Configuration Manual

MSc Research Project
Data Analytics

Vijit Laxman Chekkala
Student ID: x18199429

School of Computing
National College of Ireland

Supervisor: Dr Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

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

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

Vijit Laxman Chekkala
Student ID: x18199429

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

- **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.

```
%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.

```
dir_path = '/Users/vijitchekkala/Desktop/Hydraulics/Data/data'
```

```
def get_files(dir_path, filename):
    return pd.read_csv(os.path.join(dir_path, filename), sep='\t', header=None)
```

Import all pressure sensors data

```
pressureFile1 = get_files(dir_path=dir_path, filename='PS1.txt')
pressureFile2 = get_files(dir_path=dir_path, filename='PS2.txt')
pressureFile3 = get_files(dir_path=dir_path, filename='PS3.txt')
pressureFile4 = get_files(dir_path=dir_path, filename='PS4.txt')
pressureFile5 = get_files(dir_path=dir_path, filename='PS5.txt')
pressureFile6 = get_files(dir_path=dir_path, filename='PS6.txt')
```

Import volume flow files

```
volumeFlow1 = get_files(dir_path=dir_path, filename='FS1.txt')
volumeFlow2 = get_files(dir_path=dir_path, filename='FS2.txt')
```

Import temperature files

```
temperature1 = get_files(dir_path=dir_path, filename='TS1.txt')
temperature2 = get_files(dir_path=dir_path, filename='TS2.txt')
temperature3 = get_files(dir_path=dir_path, filename='TS3.txt')
temperature4 = get_files(dir_path=dir_path, filename='TS4.txt')
```

Import rest of the data files: pump efficiency, vibrations, cooling efficiency, coolin power, efficiency factor

```
pump1 = get_files(dir_path=dir_path, filename='EPS1.txt')
vibration1 = get_files(dir_path=dir_path, filename='VS1.txt')
coolingE1 = get_files(dir_path=dir_path, filename='CE.txt')
coolingP1 = get_files(dir_path=dir_path, filename='CP.txt')
effFactor1 = 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 = ['Coolers']
y_valveCondition.columns = ['Valves']
y_pumpLeak.columns = ['Pump_Leakage']
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)
```

```
Y.shape #checking the shape of the target variables data frame
```

```
(2205, 5)
```

Average the cycle data of the independent variables

```
def mean_conversion(df):
    df1 = pd.DataFrame()
    df1 = df.mean(axis = 1)
    return df1
```

Figure 3: Loading Data

Applying the conversion function to the independent variables

```
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

```
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 indepndent variables
```

	PS1	PS2	PS3	PS4	PS5	PS6	FS1	FS2	TS1	TS2	TS3	TS4	P1	VS1	C
0	160.673492	109.466914	1.991475	0.0	9.842170	9.728097	6.709815	10.304592	35.621983	40.978767	38.471017	31.745250	2538.929167	0.576950	39.601
1	160.603320	109.354890	1.976234	0.0	9.635142	9.529488	6.715315	10.403098	36.676967	41.532767	38.978967	34.493867	2531.498900	0.565850	25.786
2	160.347720	109.158845	1.972224	0.0	9.530548	9.427949	6.718522	10.366250	37.880800	42.442450	39.631950	35.646150	2519.928000	0.576533	22.218
3	160.188088	109.064807	1.946575	0.0	9.438827	9.337430	6.720565	10.302678	38.879050	43.403983	40.403383	36.579467	2511.541633	0.569267	20.459
4	160.000472	108.931434	1.922707	0.0	9.358762	9.260636	6.690308	10.237750	39.803917	44.332750	41.310550	37.427900	2503.449500	0.577367	19.787

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.

```
---
title: "Research in Computing"
author: "Vijit Laxman Chekkala"
output:
  flexdashboard::flex_dashboard:
    orientation: rows
runtime: shiny
---

```{r global, include=FALSE}

library(ggplot2)
library(plotly)
library(plyr)
library(flexdashboard)
library(mlbench)
library(caret)
library(dplyr)
library(GGally)
library(plyr)
library(psych)
library(shiny)
# create some data
df <- read.csv("H.csv")
df_target <- read.csv("HH_target.csv")

#df <- select(df, -1)
#df_target <- select(df_target, -1)

data_final <- cbind(df,df_target)

data_final$Coolers<-as.factor(data_final$Coolers)
data_final$Valves<-as.factor(data_final$Valves)
data_final$Pump_Leakage<-as.factor(data_final$Pump_Leakage)
data_final$Accumulator<-as.factor(data_final$Accumulator)
data_final$Stable<-as.factor(data_final$Stable)

str(df)
#percentage
str(data_final)
per_data <- data_final %>%
  count(Coolers)%>%
  mutate(per=n/sum(n), per_label=paste0(round(per*100),"%"))
```

Data Exploration
==

Row

Correlation Matrix

```{r}
#Add control to the heat map
ggcorr(df,
       nbreaks = 6,
       low = "steelblue",
       mid = "white",
       high = "darkred",
       geom = "circle")
```


Histogram
==

Row {.tabset .tabset-fade}

Histogram of Independent variables

```{r}
# Histogram

library(plyr)
library(psych)
 #error, not numeric
multi.hist(df[,sapply(df, is.numeric)])
```
```

```
### Pressure sensor data
```{r}
# Histogram
#hist_1 <- c(df$PS1,df$PS2,df$PS3,df$PS4,df$PS5,df$PS6)
hist(df$PS1)
hist(df$PS2)
hist(df$PS3)
hist(df$PS4)
hist(df$PS5)
hist(df$PS6)
```

Temperature data
```{r}
# Histogram
hist(df$TS1)
hist(df$TS2)
hist(df$TS3)
hist(df$TS4)
```

Volume flow data
```{r}
# Histogram
hist(df$FS1)
hist(df$FS2)
```

Cooling efficiency
```{r}
# Histogram
hist(df$CE)
hist(df$CP)
```

Pump efficiency, Vibrations and Efficiency factor
```{r}
# Histogram
class(df$P1)
hist(df$P1)
hist(df$VS1)
hist(df$SE1)
```
Variable importance using Random Forest
===

Row {.tabset .tabset-fade}

Cooler condition
```{r}
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 \n- Cooler Condition") +
  scale_y_continuous(labels = scales::percent_format())

rf_Cooler<- randomForest(Coolers~. -Valves -Pump_Leakage -Accumulator -Stable, data=data_final)

#varimp
varImpPlot(rf_Cooler,bg = "red", pch=22)
```

Valve condition

```{r}
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 \n- Valve Condition") +
  scale_y_continuous(labels = scales::percent_format())
#valve variable importance
rf_valve<- randomForest(Valves~. -Coolers -Pump_Leakage -Accumulator -Stable, data=data_final)
#varimp
varImpPlot(rf_valve,bg = "red", pch=22)
```
```

```
### Internal Pump Leakage
```{r}
ggplot(data = data_final) +
  geom_bar(mapping = aes(x = Pump_Leakage, y = ..prop.., group = 1), stat="count",
           color="white",fill="lightblue") + ylab("Percentage")+
  ggtitle("Percentage Distribution of Target \n- Internal Pump Leakage") +
  scale_y_continuous(labels = scales::percent_format())
#pumpleakage
rf_pump_leakage<- randomForest(Pump_Leakage~. -Coolers -Valves -Accumulator -Stable, data=data_final)
#varimp
varImpPlot(rf_pump_leakage,bg = "red", pch=22)
```

Hydraulic accumulator/bar
```{r}
ggplot(data = data_final) +
  geom_bar(mapping = aes(x = Accumulator, y = ..prop.., group = 1), stat="count",
           color="white",fill="lightblue") + ylab("Percentage")+
  ggtitle("Percentage Distribution of Target \n- Hydraulic accumulator/bar") +
  scale_y_continuous(labels = scales::percent_format())
#accumulator
rf_accumulator<- randomForest(Accumulator~. -Coolers -Valves -Pump_Leakage -Stable, data=data_final)
#varimp
varImpPlot(rf_accumulator,bg = "red", pch=22)
```

Stable Flag
```{r}
ggplot(data = data_final) +
  geom_bar(mapping = aes(x = Stable, y = ..prop.., group = 1), stat="count",
           color="white",fill="lightblue") + ylab("Percentage")+
  ggtitle("Percentage Distribution of Target \n- Stable Flag") +
  scale_y_continuous(labels = scales::percent_format())
#Stable
rf_stable<- randomForest(Stable~., data=data_final)
#varimp
varImpPlot(rf_stable,bg = "red", pch=22)
```
Outliers
===

Row {.tabset .tabset-fade}

Outliers
```{r}
library(ggplot2)
ggplot(stack(df), aes(x = ind, y = values)) +
  geom_boxplot()
```

importance(rf_stable$IncNodePurity)
IncNodePurity
add(rf_stable)
```

Figure 5: Publishing Exploratory Data Analysis

- **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.

```
from sklearn.preprocessing import QuantileTransformer
quantile = QuantileTransformer(output_distribution='normal')
data_trans = quantile.fit_transform(X)
X_scaled = DataFrame(data_trans)
X_scaled.hist(figsize=(20,10))
```

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

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

Eigen vectors and Eigen values from Covaraince matrix

```
eig_vals, eig_vecs = np.linalg.eig(covariance_matrix) #linear algebra functions)
print('\n Eigenvectors \n%s' %eig_vecs)
```

```
print('\n Eigenvalues \n%s' %eig_vals)
```

```
 Eigenvalues
[1.64501320e+01 4.75469577e+00 2.14350408e+00 9.05906161e-01
 5.67112705e-01 2.80096412e-01 1.41871825e-01 1.10771012e-01
 6.20126535e-02 4.98341144e-02 4.54836386e-02 3.44310737e-02
 3.17615212e-02 1.51880101e-02 2.48554591e-03 6.36783407e-03
 8.62584028e-03]
```

```
#we reduce dimension to 1 dimensin, since 1 eigen vectos has enough variance
eig_vals[0]/sum(eig_vals)
```

```
0.6423253427157957
```

```
# for selecting PCA2
```

```
eig_vals[1]/sum(eig_vals)
```

```
0.18565574952007965
```

```
#project data points onto selected eigen vectos
```

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

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

```
projected_X_1
```

```
array([-0.51727183, -1.56018171, -2.25026577, ...,  6.69613898,
        7.48794038,  8.32365439])
```

```
projected_X_2
```

```
array([-5.30769498, -4.85704267, -4.47517591, ..., -2.6227051 ,
       -3.09779308, -3.57361064])
```

```
result = pd.DataFrame(projected_X_1)
```

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

```
result['y-axis']=0.0
```

```
result['label']= Y['Coolers']
```

```
result = result.drop('label',axis=1)
```

```
result = result.rename(columns={0:'PC1'})
```

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
```

```
%time pca_1= pca.fit_transform(data_trans)
```

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

```
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

```
from sklearn.manifold import TSNE
```

```
data_1 = data_trans[0:2205,:] #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(tsne_df,size=6).map(plt.scatter,'Dim_1','Dim_2')
plt.title('t-SNE')
```

Figure 8: t-distributed stochastic neighbor embedding

UMAP

```
import umap
from umap import UMAP
fit = umap.UMAP(n_neighbors=200, min_dist=0.0, 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  ],
       ...,
       [ 10.841507  ,  11.021847  ],
       [ 10.851933  ,  11.027818  ],
       [ 10.845726  ,  11.021327  ]], dtype=float32)
```

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

```
umap_data
```

```
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.

```
X_train_as, y_train_as = oversample.fit_resample(X_train_a, y_train_a)
```

```
acc_smote = pd.concat([X_train_as,y_train_as],axis=1)
```

```
sns.catplot(x="Accumulator", kind="count", color='lightblue',data=acc_smote);
```

Figure 10: SMOTE for classification imbalance

- **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

```
logModel_a = 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_al = RandomizedSearchCV(logModel_a, param_distributions = params, cv = 10,verbose=3)
```

```
best_clf = clf_al.fit(umap_data,z4)
```

Figure 11: RandomisedSearchCV for Hyperparameter Optimization – Logistic Regression

```
best_clf.best_estimator_
```

```
LogisticRegression(C=3792.690190732246, max_iter=2500, solver='liblinear')
```

```
print (f'Accuracy - : {best_clf.score(umap_data,z4):.3f}')
```

```
Accuracy - : 0.340
```

```
classifier_al = LogisticRegression(C=0.012742749857031334, max_iter=2500, penalty='none',
                    solver='newton-cg')
#values from Randomised Search CV
classifier_al.fit(X_train_as, y_train_as)
```

```
/Users/vijitchekkala/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:1321: UserWarning: Setti
ng penalty='none' will ignore the C and l1_ratio parameters
  "Setting penalty='none' will ignore the C and l1_ratio "
```

```
LogisticRegression(C=0.012742749857031334, max_iter=2500, penalty='none',
                   solver='newton-cg')
```

```
y_pred_al = classifier_al.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_al = confusion_matrix(y_test_a, y_pred_al)

print ("Confusion Matrix : \n", cm_al)
print(classification_report(y_test_a, y_pred_al))
```

Figure 12: Logistic Regression Model and Evaluation – Accumulators

- **XGBoost Model and Evaluation - Accumulators**

XGBoost for hydraulic accumulator

```
params={
  "objective":['binary','multiclass'],
  "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 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_ax=xgboost.XGBClassifier()
random_search_a = RandomizedSearchCV(classifier_ax,param_distributions=params,n_iter=5,cv=10)
random_search_a.fit(umap_data,z4)
```

```
random_search_a.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.3, gamma=0.2, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.25, max_delta_step=0, max_depth=4,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)
```

```
print (f'Accuracy - : {random_search_a.score(umap_data,z4):.3f}')
```

```
Accuracy - : 0.933
```

Figure 13: RandomisedSearchCV for Hyperparameter Optimisation - XGBoost

```
xgb_ax = xgb.XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.1, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.1, max_delta_step=0, max_depth=6,
              min_child_weight=3, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)




xgb_ax.fit(X_train_as, y_train_as)
```

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.1, gpu_id=-1,
              importance_type='gain', interaction_constraints=None,
              learning_rate=0.1, max_delta_step=0, max_depth=6,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=0, num_parallel_tree=1,
              objective='multi:softprob', random_state=0, reg_alpha=0,
              reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method=None, validate_parameters=False, verbosity=None)
```

```
y_pred_ax = xgb_ax.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_ax = confusion_matrix(y_test_a, y_pred_ax)

print ("Confusion Matrix : \n", cm_ax)
print(classification_report(y_test_a, y_pred_ax))
```

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' : [0.5,0.7],
    'subsample' : [0.5,0.7],
    'min_split_gain' : [0.01,0.02,0.03,0.04,0.05],
    'min_data_in_leaf':[2,4,6,8,10],
    'objective': ['binary','multiclass'],
    'num_class':[1,2,3,4,5],
    'metric': ['multi_logloss','roc','auc']
    }
```

```
clf_algb = lgb.LGBMClassifier()
random_search_algb = RandomizedSearchCV(clf_algb,param_distributions=Params,cv = 10)

random_search_algb.fit(umap_data,z4)
```

```
random_search_algb.best_estimator_
```

```
LGBMClassifier(boosting_type='goss', colsample_bytree=0.5, learning_rate=0.15,
               max_depth=7, metric='multi_logloss', min_data_in_leaf=2,
               min_split_gain=0.05, num_class=1, num_leaves=200,
               objective='binary', subsample=0.7)
```

```
print (f'Accuracy - : {random_search_algb.score(umap_data,z4):.3f}')
```

```
Accuracy - : 0.937
```

Figure 15: RandomisedSearchCV for Hyperparameter Optimization – LightGBM

```
lgb_algb = lgb.LGBMClassifier(colsample_bytree=0.5, learning_rate=0.15, max_depth=6,
               metric='multi_logloss', min_data_in_leaf=10, min_split_gain=0.03,
               num_class=2, num_leaves=200, objective='binary', subsample=0.5)

lgb_algb.fit(X_train_as, y_train_as)


y_pred_algb = lgb_algb.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_algb = confusion_matrix(y_test_a, y_pred_algb)

print ("Confusion Matrix : \n", cm_algb)
print(classification_report(y_test_a, y_pred_algb))
```

Figure 16: LightGBM Model and Evaluation – Accumulators

- ## Catboost Model and Evaluation - Accumulators

Catboost for hydraulic accumulator

```
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' : [5,6,7,8],
    'min_data_in_leaf':[2,4,6,8,10],
    'num_leaves': [90,200]
    }
```

```
clf_acat = CatBoostClassifier(iterations=200)
```

```
random_search_acat = RandomizedSearchCV(clf_acat,param_distributions=params_cat,cv = 10)
```

```
random_search_acat.fit(umap_data,z4)
```

```
random_search_acat.best_params_
```

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

```
print (f'Accuracy - : {random_search_acat.score(umap_data,z4):.3f}')
```

```
Accuracy - : 0.930
```

Figure 17: RandomisedSearchCV for Hyperparameter Optimisation – CatBoost

```
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 : \n", cm_acat)
print(classification_report(y_test_a, y_pred_acat))
```

Figure 18: CatBoost Model and Evaluation – Accumulators

- **Random Forest Model and Evaluation - Accumulators**

Random forest for Hydraulic Accumulator

```
X_train_ra, X_test_ra, y_train_ra, y_test_ra = train_test_split(X, z4, test_size = 0.4)
```

```
rf_p_dist={'max_depth':[3,5,10,None],
              'n_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),
              }
```

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

```
rf_parameters, rf_ht_score = hypertuning_rscv(est, rf_p_dist, 40, X, z4)
```

```
rf_parameters
```

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

```
rf_ht_score
```

```
0.7125668449197862
```

Figure 19: RandomisedSearchCV for Hyperparameter Optimisation – Random Forest

```
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_as,y_train_as)


y_pred_arf = rf_a.predict(X_test_a)

from sklearn.metrics import confusion_matrix
cm_arf = confusion_matrix(y_test_a, y_pred_arf)

print ("Confusion Matrix : \n", cm_arf)
print(classification_report(y_test_a, y_pred_arf))
```

Figure 20: Random Forest Model and Evaluation – Accumulators

- **Artificial Neural Network Model and Evaluation - Accumulators**

```
[ ] y4=Y['Accumulator']
```

```
[ ] X_train_a,X_test_a,y_train_a,y_test_a = train_test_split(X,y4,test_size=0.4)
```

```
[ ] # Initialising the ANN
    # Initialising the ANN
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(units = 131, kernel_initializer = 'uniform', activation = 'relu', input_dim = 17))

    # Adding the second hidden layer
    classifier.add(Dense(units = 131, kernel_initializer = 'uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(units = 131, 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_a, y_train_a, batch_size =50 , epochs = 200)
```

```
[ ] from sklearn.metrics import confusion_matrix

    y_pred_a = classifier.predict(X_test_a)

    predictions_a = np.argmax(y_pred_a, axis=-1)

    cm_a = confusion_matrix(y_test_a, predictions_a)

    print ("Confusion Matrix : \n", cm_a)
    print(classification_report(y_test_a, predictions_a))
```

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**

Data Transformation for stable

```
Z= pd.concat([y_coolerCondition,y_valveCondition,y_pumpLeak,y_hydraulicAcc,y_stableFlag],axis=1)
```

```
z2 = Z['Valves']
```

```
z4 = Z['Accumulator']
```

```
Coolers_stable= y1.replace([3,20,100],['close to failure','reduced Efficieny', 'full efficiency'])
```

```
Valve_stable = z2.replace([100,90,80,73],['optimal condition','small lag','severe lag','close to failure'])
```

```
Pump_stable = y3.replace([0,1,2],['no leakage','weak leakage','severe leakage'])
```

```
Accumulator_stable = z4.replace([130,115,100,90],['optimal','slighlty reduced','severely reduced','close to failure'])
```

```
stable_data = pd.concat([Coolers_stable,Valve_stable,Pump_stable,Accumulator_stable],axis=1)
```

```
stable_data
```

```
s1=X
```

```
s2=stable_data
```

```
s3=y5
```

```
final_stable = pd.concat([s1,s2],axis=1)
```

```
final_stable
```

```
stable_flag = pd.get_dummies(final_stable,drop_first=True) #dropping the first encoded variable
```

```
stable_flag
```

Figure 22: Transformation of data for Stable Condition

- **Quantile Transform Scalar and UMAP for Stable Condition**

Scaling and reducing dimensions for Stable conditions

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

```
quantile_stable
```

```
array([[ 0.43211465,  0.71214978,  1.01078211, ...,  5.19933758,
        -5.19933758, -5.19933758],
       [ 0.36885956,  0.62628276,  0.91046358, ...,  5.19933758,
        -5.19933758, -5.19933758],
       [ 0.20609304,  0.48941098,  0.90251742, ...,  5.19933758,
        -5.19933758, -5.19933758],
       ...,
       [ 1.06501769,  1.19254651,  1.06130814, ..., -5.19933758,
        -5.19933758, -5.19933758],
       [ 1.09358945,  1.22006734,  1.82333966, ..., -5.19933758,
        -5.19933758, -5.19933758],
       [ 1.10684139,  1.21682508,  1.4844574 , ..., -5.19933758,
        -5.19933758, -5.19933758]])
```

```
stable_scaled = DataFrame(quantile_stable)
```

```
stable_scaled
```

```
%time umap_stable = fit.fit_transform(stable_scaled)
```

```
CPU times: user 14.7 s, sys: 645 ms, total: 15.4 s
Wall time: 10.5 s
```

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

Figure 23: Quantile Transform and UMAP for Stable Condition

- **Data Splitting and SMOTE for Stable Condition**

Splitting data and Hanlding Classification Imbalance

```
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(umap_stable, y5,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)
```

```
sns.catplot(x="Stable", kind="count", color='lightblue',data=s_smote);
```

Figure 24: Data Splitting and SMOTE for Stable Condition

- **Logistic Regression Model and Evaluation - Stable Condition**

```
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(umap_stable,y5)
```

```
best_clf_s.best_estimator_
```

```
LogisticRegression(C=78.47599703514607, max_iter=2500)
```

```
print (f'Accuracy - : {best_clf_s.score(umap_stable,y5):.3f}')
```

```
Accuracy - : 0.810
```

Figure 25: RandomisedSearchCV for Logistic Regression – Stable Condition

```
classifier_s = LogisticRegression(C=1438.44988828766, max_iter=5000, solver='liblinear')
#values from Randomised Search CV
classifier_s.fit(X_train_ss, y_train_ss)
```

```
LogisticRegression(C=1438.44988828766, max_iter=5000, solver='liblinear')
```

```
y_pred_s = classifier_s.predict(X_test_s)

from sklearn.metrics import confusion_matrix
cm_s = confusion_matrix(y_test_s, y_pred_s)

print ("Confusion Matrix : \n", cm_s)
print(classification_report(y_test_s, y_pred_s))
```

Figure 26: Logistic Regression Model and Evaluation for Stable Condition

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

XGBOOST for stable

```
params={
  "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 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=xgboost.XGBClassifier()

random_search_s=RandomizedSearchCV(classifier_s,param_distributions=params,n_iter=5,cv=10)

random_search_s.fit(umap_stable,y5)
```

```
random_search_s.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=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 27: RandomisedSearchCV for Xgboost – Stable Condition

```python
xgb_sx = xgb.XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=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,  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)



xgb_sx.fit(X_train_s, y_train_s)
```

```
XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
              colsample_bynode=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)
```

```python
y_pred_sx = xgb_sx.predict(X_test_s)

from sklearn.metrics import confusion_matrix
cm_sx = confusion_matrix(y_test_s, y_pred_sx)

print ("Confusion Matrix : \n", cm_sx)
print(classification_report(y_test_s, y_pred_sx))
```

Figure 28: XGBoost Model and Evaluation – Stable Condition

- **LightGBM Model and Evaluation – Stable Condition**

Lightgbm for Stable Condition

```python
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' : [0.5,0.7],
    'subsample' : [0.5,0.7],
    'min_split_gain' : [0.01,0.02,0.03,0.04,0.05],
    'min_data_in_leaf':[2,4,6,8,10],
    'metric': ['multi_logloss','roc','auc']
    }
```

```python
#modelling
clf_slgb = lgb.LGBMClassifier()
random_search_slgb = RandomizedSearchCV(clf_slgb,param_distributions=Params,cv = 10)

random_search_slgb.fit(umap_stable,y5)
```

```python
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)
```

```python
print (f'Accuracy - : {random_search_slgb.score(umap_stable,y5):.3f}')
```

```
Accuracy - : 0.958
```

Figure 29: RandomisedSearchCV for LightGBM – Stable Condition

```
lgb_slgb = lgb.LGBMClassifier(colsample_bytree=0.5, max_depth=5, metric='roc',
               min_data_in_leaf=8, min_split_gain=0.01, num_leaves=90,
               subsample=0.7)

lgb_slgb.fit(X_train_ss, y_train_ss)


y_pred_slgb = lgb_slgb.predict(X_test_s)

from sklearn.metrics import confusion_matrix
cm_slgb = confusion_matrix(y_test_s, y_pred_slgb)

print ("Confusion Matrix : \n", cm_slgb)
print(classification_report(y_test_s, y_pred_slgb))
```

Figure 30: LightGBM Model and Evaluation for Stable Condition

- **Catboost Model and Evaluation - Stable Condition**

CATBOOST for stable

```
#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' : [5,6,7,8],
    'min_data_in_leaf':[2,4,6,8,10],
    'num_leaves': [90,200]
    }
```

```
clf_scat = CatBoostClassifier(iterations=200)
```

```
random_search_scat = RandomizedSearchCV(clf_scat,param_distributions=params_cat,cv = 5)
```

```
random_search_scat.fit(umap_stable,y5)
```

```
random_search_scat.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_scat.score(umap_stable,y5):.3f}')
```

```
Accuracy - : 0.962
```

Figure 31: RandomisedSearchCV for CatBoost – Stable Condition

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

```
cat_boost_s.fit(X_train_s, y_train_s)


y_pred_scat = cat_boost_s.predict(X_test_s)

from sklearn.metrics import confusion_matrix
cm_scat = confusion_matrix(y_test_s, y_pred_scat)

print ("Confusion Matrix : \n", cm_scat)
print(classification_report(y_test_s, y_pred_scat))
```

Figure 32: CatBoost Model and Evaluation – Stable Condition

- **Random Forest Model and Evaluation – Stable Condition**

Random Forest for stable

```
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],
              'n_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):
    rdmsearch = RandomizedSearchCV(est, param_distributions=p_distr,
                                   n_iter=nbr_iter, cv=10)
    rdmsearch.fit(stable_scaled,y5)
    ht_params = rdmsearch.best_params_
    ht_score = rdmsearch.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
```

Figure 33: RandomisedSearchCV for Random Forest – Stable Condition

```
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 : \n", cm_rs)
print(classification_report(y_test_rs, y_pred_rs))
```

Figure 34: Random Forest Model and Evaluation – Stable Condition

- **Artificial Neural Network Model and Evaluation – Stable Condition**

```
[ ] 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
```

```
[ ] # Initialising the ANN
    # Initialising 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 : \n", 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
```

Figure 36: Importing libraries for WebApp

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

Figure 37: Link to the EDA Dashboard

```python
#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(f"## {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')
)
```

Figure 38: Web Layout

```python
#Reading the csv foles from GitHub
url_1 = 'https://github.com/cvijit/hydraulics-research/blob/master/cooler_deploy.csv?raw=true'
url_2 = 'https://github.com/cvijit/hydraulics-research/blob/master/valve_deploy.csv?raw=true'
url_3 = 'https://github.com/cvijit/hydraulics-research/blob/master/pump_deploy.csv?raw=true'
url_4 = 'https://github.com/cvijit/hydraulics-research/blob/master/accumulator_deploy.csv?raw=true'
url_5 = 'https://github.com/cvijit/hydraulics-research/blob/master/stable_deploy.csv?raw=true'

#Function to read the dataset according to the choosen dataset
def get_dataset(dataset_name):
    if dataset_name == 'Coolers':
        data = pd.read_csv(url_1,index_col=0)
    elif dataset_name == 'Valves':
        data = pd.read_csv(url_2,index_col=0)
    elif dataset_name == 'Pump_Leakage':
        data = pd.read_csv(url_3,index_col=0)
    elif dataset_name == 'Hydraulic_Accumulator':
        data = pd.read_csv(url_4,index_col=0)
    else:
        data = pd.read_csv(url_5,index_col=0)
    X = data.iloc[:,0:17]
    y= data.iloc[:,-1]
    return X, y

X, y= get_dataset(dataset_name)

#Defining the shape and number of the dataset
st.write('Shape of dataset', X.shape)
st.write('Number of classes',len(np.unique(y)))
```

Figure 39: Loading Data

```python
#Selecting parameters for classification algorithm
def add_parameter_ui(clf_name):
    params = dict()
    if clf_name == "Logistic Regression":
        C = st.sidebar.slider("C",0.001,0.1)
        params["C"] = C
        max_iter = st.sidebar.slider("max_iter",500,3000)
        params["max_iter"]= max_iter
        solver = st.sidebar.selectbox("solver",('lbfgs','newton-cg','liblinear','sag','saga'))
        params["solver"]=solver
    elif clf_name == "XGBoost":
        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','roc','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",0.1,0.2)
        params['max_features'] = max_features
        n_estimators = st.sidebar.slider("n_estimators",100,500)
        params['n_estimators'] = n_estimators
    return params

params = add_parameter_ui(classifier_name)

#Fitting the parameters into the choosen classification model
def get_classifier(clf_name, params):
    if clf_name == 'Logistic Regression':
        clf = LogisticRegression(C=params["C"], max_iter=params["max_iter"],
                                 solver=params["solver"])
    elif clf_name == 'XGBoost':
        clf = xgb.sklearn.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':
        dtrain=lgb.Dataset(X,label=y)
        clf = lgb.LGBMClassifier(objective=params["objective"],metric=params["metric"],
                                 boosting_type=params["boosting_type"],learning_rate=params["learning_rate"],
                                 max_depth=params['max_depth'],colsample_bytree=params['colsample_bytree']),dtrain,100
    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
```

Figure 40: Hyperparameters for Classification Models

```python
clf = get_classifier(classifier_name, params)


#### CLASSIFICATION ####
#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(f'Classifier = {classifier_name}')
st.write(f'Accuracy =', acc*100)
st.write(f'Confusion matrix =', cm)
```

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.