**ENGINE CONDITION**

**TREND MONITORING**

*Failure Prediction of an Aircraft Engine*

By

Shreyas S

Machine Learning Intern

Faststream Technologies

Arkere, Bangalore - 560076

****Email: [hello@shreyas.im](mailto:hello@shreyas.im)

**TABLE OF CONTENTS**

1. **Introduction**
   1. Background
   2. Engine Condition Trend Monitoring
   3. Machine Learning
   4. Objectives
   5. Dependencies and Tools
   6. Assumptions
2. **Methodology**
3. **Implementation**
   * 1. Generating Data
     2. Applying Multiple Linear Regression
     3. Applying Decision Tree
     4. Applying Random Forest
     5. Testing Machine Learning Models
     6. Visualization of Predictions
     7. Computing R2 Score
4. **Results**
5. **Conclusions**
6. **Future Enhancements**

**INTRODUCTION**

This report will be of interest for Pilots, Aircraft Engine Manufacturers, Data Scientists and Researchers. This report provides an overview of how Machine Learning algorithms can help predict the probability of failure of aircraft engines. This process is called Engine Condition Trend Monitoring.

* 1. **Background**

Faststream Technologies is a vanguard of technology solutions, specializing in Product & System Engineering, IoT, Big Data, Security, and Application Development with a global footprint across North America, EMEA, and APAC. With over 200+ clients, Faststream Technologies enables Digital Transformation for enterprises by delivering a flawless customer experience, business competence, and deep insights through an integrated set of disruptive technologies and expertise. We are passionate about delivering well-organized, inventive and world-class hardware and software solutions, with a focus on Healthcare, Aerospace, Semiconductors, Automotive, Consumer Electronics, Home Automation, Telecommunications, Security, Retail, and E-Commerce.

Faststream Technologies works at the juncture of business and technology, assisting clients with advancing their product and business performance through sustainable information technology solutions. Faststream Technologies drives innovation to help clients advance their product design, business processes, and application development. Our engineering team’s deep expertise in transforming design specs into marketable hardware products — through ASIC design services that include RTL design, design verification and physical design for digital and analogue/mixed-signal semiconductors — is a key differentiator to our suite of application development capabilities.

For today’s challenges like embedded processor SoC specifications, Faststream Technologies delivers all of the required firmware/embedded software, positioning us as the turnkey ‘concept-to-product’ design company. The team is led by a group of focused senior executives and Technologists who complement each other with significant industry experience in building turnkey solutions. Many of our technologists have multiple patents to their credit in the areas of Analog/Mixed-Signal Design, IoT and embedded systems.

* 1. **Engine Condition Trend Monitoring**

Engine Condition Trend Monitoring (ECTM) is defined as using engine operational data to find symptoms of damage, deterioration or excessive wear. Essentially, it’s a technique to continuously monitor the health of engines. By tracking a known set of parameters like altitude, noise from the fan blade, exhaust gas temperature of turbine, fuel flow of turbine, low pressure fan speed, high pressure rotor speed, mechanical problems in the engine, oil leaks, fuel pump etc., operators are able to predict needed maintenance before a failure occurs. This doesn’t stop the initial problem from happening but is a great tool to predict a failure before anything catastrophic occurs.

Although flight crews are able to notice big or sudden changes to performance, ECTM will identify subtle changes over a period of time that a flight crew won’t. Every flight, the aircraft is in different environmental conditions that may not give a clear picture of performance to the flight crews. ECTM will calculate the information provided, correct it for standard day conditions and predict when the engine might fail.

Properly trained, humans can manually do the analysis, but the risk for error is great. Computers and software is used to identify and interpret trends and aircraft operator is notified of any anomalies that need to be addressed.

* 1. **Machine Learning**

Machine Learning is the science (and art) of programming computers so they can learn from data. For example, your spam filter is a Machine Learning program that can learn to flag spam given examples of spam emails (e.g., flagged by users) and examples of regular (non-spam) emails. The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample). In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks.

***Supervised Learning***

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semi-supervised learning, and Reinforcement Learning. In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels.

***Unsupervised learning***

It is the machine learning task of inferring a function that describes the structure of "unlabelled" data (i.e. data that has not been classified or categorized). Since the examples given to the learning algorithm are unlabelled, there is no straightforward way to evaluate the accuracy of the structure that is produced by the algorithm—one feature that distinguishes unsupervised learning from supervised learning and reinforcement learning

* 1. **Objectives**

The goal of the project is to validate Engine Condition Trend Monitoring’s performance across three Machine Learning Regression Algorithms – Multiple Linear Regression, Decision Tree and Random Forest.

* 1. **Dependencies and Tools**

1. Python – a general-purpose interpreted, interactive, object-oriented, and high-level programming language
2. Anaconda – a free and open source distribution of the Python and R programming languages for data science and machine learning related applications, that aims to simplify package management and deployment
3. Numpy – the fundamental package for scientific computing with Python.
4. Scipy - a Python-based ecosystem of open-source software for mathematics, science, and engineering
5. Pandas - pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
6. Matplotlib – a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
7. Sci-kit learn – a free software machine learning library for the Python programming language.
   1. **Assumptions**

Data, in the absence of its availability, has been generated based on nominal value and random degradation.

A minimum and maximum value for every component has been preset. An increase in the value of any one of the components beyond its maximum value, affects the probability of failure of that component. The failure of each component constitutes to the total probability of failure of the whole Turbine Engine. The Turbine Engine fails completely when the probability reaches 1.

The minimum, maximum and value when probability of failure reaches 1 under normal conditions are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Components** | **Minimum value** | **Value after which degradation starts** | **Value when probability of failure reaches 1** |
| Fan Blade Noise | 130dB | 137dB | 143dB |
| Exhaust Gas Temperature of Turbine | 1300oC | 1900 oC | 2500 oC |
| Fuel Flow of Turbine | 5000kg/hr | 5500kg/hr | 6000kg/hr |
| Low Pressure Fan Speed | 12000rpm | 14000rpm | 16000rpm |
| High Pressure Rotor Speed | 10000rpm | 12000rpm | 14000rpm |

**METHODOLOGY**

Generate Training Data

Train Machine Learning Algorithms with the generated data

Save the Model

Generate Test Data

Fit the saved Model to the Test Set

Predict the Probability of Failure

Visualization of Predictions

Compute R2 Score

* Training data is first generated based on all the assumptions made.
* Machine Learning models are built using Multiple Linear Regression, Decision Tree and Random Forest and are saved.
* Some more data for Test Set is generated.
* The saved models are then loaded to the Test Set.
* The probability of failure of each engine in the Test Set data is predicted.
* Graph is plotted against Age and Probability of Failure with one line representing nominal values and another line representing the random degradation values generated in the Test Set.
* Finally, R2 Score is computed for each engine and the results are compared.

**IMPLEMENTATION**

There are 2 datasets generated – Training Set and Test Set.

Data for 1000 engines with random degradation have been generated as Training Set which is used to train the Machine Learning Models.

Data for one engine in 10 different scenarios with random degradation are generated as Test Set to validate the trained Machine Learning Model.

1. **Generating Data**

The Training and Test Set data is generated using a python library called numpy. Numpy has methods to randomly generate numbers in a given range, store them in an array and perform operations on them very efficiently. Sample data is given below:

﻿import numpy as np

import pdb

import pandas as pd

head = ["engine", "month", "noise", "egt", "ff", "n1", "n2", "fp\_noise", "fp\_egt", "fp\_ff", "fp\_n1", "fp\_n2", "total\_fp"]

start\_noise = 130

start\_egt = 1300

start\_ff = 5000

start\_n1 = 12000

start\_n2 = 10000

max\_noise = 137

max\_egt = 1900

max\_ff = 5500

max\_n1 = 14000

max\_n2 = 12000

df = pd.DataFrame(columns=head)

for engine in range(1, 1001):

months = np.arange(1, 61)

fp\_noise = np.zeros(60)

fp\_egt = np.zeros(60)

fp\_ff = np.zeros(60)

fp\_n1 = np.zeros(60)

fp\_n2 = np.zeros(60)

total\_fp = np.zeros(60)

range\_noise = np.random.uniform(0.1, 0.55, size=59)

range\_egt = np.random.randint(15, 41, size=59)

range\_ff = np.random.randint(10, 34, size=59)

range\_n1 = np.random.randint(68, 95, size=59)

range\_n2 = np.random.randint(68, 95, size=59)

noise = np.cumsum(range\_noise) + start\_noise

noise = np.insert(noise, 0, start\_noise)

egt = np.cumsum(range\_egt) + start\_egt

egt = np.insert(egt, 0, start\_egt)

ff = np.cumsum(range\_ff) + start\_ff

ff = np.insert(ff, 0, start\_ff)

n1 = np.cumsum(range\_n1) + start\_n1

n1 = np.insert(n1, 0, start\_n1)

n2 = np.cumsum(range\_n2) + start\_n2

n2 = np.insert(n2, 0, start\_n2)

noise\_index = np.where(noise >= max\_noise)[0][0]

fp\_noise[noise\_index: ] = (noise[noise\_index:] - max\_noise) / 0.06

fp\_noise = fp\_noise/100

fp\_noise[fp\_noise > 1] = 1

egt\_index = np.where(egt >= max\_egt)[0][0]

fp\_egt[egt\_index: ] = (egt[egt\_index:] - max\_egt) / 6

fp\_egt = fp\_egt/100

fp\_egt[fp\_egt > 1] = 1

ff\_index = np.where(ff >= max\_ff)[0][0]

fp\_ff[ff\_index: ] = (ff[ff\_index:] - max\_ff) / 5

fp\_ff = fp\_ff/100

fp\_ff[fp\_ff > 1] = 1

n1\_index = np.where(n1 >= max\_n1)[0][0]

fp\_n1[n1\_index: ] = (n1[n1\_index:] - max\_n1) / 20

fp\_n1 = fp\_n1/100

fp\_n1[fp\_n1 > 1] = 1

n2\_index = np.where(n2 >= max\_n2)[0][0]

fp\_n2[n2\_index: ] = (n2[n2\_index:] - max\_n2) / 20

fp\_n2 = fp\_n2/100

fp\_n2[fp\_n2 > 1] = 1

total\_fp = (fp\_noise + fp\_egt + fp\_ff + fp\_n1 + fp\_n2) / 5

data = pd.DataFrame({'engine': engine, 'month': months, 'noise': noise, 'egt': egt, 'ff': ff, 'n1': n1, 'n2': n2, 'fp\_noise': fp\_noise, 'fp\_egt': fp\_egt, 'fp\_ff': fp\_ff, 'fp\_n1': fp\_n1, 'fp\_n2': fp\_n2, 'total\_fp': total\_fp})

df = df.append(data, sort=False)

df.to\_csv('../../data/train\_data/train\_data.csv', index=False)

1. **Applying** **Multiple Linear Regression**

Multiple Linear Regression is a linear approach to modelling the relationship between a scalar response (dependent variable) and multiple explanatory variables (independent variables).

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import pickle

# Importing the dataset

dataset = pd.read\_csv('../../../data/train\_data/train\_data.csv')

X = dataset.iloc[:, 1:12].values

y = dataset.loc[:, ['total\_fp']].values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Splitting the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Fitting Simple Linear Regression to the Training set

regressor = LinearRegression()

regressor.fit(X\_train, y\_train)

# Predicting the results

y\_pred = regressor.predict(X\_test)

y\_pred[y\_pred > 0.99] = 1

y\_pred[y\_pred < 0] = 0

# Saving the model

with open('../../../models/multiple\_linear\_regression/mlr\_total\_fp.pkl', 'wb') as f:

pickle.dump(regressor, f)

1. **Applying** **Decision Tree Regression**

Decision tree builds regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf node represents a decision on the numerical target. The topmost decision node in a tree which corresponds to the best predictor called root node.

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

import pickle

# Importing the dataset

dataset = pd.read\_csv('../../../data/train\_data/train\_data.csv')

X = dataset.iloc[:, 1:12].values

y = dataset.loc[:, ['total\_fp']].values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Splitting the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Fitting Simple Linear Regression to the Training set

regressor = DecisionTreeRegressor(random\_state = 0)

regressor.fit(X\_train, y\_train)

# Predicting the results

y\_pred = regressor.predict(X\_test)

y\_pred[y\_pred > 0.99] = 1

y\_pred[y\_pred < 0] = 0

# Saving the model

with open('../../../models/decision\_tree/dt\_total\_fp.pkl', 'wb') as f:

pickle.dump(regressor, f)

1. **Applying** **Random Forest Regression**

The Random Forest Regression is a type of Machine Learning Algorithm that makes predictions by combining decisions from a sequence of base models. More formally we can write this class of models as:

*g(x) = f0(x) + f1(x) + f2(x) + …*

where the final model *g* is the sum of simple base models *fi*. Here, each base classifier is a simple Decision Tree.

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestRegressor

import pickle

# Importing the dataset

dataset = pd.read\_csv('../../../data/train\_data/train\_data.csv')

X = dataset.iloc[:, 1:12].values

y = dataset.loc[:, ['total\_fp']].values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Splitting the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=False)

# Fitting Simple Linear Regression to the Training set

regressor = RandomForestRegressor(n\_estimators = 10, random\_state = 0)

regressor.fit(X\_train, y\_train)

# Predicting the results

y\_pred = regressor.predict(X\_test)

y\_pred[y\_pred > 0.99] = 1

y\_pred[y\_pred < 0] = 0

# Saving the model

with open('../../../models/random\_forest/rf\_total\_fp.pkl', 'wb') as f:

pickle.dump(regressor, f)

1. **Testing Machine Learning Models**

**Multiple Linear Regression:**

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.linear\_model import LinearRegression

import pickle

import csv

from sklearn.metrics import r2\_score

# Importing the dataset

dataset = pd.read\_csv('../../../data/test\_data/test\_data\_1.csv')

# ==================================

# ==================================

# Noise

# ==================================

X\_noise\_data = dataset.loc[:, ['month', 'noise']]

X\_noise = X\_noise\_data.values

# Encoding categorical data

labelencoder\_X\_noise = LabelEncoder()

X\_noise[:, 0] = labelencoder\_X\_noise.fit\_transform(X\_noise[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_noise = onehotencoder.fit\_transform(X\_noise).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_noise = np.delete(X\_noise, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_noise.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_noise = regressor.predict(X\_noise)

y\_pred\_noise[y\_pred\_noise > 0.99] = 1

y\_pred\_noise[y\_pred\_noise < 0] = 0

dataset['mlr\_fp\_noise'] = y\_pred\_noise

# ==================================

# EGT

# ==================================

X\_egt\_data = dataset.loc[:, ['month', 'egt']]

X\_egt = X\_egt\_data.values

# Encoding categorical data

labelencoder\_X\_egt = LabelEncoder()

X\_egt[:, 0] = labelencoder\_X\_egt.fit\_transform(X\_egt[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_egt = onehotencoder.fit\_transform(X\_egt).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_egt = np.delete(X\_egt, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_egt.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_egt = regressor.predict(X\_egt)

y\_pred\_egt[y\_pred\_egt > 0.99] = 1

y\_pred\_egt[y\_pred\_egt < 0] = 0

dataset['mlr\_fp\_egt'] = y\_pred\_egt

# ==================================

# FF

# ==================================

X\_ff\_data = dataset.loc[:, ['month', 'ff']]

X\_ff = X\_ff\_data.values

# Encoding categorical data

labelencoder\_X\_ff = LabelEncoder()

X\_ff[:, 0] = labelencoder\_X\_ff.fit\_transform(X\_ff[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_ff = onehotencoder.fit\_transform(X\_ff).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_ff = np.delete(X\_ff, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_ff.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_ff = regressor.predict(X\_ff)

y\_pred\_ff[y\_pred\_ff > 0.99] = 1

y\_pred\_ff[y\_pred\_ff < 0] = 0

dataset['mlr\_fp\_ff'] = y\_pred\_ff

# ==================================

# N1

# ==================================

X\_n1\_data = dataset.loc[:, ['month', 'n1']]

X\_n1 = X\_n1\_data.values

# Encoding categorical data

labelencoder\_X\_n1 = LabelEncoder()

X\_n1[:, 0] = labelencoder\_X\_n1.fit\_transform(X\_n1[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n1 = onehotencoder.fit\_transform(X\_n1).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n1 = np.delete(X\_n1, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_n1.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n1 = regressor.predict(X\_n1)

y\_pred\_n1[y\_pred\_n1 > 0.99] = 1

y\_pred\_n1[y\_pred\_n1 < 0] = 0

dataset['mlr\_fp\_n1'] = y\_pred\_n1

# ==================================

# N2

# ==================================

X\_n2\_data = dataset.loc[:, ['month', 'n2']]

X\_n2 = X\_n2\_data.values

# Encoding categorical data

labelencoder\_X\_n2 = LabelEncoder()

X\_n2[:, 0] = labelencoder\_X\_n2.fit\_transform(X\_n2[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n2 = onehotencoder.fit\_transform(X\_n2).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n2 = np.delete(X\_n2, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_n2.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n2 = regressor.predict(X\_n2)

y\_pred\_n2[y\_pred\_n2 > 0.99] = 1

y\_pred\_n2[y\_pred\_n2 < 0] = 0

dataset['mlr\_fp\_n2'] = y\_pred\_n2

# ==================================

# Failure Probability

# ==================================

X\_data = dataset.loc[:, ['month', 'noise', 'egt', 'ff', 'n1', 'n2', 'mlr\_fp\_noise', 'mlr\_fp\_ff', 'mlr\_fp\_egt', 'mlr\_fp\_n1', 'mlr\_fp\_n2']]

X = X\_data.values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/multiple\_linear\_regression/mlr\_total\_fp.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred = regressor.predict(X)

y\_pred[y\_pred < 0] = 0

y\_pred[np.where(y\_pred >= 0.98)[0][0]: ] = 1

dataset['mlr\_total\_fp'] = y\_pred

# Writing to csv file

dataset.to\_csv('../../../data/test\_data/test\_data\_1.csv', index=False)

# ==================================

# Importing normal dataset

normal\_dataset = pd.read\_csv('../../../data/normal\_data.csv')

X\_normal = normal\_dataset

y\_normal = normal\_dataset.loc[:, 'total\_fp'].values

y\_normal[y\_normal > 1] = 1

# Plotting graph

plt.figure(figsize=(20, 15))

plt.plot(X\_data['month'][0:60], y\_normal[0:60], color = 'green', linestyle='-', marker='.', label='Age under normal conditions')

plt.plot(X\_data['month'][0:60], y\_pred.ravel(), color = 'blue', linestyle='-', marker='.', label='Predicted age under abnormal conditions')

plt.axvline(x=np.where(y\_pred==1)[0][0]+1, color='red', label='Predicted Failure Month')

plt.axvline(x=np.where(y\_normal==1)[0][0]+1, color='orange', label='Normal Failure Month')

plt.xticks(np.arange(1, 62, 2))

plt.yticks(np.arange(0, 1.05, 0.05))

plt.title('Age (in months) vs Probability of Failure')

plt.legend(loc='best')

plt.xlabel('Age (in months)')

plt.ylabel('Probability of Failure')

plt.show()

plt.savefig('../../../outputs/multiple\_linear\_regression/mlr\_test\_1.png')

# Calculating r2 score

y\_actual = dataset['act\_total\_fp'].values

accuracy = r2\_score(y\_actual, y\_pred)

**Decision Tree Regression:**

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

import pickle

import csv

from sklearn.metrics import r2\_score

# Importing the dataset

dataset = pd.read\_csv('../../../data/test\_data/test\_data\_1.csv')

# ==================================

# ==================================

# Noise

# ==================================

X\_noise\_data = dataset.loc[:, ['month', 'noise']]

X\_noise = X\_noise\_data.values

# Encoding categorical data

labelencoder\_X\_noise = LabelEncoder()

X\_noise[:, 0] = labelencoder\_X\_noise.fit\_transform(X\_noise[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_noise = onehotencoder.fit\_transform(X\_noise).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_noise = np.delete(X\_noise, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_noise.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_noise = regressor.predict(X\_noise)

y\_pred\_noise[y\_pred\_noise > 0.99] = 1

y\_pred\_noise[y\_pred\_noise < 0] = 0

dataset['dt\_fp\_noise'] = y\_pred\_noise

# ==================================

# EGT

# ==================================

X\_egt\_data = dataset.loc[:, ['month', 'egt']]

X\_egt = X\_egt\_data.values

# Encoding categorical data

labelencoder\_X\_egt = LabelEncoder()

X\_egt[:, 0] = labelencoder\_X\_egt.fit\_transform(X\_egt[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_egt = onehotencoder.fit\_transform(X\_egt).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_egt = np.delete(X\_egt, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_egt.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_egt = regressor.predict(X\_egt)

y\_pred\_egt[y\_pred\_egt > 0.99] = 1

y\_pred\_egt[y\_pred\_egt < 0] = 0

dataset['dt\_fp\_egt'] = y\_pred\_egt

# ==================================

# FF

# ==================================

X\_ff\_data = dataset.loc[:, ['month', 'ff']]

X\_ff = X\_ff\_data.values

# Encoding categorical data

labelencoder\_X\_ff = LabelEncoder()

X\_ff[:, 0] = labelencoder\_X\_ff.fit\_transform(X\_ff[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_ff = onehotencoder.fit\_transform(X\_ff).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_ff = np.delete(X\_ff, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_ff.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_ff = regressor.predict(X\_ff)

y\_pred\_ff[y\_pred\_ff > 0.99] = 1

y\_pred\_ff[y\_pred\_ff < 0] = 0

dataset['dt\_fp\_ff'] = y\_pred\_ff

# ==================================

# N1

# ==================================

X\_n1\_data = dataset.loc[:, ['month', 'n1']]

X\_n1 = X\_n1\_data.values

# Encoding categorical data

labelencoder\_X\_n1 = LabelEncoder()

X\_n1[:, 0] = labelencoder\_X\_n1.fit\_transform(X\_n1[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n1 = onehotencoder.fit\_transform(X\_n1).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n1 = np.delete(X\_n1, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_n1.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n1 = regressor.predict(X\_n1)

y\_pred\_n1[y\_pred\_n1 > 0.99] = 1

y\_pred\_n1[y\_pred\_n1 < 0] = 0

dataset['dt\_fp\_n1'] = y\_pred\_n1

# ==================================

# N2

# ==================================

X\_n2\_data = dataset.loc[:, ['month', 'n2']]

X\_n2 = X\_n2\_data.values

# Encoding categorical data

labelencoder\_X\_n2 = LabelEncoder()

X\_n2[:, 0] = labelencoder\_X\_n2.fit\_transform(X\_n2[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n2 = onehotencoder.fit\_transform(X\_n2).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n2 = np.delete(X\_n2, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_n2.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n2 = regressor.predict(X\_n2)

y\_pred\_n2[y\_pred\_n2 > 0.99] = 1

y\_pred\_n2[y\_pred\_n2 < 0] = 0

dataset['dt\_fp\_n2'] = y\_pred\_n2

# ==================================

# Failure Probability

# ==================================

X\_data = dataset.loc[:, ['month', 'noise', 'egt', 'ff', 'n1', 'n2', 'dt\_fp\_noise', 'dt\_fp\_ff', 'dt\_fp\_egt', 'dt\_fp\_n1', 'dt\_fp\_n2']]

X = X\_data.values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/decision\_tree/dt\_total\_fp.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred = regressor.predict(X)

y\_pred[y\_pred < 0] = 0

y\_pred[np.where(y\_pred >= 0.98)[0][0]: ] = 1

dataset['dt\_total\_fp'] = y\_pred

# Writing to csv file

dataset.to\_csv('../../../data/test\_data/test\_data\_1.csv', index=False)

# ==================================

# Importing normal dataset

normal\_dataset = pd.read\_csv('../../../data/normal\_data.csv')

X\_normal = normal\_dataset

y\_normal = normal\_dataset.loc[:, 'total\_fp'].values

y\_normal[y\_normal > 1] = 1

# Plotting graph

plt.figure(figsize=(20, 15))

plt.plot(X\_data['month'][0:60], y\_normal[0:60], color = 'green', linestyle='-', marker='.', label='Age under normal conditions')

plt.plot(X\_data['month'][0:60], y\_pred.ravel(), color = 'blue', linestyle='-', marker='.', label='Predicted age under abnormal conditions')

plt.axvline(x=np.where(y\_pred==1)[0][0]+1, color='red', label='Predicted Failure Month')

plt.axvline(x=np.where(y\_normal==1)[0][0]+1, color='orange', label='Normal Failure Month')

plt.xticks(np.arange(1, 62, 2))

plt.yticks(np.arange(0, 1.05, 0.05))

plt.title('Age (in months) vs Probability of Failure')

plt.legend(loc='best')

plt.xlabel('Age (in months)')

plt.ylabel('Probability of Failure')

plt.show()

plt.savefig('../../../outputs/decision\_tree/dt\_test\_1.png')

# Calculating r2 score

y\_actual = dataset['act\_total\_fp'].values

accuracy = r2\_score(y\_actual, y\_pred)

**Random Forest Regression:**

﻿# Importing the libraries

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

import pickle

import csv

from sklearn.metrics import r2\_score

# Importing the dataset

dataset = pd.read\_csv('../../../data/test\_data/test\_data\_1.csv')

# ==================================

# ==================================

# Noise

# ==================================

X\_noise\_data = dataset.loc[:, ['month', 'noise']]

X\_noise = X\_noise\_data.values

# Encoding categorical data

labelencoder\_X\_noise = LabelEncoder()

X\_noise[:, 0] = labelencoder\_X\_noise.fit\_transform(X\_noise[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_noise = onehotencoder.fit\_transform(X\_noise).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_noise = np.delete(X\_noise, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_noise.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_noise = regressor.predict(X\_noise)

y\_pred\_noise[y\_pred\_noise > 0.99] = 1

y\_pred\_noise[y\_pred\_noise < 0] = 0

dataset['rf\_fp\_noise'] = y\_pred\_noise

# ==================================

# EGT

# ==================================

X\_egt\_data = dataset.loc[:, ['month', 'egt']]

X\_egt = X\_egt\_data.values

# Encoding categorical data

labelencoder\_X\_egt = LabelEncoder()

X\_egt[:, 0] = labelencoder\_X\_egt.fit\_transform(X\_egt[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_egt = onehotencoder.fit\_transform(X\_egt).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_egt = np.delete(X\_egt, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_egt.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_egt = regressor.predict(X\_egt)

y\_pred\_egt[y\_pred\_egt > 0.99] = 1

y\_pred\_egt[y\_pred\_egt < 0] = 0

dataset['rf\_fp\_egt'] = y\_pred\_egt

# ==================================

# FF

# ==================================

X\_ff\_data = dataset.loc[:, ['month', 'ff']]

X\_ff = X\_ff\_data.values

# Encoding categorical data

labelencoder\_X\_ff = LabelEncoder()

X\_ff[:, 0] = labelencoder\_X\_ff.fit\_transform(X\_ff[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_ff = onehotencoder.fit\_transform(X\_ff).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_ff = np.delete(X\_ff, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_ff.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_ff = regressor.predict(X\_ff)

y\_pred\_ff[y\_pred\_ff > 0.99] = 1

y\_pred\_ff[y\_pred\_ff < 0] = 0

dataset['rf\_fp\_ff'] = y\_pred\_ff

# ==================================

# N1

# ==================================

X\_n1\_data = dataset.loc[:, ['month', 'n1']]

X\_n1 = X\_n1\_data.values

# Encoding categorical data

labelencoder\_X\_n1 = LabelEncoder()

X\_n1[:, 0] = labelencoder\_X\_n1.fit\_transform(X\_n1[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n1 = onehotencoder.fit\_transform(X\_n1).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n1 = np.delete(X\_n1, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_n1.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n1 = regressor.predict(X\_n1)

y\_pred\_n1[y\_pred\_n1 > 0.99] = 1

y\_pred\_n1[y\_pred\_n1 < 0] = 0

dataset['rf\_fp\_n1'] = y\_pred\_n1

# ==================================

# N2

# ==================================

X\_n2\_data = dataset.loc[:, ['month', 'n2']]

X\_n2 = X\_n2\_data.values

# Encoding categorical data

labelencoder\_X\_n2 = LabelEncoder()

X\_n2[:, 0] = labelencoder\_X\_n2.fit\_transform(X\_n2[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X\_n2 = onehotencoder.fit\_transform(X\_n2).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X\_n2 = np.delete(X\_n2, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_n2.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred\_n2 = regressor.predict(X\_n2)

y\_pred\_n2[y\_pred\_n2 > 0.99] = 1

y\_pred\_n2[y\_pred\_n2 < 0] = 0

dataset['rf\_fp\_n2'] = y\_pred\_n2

# ==================================

# Failure Probability

# ==================================

X\_data = dataset.loc[:, ['month', 'noise', 'egt', 'ff', 'n1', 'n2', 'rf\_fp\_noise', 'rf\_fp\_ff', 'rf\_fp\_egt', 'rf\_fp\_n1', 'rf\_fp\_n2']]

X = X\_data.values

# Encoding categorical data

labelencoder\_X = LabelEncoder()

X[:, 0] = labelencoder\_X.fit\_transform(X[:, 0])

onehotencoder = OneHotEncoder(categorical\_features = [0])

X = onehotencoder.fit\_transform(X).toarray()

# Avoiding dummy variable trap

categories = [0]

dummies = []

dummies\_sum = 0

for category in categories:

dummies\_sum += (dataset.iloc[:, category].unique().size) \* category

dummies.append(dummies\_sum)

X = np.delete(X, dummies, 1)

# Loading and fitting the regression model

with open('../../../models/random\_forest/rf\_total\_fp.pkl', 'rb') as f:

regressor = pickle.load(f)

# Predicting the results

y\_pred = regressor.predict(X)

y\_pred[y\_pred < 0] = 0

y\_pred[np.where(y\_pred >= 0.98)[0][0]: ] = 1

dataset['rf\_total\_fp'] = y\_pred

# Writing to csv file

dataset.to\_csv('../../../data/test\_data/test\_data\_1.csv', index=False)

# Importing normal dataset

normal\_dataset = pd.read\_csv('../../../data/normal\_data.csv')

X\_normal = normal\_dataset

y\_normal = normal\_dataset.loc[:, 'total\_fp'].values

y\_normal[y\_normal > 1] = 1

# Plotting graph

plt.figure(figsize=(20, 15))

plt.plot(X\_data['month'][0:60], y\_normal[0:60], color = 'green', linestyle='-', marker='.', label='Age under normal conditions')

plt.plot(X\_data['month'][0:60], y\_pred.ravel(), color = 'blue', linestyle='-', marker='.', label='Predicted age under abnormal conditions')

plt.axvline(x=np.where(y\_pred==1)[0][0]+1, color='red', label='Predicted Failure Month')

plt.axvline(x=np.where(y\_normal==1)[0][0]+1, color='orange', label='Normal Failure Month')

plt.xticks(np.arange(1, 62, 2))

plt.yticks(np.arange(0, 1.05, 0.05))

plt.title('Age (in months) vs Probability of Failure')

plt.legend(loc='best')

plt.xlabel('Age (in months)')

plt.ylabel('Probability of Failure')

plt.show()

plt.savefig('../../../outputs/random\_forest/rf\_test\_1.png')

# Calculating r2 score

y\_actual = dataset['act\_total\_fp'].values

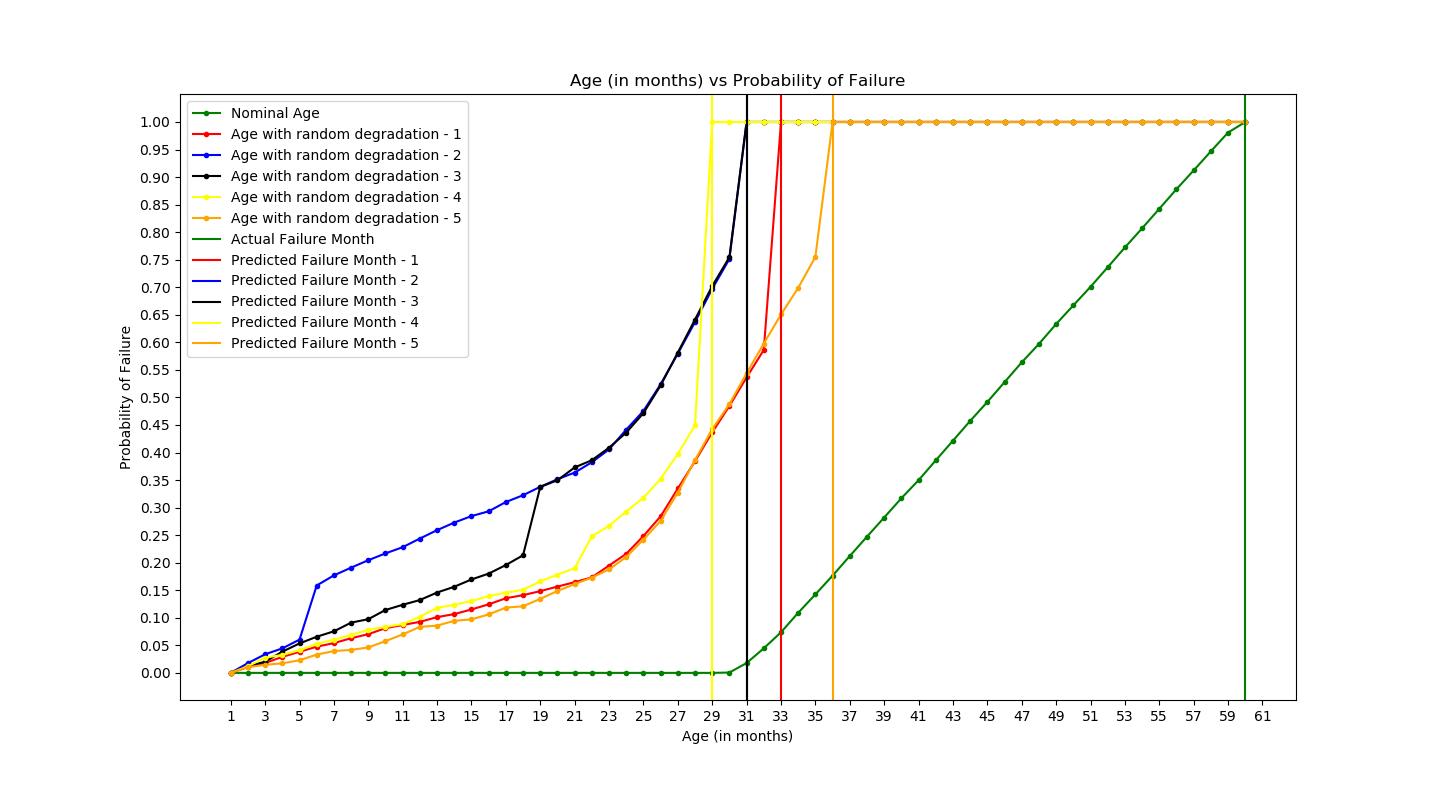
accuracy = r2\_score(y\_actual, y\_pred)

1. **Plotting Graphs**

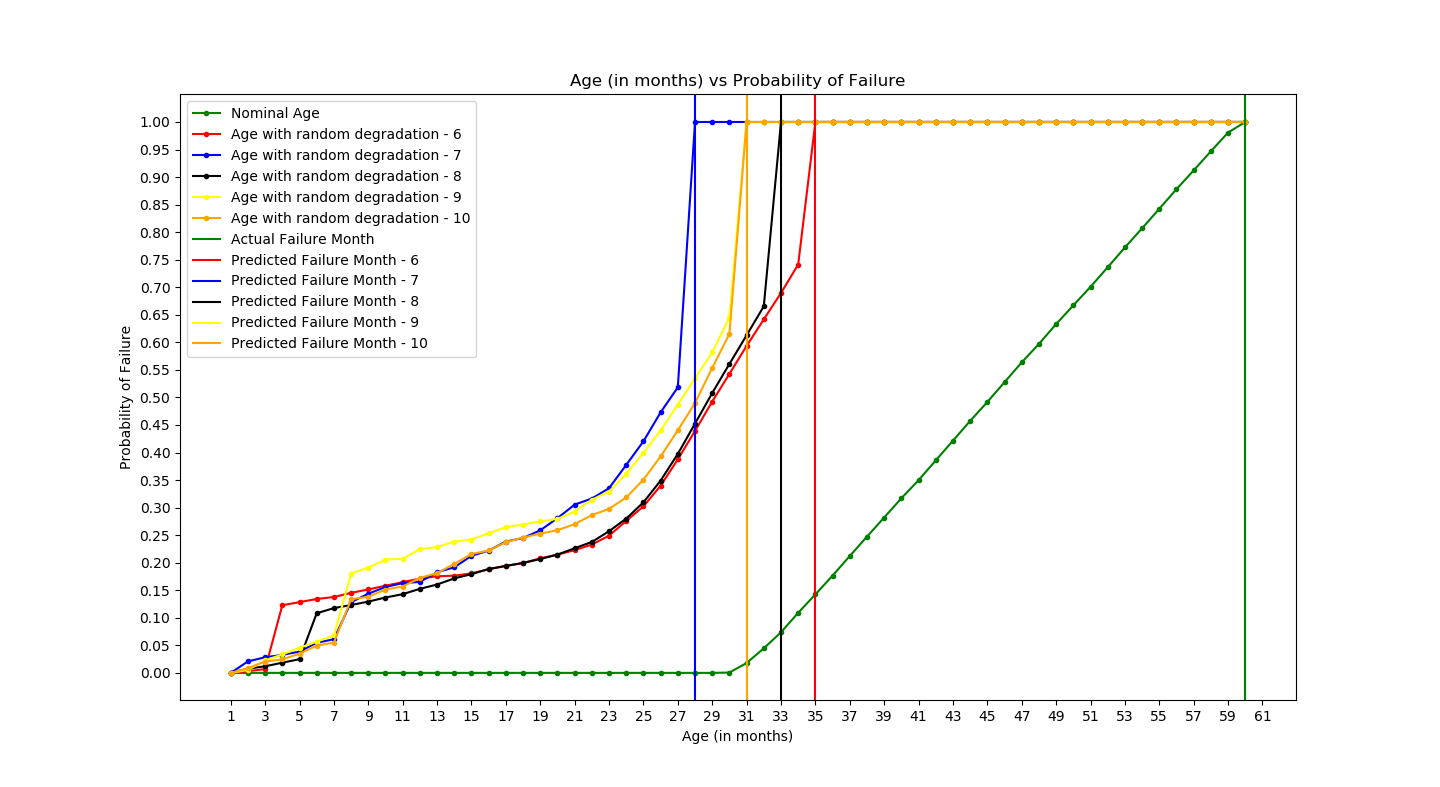
Graphs are plotted against Age and Probability of Failure. 5 engines are taken at once in a single graph.

**Multiple Linear Regression:**

Plot 1:

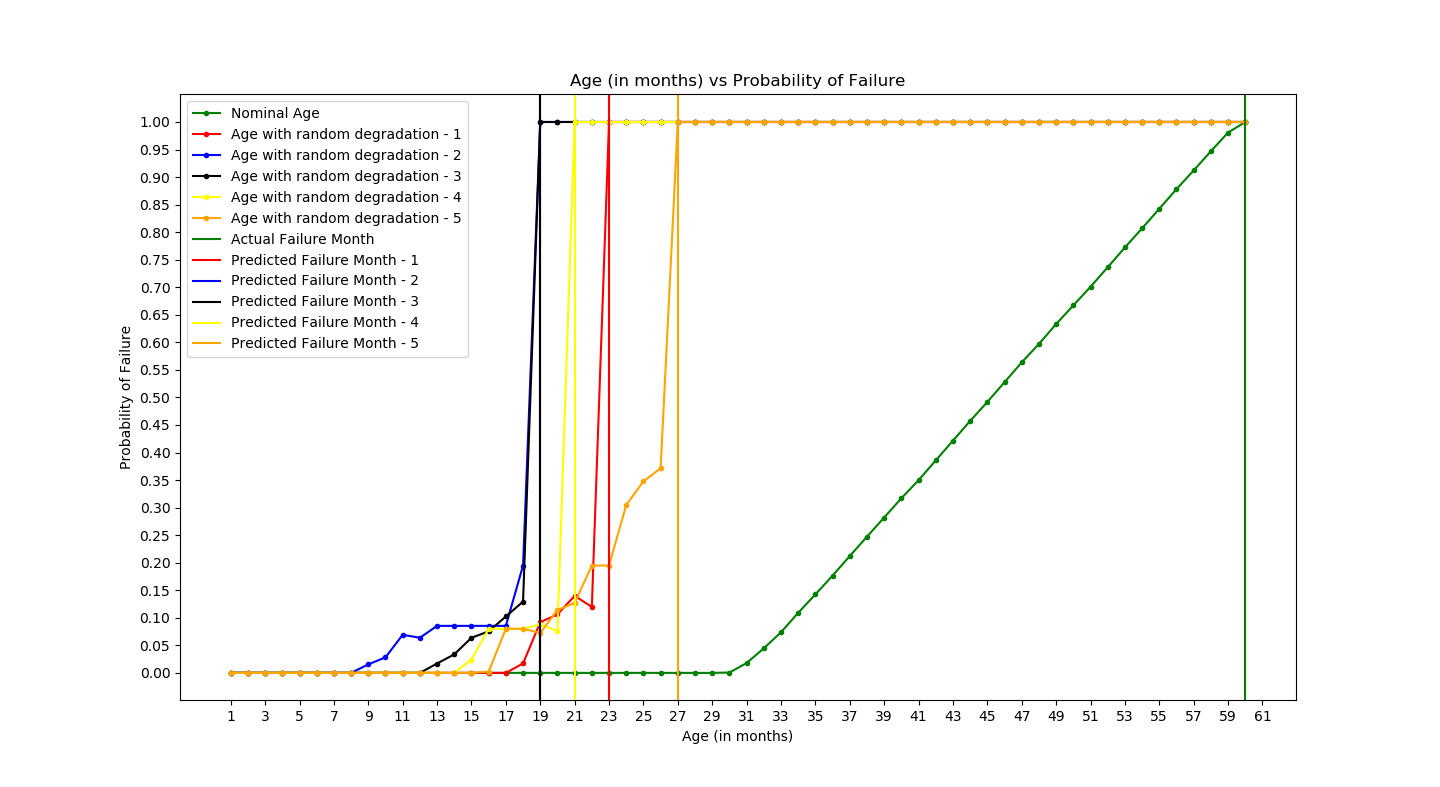
****

Plot 2:

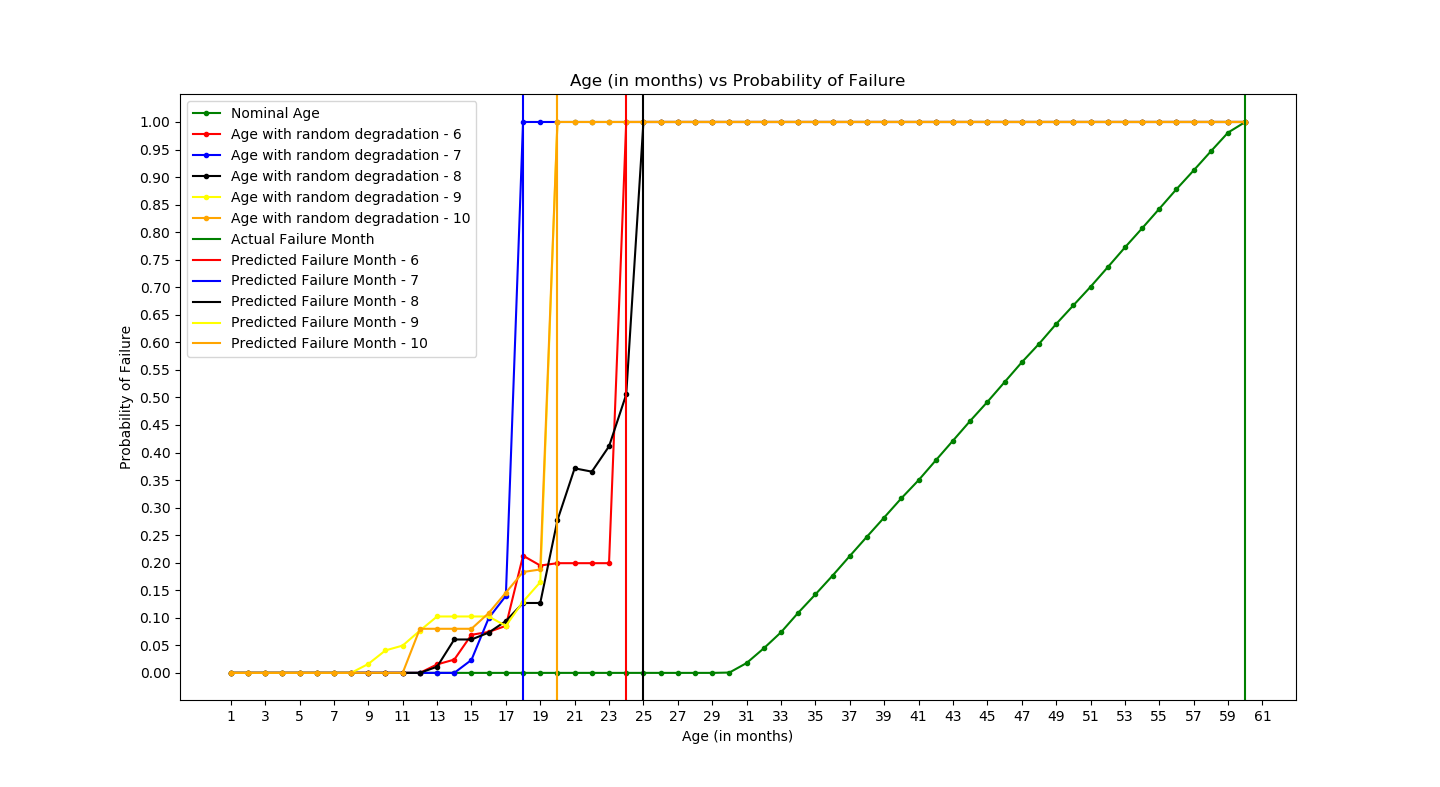


**Decision Tree Regression:**

Plot 1:

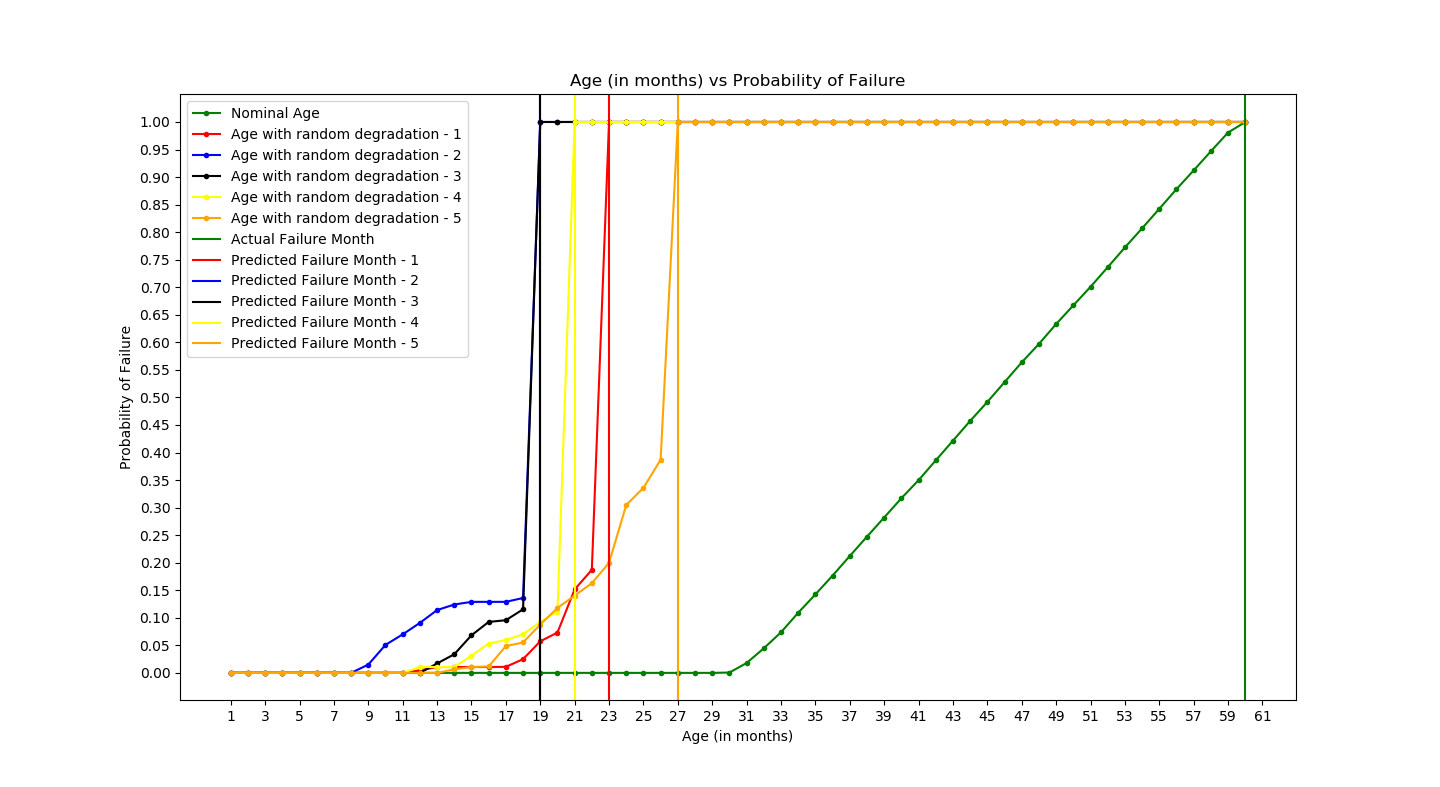


Plot 2:

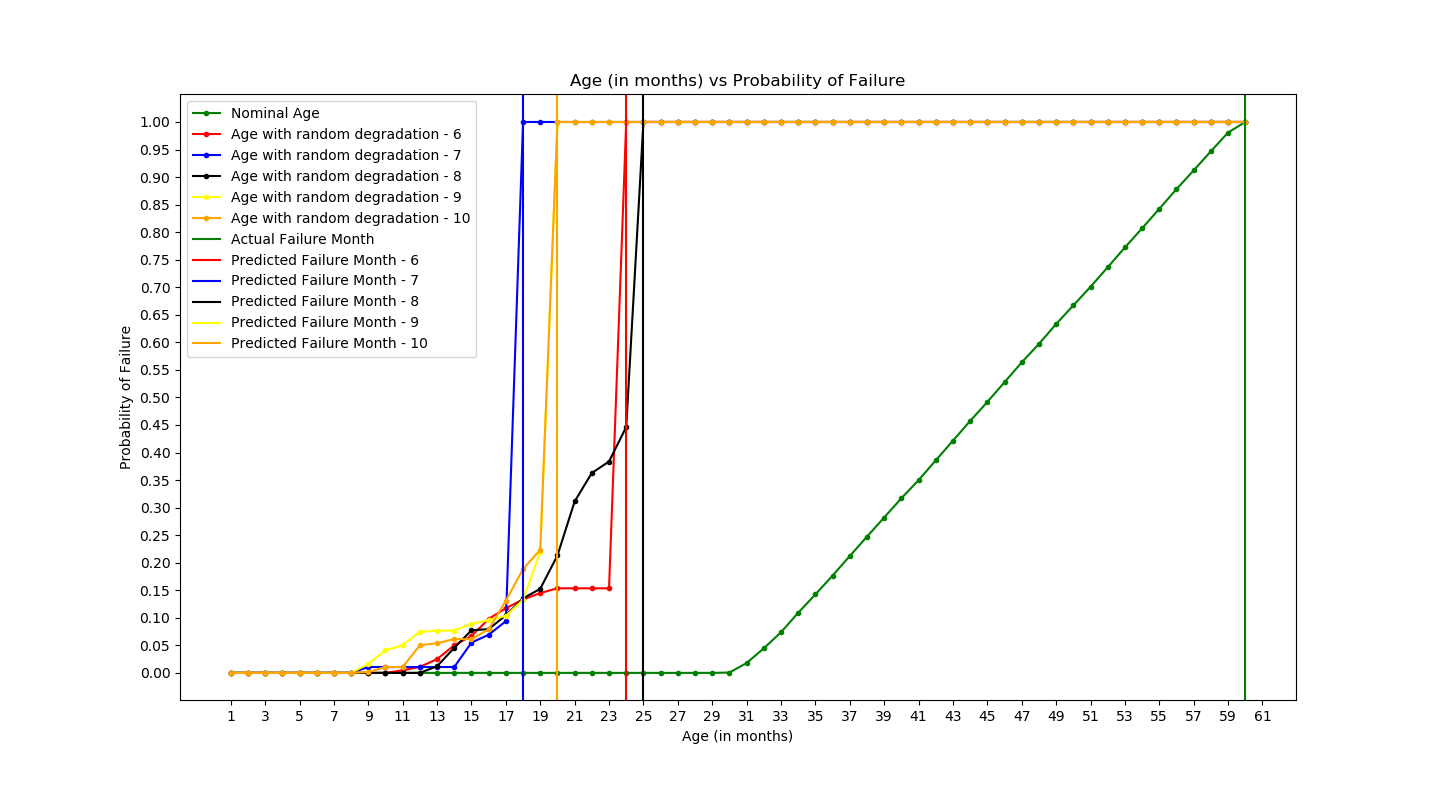


**Random Forest Regression:**

Plot 1:



Plot 2:



1. **Computing R2 Score**

The results of all the three algorithms are compared using R2 Score.

R2 Score is a statistical measure of how close the data are to the fitted regression line. R2 is always between 0 and 1, 0 being the least and 1 being the highest.

R2 = Explained Variation/Total Variation

All the R2 scores are listed below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Engine** | **Multiple Linear Regression** | **Decision Tree** | **Random Forest** |
| 1 | 0.9472 | 0.9775 | 0.9782 |
| 2 | 0.8885 | 0.9937 | 0.9959 |
| 3 | 0.8783 | 0.9799 | 0.9799 |
| 4 | 0.8962 | 0.9799 | 0.7548 |
| 5 | 0.9128 | 0.9982 | 0.9983 |
| 6 | 0.9572 | 0.8865 | 0.9049 |
| 7 | 0.902 | 0.9909 | 0.99 |
| 8 | 0.8934 | 0.9565 | 0.9537 |
| 9 | 0.9212 | 0.8958 | 0.8891 |
| 10 | 0.9178 | 0.9376 | 0.9358 |

**CONCLUSION**

In a real world scenario, we would compare the predicted value to the real world value and then compute the accuracy of each algorithm. Due to the absence of real world data, the results of all the three algorithms are compared using R2 Score.

Five out of ten times, Decision Tree’s scores were closer to 1 than any other algorithm

Four out of ten times, Random Forest’s scores were closest to 1

Multiple Linear Regression scores were closest to 1 only one out of ten times.

Therefore, Decision Tree performed better when compared to Random Forest and Multiple Linear Regression.

**FUTURE ENHANCEMENTS**

All the data used in this project was based on a few assumptions.

Next step would be to collect real world data from an actual aircraft engine and then perform predictive analysis in a similar manner.