Configuration Manual for Proactive Equipment Maintenance in Manufacturing: Leveraging Modern Deep Learning for Fault Prediction

MSc Research Project
MSc Artificial Intelligence

Sasi Venkata Krishna Dabbakuti

x23141891

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Sasi Venkata Krishna Dabbakuti
Student ID:	x23141891
Programme:	MSc Artificial Intelligence
Year:	2023 -2024
Module:	MSc Research Project
Supervisor:	Rejwanul Haque
Submission Due Date:	16/09/2024
Project Title:	Proactive Equipment Maintenance in Manufacturing: Lever-aging Modern Deep Learning for Fault Prediction
Word Count:	661
Page Count:	09

Signature: Sasi Venkata Krishna

Date: 16/09/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	YES
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	YES
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.	YES

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Proactive Equipment Maintenance in Manufacturing: Leveraging Modern Deep Learning for Fault Prediction

Sasi Venkata Krishna Dabbakuti

x23141891

1. Introduction

This configuration manual is designed to guide users through the setup, installation, and execution of the predictive maintenance system developed as part of the research study on proactive equipment maintenance. The system leverages advanced machine learning and deep learning models to predict equipment failures, thereby enabling proactive maintenance strategies in manufacturing environments. This manual provides detailed instructions on system specifications, software requirements, installation procedures, dataset acquisition, and execution of the code implementation.

2. System Specification

To successfully run the predictive maintenance system, the following system specifications are recommended:

- Operating System: Windows 10 or later, macOS 10.15 or later, or a Linux distribution (e.g., Ubuntu 18.04 or later)
- Processor: Intel Core i5 or AMD equivalent (quad-core) or higher
- RAM: Minimum 8 GB (16 GB recommended)
- Storage: At least 20 GB of free disk space
- GPU: Optional but recommended for deep learning model training (NVIDIA GPU with CUDA support)
- Internet Connection: Required for downloading datasets and software packages

3. Software Used:

The predictive maintenance system is developed using the following software tools and libraries:

- **Python:** Programming language used for implementing predictive models.
- **Anaconda:** A distribution of Python and R for scientific computing and data science, which includes package management and deployment.
- **Jupyter Notebook:** An interactive development environment for writing and running Python code.
- **TensorFlow:** An open-source deep learning framework used for building and training the Neural Network model.
- **Scikit-learn:** A Python library for machine learning that includes tools for data preprocessing, model training, and evaluation.
- **Pandas:** A Python library used for data manipulation and analysis.
- **NumPy:** A Python library for numerical computing.

- **Matplotlib & Seaborn:** Python libraries for data visualization.
- **Plotly:** A Python library for creating interactive plots and visualizations.
- **Imbalanced-learn:** A Python library for handling imbalanced datasets.

4. Dataset Source

The dataset used in this study is sourced from Kaggle. To download the dataset:

- Visit the Kaggle Predictive Maintenance Dataset page. [Pls click Control + link to redirect to dataset Kaggle website]
- Sign in with your Kaggle account (or create one if you don't have an account).
- Download the dataset (e.g., predictive_maintenance_dataset.csv) to your local machine.
- Place the downloaded dataset in a directory where the code implementation can access it.

5. Execution of the Code Implementation

To execute the predictive maintenance system, follow these steps:

A. Prepare the Environment:

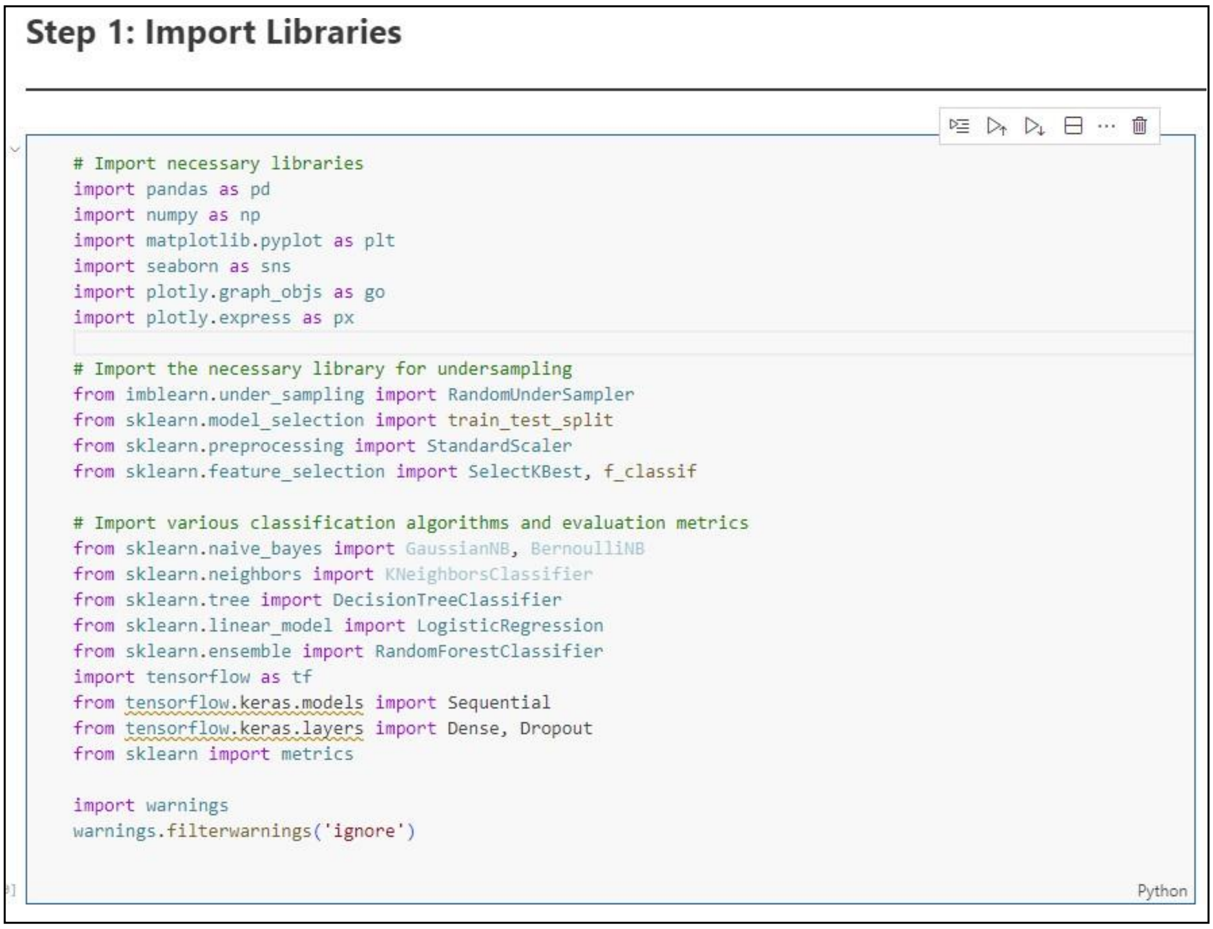
- Ensure that all the required software and libraries are installed as per the steps outlined above.
- Ensure that the dataset is downloaded and placed in the appropriate directory.

B. Launch Jupyter Notebook:

- Open the Anaconda Prompt (Windows) or Terminal (macOS/Linux).
- Activate the virtual environment (if applicable) using: conda activate predictive_maintenance
- Navigate to the directory containing the Jupyter Notebook file (.ipynb) that contains the code implementation.
- Launch Jupyter Notebook by running: Jupyter notebook
- Open the notebook file in the browser interface.

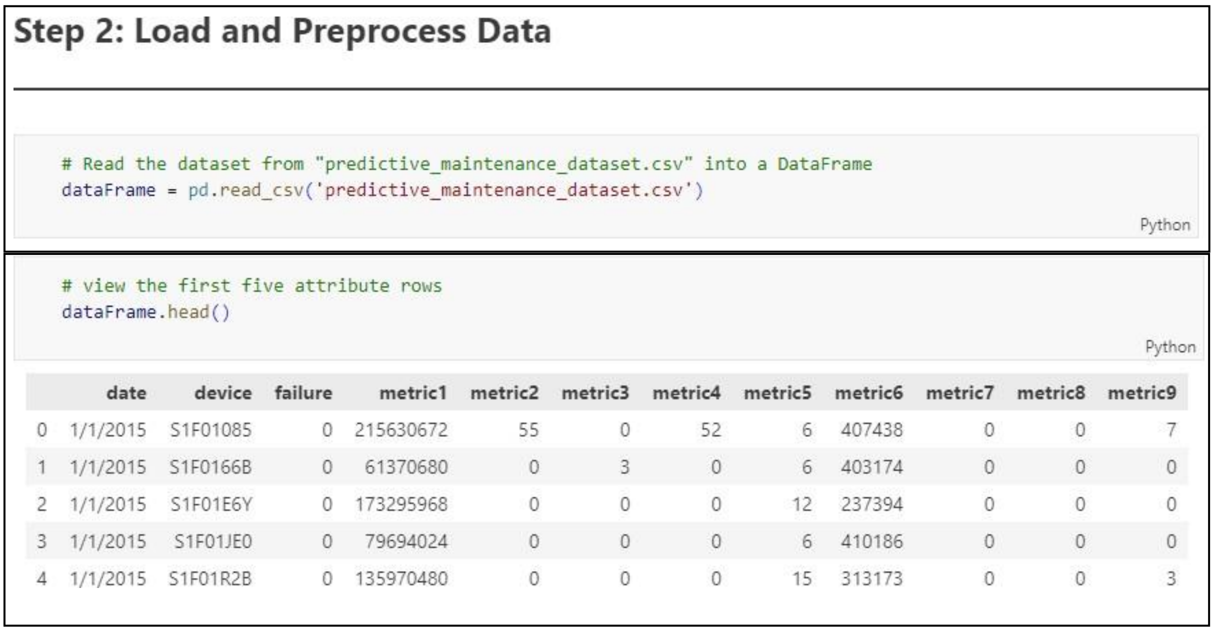
C. Execute the Notebook Cells:

- The notebook is divided into cells, each containing a portion of the code. Run each cell sequentially by selecting the cell and pressing Shift + Enter.
- Begin with data loading and preprocessing, followed by exploratory data analysis, model training, and evaluation.
- As the cells execute, visualizations and outputs will be generated inline, allowing you to monitor the progress and results.



Step 1: Import Libraries

```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
import plotly.express as px

# Import the necessary library for undersampling
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif

# Import various classification algorithms and evaluation metrics
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn import metrics

import warnings
warnings.filterwarnings('ignore')
```

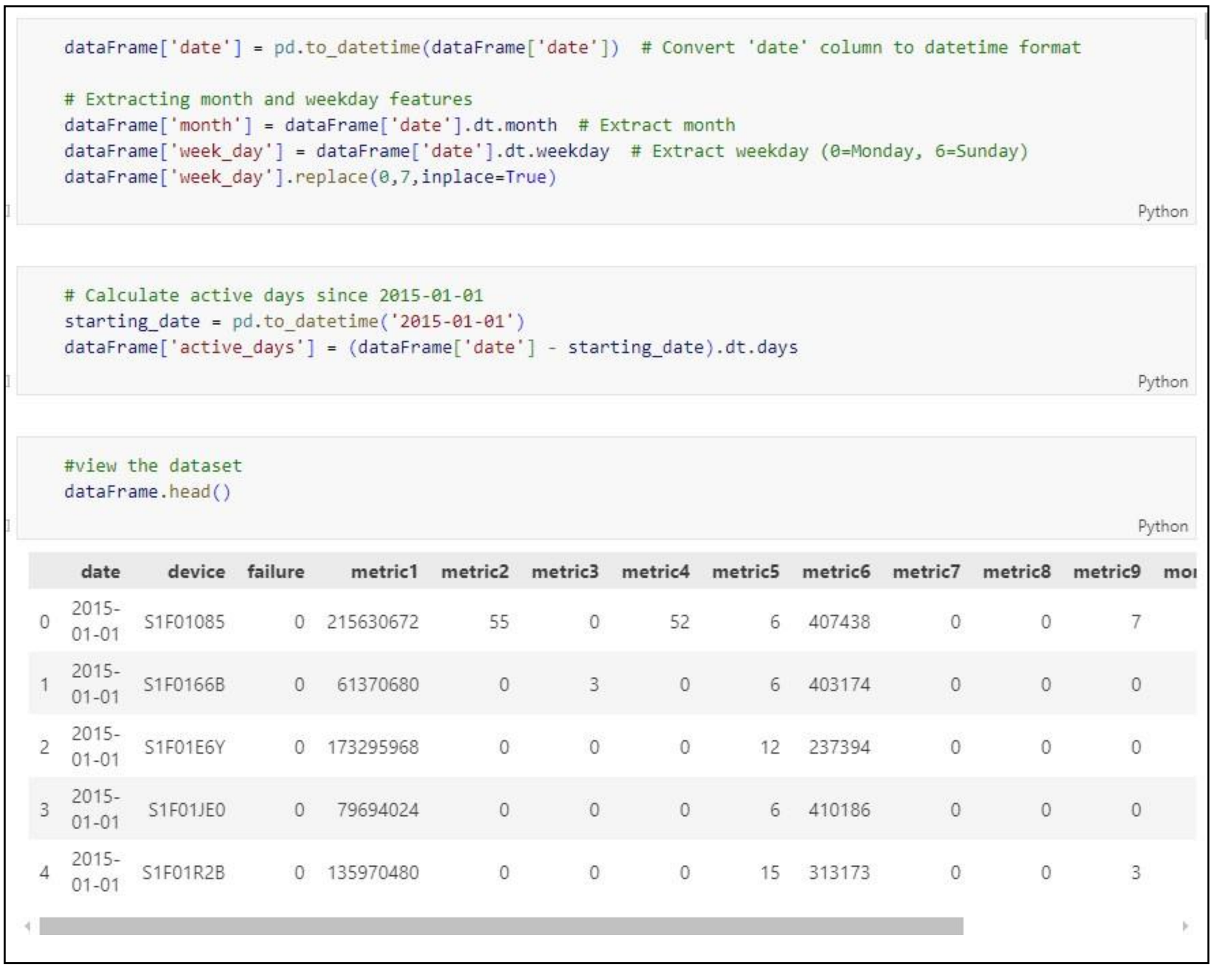


Step 2: Load and Preprocess Data

```python
# Read the dataset from "predictive_maintenance_dataset.csv" into a DataFrame
dataFrame = pd.read_csv('predictive_maintenance_dataset.csv')
```

```python
# view the first five attribute rows
dataFrame.head()
```

	date	device	failure	metric1	metric2	metric3	metric4	metric5	metric6	metric7	metric8	metric9
0	1/1/2015	S1F01085	0	215630672	55	0	52	6	407438	0	0	7
1	1/1/2015	S1F0166B	0	61370680	0	3	0	6	403174	0	0	0
2	1/1/2015	S1F01E6Y	0	173295968	0	0	0	12	237394	0	0	0
3	1/1/2015	S1F01JE0	0	79694024	0	0	0	6	410186	0	0	0
4	1/1/2015	S1F01R2B	0	135970480	0	0	0	15	313173	0	0	3



```python
dataFrame['date'] = pd.to_datetime(dataFrame['date'])  # Convert 'date' column to datetime format

# Extracting month and weekday features
dataFrame['month'] = dataFrame['date'].dt.month  # Extract month
dataFrame['week_day'] = dataFrame['date'].dt.weekday  # Extract weekday (0=Monday, 6=Sunday)
dataFrame['week_day'].replace(0,7,inplace=True)
```

Python

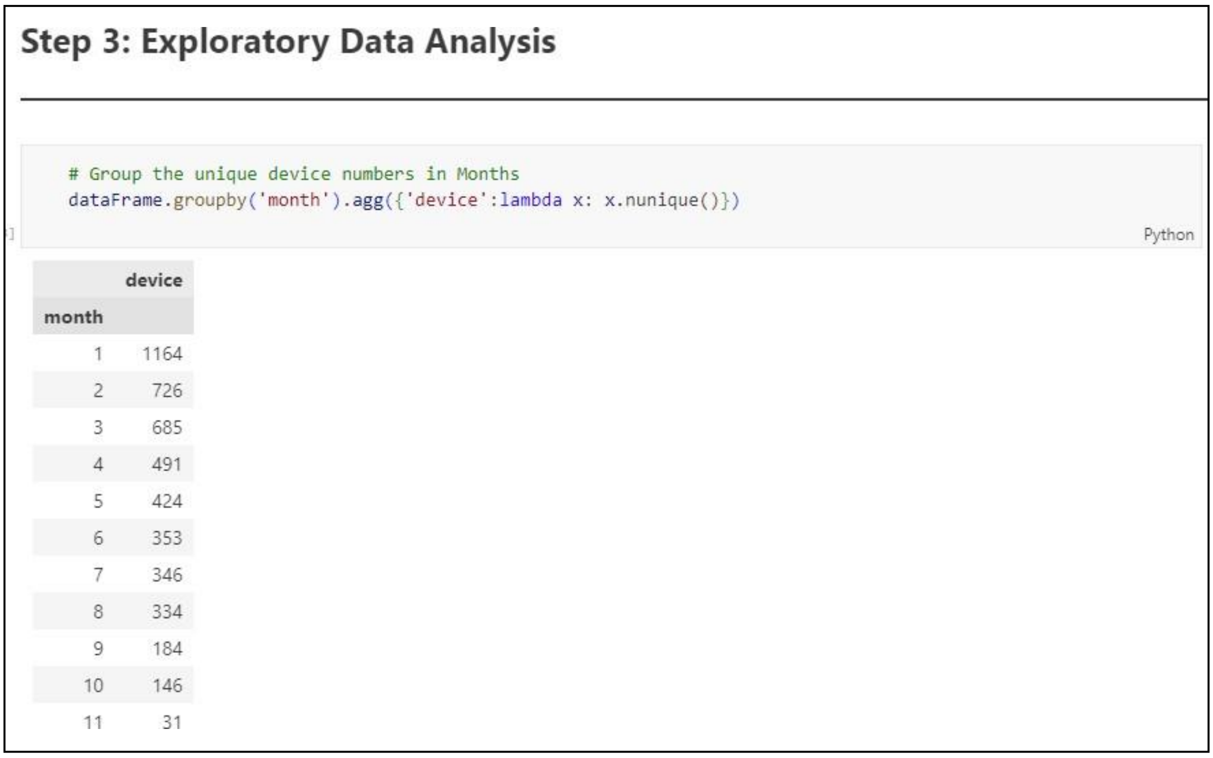
```python
# Calculate active days since 2015-01-01
starting_date = pd.to_datetime('2015-01-01')
dataFrame['active_days'] = (dataFrame['date'] - starting_date).dt.days
```

Python

```python
#view the dataset
dataFrame.head()
```

Python

	date	device	failure	metric1	metric2	metric3	metric4	metric5	metric6	metric7	metric8	metric9	mo
0	2015-01-01	S1F01085	0	215630672	55	0	52	6	407438	0	0	7	
1	2015-01-01	S1F0166B	0	61370680	0	3	0	6	403174	0	0	0	
2	2015-01-01	S1F01E6Y	0	173295968	0	0	0	12	237394	0	0	0	
3	2015-01-01	S1F01JE0	0	79694024	0	0	0	6	410186	0	0	0	
4	2015-01-01	S1F01R2B	0	135970480	0	0	0	15	313173	0	0	3	

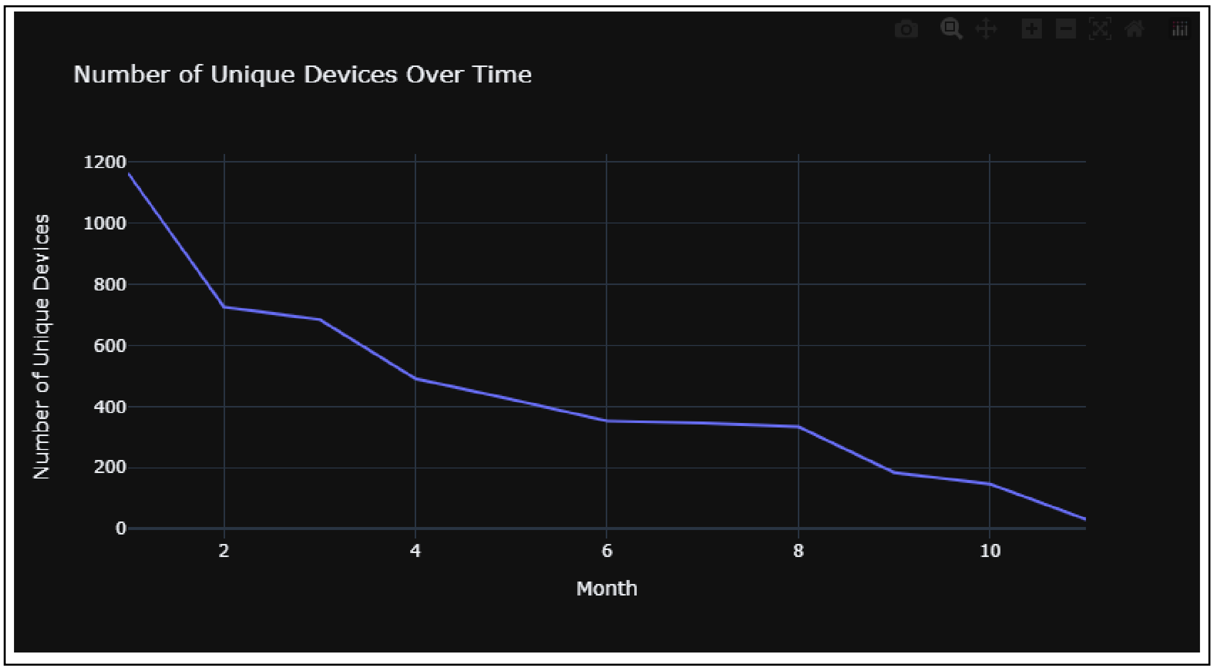


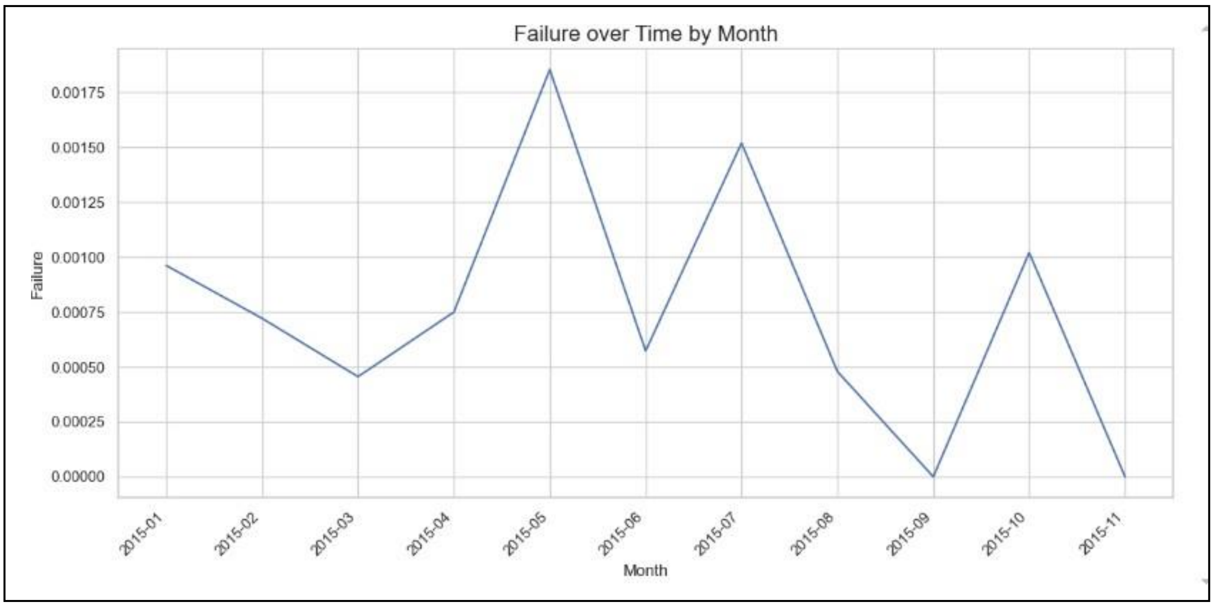
Step 3: Exploratory Data Analysis

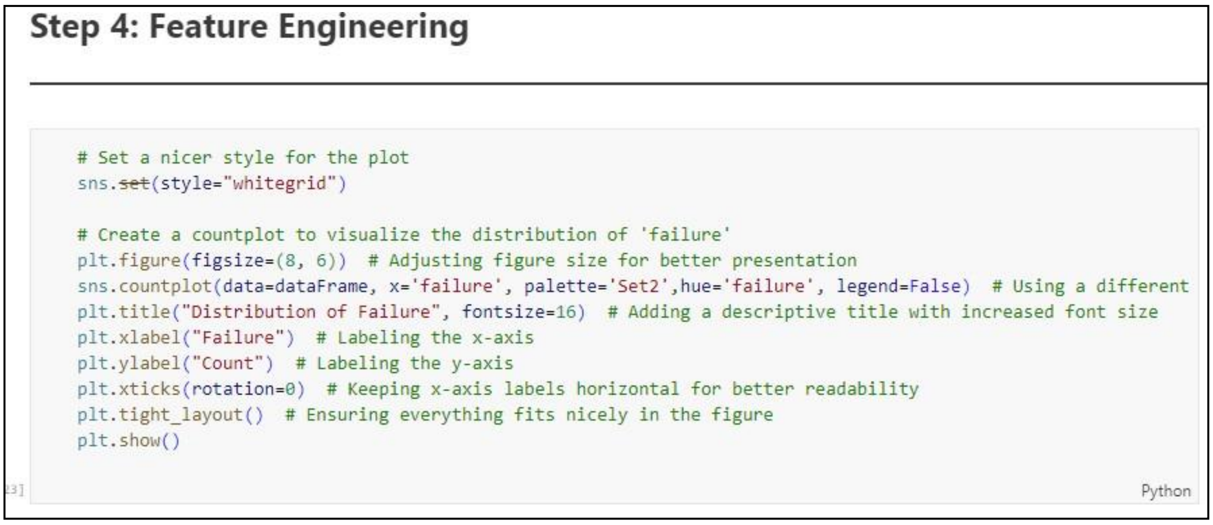
```python
# Group the unique device numbers in Months
dataFrame.groupby('month').agg({'device':lambda x: x.nunique()})
```

Python

	device
month	
1	1164
2	726
3	685
4	491
5	424
6	353
7	346
8	334
9	184
10	146
11	31

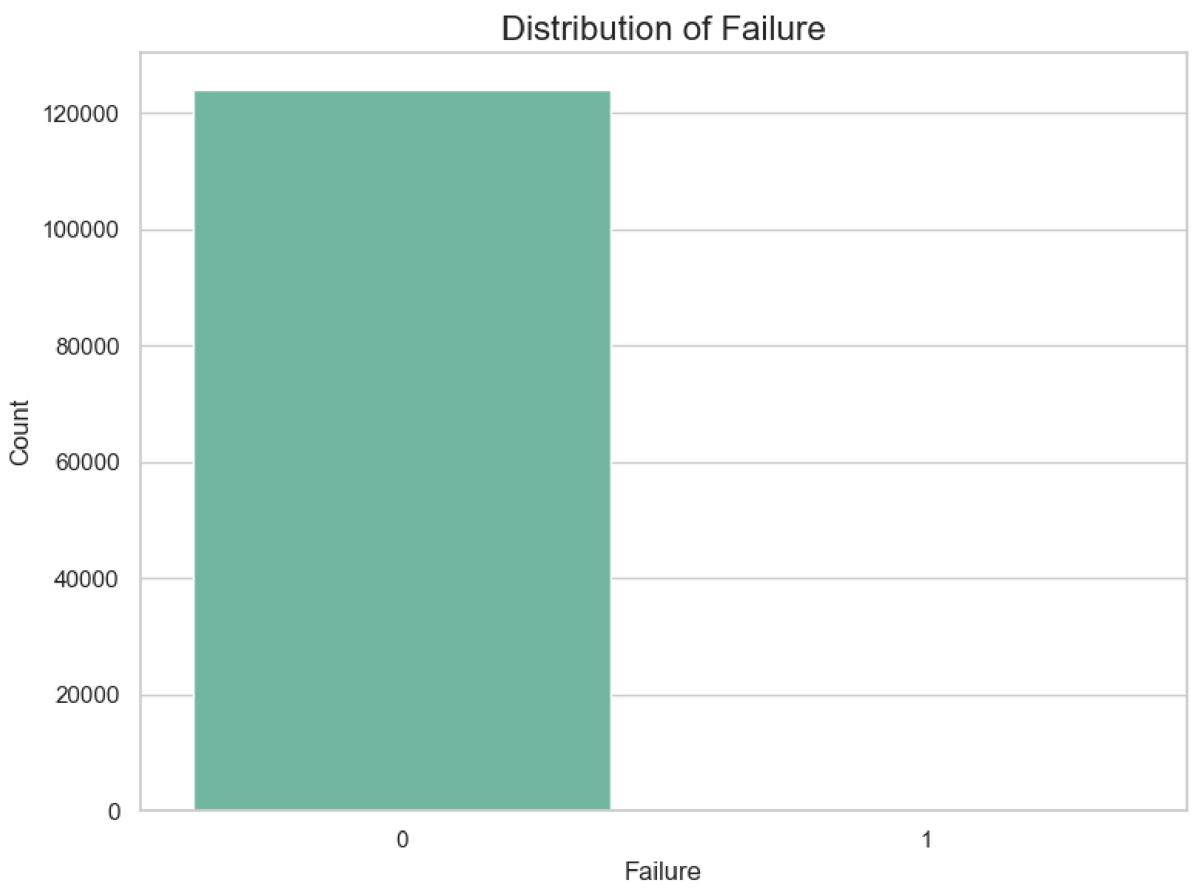


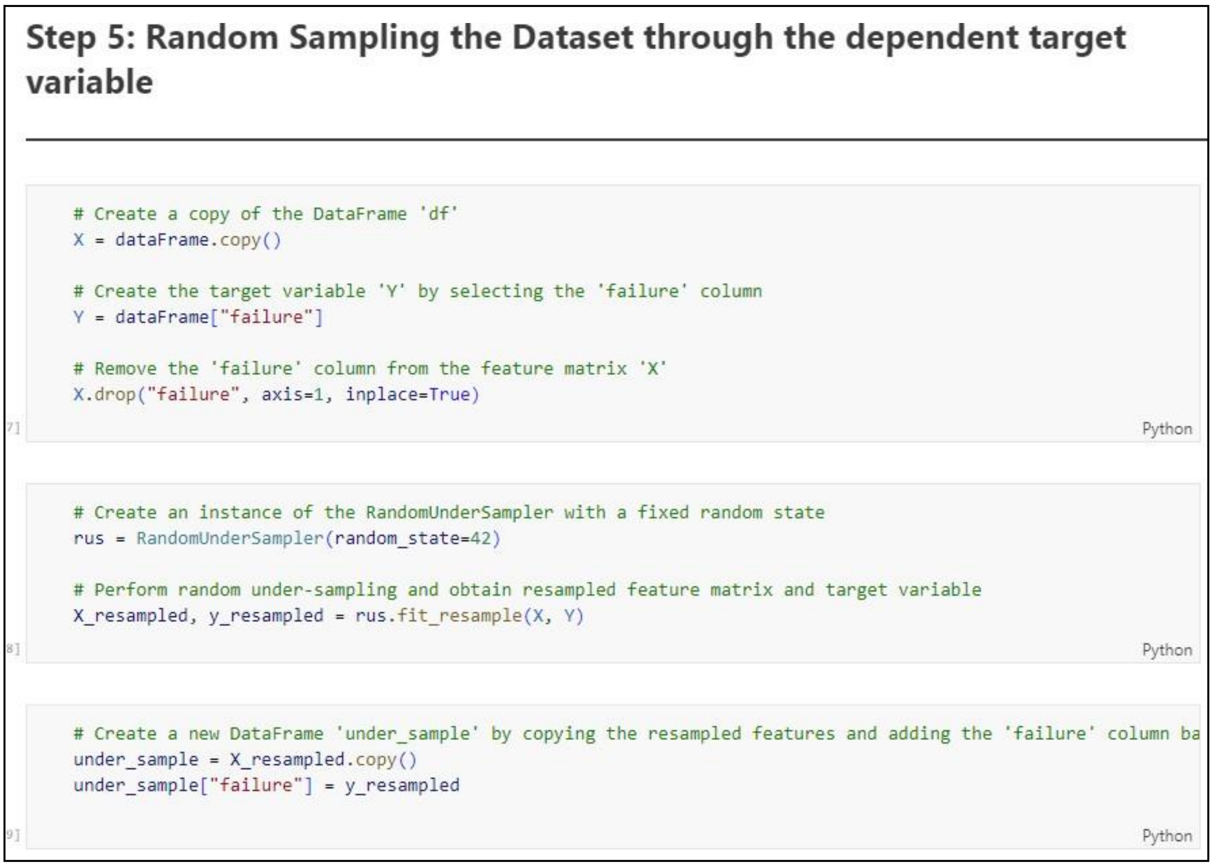




Step 4: Feature Engineering

```
# Set a nicer style for the plot
sns.~~set~~(style="whitegrid")

# Create a countplot to visualize the distribution of 'failure'
plt.figure(figsize=(8, 6))  # Adjusting figure size for better presentation
sns.countplot(data=dataFrame, x='failure', palette='Set2',hue='failure', legend=False)  # Using a different
plt.title("Distribution of Failure", fontsize=16)  # Adding a descriptive title with increased font size
plt.xlabel("Failure")  # Labeling the x-axis
plt.ylabel("Count")  # Labeling the y-axis
plt.xticks(rotation=0)  # Keeping x-axis labels horizontal for better readability
plt.tight_layout()  # Ensuring everything fits nicely in the figure
plt.show()
```



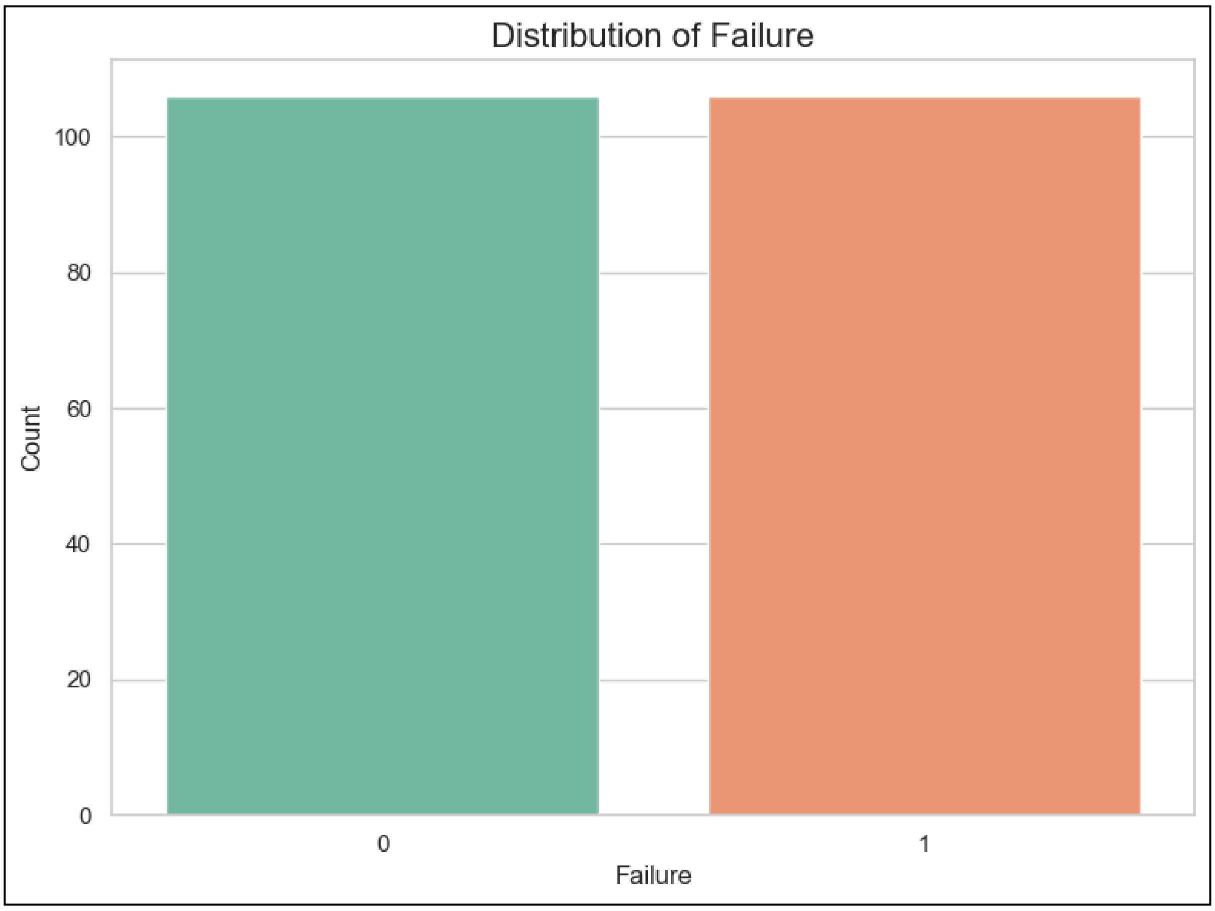


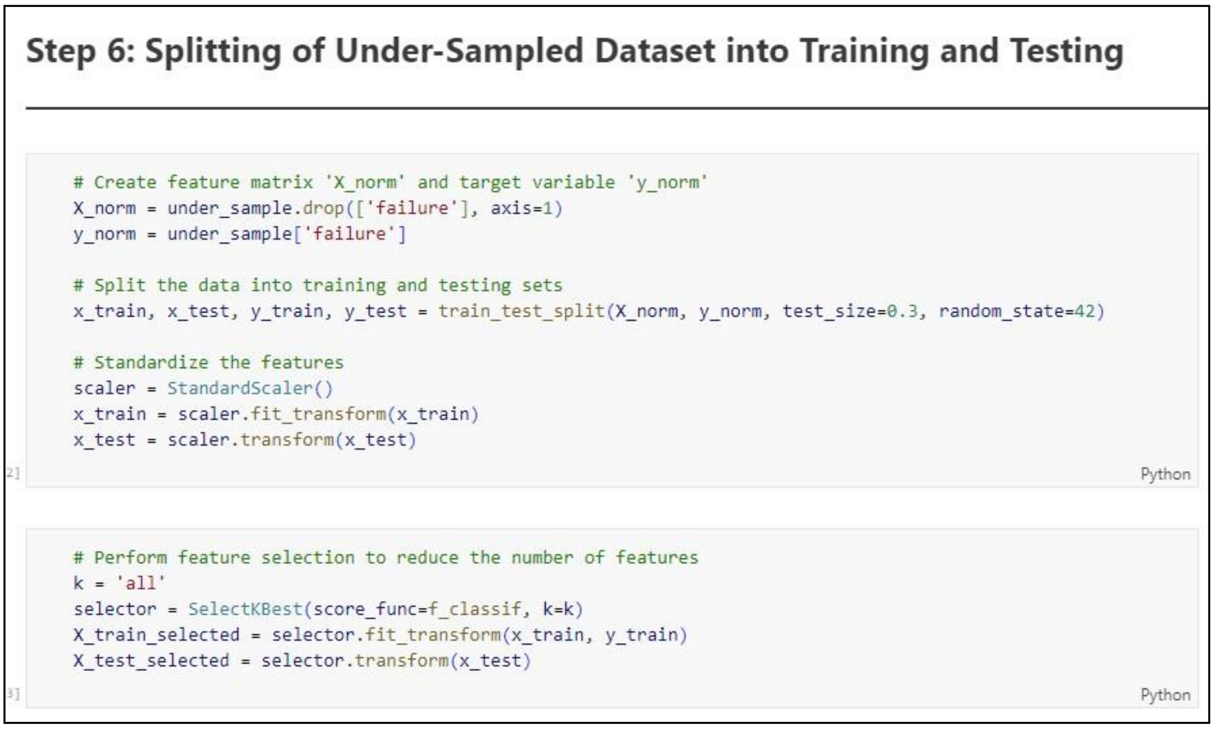
Step 5: Random Sampling the Dataset through the dependent target variable

```python
# Create a copy of the DataFrame 'df'
X = dataFrame.copy()

# Create the target variable 'Y' by selecting the 'failure' column
Y = dataFrame["failure"]

# Remove the 'failure' column from the feature matrix 'X'
X.drop("failure", axis=1, inplace=True)
```

```python
# Create an instance of the RandomUnderSampler with a fixed random state
rus = RandomUnderSampler(random_state=42)

# Perform random under-sampling and obtain resampled feature matrix and target variable
X_resampled, y_resampled = rus.fit_resample(X, Y)
```

```python
# Create a new DataFrame 'under_sample' by copying the resampled features and adding the 'failure' column ba
under_sample = X_resampled.copy()
under_sample["failure"] = y_resampled
```

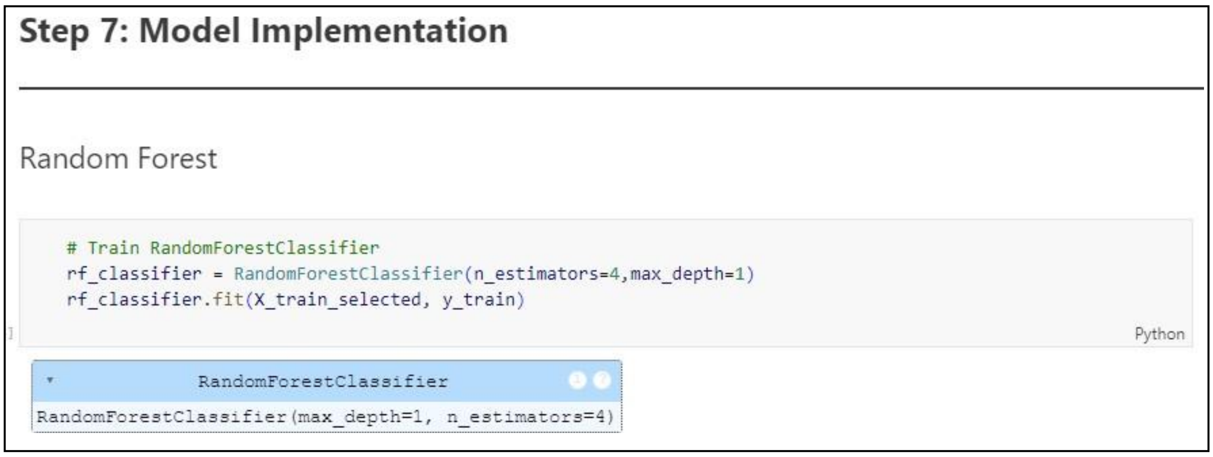




Step 6: Splitting of Under-Sampled Dataset into Training and Testing

```
# Create feature matrix 'X_norm' and target variable 'y_norm'
X_norm = under_sample.drop(['failure'], axis=1)
y_norm = under_sample['failure']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X_norm, y_norm, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

```
# Perform feature selection to reduce the number of features
k = 'all'
selector = SelectKBest(score_func=f_classif, k=k)
X_train_selected = selector.fit_transform(x_train, y_train)
X_test_selected = selector.transform(x_test)
```

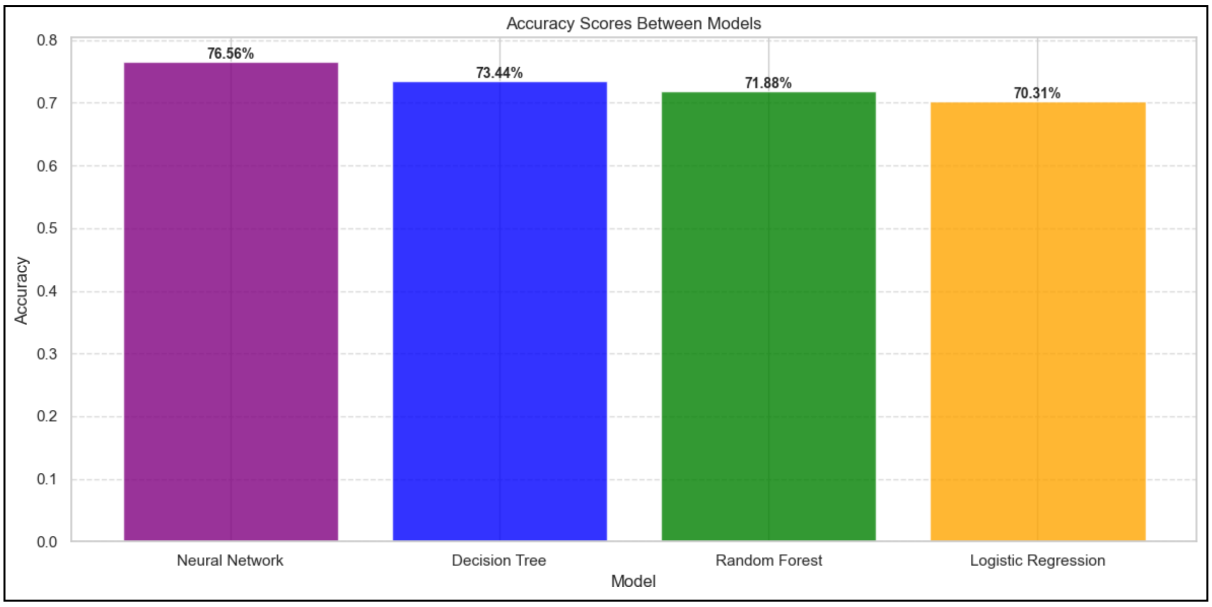


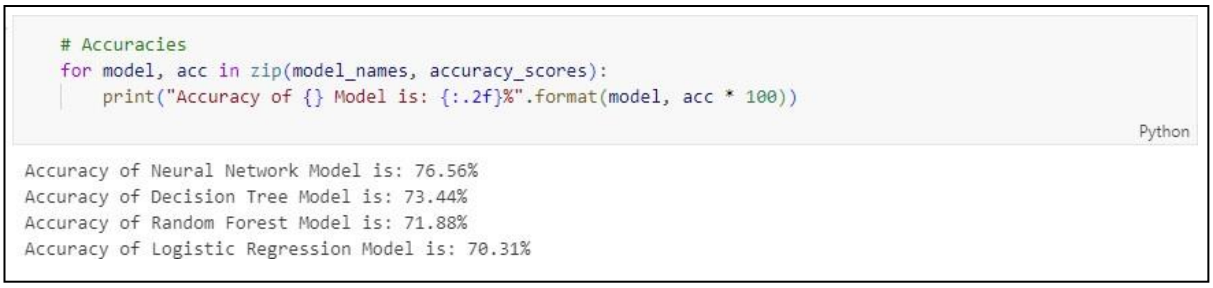
Step 7: Model Implementation

Random Forest

```python
# Train RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=4,max_depth=1)
rf_classifier.fit(X_train_selected, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=1, n_estimators=4)
```





```python
# Accuracies
for model, acc in zip(model_names, accuracy_scores):
    print("Accuracy of {} Model is: {:.2f}%".format(model, acc * 100))
```

```
Accuracy of Neural Network Model is: 76.56%
Accuracy of Decision Tree Model is: 73.44%
Accuracy of Random Forest Model is: 71.88%
Accuracy of Logistic Regression Model is: 70.31%
```

D. Review the Results:

- Upon completing the execution of all cells, review the outputs, including model evaluation metrics, confusion matrices, and visualizations.
- Compare the performance of different models and analyze their effectiveness in predicting equipment failures.

E. Modify and Experiment:

- You can modify the code to experiment with different models, hyperparameters, or preprocessing techniques.
- Save any changes to the notebook and re-run the cells to see how they impact the results.

By following this configuration manual, you will be able to set up, install, and execute the predictive maintenance system, allowing you to explore and replicate the results of the research study.

References

(1) Anaconda: https://docs.anaconda.com/free/anaconda/install/windows/

(2) Kaggle Dataset Source: Kaggle Predictive Maintenance Dataset page

(3) Python, W. (2021). Python. Python releases for windows, 24.

(4) babu Venkateswara, R. (2022). Configuration Manual.

(5) Gollapudi, S. (2016). Practical machine learning. Packt Publishing Ltd.

(6) Ganapathi, A., Datta, K., Fox, A., & Patterson, D. (2009, March). A case for machine learning to optimize multicore performance. In First USENIX Workshop on Hot Topics in Parallelism (HotPar'09).

(7) Phelps, R., Krasnicki, M., Rutenbar, R. A., Carley, L. R., & Hellums, J. R. (2000). Anaconda: simulation-based synthesis of analog circuits via stochastic pattern search. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19(6), 703-717.