Configuration Manual

MSc Research Project
Data Analytics

Vijit Laxman Chekkala
Student ID: x18199429

School of Computing
National College of Ireland

Supervisor: Dr Vladimir Milosavljevic
**Student Name:** Mr. Vijit Laxman Chekkala  
**Student ID:** X18199429  
**Programme:** Data Analytics  
**Year:** 2019  
**Module:** MSc Research Project  
**Lecturer:** Dr Vladimir Milosavljevic  
**Submission Due Date:** 28/09/2020  
**Project Title:** Configuration Manual  
**Word Count:** 1495  
**Page Count:** 26

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. 

All 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:** Vijit Laxman Chekkala  
**Date:** 22nd September 2020

### PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attach a completed copy of this sheet to each project (including multiple copies)</td>
<td>□</td>
</tr>
<tr>
<td><strong>Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).</strong></td>
<td>□</td>
</tr>
<tr>
<td><strong>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.</strong></td>
<td>□</td>
</tr>
</tbody>
</table>

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): |  

1 Introduction

Every stage of the research “Predictive Maintenance for Fault Diagnosis and Failure Prognosis in Hydraulic System” is discussed in the configuration manual. The components of the hydraulic system – cooler, valve, pump and accumulators are based on multi-class classification problem and the stability of the hydraulic system is based on binary classification problem. This manual is to explain the system requirements in Chapter 2, project development in Chapter 3.

2 System Requirements

Hardware and software configuration for the research project development are discussed below:

- **Hardware configuration:** The overview of the hardware configuration used for the research project is shown in Figure 1.

![Figure 1: Hardware Configuration](image-url)
• **Software configuration:** The software used for the entire process of the research project is discussed as follows:
  
  • **RStudio:** A dashboard is created to perform the exploratory data analysis and published to web using Shiny package in R programming language.
  
  • **Jupyter Notebook:** After exploring the data, model building (Logistic Regression, Xgboost, LightGBM, Catboost and Random Forest) and evaluation process in done in the jupyter notebook using the Python programming language.
  
  • **Google Colaboratory:** Due to low system efficiency the deep learning model – Artificial Neural Network (ANN) is developed in Google Colaboratory.
  
  • **Spyder:** For deployment purpose, the python code is used in Spyder software.
  
  • **GitHub:** The model loaded, the web interface, along with Heroku setup configuration are stored in GitHub.
  
  • **Heroku:** The research project is deployed by connecting Heroku software to the saved files in the GitHub.

3 **Project Development**

• **Importing Libraries:** The required libraries for implementation are imported in the jupyter notebook using python programming language.

```python
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime as dt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report, f1_score
import xgboost as xgb
import os
from imblearn.over_sampling import SMOTE
```  

**Figure 2: Importing Libraries**

• **Loading Data:** After importing the libraries, the data is downloaded from the UCI machine learning repository. The raw data file is divided into sensor data and a profile file where the independent and dependent variables are stored in text format. According to the description of the dataset given, the text files are converted into comma separated values (csv) format and loaded into the jupyter notebook. Depending on the duration of the data cycle of sensor data mentioned, the average is taken of the independent variables where 44,680 number of attributes are averaged into 2205 number of instances. The independent and dependent variables are stored in X and Y dataframe respectively. This process is shown in Figure 3.
```python
import pandas as pd

# Import all pressure sensors data
pressureFile1 = get_files(dir_path=dir_path, filename='P01.txt')
pressureFile2 = get_files(dir_path=dir_path, filename='P02.txt')
pressureFile3 = get_files(dir_path=dir_path, filename='P03.txt')
pressureFile4 = get_files(dir_path=dir_path, filename='P04.txt')
pressureFile5 = get_files(dir_path=dir_path, filename='P05.txt')
pressureFile6 = get_files(dir_path=dir_path, filename='P06.txt')

# Import volume flow files
volumeFlow1 = get_files(dir_path=dir_path, filename='V01.txt')
volumeFlow2 = get_files(dir_path=dir_path, filename='V02.txt')

# Import temperature files
temperature1 = get_files(dir_path=dir_path, filename='T01.txt')
temperature2 = get_files(dir_path=dir_path, filename='T02.txt')
temperature3 = get_files(dir_path=dir_path, filename='T03.txt')
temperature4 = get_files(dir_path=dir_path, filename='T04.txt')

# Import rest of the data files: pump efficiency, vibrations, cooling efficiency, coolin power, efficiency factor
pumpEff = get_files(dir_path=dir_path, filename='EP01.txt')
vibrationEff = get_files(dir_path=dir_path, filename='V01.txt')
coolingEff = get_files(dir_path=dir_path, filename='CE.txt')
coolingPower = get_files(dir_path=dir_path, filename='CP.txt')
effFactor = get_files(dir_path=dir_path, filename='SE.txt')

# Import Label data from profile file
profile = get_files(dir_path=dir_path, filename='profile.txt')

# Split the profile into relevent target labels
y_coolerCondition = pd.DataFrame(profile.iloc[:, 0])
y_valveCondition = pd.DataFrame(profile.iloc[:, 1])
y_pumpLeak = pd.DataFrame(profile.iloc[:, 2])
y_hydraulicAcc = pd.DataFrame(profile.iloc[:, 3])
y_stableFlag = pd.DataFrame(profile.iloc[:, 4])

# Assigning Column Names for target variables
y_coolerCondition.columns = ['Cooler']
y_valveCondition.columns = ['Valves']
y_pumpLeak.columns = ['Pump Leaking']
y_hydraulicAcc.columns = ['Accumulator']
y_stableFlag.columns = ['Stable']

# Combine all dataframes
Y = pd.concat([y_coolerCondition, y_valveCondition, y_pumpLeak, y_hydraulicAcc, y_stableFlag], axis=1)

# Checking the shape of the target variables data frame
Y.shape

# Average the cycle data of the independent variables
mean_conversion(df):
    df1 = pd.DataFrame()
    df1 = df1.mean(axis = 1)
    return df1
```

Figure 3: Loading Data
Applying the conversion function to the independent variables

```python
PS1 = pd.DataFrame(mean_conversion(pressureFile1))
PS1.columns = ['PS1']

PS2 = pd.DataFrame(mean_conversion(pressureFile2))
PS2.columns = ['PS2']

PS3 = pd.DataFrame(mean_conversion(pressureFile3))
PS3.columns = ['PS3']

PS4 = pd.DataFrame(mean_conversion(pressureFile4))
PS4.columns = ['PS4']

PS5 = pd.DataFrame(mean_conversion(pressureFile5))
PS5.columns = ['PS5']

PS6 = pd.DataFrame(mean_conversion(pressureFile6))
PS6.columns = ['PS6']

FS1 = pd.DataFrame(mean_conversion(volumeFlow1))
FS1.columns = ['FS1']

FS2 = pd.DataFrame(mean_conversion(volumeFlow2))
FS2.columns = ['FS2']

TS1 = pd.DataFrame(mean_conversion(temperature1))
TS1.columns = ['TS1']

TS2 = pd.DataFrame(mean_conversion(temperature2))
TS2.columns = ['TS2']

TS3 = pd.DataFrame(mean_conversion(temperature3))
TS3.columns = ['TS3']

TS4 = pd.DataFrame(mean_conversion(temperature4))
TS4.columns = ['TS4']

P1 = pd.DataFrame(mean_conversion(pump1))
P1.columns = ['P1']

VS1 = pd.DataFrame(mean_conversion(vibration1))
VS1.columns = ['VS1']

CE1 = pd.DataFrame(mean_conversion(coolingE1))
CE1.columns = ['CE1']

CP1 = pd.DataFrame(mean_conversion(coolingP1))
CP1.columns = ['CP1']

SE1 = pd.DataFrame(mean_conversion(effFactor1))
SE1.columns = ['SE1']
```

Combine all dataframes

```python
X = pd.concat([PS1, PS2, PS3, PS4, PS5, PS6, FS1, FS2, TS1, TS2, TS3, TS4, P1, VS1, CE1, CP1, SE1], axis=1)
```

`X.head(5)` # Checking the top 5 rows of the independent variables

<table>
<thead>
<tr>
<th>PS1</th>
<th>PS2</th>
<th>PS3</th>
<th>PS4</th>
<th>PS5</th>
<th>PS6</th>
<th>FS1</th>
<th>FS2</th>
<th>TS1</th>
<th>TS2</th>
<th>TS3</th>
<th>TS4</th>
<th>P1</th>
<th>VS1</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
<td>1.60</td>
</tr>
<tr>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
<td>1.55</td>
</tr>
<tr>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
<td>1.40</td>
</tr>
</tbody>
</table>
```

**Figure 4: Transforming Independent Variables**
- **Publishing Exploratory Data Analysis Dashboard**: After successfully loading the data into the jupyter notebook in csv format, the data is used in RStudio to create a dashboard where the data is explored using a correlation plot, histogram to check the distribution of data, variable importance using Random Forest. The dashboard is published to web using the Shiny package.

```r
# r g global, include=FALSE
library(ggplot2)
library(plotly)
library(plyr)
library(flexdashboard)
library(magick)
library(shiny)
library(shinydashboard)
library(shinyui)

# create some data
df <- read.csv("MY_data.csv")
df_target <- read.csv("ML_target.csv")
df <- select(df, -1)
df_target <- select(df_target, -1)
data_final <- cbind(df, df_target)
df_final[Coolers <- as.factor(df_final[Coolers])
df_final[Values <- as.factor(df_final[Values])
df_final[Pump_Leakage <- as.factor(df_final[Pump_Leakage])
df_final[Accumulator <- as.factor(df_final[Accumulator])
df_final[Stable <- as.factor(df_final[Stable])

str(df)
summary(df)

split(df)

# Data Exploration

### Correlation Matrix

```
### Pressure sensor data

```r
# Histogram
hist(df$P1)
hist(df$P2)
hist(df$P3)
```

### Temperature data

```r
# Histogram
hist(df$T1)
hist(df$T2)
```

### Volume flow data

```r
# Histogram
hist(df$V1)
hist(df$V2)
```

### Cooling efficiency

```r
# Histogram
hist(df$C1)
```

### Pump efficiency, Vibrations and Efficiency factor

```r
# Histogram
class(df$P2)
```

---

Variable Importance using Random Forest

```r
Row (.tdset.tdset-Tale)
```

### Cooler condition

```r
# Histogram
library(randomforest)
ggplot(data = data_final) + geom_bar(mapping = aes(x = Coolers, y = ...prop..., group = 1), stat="count", color="white",fill="lightblue") + ylab("Percentage")
ggtitle("Percentage Distribution of Target in Cooler condition") + scale_y_continuous(labels = scales::percent_format())
rf_Cool <- randomForest(Coolers ~ Valves Pump Leakage Accumulator Stable, data=data_final)
```

### Valve condition

```r
# Histogram
ggplot(data = data_final) + geom_bar(mapping = aes(x = Valves, y = ...prop..., group = 1), stat="count", color="white",fill="lightblue") + ylab("Percentage")
ggtitle("Percentage Distribution of Target in Valve Condition") + scale_y_continuous(labels = scales::percent_format())
rf_valve <- randomForest(Valves ~ Coolers Pump Leakage Accumulator Stable, data=data_final)
```

---

```r
rf_Cool,rf_valve,by = "red", pch=22
```
• **Scaling Data:** From the distribution of data, it is observed that the independent variables are skewed and therefore the data is brought to normal distribution using Quantile Transform Scalar.

```python
from sklearn.preprocessing import QuantileTransformer
quantile = QuantileTransformer(output_distribution='normal', n_quantiles=1000, random_state=0)
quantile.fit(X_train)
X_train = quantile.transform(X_train)
```

**Figure 6. Quantile Transform Scalar**

• **Dimensionality Reduction:** For reducing the dimensions, three techniques are used. Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection (UMAP).
Principal Component Analysis

Covariance matrix of features

```python
# features are columns from x_std
features = X_scaled.T
covariance_matrix = np.cov(features)
print(covariance_matrix)
```

Eigenvectors and Eigen values from Covariance matrix

```python
eig_vals, eig_vecs = np.linalg.eig(covariance_matrix) # linear algebra functions
print('\nEigenvectors \n\n\neig_vals

Eigenvalues
1.64503212e+01 4.75466576e+00 2.14350480e+00 9.05906161e+00
5.67111705e+01 2.80966142e+01 4.18718535e+01 1.30771612e+01
6.20126533e-02 4.98341144e-02 4.54836386e-02 3.44310737e-02
3.17615212e-02 1.51880102e-02 2.48555916e-03 6.36783479e-03
8.62944028e-03
```

# we reduce dimension to 1 dimension, since 1 eigenvector has enough variance
eig_val[0]/sum(eig_vals)

0.6423253427157957

# for selecting PCA2

eig_val[1]/sum(eig_vals)

0.18565574952007965

# project data points onto selected eigenvectors

```python
projected_X_1 = data_trans.dot(eig_vecs.T[0])
projected_X_2 = data_trans.dot(eig_vecs.T[1])
```

```python
projected_X_1

array([-0.51727163, -1.56018171, -2.25026577, ..., 6.69613898, 7.48794038, 8.32365393])
```

```python
projected_X_2

array([-5.30769498, -4.85704267, -4.47517591, ..., -2.62260511, -3.09779309, -3.57360646])
```

result = pd.DataFrame(projected_X_1)

```python
result['PC2'] = pd.DataFrame(projected_X_2)
```

```python
result['y-axis'] = 0.0
result['label'] = Y['Coolers']
result = result.drop('label', axis=1)
result = result.rename(columns={0: 'PC1'})
```

```python
from sklearn import decomposition
```

```python
pca = decomposition.PCA(n_components=2)
```

```python
%time pca_l = pca.fit_transform(data_trans)
```

CPU times: user 134 ms, sys: 24.1 ms, total: 158 ms
Wall time: 112 ms

```python
plt.figure(figsize=(20,10))
sns.FacetGrid(pca_final, size=6).map(plt.scatter, 'PC1', 'PC2')
plt.title('PCA')
```

Figure 7: Principal Component Analysis
**t-distributed stochastic neighbor embedding**

```python
from sklearn.manifold import TSNE
data_1 = data_trans[0:2205,1] #using the array instead of dataframe to implement t-sne
labels_1 = Y['Coolers']
mod_tsne = TSNE(n_components = 2, random_state=0)
time tsne_data=mod_tsne.fit_transform(data_1)
CPU times: user 45.3 s, sys: 586 ms, total: 45.8 s
Wall time: 15.8 s

Creating a new data frame which help us in plotting the result data

tsne_data=np.vstack((tsne_data.T,labels_1)).T
tsne_df=pd.DataFrame(data=tsne_data,columns=['Dim_1','Dim_2','Label'])
label = Y['Coolers']

plt.figure(figsize=(20,10))
sns.FacetGrid(tsnne_df,size=6).map(plt.scatter,'Dim_1','Dim_2');plt.title('t-SNE')
```

**Figure 8: t-distributed stochastic neighbor embedding**

**UMAP**

```python
import umap
from umap import UMAP
fit = umap.UMAP(n_neighbors=200, min_dist=0.5, n_components=2)
time u = fit.fit_transform(data_trans)
CPU times: user 21.4 s, sys: 357 ms, total: 21.8 s
Wall time: 15.4 s

u

array([[ 0.4044028 , -11.646117 ],
       [ 0.41534036, -11.496172 ],
       [ 0.543252 , -11.514414 ],
       [ 1.0841507 , 11.021847 ],
       [ 1.0851933 , 11.027818 ],
       [ 1.0845726 , 11.021337 ]], dtype=float32)

umap_data = pd.DataFrame({'Dim_1': u[:, 0], 'Dim_2': u[:, 1]})

plt.figure(figsize=(20,10))
sns.FacetGrid(umap_data,size=6).map(plt.scatter,'Dim_1','Dim_2');plt.title('UMAP')
```

**Figure 9: Uniform Manifold Approximation and Projection (UMAP)**

- **SMOTE for Classification Imbalance**: To deal with imbalance in the valve, pump, accumulator and stable conditions, the sampling technique SMOTE is used to balance the dataset. SMOTE is applied on the training data. If it is applied on the test data, then more synthetic data will give high accuracy.
• **Model Implementation and Evaluation**: A set of parameters are given to all the classification algorithms and the entire dataset is divided into 60% training and 40% test data. RandomisedSearchCV is used to find the best parameters and these parameters are tested on the entire dataset. To train the classification algorithms, the same parameters are provided to the model and then fitted for prediction. The evaluation techniques are carried out by confusion matrix and classification report.

• **Part 1 – Multi-class Classification:**
  • **Logistic Regression Model and Evaluation - Accumulators**

---

**Logistic regression for Hydraulic Accumulator**

```python
logModel_a = LogisticRegression()
pars = [
    {'penalty': ['l2'], 'l2', 'elasticnet', 'none'},
    'C': np.logspace(-4, 4, 20),
    'solver': ['lbfgs', 'newton-cg', 'liblinear', 'sag', 'saga'],
    'max_iter': [100, 1000, 2500, 5000]
]
clf_al = RandomizedSearchCV(logModel_a, pars, distributions = params, cv = 10, verbose=3)
best_clf = clf_al.fit(umap_data, z4)
```

**Figure 11: RandomisedSearchCV for Hyperparameter Optimization – Logistic Regression**

```python
best_clf.best_estimator_
LogisticRegression(C=0.01274274985763334, max_iter=2500, penalty='none', solver='newton-cg')
```

**Figure 12: Logistic Regression Model and Evaluation – Accumulators**
XGBoost Model and Evaluation - Accumulators

Figure 13: RandomisedSearchCV for Hyperparameter Optimisation - XGBoost

Figure 14: XGBoost Model and Evaluation
• LightGBM Model and Evaluation - Accumulators

**LightGBM for Hydraulic accumulator**

Params = {
    'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
    'num_leaves': [90, 200],
    'boosting_type': ['gbdt', 'dart', 'goss'],
    'max_depth': [5, 6, 7, 8],
    'colsample_bytree': [0.3, 0.7],
    'subsample': [0.5, 0.7],
    'min_split_gain': [0.01, 0.02, 0.03, 0.04, 0.05],
    'min_data_in_leaf': [2, 4, 6, 8, 10],
    'objective': ['binary', 'multiclass'],
    'num_class': [1, 2, 3, 4, 5],
    'metric': ['multi_logloss', 'roc', 'auc']
}

clf_alg = lgb.LGBMClassifier()
random_search_alg = RandomizedSearchCV(clf_alg, param_distributions=Params, cv=10)
random_search_alg.fit(umpa_data, s4)
random_search_alg.best_estimator_

LGBMClassifier(boostering_type='goss', colsample_bytree=0.5, learning_rate=0.15,
max_depth=6, metric='multi_logloss', min_data_in_leaf=2,
min_split_gain=0.05, min_class=1, num_leaves=200,
objective='binary', subsample=0.7)

print (f'Accuracy - : {random_search_alg.score(umpa_data, s4)):3f}')

Accuracy - : 0.937

---

**Figure 15: RandomisedSearchCV for Hyperparameter Optimization – LightGBM**

---

**Figure 16: LightGBM Model and Evaluation – Accumulators**

• Catboost Model and Evaluation - Accumulators

**Catboost for hydraulic accumulator**

params_cat = {
    'grow_policy': ['Lessguide'],
    'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
    'bootstrap_type': ['Bayesian', 'Bernoulli', 'MVS', 'Poisson'],
    'depth': [3, 6, 7, 8],
    'min_data_in_leaf': [2, 4, 6, 8, 10],
    'num_leaves': [90, 200]
}

clf_cat = CatBoostClassifier(iterations=200)
random_search_cat = RandomizedSearchCV(clf_cat, param_distributions=params_cat, cv=10)
random_search_cat.fit(umpa_data, s4)
Figure 17: RandomisedSearchCV for Hyperparameter Optimisation – CatBoost

```python
random_search_cat.best_params

{'num_leaves': 200,
 'min_data_in_leaf': 4,
 'learning_rate': 0.25,
 'grow_policy': 'Lossguide',
 'depth': 6,
 'bootstrap_type': 'bayesian'}

print('Accuracy = : ' (random_search_cat.score(umap_data, z4)))
```

Accuracy = : 0.930

Figure 18: CatBoost Model and Evaluation – Accumulators

- Random Forest Model and Evaluation - Accumulators

```python
cat_boost_a = CatBoostClassifier(num_leaves= 200,
min_data_in_leaf= 10,
learning_rate= 0.25,
grow_policy= 'Lossguide',
depth= 5,
bootstrap_type= 'Bernoulli')

cat_boost_a.fit(X_train_as, y_train_as)

y_pred_acat = cat_boost_a.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_acat = confusion_matrix(y_test_a, y_pred_acat)

print ('Confusion Matrix : \

print(classification_report(y_test_a, y_pred_acat))
```

Figure 19: RandomisedSearchCV for Hyperparameter Optimisation – Random Forest

```python
rf_p_dist={'max_depth':[3,5,10,None],
 'n_estimators':[50,100,150,200,250,300,350,400,450,500],
 'max_features':'randidx(1,3),
 'criterion': ['gini', 'entropy'],
 'bootstrap':[True,False],
 'min_samples_leaf':randint(1,4),
 }

def hypertuning_rscv(est, p_distr, n_iter, X, z4):
 rdmssearch = RandomizedSearchCV(est, param_distributions=p_distr,
 n_iter=n_iter, cv=10)
 #CV = Cross-Validation ( here using Stratified KFold CV)
 rdmssearch.fit(X, z4)
 ht_params = rdmssearch.best_params
 ht_score = rdmssearch.best_score_
 return ht_params, ht_score

df_parameters, rf_ht_score = hypertuning_rscv(est, rf_p_dist, 40, X, z4)

df_parameters

{'bootstrap': False,
 'criterion': 'entropy',
 'max_depth': None,
 'max_features': 2,
 'min_samples_leaf': 3,
 'n_estimators': 100}

rf_ht_score
0.712566449197862
```

Figure 19: RandomisedSearchCV for Hyperparameter Optimisation – Random Forest
Figure 20: Random Forest Model and Evaluation – Accumulators

- Artificial Neural Network Model and Evaluation - Accumulators

```python
rf_a = RandomForestClassifier(bootstrap = False, criterion='entropy', max_depth=None, max_features=2, min_samples_leaf=2, n_estimators=100)
rf_a.fit(X_train_a, y_train_a)
y_predarf = rf_a.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_arf = confusion_matrix(y_test_a, y_predarf)
print('Confusion Matrix : 
', cm_arf)
print(classification_report(y_test_a, y_predarf))
```

Figure 21: Artificial Neural Network Model and Evaluation for Accumulators

- Part 2 – Binary Classification: Here the data is encoded using one hot encoding and then the first variable created is dropped. Then the stable data is scaled, and the dimensions are reduced. The modelling and the implementation part are the same as defined above.
Data Preparation for Stable condition

Figure 22: Transformation of data for Stable Condition
- Quantile Transform Scalar and UMAP for Stable Condition

Scaling and reducing dimensions for Stable conditions

```python
quantile_stable = quantile.fit_transform(stable_flag)
```

- Data Splitting and SMOTE for Stable Condition

Splitting data and Handling Classification Imbalance

```python
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(unmap_stable, y, test_size=0.4)
X_train_ss, y_train_ss = oversample.fit_resample(X_train_s, y_train_s)
s_smote = pd.concat([X_train_ss, y_train_ss], axis=1)
```

- Logistic Regression Model and Evaluation - Stable Condition

```python
logmodel_s = LogisticRegression()
params = {
    'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs', 'newton-cg', 'liblinear', 'sag', 'saga'],
    'max_iter' : [100, 1000, 2500, 5000]
}
clf_s = RandomizedSearchCV(logmodel_s, param_distributions = params, cv = 10)
best_clf_s = clf_s.fit(unmap_stable, y)
```

Figure 23: Quantile Transform and UMAP for Stable Condition

Figure 24: Data Splitting and SMOTE for Stable Condition
Figure 25: RandomisedSearchCV for Logistic Regression – Stable Condition

```
best_clf_s.best_estimator_
LogisticRegression(C=7.847599703514607, max_iter=2500)
print(f'Accuracy - : {best_clf_s.score(umap_stable,y5):.3f}')
Accuracy - : 0.810
```

Figure 26: Logistic Regression Model and Evaluation for Stable Condition

- **XGBoost Model and Evaluation - Stable Condition**

```
# Values from Randomised Search CV
classifier_s.fit(X_train_st, y_train_st)
LogisticRegression(C=1438.449888828766, max_iter=5000, solver='liblinear')

y_pred_s = classifier_s.predict(X_test_st)
from sklearn.metrics import confusion_matrix
cm_s = confusion_matrix(y_test_s, y_pred_s)
print(f'Confusion Matrix : \\
', cm_s)
pdf(classification_report(y_test_s, y_pred_s))
```

Figure 27: RandomisedSearchCV for Xgboost – Stable Condition

```
params={
    'learning_rate' : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ],
    'max_depth' : [ 3, 4, 5, 6, 10, 12, 15],
    'min_child_weight' : [ 1, 3, 5, 7 ],
    'gamma' : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
    'colsample_bytree' : [ 0.3, 0.4, 0.5 , 0.7 ]
}

classifier_s=xgb.XGBClassifier()
random_search_s=RandomizedSearchCV(classifier_s, param_distributions=params, n_iter=5, cv=10)
random_search_s.fit(umap_stable,y5)

XGBClassifier(base_score=0.5, booster=None, colsample_by_level=1, colsample_bytree=0.3, gamma=0.0, gpu_id=-1, importance_type='gain', interaction_constraints=None, learning_rate=0.1, max_delta_step=0, max_depth=12, min_child_weight=5, missing='nan', monotone_constraints=None, n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method=None, validate_parameters=False, verbosity=None)

print(f'Accuracy - : {random_search_s.score(umap_stable,y5):.3f}')
Accuracy - : 0.963
```
Figure 28: XGBoost Model and Evaluation – Stable Condition

- LightGBM Model and Evaluation – Stable Condition

**Lightgbm for Stable Condition**

```python
xgb_x = xgb.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bytree=1, gamma=0.0, gpu_id=-1, importance_type='gain', interaction_constraints='None', learning_rate=0.1, max_delta_step=0, max_depth=12, min_child_weight=1, monotone_constraints='None', n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='gpu_hist', validate_parameters=False, verbosity=0)

taxi_xgb.fit(X_train_s, y_train_s)

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bytree=1, gamma=0.0, gpu_id=-1, importance_type='gain', interaction_constraints='None', learning_rate=0.1, max_delta_step=0, max_depth=12, min_child_weight=1, missing='nan', monotone_constraints='None', n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='gpu_hist', validate_parameters=False, verbosity=0)

y_pred_x = xgb_x.predict(X_test_s)
rom sklearn.metrics import confusion_matrix

cm_x = confusion_matrix(y_test_s, y_pred_x)

print("Confusion Matrix:
\n", cm_x)
print(classification_report(y_test_s, y_pred_x))
```

**Figure 29: RandomisedSearchCV for LightGBM – Stable Condition**

```python
Params =
    'learning_rate' : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
    'num_leaves' : [30, 60, 90, 120],
    'boosting_type' : ['gbdt', 'dart', 'goss'],
    'max_depth' : [5, 6, 7, 8],
    'colsample_bytree' : [0.5, 0.7],
    'subsample' : [0.5, 0.7],
    'min_split_gain' : [0.01, 0.02, 0.03, 0.04, 0.05],
    'min_data_in_leaf' : [2, 4, 8, 10],
    'metric' : ['multi_logloss', 'roc_auc', 'auc']

#modelling
clf_slgb = lgb.LGBMClassifier()
random_search_slgb = RandomizedSearchCV(clf_slgb, param_distributions=Params, cv = 10)
random_search_slgb.fit(X_train_s,Y_train_s)

random_search_slgb.best_estimator_

LGBMClassifier(boosting_type='dart', colsample_bytree=0.7, max_depth=6, metric='auc', min_data_in_leaf=8, min_split_gain=0.01, num_leaves=200, subsample=0.7)

print(f'Accuracy: {random_search_slgb.score(X_train_s,Y_train_s):.3f}')

Accuracy = 0.958
**Catboost Model and Evaluation - Stable Condition**

**CATBOOST for stable**

```python
#setting parameters for catboost
params_cat = {
    'grow_policy': ['Lossguide'],
    'learning_rate': [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
    'bootstrap_type': ['Bayesian', 'Bernoulli', 'MVS', 'Poisson'],
    'depth': [5, 4, 7, 8],
    'min_data_in_leaf': [2, 4, 6, 8, 10],
    'num_leaves': [90, 200]
}

catboost = CatBoostClassifier(iterations=200)

random_search = RandomizedSearchCV(catboost, param_distributions=params_cat, cv=5)

random_search.fit(train_X, train_y)

random_search.best_params_

{'num_leaves': 90, 'min_data_in_leaf': 8, 'learning_rate': 0.05, 'grow_policy': 'Lossguide', 'depth': 5, 'bootstrap_type': 'Bayesian'}

print(f'Accuracy - : {random_search.score(train_X, train_y):.3f}')

Accuracy - : 0.962
```

**Figure 31: RandomisedSearchCV for CatBoost – Stable Condition**

```python
catboost = CatBoostClassifier(num_leaves=200, min_data_in_leaf=2, learning_rate=0.1, grow_policy='Lossguide', depth=7, bootstrap_type='Bernoulli')

catboost.fit(X_train, y_train)

y_pred = catboost.predict(X_test)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

print(f'Confusion Matrix : 
{cm}')

print(classification_report(y_test, y_pred))
```

**Figure 32: CatBoost Model and Evaluation – Stable Condition**
Random Forest for stable

```python
x_train_rs, x_test_rs, y_train_rs, y_test_rs = train_test_split(stable_scaled, y5, test_size = 0.4)

#Setting parameter for random forest classifier
est = RandomForestClassifier()
rf_p_dist={'max_depth':[3,5,10,None],
'num_estimators':[50,100,150,200,250,300,350,400,450,500],
'max_features':'randint(1,3),
'criterion':['gini', 'entropy'],
'bootstrap':[True,False],
'min_samples_leaf':randint(1,4),
}

#fitting randomisedsearchcv for random forest
def hypertuning_rscv(est, p_distr, nbr_iter, stable_scaled,y5):
    rdsearch = RandomizedSearchCV(est, param_distributions=p_distr,
                                    n_iter=nbr_iter, cv=10)
    rdsearch.fit(stable_scaled,y5)
    ht_params = rdsearch.best_params_
    ht_score = rdsearch.best_score_
    return ht_params, ht_score

rf_parameters, rf_ht_score = hypertuning_rscv(est, rf_p_dist, 40, stable_scaled, y5)

rf_parameters

{‘bootstrap’: False,
’criterion’: ’gini’,
’max_depth’: 3,
’max_features’: 2,
’min_samples_leaf’: 2,
’n_estimators’: 200}

rf_ht_score

rf_s = RandomForestClassifier(bootstrap = True,
                              criterion = ‘entropy’,
                              max_depth= 3,
                              max_features= 1,
                              min_samples_leaf = 2,
                              n_estimators= 150)

rf_s.fit(x_train_rs,y_train_rs)

# In[132]:

y_pred_rs = rf_s.predict(x_test_rs)

from sklearn.metrics import confusion_matrix

cm_rs = confusion_matrix(y_test_rs, y_pred_rs)

print ("Confusion Matrix :
| " , cm_rs)
print(classification_report(y_test_rs, y_pred_rs))
```

Figure 33: RandomisedSearchCV for Random Forest – Stable Condition

Figure 34: Random Forest Model and Evaluation – Stable Condition
• Artificial Neural Network Model and Evaluation – Stable Condition

```python
y5 = Y('Stable')

X_train, X_test, y_train, y_test = train_test_split(X_stable, y5, test_size=0.4)

# Importing the Keras libraries and packages
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initializing the ANN
# Initializing the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 101, kernel_initializer = 'uniform', activation = 'relu', input_dim = 2))

# Adding the second hidden layer
classifier.add(Dense(units = 101, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
classifier.add(Dense(units = 101, kernel_initializer = 'uniform', activation = 'softmax'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size =200 , epochs = 100)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# from sklearn.metrics import confusion_matrix
y_pred = classifier.predict(X_test)
predictions = np.argmax(y_pred, axis=1)

cm = confusion_matrix(y_test, predictions)

print ('Confusion Matrix :
', cm)
print(classification_report(y_test, predictions))
```

Figure 35: Artificial Neural Network Model and Evaluation for Stable Condition

• Project Deployment: From 6 classification model, Logistic regression, XGBoost and Random Forest models are deployed using Streamlit library in Python. The Spyder environment is used for deployment and all the files are uploaded in the GitHub. Using an online platform Heroku, the GitHub repository is connected and then deployed at: https://webapp-research.herokuapp.com/. At the top of the web app, the EDA dashboard button is fitted which opens on a new tab. This section explains the published EDA using Shiny apps and can also be accessed from here: https://hydrauliceda.shinyapps.io/hydraulics_dashboard/
```python
import streamlit as st
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
import lightgbm as lgb
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, f1_score
import bokeh
import bokeh.layouts
import bokeh.models
from bokeh.models.widgets import Div

# Link to the dashboard in the web app
if st.button('EDA Dashboard'):
    js = """window.open('https://hydraulicaeda.shinyapps.io/draft1_blank#section-data-exploration/')"""
    # New tab or window
    html = """<a href=""""{}"""">.format(js)"""
    div = Div(text=html)
    st.bokeh_chart(div)

# Title
st.title('Research Project')
st.write(""
# Explore different classifier and datasets to monitor status of Hydraulic System and their components
"")

# Choosing the dataset
dataset_name = st.sidebar.selectbox('Select Dataset',
    ('Coolers', 'Valves', 'Pump Leakage', 'Hydraulic Accumulator', 'Stable Condition'))

st.write("""# (dataset_name) Dataset"
"")

# Choosing the classification algorithm
classifier_name = st.sidebar.selectbox('Select classifier',
    ('Logistic Regression', 'Artificial Neural Network', 'XGBoost', 'Light GBM',
     'CatBoost', 'Random Forest'))

# Reading the csv files from GitHub
url1 = 'https://github.com/cviijill/hydraulics-research/blob/master/coolers_display.csv?raw=true'
url2 = 'https://github.com/cviijill/hydraulics-research/blob/master/valves_display.csv?raw=true'
url3 = 'https://github.com/cviijill/hydraulics-research/blob/master/pump_leakage.csv?raw=true'

# Function to read the dataset according to the chosen dataset
def get_dataset(dataset_name):
    if dataset_name == 'Coolers':
        data = pd.read_csv(url1, index_col=0)
    elif dataset_name == 'Valves':
        data = pd.read_csv(url2, index_col=0)
    elif dataset_name == 'Pump Leakage':
        data = pd.read_csv(url3, index_col=0)
    elif dataset_name == 'Hydraulic Accumulator':
        data = pd.read_csv(url4, index_col=0)
    else:
        data = pd.read_csv(url5, index_col=0)

    X = data.iloc[:, :-1]
    y = data.iloc[:, -1]
    return X, y

# Defining the shape and number of the dataset
X, y = get_dataset(dataset_name)

st.write('Shape of dataset', X.shape)
st.write('Number of classes', len(np.unique(y)))
```

Figure 36: Importing libraries for WebApp

Figure 37: Link to the EDA Dashboard

Figure 38: Web Layout

Figure 39: Loading Data
def add_parameter_ui(clf_name):
    params = dict()
    if clf_name == "logistic regression":
        C = st.sidebar.slider("C", 0.001, 10.0, 1.0)
        params["C"] = C
        max_iter = st.sidebar.slider("max_iter", 500, 3000)
        params["max_iter"] = max_iter
        solver = st.sidebar.selectbox("solver", ["newton-cg", "sag", "saga", "liblinear"])
        params["solver"] = solver
    elif clf_name == "XGBM":
        min_child_weight = st.sidebar.slider("min_child_weight", 0, 5)
        params["min_child_weight"] = min_child_weight
        max_depth = st.sidebar.slider("max_depth", 1, 10)
        params["max_depth"] = max_depth
        learning_rate = st.sidebar.slider("learning_rate", 0.01, 1.0)
        params["learning_rate"] = learning_rate
        gamma = st.sidebar.slider("gamma", 0.01, 1.0)
        params["gamma"] = gamma
        colsample_bytree = st.sidebar.slider("colsample_bytree", 0, 1)
        params["colsample_bytree"] = colsample_bytree
    elif clf_name == "Light GBM":
        objective = st.sidebar.selectbox("objective", ["binary", "multiclass"])
        params["objective"] = objective
        metric = st.sidebar.selectbox("metric", ["multi_logloss", "logloss", "auc"])
        params["metric"] = metric
        boosting_type = st.sidebar.selectbox("boosting_type", ["gbdt", "dart", "goss"])
        params["boosting_type"] = boosting_type
        learning_rate = st.sidebar.slider("learning_rate", 0.01, 1.0)
        params["learning_rate"] = learning_rate
        max_depth = st.sidebar.slider("max_depth", 1, 10)
        params["max_depth"] = max_depth
        colsample_bytree = st.sidebar.slider("colsample_bytree", 0, 1)
        params["colsample_bytree"] = colsample_bytree
    else:
        bootstrap = st.sidebar.selectbox("bootstrap", ["True", "False"])
        params["bootstrap"] = bootstrap
        criterion = st.sidebar.selectbox("criterion", ["entropy", "gini"])
        params["criterion"] = criterion
        max_depth = st.sidebar.slider("max_depth", 1, 20)
        params["max_depth"] = max_depth
        max_features = st.sidebar.slider("max_features", 0.5, 0.2)
        params["max_features"] = max_features
        n_estimators = st.sidebar.slider("n_estimators", 100, 500)
        params["n_estimators"] = n_estimators
    return params

def get_classifier(clf_name, params):
    if clf_name == "logistic regression":
        clf = LogisticRegression(parsm["C"], max_iter=params["max_iter"],
        solver=params["solver"])
    elif clf_name == "XGBM":
        clf = xgb.XGBClassifier(min_child_weight=params["min_child_weight"],
        max_depth=params["max_depth"],
        learning_rate=params["learning_rate"],
        gamma=params["gamma"],
        colsample_bytree=params["colsample_bytree"])
    elif clf_name == "Light GBM":
        clf = LGBMClassifier(objective=params["objectivc"], metric=params["metric"],
        boosting_type=params["boosting_type"],
        learning_rate=params["learning_rate"],
        max_depth=params["max_depth"],
        max_features=params["max_features"],
        n_estimators=params["n_estimators"])
    else:
        clf = RandomForestClassifier(bootstrap=params["bootstrap"],
        criterion=params["criterion"],
        max_depth=params["max_depth"],
        max_features=params["max_features"],
        n_estimators=params["n_estimators"])
    return clf

clf = get_classifier(classifier_name, params)
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
# Fitting the model
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
# Evaluating the model
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
# Displaying Results
st.write("Classifier: " + classifier_name)
st.write("Accuracy: " + acc)
st.write("Confusion Matrix:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

Figure 40: Hyperparameters for Classification Models

Figure 41: Model and Evaluation
4 Acknowledgment

I would specifically like to appreciate my supervisor Dr Vladimir Milosavljevic for guiding me throughout the entire development of this research study. The quality of guidance achieved throughout the entire semester was really helpful and inspiring. A very big thanks goes to family and my friends for continuously motivating me throughout the project development process. I am grateful for the researchers who made the data publicly available after conducting their research and mentioning a chance of improvement for next researchers.