Redback Operations
Data Science and Analytics Team
M.TELLEY

DOCUMENTATION
ML Algorithm and Models

Purpose
Currently, Strava has a working model that will predict power output, however, on review it is quite inaccurate; Strava uses a mix of wind resistance, rolling resistance, gravity and acceleration. This is the frustration of many Amateurs cyclists using the application. As part of the Workout Analysis, prediction models could be used where certain data points were not immediately available i.e, power. As an example, power meters on certain indoor trainers and cycles are not factory fitted and are typically cost-prohibitive to purchase. Moreover, power or wattage output is a critical performance metric in performance cycling and can heavily influence a cyclist's training behaviour.

Scope
Investigate at minimum two different algorithms/models that could be used to predict power output. Based on research the following algorithms/models were selected:

1. Scikit-learn Linear Model - LinearRegression
2. Scikit-learn Linear Model - ElasticNet
3. Scikit-learn Random Forest - RandomForestRegressor

Prior to working on the implementation of the said algorithms/models listed above, vigorous data analysis needed to be completed to begin understanding and formulating a set of meaningful features and become familiar with key data trends, dependencies and relationships.

Data Analysis
A total of four different python projects using PyCharmEdu were created. As each project was completed, the level of data analysis increased - This was to ensure a steady learning pace both with the data and using Python and key libraries such as but not limited to Matlab, Pandas, Numpy etc. Moreover, an attempt was made to remotely query via API the BigQuery database to simulate a real-world environment i.e, assuming the Database is being consistently updated with new data, static local CSV files will not suffice.

1 - Big Query API and rudimentary data analysis
File: 1_bigquery_stat_analysis

The first seconds of the project aim to establish a connection with the BigQuery database, query the database for User 1 data and then store the information in a pandas dataframe. Note, the tables in the BigQuery had already undergone preprocessing in order to simplify the query requests in PyCharm.

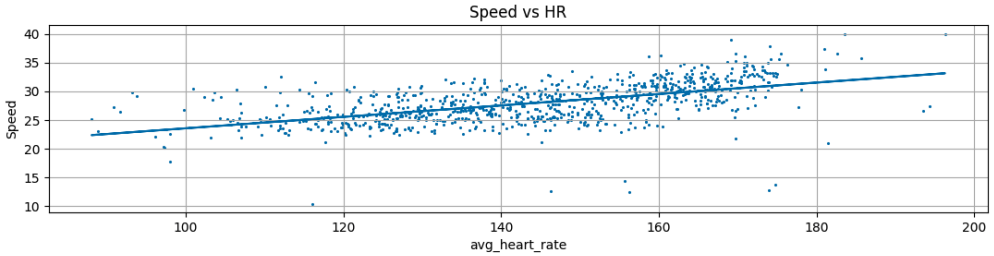
```
#Get Data
sql_query = """
    SELECT
        *
    FROM

`BIGQUERYTABLE.fitness_user_summary_features.feature-summary-filtered
`
    WHERE avg_power IS NOT NULL AND enhanced_avg_speed IS NOT NULL
 """

#Get User 1 Data
sql_query = """
    SELECT
        *
    FROM

`BIGQUERYTABLE.fitness_user_summary_features.feature-summary-filtered
-user1`
    WHERE avg_power IS NOT NULL AND enhanced_avg_speed IS NOT NULL
"""
```

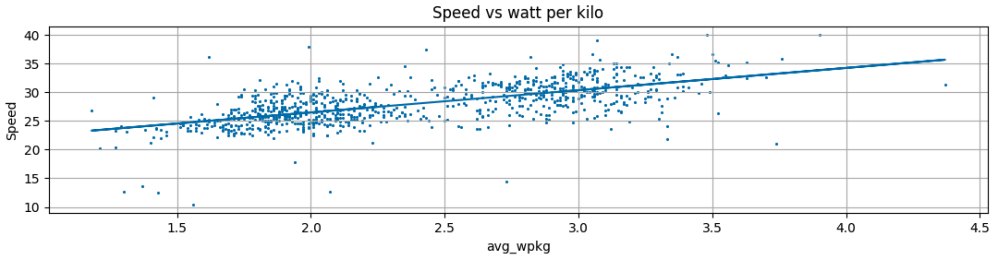
Once data for both all available users and User1 separately were stored in respective dataframes, correlation of numeric data attributes was conducted using Pandas Corr() function, to begin to understand the relationships (if any) between the data attributes.
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html

```
#Reformat DF to have only numeric data attributes
dfCor = df2[['distance',
             'enhanced_avg_speed',# 'avg_power',
             'avg_wpkg',
             'avg_power',
             'avg_heart_rate',
             'avg_max_heart_rate_perct',
             'day_of_week',
             'hour']]
matrix = dfCor.corr()
print(matrix)
corrplot(matrix, size_scale=500, marker='s')
```

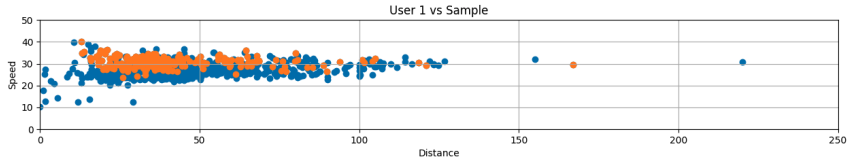
```
matrix.to_csv("data_corr.csv")
```

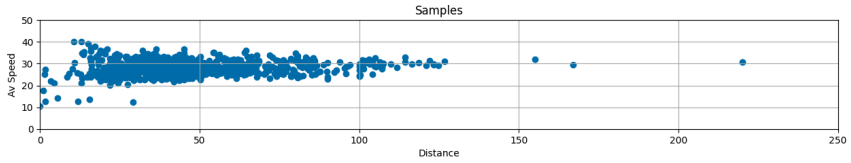
Once the correlation analysis was complete, attempts at visualising relationships of interest were conducted for - Speed vs Heart Rate and Speed vs Watts per Kilogram

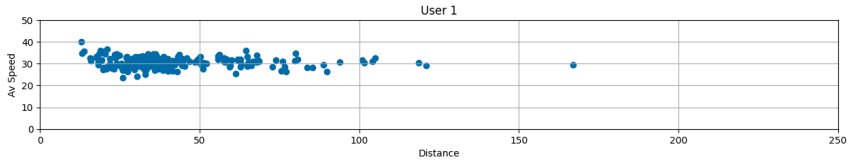




After plots were created and shown, an attempt to understand how the target user's (User1) data metrics differed (if any) from the sample data metrics:





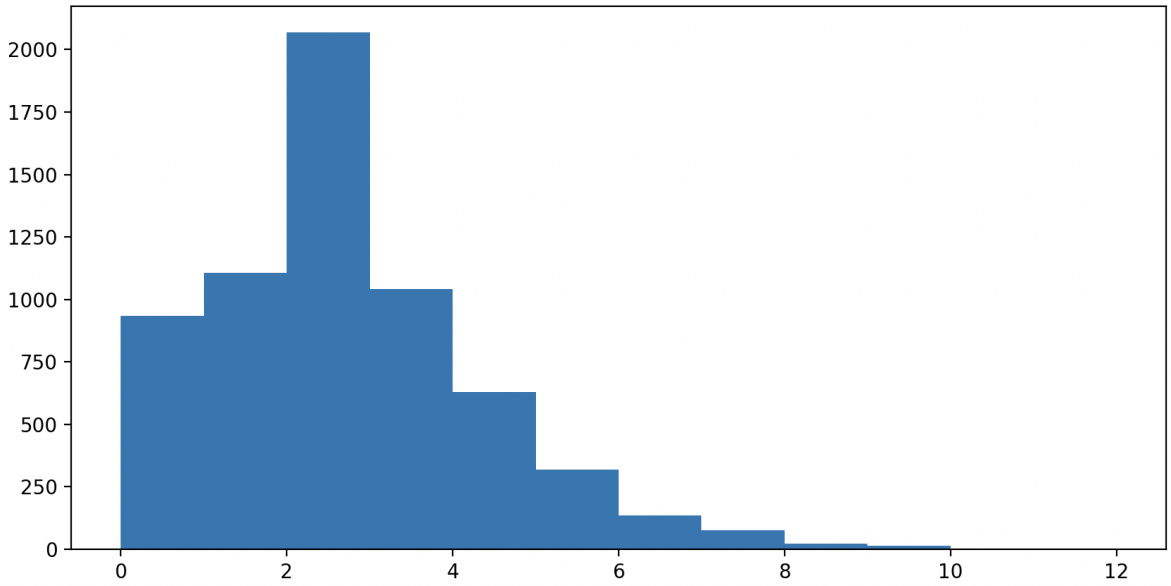


This concluded the first attempt at a) connecting via API to the Bigquery database to query, pull and store data and then a few minor attempts at analysing the data available.

2 - Supplementary Bigquery and Data Analysis
File: 2_workout_data_modelling

For the second attempt at using PyCharm to query the BigQuery database and conduct analysis work, a much larger dataset was used; the number of rows = >6,000.

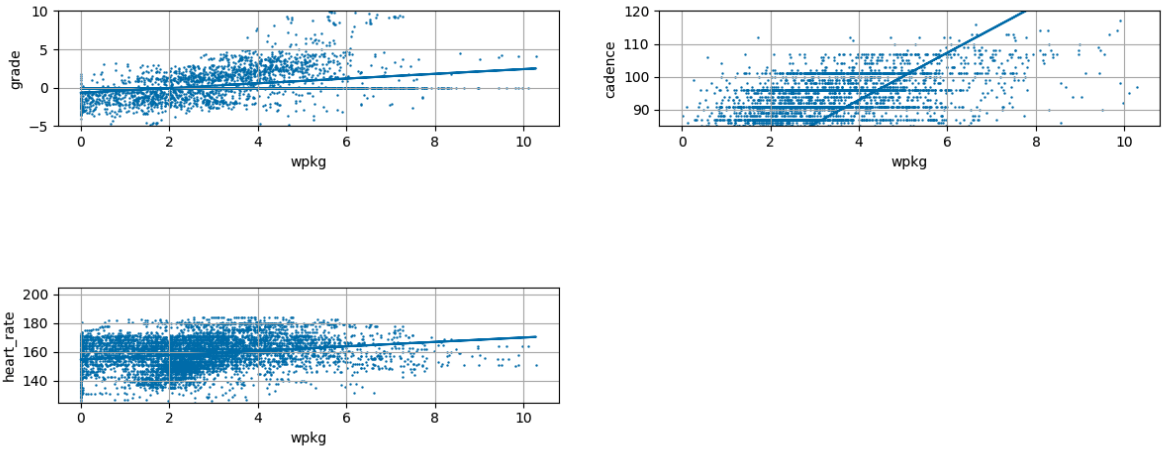
The initial focus was to understand the distribution of power using a standardised measure called Wattage per kilogram (wpkg) - The plot below showed the distribution of wpkg was predominantly between 2 and 3 wpkg - understand what a ‘normal’ wpkg value was going to be important in the future.



Thereafter, a correlation plot was shown to aid in visualising the analysis of key performance metrics. The following resource was used: https://pypi.org/project/heatmapz/. It was clear that cadence was becoming a data metric of key interest in relation to power-related metrics (wpkg,

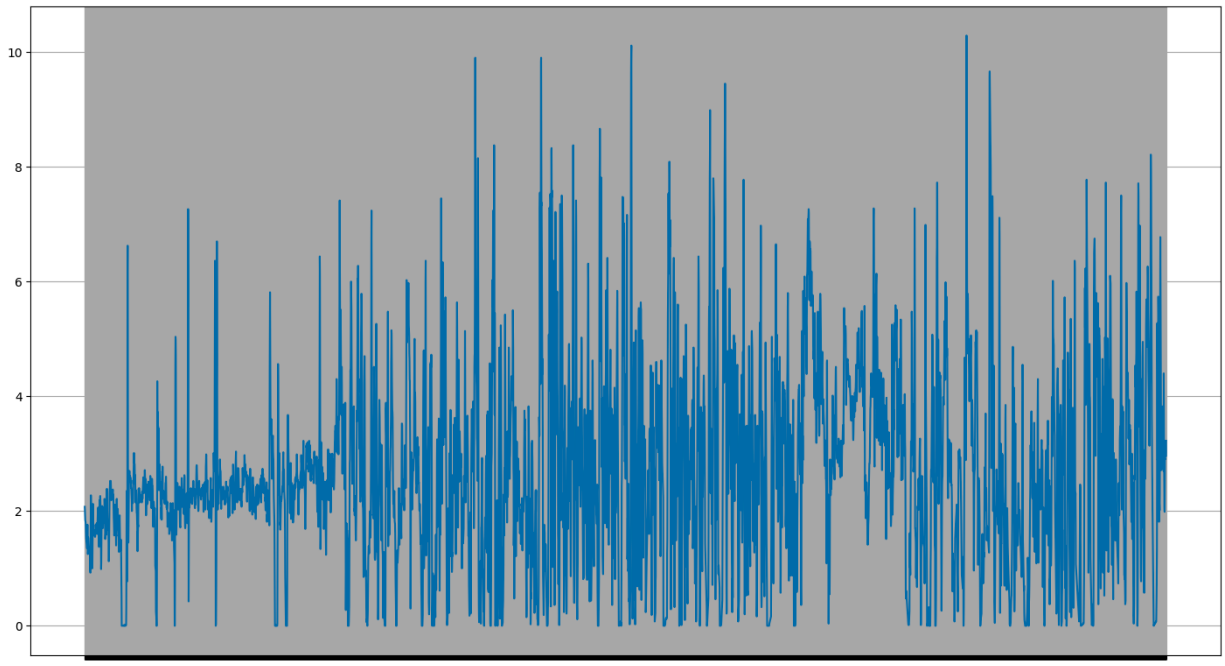
power and the % of functional threshold power - FTP).

The top three metrics of interest were then plotted in a scatter plot against wpkg to again, aiding visualise data analysis:



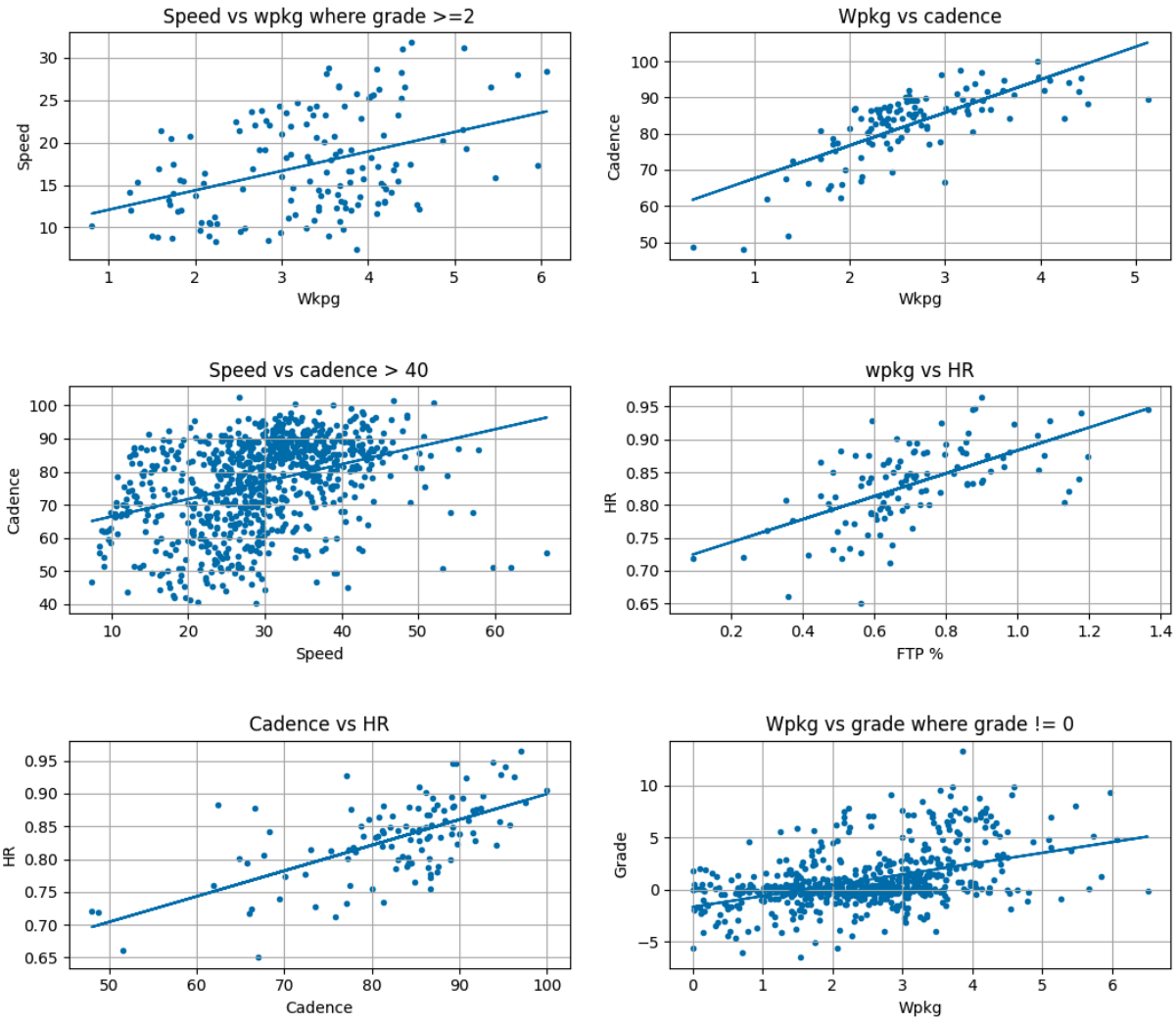
To finish off this PyCharm project, an attempt was made to plot over power data - This presented problems as there were too many records to present and the deviation between the

data points produced a very noisy plot. The data was recorded in intervals of 1 second - Further work was needed to get data into different time intervals.



3 - Continuation of the Previous Project
File: 3_bigquery_summary_stats

Further work was conducted in BigQuery to wrangle data into a format that would be conducive to future data analysis. The big change was getting activity data into 1-minute intervals along with being to focus on a single activity session (had defined start and end points). To begin with, correlations were assessed again to determine if anything had changed and a larger number of performance metrics were assessed:



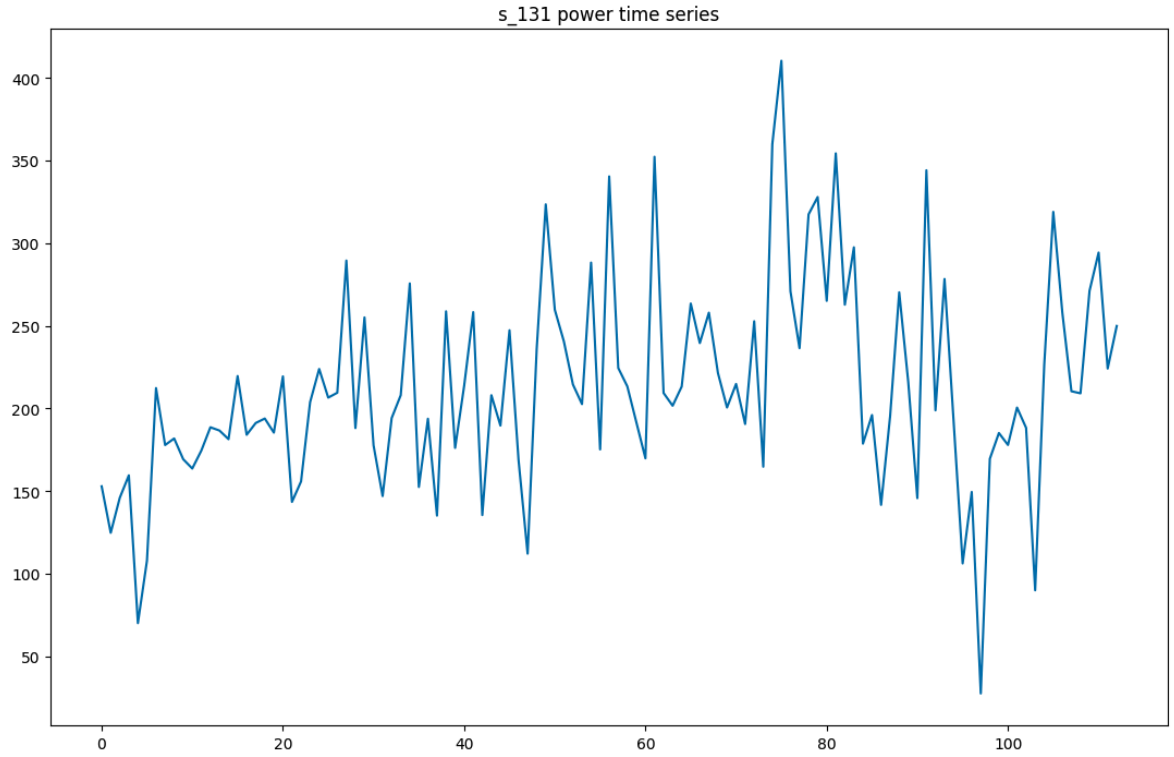
Unlike previous attempts at Data Analysis, filtering of data did occur:

1. grade != 0 - An attempt was made to assess how grade influenced power where the grade was not zero. The bottom right plot above shows a somewhat strong linear relationship ie when a cyclist is riding up an incline it is likely that power output will be higher
2. Grade > 2 - Attempt to assess the impact grade has on speed and wpkg where the grade is > 2; it was clear that a higher speed was positively associated with high wpkg.

3. Cadence > 40 - In an attempt to remove idling movement i.e., when cadence is equal to zero or very low (<40), the relationship between speed and cadence was assessed. There was a weak positive relationship between the two variables.

The outcome of future analysis helps establish a stronger view of the key elements that might influence power output.

To finish, a plot of power output (watts) for a single session (**session # 131**) was attempted - Session 131 will be used later as the key dataset to compare model performance. However, unlike previous attempts, the time interval or step was set at 1 minute. The plot below is clearer and more readable and forms the basis for future analysis. The ability to potentially test and learn in 1-second intervals but display in 1-minute intervals would later be established.



This concluded data analysis efforts.

Modeling

Scikit-learn Linear Model - LinearRegression
File: Elastic-Net-Regression-Models-Workout-Data

To begin and as identified earlier, a basic Linear Regression model would be adopted initially. The first parts of this project focus on statistical analysis building on previous data analysis efforts. Moreover, the dataset used for this model was initially adopted in 5-minute intervals to aggregate (average metric values over a 5-minute period) the data to reduce the deviation in

data attribute values and the power metrics selected were watts as opposed to watts per kilogram. Thereafter the standard 1-second interval would be used.

Linear regression is the most basic form, where the model is not penalised for its choice of weights, at all. That means, during the training stage, if the model feels like one particular feature is particularly important, the model may place a large weight on the feature or data attribute. This may become problematic as the size of the data set becomes larger (Wenwei Xu)

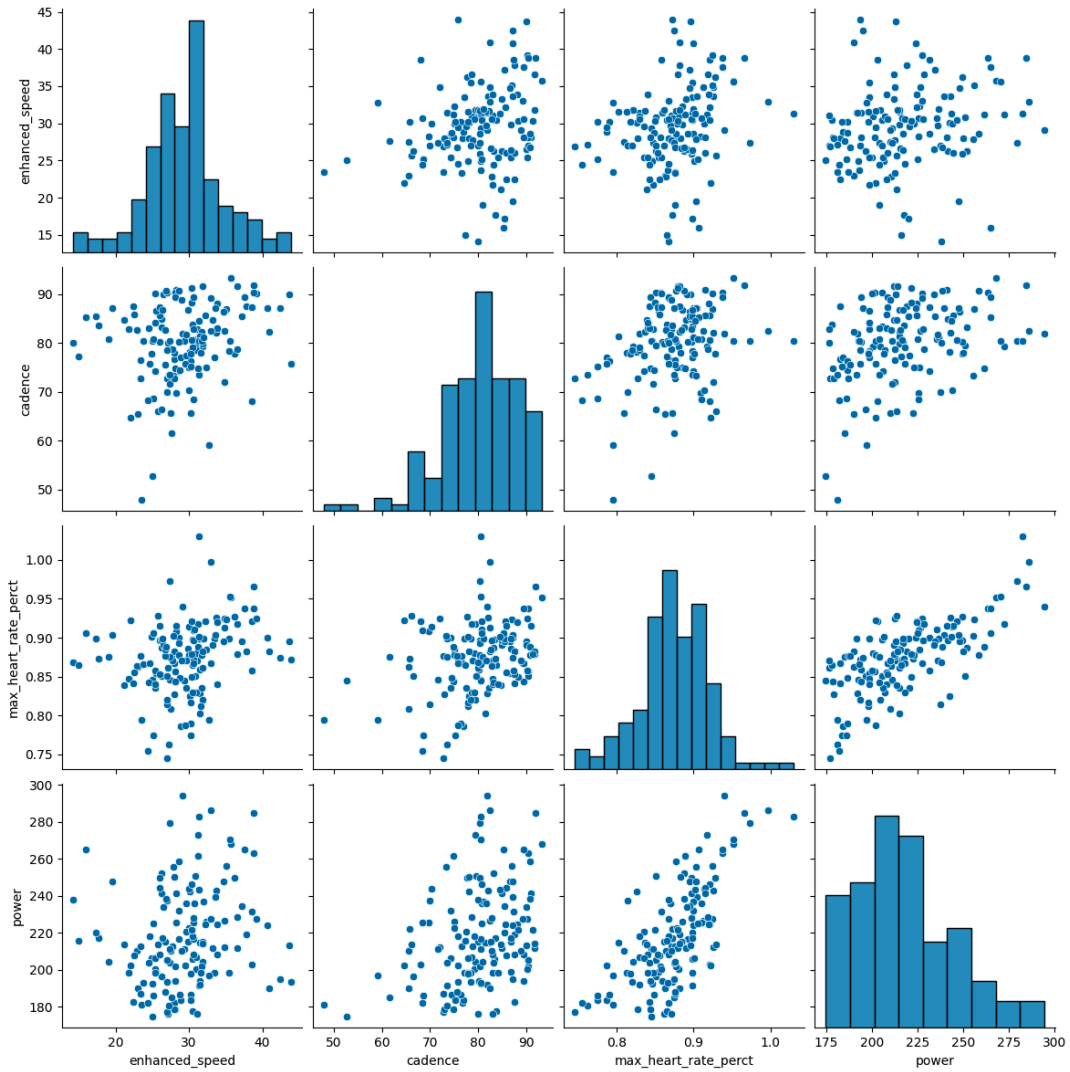
Dataset dynamics:

- Train set (5-minute): Over 12 hours of activity were used, equating to over 145 rows of data.
- Test set (5-minute): Only 48 hours of activity were used, equating to over 580 rows of data.
- Train set (1-second): Over 9 hours of activity were used, equating to over 33,750 rows of data.
- Test set (1-second): Over 37.5 hours of activity were used, equating to over 135,000 rows of
- data.

The increase in the amount of data was also a considered effort - The impact to model performance where data was excessively more abundant would be monitored.

5 minutes:

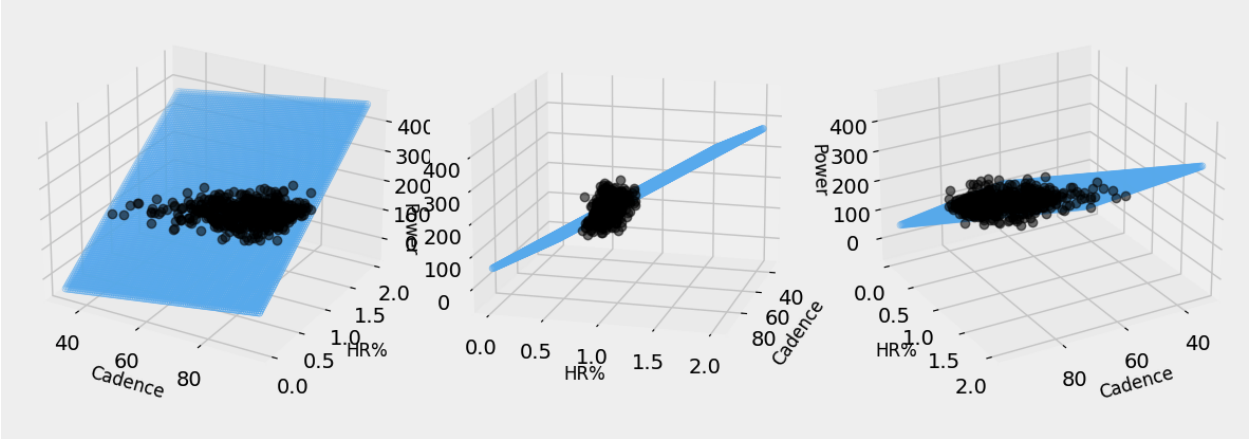
A quick assessment of metric correlations and distribution was calculated:



Over the available data attributes - cadence and % of max heart rate (max heart rate is calculated using the HUNT formula, Age = 33). These two attributes would be used in the model.

Outputs

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  power   R-squared:                       0.453
Model:                            OLS   Adj. R-squared:                  0.451
Method:                 Least Squares   F-statistic:                     239.3
Date:                Tue, 13 Dec 2022   Prob (F-statistic):           2.10e-76
Time:                        20:23:10   Log-Likelihood:                -2709.4
No. Observations:                 580   AIC:                             5425.
Df Residuals:                     577   BIC:                             5438.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -90.6007     16.413     -5.520      0.000    -122.838     -58.364
cadence        1.5224      0.114     13.367      0.000       1.299       1.746
max_heart_rate_perct   204.2133     20.591      9.918      0.000     163.772     244.655
==============================================================================
Omnibus:                        8.998   Durbin-Watson:                   1.127
Prob(Omnibus):                  0.011   Jarque-Bera (JB):               13.197
Skew:                           0.100   Prob(JB):                      0.00136
Kurtosis:                       3.711   Cond. No.                     1.85e+03
==============================================================================
```



R^2 range between 0 and 1, where R^2=0 means there is no linear relationship between the variables and R^2=1 shows a perfect linear relationship. In our case, we got an R^2 score of 0.45 which means 45% of our dependent variable can be explained using our independent variables.

Model Validation

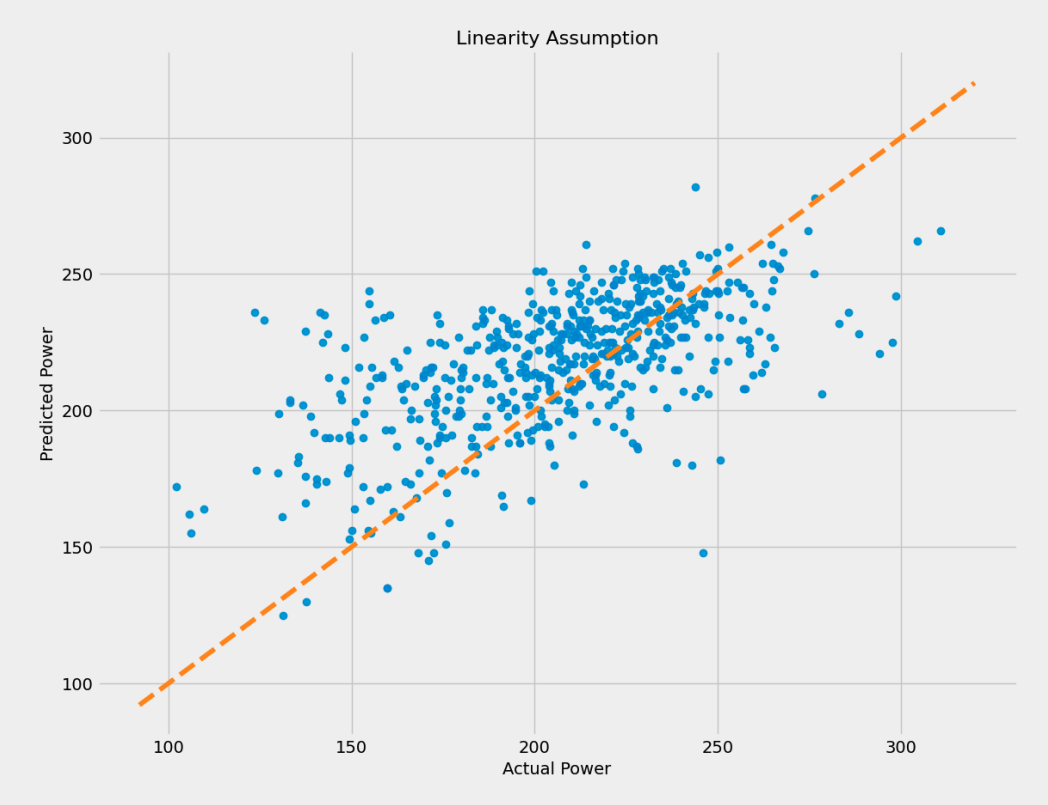
We can evaluate a model by looking at its coefficient of determination (R^2), F-test, t-test, and also residuals. Before we continue we will rebuild our model using the statsmodel library with

the OLS() function. Then we will print the model summary using the summary() function on the model. The model summary contains lots of important values we can use to evaluate our model.

Prediction = Intercept - Coefficients1-x_1 + Coefficients2-x_2. The intercept value is the estimated average value of our dependent variable when all of the values of our independent variables are 0

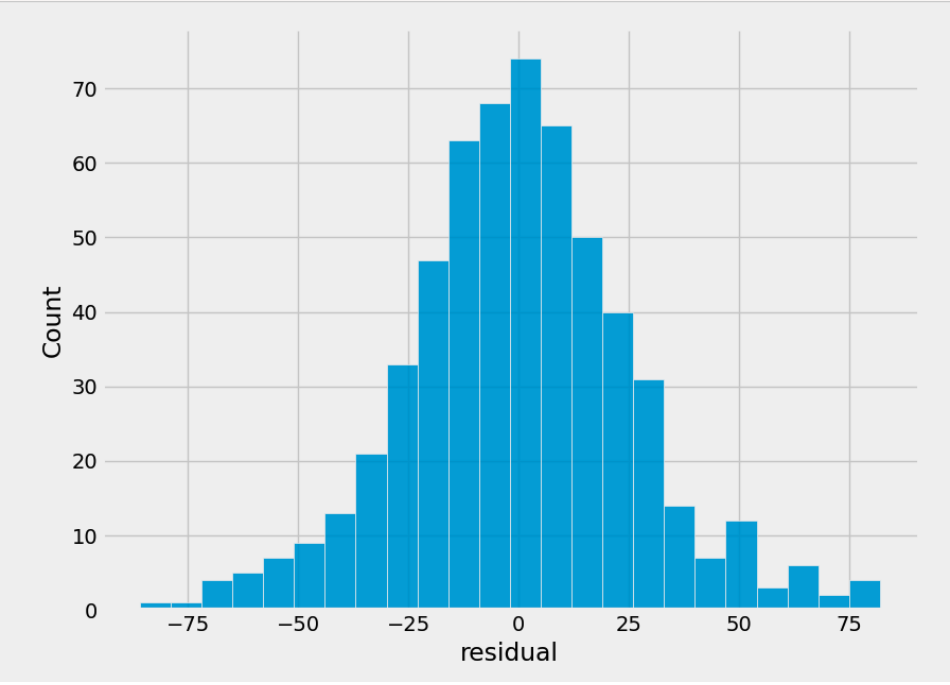
Intercept: -153.09521518
Coefficients: 0.50632036, 378.11543937
Model R^2: 0.4533
F-statistic: 239.28687410590658
Probability of observing value at least as high as F-statistic: 2.10

Linearity
This assumes that there is a linear relationship between the independent variables and the dependent variable. In our case since we have multiple independent variables, we can do this by using a scatter plot to see our predicted values versus the actual values. Plotting the observed vs predicted values



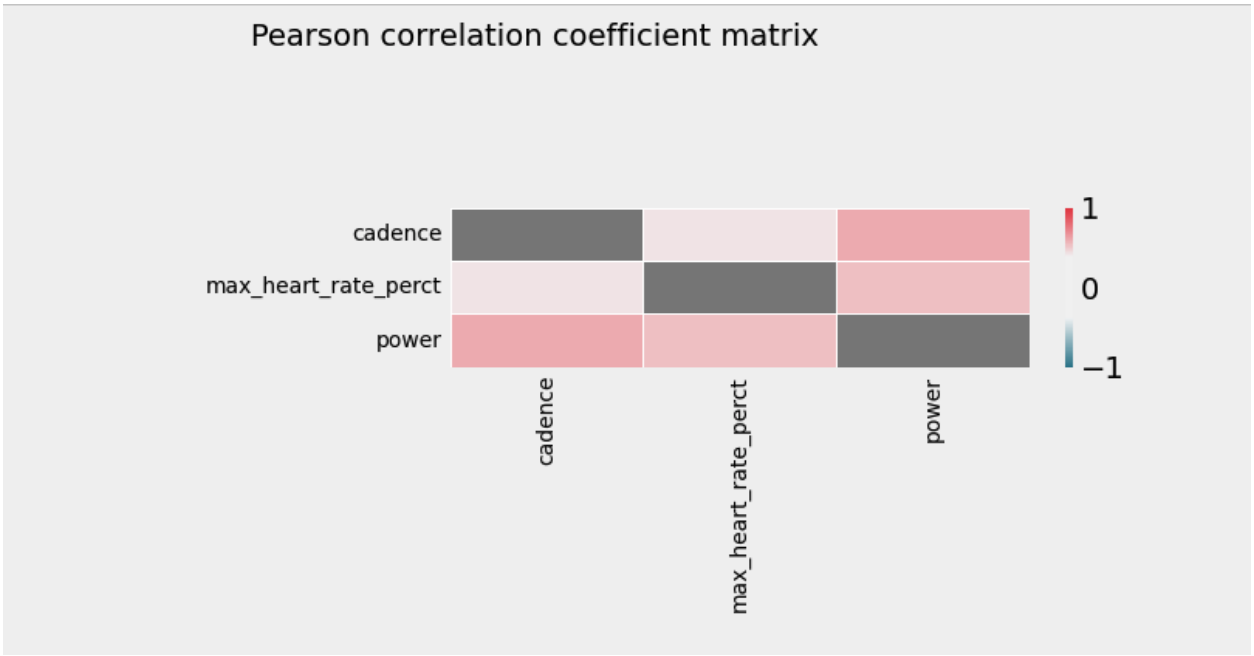
Normality
This assumes that the error terms of the model are normally distributed. We will examine the normality of the residuals by plotting it into a histogram and looking at the p-value from the Anderson-Darling test for normality. We will use the normal_ad() function from statsmodel to calculate our p-value and then compare it to the threshold of 0.05, if the p-value we get is higher than the threshold then we can assume that our residual is normally distributed.

p-value from the test Anderson-Darling test below 0.05 generally means non-normal: 0.00047.
Residuals are not normally distributed



Multicollinearity
This assumes that the predictors used in the regression are not correlated with each other. To identify if there are any correlations between our predictors we can calculate the Pearson correlation coefficient between each column in our data using the corr() function from Pandas dataframe. Then we can display it as a heatmap using heatmap() function from Seaborn.

The output of the heatmap shows us that the independent variables are affecting each other and that there is multicollinearity in our data.



Autocorrelation

Autocorrelation is the correlation of the errors (residuals) over time. Used when data are collected over time to detect if autocorrelation is present. Autocorrelation exists if residuals in one time period are related to residuals in another period. We can detect autocorrelation by performing the Durbin-Watson test to determine if either a positive or negative correlation is present. In this step, we will use the durbin_watson () function from statsmodel to calculate our Durbin-Watson score and then assess the value with the following condition:

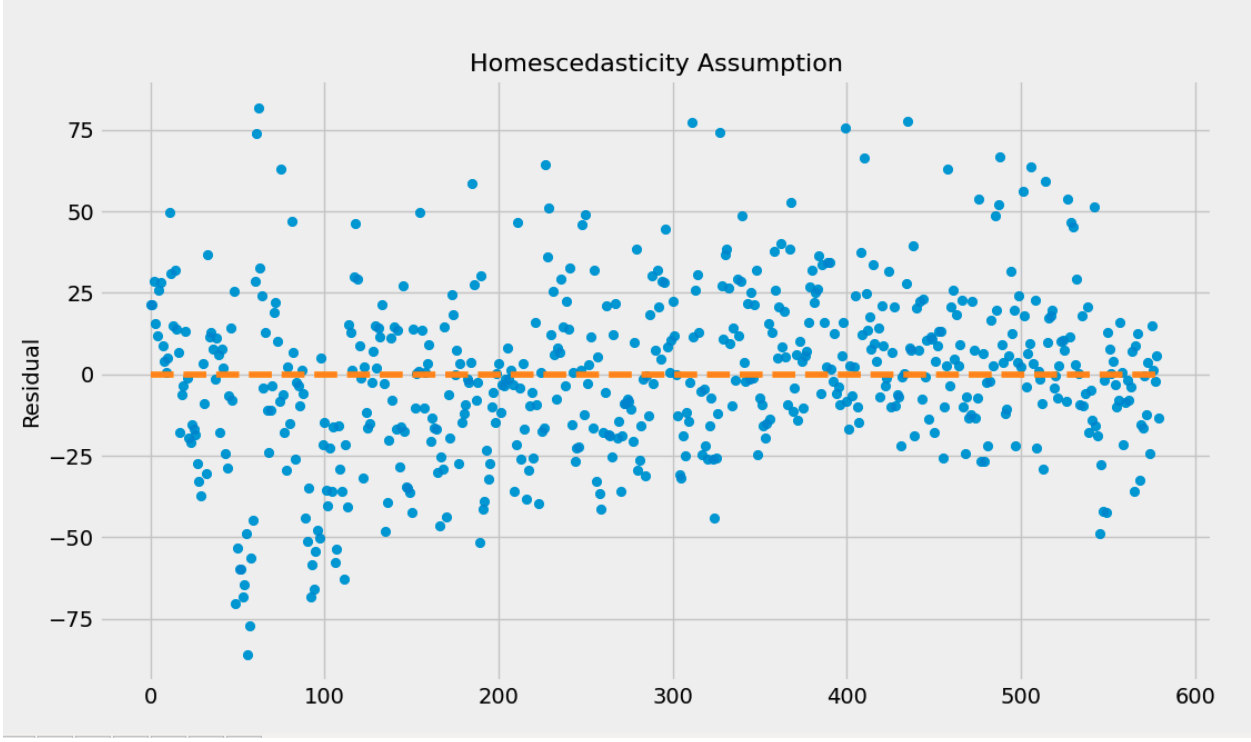
Outcomes

1. If the Durbin-Watson score is less than 1.5 then there is a positive autocorrelation and the assumption is not satisfied
2. If the Durbin-Watson score is between 1.5 and 2.5 then there is no autocorrelation and the assumption is satisfied
3. If the Durbin-Watson score is more than 2.5 then there is a negative autocorrelation and the assumption is not satisfied
4. We can assume that there are Signs of positive autocorrelation in our residual.

Durbin-Watson: 1.127 = Signs of positive autocorrelation; assumption not satisfied

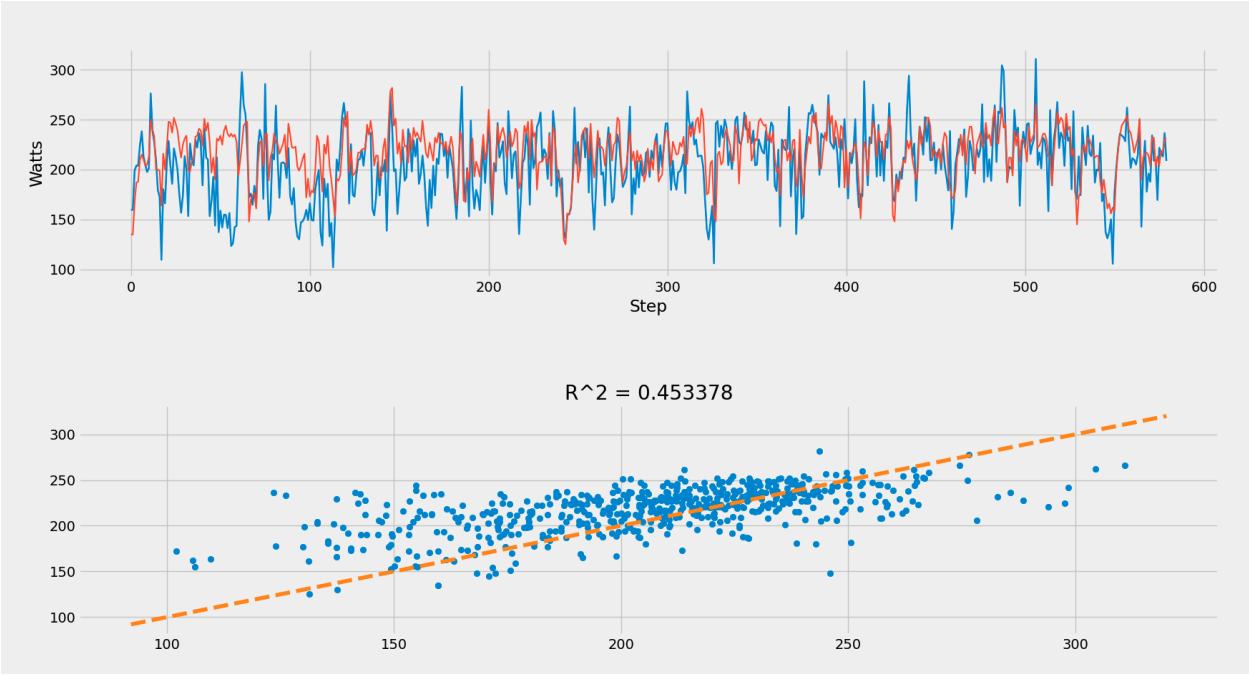
Homoscedasticity

This assumes homoscedasticity, which is the same variance within our error terms. Heteroscedasticity, the violation of homoscedasticity, occurs when we don't have an even variance across the error terms. To detect homoscedasticity, we can plot our residual and see if the variance appears to be uniform - which they don't seem to be.



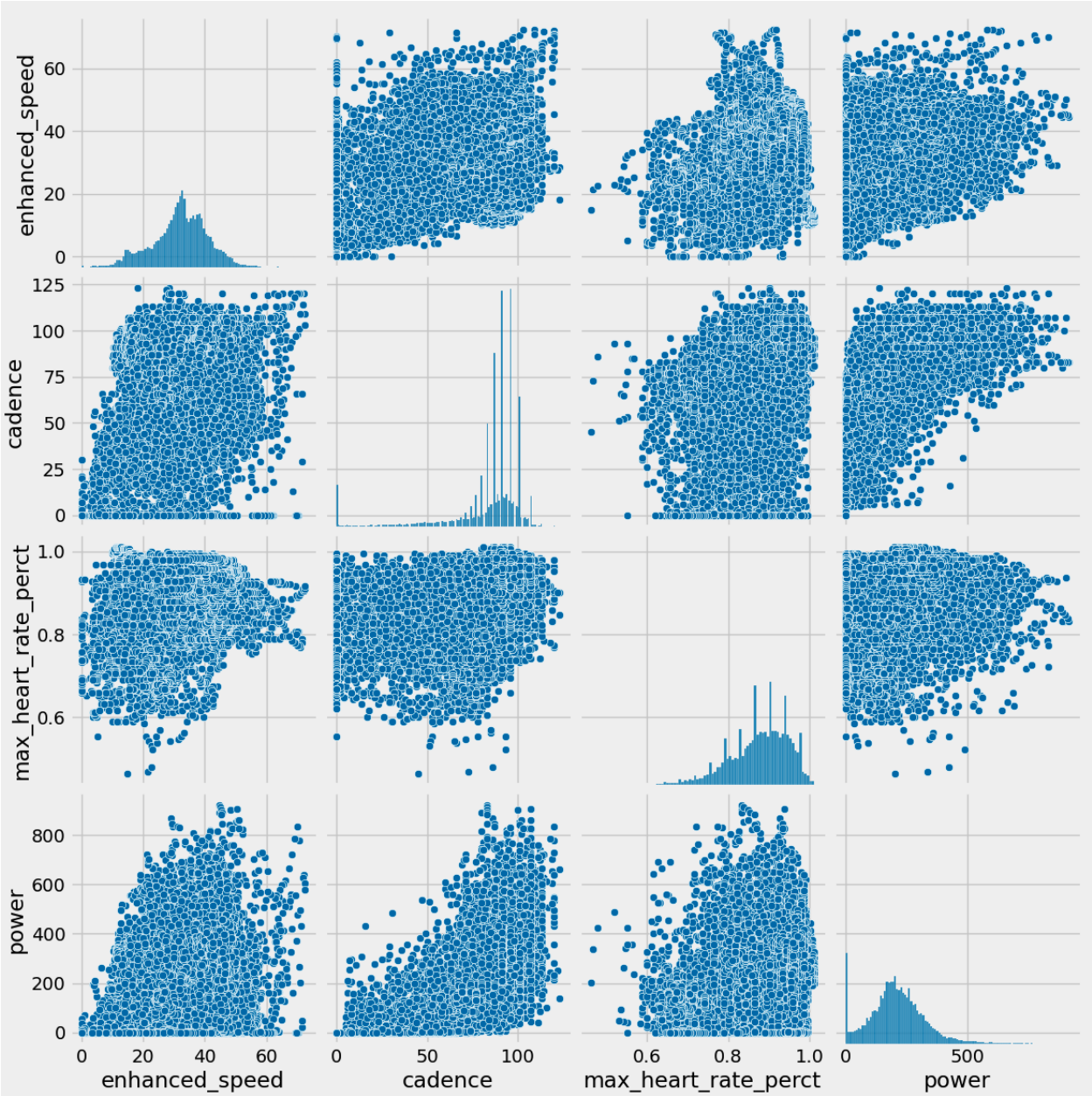
Results

Our model has failed to pass all the tests in the model validation steps, so we can't conclude that our model can perform well to predict power output using the two independent variables, cadence and max HR %. Our model only has an R^2 score of ~45%, which means that there are still about 57% unknown factors that are affecting our power output. This analysis will inform future efforts to predict power output



1 Second:

A quick assessment of metric correlations and distribution was calculated:

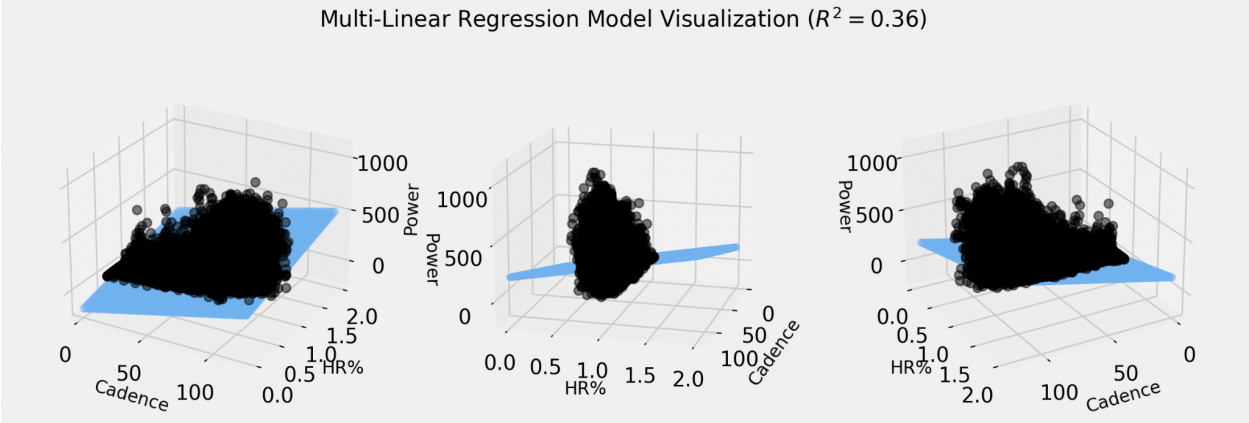


OLS Regression Results

Dep. Variable:	power	R-squared:	0.362
Model:	OLS	Adj. R-squared:	0.362
Method:	Least Squares	F-statistic:	3.846e+04
Date:	Tue, 13 Dec 2022	Prob (F-statistic):	0.00
Time:	20:54:27	Log-Likelihood:	-8.0161e+05
No. Observations:	135455	AIC:	1.603e+06
Df Residuals:	135452	BIC:	1.603e+06
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-152.2997	2.723	-55.929	0.000	-157.637	-146.962
cadence	2.8752	0.011	250.475	0.000	2.853	2.898
max_heart_rate_perct	151.4784	3.285	46.117	0.000	145.041	157.916

Omnibus:	49204.492	Durbin-Watson:	0.183
Prob(Omnibus):	0.000	Jarque-Bera (JB):	265726.393
Skew:	1.669	Prob(JB):	0.00
Kurtosis:	8.994	Cond. No.	1.43e+03



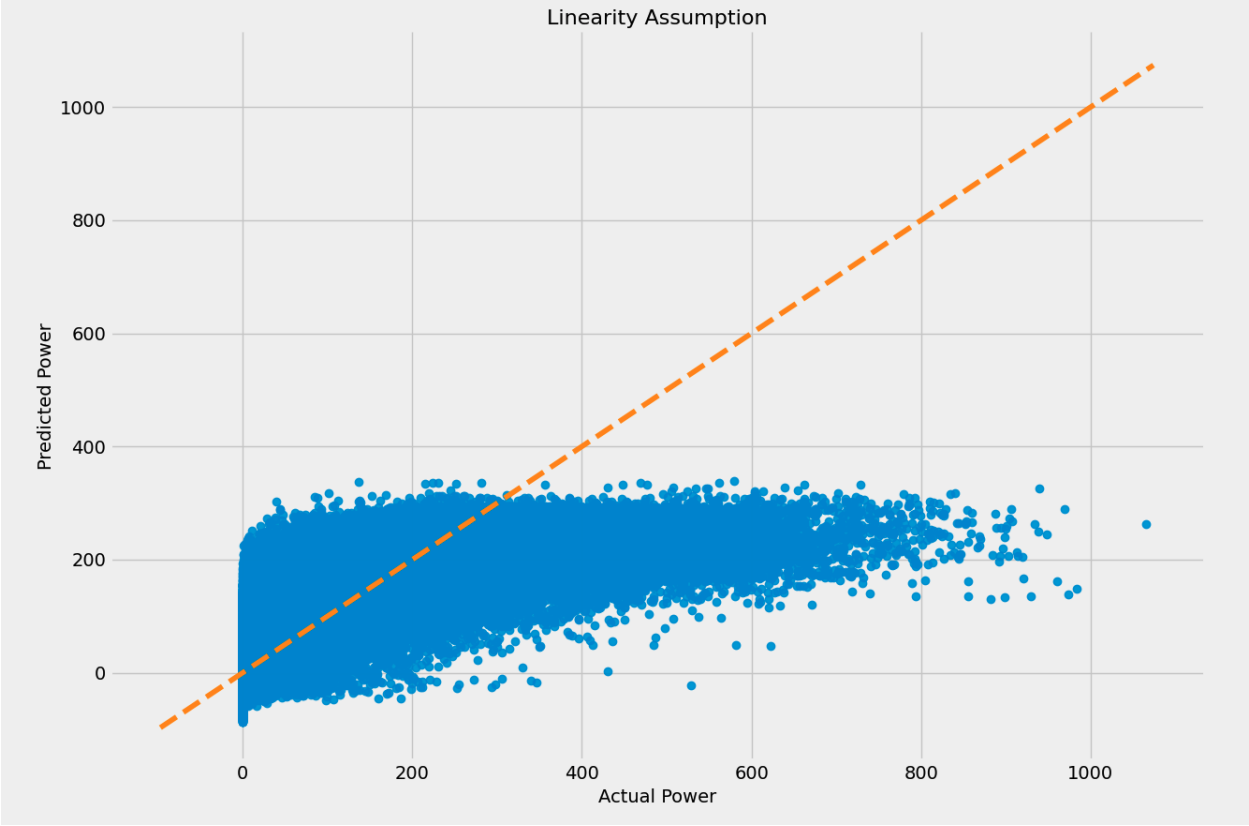
R^2 range between 0 and 1, where $R^2=0$ means there is no linear relationship between the variables and $R^2=1$ shows a perfect linear relationship. In our case, we got an R^2 score of 0.36 which means 36% of our dependent variable can be explained using our independent variables.

Model Validation
We can evaluate a model by looking at its coefficient of determination (R^2), F-test, t-test, and also residuals. Before we continue we will rebuild our model using the statsmodel library with the OLS() function. Then we will print the model summary using the summary() function on the model. The model summary contains lots of important values we can use to evaluate our model.

Prediction = Intercept - Coefficients1-x_1 + Coefficients2-x_2. The intercept value is the estimated average value of our dependent variable when all of the values of our independent variables are 0

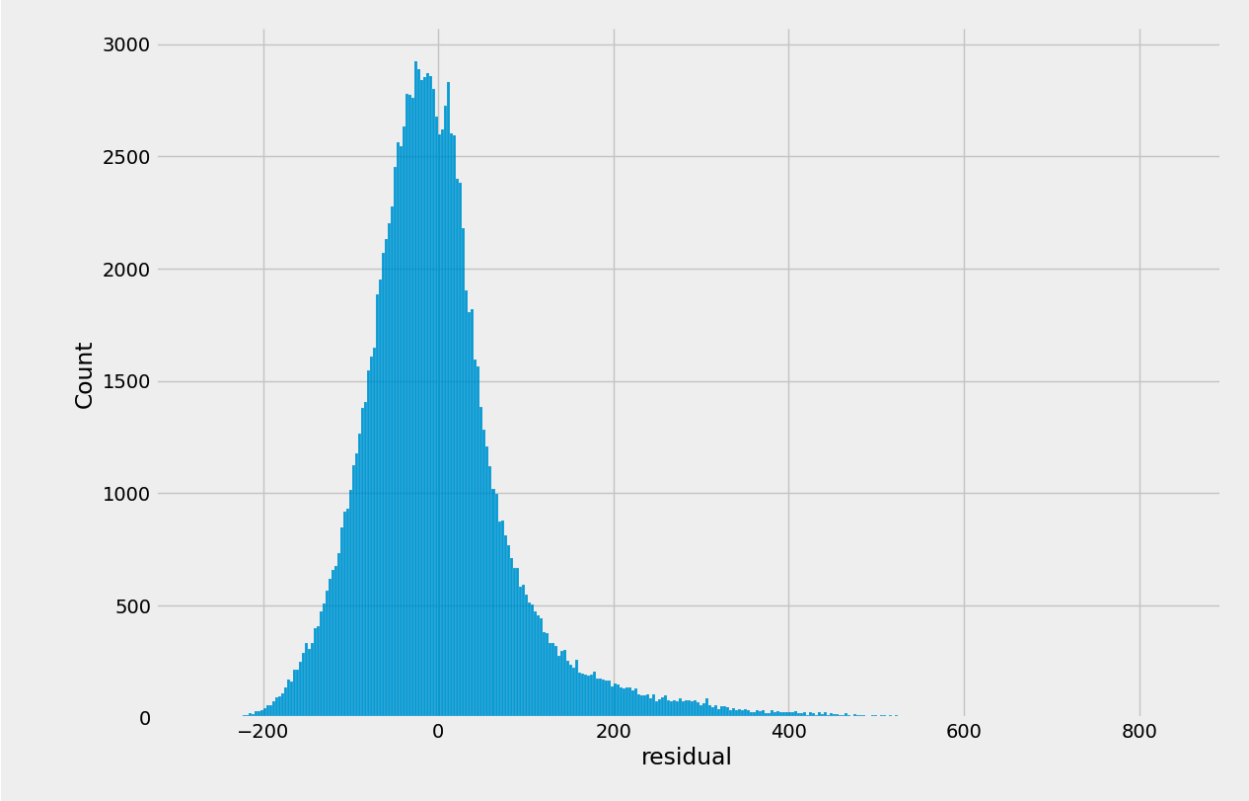
Intercept: -131.63195875
Coefficients: 3.13720344 92.68999586
Model R^2: 0.362
F-statistic: 38455.55
Probability of observing value at least as high as F-statistic: 0.0

Linearity
The 1-second data set performs extremely poorly



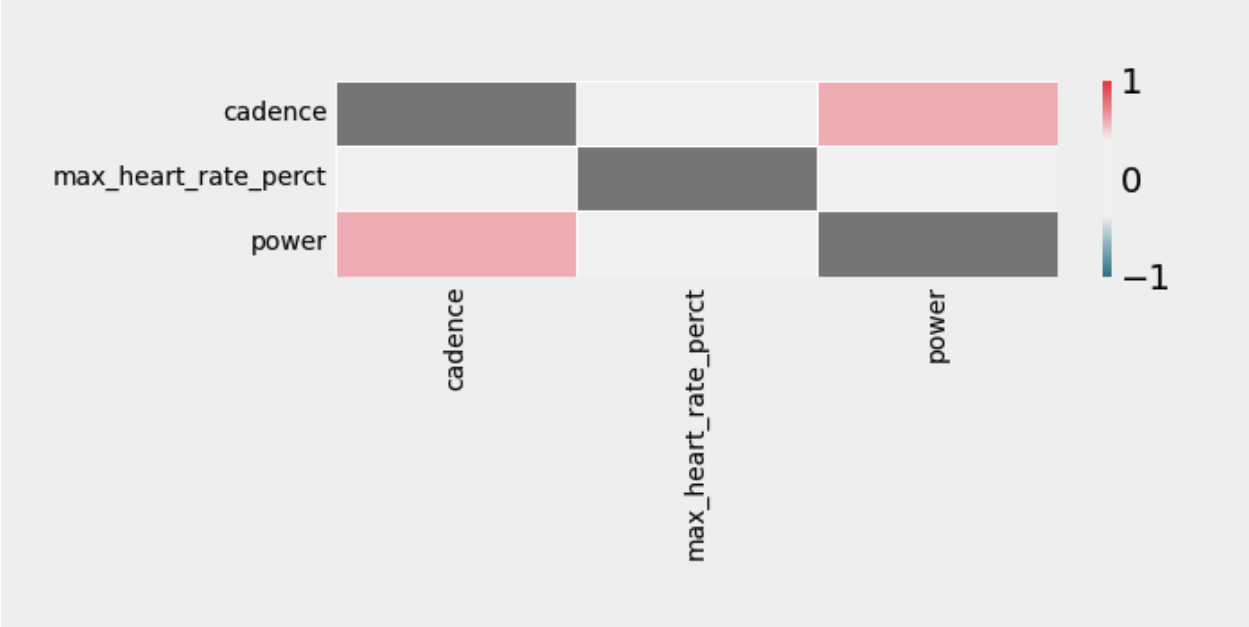
Normality

p-value from the test Anderson-Darling test below 0.05 generally means non-normal: 0.00.
Residuals are not normally distributed



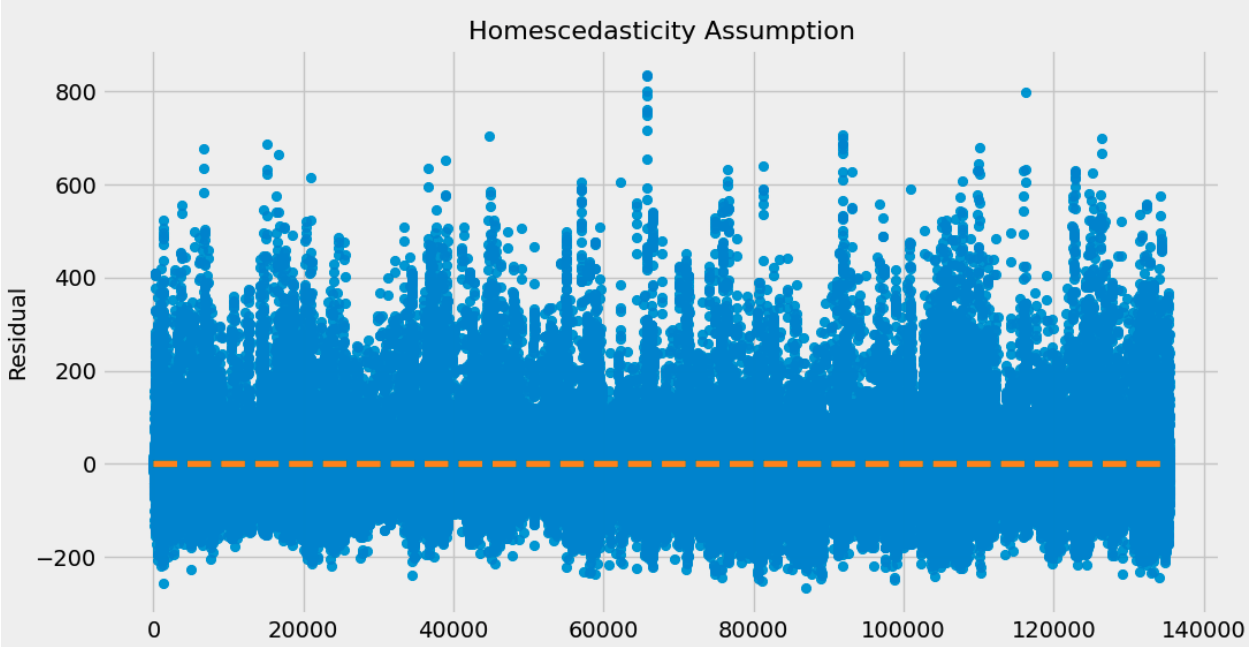
Multicollinearity

Unlike the 5-minute data set, the output of the heatmap shows us that the independent variables are not affecting each other and that there is no multicollinearity in our data.



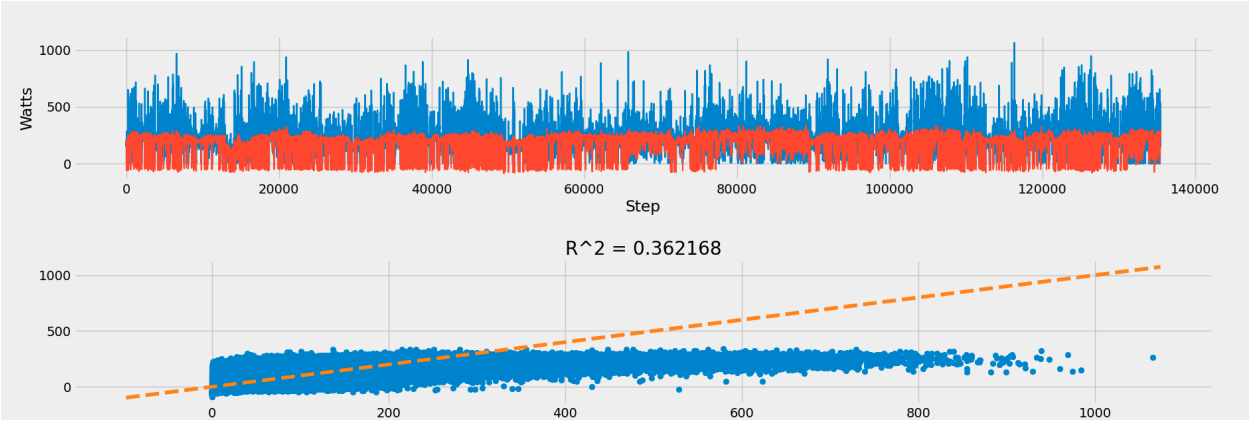
Autocorrelation
Durbin-Watson: 1.183 = Signs of positive autocorrelation; assumption not satisfied

Homoscedasticity
To detect homoscedasticity, we can plot our residual and see if the variance appears to be uniform - which they don't seem to be.



Results

Our model has failed to pass all the tests in the model validation steps and has performed worse compared to the 5-minute data set, so we can't conclude that our model can perform well to predict power output using the two independent variables, cadence and max HR %. Our model only has an R^2 score of ~36%, which means that there are still about 64% unknown factors that are affecting our power output. This analysis will inform future efforts to predict power output:



Resource: https://medium.com/swlh/multi-linear-regression-using-python-44bd0d10082d

Summary
Linear Regression is not a good fit for this problem for a range of reasons; prediction needs to be highly accurate, the amount of data is too large (overfitting is a real problem), lack of penalty or rewards functions and there are a number data attributes (features) that are not being included in the model.

Scikit-learn Linear Model - ElasticNet

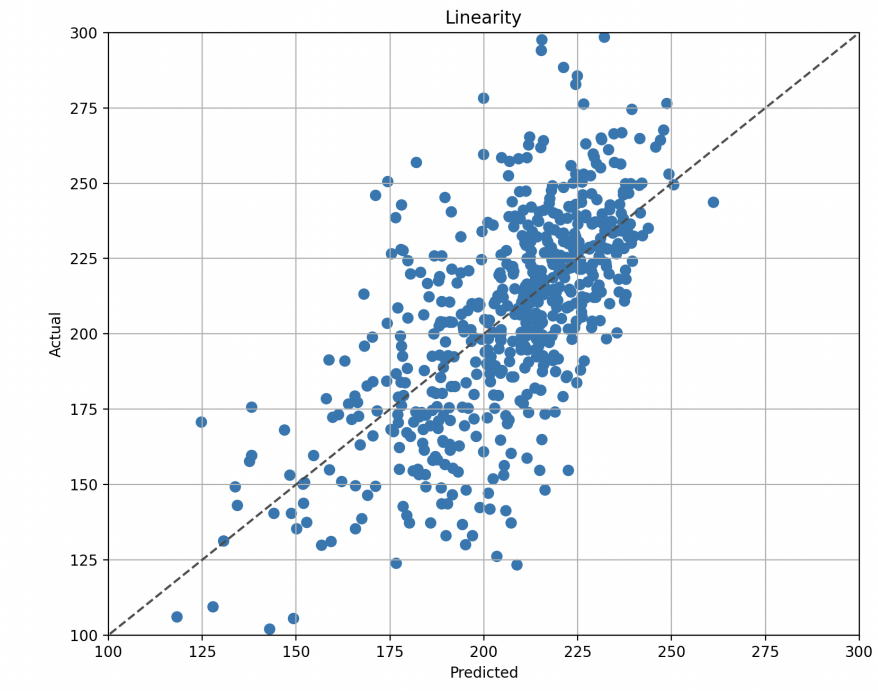
Elasticnet regression combines Ridge and Lasso regression into a single algorithm. The hyperparameters Alpha and L1 ratio, control the penalties imposed on the algorithm. For example, if L1 ratio =0 we have ridge regression, if L1 ratio = 1 we have lasso. ElasticNet should eliminate overfitting and improve overall accuracy.

The same datasets are used across Elasticnet and LinearRegression.

Run before tuning Elastic Net Hyperparameters

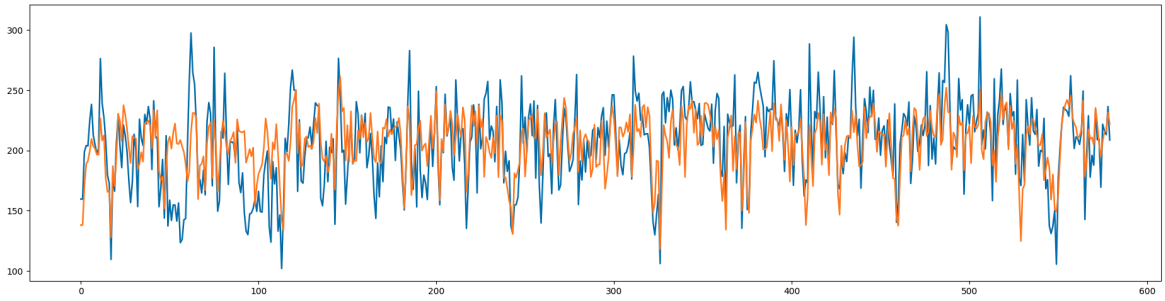
5-minute

Linearity
Groupings are much tighter than before.

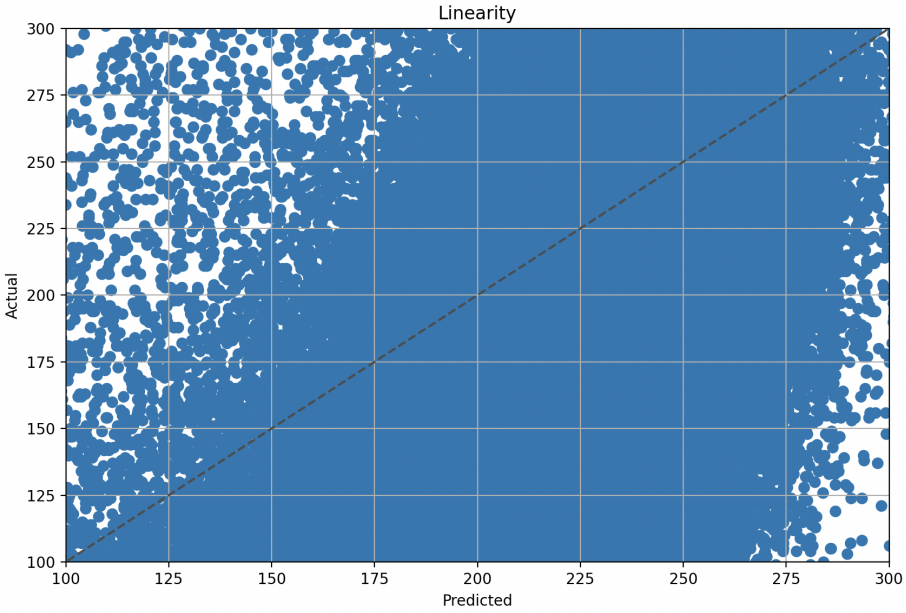


Predictions / Performance

$R^2 = 0.454$

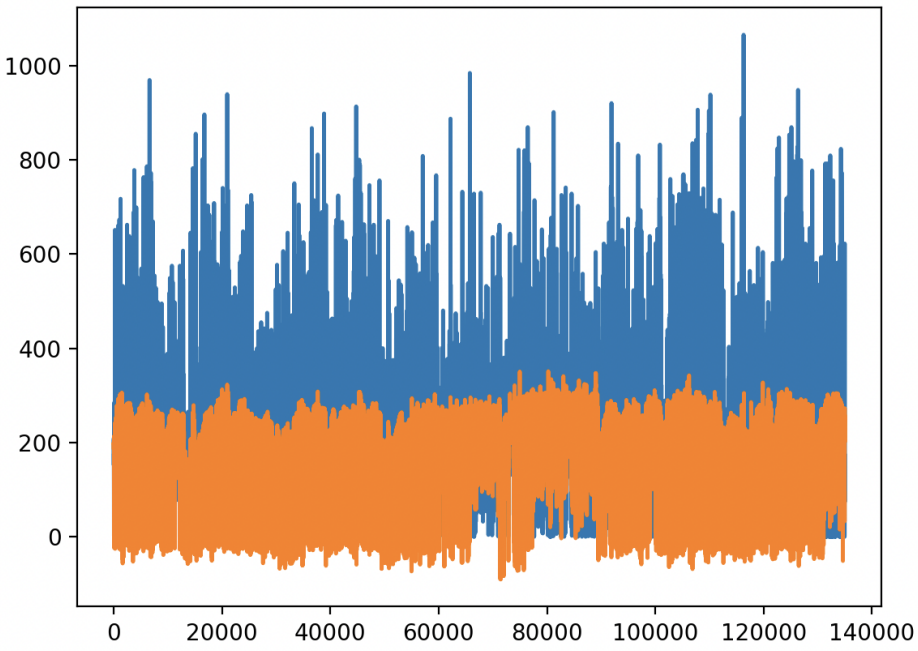


1-Second



Predictions / Performance

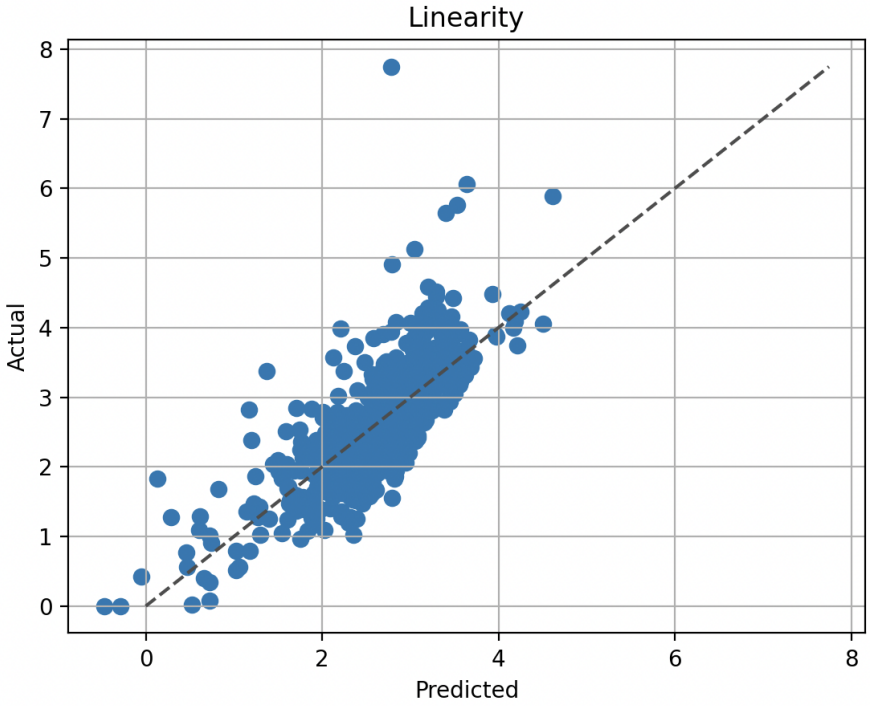
$R^2 = 0.37$



During the process of running the models, it became clear that using power wattage as opposed to power per kilogram was going to be problematic - using a weight-standardised metric was going to be more effective. Moreover, hyperparameter tuning was employed.

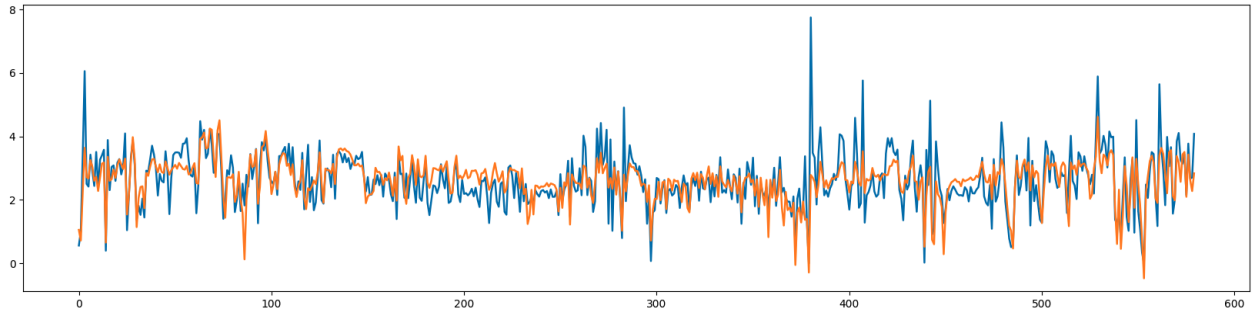
After tuning Elastic Net Hyperparameters Using ElasticNet

1 minute
Watt per kilogram



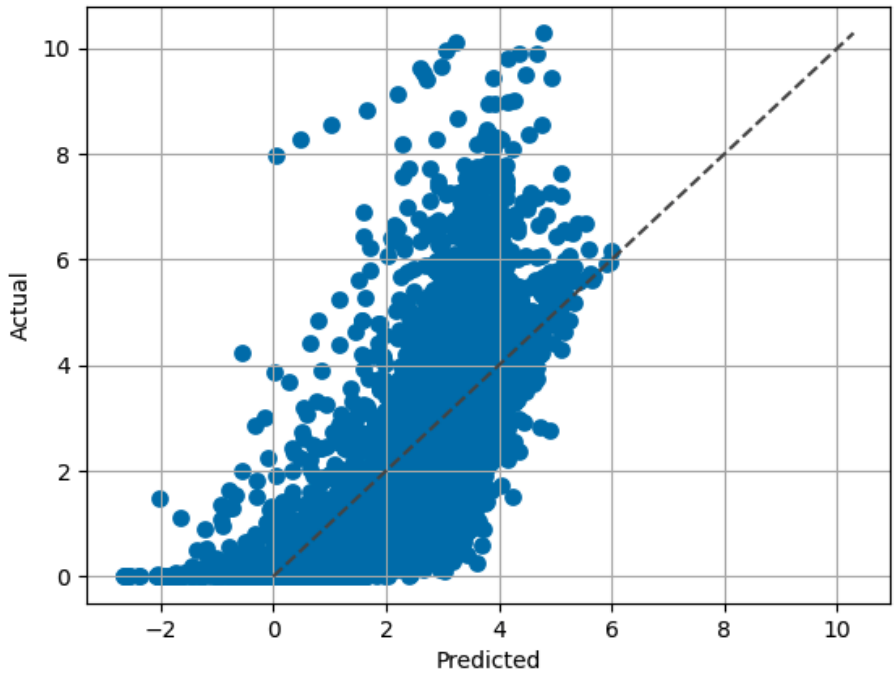
Predictions / Performance

$R^2 = 0.57$

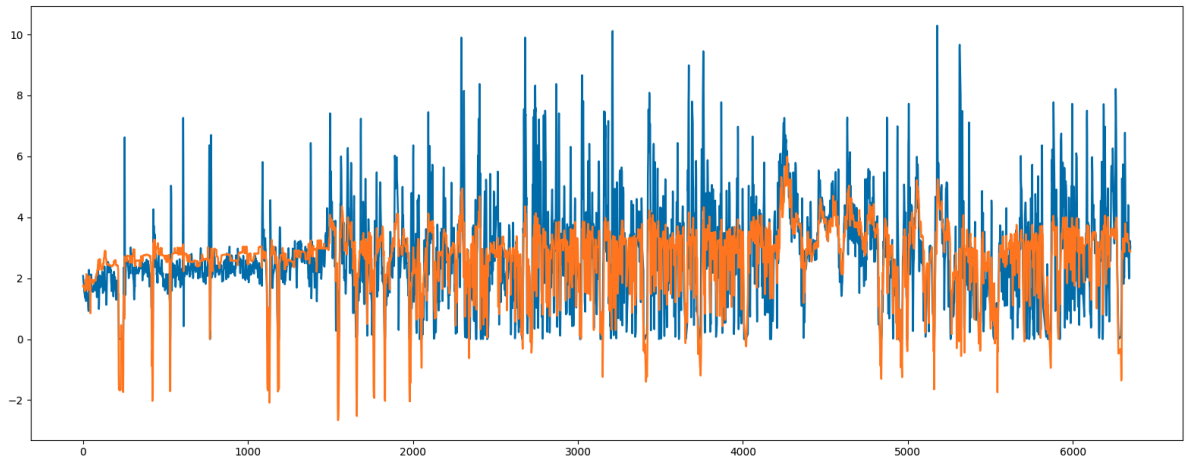


After tuning Elastic Net Hyperparameters Using ElasticNet

1 minute
Watt per kilogram



Predictions / Performance
R^2 = 0.47 (overfitting a problem again…)

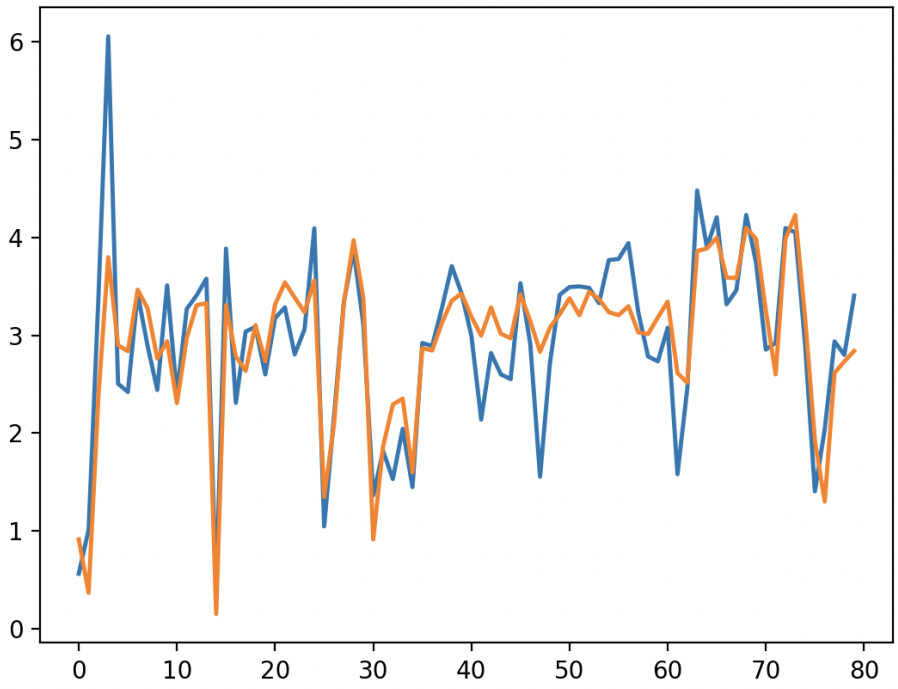


Summary

Whilst ElasticNet increased accuracy after hyperparameter tuning, it still failed when the amount of data (the number of rows) increased. Further work could be conducted to optimise this implementation of ElasticNet; however, the adoption of such an algorithm is limited due to the

dynamics of the problem we are looking to solve. As a further extension and as sklearn's algorithm cheat sheet says, Stochastic Gradient Descent Regressor could be assessed when the number of rows in a given data set exceeds 100k.

It is clear that ElasticNet might be effective when the number of rows/observations is limited to max 100 ($R^2 > 0.75$) see below:



Resources: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html
Resources: https://machinelearningmastery.com/elastic-net-regression-in-python/

Scikit-learn Random Forest - RandomForestRegressor

Random forest is a supervised machine learning algorithm that is constructed from multiple decision trees to create a "forest." It can be used for both classification and regression problems in R and Python. Based on previous results listed further above, Random forest has a default ability to correct for decision trees' habit of overfitting to their training set - This should of course correct a lot of the issues previous models have faced.

Slightly different data sets have been used for the Random Forest implementation > Additional features have been added to the dataset however the length (number of rows) is similar to the Elastic net dataset (n=6350) where the time interval is in seconds and data is derived from a single user.

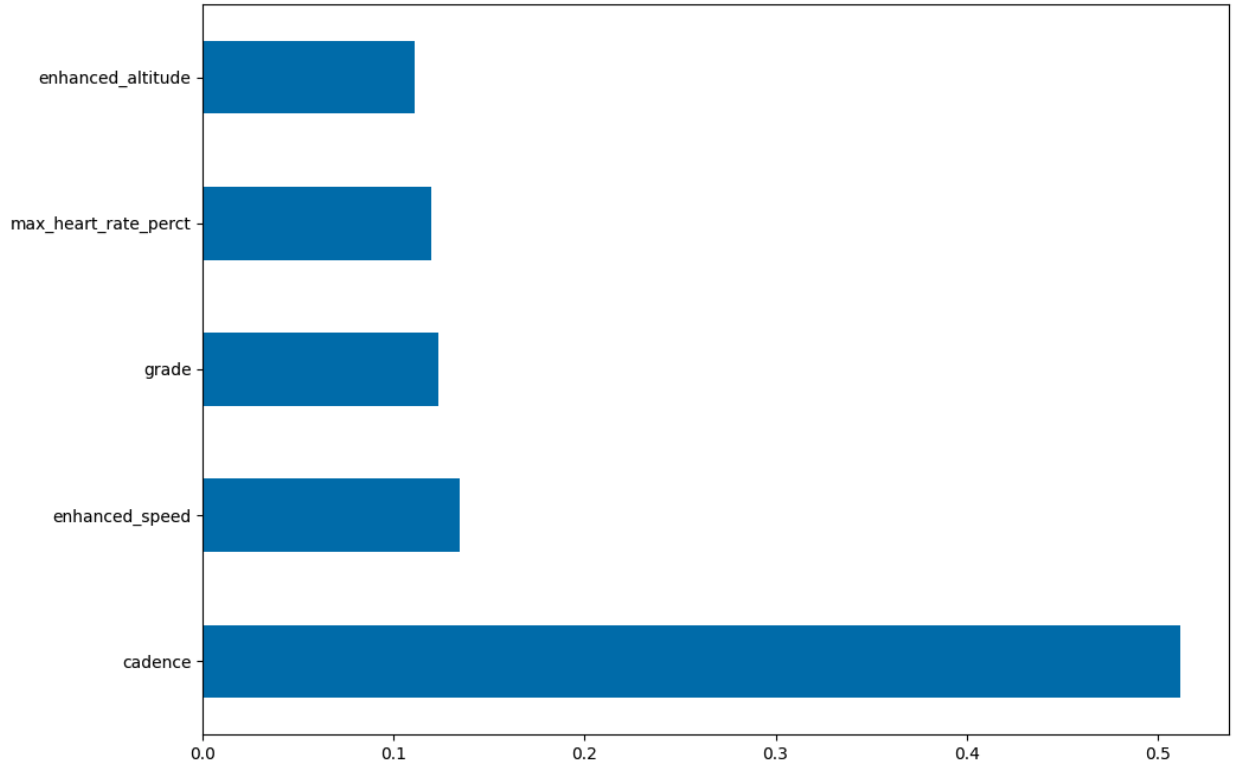
Additionally, a user is offered a choice to either use a short dataset (n=1000) where the time interval is in seconds, a single session dataset (n=112) where the time interval is in minutes or another user's (User 2) data (n= 8,238)

Findings:

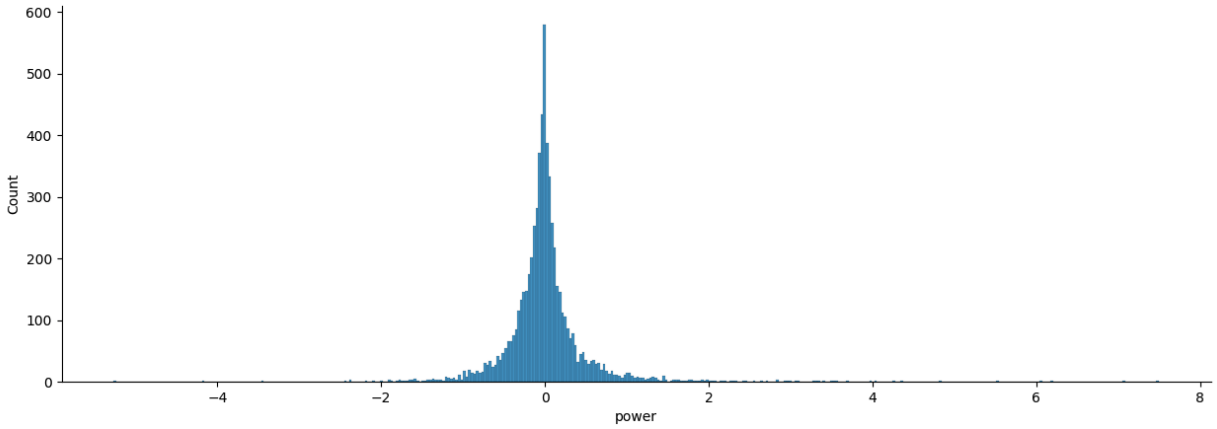
Dataset 1 (n = 6,350)

Correlation Matrix:

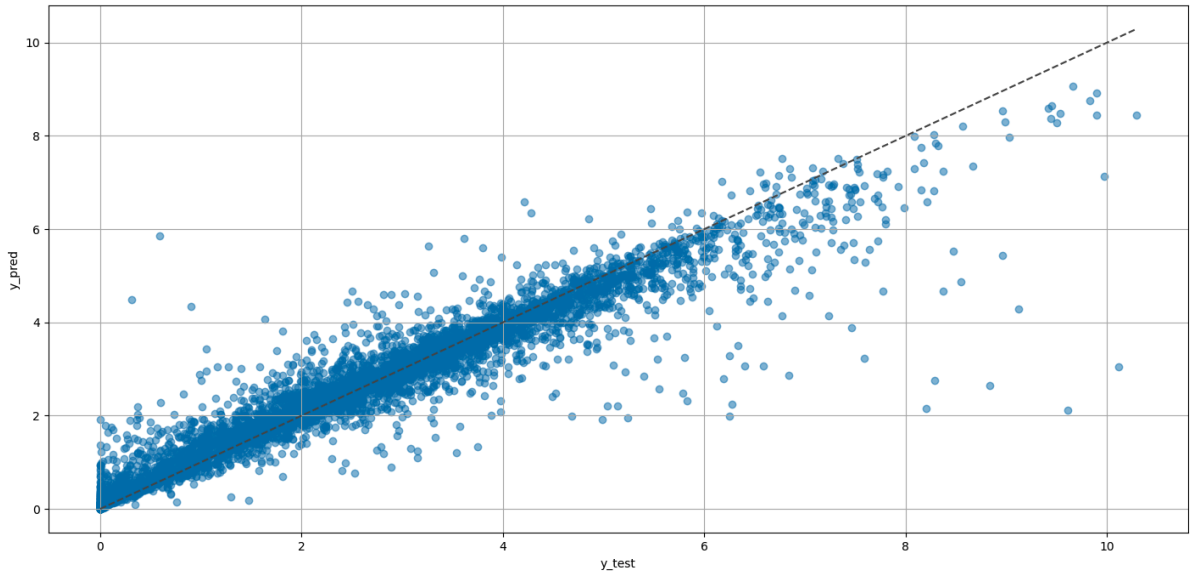
Feature Analysis (Importance):



Distribution Plot (Check model performance)

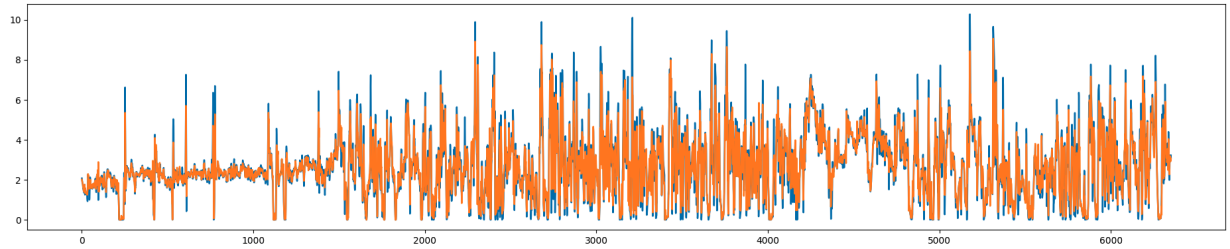


Scatter Plot: Prediction vs Actual



Predictions / Performance *(R^2)*
Accuracy for training dataset: 0.96%
Accuracy for test dataset: 0.69%

Production
Accuracy for production: 0.90%
MAE: 0.26
MSE: 0.27
RMSE: 0.52



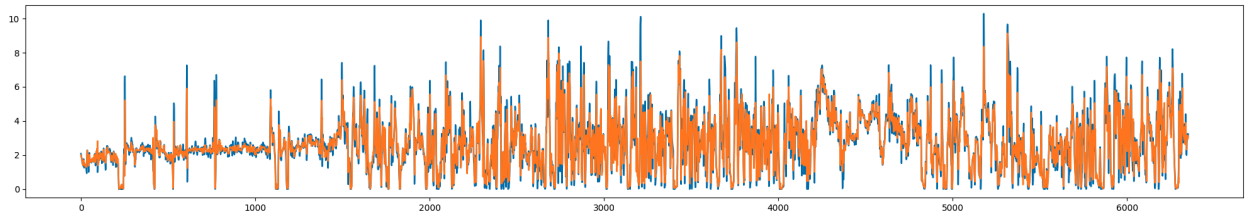
<u>Hyperparameter Tuning - Randomized Search CV</u>
There are two techniques of Hyperparameter tuning i.e
1) RandomizedSearchCv
2) GridSearchCV
We use RandomizedSearchCv because it is much faster than GridSearchCV Number of trees in random forest

Dataset 1 (n = 6,350) After hyperparameter tuning:

Predictions / Performance *(R^2)*

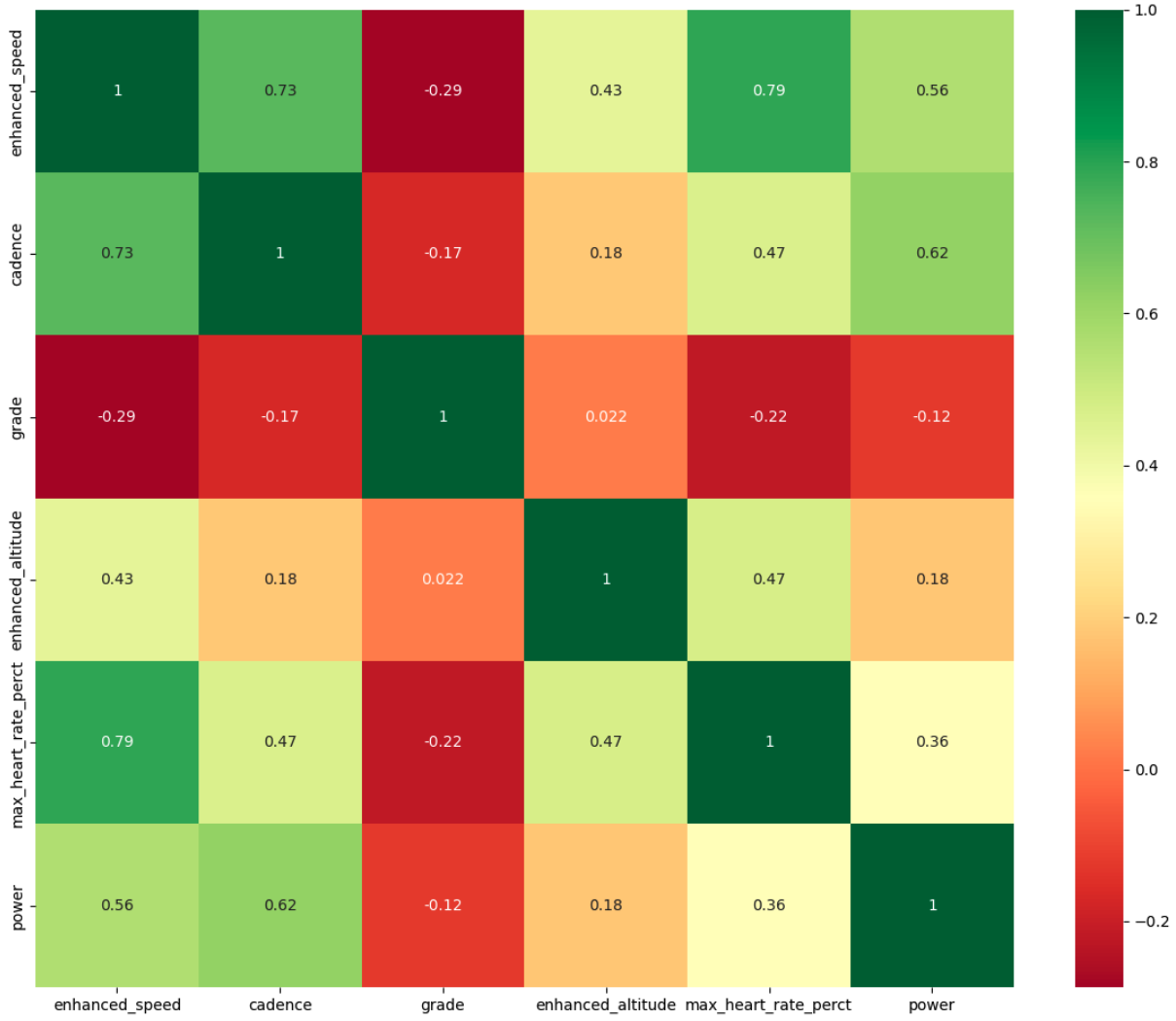
A minor ~1% increase in performance

Production
R^2: 0.91
MAE: 0.28
MSE: 0.25
RMSE: 0.50

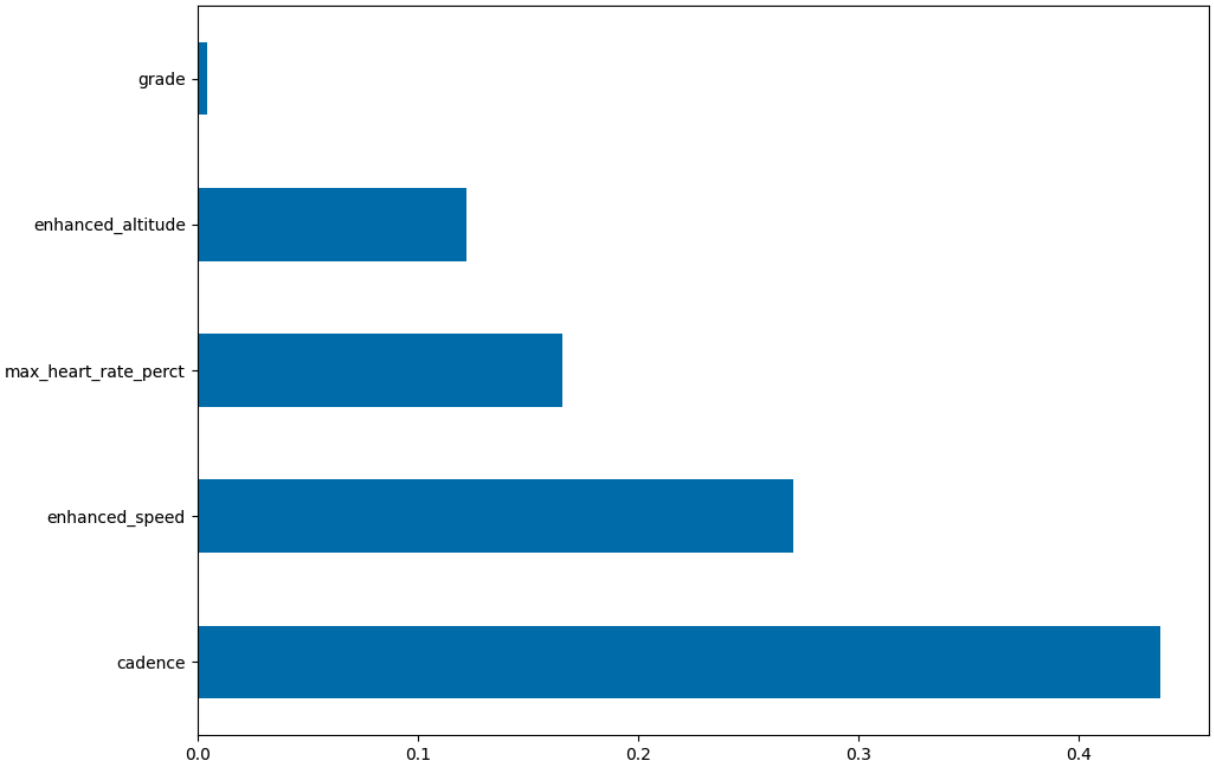


Dataset 1 (n = 1,000) - Shortened

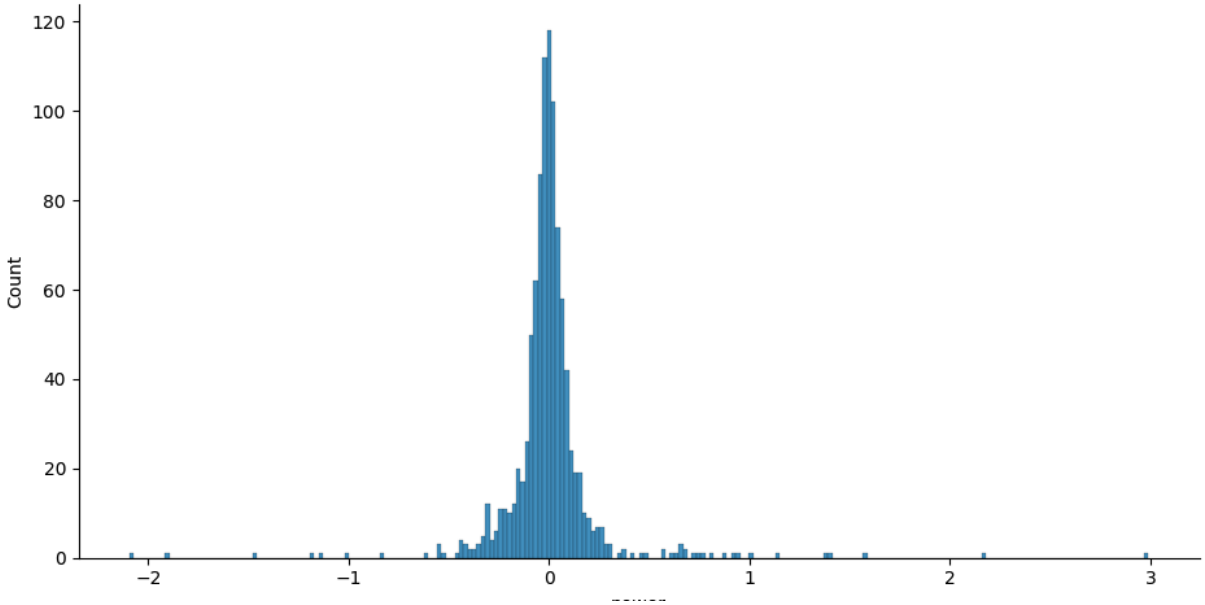
Correlation Matrix: *Strong correlations are not present*



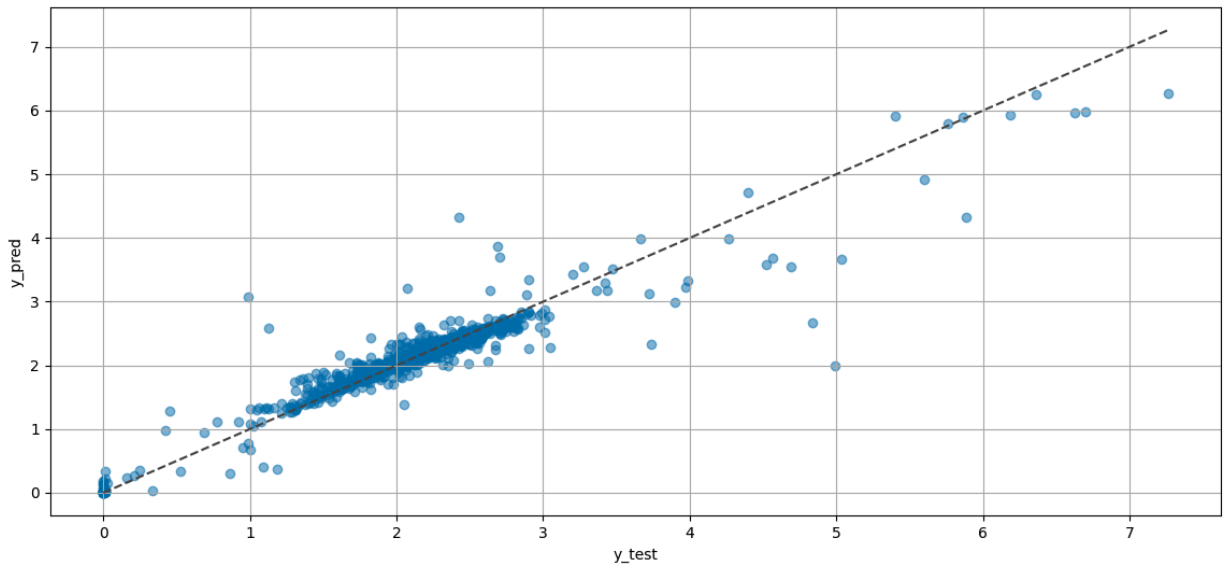
Feature Analysis (Importance): A more distributed shared of feature importance



Distribution Plot (Check model performance)



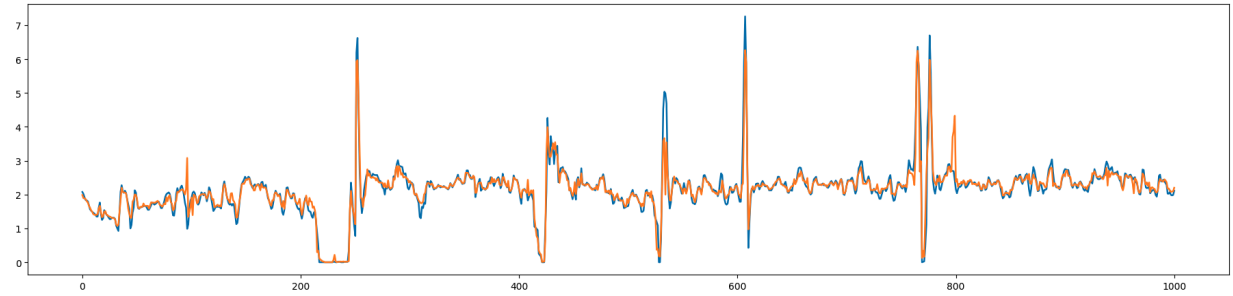
Scatter Plot: Prediction vs Actual



Predictions / Performance *(R^2)*

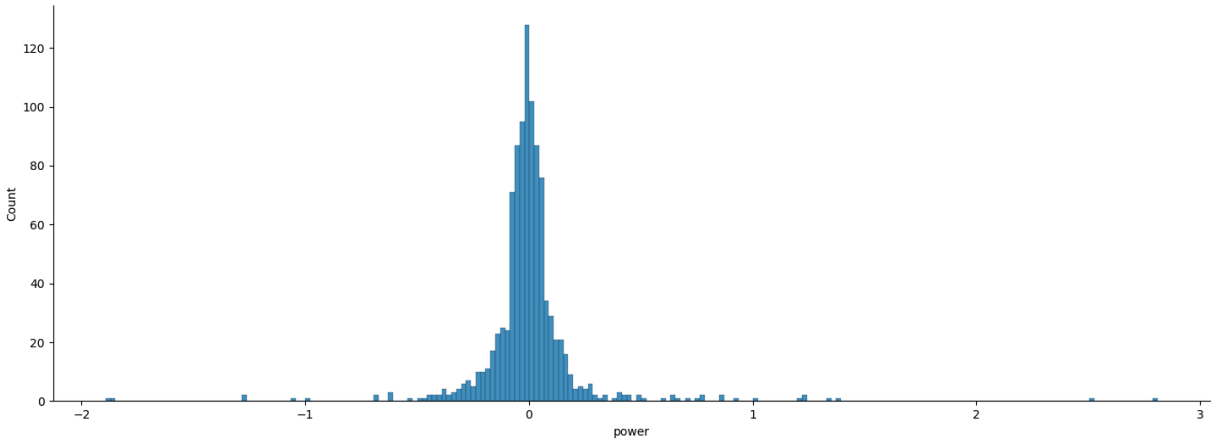
Accuracy for training dataset: 0.95
Accuracy for test dataset: 0.61
Accuracy for production: 0.89

Production
R^2: 0.89
MAE: 0.12
MSE: 0.06
RMSE: 0.25

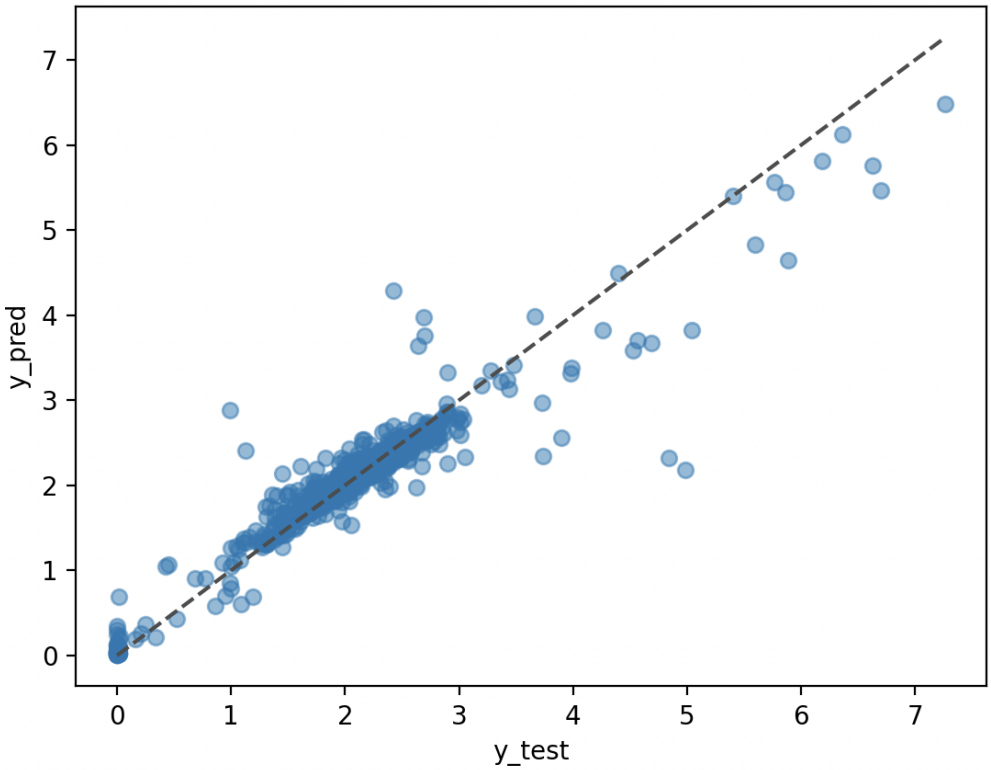


Dataset 1 (n = 1,000) - Shortened

Distribution Plot (Check model performance)

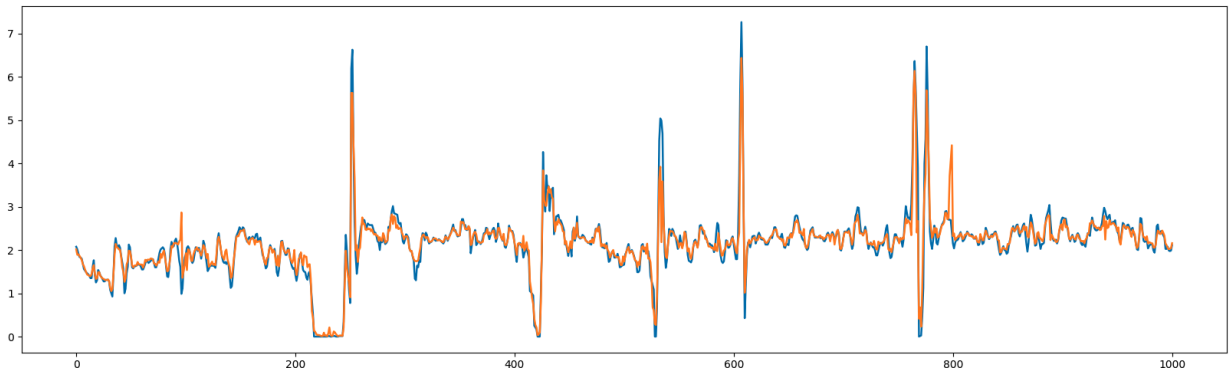


Scatter Plot: Prediction vs Actual



Predictions / Performance *(R^2)*
Slight improvement after tuning:

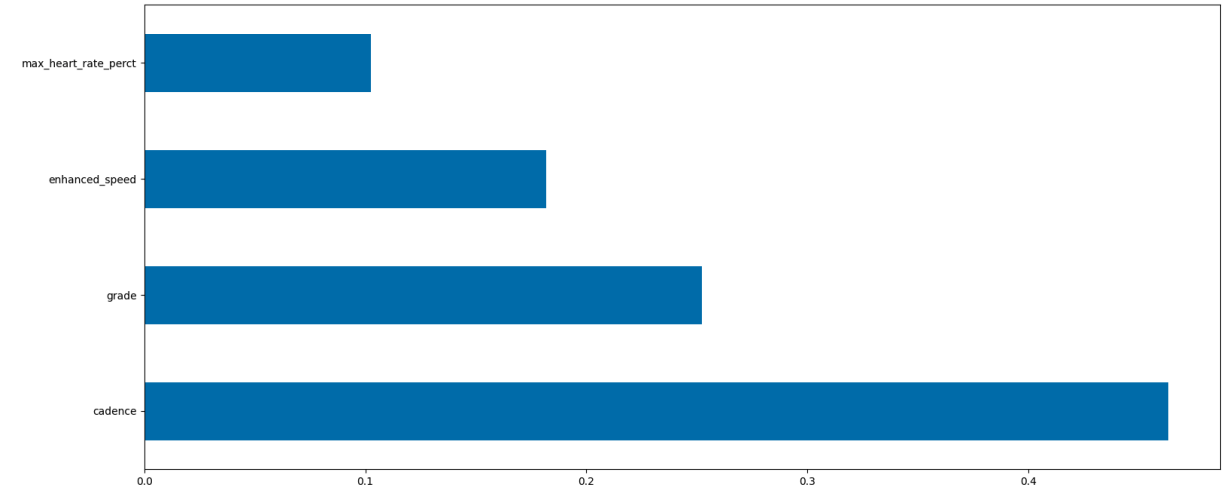
Production
R^2: 0.90
MAE: 0.11
MSE: 0.059
RMSE: 0.24



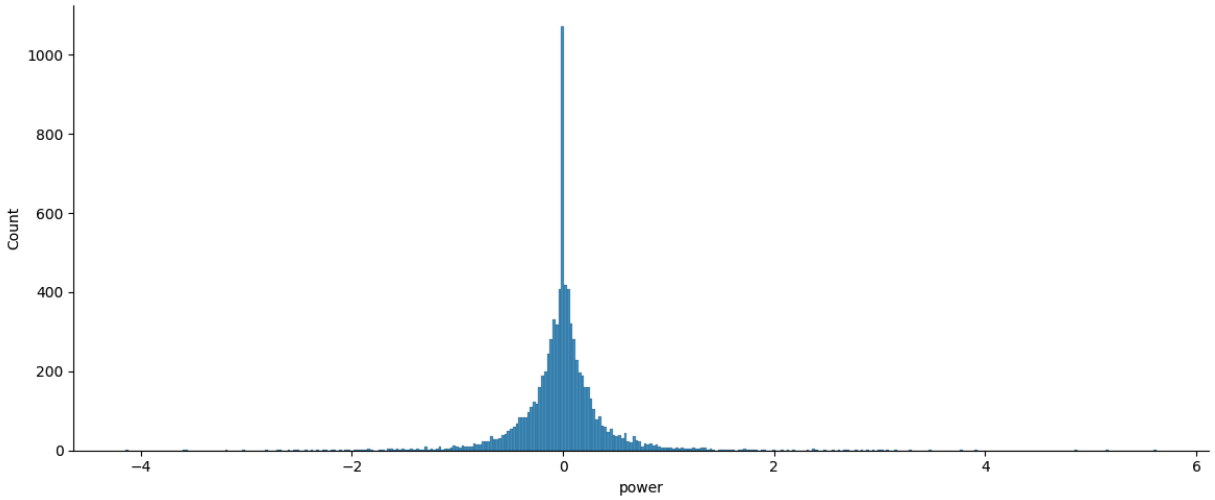
Dataset 2 (n =8,238)

Correlation Matrix:

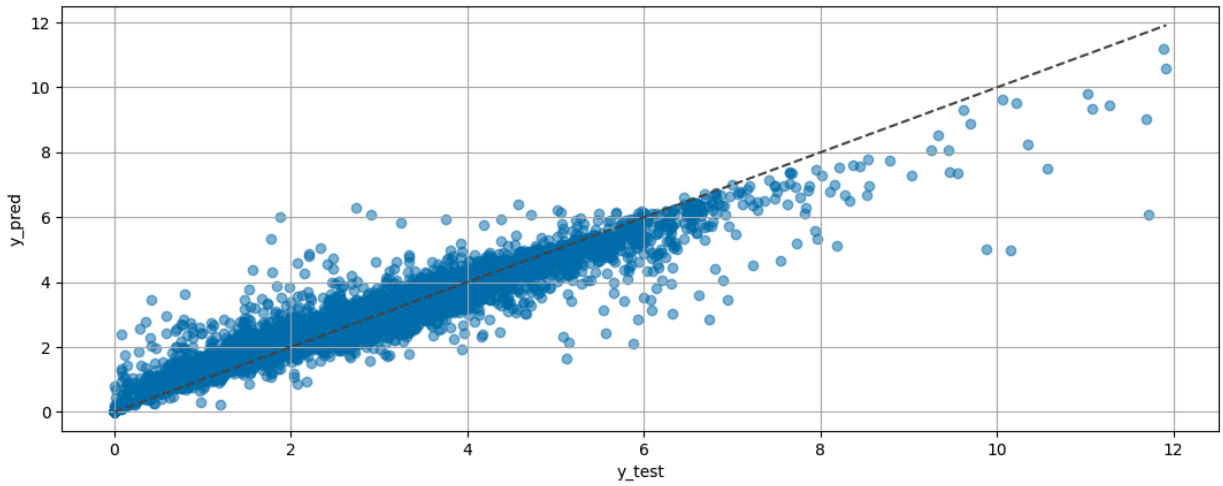
Feature Analysis (Importance):



Distribution Plot (Check model performance)

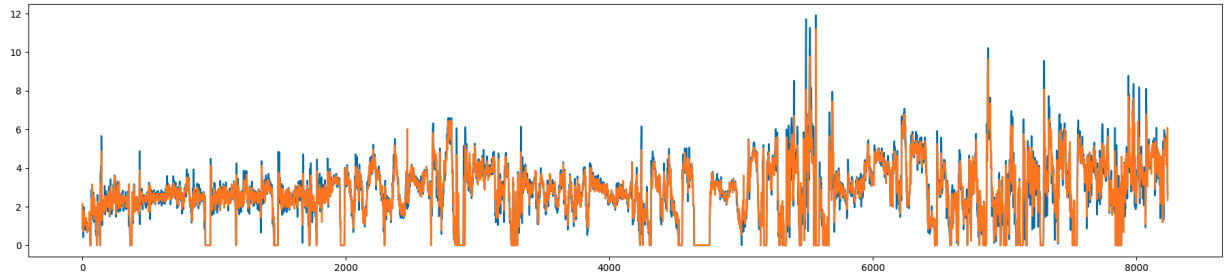


Scatter Plot: Prediction vs Actual



Predictions / Performance *(R²)*
Accuracy for training dataset: 0.96
Accuracy for test dataset: 0.73
Accuracy for production: 0.92

Production
R²: 0.92
MAE: 0.25
MSE: 0.20
RMSE: 0.44

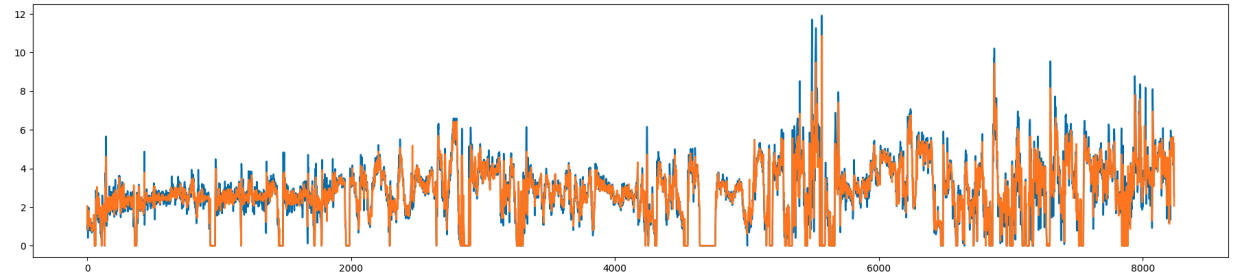


Dataset 2 (n =8,238) After hyperparameter tuning:

Predictions / Performance *(R²)*

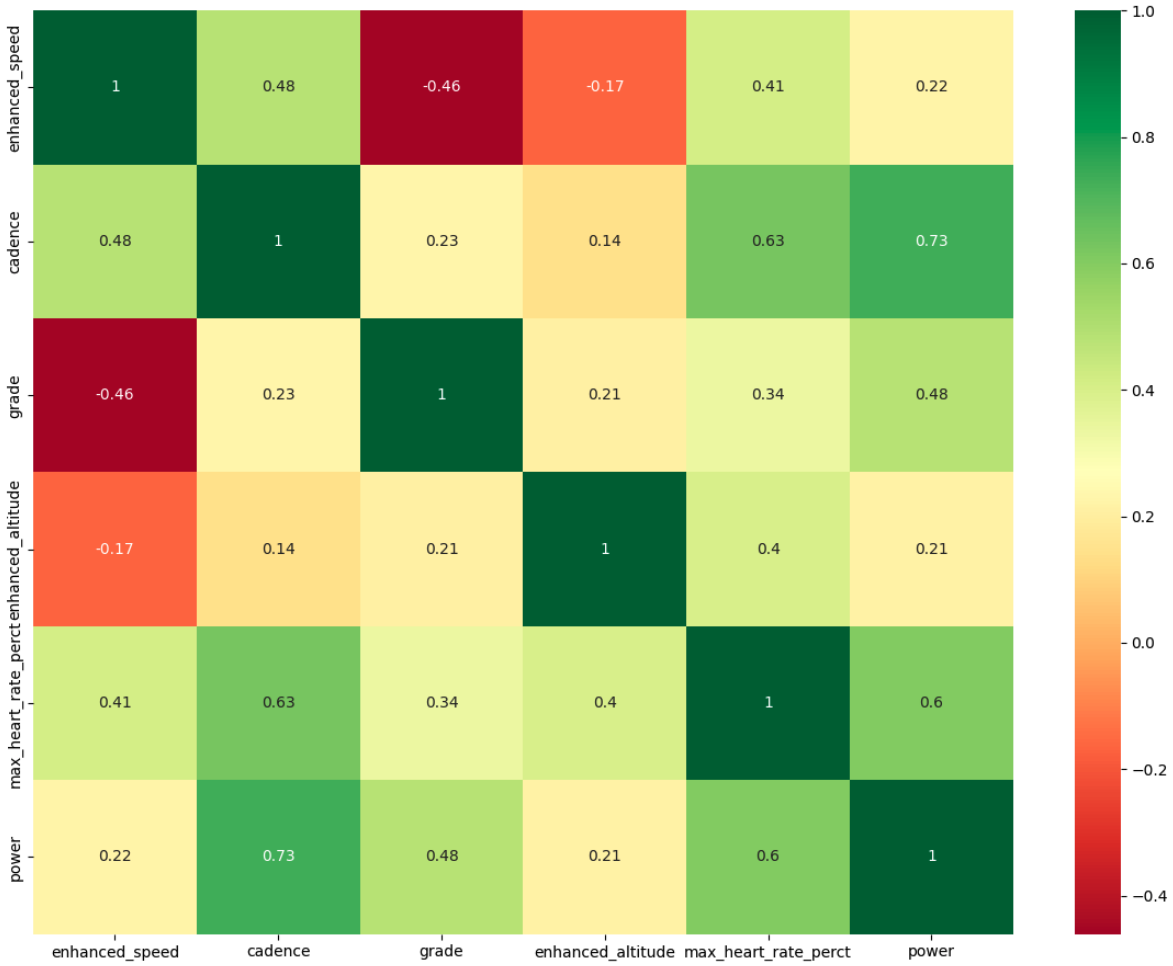
A minor ~1% increase in performance

Production
R²: 0.92
MAE: 0.25
MSE: 0.19
RMSE: 0.43

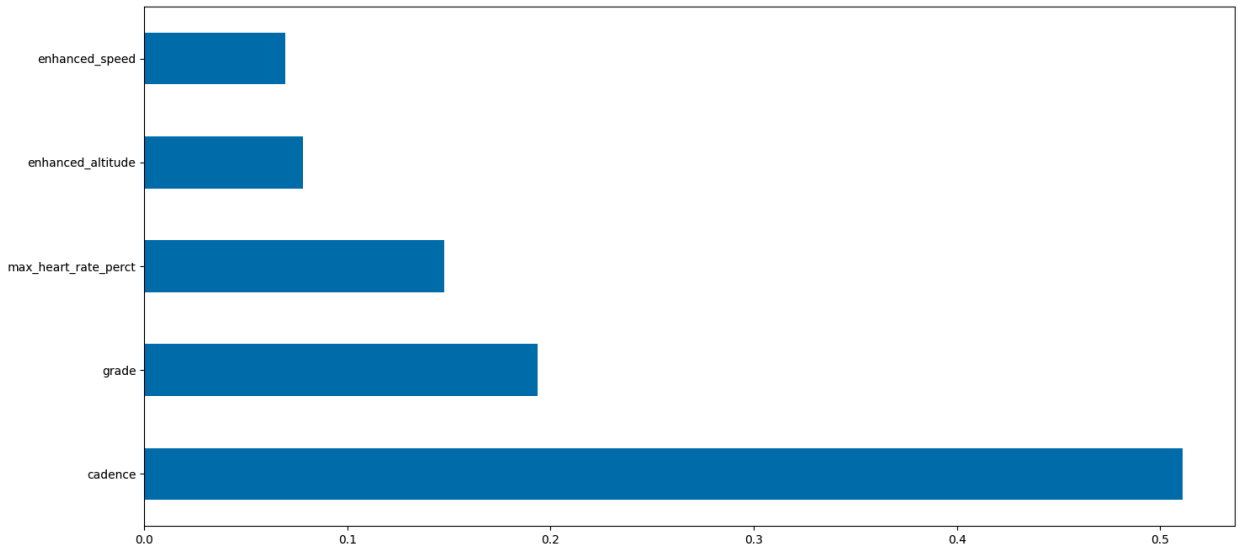


Dataset 3 (n = 112)

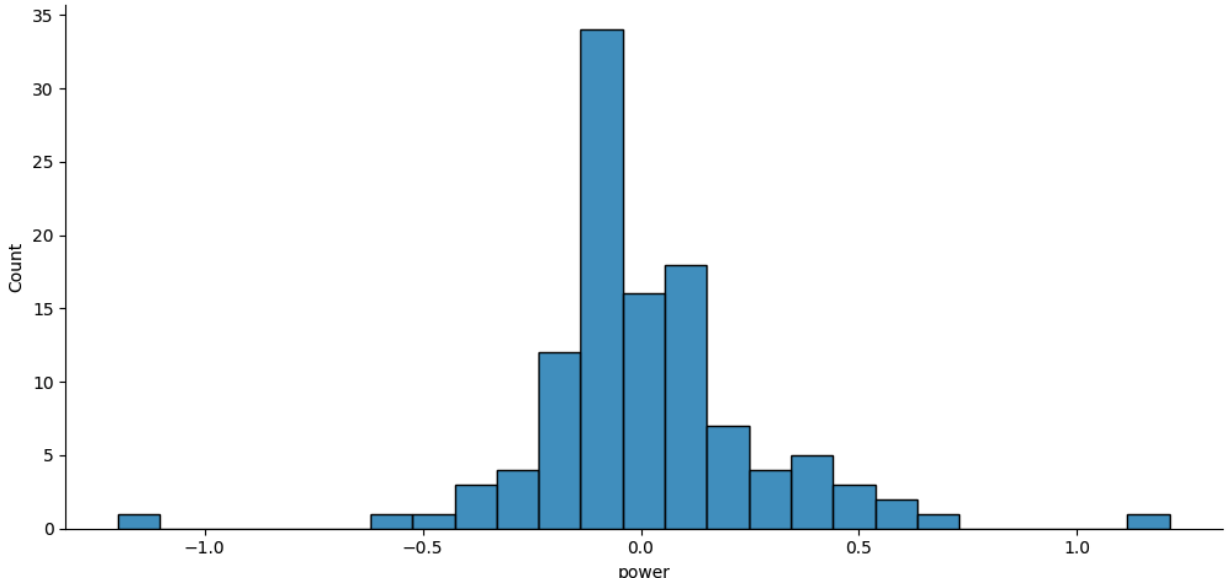
Correlation Matrix:



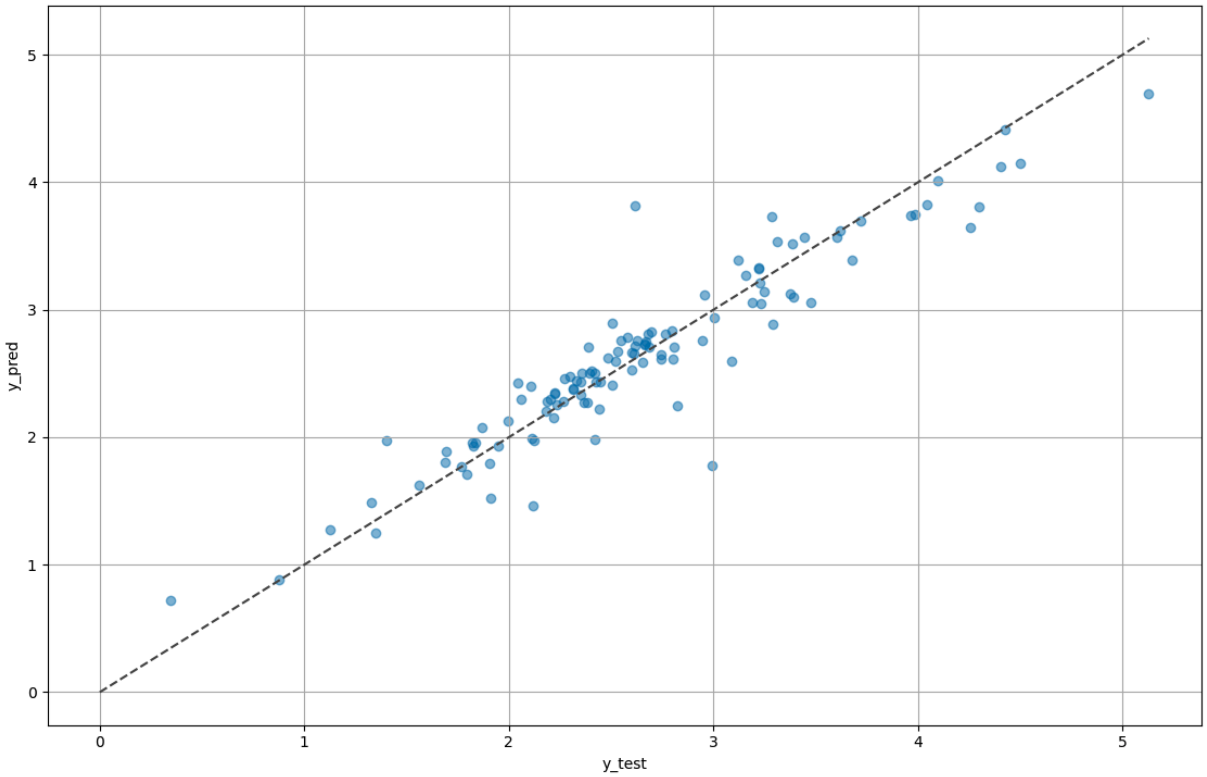
Feature Analysis (Importance):



Distribution Plot (Check model performance)

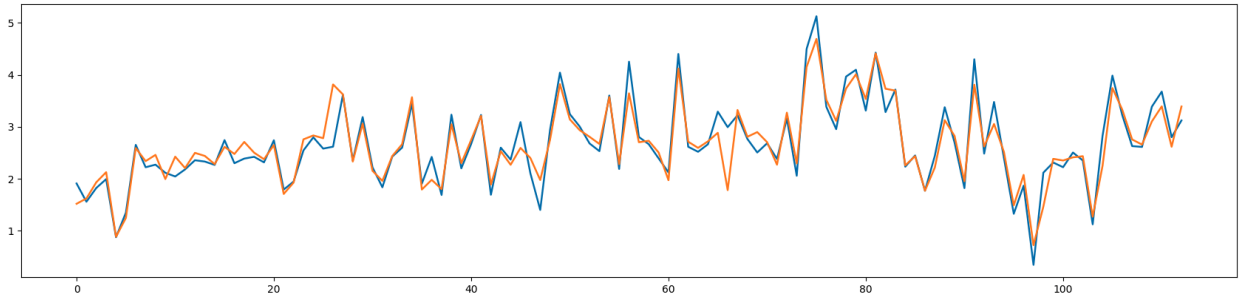


Scatter Plot: Prediction vs Actual



Predictions / Performance *(R^2)*
Accuracy for training dataset: 0.94
Accuracy for test dataset: 0.63
Accuracy for production: 0.88

Production
R^2: 0.88
MAE: 0.18
MSE: 0.074
RMSE: 0.27

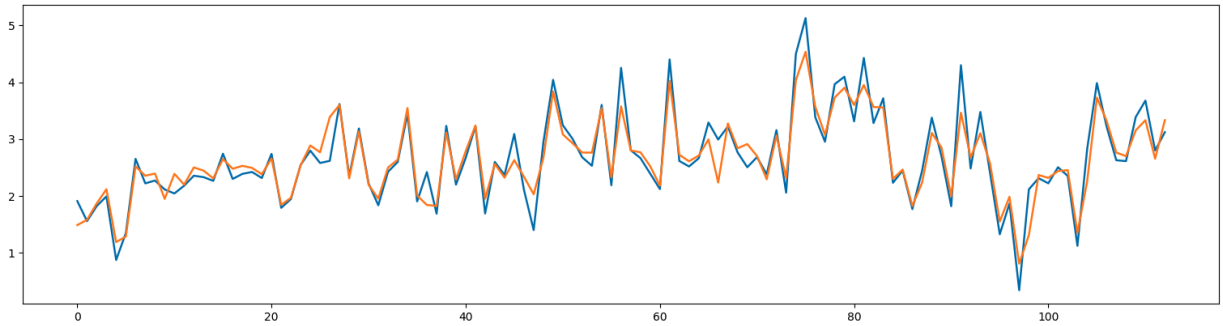


Dataset 3 (n = 112) After hyperparameter tuning:

Predictions / Performance *(R^2)*

A minor ~1% increase in performance

Production
R^2: 0.88
MAE: 0.19
MSE: 0.072
RMSE: 0.26

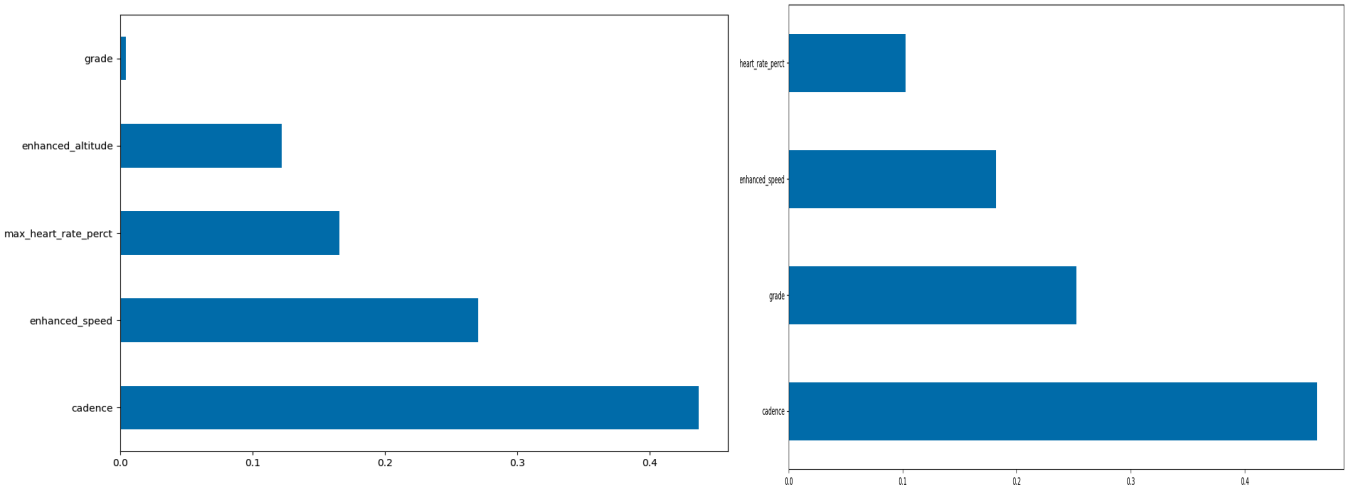


General Summary on Results

Model performance was optimal when nrows >= 1000, and the time interval was set as seconds. Moreover, MAE was optimal when the dataset was 1,000 rows; There is a balance in how much data can be used; too little and accuracy suffers, too much and the gap between predicted and actual begins to open. Therefore, further research and effort need to be invested in pursuing models such as LSTM or ANN (deep learning).

Dataset	User 1	User 1 Short	User 2	Session 131
Nrows	6,350	1,000	8,238	112
R^2	0.91	0.91	0.92	0.88
MAE	0.28	0.11	0.25	0.19
MSE	0.25	0.059	0.19	0.072
RMSE	0.5	0.24	0.43	0.26

Interestingly, feature ranking did differ from User (User 1 left) to User (User 2 right): Both had cadence as the key feature however grade was ranked last in one instance vs second in another. This shows how random forest works to weigh features - Both models achieved similar performance levels.



Summary

To ensure some consistency across all efforts, the Session 131 dataset was used across all three models, results below; As expected Random Forest was the strongest performer; this comes to problem statement and model sit - Random Forest met the basic requirements for being able to effectively predict power out for a given Cyclist's workout session using a range of other cycling performance metrics.