

Configuration Manual

MSc Research Project Programme Name

Busra Sevinc Student ID: X22183957

School of Computing National College of Ireland

Supervisor:

Dr. Muslim Jameel Syed

National College of Ireland

MSc Project Submission Sheet



School of Computing

Student Name:	Busra Sevinc
Student ID:	X22183957
Programme:	MSC Artificial Intelligence Year: 2024
Module:	Practicum (MSCAI1)
Lecturer:	Dr. Muslim Jameel Syed
Due Date:	12.08.2024
Project Title:	Enhancing Wind Turbine Longevity: A Comparative Study of Deep Learning and Traditional Machine Learning Techniques for Predicting Remaining Useful Life
Word Count:	Page Count:

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

<u>ALL</u> internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:

Bullto

Date: 11.08.2024.....

.....

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple	
copies)	
Attach a Moodle submission receipt of the online project	
submission, to each project (including multiple copies).	
You must ensure that you retain a HARD COPY of the project, both	
for your own reference and in case a project is lost or mislaid. It is not	
sufficient to keep a copy on computer.	

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only		
Signature:		

Date:	
Penalty Applied (if applicable):	

Configuration Manual

Forename Surname Student ID:

1 Introduction

This configuration manual outlines the methodologies, system requirements, and data manipulation methods employed in the study titled "Enhancing Wind Turbine Longevity". The venture uses deep learning models as well as typical machine learning to forecast how long any part of a wind turbine can be used before it becomes ineffective; this will thus concentrate on improving maintenance strategies and operational efficiency. According to predictive maintenance research done in this field, machine learning greatly enhances both the reliability and accuracy of Remaining Useful Life (RUL) estimations (Teng et al., 2016; Elasha et al., 2019). In order to ensure that predictions are strong enough for validation purposes robustly validated against other models, evaluation metrics such as Mean Absolute Error (MAE) are used for measuring performance. This manual facilitates serves as a thorough guidebook for academics or practitioners, making it easier to replicate the study's computer environment and model implementation.

2 Environment Evaluation

The study was conducted using a personal computer with the following specifications:

- **Operating System:** Windows 11 Home 64-bit (10.0, Build 22621) (22621.ni_release.220506-1250), with regional language settings in English.
- System Manufacturer: ASUSTeK COMPUTER INC.
- System Model: ASUS TUF Dash F15 FX517ZE_FX517ZE
- **BIOS Version:** FX517ZE.315 (type: UEFI)
- **Processor:** 12th Gen Intel(R) Core(TM) i5-12450H (12 CPUs), operating at ~2.0GHz.
- **Memory:** A total of 16GB (16384MB) RAM was installed with 16006MB available for OS tasks.
- Page File Memory: A total of 10116MB was used, with 14545MB available.
- **DirectX Version:** DirectX 12

3 System Requirements

- **Operating System:** Compatible with Windows, macOS, or Linux.
- **Processor:** Intel i5 or equivalent.
- **RAM:** 8 GB or higher recommended.
- Storage: At least 10 GB of free space for data handling and processing.

4 Data Source and Simulation

Background:

The goal of this study is to forecast the Remaining Useful Life (RUL) of wind turbine bearings, which are essential for reducing turbine downtime, using simulated data.

Data Simulations Process and Tools:

To model and analyse bearing stress, wear, and operational parameters including load, lubrication, vibration, and temperature, simulations make use of MATLAB/Simulink and ANSYS tools. This comprises:

- MATLAB/Simulink: For dynamic load modelling.
- ANSYS Mechanical and CFD: For stress and fluid dynamics simulations.
- ANSYS DesignXplorer: For optimization and parameter analysis.
- **Data Set Description:** The dataset "Wind Turbine Rotor RUL Prediction Data" includes: Vibration Level, Rotor Temperature, Operational Hours, Maintenance Frequency: Operational metrics of turbine rotors.
- RUL (Remaining Useful Life): Predictive metric for maintenance scheduling.

Purpose and Use of Data:

Model training and validation are supported by the simulated data, which improves the predicted accuracy of maintenance plans and operational scheduling.

5 Data Preparation and Analysis

Figure 1: Import Libraries



Figure 1 shows a snapshot of a Python script that imports different libraries for activities like preprocessing, profiling, data manipulation, visualization, and creating machine learning models.

Figure 2	2: Data	Load
----------	---------	------

<pre># Load the insurance data file df = pd.read_excel('Wind_Turbine_Rotor_RUL_Prediction_Data_Large.xlsx', sheet_name='Data') pd.set_option('display.max_columns', None) df.head(5) </pre>					
	Vibration Level	Rotor Temperature	Operational Hours	Maintenance Frequency	RUL
0	13.179846	54.156811	0.000000	2	20014.044729
1	11.957460	80.612359	0.200002	0	19799.613037
2	11.868327	62.389137	0.400004	2	20018.527587
3	8.370583	49.259427	0.600006	1	19966.434739
4	9.497515	56.561055	0.800008	1	19947.663786

According to Figure 2, the wind turbine rotor RUL prediction data is loaded into a pandas DataFrame from an Excel file titled

'Wind_Turbine_Rotor_RUL_Prediction_Data_Large.xlsx' with all columns displayed for initial inspection.

Figure 3: Data Information

d ✓	f.info() 0.0s			
<cla Range Data</cla 	ss 'pandas.core.frame.Da eIndex: 100000 entries, columns (total 5 column	ataFrame'> 0 to 99999 ns):		
#		Non-Null Count	Dtypo	
#	Cordinit	NOIT-NULL COUNC	осуре	
		40000011		
0	Vibration Level	100000 non-null	†10at64	
1	Rotor Temperature	100000 non-null	float64	
2	Operational Hours	100000 non-null	float64	
3	Maintenance Frequency	100000 non-null	int64	
4	RUL	100000 non-null	float64	
dtypes: float64(4), int64(1)				
memory usage: 3.8 MB				

Figure 3 displays the output from df.info() for a pandas DataFrame, listing 100,000 entries across five columns with data types and memory usage detailed.

Figure 4: Data Description

# N df. ✓ 0.0	lumeric values describe())s				
	Vibration Level	Rotor Temperature	Operational Hours	Maintenance Frequency	RUL
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	10.007821	60.036388	10000.000000	1.999110	10039.922698
std	1.995092	9.993803	5773.589295	1.412533	5774.731116
min	0.295765	13.054122	0.000000	0.000000	0.000000
25%	8.667228	53.250352	5000.000000	1.000000	5041.871956
50%	10.007524	60.052501	10000.000000	2.000000	10039.482530
75%	11.349981	66.749050	15000.000000	3.000000	15035.904843
max	18.483544	102.858556	20000.000000	11.000000	20596.895886

Figure 4 displays the output from df.describe() for a DataFrame, summarizing statistics like count, mean, standard deviation, min, quartiles, and max for columns including Vibration Level, Rotor Temperature, Operational Hours, Maintenance Frequency, and RUL.

5.1 Exploratory Data Analysis

Figure 5: Identifying Missing Values

<pre># Missing values per column print(df.isnull().sum())</pre>				
✓ 0.0s				
Vibration Level	0			
Rotor Temperature	0			
Operational Hours	0			
Maintenance Frequency	0			
RUL	0			
dtype: int64				

Figure 5 shows how many missing values there are in the data.

Figure 6: Identifying Outliers

Figure 6 indicates the range of the Vibration Level, Rotor Temperature, Operational Hours columns.

Figure 7: Identifying Outliers



Figure 7 indicates the range of the Maintenance Frequency, Remaining Useful Life(RUL) columns.

Figure 8: Identifying Outliers

```
# Capping outliers for Vibration Level
vibration_cap_upper = np.percentile(df['Vibration Level'], 99)
vibration_cap_lower = np.percentile(df['Vibration Level'], 1)
df['Vibration Level'] = np.clip(df['Vibration Level'], vibration_cap_lower, vibration_cap_upper)
# Keeping Maintenance Frequency and RUL outliers but reviewing extreme cases
df.loc[df['Maintenance Frequency'] > 10, 'Maintenance Frequency'] = np.nan
# Re-checking min and max values after adjustments
print('Adjusted Min and Max Vibration Level:', df['Vibration Level'].min(), df['Vibration Level'].max())
print('Maintenance Frequencies greater than expected:', df['Maintenance Frequency'].max())
print('RUL range:', df['RUL'].min(), df['RUL'].max())
</ O.0s
Adjusted Min and Max Vibration Level: 5.3390452129367905 14.661037438630405
Maintenance Frequencies greater than expected: 10.0
RUL range: 0.0 20596.89588614355
```

Figure 8 shows the limitation for the vibration level.

6 Data Preparation

In the report's data preparation part, it is shown different ways for dealing with outliers in variables like Rotor Temperature, Vibration Level, Operational Hours, Maintenance Frequency and RUL. One such way is calculating statistical thresholds through standard deviations as well as percentiles for outlier detection and removal thus ensuring robustness of the data for further analysis.

Figure 9: Handling Outlier Values for Rotor Temperature



Figure 9 deletes the data outside the Rotor Temperature column.

Figure 10: Handling Outlier Values for Vibration Level



Figure 10 shows the data outside the Rotor Temperature column.

Figure 11: Handling Outlier Values for Vibration Level



Figure 11 shows the data outside the Rotor Temperature column.

Figure 12: Handling Outlier Values for Operational Hours

	Operational Hours
27]	<pre>df['Operational Hours'] = np.log1p(df['Operational Hours']) print('Adjusted Operational Hours:', df['Operational Hours'].min(), df['Operational Hours'].max()) <!-- 0.0s</pre--></pre>
	Adjusted Operational Hours: 0.0 9.90353755128617
.28]	<pre># Outlier Removal # Calculate the 1st and 99th percentiles cap_lower = np.percentile(df['Operational Hours'], 1) cap_upper = np.percentile(df['Operational Hours'], 99) # Removing outliers outside the 1st and 99th percentile range df = df[(df['Operational Hours'] >= cap_lower) & (df['Operational Hours'] <= cap_upper)] print('Number of rows after removing Operational Hours outliers:', len(df)) print('Adjusted Operational Hours:', df['Operational Hours'].min(), df['Operational Hours'].max()) \$\square\$ 0.06\$</pre>
	Number of rows after removing Operational Hours outliers: 85722

Figure 12 deletes the data outside the outlier in the operational hours column to make accurate predictions.

Figure 13: Handling Outlier Values for Maintenance Frequency



Figure 13 deletes the data outside the outlier in the maintenance frequency column to make accurate predictions.

Figure 14: Handling Outlier Values for RUL



Figure 14 deletes the data outside the outlier in the RUL column to make accurate predictions.

7 Model Building

This part of the report is dedicated to building a model. Here we explain how to scale features, divide data into training and testing sets and use different machine learning models. We consider such specific models as Linear Regression and Random Forest, Gradient Boosting, Support Vector Regression, XGBoost and LSTM Regressor. Each model is described in terms of its configuration, training and evaluation processes where mean squared error (MSE), mean absolute error (MAE) and R-squared metrics are applied for measuring performance. Therefore this passage represents a step-by-step methodology that can be used in predicting RUL for wind turbine components with advanced regression techniques.

Figure 15: Scaling Dataframe

S	Scaling DataFrame (Min Max Scaling, Standard Scaling, etc)						
34]	<pre># Initialize the scaler scaler = StandardScaler() # List of columns to scale columns_to_scale = ['Vibration Level', 'Rotor Temperature', 'Operational Hours', 'Maintenance Frequency', 'RUL'] # Apply scaling df[columns_to_scale] = scaler.fit_transform(df[columns_to_scale]) <!-- 0.0s</th--></pre>						
	df.	.head <mark>(</mark> 5)					
	✓ 0.0)s					
		Vibration Level	Rotor Temperature	Operational Hours	Maintenance Frequency	RUL	
	1015	0.198914	2.040458	-4.406076	-1.442543	1.728538	
	1026	-0.328444	0.987493	-4.393111	-1.442543	1.731350	
	1039	0.104016	2.136752	-4.377965	-1.442543	1.727826	
	1083	-0.552993	0.875271	-4.328067	-1.442543	1.730161	
	1090	0.882560	-0.312080	-4.320316	-1.442543	1.727180	

Figure 15 shows the scaling of the Dataframe.

Figure 16: Data Splitting



Figure 16 shows the data splitting of the Dataframe.

```
Figure 17: Linear Regression and Random Forest
```

```
# Initialize the models
lr model = LinearRegression()
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
# Train the models
lr_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
# Predict on the test set
y_pred_lr = lr_model.predict(X_test)
y_pred_rf = rf_model.predict(X_test)
# Evaluate the model
mae_lr = mean_absolute_error(y_test, y_pred_lr)
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_lr = r2_score(y_test, y_pred_lr)
r2_rf = r2_score(y_test, y_pred_rf)
# Print evaluation metrics with four decimal points
print(f"Linear Regression MAE: {mae lr:.4f}"
print(f"Random Forest MAE: {mae_rf:.4f}")
print(f"Linear Regression MAE: {mse_lr:.4f}")
print(f"Random Forest MAE: {mse_rf:.4f}")
print(f"Linear Regression MAE: {r2_lr:.4f}")
print(f"Random Forest MAE: {r2_rf:.4f}")
```

Figure 17 shows the application of Linear Regression and Random Forest models.

Figure 18: Gradient Boosting



Figure 18 shows the application of Gradient Boosting model.

Figure 19: Support Vector Regressor

```
Support Vector Regressor
# Initialize the Support Vector Regressor
svr_model = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=0.1)
# Train the model
svr_model.fit(X_train, y_train)
# Predict on the test set
y_pred_svr = svr_model.predict(X_test)
# Evaluate the model
mse_svr = mean_squared_error(y_test, y_pred_svr)
r2_svr = r2_score(y_test, y_pred_svr)
mae_svr = mean_absolute_error(y_test, y_pred_svr) # Calculating MAE
# Print evaluation metrics
print(f"Support Vector Regression MAE: {mse_svr:.4f}")
print(f"Support Vector Regression MAE: {mae_svr:.4f}") # Display MAE
```

Figure 19 shows the application of Support Vector Regressor model.

Figure 20: XGBoost Regression



Figure 20 shows the application of XGBoost Regression model.

Figure 21: LSTM Regressor



Figure 21 shows the application of LSTM Regressor model.

8 Model Evaluation and Comparison

In this part, we give a close look to many predictive models : Linear Regression, Random Forest, Gradient Boosting, Support Vector Regression (SVR), XGBoost and LSTM. They are assessed with the help of several key performance metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE) and R² for determining their prediction ability on Remaining Useful Life (RUL). GridSearchCV is applied as a technique to improve Random

Forest model's performance greatly. For each model, we use visualizations to show actual vs. predicted values of RUL which makes it easy to comprehend their predictive powers at one glance. Furthermore, in doing feature importance analysis especially with Random Forests; critical variables affecting forecasts can be established easily. Plotting MAE scores across different models helps us know which one would work best in practice through a comprehensive comparative study.

Figure 22: Linear Regression and Random Forest

```
Linear Regression MAE: 0.3254
Random Forest MAE: 0.0015
Linear Regression MAE: 0.1529
Random Forest MAE: 0.0000
Linear Regression MAE: 0.8464
Random Forest MAE: 1.0000
```

Figure 22 shows the results obtained from the Linear Regression and Random Forest models.

Figure 23: Gradient Boosting

Gradient Boosting MSE: 0.0001 Gradient Boosting R^2: 0.9999 Gradient Boosting MAE: 0.0063

Figure 23 shows the results obtained from the Gradient Boosting model.

Figure 24: Support Vector Regression

Support Vector Regression MSE: 0.0037 Support Vector Regression R^2: 0.9963 Support Vector Regression MAE: 0.0530

Figure 24 shows the results obtained from the Support Vector Regression model.

Figure 25: XGBoost Regressor

XGBoost	Regressor	MSE:	0.0071
XGBoost	Regressor	R^2:	0.9928
XGBoost	Regressor	MAE:	0.0719

Figure 25 shows the results obtained from the XGBoost Regressor model.

Figure 26: LSTM

Epoch 87/100
3346/3346 [=======================] - 8s 2ms/step - loss: 0.0113 - val_loss: 6.4663e-04
Epoch 88/100
3346/3346 [===============================] - 7s 2ms/step - loss: 0.0112 - val_loss: 9.2410e-04
Epoch 89/100
3346/3346 [============================] - 7s 2ms/step - loss: 0.0112 - val_loss: 3.0460e-04
Epoch 90/100
3346/3346 [========================] - 7s 2ms/step - loss: 0.0113 - val_loss: 5.9879e-04
Epoch 91/100
3346/3346 [============================] - 7s 2ms/step - loss: 0.0112 - val_loss: 5.9439e-04
Epoch 92/100
3346/3346 [=======================] - 7s 2ms/step - loss: 0.0113 - val_loss: 2.6317e-04
Epoch 93/100
3346/3346 [===========================] - 8s 2ms/step - loss: 0.0113 - val_loss: 3.4899e-04
Epoch 94/100
3346/3346 [==========================] - 8s 2ms/step - loss: 0.0112 - val_loss: 3.6309e-04
Epoch 95/100
3346/3346 [=======================] - 7s 2ms/step - loss: 0.0113 - val_loss: 4.9518e-04
Epoch 96/100
3346/3346 [============================] - 7s 2ms/step - loss: 0.0113 - val_loss: 4.1171e-04
Epoch 97/100
3346/3346 [============================] - 8s 2ms/step - loss: 0.0111 - val_loss: 2.0049e-04
Epoch 98/100
3346/3346 [===========================] - 7s 2ms/step - loss: 0.0113 - val_loss: 2.7150e-04
Epoch 99/100
3346/3346 [===========================] - 7s 2ms/step - loss: 0.0112 - val_loss: 3.0126e-04
Epoch 100/100
3346/3346 [==============================] - 8s 2ms/step - loss: 0.0113 - val_loss: 1.8225e-04
523/523 [=====================] - 1s 877us/step
LSTM Regressor MSE: 0.0002
LSTM Regressor R^2: 0.9998
ISTM Regressor MAE+ 0 0089

Figure 26 shows the results obtained from the LSTM model.

Figure 27: Model Fine Tuning



Figure 27 shows the Model Fine Tuning process code and the result of the best parameters obtained.

Figure 28: Model Validation

```
# Best model from GridSearchCV
best_rf_model = grid_search.best_estimator_
# Predict using the best model
y_pred_best_rf = best_rf_model.predict(X_test)
best_mse_rf = mean_squared_error(y_test, y_pred_best_rf)
best_r2_rf = r2_score(y_test, y_pred_best_rf)
best_mae_rf = mean_absolute_error(y_test, y_pred_best_rf)
# Print the results for MSE, R^2, and MAE
print("Optimized Random Forest MSE: {:.4f}".format(best_mse_rf))
print("Optimized Random Forest R^2: {:.4f}".format(best_r2_rf))
print("Optimized Random Forest MAE: {:.4f}".format(best_mae_rf))
```

Figure 28 shows the result obtained by applying the best parameters obtained.

Figure 29: Feature Importance



Figure 29 Code written to determine Feature Importance.

Figure 30: Show All Results

```
# Data setup
time_cycles = np.arange(len(y_test))
# Plot for Linear Regression
plt.figure(figsize=(6, 4))
plt.plot(time_cycles, y_test, 'k-', label='Actual RUL')
plt.plot(time_cycles, y_pred_lr, 'b--', label='Predicted RUL (Linear Regression)')
plt.fill_between(time_cycles, y_test, y_pred_lr, color='blue', alpha=0.1)
plt.title(f'Linear Regression (MAE: {mae_lr:.4f}))
plt.xlabel('Time Cycles')
plt.ylabel('RUL (cycles)
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(6, 4))
plt.plot(time_cycles, y_test, 'k-', label='Actual RUL')
plt.plot(time_cycles, y_pred_rf, 'g--', label='Predicted RUL (Random Forest)')
plt.fill_between(time_cycles, y_test, y_pred_rf, color='green', alpha=0.1)
plt.title(f'Random Forest (MAE: {mae_rf:.4f})')
plt.xlabel('Time Cycles')
plt.legend()
plt.grid(True)
plt.show()
```

```
Figure 30 Code written to display all applied models on the graph.
```

Figure 31: Show All Results

```
plt.figure(figsize=(6, 4))
plt.plot(time_cycles, y_test, 'k-', label='Actual RUL')
plt.plot(time_cycles, y_pred_xgb, 'r--', label='Predicted RUL (XGBoost)')
plt.fill_between(time_cycles, y_test, y_pred_xgb, color='red', alpha=0.1)
plt.title(f'XGBoost (MAE: {mae_xgb:.4f})'
plt.xlabel('Time Cycles')
plt.legend()
plt.grid(True)
plt.show()
plt.figure(figsize=(6, 4))
plt.figure('fgsize('0, 4))
plt.plot(time_cycles, y_test, 'k-', label='Actual RUL')
plt.plot(time_cycles, y_pred_gbr, 'b--', label='Predicted RUL (Gradient Boosting)')
plt.fill_between(time_cycles, y_test, y_pred_gbr, color='blue', alpha=0.1)
plt.title(f'Gradient Boosting (MAE: {mae_gbr:.4f})')
plt.xlabel('Time Cycles')
plt.legend()
plt.grid(True)
plt.show()
# Plot for Support Vector Regression
plt.figure(figsize=(6, 4))
plt.plot(time_cycles, y_test, 'k-', label='Actual RUL')
plt.plot(time_cycles, y_pred_svr, 'g--', label='Predicted RUL (SVR)')
plt.fill_between(time_cycles, y_test, y_pred_svr, color='green', alpha=0.1)
plt.title(f'SVR (MAE: {mae_svr:.4f})')
plt.xlabel('Time Cycles')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 31 Code written to display all applied models on the graph.

Figure 32: Comparison All Models



Figure 32 is the code for comparing all applied models in a single graph.

References

Elasha, F., Shanbr, S., Li, X. & Mba, D. (2019). 'Prognosis of a wind turbine gearbox bearing using supervised machine learning', Sensors, vol. 19, no. 14, art. 3092.

Teng, W., Zhang, X., Liu, Y., Kusiak, A. & Ma, Z. (2016). 'Prognosis of the remaining useful life of bearings in a wind turbine gearbox', Energies, vol. 10, no. 1, art. 32.