter to a particular pulse of power (10-30 seconds), with different values of tc1 and tc2, as illustrated by Figure 3. As the plot shows, the filtered power shows the expected responses according to physiology. Power can be thought of as the precursor of predicted heart rate. If the two constants are equal, then their response is symmetric, yet when they are not equal, the two tails of the response exhibit lag analogous to their inequality. With a slight change in tc1 or tc2, the response is changing respectively.

Flow Diagram

The algorithm flow diagram is

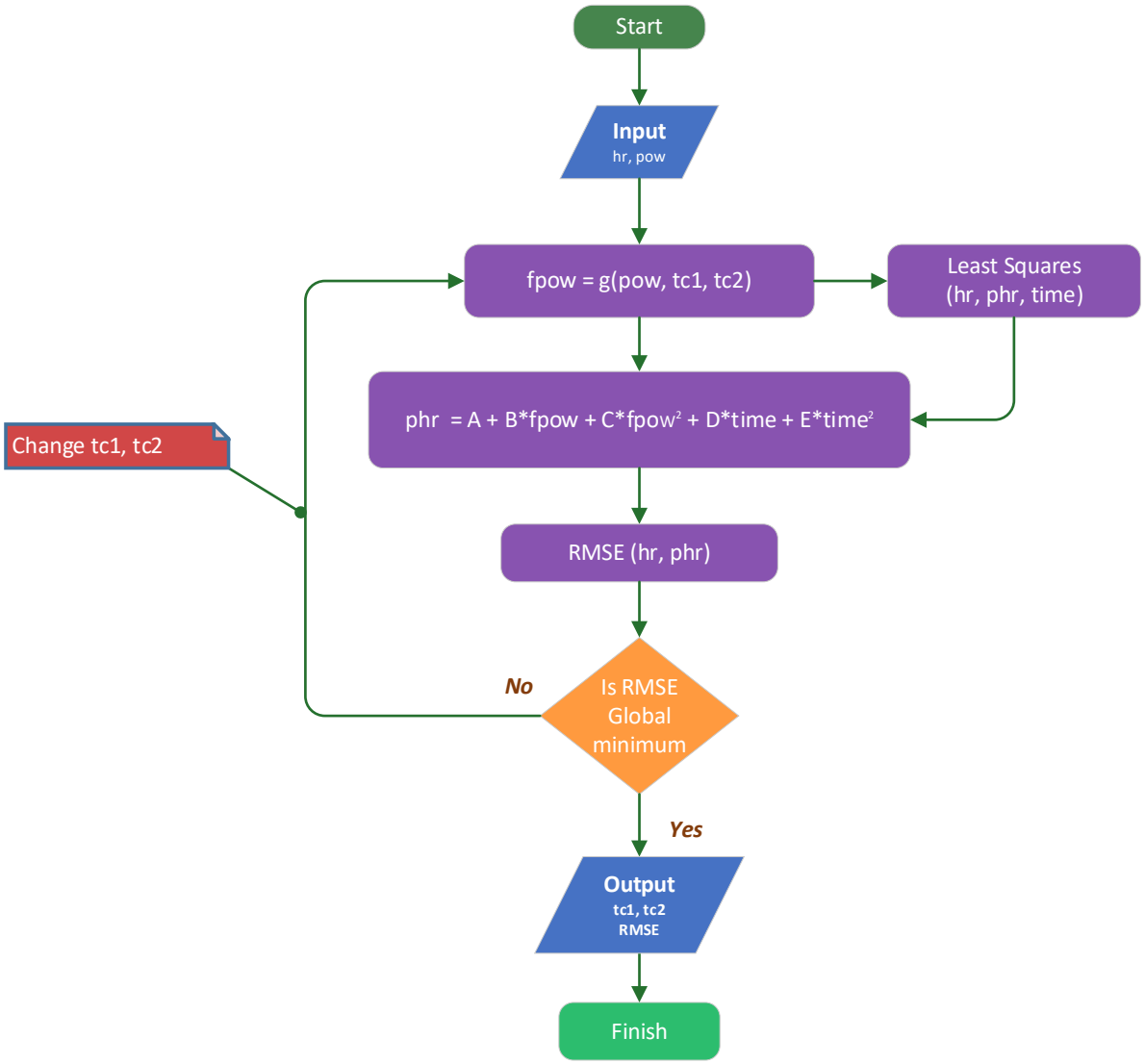


***Fig. 4**: Core algorithm flowchart*

The core algorithm refers to two outputs, tc1 and tc2. The methodology used involves initial filtering of power (function g(pow, tc1, tc2)), followed by least-squares analytical fitting between filtered power, time, and heart rate. The model parameters, found using optimization methods, were those which together yield the smallest root mean squared error (RMSE) between predicted and true heart rate. Here we can see the use of the least-squares fitting; unknown constants A, B, C, D, and E allow for variation of absolute heart rate with power and time without affecting the estimation of the underlying time constants but are not used in this analysis. As it was found by running on real athletes' data (amateur and semi-professional), it was clearly shown that both tc1 and tc2 were very reasonable and agreed with physiology.

The confidence interval for each model parameter (i.e. the time constant tc1 or tc2), should be minimized for the best fit model. A 2D likelihood grid was constructed assuming the error is Gaussian distributed. The marginal distributions were calculated. The confidence intervals using the entire data sets were found to be relatively large, therefore a way to narrow them down had to be found. Training sessions that have clearly defined pulses, were found to have narrower confidence intervals. The 2-D probability surface of such training session is delta-like, which also corresponds to shorter confidence interval widths. It was therefore decided that an extended algorithm that could select those regions of higher structure and thus provide more accurate results should be developed.

This was done in practice by rewarding large heart rate variations on the sides of the region, but penalizing both frequent as well as intra-step variations. As there are several local optima, the optimization algorithm is initialized at several points in the data set, thus trying to find a global optimum for values tc1 and tc2 with respect to RMSE. The detailed evaluation is still under investigation.

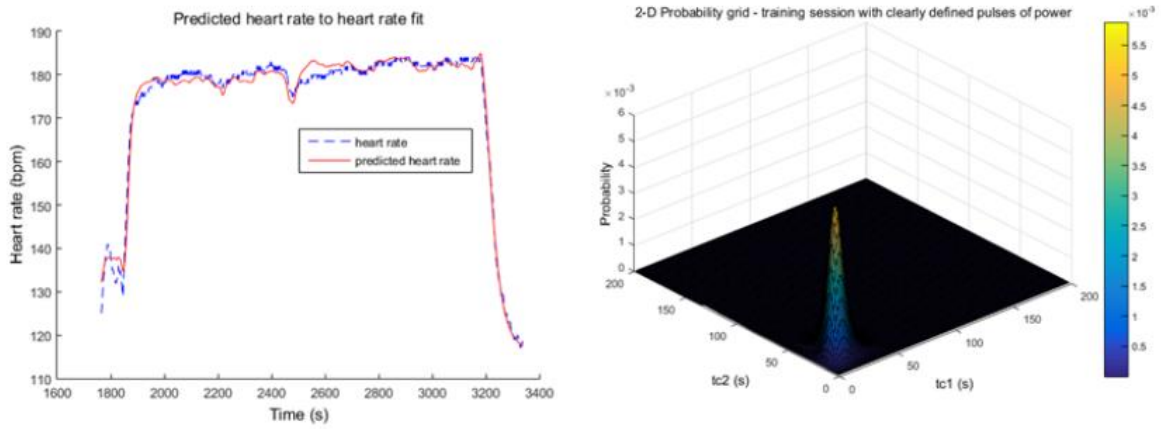


***Fig. 5**: The data fit and the 2-D Probability grid surface plot of the training session with a clearly defined pulse of power, showing a more delta-like peak of probability at estimated values of tc1 and tc2 (Corresponding to max likelihood). The grid surface is normalized to 1*

Comparing the result with an established model

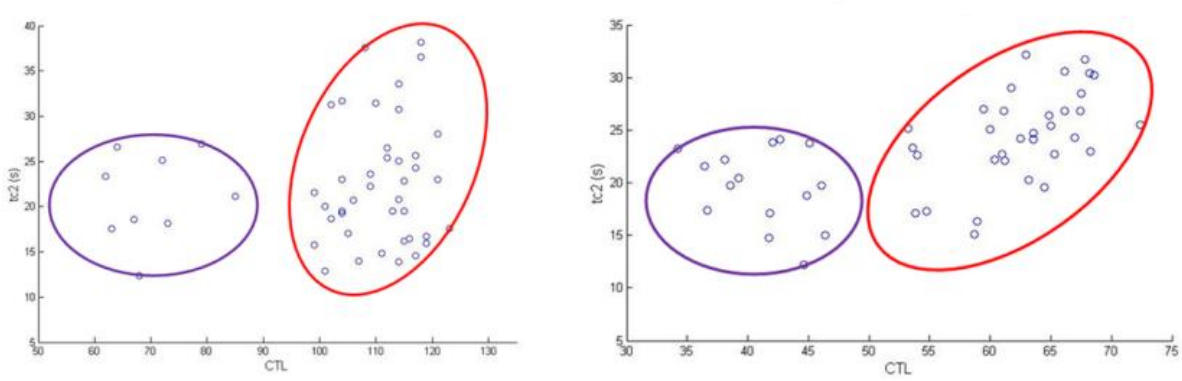


***Fig. 6**: Model parameter tc2 plotted against Chronic Training Load (CTL) for Athletes 1 and 2. tc2 shows a possible linear relationship with CTL after a particular threshold of CTL. The threshold is higher for the younger athlete (20 years old) when compared with the older athlete (50 years old), as expected.*

The research was done for two athletes with large training data. The Performance Manager concept is based on the widely known Training Stress Score (TSS), the measure of the effort the cyclist put during a training session. Both the intensity and the duration of each training session are controlled by TSS. It can be viewed as a predictor of the amount of glycogen utilized in each workout. TSS is then used to calculate the Chronic Training Load (CTL) and Acute Training Load (ATL), which are usually the 42 and 7 days exponentially weighted moving averages of daily TSS. CTL is thought to provide a measure of how much an athlete has been training historically, whereas ATL is a measure of how much an athlete has been training recently, or acutely. The difference between these two values is called the Training Stress Balance and provides a measure of freshness. The overtraining is detected as the tc2 output of the algorithm for those two athletes. A high value of CTL is considered as having a positive impact on the athlete, whereas ATL has a negative impact.

Conclusion

This research is focused on building an open-source personalized fitness predictor model using a simple, algorithmic methodology, compatible with both physiological as well as empirical models. The model has provided realistic evidence that indicates that long-term effects of training related to fatigue can be captured in a set of model parameters and potentially this could also be extended to capture effects related to the performance of the athlete. The performance parameters are independent of the training schedule but have an impact on the athlete.

Document History

Task Name	Date added	General Info	
Initial Version	17-Mar-2022	Abstract, Introduction, Dataset details, Data Understanding of subject/Cyclist 1 Relation between target and other feature variables. Explored the correlation among various features by using cor() function and visualize using heatmap.	The data is taken from two trial workouts, one Wingate and one incremental test. This week we mainly focused on understanding the target and other numeric features.
V1.1	24-Mar-2022	EDA – Univariate , Bivariate data analysis sections added Visualization added for each type of EDA for subject-1 data for all 4 types of trials. Missing data treatments Outlier treatment	
V1.2	31-Mar-2022	Build the first Model Section – "Building Model-1 (NN-MLPRegressor)	
V1.3	7-Apr-2022	Building Model-2: Linear Regression	
V1.4	15-Apr-2022	Building Model 3: Decision Tree	
V1.5	5-May-2022	The Final Analysis and Conclusion added.	

Introduction

Artificial Intelligence, Machine Learning, Deep Learning are disrupting almost every industry. AI/Datascience is also revolutionizing the healthcare, wellness, fitness industry. Over the decades we have been acknowledging the fact "Health is wealth". However, the Covid pandemic taught us the universal truth a hard way. We recognized the importance of every aspect of a healthy lifestyle, awareness, and increased focus of science and technology in healthcare.

The role of Data analysis, Data Science, AI is expanding fast in sports, health & fitness programs.

One such use case is study of oxygen uptake in (VO2), Both muscle and pulmonary VO(2) rise in an exponential fashion after the onset of exercise. It can take 2-4 min, or even longer at higher work rates to reach the steady-state. Slow VO(2) kinetics increase the so-called O(2) deficit and obligate a greater contribution from anaerobic mechanisms of ATP production (involving the breakdown of muscle high energy phosphates and lactate production from glycogen) to meet the ATP requirement of the exercise task.

Measurement of oxygen uptake is expensive and not everyone can afford to have such expensive tools. The purpose of this research is to study few Machine Learning algorithms with simple measurable data to predict the oxygen uptake for cyclists.

Methods

As per the current research the below steps will be followed

- Data understanding
- Data cleaning & preparation
    - cleaning missing values
    - removing redundant columns
    - imputations
    - outliers treatment
    - set the proper datatype
    - one hot encoding/dummy variables etc.)
- Data Analysis
- Model Development
- Model Evaluation
- Prediction

Extrapolatory Data Analysis (EDA)

We are using the sample data from kaggle:

https://www.kaggle.com/stpeteishii/cycling-vo2-with-autoviz/data

The data has been collected from 7 cyclists (called subject). Each cyclist has gone through two different protocol tests – protocol-1 & protocol-2 before going through Wingate Test. The incremental test results are also captured in a separate dataset.

As of now we want to study 3 models with Machine Learning & Deep Learning - Neural Network, LSTM Models.

The data includes power output (P), respiratory frequency (Rf), heart rate (HR), cadence (ω). Features are RF, HR, w.

Task Research Detail

The code is developed in Jupyter notebook & python. The code screenshots and output screen shots are attached below.

This is in-progress document. More EDA details will be shared next week along with algorithm research understanding.

We will also share the flow diagram and approach to solve this use case.

Data Understanding

Oxygen uptake Prediction for Cycling- EDA & Investigation

Part 1

1. Data understanding
2. Data cleaning & preparation
   - cleaning missing values
   - removing redundant columns
   - imputations
   - outliers tratment
   - set the proper datatype
   - one hot encoding/dummy variables etc.)
3. Data Analysis
4. Model Development
5. Model Evaluation
6. Prediction

Import required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Load data and read data into pandas data frame

Load the data from Cyclist-1 or Subject-1's exercise trials

In [22]:
```
# reading data files

df1 = pd.read_csv('sbj_1_I.csv')
df2 = pd.read_csv('sbj_1_II.csv')
df3 = pd.read_csv('sbj_1_Wingate.csv')
df4 = pd.read_csv('sbj_1_incremental.csv')
```

4 datasets of subject-1 are converted to 4 dataframes – df1, df2, df3, df4.

- We started the EDA and explored the dataset as below.

Data understanding

In [3]:
```
# Look at first few records of the dataset
print(df1.head())
```

```
   time  Power      Oxygen  Cadence         HR         RF
0     1    0.0  318.400000        0  75.600000  20.100000
1     2    0.0  356.166667        0  75.666667  19.750000
2     3    0.0  403.285714        0  75.714286  19.428571
3     4    0.0  456.250000        0  75.750000  19.125000
4     5    0.0  478.925926        0  75.740741  18.962963
```

In [4]:
```
print(df2.head())
```

```
   time  Power      Oxygen  Cadence         HR         RF
0     2    0.0  454.500000      0.0  69.600000  26.300000
1     3    0.0  501.583333      0.0  69.500000  25.083333
2     4    0.0  524.261905      0.0  69.523810  24.166667
3     5    0.0  531.687500      0.0  69.625000  23.437500
4     6    0.0  528.944444      0.0  69.777778  22.833333
```

In [5]:
```
print(df3.head())
```

```
   time  Power      Oxygen  Cadence         HR         RF
0     2    0.0  329.533333      0.0  82.333333  18.400000
1     3    0.0  345.333333      0.0  82.000000  18.666667
2     4    0.0  361.000000      0.0  81.571429  18.857143
3     5    0.0  387.312500      0.0  81.187500  19.312500
4     6    0.0  420.722222      0.0  80.833333  19.944444
```

In [6]:
```
print(df4.head())
```

```
   time  Power    Oxygen  Cadence    HR    RF
0     3    0.0  602.0000      0.0  86.0  16.0
1     4    0.0  578.1250      0.0  86.0  16.0
2     5    0.0  558.7500      0.0  86.0  16.0
3     6    0.0  542.1875      0.0  86.0  16.0
4     7    0.0  527.5000      0.0  86.0  16.0
```

In [7]:
```
# inspect the structure etc.
print(df1.info(), "\n")
print(df1.shape)
print("***********\n")

print(df2.info(), "\n")
print(df2.shape)
print("***********\n")

print(df3.info(), "\n")
print(df3.shape)
print("***********\n")

print(df4.info(), "\n")
print(df4.shape)
print("***********\n")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2819 entries, 0 to 2818
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   time     2819 non-null   int64
 1   Power    2819 non-null   float64
 2   Oxygen   2819 non-null   float64
 3   Cadence  2819 non-null   int64
 4   HR       2819 non-null   float64
 5   RF       2819 non-null   float64
dtypes: float64(4), int64(2)
memory usage: 132.3 KB
None

(2819, 6)
***********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2579 entries, 0 to 2578
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     2579 non-null   int64  
 1   Power    2579 non-null   float64
 2   Oxygen   2579 non-null   float64
 3   Cadence  2579 non-null   float64
 4   HR       2579 non-null   float64
 5   RF       2579 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 121.0 KB
None

(2579, 6)
**********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 751 entries, 0 to 750
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     751 non-null    int64  
 1   Power    751 non-null    float64
 2   Oxygen   751 non-null    float64
 3   Cadence  751 non-null    float64
 4   HR       751 non-null    float64
 5   RF       751 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 35.3 KB
None

(751, 6)
**********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2831 entries, 0 to 2830
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   time     2831 non-null   int64
 1   Power    2831 non-null   float64
 2   Oxygen   2831 non-null   float64
 3   Cadence  2831 non-null   float64
 4   HR       2831 non-null   float64
 5   RF       2831 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 132.8 KB
None

(2831, 6)
**********
```

Univariate

Univariate Analysis

In univariate analysis, each variable is analyzed individually and we don't look at more than one variable at a time. It is the simplest and basic form of analysis.

Univariate Analysis can be done for two kinds of variables- Categorical and Numerical. In our dataset all the columns are numeric.

Numerical Variable

Various descriptive statistics such as Measures of Frequency (count), Shape (skewness, Kurtosis), Variability (Minimum value, Maximum value, Range, Quantile, Variance, Standard Deviation), Central Tendency (Mean, Median, Mode) can be used to explore a numerical variable. The various visualization techniques that can be used are mainly histogram and box plot.

There is no missing value or NAN values in any of the data sets. All the columns are numeric.

Understanding the data distribution of numeric data through visualization

```
: # understanding the data distribution of numeric data
print(df1.describe())
print("*************\n")

print(df2.describe())
print("*************\n")

print(df3.describe())
print("*************\n")

print(df4.describe())
print("*************\n")
```

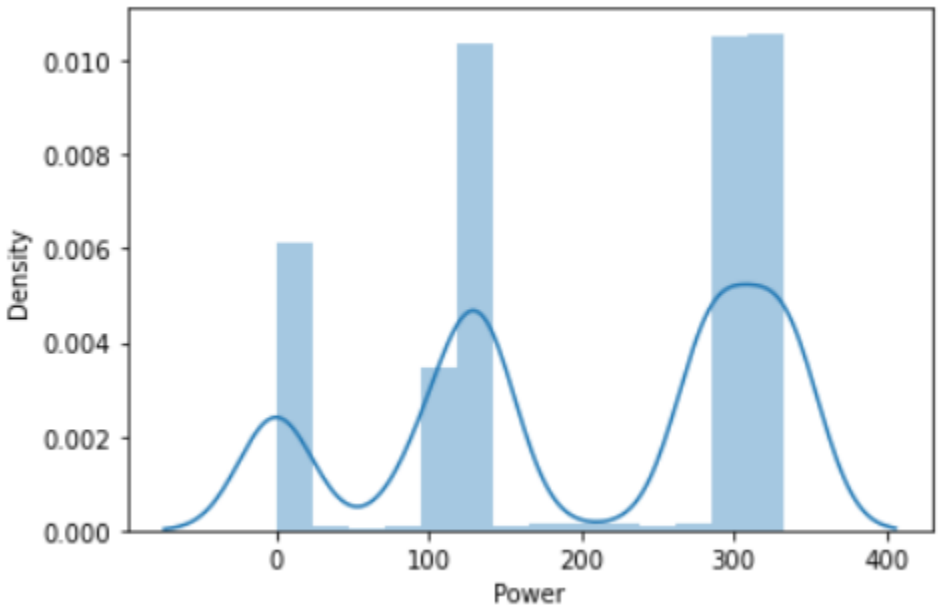
```
                time         Power       Oxygen      Cadence           HR  \
count  2819.000000  2819.000000  2819.000000  2819.000000  2819.000000
mean   1410.000000   200.824761  2827.277800    76.974104   145.176680
std     813.919529   119.034937  1274.053153    31.670372    33.330737
min       1.000000     0.000000   168.476667     0.000000    72.358333
25%     705.500000   134.000000  1992.500000    90.000000   125.375000
50%    1410.000000   286.000000  3331.200000    90.000000   151.200000
75%    2114.500000   313.100000  3909.850000    90.000000   174.000000
max    2819.000000   333.000000  4575.600000    90.000000   189.000000

                RF
count  2819.000000
mean     39.411175
std      10.463798
min      11.590000
25%      33.750000
50%      41.350000
75%      46.900000
max      61.000000
*************
```

```
              time        Power       Oxygen      Cadence           HR  \
count  2579.000000  2579.000000  2579.000000  2579.000000  2579.000000
mean   1291.000000   117.853625  1741.433585    73.154323   106.500047
std     744.637496   107.393545   992.355408    33.159856    26.666996
min       2.000000     0.000000   140.250000     0.000000    65.000000
25%     646.500000     0.000000   730.825000    80.000000    81.016667
50%    1291.000000   134.000000  1862.850000    90.000000   110.000000
75%    1935.500000   207.600000  2634.075000    90.000000   127.025000
max    2580.000000   304.000000  3636.900000   100.000000   152.000000

                RF
count  2579.000000
mean     29.790830
std       7.008515
min      12.250000
25%      23.500000
50%      30.600000
75%      35.700000
max      49.900000
*************
```

```
             time       Power       Oxygen     Cadence          HR          RF
count  751.000000  751.000000   751.000000  751.000000  751.000000  751.000000
mean   377.000000   25.178616   877.716183    6.129668  103.634102   27.155250
std    216.939316  116.387454   691.745859   25.854521   17.115343   10.926758
min      2.000000    0.000000   219.480000    0.000000   78.850000   11.428571
25%    189.500000    0.000000   515.705000    0.000000   86.040000   18.658333
50%    377.000000    0.000000   678.866667    0.000000  101.750000   25.616667
75%    564.500000    0.000000   865.350000    0.000000  117.000000   31.150000
max    752.000000  823.855526  3822.200000  153.268421  140.250000   69.300000
*************
```

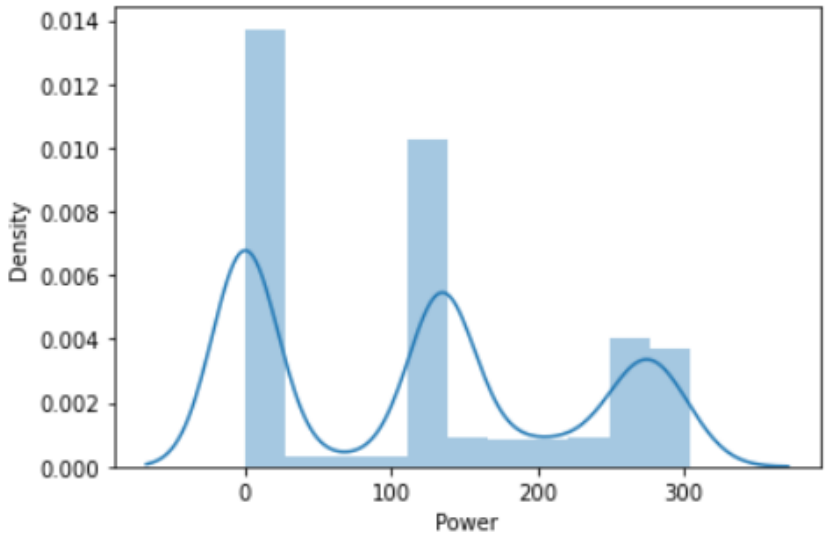
```
              time        Power       Oxygen      Cadence           HR  \
count  2831.000000  2831.000000  2831.000000  2831.000000  2831.000000
mean   1418.000000   148.163193  2241.277737    66.856235   128.732436
std     817.383631   129.748206  1298.569808    39.215841    29.108385
min       3.000000     0.000000   105.273333     0.000000    79.340000
25%     710.500000     0.000000  1036.703947     0.000000   106.066667
50%    1418.000000   140.000000  2156.600000    90.000000   121.750000
75%    2125.500000   260.000000  3380.175000    90.000000   158.125000
max    2833.000000   380.000000  4732.600000    90.000000   177.400000

                RF
count  2831.000000
mean     29.925802
std       9.799764
min      10.580000
25%      23.412500
50%      29.233333
75%      34.300000
max      62.100000
*************
```

```
# Plot Power - target variable for each dataset

sns.distplot(df1['Power'])
plt.show()
```



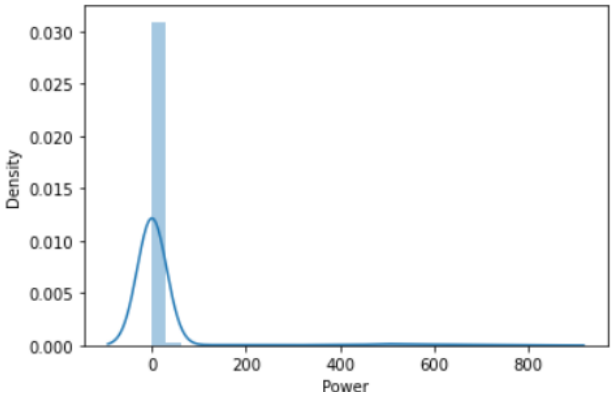

```
sns.distplot(df2['Power'])
plt.show()
```

```
C:\Users\admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: F
s a deprecated function and will be removed in a future version. Please ada
`displot` (a figure-level function with similar flexibility) or `histplot`
r histograms).
  warnings.warn(msg, FutureWarning)
```



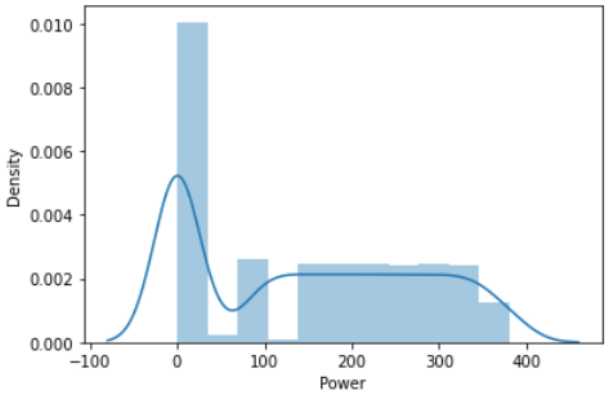
```
sns.distplot(df3['Power'])
plt.show()
```

```
C:\Users\admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



:
```
sns.distplot(df4['Power'])
plt.show()
```

```
C:\Users\admin\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

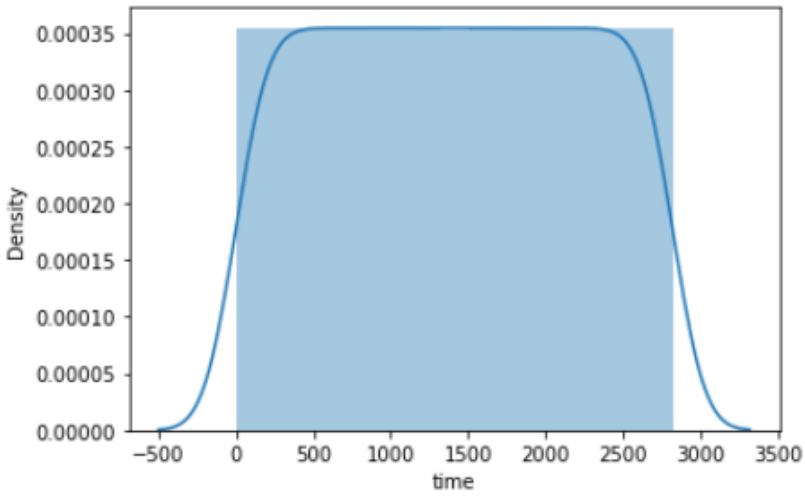


Subject-1 Trail-1 dataset

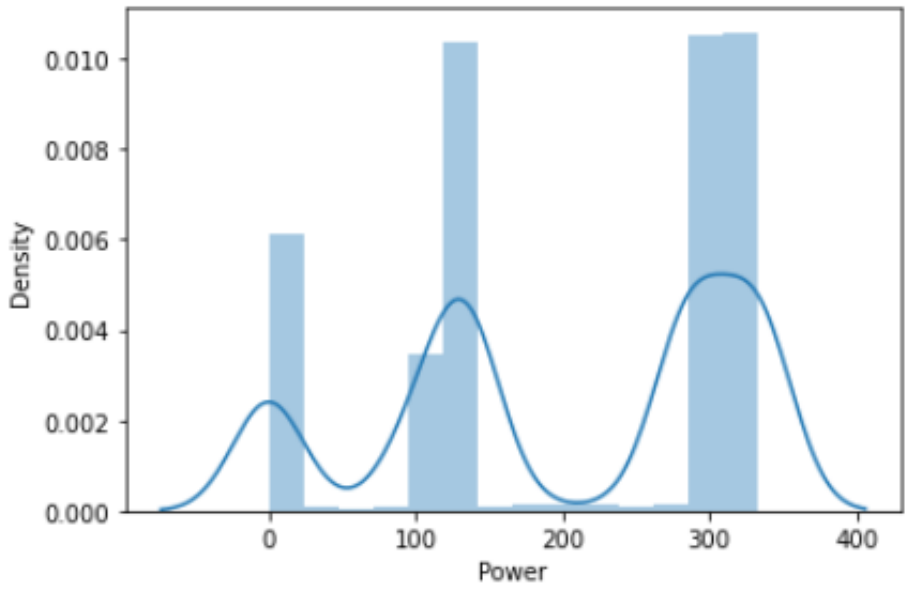
Let's check the data distribution of other numeric features: dataset – subject-1

```
#fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
for var in df3.columns:

    print("********The data distribution of", var, "********\n")

    #plt.figure(figsize=(6, 4))
    sns.distplot(df1[var])
    plt.show()
```

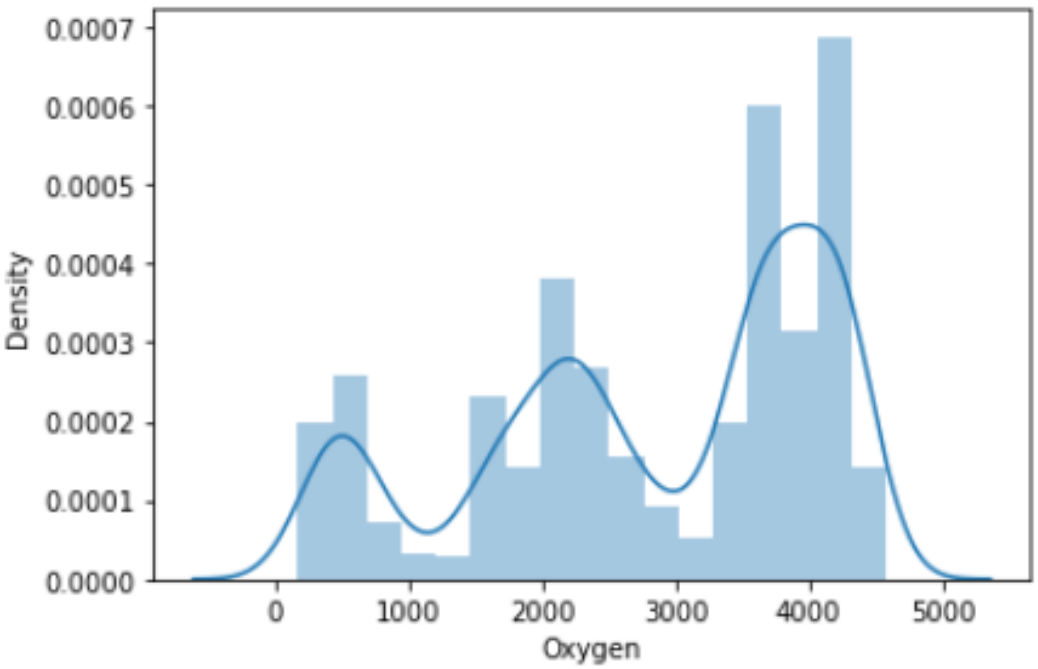
```
********The data distribution of time ********
```



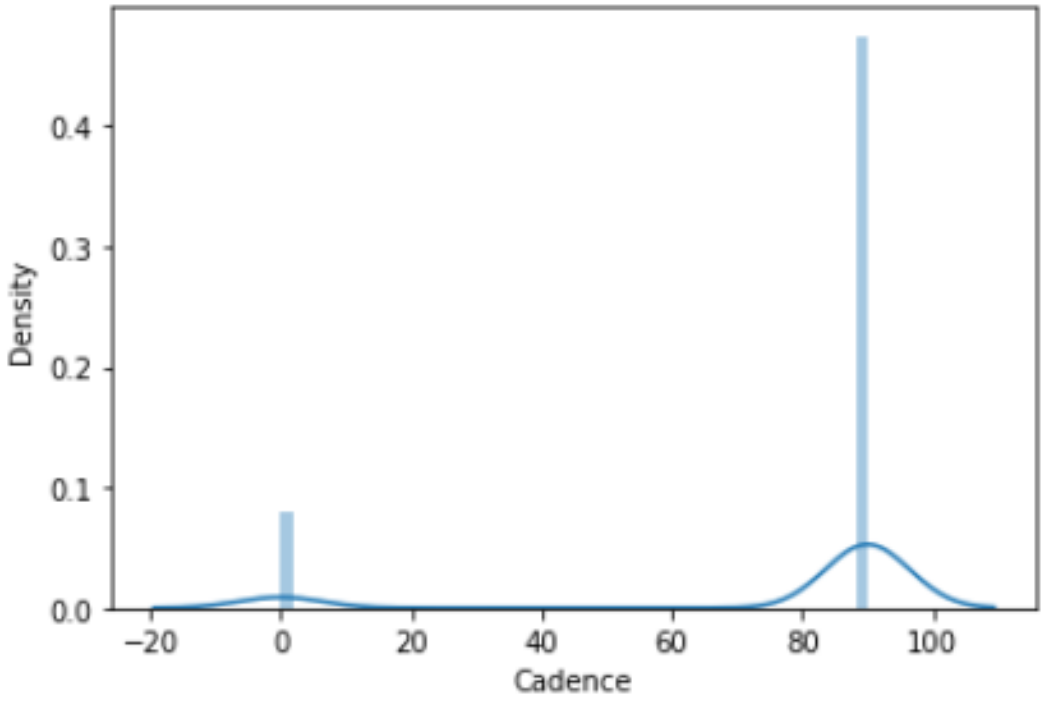

```
********The data distribution of Power ********
```



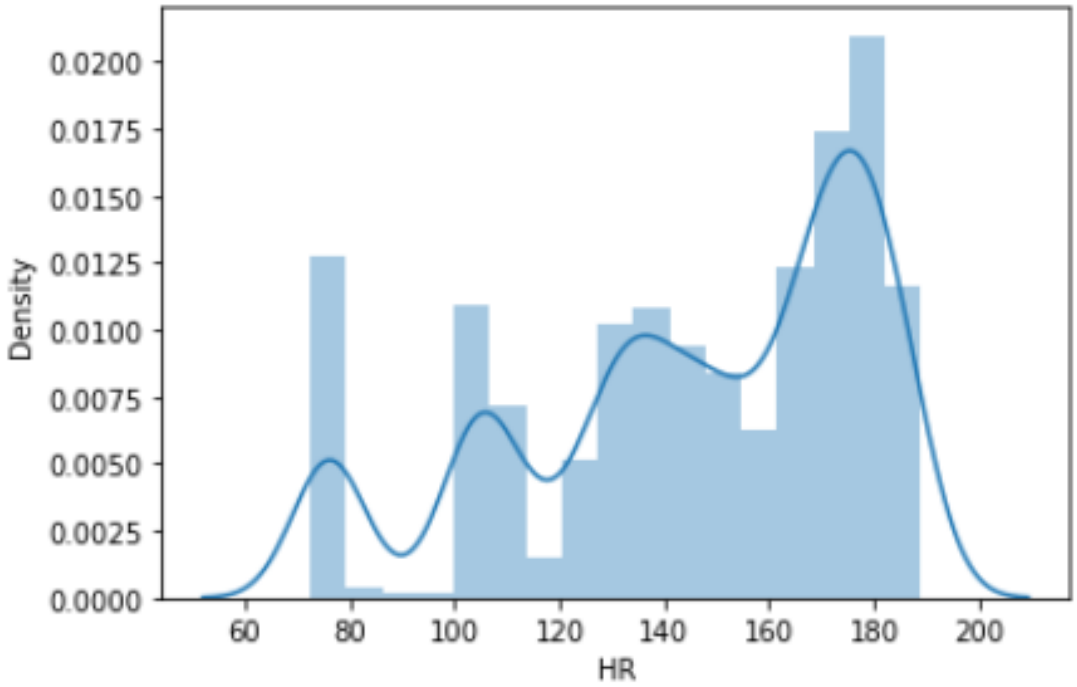

```
********The data distribution of Oxygen ********
```



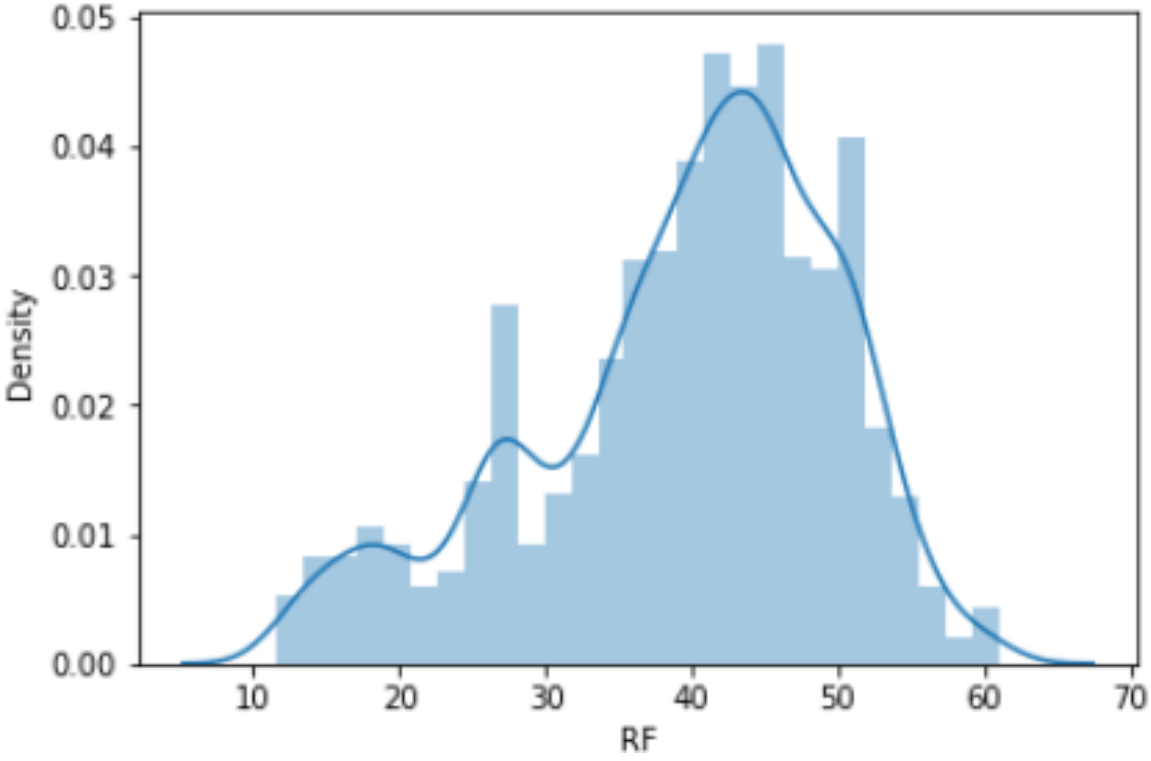
********The data distribution of Cadence ********

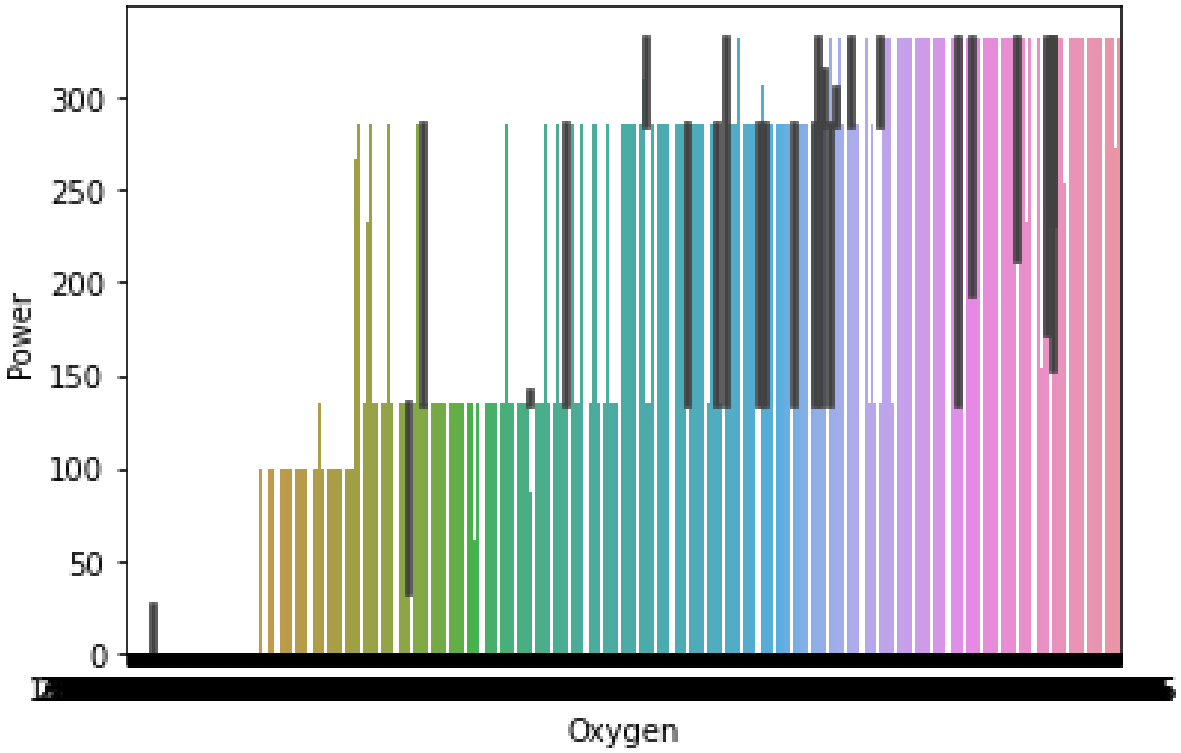


********The data distribution of HR ********



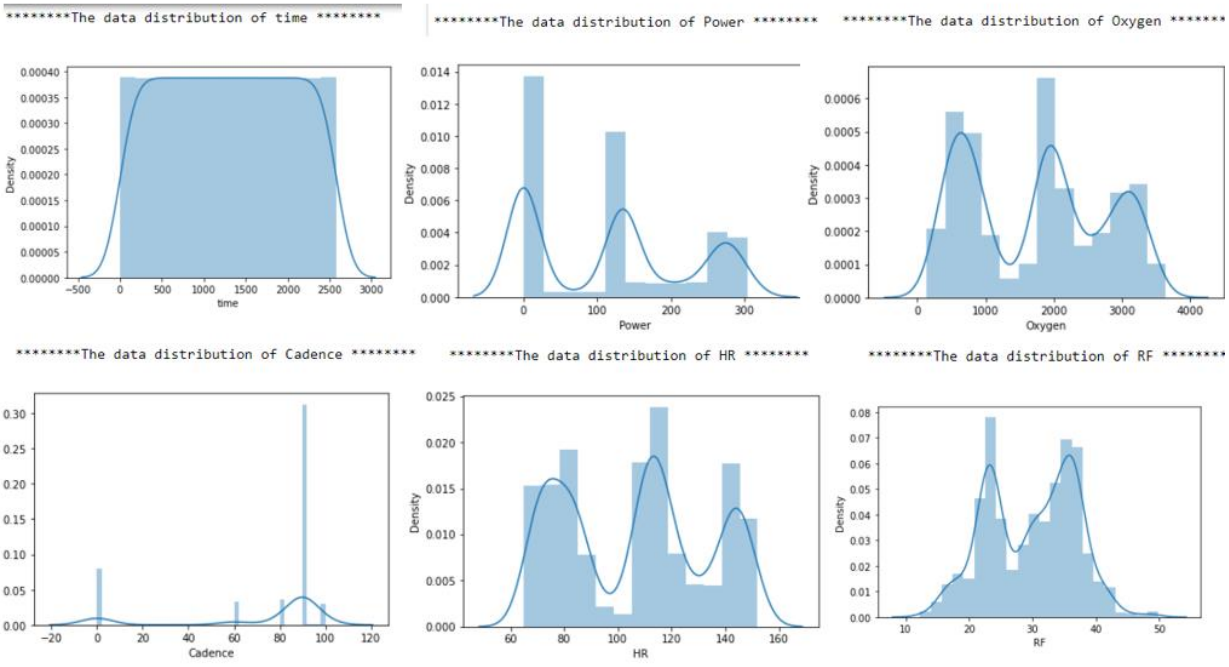
********The data distribution of RF ********





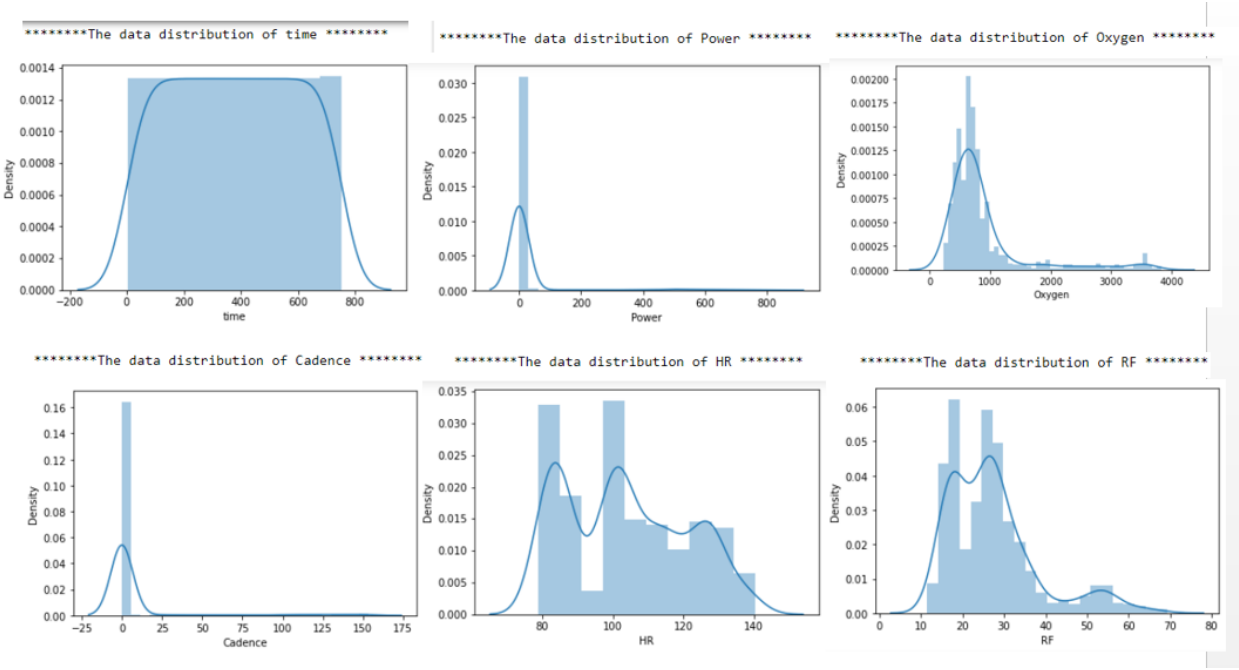
Subject-1 Trail-2 dataset

```
for var in df2.columns:

    print("*********The data distribution of", var, "*********\n")

    #plt.figure(figsize=(6, 4))
    sns.distplot(df1[var])
    plt.show()
```



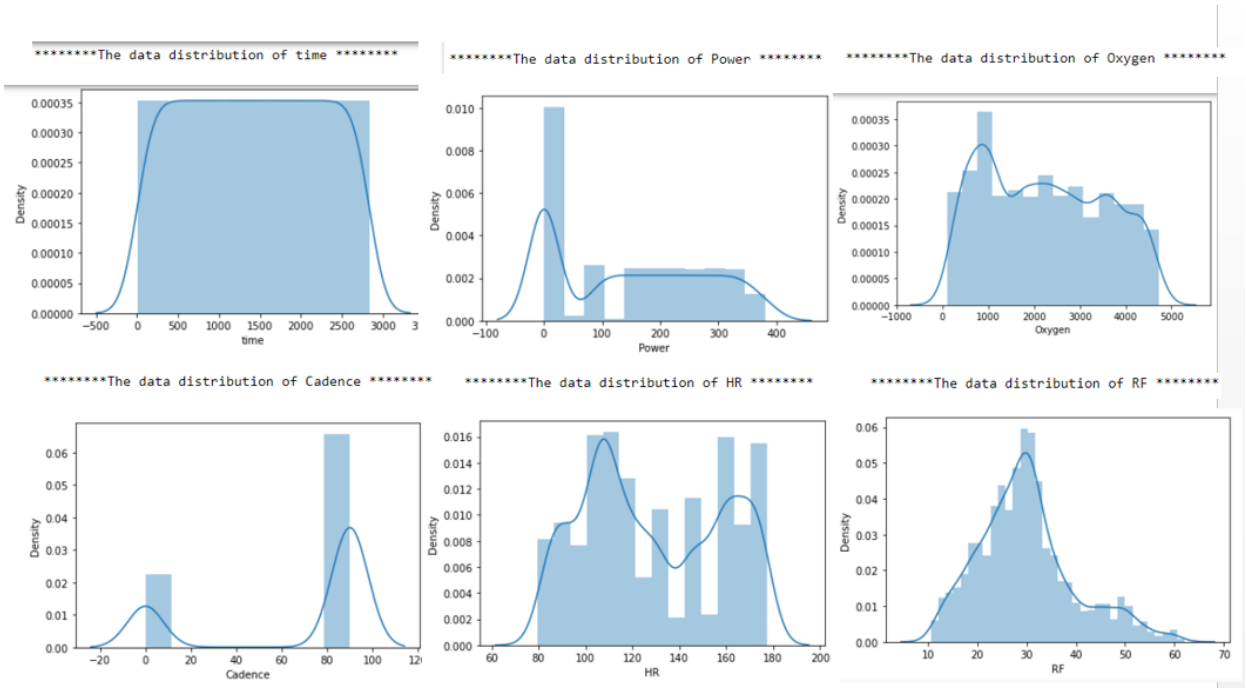
Subject-1 Wingate dataset

```
#fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
for var in df3.columns:

    print("*********The data distribution of", var, "*********\n")

    #plt.figure(figsize=(6, 4))
    sns.distplot(df3[var])
    plt.show()
```



Subject-1 Incremental Trial dataset

```
#fig, axes = plt.subplots(1, 3, figsize=(15, 5), sharey=True)
for var in df4.columns:

    print("*********The data distribution of", var, "*********\n")

    #plt.figure(figsize=(6, 4))
    sns.distplot(df4[var])
    plt.show()
```



Bivariate

Bivariate Analysis

Bivariate analysis is the analysis of two variables where two variables are analyzed to explore the relationship/association between them. Various inferential statistics can be used to perform Bivariate Analysis. Bivariate Analysis are of the following types-

- **Bivariate Analysis of two Numerical Variables (Numerical-Numerical)**
- Bivariate Analysis of two categorical Variables (Categorical-Categorical)
- Bivariate Analysis of one numerical and one categorical variable (Numerical-Categorical)

In our case **Numerical-Numerical bivariate analysis will be applicable.**

Numerical-Numerical

Inferential Statistics such as Correlation Coefficient can be used to explore two numerical variables. Visualization techniques such as Scatterplot can be used.

Correlation Plots

Correlation matrix among features and target variable

```
]: # Correlation matrix (it can be viewed in excel for better visibility)
cor1 = df1.corr()
print (cor1)
print("********************\n")

cor2 = df2.corr()
print (cor2)
print("********************\n")

cor3 = df3.corr()
print (cor3)
print("********************\n")

cor4 = df4.corr()
print (cor4)
print("********************\n")
```

```
              time     Power    Oxygen   Cadence        HR        RF
time      1.000000 -0.045991  0.123440 -0.053016  0.457707  0.485523
Power    -0.045991  1.000000  0.569862  0.973656  0.415951  0.572445
Oxygen    0.123440  0.569862  1.000000  0.538260  0.720907  0.777208
Cadence  -0.053016  0.973656  0.538260  1.000000  0.437869  0.568568
HR        0.457707  0.415951  0.720907  0.437869  1.000000  0.736971
RF        0.485523  0.572445  0.777208  0.568568  0.736971  1.000000
********************

              time     Power    Oxygen   Cadence        HR        RF
time      1.000000  0.294797  0.363032 -0.261850  0.555785  0.730681
Power     0.294797  1.000000  0.958823  0.670244  0.895318  0.597352
Oxygen    0.363032  0.958823  1.000000  0.646556  0.949023  0.699814
Cadence  -0.261850  0.670244  0.646556  1.000000  0.450467  0.134131
HR        0.555785  0.895318  0.949023  0.450467  1.000000  0.768625
RF        0.730681  0.597352  0.699814  0.134131  0.768625  1.000000
********************
```

	time	Power	Oxygen	Cadence	HR	RF
time	1.000000	0.111378	0.231926	0.092603	0.423875	0.622663
Power	0.111378	1.000000	0.922984	0.694147	0.834230	0.692100
Oxygen	0.231926	0.922984	1.000000	0.719498	0.955023	0.829046
Cadence	0.092603	0.694147	0.719498	1.000000	0.665395	0.639121
HR	0.423875	0.834230	0.955023	0.665395	1.000000	0.893650
RF	0.622663	0.692100	0.829046	0.639121	0.893650	1.000000

****

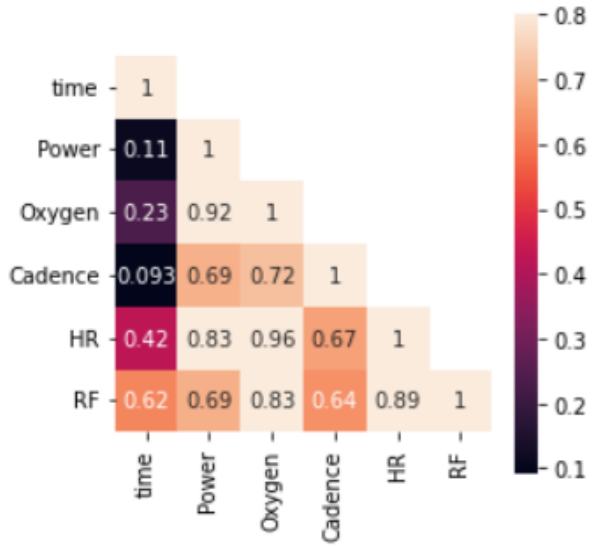
	time	Power	Oxygen	Cadence	HR	RF
time	1.000000	0.466133	0.524447	0.180028	0.595904	0.701075
Power	0.466133	1.000000	0.948873	0.555516	0.939592	0.760984
Oxygen	0.524447	0.948873	1.000000	0.605040	0.984718	0.826076
Cadence	0.180028	0.555516	0.605040	1.000000	0.574006	0.496814
HR	0.595904	0.939592	0.984718	0.574006	1.000000	0.837396
RF	0.701075	0.760984	0.826076	0.496814	0.837396	1.000000

****

Heatmap visualization of correlation matrix

```
]: # heatmap
   mask1 = np.array(cor1)
   mask1[np.tril_indices_from(mask1)] = False
   fig,ax= plt.subplots()
   fig.set_size_inches(4,4)
   sns.heatmap(cor1, mask=mask1,vmax=.8, square=True,annot=True)
```

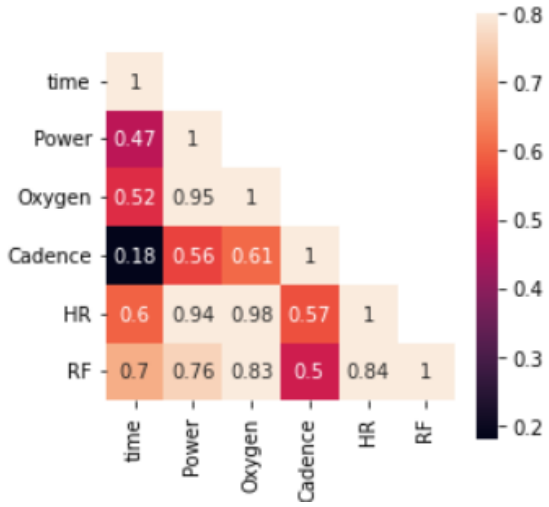
```
]: <AxesSubplot:>
```



Target variable Power is highly correlated with RF, HR, Oxygen, Cadence. Among features HR & Oxygen, RF & Oxygen, HR & RF are highly correlated.

```
# heatmap
mask2 = np.array(cor2)
mask2[np.tril_indices_from(mask2)] = False
fig,ax= plt.subplots()
fig.set_size_inches(4,4)
sns.heatmap(cor2, mask=mask2,vmax=.8, square=True,annot=True)
```

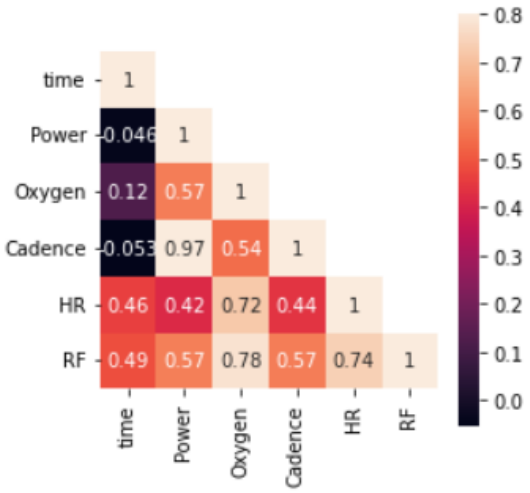
```
<AxesSubplot:>
```



Target variable Power is highly correlated with RF, HR, Oxygen, Cadence. Among features HR & Oxygen , RF & Oxygen, HR & RF are highly correlated.

```
mask3 = np.array(cor3)
mask3[np.tril_indices_from(mask3)] = False
fig,ax= plt.subplots()
fig.set_size_inches(4,4)
sns.heatmap(cor3, mask=mask3,vmax=.8, square=True,annot=True)
```

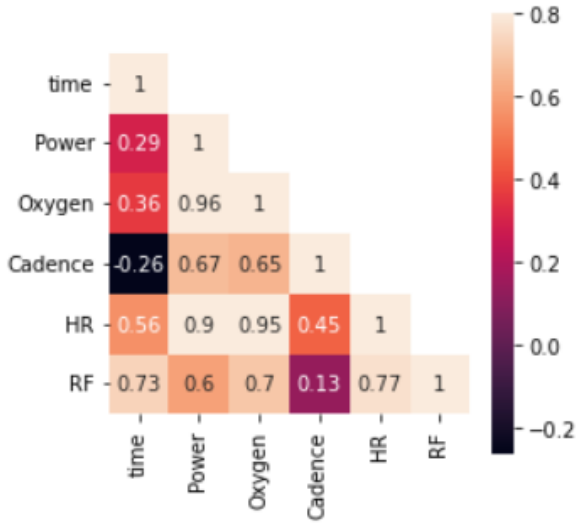
```
<AxesSubplot:>
```



Target variable Power is highly correlated with Cadence. Among features HR & Oxygen, RF & Oxygen, HR & RF are highly correlated.

```
# heatmap
mask4 = np.array(cor4)
mask4[np.tril_indices_from(mask4)] = False
fig,ax= plt.subplots()
fig.set_size_inches(4,4)
sns.heatmap(cor4, mask=mask4,vmax=.8, square=True,annot=True)
```

```
<AxesSubplot:>
```

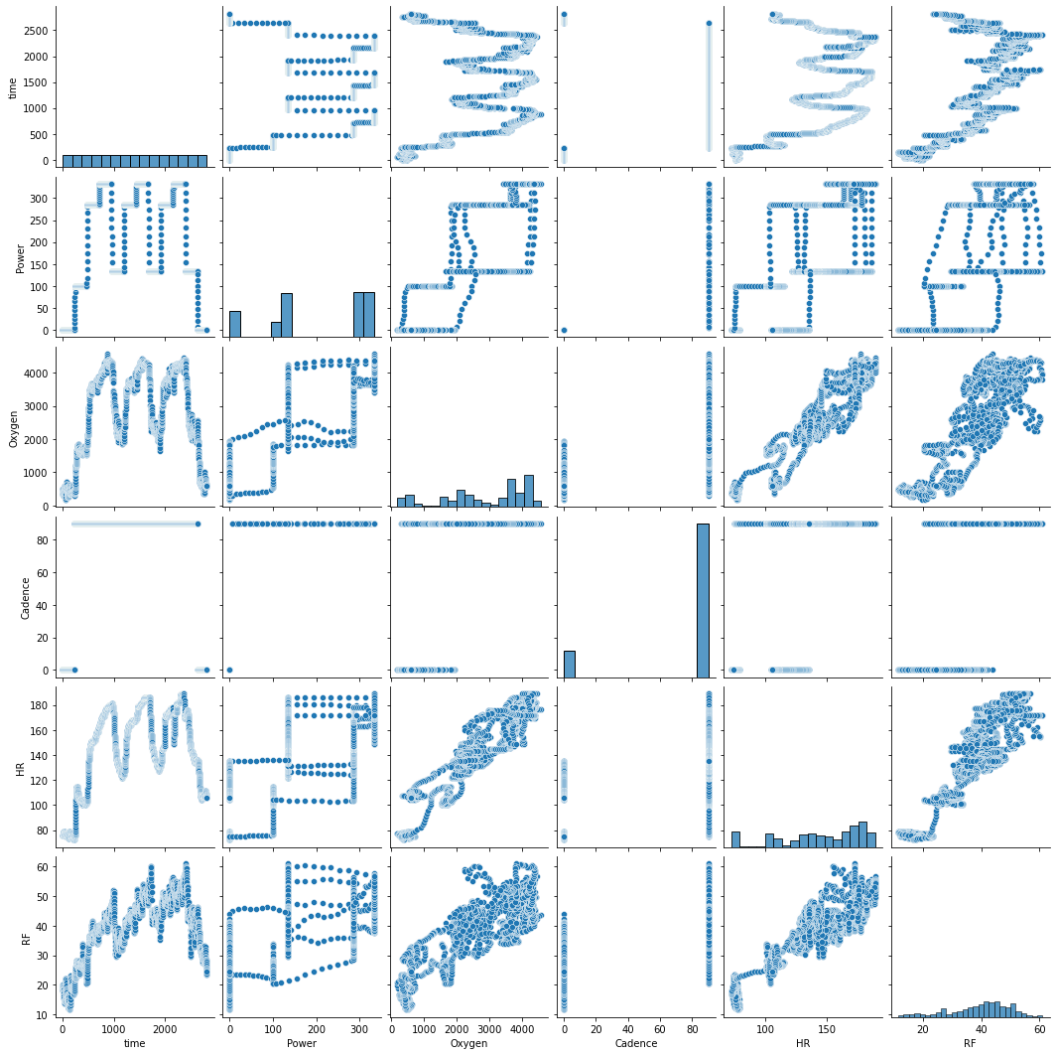


Target variable Power is highly correlated with HR, Oxygen, Cadence. Among features HR & Oxygen , HR & RF , Oxygen & RF are highly correlated.

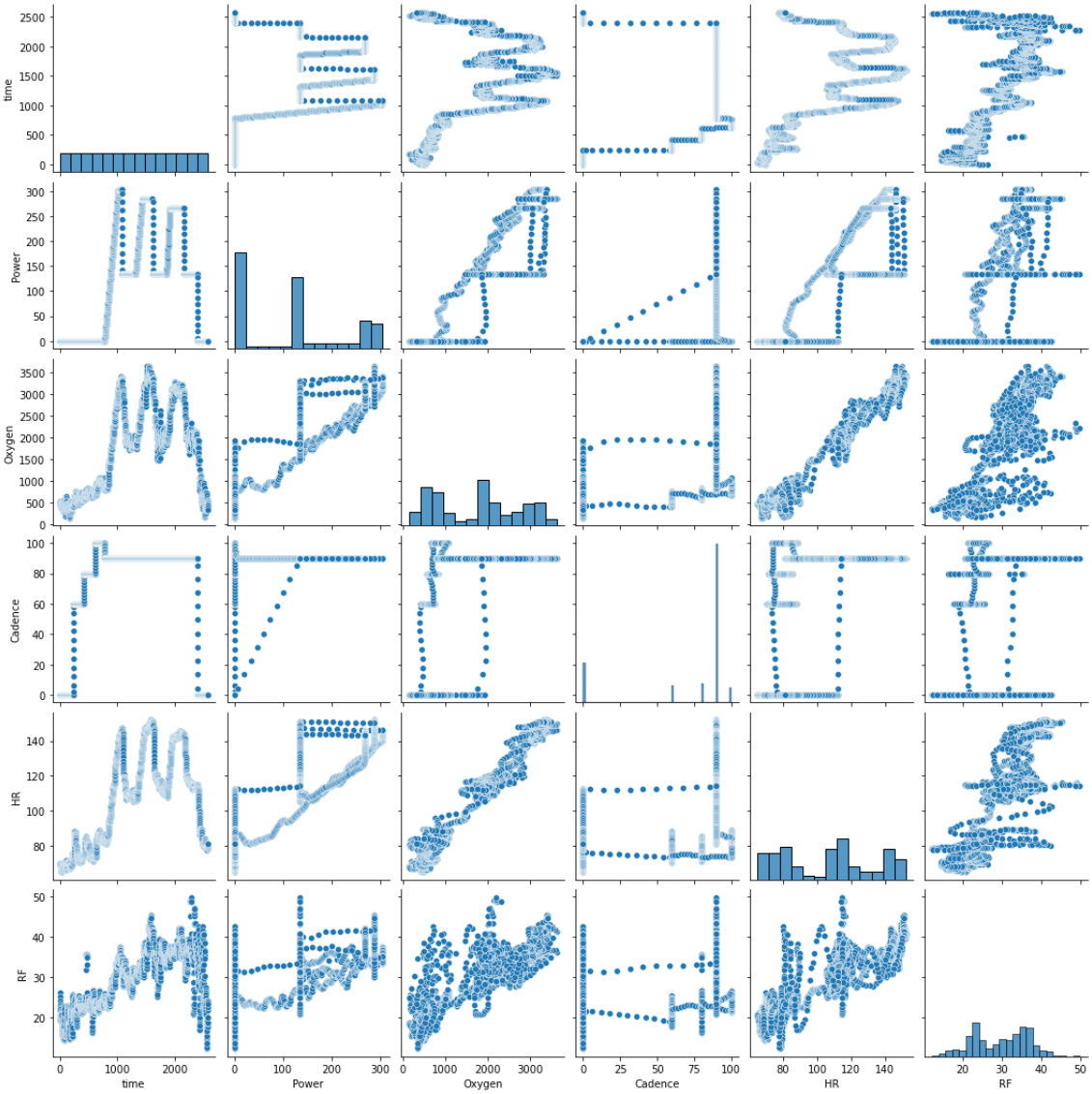
Visualization with Pairplot

Pairplot to visualize the feature correlations in subject-1 dataset for trtal-1, trial-2, Wingate & incremental training datasets.

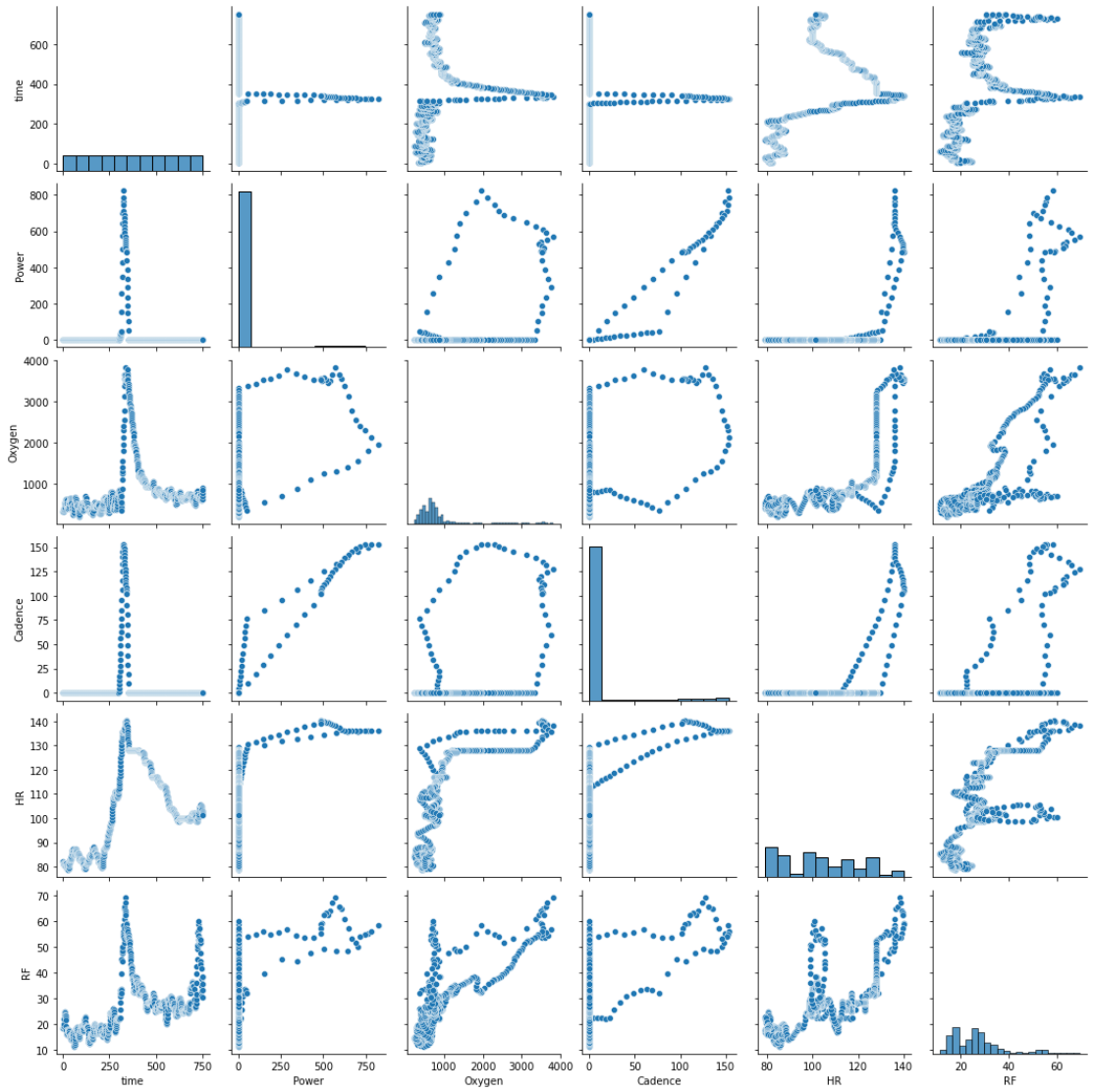
```
# Pairwise scatter plot between any two features
sns.pairplot(df1)
plt.show()
```



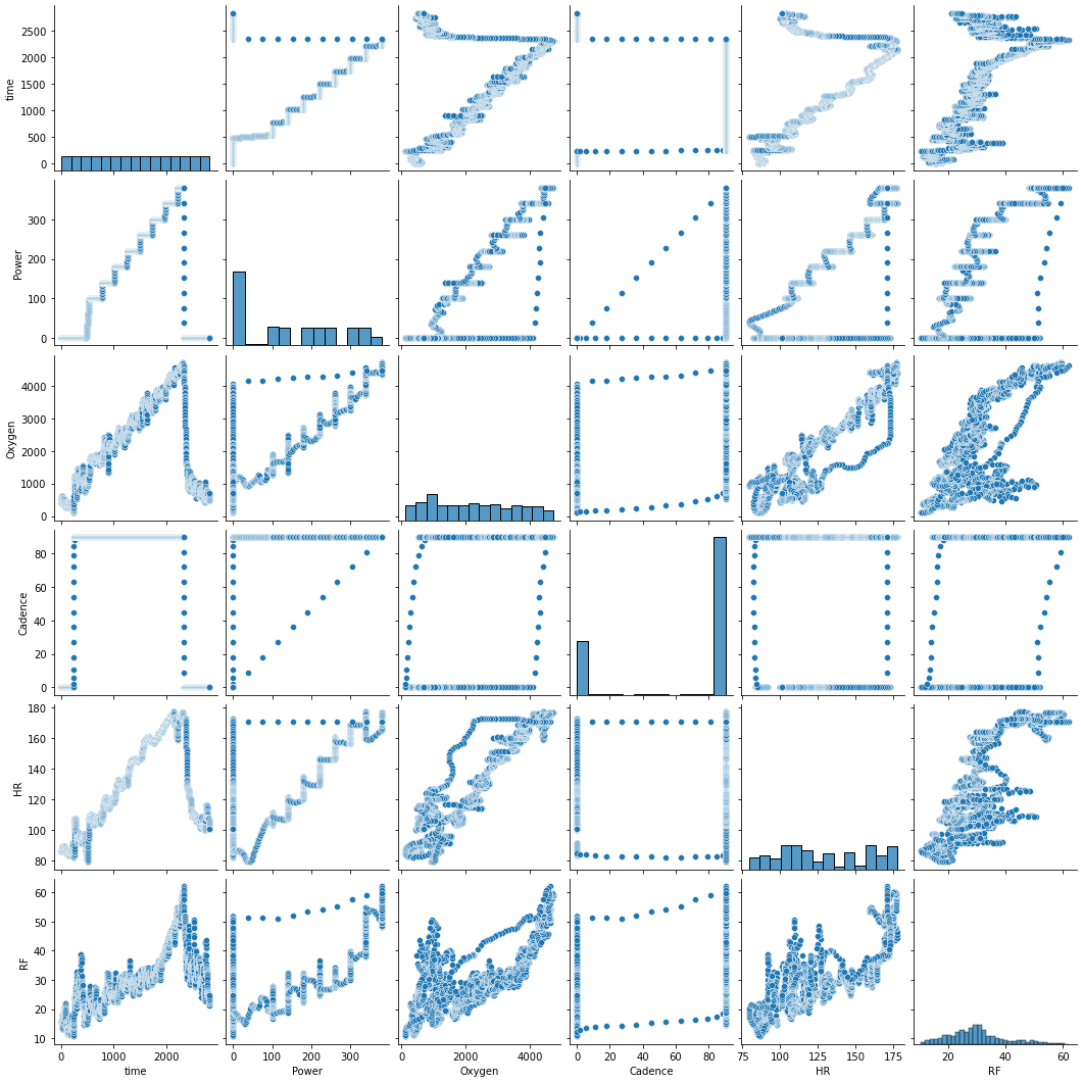
```
# Pairwise scatter plot between any two features
sns.pairplot(df2)
plt.show()
```



```
# Pairwise scatter plot between any two features
sns.pairplot(df3)
plt.show()
```



```
# Pairwise scatter plot between any two features
sns.pairplot(df4)
plt.show()
```



Data Cleanup

In [7]:
```
# inspect the structure etc.
print(df1.info(), "\n")
print(df1.shape)
print("***********\n")

print(df2.info(), "\n")
print(df2.shape)
print("***********\n")

print(df3.info(), "\n")
print(df3.shape)
print("***********\n")

print(df4.info(), "\n")
print(df4.shape)
print("***********\n")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2819 entries, 0 to 2818
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     2819 non-null   int64  
 1   Power    2819 non-null   float64
 2   Oxygen   2819 non-null   float64
 3   Cadence  2819 non-null   int64  
 4   HR       2819 non-null   float64
 5   RF       2819 non-null   float64
dtypes: float64(4), int64(2)
memory usage: 132.3 KB
None

(2819, 6)
***********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2579 entries, 0 to 2578
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     2579 non-null   int64  
 1   Power    2579 non-null   float64
 2   Oxygen   2579 non-null   float64
 3   Cadence  2579 non-null   float64
 4   HR       2579 non-null   float64
 5   RF       2579 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 121.0 KB
None

(2579, 6)
**********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 751 entries, 0 to 750
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     751 non-null    int64  
 1   Power    751 non-null    float64
 2   Oxygen   751 non-null    float64
 3   Cadence  751 non-null    float64
 4   HR       751 non-null    float64
 5   RF       751 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 35.3 KB
None

(751, 6)
**********
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2831 entries, 0 to 2830
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   time     2831 non-null   int64  
 1   Power    2831 non-null   float64
 2   Oxygen   2831 non-null   float64
 3   Cadence  2831 non-null   float64
 4   HR       2831 non-null   float64
 5   RF       2831 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 132.8 KB
None

(2831, 6)
**********
```

As per the above Data information there is no missing data in any of the datasets.

Treat Outlier

Handling Outlier

```python
# Treatment of outliers - function to calculate interquartile range
from numpy.random import randn
from numpy import percentile

def outlier_treatment(datacolumn):

    q25, q75, q50 = percentile(datacolumn, 25), percentile(datacolumn, 75), percentile(datacolumn, 50)
    iqr = q75 - q25

    # calculate the outlier cutoff
    cut_off = iqr * 1.5
    lower, upper = q25 - cut_off, q75 + cut_off
    return iqr, lower, upper, q25, q75, q50
```

```
#Treat variables for outliers - SalePrice
iqr, lower, upper, q25, q75, q50 = outlier_treatment(df1["Power"])

print(iqr)
print(lower)
print(upper)

Upper_outliers =df1.loc[df1['Power'] > upper]
Lower_outliers =df1.loc[df1['Power'] < lower]
print(" Number of rows present above upper cutoff : ", Upper_outliers.shape)
print(" Number of rows present above upper cutoff : ",Lower_outliers.shape)
print('Percentiles: 25th=%.3f, 75th=%.3f, 50th=%.3f, IQR=%.3f' % (q25, q75, q50, iqr))
```

```
179.10000000000002
-134.65000000000003
581.75
 Number of rows present above upper cutoff :  (0, 6)
 Number of rows present above upper cutoff :  (0, 6)
Percentiles: 25th=134.000, 75th=313.100, 50th=286.000, IQR=179.100
```

As per the code there is no outlier to treat.

Building Model-1 (NN MLPRegressor)

Algorithm Selection

The established method is that oxygen supply is the input and power is the output. But assessing the current fitness of the athlete, even though it can certainly be related, is not the same as predicting his fitness on the day of the event. Hence new study is proposing a model which can take into account both the capacity as well as the fatigue of the athlete on that day.

The relationship between steady-state power and Oxygen under carefully controlled conditions is proven to be a good indicator of fitness. However, we cannot ignore the time dependencies. The time constant is less dependent on the type of training being undertaken (intervals, sustained efforts, long rides, etc) than the absolute value of heart rate required for a given power.

Measurement of oxygen uptake is expensive and not everyone can afford to have such expensive tools. The purpose of this POC is to study this new algorithm that can predict the oxygen uptake for cyclists using an alternative method. Racing cyclists need personalized optimized fitness training. The proposed algorithm is focused to achieve a discrete-event non-linear model where one of the fitness attributes - “power” is used in a different way. **Power is treated as input** which will control the body to respond accordingly; rather than as the output which is a normal technique used in most of the algorithms. **Oxygen uptake will be measured as Target variable.**

In next few weeks we will develop various models to explore these dynamic dependency factors. This approach looks promising to capture the current state of the body by looking closely at how the latter responds to the varying stress the athlete exerts on it

This week we have built our first model with this concept. Let’s explore this in the next few sections -
Oxygen is numeric continuous number and labeled variable. Hence this is Supervised Regression Problem.

We are selecting our first **model Neural Network - Multi-layer Perceptron Regressor (MLPRegressor**)

Loading the ML Framework

Neural Network - Multi-layer Perceptron Regressor (MLPRegressor)

A Multi-layer Perceptron (MLP) Regression System is a multilayer feedforward neural network training system that implements multi-layer perceptron regression algorithm to solve a Multi-layer Perceptron Regression Task.

This is scikit-learn model. Here is the scikit-learn framework required for this model –

```
#Loading the libraries

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.neural_network import MLPRegressor

from sklearn import metrics

from sklearn.model_selection import GridSearchCV
```

Creating the Training DC

The Oxygen is set as target variable. Training vs Test data split is kept as 80:20. The training dataset features are standardized using sklearn.preprocessing.StandardScaler which is a pre processing technique. Standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

```
#Creating Training data - "Oxygen" is the Target variable

x = df1.drop('Oxygen', axis=1)
y = df1['Oxygen']

trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)
```

```
#Standardize the training data - only feature set

sc=StandardScaler()

scaler = sc.fit(trainX)
trainX_scaled = scaler.transform(trainX)
testX_scaled = scaler.transform(testX)
```

Building the model

Class MLPRegressor implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values.

```
#Building the NN MLPRegressor Model & set the hyperparameters with default value
mlp_reg = MLPRegressor(hidden_layer_sizes=(150,100,50),
                       max_iter = 300,activation = 'relu',
                       solver = 'adam')

#Fit the model with training data
mlp_reg.fit(trainX_scaled, trainY)
```

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (300) reached and the optimization hasn't converged yet.
  warnings.warn(
```

```
MLPRegressor(hidden_layer_sizes=(150, 100, 50), max_iter=300)
```

The hidden layer parameters are tuning parameters and will be used for model optimization. Activation layer is “Relu” as it is a repressor model.

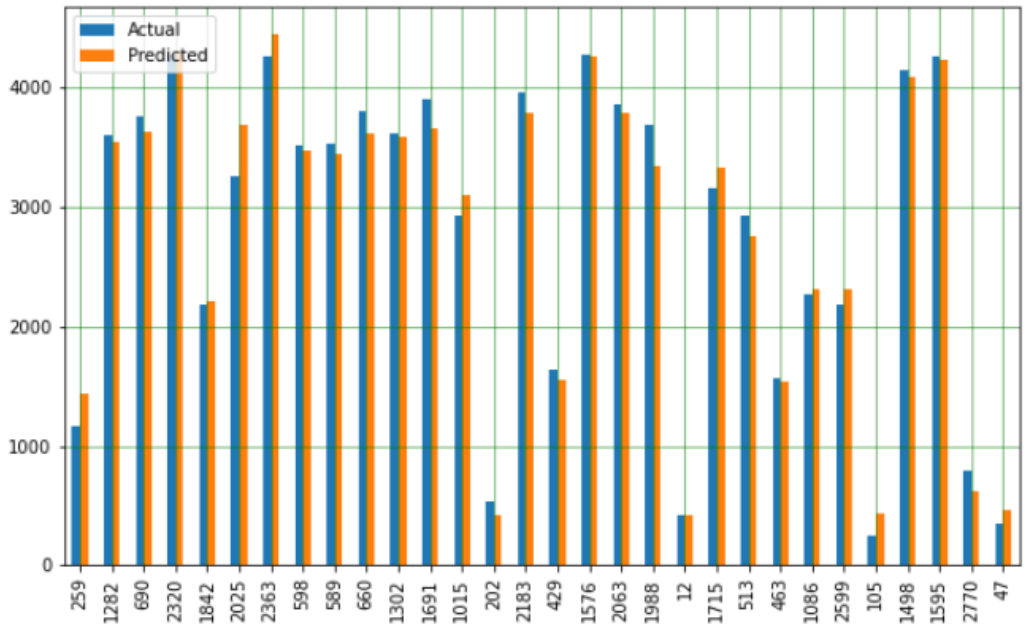
Model evaluation

```
#Model Evaluation using test set.
y_pred = mlp_reg.predict(testX_scaled)
```

```
#creating dataframe to display the Predicted value against Actual value
df_temp = pd.DataFrame({'Actual': testY, 'Predicted': y_pred})
df_temp.head()
```

	Actual	Predicted
259	1168.70	1442.468154
1282	3603.10	3538.180706
690	3760.90	3636.291973
2320	4273.05	4306.554596
1842	2184.00	2217.496078

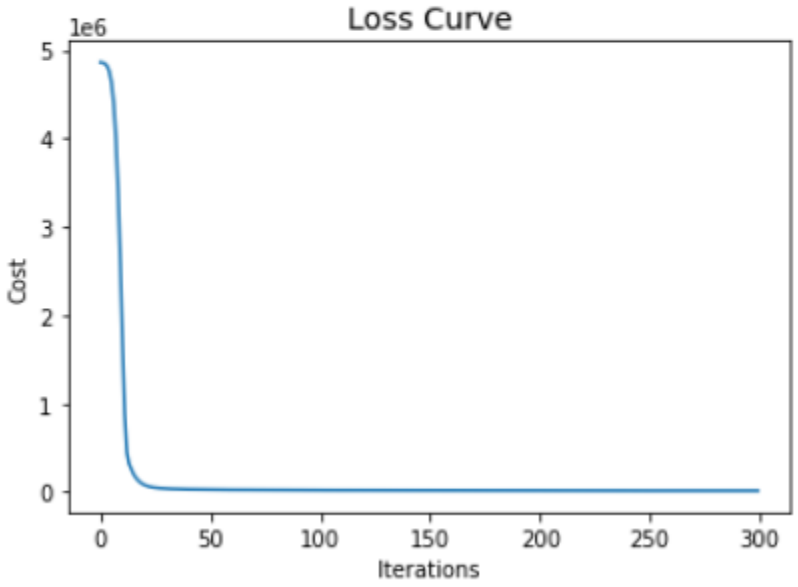
```
df_temp = df_temp.head(30)
df_temp.plot(kind='bar',figsize=(10,6))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```




```
#Calculating the Model KPIs error values

print('Mean Absolute Error:', metrics.mean_absolute_error(testY, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(testY, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(testY, y_pred)))
```

```
Mean Absolute Error: 127.10558763306469
Mean Squared Error: 31071.99087138977
Root Mean Squared Error: 176.27249039878507
```

```
#Visualize the loss factor

plt.plot(mlp_reg.loss_curve_)
plt.title("Loss Curve", fontsize=14)
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.show()
```



Model Optimization

GridSearchCV is a function that comes in Scikit-learn's model_selection package. This function helps to loop through predefined hyperparameters and fit estimator (model) on the training set.

```
#Trying to tune the hyparameters to improve the mmodel output by reducing the loss

param_grid = {
    'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,50,30)],
    'max_iter': [50, 100],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive'],
}
```

```
#Display the best parameters & build the best fit model with these parameters

grid = GridSearchCV(mlp_reg, param_grid, n_jobs= -1, cv=5)
grid.fit(trainX_scaled, trainY)

print(grid.best_params_)
```

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:922: UserWarning: One or more of the test scores
are non-finite: [ 0.73370423 -4.92359638  0.61975303 -4.83505809  0.76377651 -4.92426559
  0.72810921 -4.83486582  0.78860489 -4.94674432  0.82343735 -4.87120369
  0.77129253 -4.94461035  0.8068822  -4.87180238  0.5963047  -4.96568572
  0.60227543 -4.91137823  0.54448381 -4.96563557  0.87271165 -4.90905326
  0.72059925 -4.92484542  0.71814979 -4.83563266  0.89382448 -4.92482983
  0.75507092 -4.83457279  0.86485036 -4.94287101  0.70413001 -4.87264996
```

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:922: UserWarning: One or more of the test scores
are non-finite: [ 0.73370423 -4.92359638  0.61975303 -4.83505809  0.76377651 -4.92426559
  0.72810921 -4.83486582  0.78860489 -4.94674432  0.82343735 -4.87120369
  0.77129253 -4.94461035  0.8068822  -4.87180238  0.5963047  -4.96568572
  0.60227543 -4.91137823  0.54448381 -4.96563557  0.87271165 -4.90905326
  0.72059925 -4.92484542  0.71814979 -4.83563266  0.89382448 -4.92482983
  0.75507092 -4.83457279  0.86485036 -4.94287101  0.70413001 -4.87264996
  0.72630336 -4.94538866  0.88608022 -4.87123285  0.53980332 -4.96587418
  0.46011397 -4.9109074   0.67332269 -4.96434443  0.67269248 -4.91311983
         nan  0.9637869          nan  0.97228313         nan  0.96414748
         nan  0.97396166         nan  0.95940863         nan  0.97184022
         nan  0.96003616         nan  0.97224345         nan  0.95356272
         nan  0.97085276         nan  0.95218045         nan  0.96984708
         nan  0.96429651         nan  0.97296608         nan  0.96451251
         nan  0.97368942         nan  0.95983088         nan  0.97230952
         nan  0.95959436         nan  0.9715934          nan  0.95268917
         nan  0.97005221         nan  0.95561666         nan  0.96919092]
  warnings.warn(
```

```
{'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (150, 100, 50), 'learning_rate': 'adaptive', 'max_iter': 100, 'so
lver': 'adam'}
```

```
C:\Users\admin\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:614: ConvergenceWarning: Stochastic
Optimizer: Maximum iterations (100) reached and the optimization hasn't converged yet.
  warnings.warn(
```

The best parameters selected by hyper parameter optimization technique GridSearchCV and displayed above.

Let's use them and build our first best fit model.

Best Fit Model

We will repeat the above Model evaluation techniques using the best parameters selected by GridSearchCV.

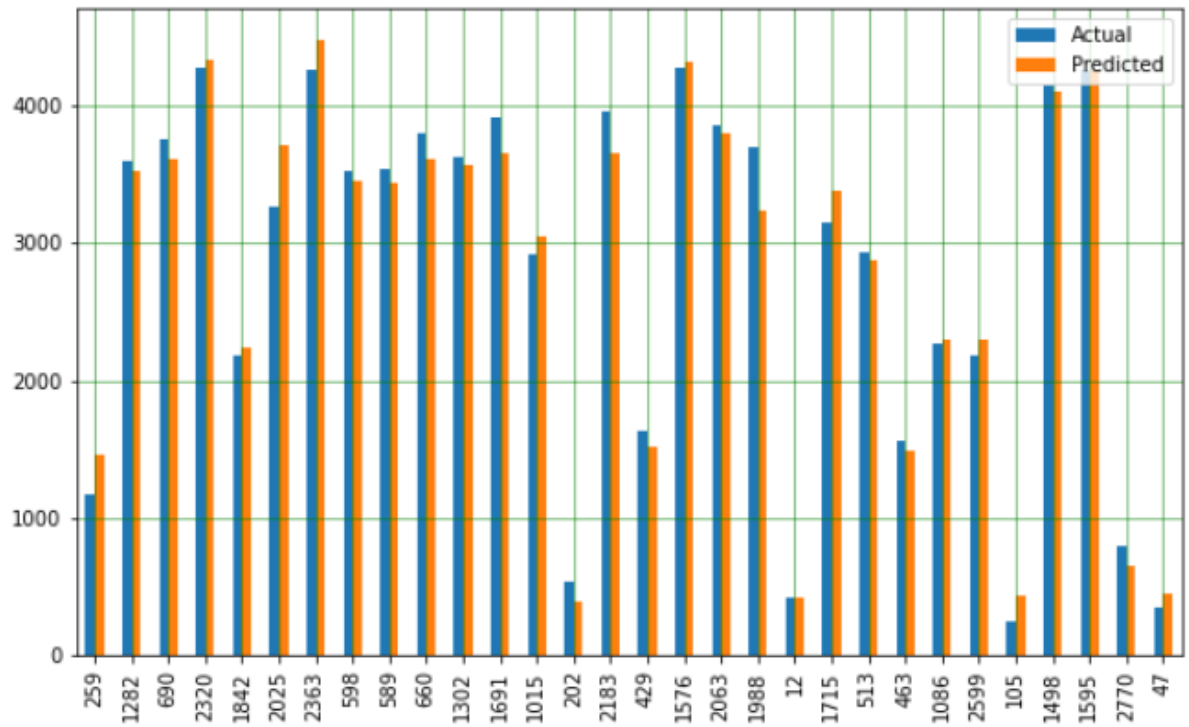
We will also calculate the Model KPIs.

```
#Now predict the test set with the best fit model

grid_predictions = grid.predict(testX_scaled)
```

```
#Dislay the predicted value against the Actual value
df_temp2 = pd.DataFrame({'Actual': testY, 'Predicted': grid_predictions})
df_temp2.head()
```

	Actual	Predicted
259	1168.70	1463.848511
1282	3603.10	3531.446868
690	3760.90	3604.997313
2320	4273.05	4334.337153
1842	2184.00	2235.385618

```
#Visualize the prediction vs Actual value of Oxygen
df_temp2 = df_temp2.head(30)
df_temp2.plot(kind='bar',figsize=(10,6))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```




```
: #Calculating the model KPIs - loss factors

print('Mean Absolute Error:', metrics.mean_absolute_error(testY, grid_predictions))
print('Mean Squared Error:', metrics.mean_squared_error(testY, grid_predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(testY, grid_predictions)))
```

```
Mean Absolute Error: 135.76009992080805
Mean Squared Error: 33930.301702359495
Root Mean Squared Error: 184.2017961431416
```

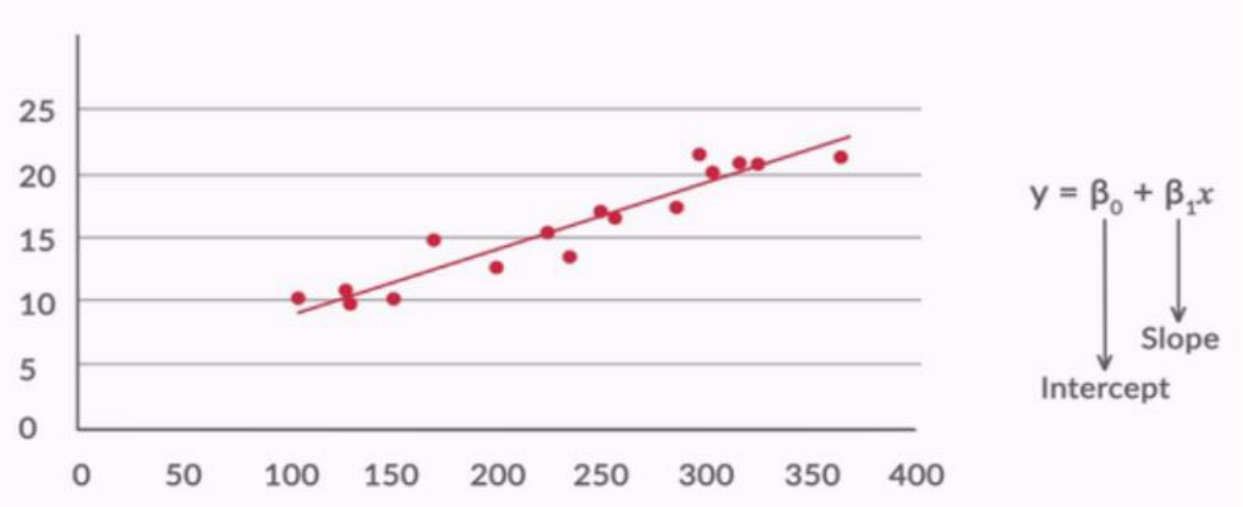
As per the result above we have built a descent model so far. We will keep exploring the other Repressor models in coming weeks.

Building Model-2: Linear Regression

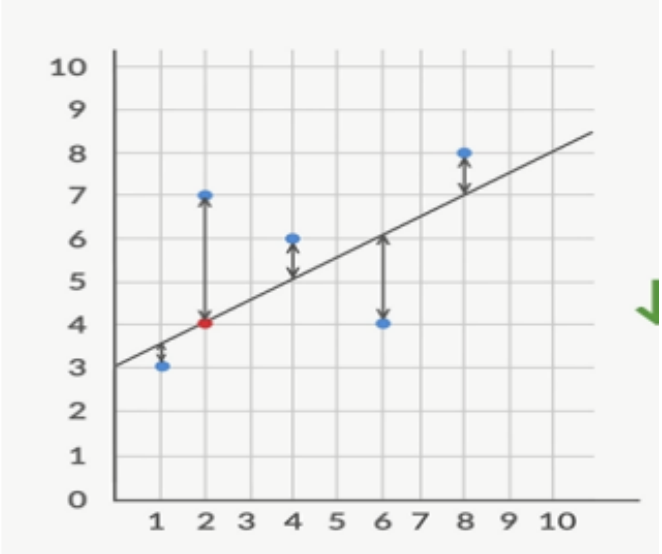
Linear Regression: As per the pair plot output we can see some kind of linear relation between Oxygen and other predictors like power, RF etc. Hence we will evaluate the Linear Regression model.

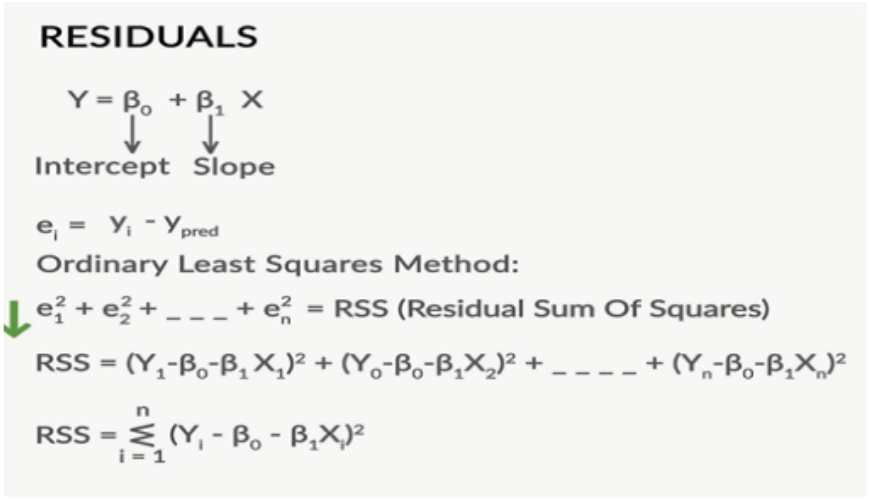
The most elementary type of regression model is the simple linear regression which explains the relationship between a dependent variable and one independent variable using a straight line. The straight line is plotted on the scatter plot of these two points.

The below is a Linear Regression diagram.



The best-fit line is found by minimizing the expression of RSS (Residual Sum of Squares) which is equal to the sum of squares of the residual for each data point in the plot. Residuals for any data point is found by subtracting predicted value of dependent variable from actual value of dependent variable





Loading the ML Framework

Load the LinearRegression library from scikit learn framework. Then we will instantiate LinearRegression model in this code.

Then fit the training data what we have already created for our previous model -NN MLPRegressor. Once the model is built with training data, the coefficients and intercept values will be calculated with lm.coef_ and lm.intercept_.

These steps are shown in the below codes:

```
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
```

```
# Instantiate
lm = LinearRegression()

# Fit a line
lm.fit(trainX_scaled, trainY)
```

```
LinearRegression()
```

```
# Print the coefficients and intercept
print(lm.coef_)
print(lm.intercept_)
```

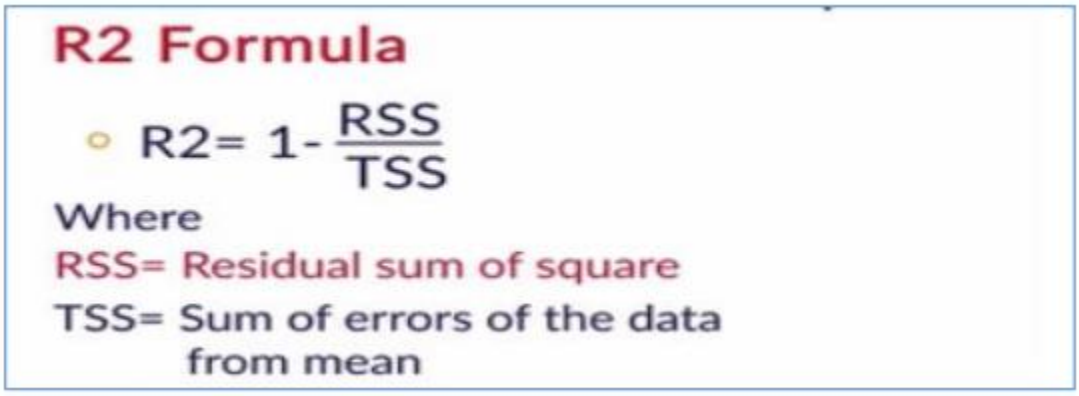
```
[-211.82299112  398.31990877    5.36455173  779.72804383  217.84724977]
2811.296478219911
```

Model evaluation

Model is evaluated by:

1. **R 2 or Coefficient of Determination**

   You also learnt an alternative way of checking the accuracy of your model, which is R2 statistics. R2 is a number which explains what portion of the given data variation is explained by the developed model. It always takes a value between 0 & 1. In general term, it provides a measure of how well actual outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model, i.e. expected outcomes. Overall, the higher the R-squared, the better the model fits your data. Mathematically, it is represented as: R 2 = 1 - (RSS / TSS)



   RSS (Residual Sum of Squares): In statistics, it is defined as the total sum of error across the whole sample. It is the measure of the difference between the expected and the actual output. A small RSS indicates a tight fit of the model to the data. It is also defined as follows:

$$RSS = \sum_{i=1}^{n} (y_i - (\alpha + \beta x_i))^2$$

   TSS (Total sum of squares): It is the sum of errors of the data points from mean of response variable. Mathematically, TSS is:

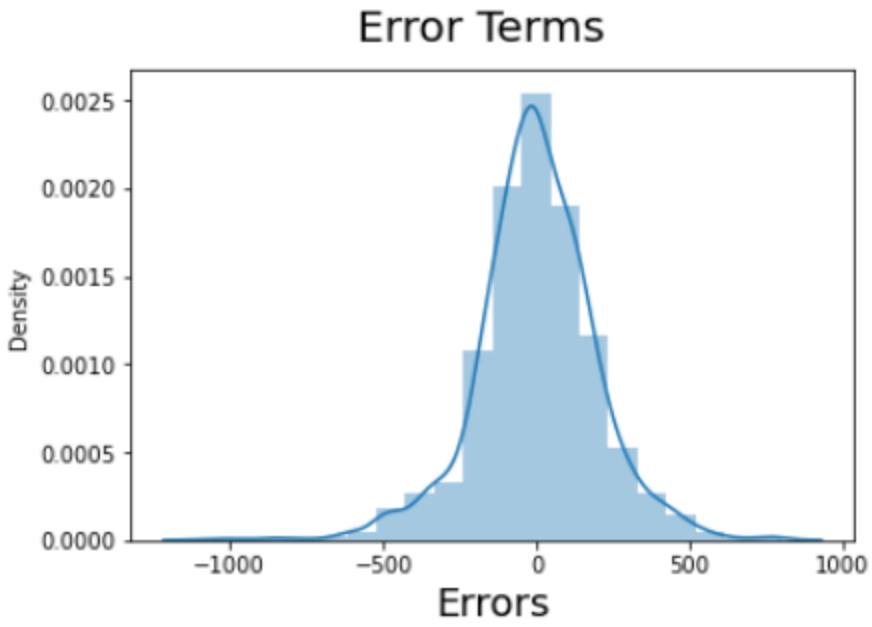
$$\mathrm{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

2. **Residual Standard Error (RSE)**

Residual Analysis

```
y_train = lm.predict(trainX_scaled)
```

```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((trainY - y_train), bins = 20)
# Plot heading
fig.suptitle('Error Terms', fontsize = 20)
# Give the X-label
plt.xlabel('Errors', fontsize = 18)
```



Prediction

In this model, the focus was more on the prediction of future results by using linear regression concepts. Broadly speaking, it is a form of predictive modelling technique which tells us the relationship between the "**Oxygen**" (target variable) and independent variables (predictors) like "**Power", "RF**" etc.

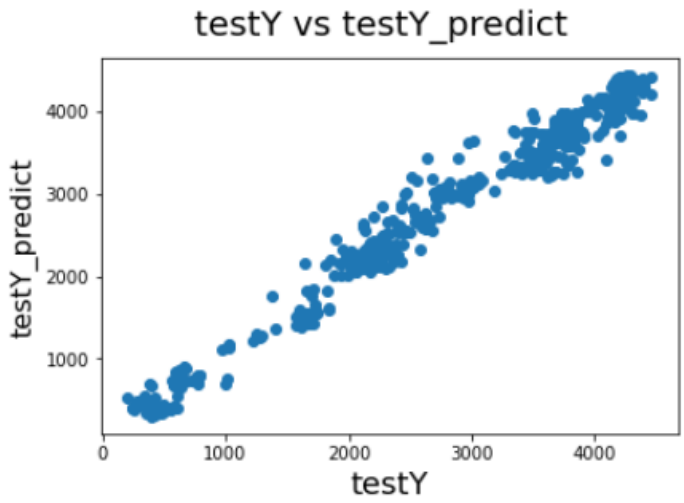
```
#trainX, testX, trainY, testY = train_test_split(x, y, test_size = 0.2)
```

```
testY_predict = lm.predict(testX_scaled)
```

- In statistical modelling, linear regression is a process of estimating the relationship among variables. The focus here is to establish the relationship between a dependent variable and one or more independent variable(s). Independent variables are also called 'predictors'.
- Regression helps you understand how the values of dependent variable changes as you change the values of 1 predictor, holding the other predictors static (or same). This means that simple linear regression in its most basic form doesn't allow you to change all the predictors at a time and measure the impact on the dependent variable. You can only change 1 at a time.
- Regression only shows relationship, i.e. correlation and NOT causality. In a very restrictive environment, regression may show causality. However, if you blindly interpret regression results as causation, it may lead to false insights. Remember: Correlation does not imply causation.
- Regression analysis is widely used for 2 purposes: a) Forecasting and b) Prediction. The uses of forecasting and prediction have substantial overlap. However, they are different and it's important to understand why, to be able to use regression effectively for each purpose. Regression guarantees 'interpolation' but not necessarily 'extrapolation'.
- Linear regression is a form of parametric regression.

```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(testY, testY_predict)
fig.suptitle('testY vs testY_predict', fontsize = 20)              # Plot heading
plt.xlabel('testY', fontsize = 18)                                 # X-label
plt.ylabel('testY_predict', fontsize = 16)
```

```
Text(0, 0.5, 'testY_predict')
```



From the above plot, it's evident that the model is doing well on the test set as well. Let's also check the R-squared and more importantly, the adjusted R-squared value for the test set.

```
# r2_score for 6 variables
from sklearn.metrics import r2_score
r2_score(trainY, y_train)
```

```
0.9759770164908319
```

```
# r2_score for 6 variables
from sklearn.metrics import r2_score
r2_score(testY, testY_predict)
```

```
0.9774315983738585
```

Both the r-squared values on training and test data are 97%.
This is the simplest model that we could build. The final predictors seem to have fairly low correlations

Building Model - 3: DecisionTreeRegressor

Let's train our model using the Scikit-learn DecisionTreeRegressor class.

```
#Import the class
from sklearn.tree import DecisionTreeRegressor


#Create an object (model)
dtr1 = DecisionTreeRegressor(max_depth=2,
                             random_state=1)

#Fit (train) the model
dtr1.fit(trainX, trainY)
```

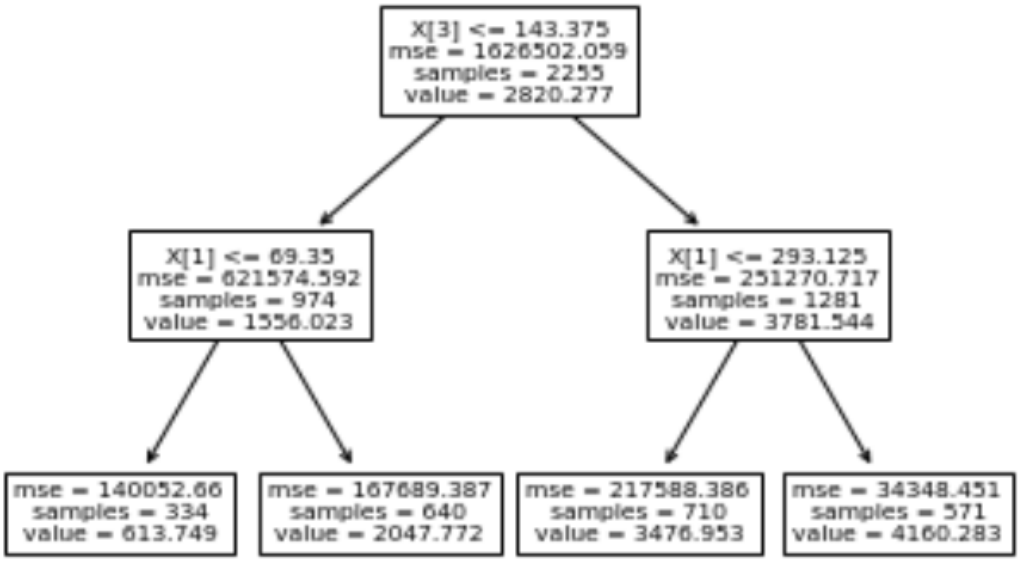
```
DecisionTreeRegressor(max_depth=2, random_state=1)
```

Now let's plot the tree

```
from sklearn import tree
```

```
tree.plot_tree(dtr1)
```

```
[Text(167.4, 181.2, 'X[3] <= 143.375\nmse = 1626502.059\nsamples = 2255\nvalue = 2820.277'),
 Text(83.7, 108.72, 'X[1] <= 69.35\nmse = 621574.592\nsamples = 974\nvalue = 1556.023'),
 Text(41.85, 36.23999999999998, 'mse = 140052.66\nsamples = 334\nvalue = 613.749'),
 Text(125.55000000000001, 36.23999999999998, 'mse = 167689.387\nsamples = 640\nvalue = 2047.772'),
 Text(251.10000000000002, 108.72, 'X[1] <= 293.125\nmse = 251270.717\nsamples = 1281\nvalue = 3781.54
4'),
 Text(209.25, 36.23999999999998, 'mse = 217588.386\nsamples = 710\nvalue = 3476.953'),
 Text(292.95, 36.23999999999998, 'mse = 34348.451\nsamples = 571\nvalue = 4160.283')]
```

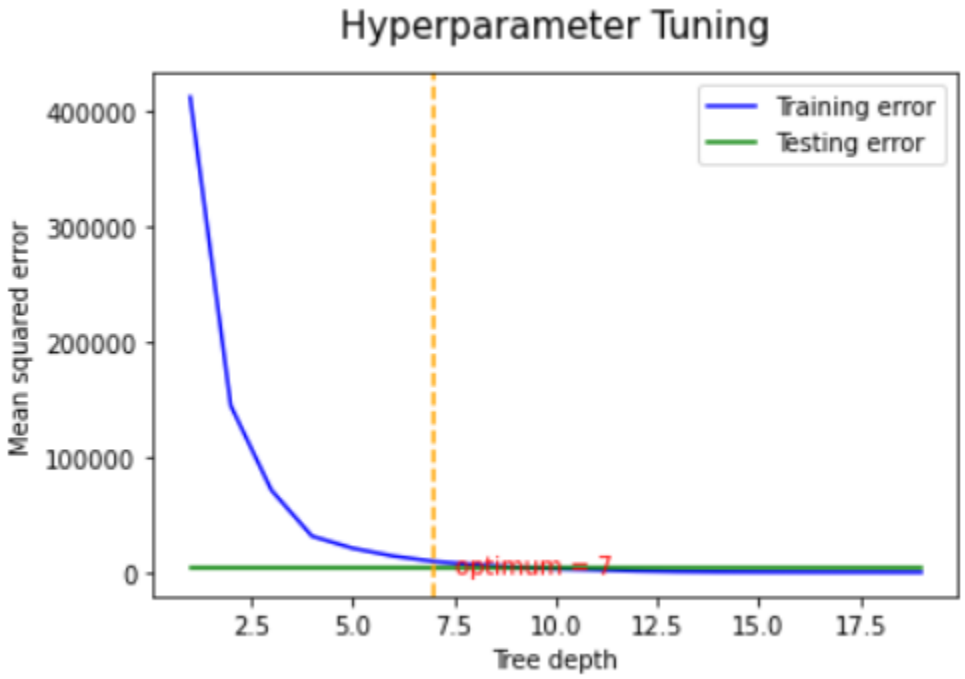


Model Optimization

Hyper parameter tuning for decision tree regression

We train the model using trainX, trainY and test it using testX, testY. This is done for different values of max_depth hyperparameter ranging from 1 to 20 and plot the testing error with the training error.

```
from sklearn.metrics import mean_squared_error as mse

max_depths = range(1, 20)
training_error = []
for max_depth in max_depths:
    model_1 = DecisionTreeRegressor(max_depth=max_depth)
    model_1.fit(trainX, trainY)
    training_error.append(mse(y, model_1.predict(x)))

testing_error = []
for max_depth in max_depths:
    model_3 = DecisionTreeRegressor(max_depth=max_depth)
    model_3.fit(trainX, trainY)
    testing_error.append(mse(testY, model_2.predict(testX)))

plt.plot(max_depths, training_error, color='blue', label='Training error')
plt.plot(max_depths, testing_error, color='green', label='Testing error')
plt.xlabel('Tree depth')
plt.axvline(x=7, color='orange', linestyle='--')
plt.annotate('optimum = 7', xy=(7.5, 1.17), color='red')
plt.ylabel('Mean squared error')
plt.title('Hyperparameter Tuning', pad=15, size=15)
plt.legend()
plt.savefig('error.png')
```



At the point of tree depth = 7, the testing error begins to increase although the training error decreases continuously. From this plot, we can confirm that the optimum value for max_depth hyperparameter is 7.

Hyper parameter Tuning with Grid Search method

Tuning hyper parameters can be done using the Grid Search method along with k-fold cross-validation. The equivalent Scikit-learn function is GridSearchCV. It finds all the hyperparameter combinations for a specified k number of folds.

We want to find the best combination from all the hyper parameter combinations for the following two hyper parameters in DecisionTreeRegressor.

```
from sklearn.model_selection import GridSearchCV

model = DecisionTreeRegressor()

gs = GridSearchCV(model,
                  param_grid = {'max_depth': range(1, 11),
                                'min_samples_split': range(10, 60, 10)},
                  cv=5,
                  n_jobs=1,
                  scoring='neg_mean_squared_error')

gs.fit(trainX, trainY)

print(gs.best_params_)
print(-gs.best_score_)
```

```
{'max_depth': 10, 'min_samples_split': 10}
9512.537289413955
```

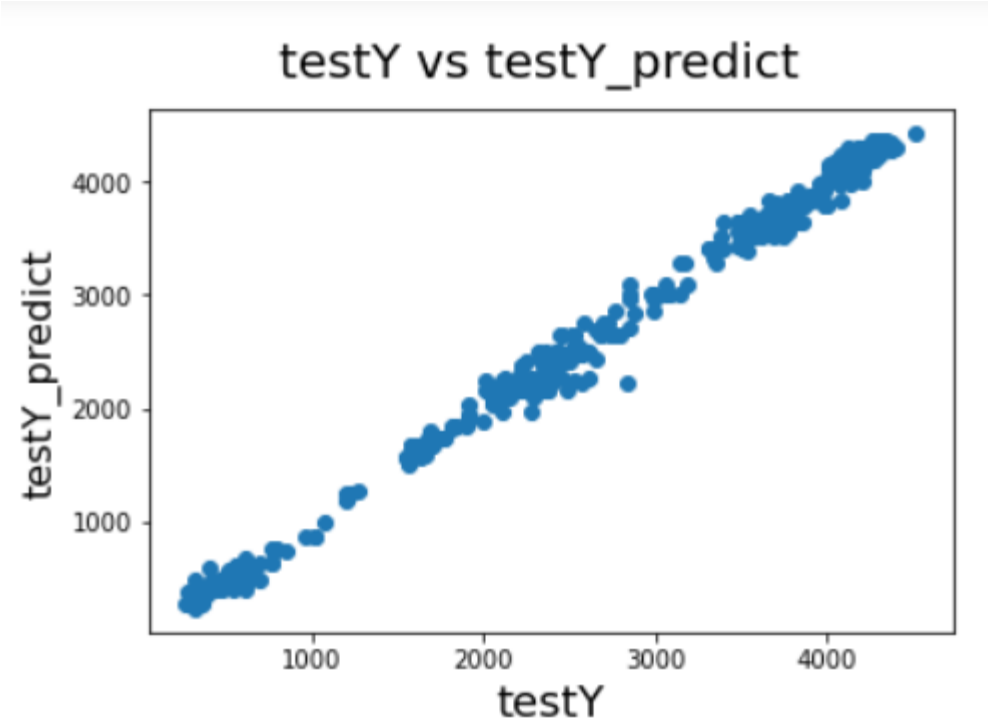
Now, we can create the best model using these optimum values. It avoids both overfitting and underfitting conditions.

```
new_model = DecisionTreeRegressor(max_depth=10,
                                  min_samples_split=10)
#or new_model = gs.best_estimator_
new_model.fit(trainX, trainY)

testY_predict = new_model.predict(testX)
```

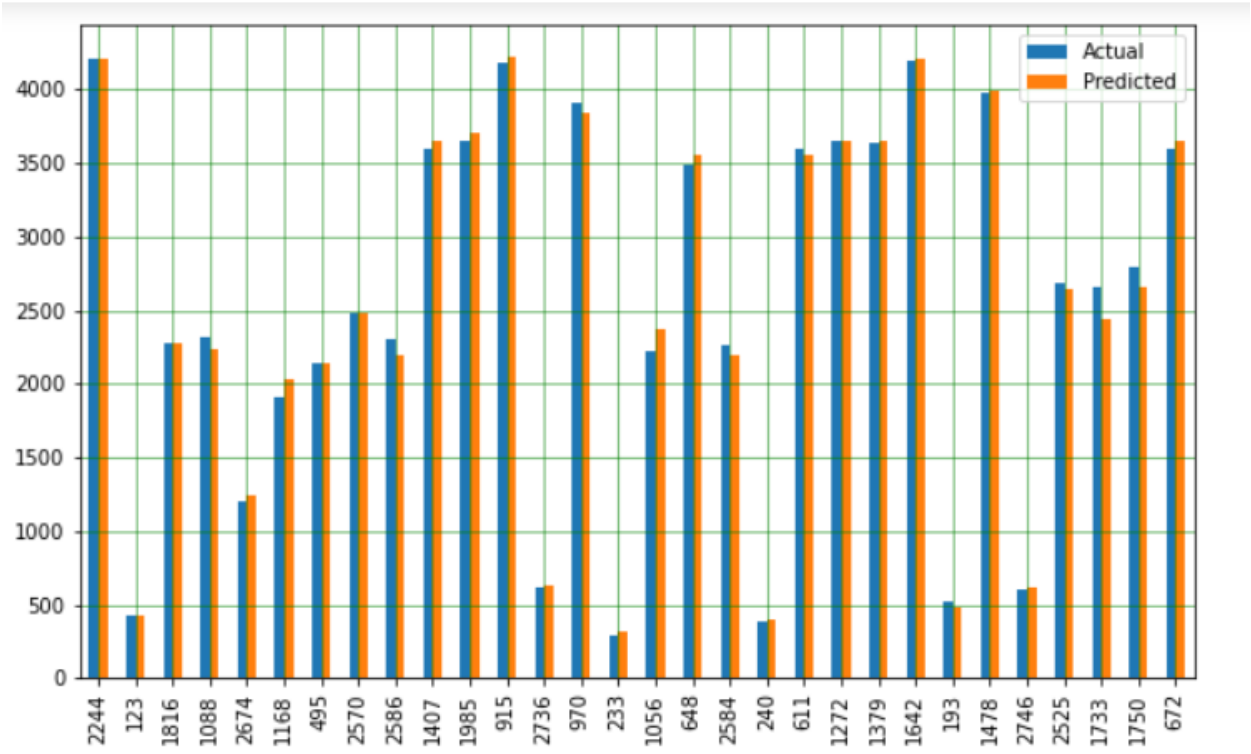
Model Evaluation

```
# Plotting y_test and y_pred to understand the spread

fig = plt.figure()
plt.scatter(testY, testY_predict)
fig.suptitle('testY vs testY_predict', fontsize = 20)              # Plot heading
plt.xlabel('testY', fontsize = 18)                             # X-label
plt.ylabel('testY_predict', fontsize = 16)
```

```
Text(0, 0.5, 'testY_predict')
```



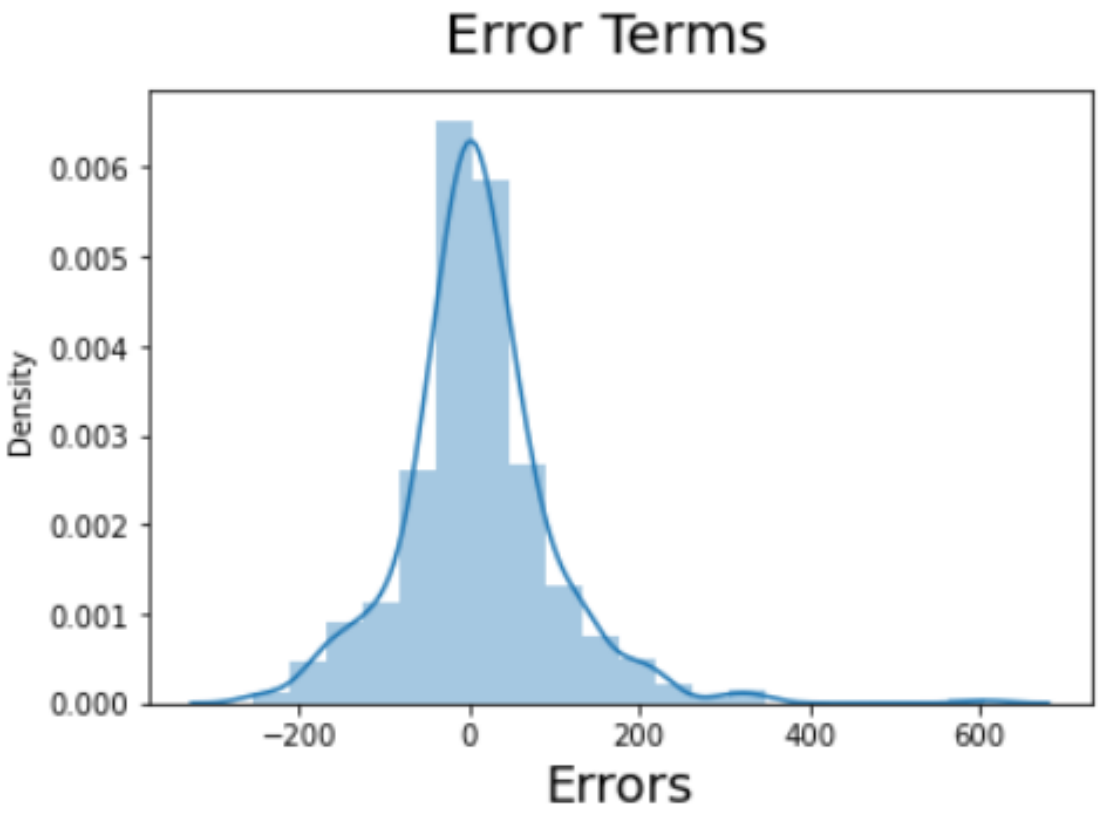
```
#Dislay the predicted value against the Actual value
df_temp2 = pd.DataFrame({'Actual': testY, 'Predicted': testY_predict})
df_temp2.head()

#Visualize the prediction vs Actual value of Oxygen
df_temp2 = df_temp2.head(30)
df_temp2.plot(kind='bar',figsize=(10,6))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



```
# Plot the histogram of the error terms
fig = plt.figure()
sns.distplot((testY - testY_predict), bins = 20)
# Plot heading
fig.suptitle('Error Terms', fontsize = 20)
# Give the X-label
plt.xlabel('Errors', fontsize = 18)
```

```
C:\Users\admin\anaconda3\lib\site-packages\seaborn\distribu
s a deprecated function and will be removed in a future ver
`displot` (a figure-level function with similar flexibility
r histograms).
  warnings.warn(msg, FutureWarning)
```

```
Text(0.5, 0, 'Errors')
```



The predicted values are quite close to actual value. The error distribution is almost Gaussian in nature. This proves the Model-3 is a descent model too.

Advantages of decision trees

- Do not require feature scaling
- Can be used for nonlinear data
- Non-parametric: Very few underlying assumptions in data
- Can be used for both regression and classification
- Easy to visualize
- Easy to interpret

Disadvantages of decision trees

- Decision tree training is computationally expensive, especially when tuning model hyperparameter via k-fold cross-validation.
- A small change in the data can cause a large change in the structure of the decision tree.

Comparing three models

Comparison Table

	Compare Evaluation Criteria	Evalun Score	Observations
NN MLP Regressor	Hyper parameters : Alpha = .0001, hidden_layer_sizes=(150,100,50), max_iter = 300	RMSE = 163	Max_iter = 300 is giving better result and less RMSE value.
	Hyper parameters : 'alpha': 0.05, 'hidden_layer_sizes': (150, 100, 50), 'learning_rate' = , 'max_iter': 100	RMSE = 203	Value has gone down.
Linear Regression	r2_score	r2_score = 0.97	The accuracy is quite descent with Linear Regression Model also.
Decision Tree Regressor	r2_score	r2_score = 0.99	Decision Tree Regressor is giving better accuracy.

Observation

- Here we have applied the three different Regression algorithms to model the Oxygen Uptake for a sample dataset.
- The POC module is built with python code. The code is modular. Each model is generated separately and evaluated with proper metrics.
- We also applied the model optimization technique and hyper parameters to improve the result.
- This gives a good foundation for the next phase of the project.

Final Analysis/Conclusion

In this research-based project we studied the impact of Power on oxygen uptake in (VO2). Measurement of oxygen uptake is expensive and not everyone can afford to have such expensive tools. The purpose of this research was to study few Machine Learning algorithms with simple measurable data to predict the oxygen uptake for cyclists and we could achieve the model with three different algorithms. All three models achieved the accuracy quite descent even with small number of test data. Hence, we can apply this model on some real data.

Real data must be cleaned and pre- processed as required and then use for training purpose. The trained model then be used to predict the Oxygen uptake for any unknown health parameters like power. With larger set of real data, we hope to get better accuracy.

This is data driven algorithmic approach and do not depend on any rule base coding technique. Hence no need to change the coding logic if there is any change in the data behavior. We only need to re train the model if there is any drift in prediction accuracy. This is exactly the purpose of AI/ML system.

Our model selection is strategical, we selected one regression model, one tree based and one neural network. We also did an extensive data analysis for all the features which shows the correlation between feature and the Oxygen uptake. If in future we want to build predictive model with other features we can use this as base research.