Table of Content

1.0 Business Understanding

1.1 Company's Background

Figure 1.1: Hotayi Electronic Sdn Bhd (Hotayi Electronic Sdn Bhd, 2017)

Hotayi Electronics Sdn Bhd is an IATF 16949, ISO 9001, and ISO 14001 certified company that provides global electronics manufacturing services in Penang. It was founded on 23 July 1992 by Lee Hung Lung. (Hotayi Electronic Sdn Bhd, 2017) Hotayi Electronics' products range from automotive, consumer, storage devices, commercial, public safety, and industrial. It is a pioneer in this sector because of its excellence in operational and continuous improvement in its manufacturing process and business flow. It manufactures a wide range of value added manufacturing solutions and services such as printed circuit boards (PCB) and other goods. A special measuring equipment is used to measure the size of a PCB that is produced. One of the equipment is 3D AOI which consists of one or two sensors for measurement. The inspection is carried out to ensure that only the optimum measurement is accepted, for other negative or positive values will be rejected. The optimal measurement is vital to ensure that the PCB is able to fit into the mechanical box during assembly, hence, maintain the quality of the board.

In addition, Hotayi Electronics has good management in their procurement, logistic service, New Product Introduction (NPI) Program, product quality, traceability, and their data. For example, Hotayi Electronics provides a Full Turnkey End-to-End Supply Chain Management in order to enhance its material quote process and maximize its leverage with suppliers. It also can deliver quick-turn prototyping and ongoing quality control and assurance for its manufactured products. Besides that, Hotayi Electronics utilizes its information technology competence by integrating innovative automation solutions into its production process and business workflow, intending to provide excellent quality level outcomes and best-in-class services for its precious customers. As a result, sales at Hotayi Electronics Sdn Bhd are surging, hiring more technical expertise and expanding its business.

In order to increase its competitiveness, Hotayi Electronics Sdn Bhd always committed to continuing to improve its competitive edge in terms of technologies, standards, and processes so

that it can satisfy its clients by fulfilling their requirements in terms of quality, cost, and delivery. Therefore, this project is carried out to develop various models and select the most suitable model to help Hotayi Electronics Sdn Bhd make data predictions for better decision-making in the future. By creating those suitable data prediction models, appropriate data science concepts, statistics, and machine learning algorithms are being used in this project to analyze and visualize the data of Hotayi Electronics Sdn Bhd. Furthermore, the independent and dependent variables will also be interpreted from the collected data at the beginning stage to train the model to learn the machine's features. Moreover, in order to reduce the model failure rate, data preparation such as data cleaning, and outlier checking is carried out in this project. Once the data is prepared and the models are trained with the independent variables, the models are tested with test data and evaluated their performance based on some criteria.

1.2 Objectives

The objective of this project is to predict the production machine failure. This objective can be achieved by training the model to analyze the machine's pattern from the data provided by Hotayi Electronics Sdn Bhd. Through knowing the past machine's behavior, the model can forecast future machine failure results for Hotayi Electronics Sdn Bhd. By achieving this objective, Hotayi Electronic Sdn Bhd can carry out some preventive and proactive ways based on the prediction results to reduce production machine downtime.

Besides, one of the objectives of this project is to present the data provided by Hotayi Electronics Sdn Bhd in a better and structured way to interpret and feature extracting. The provided data will be checked in this project to remove and fix those unusual and missing values. The data provided will also be visualized in various charts such as bar charts, pie charts, and line charts. Thus the data is prepared and presented in a more readable way for Hotayi Electronics Sdn Bhd to interpret.

Furthermore, this project also can help Hotayi Electronic Sdn Bhd to optimize the production machine operations. This is because Hotayi Electronic Sdn Bhd can make better decisions in managing the production machine operation, such as making amendments on the input points for the machine to process based on the analysis results from this project.

1.3 Requirements, assumptions and constraints

In order to develop this project, there are a few requirements, assumptions, and constraints:
1. This project has a time constraint because it needs to be done and submitted by Friday , 10 September 2021.

2. A team of four developers develops this project, so the tasks completed within the given period are limited. As a result, each developer will handle one modeling technique and then evaluate the impact to find the best modeling technique for developing the system of Hotayi Electronics Sdn Bhd.

3. This project has no budget at all. Thus, the resources used, such as the software and hardware, should be open source and free of charge.

4. Hotayi Electronics Sdn Bhd provides the dataset used to develop this project. Hence, there should be no copyright issues as well as any data security issues.

5. An assumption toward the reliability and correctness of the dataset is made due to the dataset is directly provided by Hotayi Electronics Sdn Bh.

1.4 Data Mining Success Criteria

Data mining is a process used to obtain valuable data from a more extensive set of raw data by using one or more software (Twin, 2019). Through the data mining process, the data patterns of Hotayi Electronics are able to analyze and assist in the machine failure rate prediction. The data mining success criteria depend on the accuracy, precision, recall, F1 score and Area Under the Curve (AUC) value. Among the five criteria, the model accuracy provides the best perspective on how well a model performs on a given dataset. This is because high model accuracy means the models already learn about the data accurately and desirably. The model accuracy can be calculated by using the number of classifications a model correctly predicts divided by the total number of predictions made (Jeremy Jordan, 2017). For precision, it indicates the ability of the model to identify and classify only the relevant data points (Will Koehrsen, 2021). It is the ratio of true positives among all positives. High model precision means the machine learning algorithms are able to identify more relevant results than irrelevant ones.

The recall is another crucial data mining evaluation criterion. It is measured using the number of true positives divided by the number of true positives plus false negatives. High recall means the model has an increased ability to find all relevant cases within the given data set (Will Koehrsen, 2021). While the F1 score is used to convey the balance between the precision and the recall. The formula to calculate the standard F1 score is 2((precision*recall)/(precision+recall)) (Jason Brownlee, 2019). An excellent F1 score means that the model has low false positives and low false negatives. The Area Under the Curve (AUC) is also an essential criterion for the model's performance evaluation as it is used to measure the model's ability to distinguish between classes (Sarang Narkhede, 2018). The higher the AUC value, the better the model's capability in determining between the positive and negative categories.

In conclusion, the high accuracy, prediction, recall, F1 score and AUC value indicate that the data mining process is successful. Therefore, for this project, the accuracy, precision, recall, F1 score and AUC value of each model should exceed 80%, which can only indicate that the data mining process is successful.

1.5 Inventory of Resources

Students from Bachelor of Computer Science (Honours) in Software Engineering, which include Foo Jia Ern (20WMR09450), Tan Huey Peng (20WMR09520), Chua Yee Chen (20WMR09448), and Cheah Su Ying (20WMR09430), are helping Hotayi Electronics Sdn Bhd to perform the machine failed rate analysis in order to predict the machine failure so the production team can plan early to reduce the machine downtime and optimize the machines' performance. Ms Noor Aida Binti Husaini and Ms Ashikin Binti Ali, the lecturer and tutor of the Faculty of Computer and Information Technology in TARUC, also assist the students in completing this project. In this case, Hotayi Electronics Sdn Bhd directly provides the source of the dataset. The software used to develop this project is Google Colaboratory and Jupyter Notebook. Google Colaboratory is a cloud service that is free to use and is provided by Google. It is chosen to be used because of its great collection of snippets and the computing power of the Google server, which causes the processing performance better. Furthermore, Jupyter is the open-source project on which Colab is based. Due to the cloud service, the Google Colab notebooks can also be shared easily by using the Google Drive account without any installation, download, or run anything. (Abhishek Sharma, 2020) The hardware used to develop this project are the personal computers because of the cost constraint of this project.

1.6 Risk

1. The dataset provided by Hotayi Electronics Sdn Bhd is separated into multiple excel files, hence, it requires compilation of multiple csv files into a single file and requires effort for data selection and cleaning.

2. The missing values are most likely causing the prediction to be failed as inaccuracy is higher . If there are quite a lot of missing values, the missing values will be replaced. If the missing value is quite less, it will be deleted.

1.7 Solution

1. Clean the data by either deleting the entire row of data or imputing valid data to the particular column before carrying out data modelling.

2.0 Data Understanding

In this project, we have chosen Result_MC1_2020_12 as our datasets. There are a total of 30 folders which consist of multiple excel files in each of them. However, only the datasets collected from Lot 1234567890 are chosen to maintain the consistency. We have combined all of the separate excel files into a new csv file by using pandas.DataFrame.sample (refer to the code below). The newly combined csv file consists of 283496 rows x 13 columns with data type of object and float64. Some extra columns such as unnamed are produced during the combination of the new csv file where there is no data in these respective columns. Hence, these unnamed columns are removed. The columns include 8 independent variables and 1 dependent variable. The list of variables used in this dataset are as follows:

Table 2.1 : List of variables.

Name	Type	Dependent/ Independent Variable	Description
IC 2D	Continuous	Independent Variable	The **total number of the components** in the 2D integrated circuits
Date	Continuous	Independent Variable	The **operation date** of the machine in December with the format DD/MM/YYYY
Time	Continuous	Independent Variable	The **operation time** of the machine in December based on a 24 hours system with the format HH:MM:SS
Machine result	Categorical	Dependent Variable	**The state of the machine** OK = good state NG = poor state
(um)Point1	Continuous	Independent Variable	The **features of the machine** measured in micrometer (um)
(um)Point2	Continuous	Independent Variable	
(um)Point3	Continuous	Independent Variable	
(um)Point4	Continuous	Independent Variable	

(um)Point5	Continuous	Independent Variable	

The dataset was given in numerous separate excel files. Hence, by implementing the below code, all excel files are able to combine together.

Code to combine excel file

```
import os
import glob
import pandas as pd
os.chdir("D:\\Hotayi\\MC1_12\\2020_12_25\\1234567890")
extension = 'csv'
all_filenames = [i for i in glob.glob('*.{}'.format(extension))]
combined_csv = pd.concat([pd.read_csv(f) for f in all_filenames ])
combined_csv.to_csv( "combined_csv.csv", index=False, encoding='utf-8-sig')
```


The missing values mostly occurs in the feature columns whereby
1. (um)Point2: 39456 (13.92%)
2. (um)Point3: 68436 (24.14%)
3. (um)Point4: 68436 (24.14%)
4. (um)Point5: 225256 (79.46%)

The missing values are either represented with 0 or dropped. Through K-Means clustering, the graph produced displayed that the points that have minimum values tend to have NG machine results.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# read data into a DataFrame
# df = pd.read_csv('DS heart dataset.csv') # jupyter
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/combined_csv.csv')
print(df.head(10))
print("Number of rows:-", len(df))
```

```
        Part  IC 2D         Lot  ... (um)Point3 (um)Point4 (um)Point5
0  124313414  13267  1234567890  ...    -1809.6    -1649.6        NaN
1  124313414  13268  1234567890  ...    -1797.3    -1700.0        NaN
2  124313414  13269  1234567890  ...    -1785.4    -1740.5        NaN
3  124313414  13270  1234567890  ...    -1831.1    -1899.3        NaN
4  124313414  13271  1234567890  ...    -1873.3    -2454.7        NaN
5  124313414  13272  1234567890  ...    -1921.4    -1891.7        NaN
6  124313414  13273  1234567890  ...    -2101.3    -1811.4        NaN
7  124313414  13274  1234567890  ...    -1835.7    -1981.9        NaN
8  124313414  13275  1234567890  ...    -1900.5    -1906.6        NaN
9  124313414  13276  1234567890  ...    -2018.4    -1885.1        NaN

[10 rows x 13 columns]
Number of rows:- 283496
```

Figure 2.1: Load dataset.

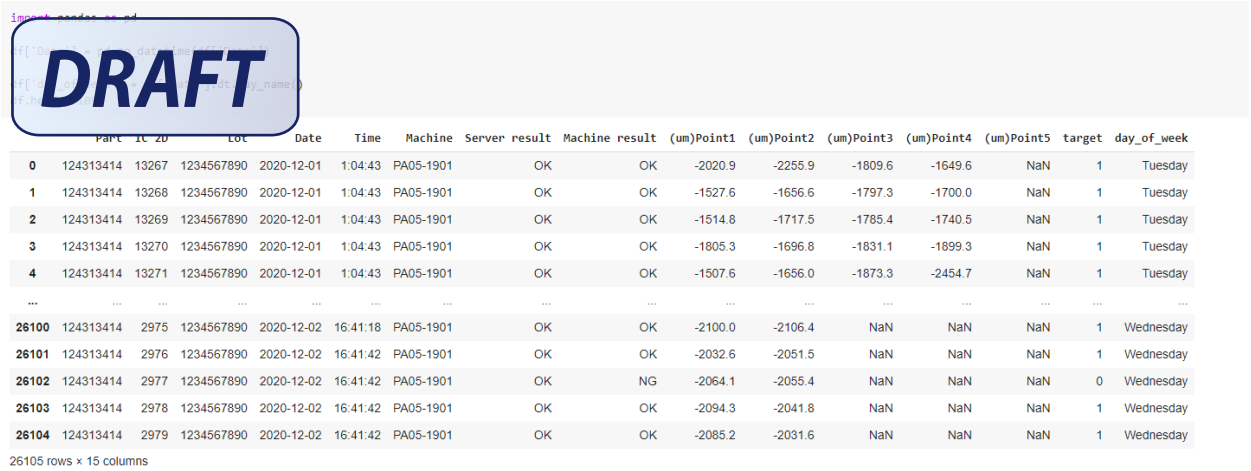
```
df.describe()
```

	Part	IC 2D	Lot	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5
count	283496.0	283496.000000	2.834960e+05	283496.000000	244040.000000	215060.000000	215060.000000	58240.000000
mean	124313414.0	9736.240945	1.234568e+09	-449.145523	-459.510359	-285.948427	-271.315478	78.580246
std	0.0	6758.020504	0.000000e+00	810.459745	811.664952	715.930290	699.306816	65.066912
min	124313414.0	1.000000	1.234568e+09	-4045.600000	-4623.900000	-3232.900000	-3835.000000	-999.400000
25%	124313414.0	3628.000000	1.234568e+09	-1470.900000	-1492.800000	-23.500000	-47.200000	40.800000
50%	124313414.0	8442.000000	1.234568e+09	6.900000	1.000000	11.700000	19.700000	66.100000
75%	124313414.0	15571.000000	1.234568e+09	27.200000	15.300000	40.300000	59.000000	101.300000
max	124313414.0	23457.000000	1.234568e+09	2120.100000	2650.900000	2280.700000	2097.600000	1845.200000

Figure 2.2: Dataset display.

2.1 Independent Variable

2.1.1 Date



	Part_ID	ID_2D	Lot	Date	Time	Machine	Server result	Machine result	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5	target	day_of_week
0	124313414	13267	1234567890	2020-12-01	1:04:43	PA05-1901	OK	OK	-2020.9	-2255.9	-1809.6	-1649.6	NaN	1	Tuesday
1	124313414	13268	1234567890	2020-12-01	1:04:43	PA05-1901	OK	OK	-1527.6	-1656.6	-1797.3	-1700.0	NaN	1	Tuesday
2	124313414	13269	1234567890	2020-12-01	1:04:43	PA05-1901	OK	OK	-1514.8	-1717.5	-1785.4	-1740.5	NaN	1	Tuesday
3	124313414	13270	1234567890	2020-12-01	1:04:43	PA05-1901	OK	OK	-1805.3	-1696.8	-1831.1	-1899.3	NaN	1	Tuesday
4	124313414	13271	1234567890	2020-12-01	1:04:43	PA05-1901	OK	OK	-1507.6	-1656.0	-1873.3	-2454.7	NaN	1	Tuesday
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
26100	124313414	2975	1234567890	2020-12-02	16:41:18	PA05-1901	OK	OK	-2100.0	-2106.4	NaN	NaN	NaN	1	Wednesday
26101	124313414	2976	1234567890	2020-12-02	16:41:42	PA05-1901	OK	OK	-2032.6	-2051.5	NaN	NaN	NaN	1	Wednesday
26102	124313414	2977	1234567890	2020-12-02	16:41:42	PA05-1901	OK	NG	-2064.1	-2055.4	NaN	NaN	NaN	0	Wednesday
26103	124313414	2978	1234567890	2020-12-02	16:41:42	PA05-1901	OK	OK	-2094.3	-2041.8	NaN	NaN	NaN	1	Wednesday
26104	124313414	2979	1234567890	2020-12-02	16:41:42	PA05-1901	OK	OK	-2085.2	-2031.6	NaN	NaN	NaN	1	Wednesday

26105 rows × 15 columns

Figure 2.3: Dataset display with 'day_of_week'.

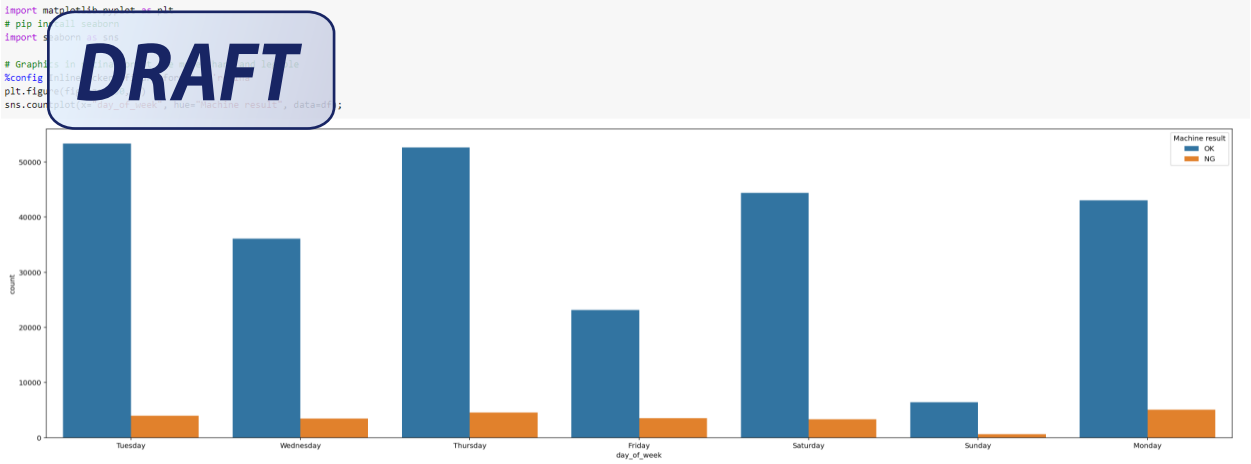


Figure 2.4: Dataset display with 'day_of_week'.

```
mon_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Monday'), 'target'].sum())
mon_total = (df.loc[(df['day_of_week'] == 'Monday'), 'target'].count())
mon_failureRate = "{:.2f}".format(((mon_total - mon_ok) / mon_total) * 100)
print("Total Failure Rate on Monday: ", mon_failureRate);

tue_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Tuesday'), 'target'].sum())
tue_total = (df.loc[(df['day_of_week'] == 'Tuesday'), 'target'].count())
tue_failureRate = "{:.2f}".format(((tue_total - tue_ok) / tue_total) * 100)
print("Total Failure Rate on Tuesday: ", tue_failureRate);

wed_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Wednesday'), 'target'].sum())
wed_total = (df.loc[(df['day_of_week'] == 'Wednesday'), 'target'].count())
wed_failureRate = "{:.2f}".format(((wed_total - wed_ok) / wed_total) * 100)
print("Total Failure Rate on Wednesday: ", wed_failureRate);

thu_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Thursday'), 'target'].sum())
thu_total = (df.loc[(df['day_of_week'] == 'Thursday'), 'target'].count())
thu_failureRate = "{:.2f}".format(((thu_total - thu_ok) / thu_total) * 100)
print("Total Failure Rate on Thursday: ", thu_failureRate);

fri_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Friday'), 'target'].sum())
fri_total = (df.loc[(df['day_of_week'] == 'Friday'), 'target'].count())
fri_failureRate = "{:.2f}".format(((fri_total - fri_ok) / fri_total) * 100)
print("Total Failure Rate on Friday: ", fri_failureRate);

sat_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Saturday'), 'target'].sum())
sat_total = (df.loc[(df['day_of_week'] == 'Saturday'), 'target'].count())
sat_failureRate = "{:.2f}".format(((sat_total - sat_ok) / sat_total) * 100)
print("Total Failure Rate on Saturday: ", sat_failureRate);

sun_ok =  (df.loc[(df['Machine result'] == 'OK') & (df['day_of_week'] == 'Sunday'), 'target'].sum())
sun_total = (df.loc[(df['day_of_week'] == 'Sunday'), 'target'].count())
sun_failureRate = "{:.2f}".format(((sun_total - sun_ok) / sun_total) * 100)
print("Total Failure Rate on Sunday: ", sun_failureRate);
```

```
Total Failure Rate on Monday:  10.56%
Total Failure Rate on Tuesday:  6.94%
Total Failure Rate on Wednesday:  8.79%
Total Failure Rate on Thursday:  7.94%
Total Failure Rate on Friday:  13.15%
Total Failure Rate on Saturday:  6.94%
Total Failure Rate on Sunday:  8.56%
```

Figure 2.5: Failure rate on each day.

'Day of week' is a new attribute derived from the date column to analyze the machine results' fluctuation. Based on Figure 2.4, it can be seen that Sunday, Wednesday, and Friday have a lower total number of machine results than other days. This is mainly because Hotayi Electronics Sdn Bhd consists of two machines to carry out their daily operations, and the machines take turns to operate. Moreover, based on Figure 2.4 and Figure 2.5, it can be seen that the machine produces more 'NG' results on Monday and Friday than on other days which means the machine has more poor state on Monday and Friday. The machine failure rate on Friday is up to 13.15% and 10.56% on Monday.

```
fig, axs = plt.subplots(figsize=(12, 4))
df['Date'] = df['Date'].astype('datetime64[ns]')
weekly_data = df.resample('W-Tue', label='right', closed = 'right', on='Date'[["t...                    d='bar', rot=0, ax=axs)
plt.xlabel("Week");  # custom x label using matplotlib
plt.ylabel("Machine 'OK' Result");
```

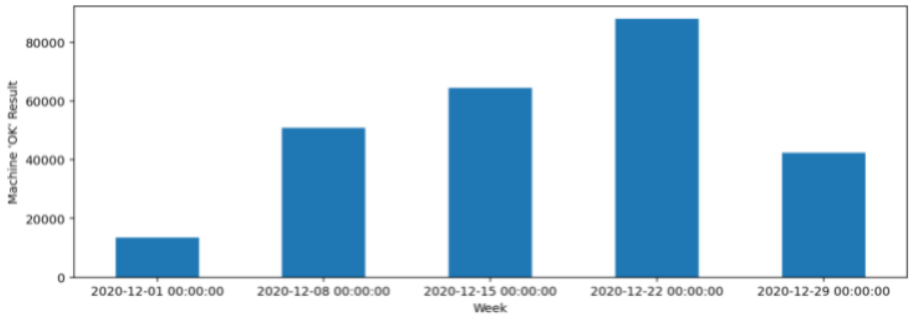



Figure 2.6: Weekly machine 'OK' result.

```
#import required libraries
import pandas as pd
from datetime import datetime
#convert date column into datetime object
df['Date'] = df['Date'].astype('datetime64[ns]')
#convert daily data to weekly
weekly_data = df.resample('W-Tue', label='right', closed = 'right', on='Da...                    ...value...(by='Date')
weekly_data = weekly_data.rename(columns= {'target' : 'Number of OK in Mac...                    ...e Week'})
display(weekly_data)
```


	Week	Part	IC 2D	Lot	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5	Number of OK in Machine Result
0	2020-12-01	1691656937712	226803564	16799999847120	-2.155614e+07	-23267016.4	-26357847.6	-26073956.6	0.0	13418
1	2020-12-08	7138200545294	662970627	70890122811690	-7.151218e+07	-51913268.3	-5319047.4	-5148868.6	1289327.8	50844
2	2020-12-15	9092531726788	748289540	90298764610380	-7.914004e+06	-7885449.2	2044919.0	2216825.6	2111720.6	64500
3	2020-12-22	11589118020150	827803321	115092591545250	2.219585e+06	631827.9	543367.8	1539211.7	0.0	87937
4	2020-12-29	5730848385400	294318311	56913579729000	-2.856821e+07	-29705002.0	-32407460.6	-30882318.7	1175465.1	42306

Figure 2.7: Total number of machine 'OK' results weekly.

Based on Figure 2.6 and Figure 2.7, it can be seen that the machine performs most 'OK' results which means good state in week 4. This is probably due to the value of (um)Point1 to (um)Point5. (um)Point1 to (um)Point4 have strong positive correlations with the machine results, and (um)Point5 has strong negative correlations with the machine results. Thus, when the value of (um)Point1 to (um)Point4 in week 4 is positive, this directly causes the machine 'OK' results to increase compared to others. While the (um)Point5 value in week 4 is 0, so it does not affect much in the machine results.

2.1.2 Time

```
fig, axs = plt.subplots(figsize=(12, 4))
timeDate = pd.to_datetime(df['Time'])
hourly_data = df.groupby(timeDate.dt.hour)["target"].sum().plot(kind='bar', rot=0, ax=axs)
plt.xlabel("Hour of the day");  # custom x label using
plt.ylabel("Machine 'OK' Result");
```

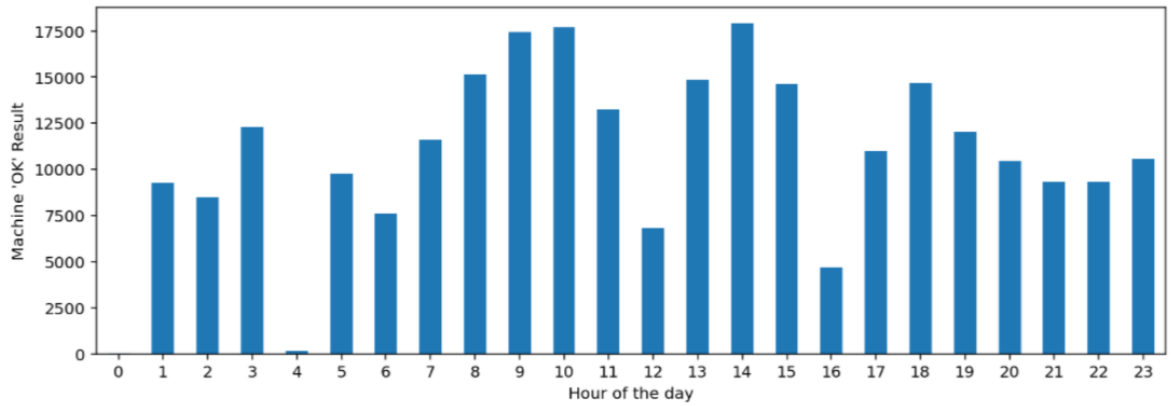



Figure 2.8: Total number of machine 'OK' results per hour in a day.

Based on Figure 2.8, it can be seen that the machine is a 24-hour based operation. However, it is required to operate more during the working hours which is around 8-11 am, 1-3 pm and 5-7 pm. Other periods are either worker's rest time or off-work time. Thus, this should be considered for the model to focus more and make more predictions in those operation peak hours.

2.1.3 (um)Point1

```
point1 = df['(um)Point1'].describe()

print('(um)Point1')
print('--------------------')
print(point1)
```

```
(um)Point1
--------------------
count    283496.000000
mean       -449.145523
std         810.459745
min       -4045.600000
25%       -1470.900000
50%           6.900000
75%          27.200000
max        2120.100000
Name: (um)Point1, dtype: float64
```

Figure 2.9: Description of (um)Point1.

```
point1_ok = df.loc[df["target"] == 1]
point1_ng = df.loc[df["target"] == 0]

# plot distplot
fig, ax = plt.subplots(figsize=(20, 6))
sns.histplot(point1_ok["(um)Point1"], color="orange", label="100% Equities", kde=True, linewidth=0)
sns.histplot(point1_ng["(um)Point1"], label="100% Equities", kde=True, linewidth=0)

ax.set(xlim=(-4200, 2200),
 xticks=[-4200, -4000, -3800, -3600, -3400, -3200, -3000, -2800, -2600, -2400, -2200, -2000, -1800, -1600, -1400, -1200, -1000,
         -800, -600, -400, -200, 0, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200])

plt.legend(["OK State", "NG State"])
plt.title('Machine Performance Point1')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

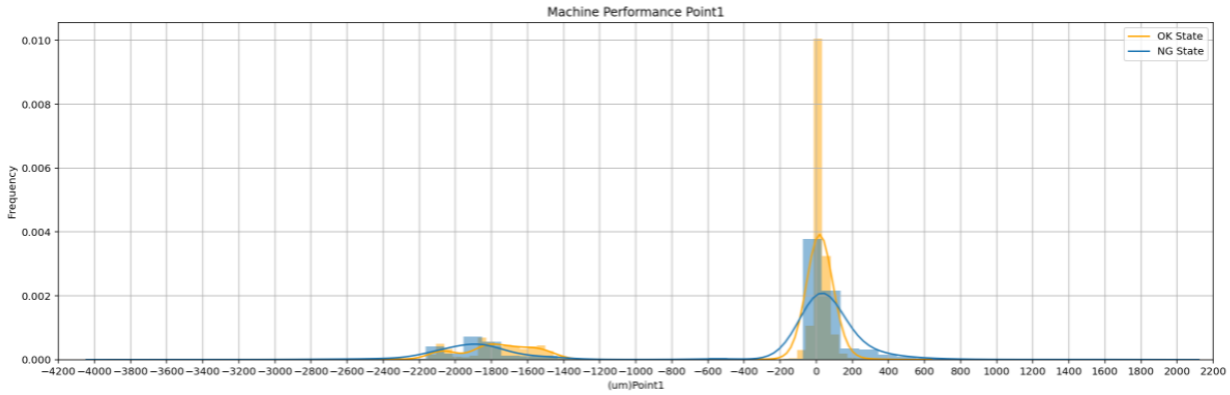



Figure 2.10: Machine Performance for (um)Point1.

The sample data is taken from the machine points which (um)Point1 ranging from -4045.6 um to 2120.1um. From the figures above, it shows that the total number of machines in poor state is smaller than the number of machines in good state. Most of the machines are in good condition ranging from -10um to 30um. While the number of machines in poor state ranged from -70um to 30um. This indicates that most of the time, (um)Point1 is maintained at an optimal measurement so that throughout the assembly of the mechanical box, the PCB is able to fit in well and hence produce a higher quality of board.

2.1.4 (um)Point2

```
point2 = df['(um)Point2'].describe()

print('(um)Point2')
print('--------------------')
print(point2)
```

```
(um)Point2
--------------------
count    244040.000000
mean       -459.510359
std         811.664952
min       -4623.900000
25%       -1492.800000
50%           1.000000
75%          15.300000
max        2650.900000
Name: (um)Point2, dtype: float64
```

Figure 2.11: Description of (um)Point2.

```
point2_ok = df.loc[df["target"] == 1]
point2_ng = df.loc[df["target"] == 0]

fig, ax = plt.subplots(figsize=(20, 6))
# sns.distplot(point2_ok["(um)Point2"], ax = ax, kde = False)
# sns.distplot(point2_ng["(um)Point2"], ax = ax, kde = False)
sns.histplot(point1_ok["(um)Point2"], color="orange", label="100% Equities", kde=True, stat="density", linewidth=0)
sns.histplot(point1_ng["(um)Point2"], label="100% Equities", kde=True, stat="density", linewidth=0)

ax.set(xlim=(-4800, 2800),
 xticks=[-4800, -4600, -4400, -4200, -4000, -3800, -3600, -3400, -3200, -3000, -2800, -2600, -2400, -2200, -2000, -1800, -1600, -1400, -1200, -1000,
         -800, -600, -400, -200, 0, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400, 2600, 2800])


plt.legend(["OK State", "NG State"])
plt.title('Machine Performance Point2')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

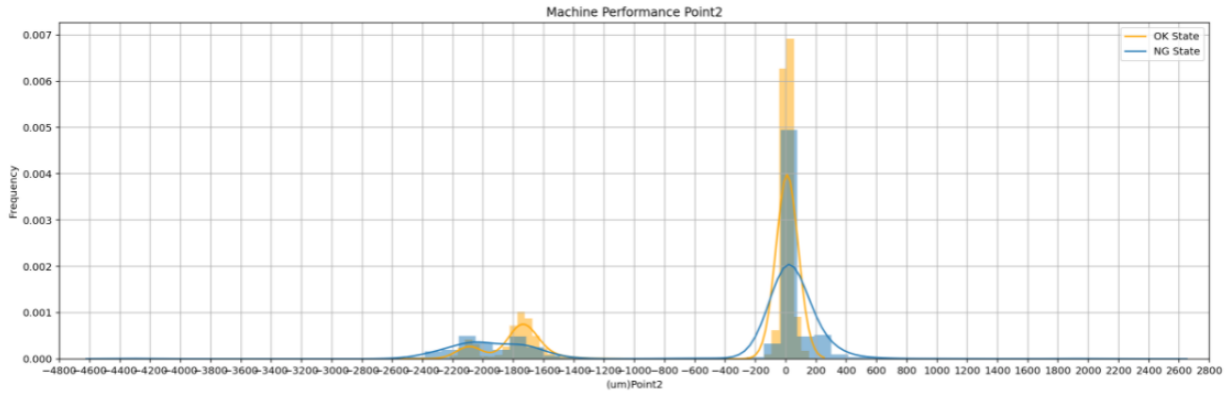



Figure 2.12: Machine Performance for (um)Point2.

The sample data is taken from the machine points which (um)Point1 ranging from -4623.9 um to 2650.9 um. Based on Figure 2.12, it shows that the total number of machines in poor state is smaller than the number of machines in good state. Most of the machines are in good condition ranging from -40um to 55um. While the number of machines in poor state ranged from -35um to 75um. However, compared to (um)Point1, (um)Point2 performance is lower. This is because based on Figure 3.3, there are more missing values in (um)Point5.

2.1.5 (um)Point3

```
point3 = df['(um)Point3'].describe()

print('(um)Point3')
print('--------------------')
print(point3)
```

```
(um)Point3
--------------------
count    215060.000000
mean       -285.948427
std         715.930290
min       -3232.900000
25%         -23.500000
50%          11.700000
75%          40.300000
max        2280.700000
Name: (um)Point3, dtype: float64
```

Figure 2.13: Description of (um)Point3.

```
point3_ok = df.loc[df["target"] == 1]
point3_ng = df.loc[df["target"] == 0]

fig, ax = plt.subplots(figsize=(20, 6))
# sns.distplot(point3_ok["(um)Point3"], ax = ax, kde = False)
# sns.distplot(point3_ng["(um)Point3"], ax = ax, kde = False)
sns.histplot(point1_ok["(um)Point3"], color="orange", label="100% Equiti
sns.histplot(point1_ng["(um)Point3"], label="100% Equities", kde=True, s

ax.set(xlim=(-3400, 2400),
 xticks=[-3400, -3200, -3000, -2800, -2600, -2400, -2200, -2000, -1800, -1600, -1400, -1200, -1000,
         -800, -600, -400, -200, 0, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200, 2400])

plt.legend(["OK State", "NG State"])
plt.title('Machine Performance Point3')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

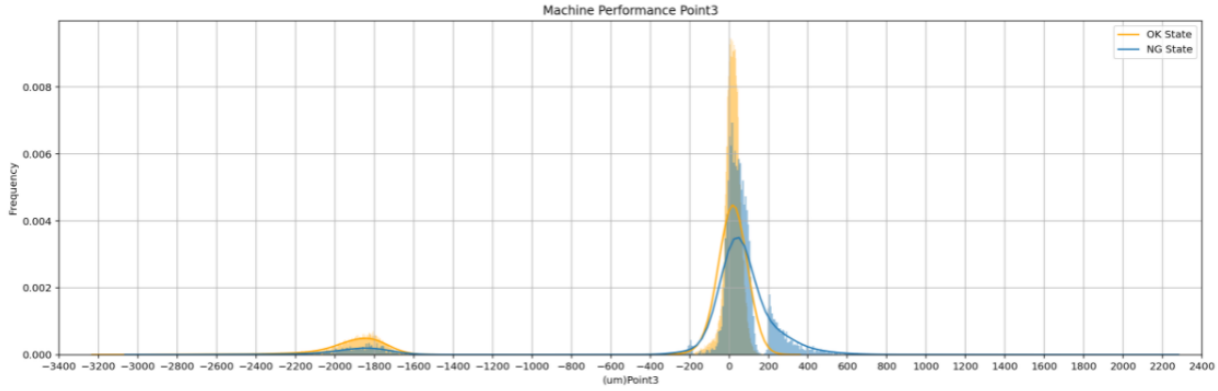



Figure 2.14: Machine Performance for (um)Point3.

The sample data is taken from the machine points which (um)Point1 ranging from -3232.9 um to 2280.7 um. Based on Figure 2.14, it shows that the total number of machines in poor state is smaller than the number of machines in good state. Most of the machines are in good condition ranging from 0um to 50um. While the number of machines in poor state ranged from 0um to 70um. However, compared to (um)Point1, and (um)Point2, (um)Point3 performance is lower as there are more missing values which is as high as 24.12% and outliers as well according to Figure 3.5. Hence, the NG_state occurs more in (um)Point3.

2.1.6 (um)Point4

```
point4 = df['(um)Point4'].describe()

print('(um)Point4')
print('--------------------')
print(point4)
```

```
(um)Point4
--------------------
count    215060.000000
mean       -271.315478
std         699.306816
min       -3835.000000
25%         -47.200000
50%          19.700000
75%          59.000000
max        2097.600000
Name: (um)Point4, dtype: float64
```

Figure 2.15: Description of (um)Point4.

```
point4_ok = df.loc[df["target"] == 1]
point4_ng = df.loc[df["target"] == 0]

fig, ax = plt.subplots(figsize=(20, 6))
# sns.distplot(point4_ok["(um)Point4"], ax = ax, kde = False)
# sns.distplot(point4_ng["(um)Point4"], ax = ax, kde = False)
sns.histplot(point1_ok["(um)Point4"], color="orange", label="100% Equities", kde=True, linewidth=0)
sns.histplot(point1_ng["(um)Point4"], label="100% Equities", kde=True, linewidth=0)

ax.set(xlim=(-4000, 2200),
 xticks=[-4000, -3800, -3600, -3400, -3200, -3000, -2800, -2600, -2400, -2200, -2000, -1800, -1600, -1400, -1200, -1000,
         -800, -600, -400, -200, 0, 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2200])
plt.legend(["OK State", "NG State"])
plt.title('Machine Performance Point4')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

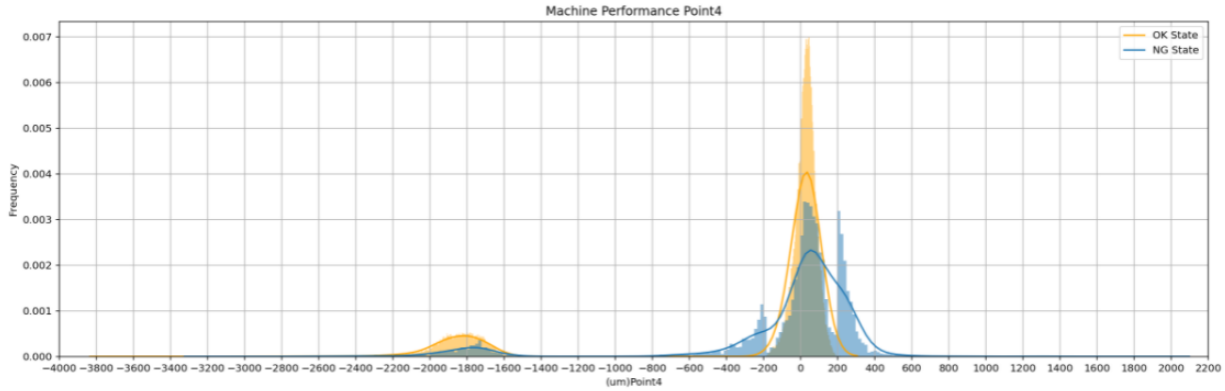



Figure 2.16: Machine Performance for (um)Point4.

The sample data is taken from the machine points which (um)Point1 ranging from -3835 um to 2097.6 um. Based on Figure 2.16, it shows that the total number of machines in poor state is

smaller than the number of machines in good state. Most of the machines are in good condition ranging from 0 um to 70 um. While the number of machines in poor state ranged from 0um to 100um. However, compared to (um)Point1, (um)Point2, and (um)Point3, (um)Point4 performance is even lower. Based on figure 3.1, (um)Point4 consists of a higher missing value which is as high as 24.12%. Although (um)Point4 and 5 are having the same percentage of missing value, however, according to Figure 3.5, (um)Point4 consists of more outliers compared to (um)Point3. Hence, the performance of (um)Point4 is lower than (um)Point3.

2.1.7 (um)Point5

```
point5 = df['(um)Point5'].describe()

print('(um)Point5')
print('--------------------')
print(point5)
```

```
(um)Point5
--------------------
count    58240.000000
mean        78.580246
std         65.066912
min       -999.400000
25%         40.800000
50%         66.100000
75%        101.300000
max       1845.200000
Name: (um)Point5, dtype: float64
```

Figure 2.17: Description of (um)Point5.

```
point5_ok = df.loc[df["target"] == 1]
point5_ng = df.loc[df["target"] == 0]

fig, ax = plt.subplots(figsize=(15, 6))
# sns.distplot(point5_ok["(um)Point5"], ax = ax, kde = False)
# sns.distplot(point5_ng["(um)Point5"], ax = ax, kde = False)
sns.histplot(point1_ok["(um)Point5"], color="orange", label="100% Equities",  linewidth=0)
sns.histplot(point1_ng["(um)Point5"], label="100% Equities", kde=True, stat=

ax.set(xlim=(-1000, 2000),
 xticks=[-1000, -800, -600, -400, -200, 0, 100, 200, 230, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000])
plt.legend(["OK State", "NG State"])
plt.title('Machine Performance Point5')
plt.ylabel('Frequency')
plt.grid()
plt.show()
```

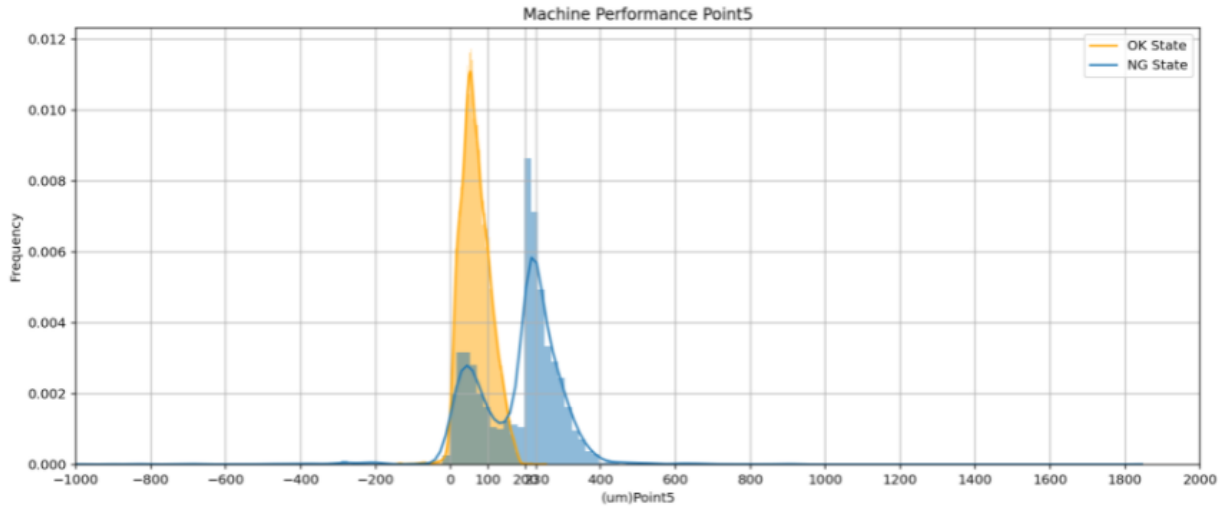



Figure 2.18: Machine Performance for (um)Point5.

The sample data is taken from the machine points which (um)Point1 ranging from -999.4 um to 1845.2 um. Based on Figure 2.18, it shows that the total number of machines in poor state is smaller than the number of machines in good state. Most of the machines are in good condition ranging from 0 um to 100 um. While the number of machines in poor state ranged from 200 um to 230 um. Compared to (um)Point1, (um)Point2, (um)Point3, and (um)Point4, the performance of (um)Point5 is the worst. This is because based on Figure 3.5, the percentage of missing values of (um)Point5 is the highest among all of the points, which is 79.46%. Hence, it will highly affect the quality of the board being produced.

```
import pandas as pd
from matplotlib import pyplot as plt
df.drop(columns = [                ], inplace = True)
corr = df.corr()
```



```
import seaborn as sns
plt.figure(figsize=(12,8))
sns.heatmap(corr, cmap="Greens",annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f58cdf4acd0>
```

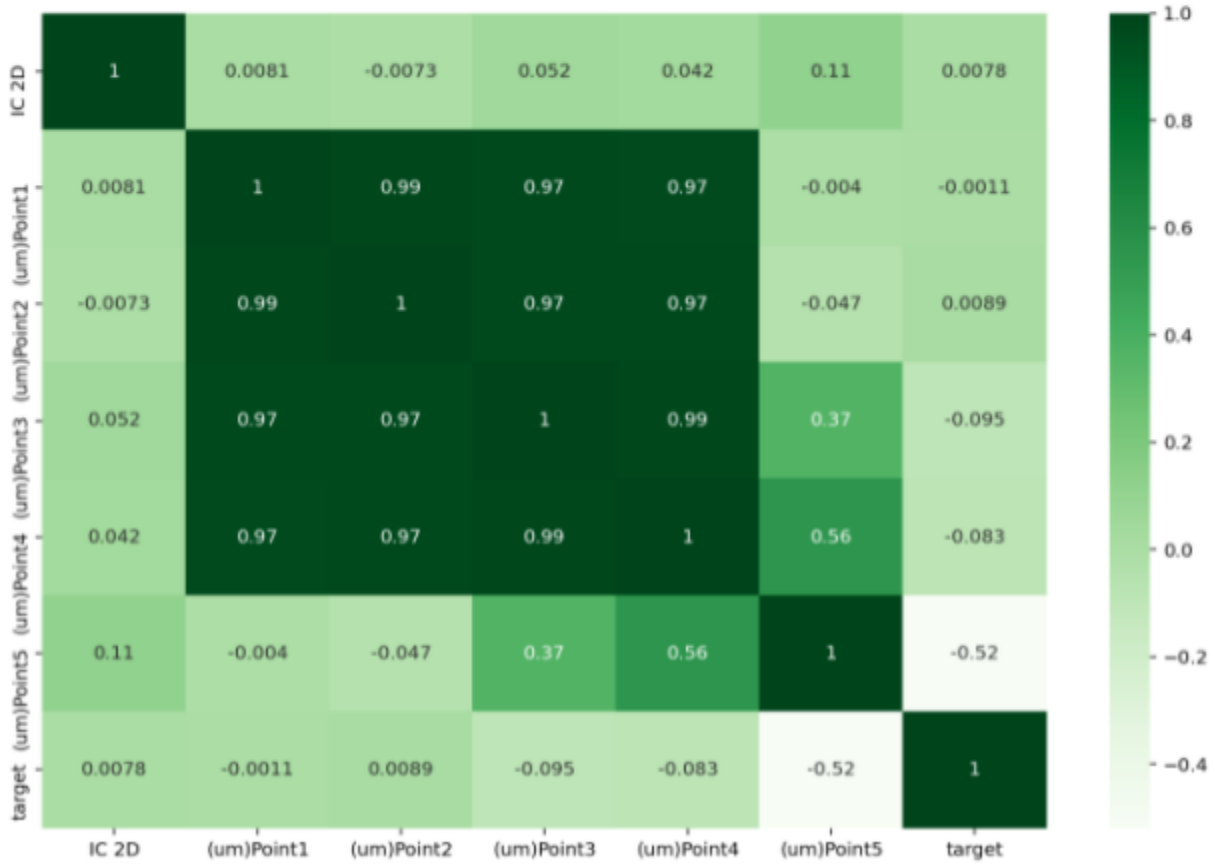


Figure 2.19: Correlation HeatMap.

Correlation heatmap as shown above is used to determine the correlation between 7 different variables based on our dataset. Based on Figure 2.19, it indicates that variables such as (um)Point1 & (um)Point2, (um)Point1 & (um)Point3, (um)Point1 & (um)Point4 are having strong positive correlation where the coefficient value is greater than 0.7. On the other hand, variables such as (um)Point5 and target (Machine result) are having strong negative correlation.

why?

There are several variables which have no correlation and whose correlation value is near to 0. For instance, IC 2D with (um)Point1, (um)Point2, (um)Point3, (um)Point4, (um)Point5. In this case (um)Point1, (um)Point2, (um)Point3, and (um)Point4 variables can be considered as features for training the models.

justify

2.2 Dependent Variable

2.2.1 Machine result

```
# Print Pie Chart
plt.figure(figsize=(5,5))
good = len(df[df['Machine result'] == 'OK'])
poor = len(df[df['Machine result'] == 'NG'])
plt.title("Performance of Machine (Hotayi)",fontsize = 15)
plt.pie(x=[good, poor], explode=[0.05, 0], labels=['OK_State ', 'NG_State'], autopct='%1.2f%%', shadow=True,startangle = 90)
plt.show()
```

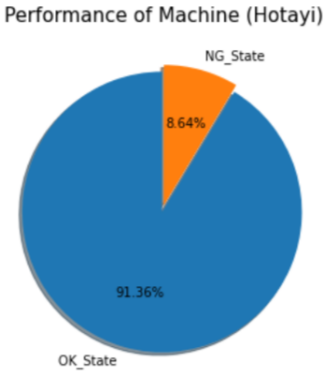


Figure 2.20: Machine results in pie chart.

```
# change value from int to string
df['target'] = df['Machine result'].replace(1, 'OK')
df['target'] = df['Machine result'].replace(0, 'NG')
```

```
target = df.target.value_counts()
print(target)
```

```
OK    259005
NG     24491
Name: target, dtype: int64
```

Figure 2.21: Conversion of machine results into integer.

```
# Bar chart
import matplotlib.pyplot as plt
import seaborn as sns


# Print Bar Chart
plt.figure(figsize=(6,12))
sns.countplot(data = df, x = 'target', palette = 'cool_r')

plt.title("Number of machine that are in OK state / NG state\n")
plt.ylabel("Number of machine")
plt.xlabel("Performance of Machine (Hotayi)")
plt.yticks(np.arange(0,283496,5000))

plt.show()
```

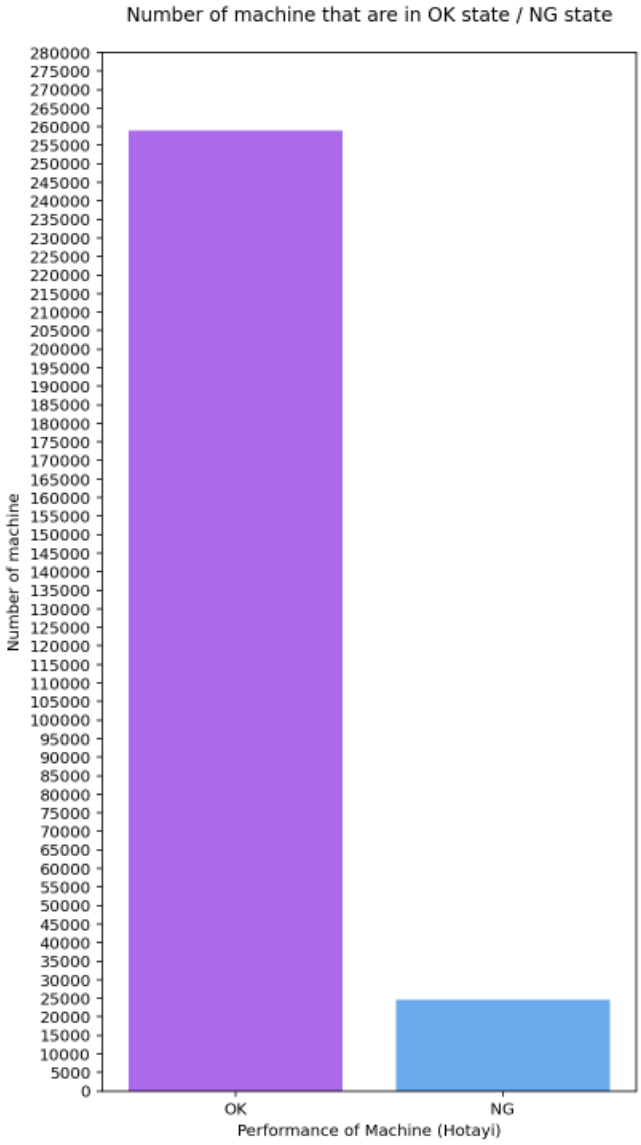


Figure 2.22: Number of machines in OK/ NG state.

Based on Figure 2.22, the machine can produce up to 260,000 'OK' results and 25, 000 'NG' results in December. In terms of percentage that shows in Figure 2.20, the machine 'OK' results percentage is up to 91.36%, and the machine 'NG' results percentage is up to 8.64%. This is because (um)Point5, which has a strong negative correlation and the worst performance, has the most missing value in this case. In contrast, (um)Point1, which has a strong positive correlation with the machine result and the best performance, does not have any missing value throughout the data. Therefore these issues cause the negative effect on the machine 'OK' results is reduced, and the positive effect is significantly increasing.

```
k=df.groupby(['target']).mean()
k.head()
```

target	Part	IC 2D	Lot	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5
NG	124313414.0	9564.144992	1.234568e+09	-446.376155	-481.260583	-66.917203	-85.237075	179.839640
OK	124313414.0	9752.513998	1.234568e+09	-449.407389	-457.120815	-307.219898	-289.386696	67.294847

Figure 2.23: Mean of variables based on machine results.

3.0 Data Preparation

The data preparation for dataset to predict the production machine failure is performed according to the data preprocessing steps listed in the CRISP-DM model where it consists of four major steps, which are data selection, data cleaning, constructing required data, and integrating the data (Smart Vision Europe, 2021). However, in this part, some touch-up on the dataset is performed through the data normalization to compare the differences between normalized and non-normalized datasets. The purpose of performing data preprocessing is to ensure the quality of the data to perform the data processing for better accuracy and consistency of the results. This allows the validity of the analytics model processed in the later process. Collected data usually comes with different formats, missing values and consists of anomalies that are required to be corrected through the data preparation processes. (Ed urns, 2020)

3.1 Data Selection

The first step of the data preparation is the data selection where the datasets are first displayed for decision making on selecting the features which are related to the machine failure or downtime. The inclusion of unrelated columns will affect the result of the data processing. We also rename some of the columns to better naming the columns. In this case, all the (um)Point1 to (um)Point5 have been renamed as Point1 to Point5.

```
[ ] df
```

	Part	IC 2D	Lot	Date	Time	Machine	Server result	Machine result	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5
0	124313414	13267	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-2020.9	-2255.9	-1809.6	-1649.6	NaN
1	124313414	13268	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1527.6	-1656.6	-1797.3	-1700.0	NaN
2	124313414	13269	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1514.8	-1717.5	-1785.4	-1740.5	NaN
3	124313414	13270	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1805.3	-1696.8	-1831.1	-1899.3	NaN
4	124313414	13271	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1507.6	-1656.0	-1873.3	-2454.7	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
283491	124313414	7969	1234567890	12/29/2020	5:17:34	PA05-1901	OK	OK	2.8	5.8	8.4	-7.5	13.9
283492	124313414	7970	1234567890	12/29/2020	5:17:34	PA05-1901	OK	OK	20.1	6.4	44.4	-9.7	28.5
283493	124313414	7971	1234567890	12/29/2020	5:17:34	PA05-1901	OK	OK	27.0	6.8	17.6	35.3	29.1
283494	124313414	7972	1234567890	12/29/2020	5:17:34	PA05-1901	OK	OK	24.3	10.4	44.4	59.5	33.8
283495	124313414	7973	1234567890	12/29/2020	5:17:34	PA05-1901	OK	OK	32.3	33.1	78.4	80.6	32.1

283496 rows × 13 columns

Figure 3.1: Description of variables.

```
[ ]  # rename the column
     df.rename(columns = {'(um)Point1':'Point1'}, inplace = True)
     df.rename(columns = {'(um)Point2':'Point2'}, inplace = True)
     df.rename(columns = {'(um)Point3':'Point3'}, inplace = True)
     df.rename(columns = {'(um)Point4':'Point4'}, inplace = True)
     df.rename(columns = {'(um)Point5':'Point5'}, inplace = True)
```

```
[ ]  df.head()
```

	Part	IC 2D	Lot	Date	Time	Machine	Server result	Machine result	Point1	Point2	Point3	Point4	Point5
0	124313414	13267	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-2020.9	-2255.9	-1809.6	-1649.6	NaN
1	124313414	13268	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1527.6	-1656.6	-1797.3	-1700.0	NaN
2	124313414	13269	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1514.8	-1717.5	-1785.4	-1740.5	NaN
3	124313414	13270	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1805.3	-1696.8	-1831.1	-1899.3	NaN
4	124313414	13271	1234567890	12/1/2020	1:04:43	PA05-1901	OK	OK	-1507.6	-1656.0	-1873.3	-2454.7	NaN

Figure 3.2: Description of variables

To provide an overview on the missing value or rows of the dataset, we visualize the dataset with the use of the missingno from the msno to show the missing value matrix. Through the missing value matrix, we are able to understand the structure of the missing value. In this case, it shows that the missing value is in the pattern of monotone. The missing value pattern is said to be monotone if the missing value on the next column is affected by the previous column (Joint Statistical Meeting, 2018). According to this dataset, if Point2 has missing value, Point3, Point4 and Point5 will have missing value and if Point3 and Point4 have missing value, Point5's value will also be missing.

```
# Overview of missing values
impor missingno as msno
%matp
msno.matrix(df)
```

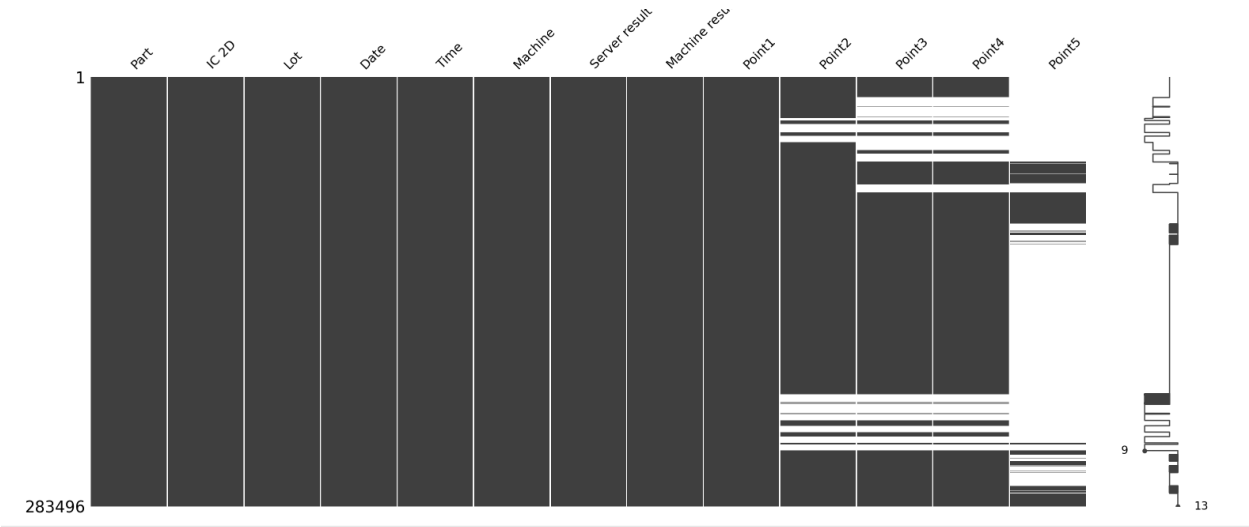


Figure 3.3: Overview of missing values.

Furthermore, we also visualize the missing value with the missing value heatmap where it shows the correlation of values between different columns. In this case, it shows there is a strong relationship between Point2, Point3 and Point4. This correlation between the missing value of Point2 and Point3 and Point4 is 70%. In other words, if there is missing value in Point2, there is 70% that there will be missing value on Point3 and 4 vice versa. The missing value of Point3 is 100% correlated with the missing value of Point4 which indicates that if Point3 is missing, Point4 will also be missing. However, Point5 shows weak correlation with Point2, Point3 and Point4 where the highest correlation is 30%.

```
msno.heatmap(df)
# from this heatmap, it shows that the missing value in point 2 correlate
# to the missing value on point 3 and 4 with 70%
```

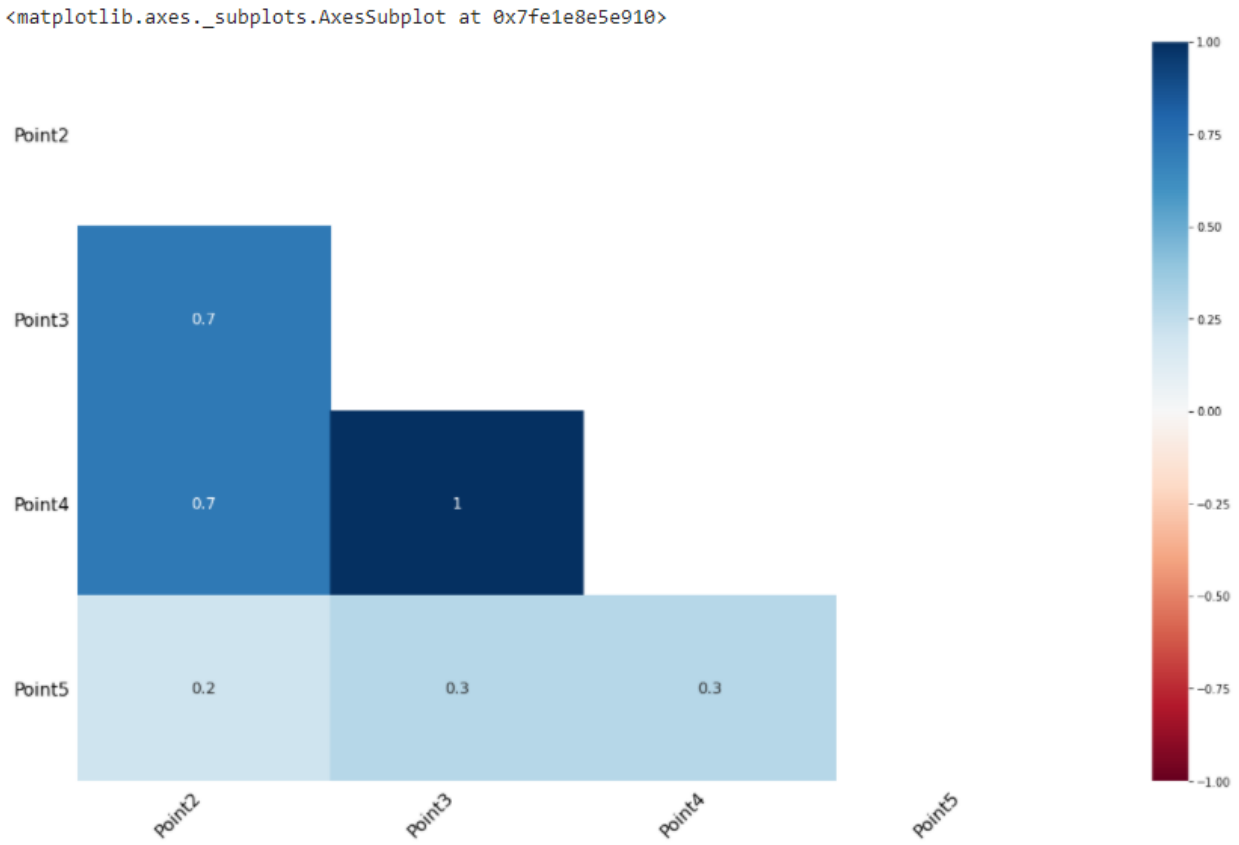


Figure 3.4: Correlation Heatmap.

Moreover, the rows of the dataset are also visualized with the percentage of missing values for each of the columns. Through the percentage of missing values, we acknowledge that the columns of Part, IC 2D, Lot, Date, Time, Machine, Server result, Machine result and Point1 do not consists of any missing values, while Point2 consists of around 14% of missing values followed by Point3 and Point4 with about 24% of missing data while Point5 have the highest missing values of around 80%.

What is the effect of performing the missing value task?

```
# set pandas display decimal to 2 decimal place
pd.options.display.float_format = "{:.2f}%".format
percent_missing = df.isnull().sum() * 100 / len(df)
percent_missing_df = pd.DataFrame(percent_missing, columns=['Percentage Missing'])
print(percent_missing_df)
# from this percentage, it shows that the column Point 5 have a very high numer of
# missing value with 79% of the column with missing value
# Furthermore, according to the correlation of missing value of Point 5 with other points
# there is a weak correlations among each of them with point 5 range from 20 to 30 percent.
# Hence, we decide to drop point 5
```


	Percentage Missing
Part	0.00%
IC 2D	0.00%
Lot	0.00%
Date	0.00%
Time	0.00%
Machine	0.00%
Server result	0.00%
Machine result	0.00%
Point1	0.00%
Point2	13.92%
Point3	24.14%
Point4	24.14%
Point5	79.46%

Figure 3.5: Percentage of missing values.

Selecting attributes (columns)

Through the visualization of data on both the columns and rows, we decided to drop the columns of Part, Lot, Machine and Server result as the values of these columns consist of duplicated or similar values. We also decided to drop the column of Point5 as it consists of a high number of missing values (around 80%) that will affect the quality of the dataset (Devi, 2020). The column of IC 2D is also dropped as it is irrelevant where we are concerned on how the operation affects the machine downtime and failure.

```
# dropping unrelated columns
# drop the duplicated or similar values
df.drop(columns=['Part','Lot','Machine','Result'], inplace = True)

# drop the column with too many missing values
# this will cause the depreciation of quality of the dataset
df.drop(columns=['Operator'], inplace = True)

# drop irrelevant column
df.drop(columns=['Date'], inplace = True)
```


Figure 3.6: Drop unwanted attributes.

3.2 Data Cleaning

After selecting the columns of the datasets, the selection of the rows or items will be performed through data cleaning. The purpose of data cleaning is to prevent the anomalies and missing values in the datasets from affecting the result or quality of analysis from the processed data (Kasture, 2020).

There are two methods that have been performed to handle the missing values on the dataset. Before the method is performed, the boxplot of the dataset is used to visualize the outliers consisting of the original datasets. From the visualized boxplot, it indicates that the datasets consist of a lot of anomalies in Point3 and Point4 where most of the values are outside the 1.5 times of the interquartile range. This is because the value outside 1.5 times of interquartile range is viewed as too far from the central value that it seems unreasonable (Stapel, 2019).

```
# Box-plot before removing outliers
import seaborn as sns
sns.boxplot(data=df, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe1e86be510>
```

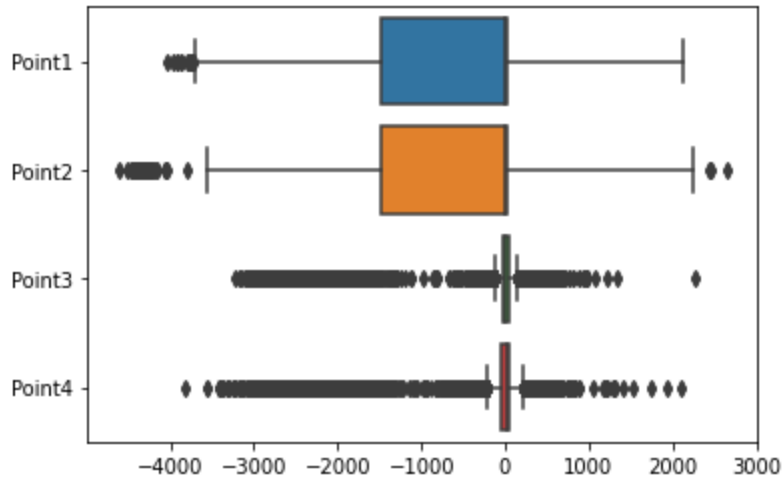


Figure 3.7: Boxplot of missing values of Point1 to 4.

The first method is the conventional method of handling missing values where each column's mean value is used to fill its corresponding missing values in each column. This method is selected as it is fast and easy to perform (Kasture, 2020). A box plot is also performed after filling the missing value with respective mean value where it shows the increasing of outliers in Point2 while decreasing of outliers in Point3 and Point4. This might be due to the increasing or decreasing interquartile range due to the fillings of mean value towards the missing data.

```
# Fill the missing value with the mean value of each column
mean = df.fi
```


Figure 3.8: Fill up missing values with mean value.

```
# plot the boxplot of dataset with handled missing value through their mean value
sns.boxplot(data=mean, orient="h")
```

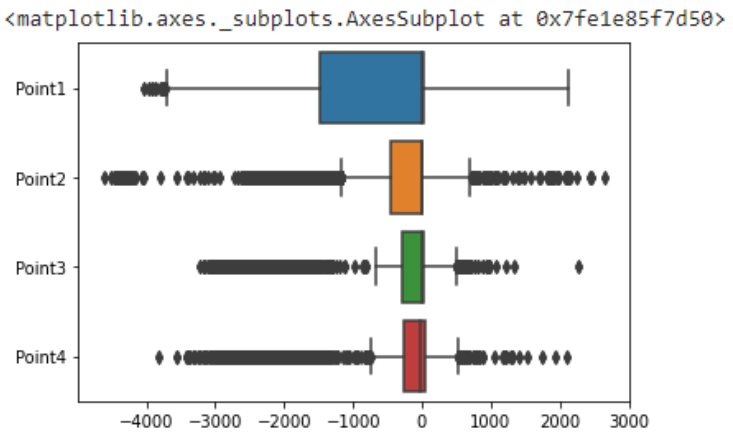


Figure 3.9: Boxplot of missing values of Point1 to 4 after substitute using mean values.

The second method is the regression imputation. The method is performed as it is noted that the missing value is not at a random order where there is correlation between the columns. The regression imputation in this case is performed using the Multiple Imputation by Chained Equation (MICE). MICE is a multiple imputation method where multiple regression will be performed through the datasets and the averages will be calculated from them for the missing values. (Iuemon96, 2020) Before performing the MICE method, the data is required to be preprocessed as the dataset is required to be loaded with the machine learning model, MICE imputer.

To perform the MICE method, the MICE package has to be loaded from the fancyimpute library to impute the missing value using the MICE imputer. After the imputation, the boxplot is also plotted where it hardly shows the boxplot result for the other value due to the Time columns. Hence, the Time column is then dropped and all the imputed datasets are again converted to data frame as the imputed values are in numpy arrays. The boxplot values show that there are no outliers in Point3 and Point4 after the handling of missing values.

```
# Install and load the R package mice
!pip inst
```


Figure 3.10: Fancyimpute installation.

```
!pip install MICE
```

Figure 3.11: MICE installation.

```
from fancyimpute import IterativeImputer as MICE
```

Figure 3.12: Import fancyimpute.

```
trans = MICE(verbose=False)
trans = KNN(verbose=False)
fitted = trans.fit(reg)
reg = fitted.transform(reg)
```



```
sns.boxplot(data=reg, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed3da5a190>
```

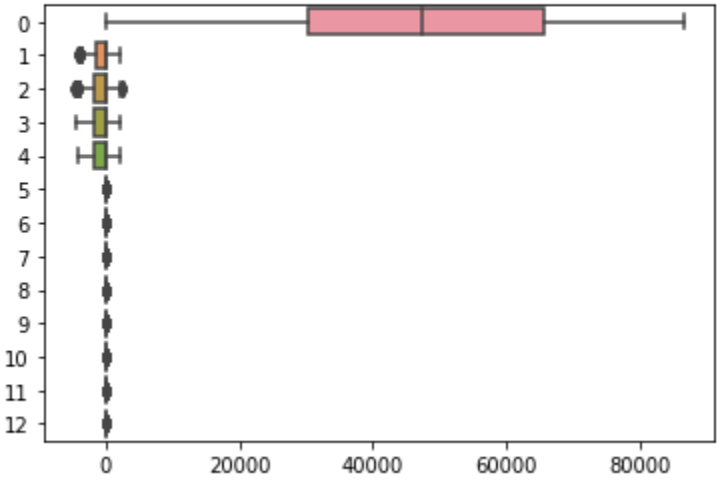


Figure 3.13: Boxplot regression data.

```
#Remove 'Time' from reg and returns the removed item
reg = pd.DataFrame(reg,
             columns=['Time', 'Point1', 'Point2', 'Point3', 'Point4', 'target', 'day_of_week_Friday'
             ,  'day_of_week_Monday', 'day_of_week_Saturday', 'day_of_week_Sunday', 'day_of_week_Thursday'
             ,  'day_of_week_Tuesday',  'day_of_week_Wednesday'])
time = reg.pop('Time')
```

Figure 3.14: Drop 'Time' column..

Why?

```
sns.boxplot(data=reg, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fed3bdf9dd0>
```

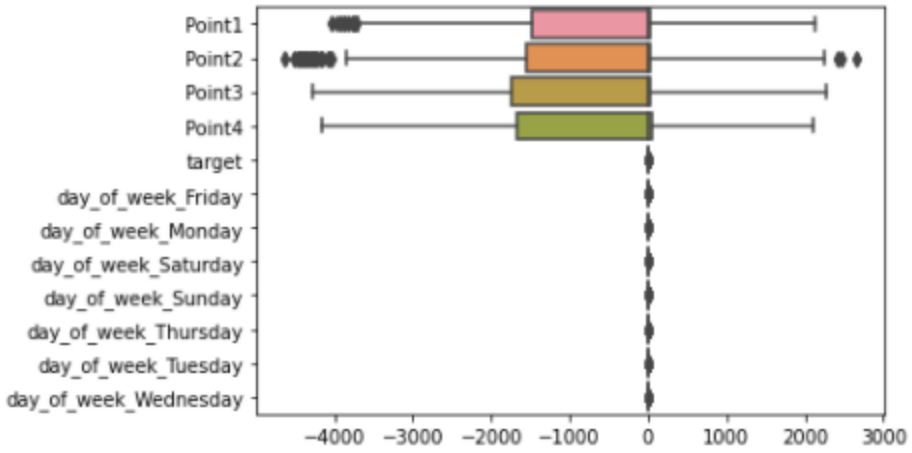


Figure 3.15: Boxplot regression data.

The outliers in both the dataset are checked for the outliers for each column with its percentage. For the dataset where it is treated using mean value for missing values, it shows that Point1 has the lowest amount of outliers with 0.01%, Point3 and Point4 have the same amount of outliers of 12.12% and followed by Point2 with the highest number of outliers with 22.12%.

On the other hand, outliers produced through the regression imputation are significantly less where Point2 recorded with only 0.02% of outliers. This is because the regression imputation will use the MICE model to predict the missing value where the missing value is treated as a dependent variable, and the other records as independent variables compared to single imputation methods where the missing values is calculated through mean or class mean where it is vary likely to produce a biased imputation that it make multiple imputation produce better results. (Shahidul Islam Khan & Abu Sayed Md Latiful Hoque, 2020)

The percentage of each of the outliers for datasets after handling with mean and regression imputation is then calculated where the mean consists of the highest around 22% of outliers in the datasets. It is decided to remove all the outliers as this dataset is a large dataset where the dropping out of the outliers will not be hurting the dataset values.

However, we do not provide any threat towards outliers for dataset which handle through multiple regression for missing values. This is because the missing value that is performed through the multiple imputations does not significantly show that it has extreme outliers as the detected outliers values are continuously near to each other.

```
from numpy import percentile
def check_outliers(col):
  q25, q75 = percentile(col, 25), percentile(col, 75)
  iqr = q75 - q25
  # calculate the outlier cutoff
  cut_off = iqr * 1.5
  lower, upper = q25 - cut_off, q75 + cut_off
  # identify outliers
  outliers = [x for x in col if x < lower or x > upper]
  print('Identified outliers: %d' % len(outliers))
  print(outliers)
  # remove outliers
  non_outliers= [x for x in col if x >= lower and x <= upper]
  print('Non-outlier observations: %d' % len(non_outliers))
  percentage = len(outliers)/len(col) * 100
  print('Percentage of outliers: %.2f' % percentage + "%")

print("Mean")
print("Point 1")
check_outliers(mean["Point1"])
print("\nPoint 2")
check_outliers(mean["Point2"])
print("\nPoint 3")
check_outliers(mean["Point3"])
print("\nPoint 4")
check_outliers(mean["Point4"])

print("\nReg")
print("Point 1")
check_outliers(reg["Point1"])
print("\nPoint 2")
check_outliers(reg["Point2"])
print("\nPoint 3")
check_outliers(reg["Point3"])
print("\nPoint 4")
check_outliers(reg["Point4"])
```


Figure 3.16: Display mean and regression for Point 1 to 4.

```
# Remove all the outliers
print("Mean remove outliers\n")
Q1 = mean.quantile(0.25)
Q3 = mean.quantile(0.75)
IQR = Q3 - Q1

mean = mean[~((mean < (Q1 - 1.5 * IQR)) |(mean > (Q3 + 1.5 * IQR))).any(axis=1)]
```

Figure 3.17: Remove all outliers.

```
Mean
Point 1
Identified outliers: 32357
[-2629.8, -2067.2, -2122.1, -2055.7, 228.8, 666.6, 270.0, 240.5, 261.5, 332.2, 93.5, 90.2, 88.8, 95.3, 120.8, 220.6, 111.7,
Non-outlier observations: 187645
Percentage of outliers: 14.71%

Point 2
Identified outliers: 48711
[278.9, 109.6, -1147.5, 236.1, 87.5, 90.4, 303.2, -696.5, -420.5, 86.5, 245.3, -120.1, 288.7, -602.3, -385.6, 82.2, 83.2, 
Non-outlier observations: 171291
Percentage of outliers: 22.14%

Point 3
Identified outliers: 44232
[-285.9484274155979, -285.9484274155979, -285.9484274155979, -285.9484274155979, -285.9484274155979, -285.9484274155979, -
Non-outlier observations: 175770
Percentage of outliers: 20.11%

Point 4
Identified outliers: 43260
[-271.3154775411527, -271.3154775411527, -271.3154775411527, -271.3154775411527, -271.3154775411527, -271.3154775411527, -
Non-outlier observations: 176742
Percentage of outliers: 19.66%


Reg
Point 1
Identified outliers: 26
[-3974.9, -3720.9, -4045.6, -3759.0, -3974.9, -3720.9, -4045.6, -3759.0, -3750.1, -3919.3, -3725.9, -3875.5, -3922.5, -3810.6, -3879.8, -
Non-outlier observations: 283470
Percentage of outliers: 0.01%

Point 2
Identified outliers: 82
[-4305.5, -4283.0, -4431.5, -4417.2, -4368.7, -4297.9, -4358.5, -4385.2, -4066.2, -4293.0, -4165.8, -4266.0, -4259.5, -4455.0, -4464.1, -
Non-outlier observations: 283414
Percentage of outliers: 0.03%

Point 3
Identified outliers: 0
[]
Non-outlier observations: 283496
Percentage of outliers: 0.00%

Point 4
Identified outliers: 0
[]
Non-outlier observations: 283496
Percentage of outliers: 0.00%
Mean remove outliers
```

Figure 3.18: Display identified outliers for both mean and regression for Point1 to 4.

After the treatment of outliers for dataset handling with mean value, it shows that there are more outliers in the dataset. This is because there is a large number of dataset accumulating around the value of -100 to 200. Hence, it causes the other values which do not clustering with each other to show as continuous outliers value in the boxplot.

```
sns.boxplot(data=mean, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fbeeba890>
```

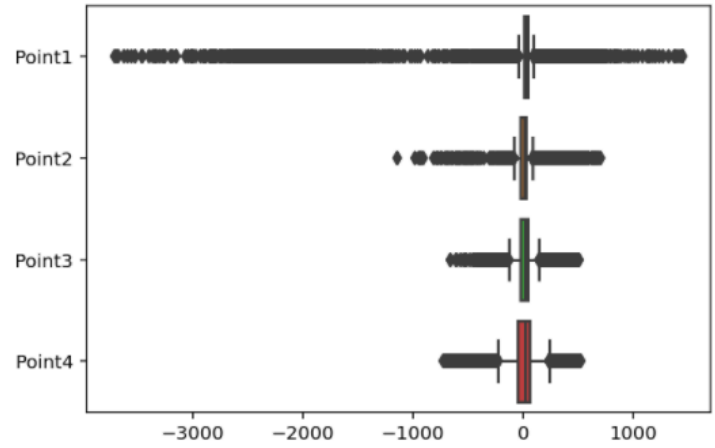


Figure 3.19: Boxplot of mean for Point 1 to 4.

```
sns.boxplot(data=reg, orient="h")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8fbee7fed0>
```

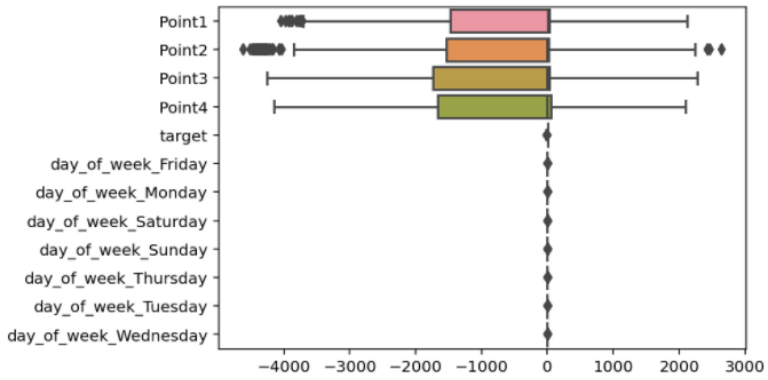


Figure 3.20: Boxplot of regression for Point 1 to 4.

3.3 Data Combination

Since the "Time" column has been dropped from the "reg" data frame, we need to join back the "Time" column by assigning the "Time" as a pandas series. After that, we perform a concat function to join both data frames together.

```
#Assign column names 'Time' to a pandas series
time = pd.DataFrame({'Time':time.values})
time.head(23123)
```

	Time
0	3,883.00
1	3,883.00
2	3,883.00
3	3,883.00
4	3,883.00
...	...
23118	31,113.00
23119	31,113.00
23120	31,113.00
23121	31,113.00
23122	31,137.00

23123 rows × 1 columns

Figure 3.21: Assign 'Time' to panda series.

```
#concate two dataframes
reg = pd.concat([time, reg], axis=1, join='inner')
```

Figure 3.22: Concat 2 dataframes.

3.4 Data Transformation

In order to ensure the data is useful and the modelling algorithm is able to understand and extract valuable information, we performed data transformation for two data frames which are "reg" and "mean". Timedeltas method is performed to convert the "Time" column from a recognized timedelta format into Timedelta type in order to transform the value of the "Time" column into seconds. Also, the data in the "Date" column is transformed to the day of week and stored in the "day_of_week" column before dropping the "Date" column. Since the "day_of_week" columns and "Machine result" columns are in categorical text currently, label encoding is applied to the "Machine result" columns and hot encoding is applied to the "Date" columns. The purpose of using label encoding for the "Machine result" column is to replace OK with 1 and NG with 0 in order to improve the performance of the modelling algorithms in predicting the y test. The purpose of using a one-hot encoder for the "day_of_week" column is to convert each

"day_of_week" value into a new column and assign a 1 or 0 value to the column. This can prevent the numeric values from being misinterpreted by modelling algorithms.

```
# Transform date to encoded value
df['Date'] = pd.to_datetime(df['Date'])
df['day_of_week'] = df['Date'].dt.day_name()
```

```
# drop date result
df.drop(columns=['Date'], inplace = True)
```

```
# Convert time to seconds

# convert duration to a timedelta
df.duration = pd.to_timedelta(df['Time'])

# get seconds for just hours, minutes, seconds
df['Time'] = df.duration.dt.seconds
```

```
# Label encode machine result
# encode the machine result into 1 or 0
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
df['target'] = labelencoder.fit_transform(df['Machine result'])

# drop the machine result column
df.drop(columns=['Machine result'], inplace = True)
```

```
# Encode day_of_week
from sklearn.preprocessing import OneHotEncoder

# creating instance of one-hot-encoder
hotencoder = OneHotEncoder(handle_unknown='ignore')

df_temp = df

# passing day_of_week column (label encoded values of day_of_week)
df = pd.DataFrame(hotencoder.fit_transform(df[['day_of_week']]).toarray())
df.columns = hotencoder.get_feature_names(['day_of_week'])

df_temp.drop(columns=['day_of_week'], inplace = True)
reg = pd.concat([df_temp, df ], axis=1)
```

```
# mean
# Transform date to encoded value
mean['Date'] = pd.to_datetime(mean['Date'])
mean['day_of_week'] = mean['Date'].dt.day_name()
```

```
# drop date result
mean.drop(columns=['Date'], inplace = True)
```

```
# Convert time to seconds

# convert duration to a timedelta
mean.duration = pd.to_timedelta(mean['Time'])

# get seconds for just hours, minutes, seconds
mean['Time'] = mean.duration.dt.seconds
```

```
# Encode day_of_week
from sklearn.preprocessing import OneHotEncoder

# creating instance of one-hot-encoder
hotencoder = OneHotEncoder(handle_unknown='ignore')

mean_temp = mean

# passing day_of_week column (label encoded values of day_of_week)
mean = pd.DataFrame(hotencoder.fit_transform(mean[['day_of_week']]).toarray())
mean.columns = hotencoder.get_feature_names(['day_of_week'])

mean_temp.drop(columns=['day_of_week'], inplace = True)
mean = pd.concat([mean_temp, mean], axis=1, join='inner')
```

```
# Label encode machine result
# encode the machine result into 1 or 0
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
mean['Target'] = labelencoder.fit_transform(mean['Machine result'])

# drop the machine result column
mean.drop(columns=['Machine result'], inplace = True)
```


Figure 3.23: Data transformation.

The train_test_split function is applied to the "reg" and "mean" datasets in order to divide the rows of the data into two different sets which are the training set and testing set. In this assignment, we have decided to fix 70% of the "reg" and "mean" datasets as the training data while 30% as the testing data. The system or machine will be fed and trained with 70% of the data so that it may learn the pattern of the data and produce better results when different data is fed. While the remaining 30% of data will be used as a testing set to see if the system can cope with different types of data and give accurate and reliable findings. Under the training data for "mean" and "reg" datasets , there will be four variables which are X_train_mean, y_train_mean, X_train_reg and y_train_reg. Similarly, the testing set for "mean" and "reg" datasets also consists of both dependent and independent variables such as X_test_mean, y_test_mean, X_test_reg and y_test_reg.

Why 70:30?

```
from sklearn.model_selection import train_test_split

#split data into training and testing sets (reg)
X_reg = reg.drop('Target', axis=1)
y_reg = reg['Target']

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg,y_reg,test_size=0.30, random_state = 42)

#split data into training and testing sets (mean)
X_mean = mean.drop('Target', axis=1)
y_mean = mean['Target']

X_train_mean, X_test_mean, y_train_mean, y_test_mean = train_test_split(X_mean,y_mean,test_size=0.30, random_state = 42)
```

Figure 3.24: Data splitting.

Table 3.1: Comparison between Accuracy of Mean and Regression Datasets

	mean	reg
K-Nearest Neighbour	0.96290	0.95974
Logistic Regression	0.93064	0.91349
Decision Tree	0.97789	0.98782
Random Forest	0.98655	0.99291

Before proceeding to 4.0 modelling, we carry out a test on the accuracy for each of the models based on the "reg" and "mean" datasets in order to determine the best dataset to be implemented by each of the models. Based on Table 3.1, it shows the summary of the accuracy of the K-Nearest Neighbour, Logistic Regression, Decision Tree, and Random Forest. According to the accuracy score for each of the models with handling missing value through regression and mean, it is found out that the result of regression and mean performed well with different models. However, we decided to select the dataset which is treated with multiple regression to handle the missing value. This is because multi-regression is much more reliable compared with the mean value that the value is acquired through averaging the value of multiple imputations. Furthermore, the boxplot significantly shows that it has lesser outliers.

Are you sure?

4.0 Modelling

There are 4 types of modelling techniques which are chosen, including K-Nearest Neighbours (KNN), Decision Tree, Logistic Regression, and Random Forest. Each of the models will first carry out GridSearchCV to obtain the optimum value for each parameter based on our dataset. Next, the data is fit into our model and performs training to display the accuracy for both train and test set.

Moreover, the confusion matrix is implemented by summarising a classification algorithm's performance. Calculating a confusion matrix increases the understanding of the classification model whether it is getting the right thing and where it is going wrong. The table and the heatmap are used to visualize the result of the confusion matrix for the decision tree model. The matrix consists of 4 values including true positive, false positive, true negative, false negative. True positive is calculated based on how many machine results "0" can be predicted correctly. False positives are calculated based on how many machine results "0" that are predicted wrongly as machine result "1". True negative is calculated based on how many machine results "1" can be predicted correctly. False positive is calculated based on how many machine results "1" that is predicted wrongly as machine result "0".A heatmap is also used to visualize the confusion matrix as well.

Additionally, by implementing classification_report using sklearn.metrics library, it can generate the precision, recall, f1-score and support for each of our models (best value = 1, worst value = 0). The calculation of precision is the ratio of tp / (tp + fp) , tp = number of true positives (*sklearn.metrics.precision_score — scikit-learn 0.24.1 documentation*, 2021). For recall the formula will be tp / (tp + fn). For F1-score, the calculation is 2 * (precision * recall) / (precision + recall). For accuracy classification score, it is used to determine the subset accuracy in multilabel classification where the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true (*sklearn.metrics.accuracy_score — scikit-learn 0.24.1 documentation*, 2021). Lastly, Area Under Graph is performed AUC to measure the quality of the model's predictions irrespective of what classification threshold is chosen.

Table 4.1: Results obtained of KNN, LR, DT, RF

	Accuracy		Accuracy	Precision	Recall	F1-score	Area Under graph (AUC)
	Test Data	Train Data					
K-Nearest Neighbours (KNN)	98.65%	100.00%	98.65%	98.72%	99.81%	99.26%	95.91%

Logistic Regression (LR)	91.45%	91.46%	91.45%	91.64%	99.75%	95.52%	47.11%
Decision Tree (DT)	97.29%	97.45%	97.29%	97.27%	99.84%	98.54%	94.91%
Random Forest (RF)	99.27%	99.99%	99.27%	99.28%	99.92%	99.60%	99.24%

4.1 K-Nearest Neighbours (KNN)

K-nearest neighbours (KNN) is one of the supervised learning algorithms for regression and classification processing. KNN uses the distance between the test data and all the training points to predict the correct class for the test data. The selection of the class is according to the K number of points which is closest to the test data. The selection is according to the KNN algorithms where the highest probability of the test data belonging to the classes of 'K' training data and class will be chosen. On the other hand, the value of mean of the 'K' selected training points will be calculated for regression. (Christopher, 2021)

In KNN, the steps of selecting which class belongs to the test data is broken down in four steps. Firstly, we will look at the test data with the available classes to classify the test data. Secondly, the distance between test data and each training point will be calculated where the commonly used methods include Euclidean Distance, Manhattan Distance and Hamming Distance. After that, each distance will be ranked. The classes will then be selected based on the class of k nearest neighbours where k indicates the count of the nearest neighbours. The nearer the test data within k numbers towards a class, the class will be selected. (Christopher, 2021)

KNN is also one of the lazy learning algorithms where KNN does not compute the distance between the test data and training data but it "memorizes" the training data. This makes the prediction of KNN expensive to perform as it is required to search throughout the dataset to make a prediction. (Raschka, 2021) As these algorithms will calculate according to the distance, normalizing the training data can improve the accuracy as shown in 3.4 Data Transformation where the normalization result provides better prediction accuracy. (Navlani, 2018)

There are a few ways to perform K-NN as there are different types of parameters provided in designing the KNN classifier. The parameters are n_neighours (k), weights, algorithm, leaf_size, p, metric, metrix_params and n_jobs.

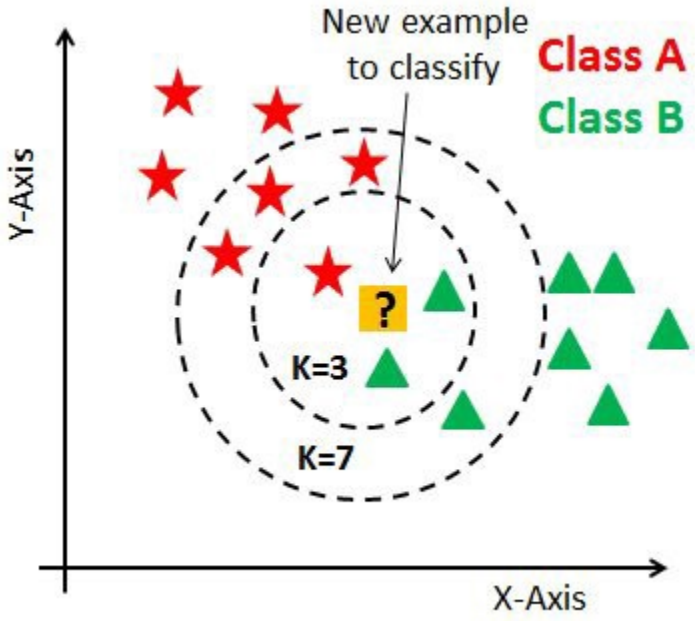


Figure 4.1: K-Nearest Neighbour Classification (Navlani, 2018)

GridSearchCV for Parameter Setting

```
# 1. Import K-Nearest Neighbours Model
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

classifier_KNN = KNeighborsClassifier()
estimator_KNN = KNeighborsClassifier(algorithm='auto')
parameters_KNN = {
    'n_neighbors': range(1,12,2),
    'p': (1,2),
    'weights': ('uniform', 'distance'),
    'metric': ('minkowski', 'chebyshev'),
}

# 2. using GridSearchCV to find the optimal KNN parameters
grid_search_KNN =GridSearchCV(
    estimator= classifier_KNN,
    param_grid= parameters_KNN,
    scoring = 'accuracy',
    n_jobs = -1,
    cv = 5
)

# KNN model learning
knn = grid_search_KNN.fit(X_train_reg, y_train_reg)

#Parameter setting that gave the best results on the hold out data.
print("Optimal parameter to product best result\n")
print(grid_search_KNN.best_params_ )
#Mean cross-validated score of the best_estimator
print('Best Score - KNN:', grid_search_KNN.best_score_ )
```

```
Optimal parameter to product best result

{'metric': 'minkowski', 'n_neighbors': 7, 'p': 2, 'weights': 'distance'}
Best Score - KNN: 0.9800803255896605
```

Figure 4.2: GridSearchCV for parameter setting.

Table 4.2: KNN Parameter Setting

Parameter	Best value
metric	minkowski

n_neigbours	7
p	2
weights	distance

In order to find the optimal parameters for the KNN classifier model, the GridSearchCV where it estimates the optimal values according to the input parameters towards KNN classifiers. In this case, only n_neighbours, p, weights and metric values are selected to find its optimal value with the range value provided. This is because computation of KNN is very expensive where including all the criteria will lead to unlimited training time. N-neighbours is the number of neighbours, weights is the way in weighting the closest neighbours during prediction, p is the power of the Minkowski metric where p = 1 equivalent to sing manhattan distance and p = 2 equivalent to using euclidean_distance, and lastly metric is the additional keyword arguments for metric function (scikit-learn developers 2019). The selected range of k neighbours are from 1 to 12 with steps number of 2 as K value is a commonly odd number to prevent ties where two classes labels achieve the same score (Brownlee 2016). The optimal value of parameters selected are minkowski metric with number of neighbours of 7, p value of 2 (euclidean_distance), and weights of distance.

Accuracy for Test and Training Set

```
from sklearn.neighbors import KNeighborsClassifier
# 4. create knn model with optimal parameter from GridSearchCV
knn = KNeighborsClassifier(metric = 'minkowski', n_neighbors = 7, p = 2, weights = 'distance')
# Fit KNN model with data
knn.fit(X_train_reg, y_train_reg)
knn_acc_test = (knn.score(X_test_reg, y_test_reg)) * 100
knn_acc_train = (knn.score(X_train_reg, y_train_reg)) * 100
print('KNN accuracy for test set: {:.2f}%'.format(knn_acc_test))
print('KNN accuracy for training set: {:.2f}%'.format(knn_acc_train))
```

```
KNN accuracy for test set: 98.65%
KNN accuracy for training set: 100.00%
```

Figure 4.3: Accuracy for test and training set.

The accuracy of testing and training set are 98.65% and 100.00% respectively. This indicates that the model is quite well-trained with the dataset which is chosen. However, there might be the potential that the model is overfitting as the training set's percentage is higher than the testing set's.

Confusion Matrix

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
knn_cm = confusion_matrix(y_test_reg,knn.predict(X_test_reg))
knn_cm
```

```
array([[ 6333,  1005],
       [  144, 77567]])
```

Figure 4.4: Confusion matrix.

```
[80] # Visualizing Confusion Matrix using Heatmap

     import matplotlib.pyplot as plt
     plt.subplots(figsize=(5,5))

     sns.heatmap(knn_cm,annot=True,linewidths=0.5)

     plt.title("K-Nearest Neighbour Confusion Matrix")
     plt.xlabel("Predicted Values")
     plt.ylabel("True Values")
     plt.show()
```

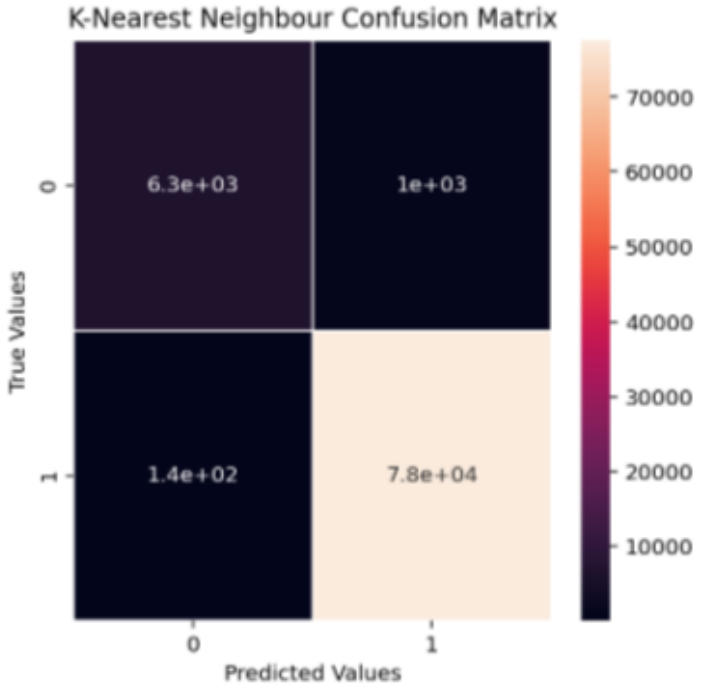


Figure 4.5: Confusion matrix in heatmap.

Based on Figure 4.4, the true positive for machine result “0” is 6333 indicating the decision tree model is able to detect 6333 machine result “0” correctly but 1005 of the predictions are wrong so it falls on the false positive. For false negative, it consists of 144 wrong predictions for machine result “1” but for true negative it consists of 77567 which means the k-nearest model predicted 77567 machine result “1”and it is true.

Confusion Report

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_score

# 5. predict labels for unknown data
y_true_knn, y_pred_knn = y_test_reg, knn.predict(X_test_reg)
print(classification_report(y_true_knn, y_pred_knn))
knn_pre = precision_score(y_true_knn, y_pred_knn) * 100
knn_rec = recall_score(y_true_knn, y_pred_knn) * 100
knn_f1 = f1_score(y_true_knn, y_pred_knn) * 100
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.86      0.92      7338
         1.0       0.99      1.00      0.99     77711

    accuracy                           0.99     85049
   macro avg       0.98      0.93      0.95     85049
weighted avg       0.99      0.99      0.99     85049
```

Figure 4.6: Confusion report.

Based on Figure 4.6, for precision, it indicates that the classifier has the ability not to label as positive a sample that is negative is 0.99. For recall, the value obtained is 0.99, indicating that the classifier has the ability to find all the positive samples is 0.99. For F1-score, the value is 0.99 while for the accuracy of the classifier is 0.99.

Area Under Graph (AUC)

```
from sklearn import metrics
y_pred_proba = knn.predict_proba(X_test_reg)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test_reg, y_pred_proba)
knn_auc = metrics.roc_auc_score(y_test_reg, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc= "+str(knn_auc),color='blue')
lw = 2
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.legend(loc=4)
plt.title("AUC of K-Nearest Neighbour \n")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.grid(True)
plt.show()
```

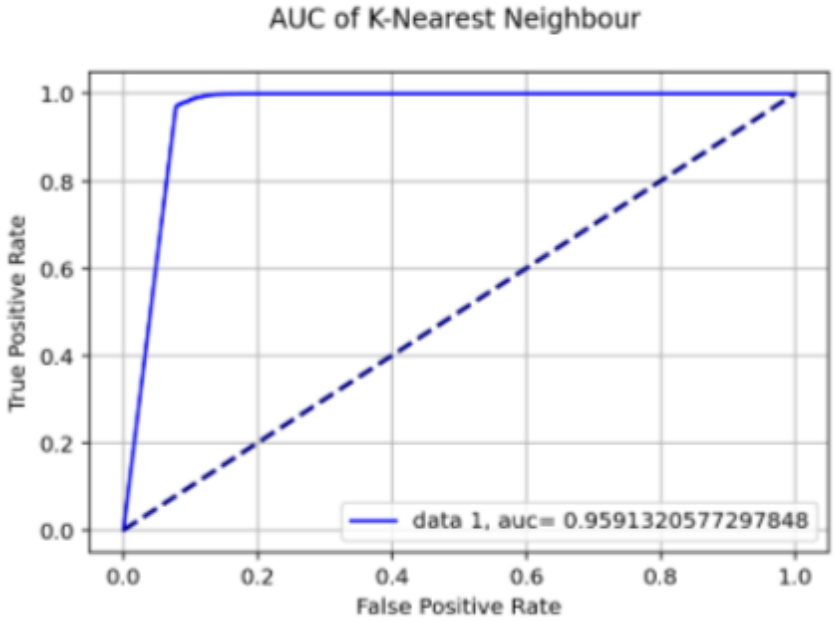



Figure 4.7: Area under graph (AUC).

AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen. (Please refer to Section 5.5 for more details) As shown in the results, above the AUC value is 0.959 indicating that the predictions are 95.9% correct.

4.2 Logistic Regression

Logistic regression is one of the supervised learning classification algorithms used to predict a categorical dependent variable from a set of independent variables. It predicts the likelihood of a target variable, so the result can be Yes or No, 0 or 1, True or False, and so on, but instead of giving exact values like 0 and 1, it delivers probabilistic values that fall between 0 and 1 (*Logistic Regression in Machine Learning - Javatpoint*, n.d.). It is a key machine learning approach because it can generate probabilities and classify new data using both continuous and discrete datasets. It can be used to classify observations based on various forms of data and quickly identify the most efficient factors for classification. Logistic Regression is very similar to Linear Regression, but Logistic Regression utilises a more sophisticated cost function instead of a linear function. This cost function is known as the "Sigmoid function" or "logistic function". The logistic regression hypothesis suggests that the cost function be limited to a value between 0 and 1. As a result, linear functions fail to describe it since it can have a value larger than 1 or less than 0, which is impossible according to the logistic regression hypothesis (Ayush Pant, 2019).

$$0 \leq h_\theta(x) \leq 1$$

Figure 4.8: Logistic regression hypothesis expectation (Ayush Pant, 2019)

The “Sigmoid function” converts any real number into another number between 0 and 1. Hence, the value of the logistic regression forms a curve like “S” form which is called a “Sigmoid function”. In machine learning, sigmoid is used to map predictions to probabilities.

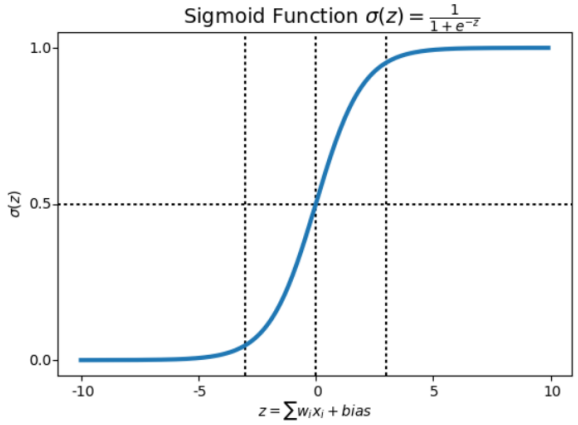


Figure 4.9: Sigmoid Function Graph (Ayush Pant, 2019).

There are some assumptions for Logistic Regression. The dependent variables in binary logistic regression must always be binary, and the desired outcome is indicated by factor level 1. There should be little or no multicollinearity in the Logistic Regression model because the independent variables must be unrelated to one another. Only the variables that are relevant should be included. Large sample size for Logistic Regression is required.

GridSearchCV for Parameter Setting

Table 4.3: Logistic Regression Parameter Setting (Refer to Appendix 9)

Parameter	Best value
Solver	lbfgs
C	1800
fit_intercept	false
tol	1e-05

1. Solver

   Solver is the algorithm to use in the optimization problem. It seeks to discover the parameter weights that reduce the cost function to the smallest possible value. There are five different solvers including newton-cg, lbfgs, liblinear, sag, and saga (Hale, 2020). In this assignment, we include 'lbfgs', 'liblinear' and 'newton-cg'.

2. C

   'C' is the inverse of regularization strength (*sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation*, 2014). In this assignment, the 'C' value included in the parameters is a list with Numpy containing five values from 1000 to 2000 with the 200 steps.

3. Fit_intercept

   'Fit_intercept' indicates whether a constant (also known as a bias or intercept) should be included in the decision function. It only consists of true and false values.

4. Tol

   'Tol' refers to tolerance for stopping criteria. When the Logistic Regression reaches the specified tolerance value, it stops training at that iteration. If a large 'tol' value is given, the Logistic Regression algorithm will stop early, which may result in poor classification; if a small value is chosen, the Logistic Regression algorithm will take longer to reach its convergence (Pothabattula, 2021). Hence, in this assignment, '1e-5' and '1e-4' are included in the hyperparameters.

Accuracy for Test and Training Set

According to Appendix 9, the accuracy of the testing and training set are 91.45% and 91.46% respectively. This shows that the model has been well-trained using the dataset that has been selected.

Confusion Matrix

Based on Appendix 10 and Appendix 11, 266 of the predictors are true negative, indicating that Logistic Regression correctly predicts the negative class, and 7072 of the predictions are false positive, indicating that Logistic Regression incorrectly predicts the negative class as positive. 77515 of the predictors are true positive, indicating that Logistic Regression correctly predicts the positive class, and only 196 values are false negative, indicating that Logistic Regression mispredicted the positive class as negative.

Confusion Report

Based on the classification_report generated using sklearn.metrics library (Refer to Appendix 12), it can be seen that the Logistic Regression model has the ability to predict the negative class correctly without predicting it as positive since the precision score has up to 0.89. The recall value is 0.91, indicating that the Logistic Regression model has the ability to find all the positive samples. For F1-score, the value obtained is 0.88. The accuracy of the Logistic Regression model is 0.91.

Area Under Graph (AUC)

As shown in the results (Refer to Appendix 13), the AUC value is 0.4711 indicating that the predictions are 47.11% correct. Hence, the Logistic Regression model is not really able to classify the positive class in the dataset correctly.

4.3 Decision Tree

The Decision Tree algorithm is one of the supervised learning algorithms. It is mainly a rule-based approach that is used to solve regression and classification problems (Deepak Patel 2021). In addition, decision Tree algorithms are widely used for predictive modeling because of their effectiveness and ease of understanding (Aunalytics 2015). The decision tree is using tree representation to split a population of data into small subsets as shown in Figure 4.10. In the decision tree, the model is trained by learning simple decision rules deduced from existing data collection before predicting an unknown outcome.

For predicting a class label for a record, we start at the root of the tree. While moving down the tree, it will break down the data set, notably the independent variables, into smaller and smaller

subsets (Aunalytics 2015). At the same time, a connected decision tree will be developed and increasing gradually. This is called divide and conquer. The final results of the models (machine results is 'OK' or 'NG') will be with the decision nodes and leaf nodes. A decision node has two or more branches. Branches are represented as the decision rules. Each rules will be directed to the conclusion of the item's target value represented by leaves. In layman terms, leaves represent the class labels, and branches represent conjunctions of features that lead to those class labels.

There are 2 types of Tree models which are the Classification Tree models and Regression Tree models. If continuous quantitative data, typically real numbers, is used to train the model, the regression tree model is more suitable to predict. Else classification tree model is more appropriate to predict the qualitative data (Aunalytics 2015). The classification tree model is used for our project because we need the model to predict the machine results, which are not countable and categorical data.

There are several steps involved in developing a decision tree which is Splitting and Pruning. Splitting is the process of partitioning the data set into subsets (Avinash Navlani 2018). It also depends on the particular variable. The decision tree is partition in a recursively manner. While pruning is an essential technique as it is used to minimize the risk of overfitting the training data by reducing the tree's size. Some unnecessary nodes will be removed from the original branch through the pruning process. However, pruning may do an inefficient job of classifying new values, although it could avoid over-fitting.

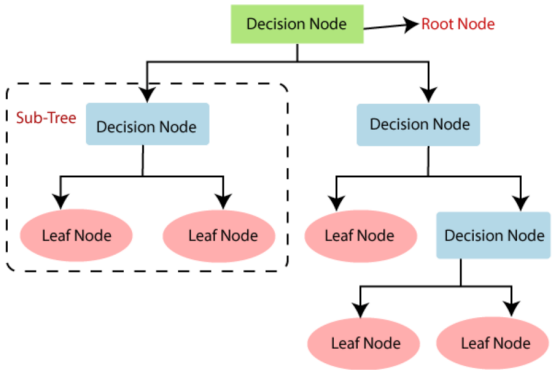


Figure 4.10 Decision Tree (Javapoint.com, 2021)

GridSearchCV for Parameter Setting

Table 4.4: Decision Tree Parameter Setting (Refer to Appendix 14)

Parameter	Best value
Criterion	entropy

max_depth	12

In order to produce a model that can give us the best result, we used Grid Search to optimize the hyper-parameters of the decision tree model. Hyper-parameters refer to those values that are used to control the model learning process and model structure. Through Grid Search and Pipeline, all combinations of hyperparameters can be passed into the model one by one. After evaluating the results of each combination of hyper-parameters, Grid Search will return a set of hyper-parameters that are the most suitable to be passed in the model to produce the best outcome (ProjectPro 2021). In this case, StandardScaler and Principal Component Analysis (PCA) are used. StandardScaler is used to eliminate outliers from the data and scales the dataset to have the mean as 0 and the standard deviation as 1. At the same time, PCA is used to diminish the features' dimension by adding new features which have most of the variance of the original data.For PCA, a parameter 'n_components' is needed. The 'n_components' refer to the number of components that need to keep after reducing the dimension. Decision Tree Classifier also needs two parameters: 'criterion' and 'max_depth' to be optimized by GridSearchCV. The maximum depth of the tree cannot be too high to prevent overfitting or too low to prevent underfitting (Avinash Navlani 2018). While the criterion is either 'gini' for the Gini index or 'entropy' for information gain (Avinash Navlani 2018). Based on Table 4.4, decision tree classifier model can produce better outcome when the criterion hyperparameter is entropy and max_depth is 12. Since entropy is the best criterion, the model will first split the data with the highest information gain and continue the process until all the children nodes in the tree are pure or until the information gain is 0.

Accuracy for Test and Training set

By using the results of the grid search, the decision tree model is trained with the dataset. The accuracy of testing and training set are 97.29% and 97.45% respectively (Refer to Appendix 15). This indicates that the model is being well-trained with the dataset which are chosen.

Confusion Matrix

Based on Appendix 16 and Appendix 17, the true positive for machine result "0" is 5157 indicating the decision tree model is able to detect 5157 machine result "0" correctly but 2181 of the predictions are wrong so it falls on the false positive. For false negative, it consists of 122 wrong predictions for machine result "1" but for true negative it consists of 77589 which means the decision tree model predicted 77589 machine result "1"and it is true.

Confusion Report

Based on Appendix 18, it can be seen that the decision tree model has up to 0.98 precision score, 0.70 recall score and 0.82 f1 score for machine result "0". For machine result "1", the precision score is 0.97, the recall score is 1.00 and the f1 support score is 0.99. The results indicates that

the decision tree classifier model reached the data mining success criteria. Thus, it is able to label the data correctly and has the ability to find all positive samples.

Area Under Graph (AUC)

As shown in the Appendix 19, the AUC value is 0.9487 indicating that the predictions are 94.87% correct. Thus, the random forest classifier is able to classify the positive class in the dataset correctly.

4.4 Random Forest

Random Forest is one of the supervised learning algorithms to perform classification and regression which involves decision trees. The training is carried out by creating a large number of decision trees in the first place. The class that is the mode of the classes or the mean forecast of the individual trees is also output by decision trees. Random decision forests can also help decision trees overcome their tendency to overfit their training sets. By using the "bagging" method, it allows the overall results of a collection of learning models to improve. This is because Random Forests are a composite of many decision trees. Individual decision trees are created for each attribute using an attribute selection indicator such as information gain, gain ratio, or Gini index. Each of the decision trees in our case study will produce one of two outcomes: yes or no. Following that, each tree casts a vote, and the most popular class is selected as the final outcome (Navlani 2018). Randomness will be added to the Random Forest model as the trees develop. Instead of dividing a node while looking for the most important feature, Random Forest will look for the best feature from a random selection of features, resulting in a wider range of outcomes and a more accurate model. As a result, the process for splitting a node in Random Forest only considers a random subset of the features. The Random Tree can be made more random by utilising random thresholds for each feature rather than the best available thresholds.

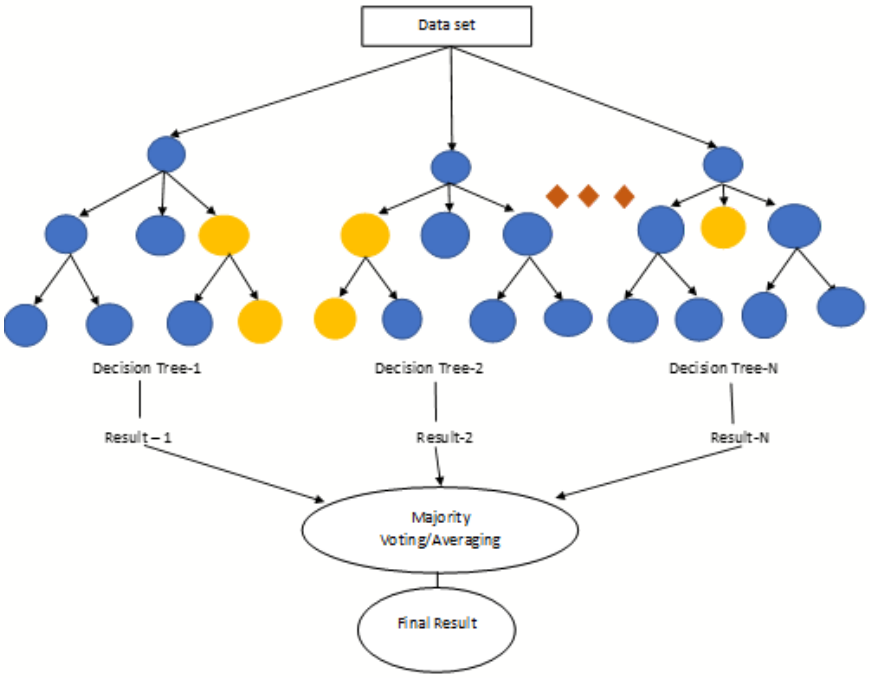


Figure 4.11: Random Forest (Dhanasekar, 2019)

GridSearchCV for Parameter Setting

Table 4.5: Random Forest Parameter Setting (Refer to Appendix 20)

Parameter	Best value
Criterion	gini
max_depth	None
max_features	4
min_samples_leaf	1
min_samples_split	2
n_estimator	40

1. Criterion

Criterion is used to measure the quality of the split by looking either at '*gini*' or '*entropy*'. Based on Appendix 20, Gini is chosen over Entropy. Gini differs from entropy in that it calculates the likelihood of a randomly selected element being wrongly labelled according to gini impurity. Entropy, on the other hand, measures the disorder of a grouping depending on the amount of information gained. (2020, Spiro)

2. Max_depth

The max depth function is used to find the maximum number of levels in a tree or the longest path between the root and leaf nodes. Based on Appendix 20, Max-depth is set to 'none' indicating that the tree will keep increasing until all leaf nodes are pure where all data on the leaf is from the same class. A larger number of splits allows each tree to better explain the variation in the data, but too many splits can cause the data to be overfit. (2020, Spiro)

3. Max_features

Max_features is used to represent the maximum number of features to consider for splitting a node. For this hyperparameter, there are a few typical choices including *int*, *auto*, *sqrt* and *log2*. Based on Appendix 20, value '4' is chosen. Model performance can improve with a larger number of features to choose from when determining the appropriate split, but this can also make

the trees less diversified, resulting in overfitting. (2020, Spiro)

4. Min_samples_leaf

Min_samples_leaf is used to represent the minimum number of samples required for a leaf node. Based on Appendix 20, the value is set to 1, then each leaf must have at least 1 sample that it classifies. This helps to keep the tree from expanding too large, which can lead to overfitting. However, as with min samples split, a large value can result in underfitting due to the tree's inability to split enough times to reach pure nodes. (2020, Spiro)

5. Min_samples_split

The min_samples_split hyperparameter is used to represent the minimum number of samples required to split an internal node. Based on Appendix 20, the value is set to 2, indicating that at least 2 samples are needed to split each internal leaf node. We can reduce the number of splits in the decision tree by increasing this hyperparameter's value, which helps to avoid overfitting. A big value, on the other hand, can cause underfitting because the tree may not be able to split enough times to reach pure nodes. (2020, Spiro)

6. n_estimators

The n_estimators hyperparameter specifies the number of trees in the forest. Based on Appendix 20, the value is set to 20 indicating that there will be 5 trees in your Forest. More trees can help improve accuracy because predictions are made from a bigger number of "votes" from a diverse collection of trees, but the more trees you have, the easier it is to run into huge computational expenses. (2020, Spiro)

Optimum N Estimator

In order to find the model that is able to make better predictions, one of the features, n_estimators is one of the Random Forest parameters that we must use to determine the exact value in order to improve the model's accuracy. This is because n_estimator represents the number of trees to build before taking the maximum voting or averages of predictions. By default, the n estimator value is set to 100. Although the higher number of trees, the better the performance. However, it will take up more time to complete the training due to the high number of trees which makes the processing and calculation. Based on Appendix 21, from 0 to 20 n_estimator, the accuracy increases rapidly from 0.9750 to 0.9937. From 20 to 40 n_estimator, the accuracy increases slightly by 0.0025. For 60 to 100 n_estimator, there is no increase in the accuracy. Hence, value 20 is chosen for the n_estimator to produce stronger and stable predictions within a shorter period.

Accuracy for Test and Training Set

Based on Appendix 22, the accuracy of testing and training set are 99.27% and 99.99% respectively. This indicates that the model is being well-trained with the dataset which are chosen.

Confusion Matrix

Based on Appendix 23, the true positive for machine result "0" is 6778 indicating the decision tree model is able to detect 6778 machine result "0" correctly but 560 of the predictions are wrong so it falls on the false positive. For false negative, it consists of 60 wrong predictions for machine result "1" but for true negative it consists of 77651 which means the decision tree model predicted 77651 machine result "1"and it is true.

Confusion Report

Based on Appendix 25, for precision, it indicates that the classifier has the ability not to label as positive a sample that is negative is 0.99. For recall, the value obtained is 0.99, indicating that the classifier has the ability to find all the positive samples is 0.99. For F1-score, the value is 0.99 while for the accuracy of the classifier is 0.99.

Area Under Graph (AUC)

Based on Appendix 26, above the AUC value is 0.99244 indicating that the predictions are 99.24% correct. Thus, the random forest classifier is able to classify the positive class in the dataset correctly.

5.0 Evaluation

Based on the classification_report generated using sklearn.metrics library, it consists of the precision, recall, f1-score and support. (best value = 1, worst value = 0)

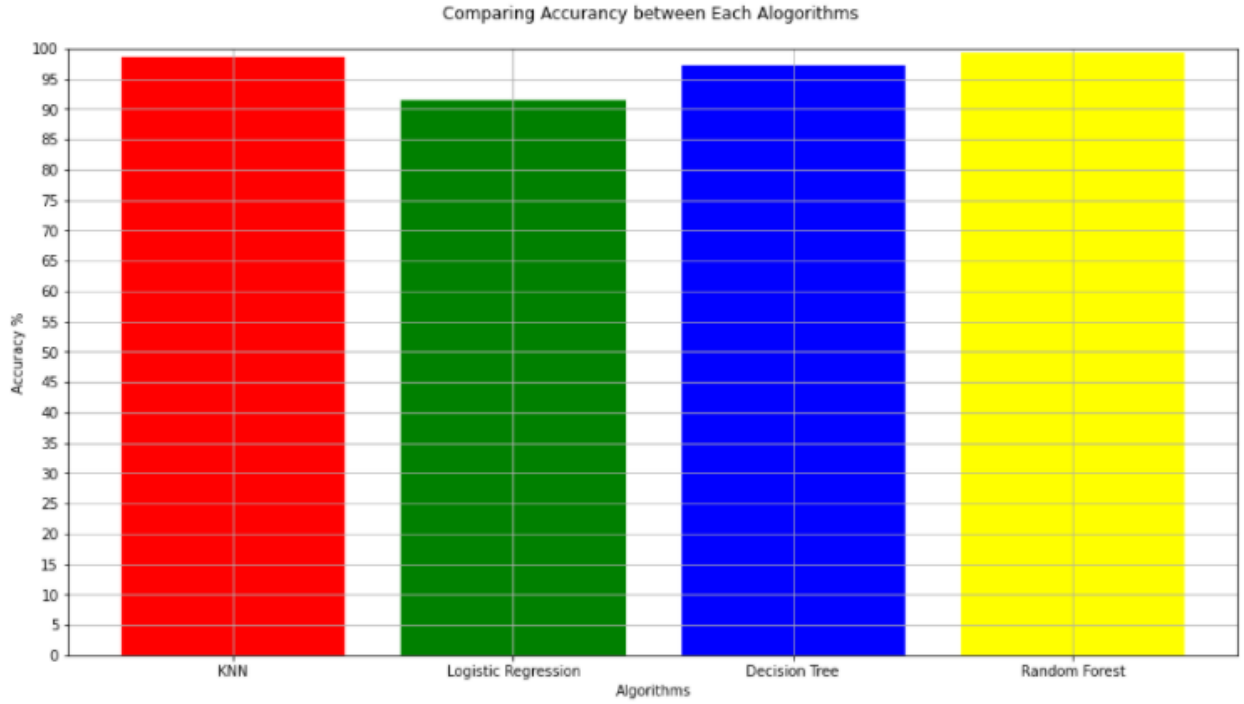
'Macro': It is used to calculate metrics for each dataset, and compute the unweighted mean without considering label imbalance (*sklearn.metrics.accuracy_score — scikit-learn 0.24.1 documentation*, 2021).

'Weighted': It is used to calculate metrics for each dataset, and compute the average weighted based on the number of true instances for each dataset (*sklearn.metrics.accuracy_score — scikit-learn 0.24.1 documentation*, 2021).

5.1 Accuracy

It is used to determine the subset accuracy in multilabel classification where the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y_true (*sklearn.metrics.accuracy_score — scikit-learn 0.24.1 documentation*, 2021).

```
import math
plt.figure(figsize=(15,8))
plt.grid(True)
plt.bar(x=name, height=acc_test,color=['red', 'green', 'blue', 'yellow'])

plt.xticks()
plt.ylim((0,100))

plt.yticks(np.arange(0,105,5))
plt.title("Comparing Accurancy between Each Alogorithms \n")
plt.ylabel("Accuracy %")
plt.xlabel("Algorithms")
plt.show()
print ("K-Nearest Neighbour: {0:.2f}%".format(acc_test[0]))
print ("Logistic Regression: {0:.2f}%
print ("Decision Tree: {0:.2f}%".f
print ("Random Forest: {0:.2f}%".f
```


```
K-Nearest Neighbour: 98.65%
Logistic Regression: 91.45%
Decision Tree: 97.29%
Random Forest: 99.27%
```

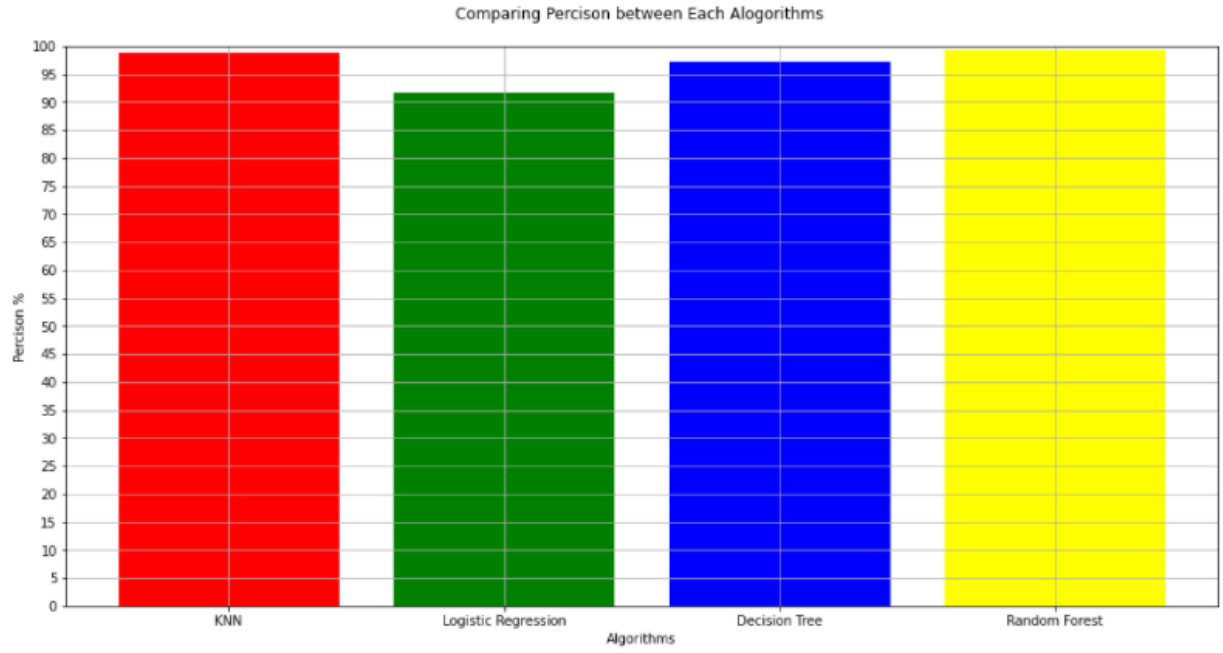
Figure 5.1: Comparing accuracy between algorithms.

Based on Figure 5.1, we can see that accuracy of all models is above 90% which fulfills the data mining criteria that we set before. This also means that all the models are well-trained with the selected dataset. Random Forest is the model that has the highest accuracy score which is up to 99.27%. On the other hand, Logistic Regression model is the one who has the lowest accuracy score which is 91.45%. The higher the accuracy score, the closer the machine results can be detected corresponding to its "true" value.

5.2 Precision

Ratio of tp / (tp + fp) , tp = number of true positives, fp = number of false positives. Precision shows the ability of the classifier not to label as positive a sample that is negative (*sklearn.metrics.precision_score — scikit-learn 0.24.1 documentation*, 2021).

```
plt.figure(figsize=(16,8))
plt.grid(True)
plt.bar(x=name, height=pre_all, color=['red', 'green', 'blue', 'yellow'])

plt.xticks()
plt.ylim((0,100))

plt.yticks(np.arange(0,105,5))
plt.title("Comparing Percison between Each Alogorithms \n")
plt.ylabel("Percison %")
plt.xlabel("Algorithms")
plt.show()
print ("K-Nearest Neighbour: {0:.2f}%".format(pre_all[0]))
print ("Logistic Regression: {0:.2f}%".format(pre_all[1]))
print ("Decision Tree: {0:.2f}%".format(pre_all[2]))
print ("Random Forest: {0:.2f}%".format(pre_all[3]))
```


```
K-Nearest Neighbour: 98.72%
Logistic Regression: 91.64%
Decision Tree: 97.27%
Random Forest: 99.28%
```

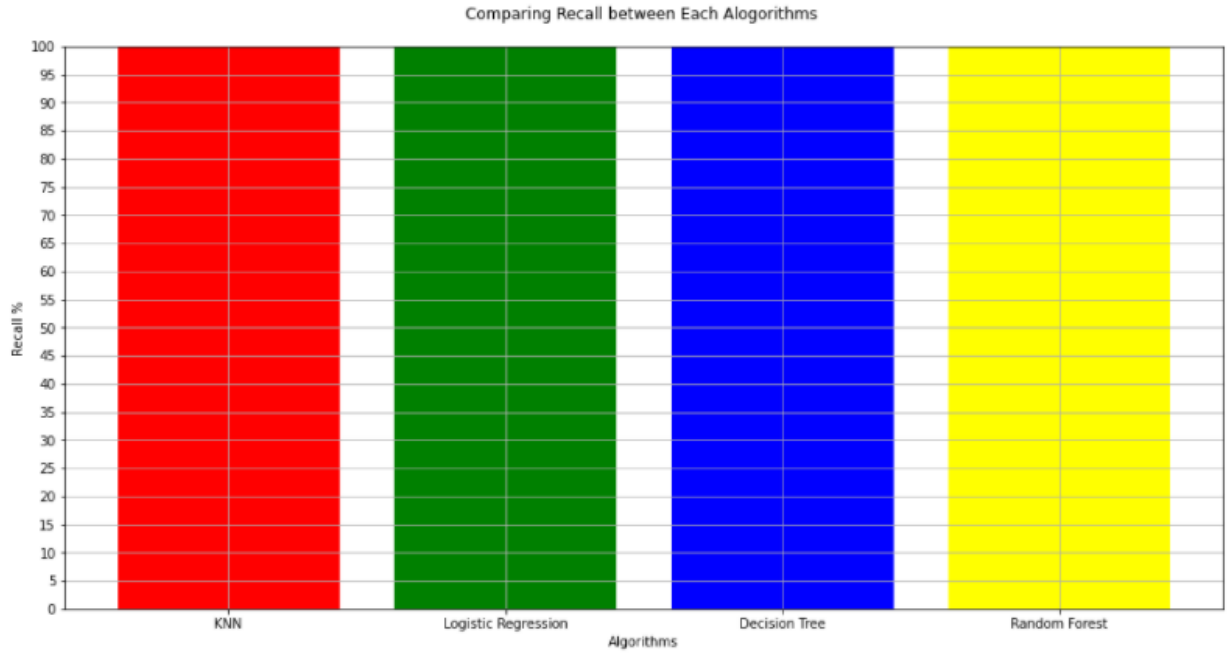
Figure 5.2: Comparing precision between algorithms.

Based on Figure 5.2, we can see that precision of all models is above 90% which fulfills the data mining criteria that we set before. Random Forest is the model that has the highest precision score which is up to 99.28%. KNN model is the second highest precision scorer which only slightly lower than the random forest model which scores up to 98.72%. On the other hand, Logistic Regression model is the one who has the lowest precision score which is 91.64%. The higher the precision score, the closer the detected results are going to be.

5.3 Recall

Ratio tp / (tp + fn) , tp = number of true positives, fn = number of false negatives. Recall shows the ability of the classifier to find all the positive samples (Ajitesh Kumar, 2020).

```
plt.figure(figsize=(16,8))
plt.grid(True)
plt.bar(x=name, height=rec_all, color=['red', 'green', 'blue', 'yellow'])

plt.xticks()
plt.ylim((0,100))

plt.yticks(np.arange(0,105,5))
plt.title("Comparing Recall between Each Alogorithms \n")
plt.ylabel("Recall %")
plt.xlabel("Algorithms")
plt.show()

print ("K-Nearest Neighbour: {0:.2f}%".format(rec_all[0]))
print ("Logistic Regression: {0:.2f}%".format(rec_all[1]))
print ("Decision Tree: {0:.2f}%".format(rec_all[2]))
print ("Random Forest: {0:.2f}%".format(rec_all[3]))
```


```
K-Nearest Neighbour: 99.81%
Logistic Regression: 99.75%
Decision Tree: 99.84%
Random Forest: 99.92%
```

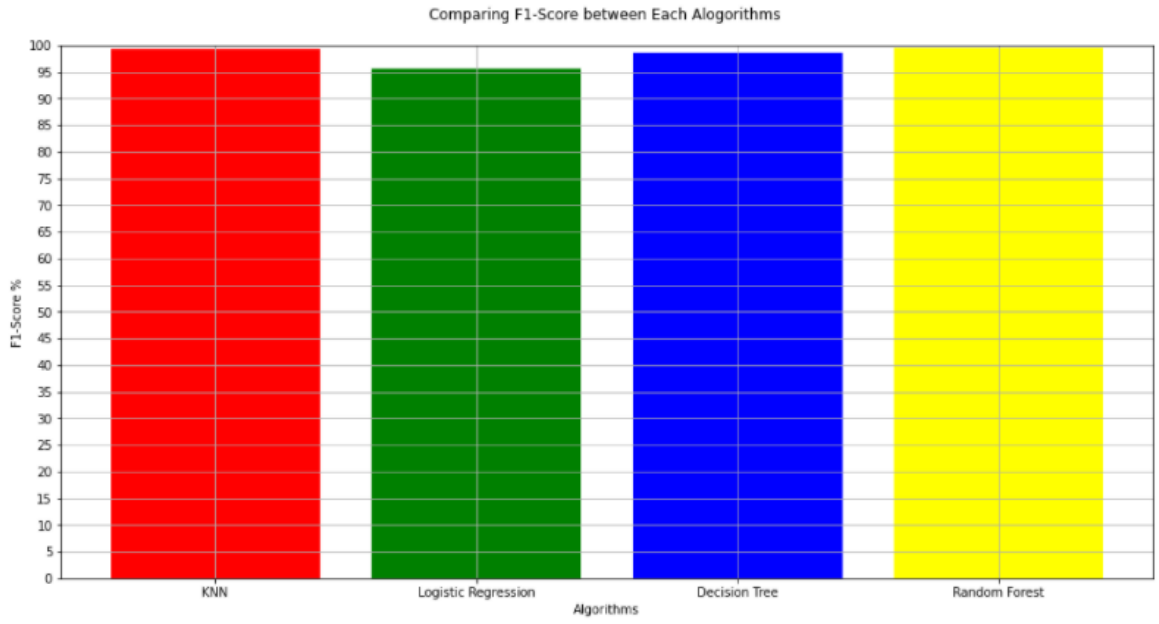
Figure 5.3: Comparing recall between algorithms.

Based on Figure 5.3, we can see that the recall of all models used is above 99% which fulfills the data mining criteria that we set before. With a recall of 99.92%, the Random Forest is the model with the highest recall of all the models. On the other hand, Logistic Regression is the model that has the lowest recall with 99.75%. The higher the recall, the more the positive class is being captured.

5.4 F1-score

The F1 score can be interpreted as a weighted average of the precision and recall. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: F1 = 2 * (precision * recall) / (precision + recall).

```
plt.figure(figsize=(16,8))
plt.grid(True)
plt.bar(x=name, height=f1_all,color=['red', 'green', 'blue', 'yellow'])

plt.xticks()
plt.ylim((0,100))

plt.yticks(np.arange(0,105,5))
plt.title("Comparing F1-Score between Each Alogorithms \n")
plt.ylabel("F1-Score %")
plt.xlabel("Algorithms")
plt.show()

print ("K-Nearest Neighbour: {0:.2f}%".format(f1_all[0]))
print ("Logistic Regression: {0:.2f}%".format(f1_all[1]))
print ("Decision Tree: {0:.2f}%".format(f1_all[2]))
print ("Random Forest: {0:.2f}%".format(f1_all[3]))
```


```
K-Nearest Neighbour: 99.26%
Logistic Regression: 95.52%
Decision Tree: 98.54%
Random Forest: 99.60%
```

Figure 5.4: Comparing F1-score between algorithms.

Based on Figure 5.4, we can see that the recall of all the models used are more than 95%. Random Forest has the highest recall which is 99.60% followed by KNN which is 99.26%. On the other hand, the model that has the lowest recall is Logistic Regression with 95.52%. The higher the recall, the more the positive class is being captured.

5.5 Area Under Graph (AUC)

The total area under the Receiver Operating Characteristic (ROC) curve is referred to as AUC. AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen. AUC uses a two-dimensional area under the full ROC curve (assuming integration) from (0,0) to (1,0) to measure the trade-off between the true positive rate and the false positive rate. (Classification: ROC Curve and AUC | Machine Learning Crash Course). For instance, the higher the AUC, the better the model is at distinguishing between good or bad machine results.

Table 5.1: Area under graph performance range.

Grade	Range
Excellent	0.90 - 1.00
Good	0.80 - 0.89
Fair	0.70 - 0.79
Poor	0.60 - 0.69
Fail	0.50 - 0.59

```
auc_all = [(al)for al in [knn_auc, lr_auc, dt_auc, rf_auc]]

plt.figure(figsize=(16,8))
plt.grid(True)
plt.bar(x=name, height=auc_all,color=['red', 'green', 'blue', 'yellow'])

plt.xticks()
plt.ylim((0,1.05))

plt.yticks(np.arange(0,1.05,0.05))
plt.title("Comparing AUC between Each Alogorithms \n")
plt.ylabel("AUC")
plt.xlabel("Algorithms")
plt.show()

print ("K-Nearest Neighbour: {0:.2f}%".format(auc_all[0]*100))
print ("Logistic Regression: {0:.2f}%".format(auc_all[1]*100))
print ("Decision Tree: {0:.2f}%".format(auc_all[2]*100))
print ("Random Forest: {0:.2f}%".format(auc_all[3]*100))
```

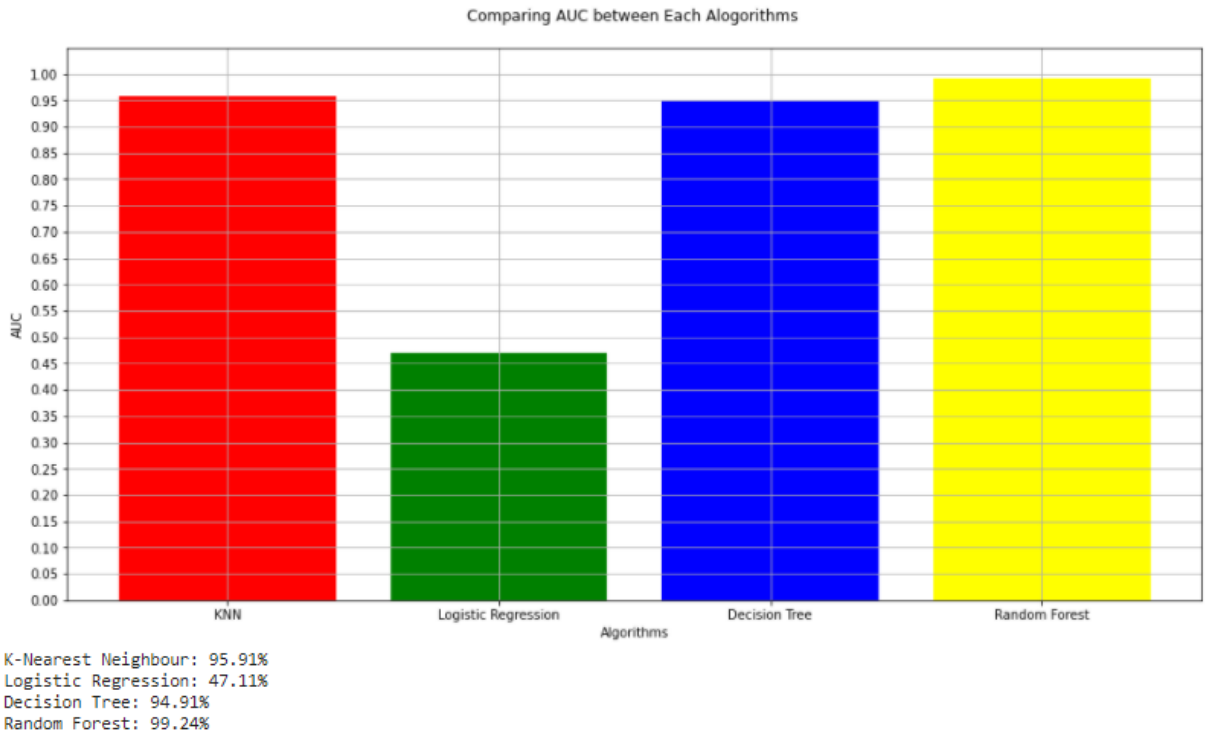


Figure 5.5: Comparing Area under graph (AUC) between algorithms.

Based on Figure 5.5, we can see that the AUC of the models including KNN, Decision Tree and Random Forest are 90% and above except for Logistic Regression. This indicates that the models have a good quality of prediction while taking into account all possible classification thresholds except Logistic Regression. Random Forest has the highest precision of 99.24%, while Logistic Regression has the lowest precision of 47.11%.

6.0 Deployment

6.1 Selection of Best Model

By making comparisons with all the models we tested, Random Forest (RF) algorithm has the best results among all the algorithms in terms of Accuracy, Precision, Recall and F1-score.

In the Evaluation part, the K-Nearest Neighbours (KNN), Logistic Regression (LR), Decision Tree (DT) and Random Forest (RF) Algorithm had these results:

Table 6.1: Results obtained of KNN, LR, DT, RF

Evaluation	K-Nearest Neighbours (KNN)	Logistic Regression (LR)	Decision Tree (DT)	Random Forest (RF)
Accuracy	98.65%	91.45%	97.29%	99.27%
Precision	98.72%	91.64%	97.27%	99.28%
Recall	99.81%	99.75%	99.84%	99.92%
F1-Score	99.26%	99.52%	98.54%	99.60%

Based on Table 6.1, it shows the summary of the accuracy, precision, recall, and F1-score of the K-Nearest Neighbour, Logistic Regression, Decision Tree, and Random Forest. From the result, as shown in the table, the Random Forest has the highest accuracy (99.27%) followed by K-Nearest Neighbour (98.65%), Decision Tree (97.29%), and Logistic Regression (91.45%). Random Forest usually provides the highest accuracy as it is more powerful and can work correctly for a large range of data items. The more the trees the higher the accuracy of the random forest (Donges, 2019). However, instead, the result of the accuracy, precision, recall and F1- score tend to be more important.

Precision and Recall are important in predicting whether the machine result is ‘OK’ or ‘NG’. High precision provides the means of low false-positive rate where the false positive means that a machine with result ‘NG’ has been identified as ‘OK’. The machine might lose the chance of providing an ‘OK’ result if the precision is not high for the selected model. In this case, the random forest is provided with the highest percentage value of precision which is 99.28% followed by K-Nearest Neighbour (98.72%), Decision Tree (97.27%), and Logistic Regression (91.64%).

On the other hand, recall calculates how many of the machines which have the result of ‘OK’ are labeled with ‘OK’ by the model. The recall is important as it prevents the machine to be operated under the context which will lead to the ‘NG’ machine result where it will cause losses towards the Hotayi Electronics Sdn Bhd. For the precision result, the random forest is still having the highest result with 99.92% followed by Decision Tree (99.84%), K-Nearest Neighbour (99.81%), and Logistic Regression with 99.75%.

Furthermore, the F1-score is also evaluated as it is the harmonic mean of precision and recall where it provides a better measure towards the incorrectly classified cases rather than only the accuracy metric. With the use of harmonic mean, it will panelized the extreme values (Huilgol, 2019). According to the results shown in the table above, the Random Forest remains the top 1 result with 99.60% of F1-score followed by K-Nearest Neighbour (99.26%), Logistic Regression (99.52%), and Decision Tree (98.54%).

In summary, the random forest is the best model where we selected for deployment as it consistently provides the highest percentage of accuracy, precision, recall, and F1-score compared to the other three models.

6.2 Model Deployment

We have a rough plan for deploying our system. We will use pickle and streamlit for deployment. Random Forest model will be saved as a pickle object in Python and loaded to develop a system tool that will assist users in estimating and predicting the machine's results, whether the machine will give an 'OK' or 'NG' result. Pickle is a Python module that is used for serialising and deserializing Python object structures, often known as marshalling or flattening (Deore, 2020). Pickling is the process of converting a Python object hierarchy into a byte stream, while unpickling is the process of converting a byte stream (from a binary file or bytes-like object) back into an object hierarchy (*pickle — Python object serialization — Python 3.7.3 documentation*, 2019). Streamlit is a Python framework for constructing Machine Learning and Data Science web apps that is open-source (Srivignesh_R, 2021).

In order to perform the deployment, the pickle and streamlit libraries have been imported. Pickle.dump() takes two parameters which are the “model” that we want to pickle and the file to which the “model” has to be saved. It serializes the “model” hierarchy and returns the bytes object representing the serialized “model”. This deployment code will be written in the deploy.py. We do not run Figure 6.1 because we are not using streamlit to deploy the system in this assignment.

```
#%%writefile deploy.py
#import pickle
#import streamlit as st
#outfile =open("Classifier.pk1", mode="wb")
#pickle.dump(model, outfile")
#outfile.close
```

Figure 6.1: Rough plan for deploying our system using streamlit.

The predictions of the machine operations result are performed when the user is running the system with the input of values of Point1, Point2, Point3, Point4, day of the week, and time of the operation performed as according to our training columns. The processed result through this system will be the operation state of the machine whether it is ‘OK’ or ‘NG’ when operated under particular conditions. Through this predicted result, it can allow Hotayi Electronics Sdn Bhd to find out the optimal time to operate the machine to reduce the machine's time.

In this case, we will use the model selected in section 6.1 Selection of Best Model and trained in Section 4.4 Random Forest for deployment. Below are the deployment steps:

```
def prediction (day,Point1,Point2,Point3,Point4,Time):
  if(Point1 == ''):
    Point1=0
  if(Point2 == ''):
    Point2=0
  if(Point3 == ''):
    Point3=0
  if(Point4 == ''):
    Point4=0
  day_of_week_Monday = 0
  day_of_week_Tuesday = 0
  day_of_week_Wednesday = 0
  day_of_week_Thursday = 0
  day_of_week_Friday = 0
  day_of_week_Saturday = 0
  day_of_week_Sunday = 0
  if(day == 0):
    day_of_week_Monday = 0
    day_of_week_Tuesday = 0
    day_of_week_Wednesday = 0
    day_of_week_Thursday = 0
    day_of_week_Friday = 0
    day_of_week_Saturday = 0
    day_of_week_Sunday = 1
  elif(day == '1'):
    day_of_week_Monday = 1
    day_of_week_Tuesday = 0
    day_of_week_Wednesday = 0
    day_of_week_Thursday = 0
    day_of_week_Friday = 0
    day_of_week_Saturday = 0
    day_of_week_Sunday = 0
```

```
elif(day == '2'):
  day_of_week_Monday = 0
  day_of_week_Tuesday = 1
  day_of_week_Wednesday = 0
  day_of_week_Thursday = 0
  day_of_week_Friday = 0
  day_of_week_Saturday = 0
  day_of_week_Sunday = 0
elif(day == '3'):
  day_of_week_Monday = 0
  day_of_week_Tuesday = 0
  day_of_week_Wednesday = 1
  day_of_week_Thursday = 0
  day_of_week_Friday = 0
  day_of_week_Saturday = 0
  day_of_week_Sunday = 0
elif(day == '4'):
  day_of_week_Monday = 0
  day_of_week_Tuesday = 0
  day_of_week_Wednesday = 0
  day_of_week_Thursday = 1
  day_of_week_Friday = 0
  day_of_week_Saturday = 0
  day_of_week_Sunday = 0
elif(day == '5'):
  day_of_week_Monday = 0
  day_of_week_Tuesday = 0
  day_of_week_Wednesday = 0
  day_of_week_Thursday = 0
  day_of_week_Friday = 1
  day_of_week_Saturday = 0
  day_of_week_Sunday = 0
```

```
    elif(day == '6'):
      day_of_week_Monday = 0
      day_of_week_Tuesday = 0
      day_of_week_Wednesday = 0
      day_of_week_Thursday = 0
      day_of_week_Friday = 0
      day_of_week_Saturday = 1
      day_of_week_Sunday = 0
    if(Time == ''):
      Time = 0
    prediction = rf_grid.predict([[Point1,Point2,Point3,Point4, day_of_week_Monday,
      day_of_week_Tuesday ,
      day_of_week_Wednesday ,
      day_of_week_Thursday ,
      day_of_week_Friday ,
      day_of_week_Saturday,
      day_of_week_Sunday,Time]])
    if(prediction == 1):
      pred = 'OK'
    else:
      pred = 'NG'
    return pred
```

```
def main():
  point1Valid = False
  point2Valid = False
  point3Valid = False
  point4Valid = False
  # validate Point1
  while point1Valid == False:
    Point1=input("Enter Point1 value:")
    try:
      Point1 = float(Point1)
      point1Valid = True
    except:
      print("Point value should only contain float value.\n")
  # validate Point2
  while point2Valid == False:
    Point2=input("Enter Point2 value:")
    try:
      Point2 = float(Point2)
      point2Valid = True
    except:
      print("Point value should only contain float value.\n")

  # validate Point3
  while point3Valid == False:
    Point3=input("Enter Point3 value:")
    try:
      Point3 = float(Point3)
      point3Valid = True
    except:
      print("Point value should only contain float value.\n")
```

```
# validate Point4
while point4Valid == False:
  Point4=input("Enter Point4 value:")
  try:
    Point4 = float(Point4)
    point4Valid = True
  except:
    print("Point value should only contain float value.\n")

# validate day input
dayValid = False
day = 0
while dayValid == False:
  day=input("Enter day value:")
  try:
    day = int(day)
    if int(day) >= 0 and int(day) <= 6
    :
      dayValid = True
    else:
      print("Invalid input the number of day should between 0 to 6.\n")
  except:
    print("Day value should only contain integer value.\n")
```

```
# validate time input
print("- Time Value -")
hour = 0
hourValid = False
while hourValid == False:
  hour=input("Enter hour of the day: ")
  try:
    hour = int(hour)
    if int(hour) >= 0 and int(hour) <= 23:
      hourValid = True
    else:
      print("Invalid input the number of hour should between 0 to 23.\n")
  except:
    print("Hour value should only contain integer value.\n")
min = 0
minValid = False
while minValid == False:
  min=input("Enter minutes of the day: ")
  try:
    min = int(min)
    if int(min) >= 0 and int(min) <= 59:
      minValid = True
    else:
      print("Invalid input the number of minutes should between 0 to 59.\n")
  except:
    print("Minutes value should only contain integer value.\n")
```

```
  sec = 0
  secValid = False
  while secValid == False:
    sec=input("Enter seconds of the day: ")
    try:
      sec = int(sec)
      if int(sec) >= 0 and int(sec) <= 59:
        secValid = True
      else:
        print("Invalid input the number of seconds should between 0 to 59.\n")
    except:
      print("Seconds value should only contain integer value.\n")

  # Calculate numer of Time by convert hour and minutes to seconds
  time = hour * 60 * 60 + min * 60 + sec
  result = prediction(day, Point1, Point2, Point3,Point4, time)
  print('Predicted machine result: ', result)
  return

main()
```


Figure 6.2: Predictions of the machine operations result.

1. Firstly, we prompt the user for 8 values: Point1, Point2, Point3, Point4, day, hour, minutes, and seconds.
2. If user input is a null value for any of the input, it will be replaced by 0.
3. If the user input for the day is less than 0 or more than 6, an error message will be displayed to the user and the user is required to re-enter the day_of_week value.
4. If the user input for the hour is less than 0 or more than 24, an error message will be displayed to the user and the user is required to re-enter the hour value.
5. If the user input for the minutes is less than 0 or more than 59, an error message will be displayed to the user and the user is required to re-enter the minutes' value.
6. If the user input for the seconds is less than 0 or more than 59, an error message will be displayed to the user and the user is required to re-enter the seconds' value.
7. The hour and minutes will then be converted to seconds and combined with the seconds' value input by the user and stored in the time variable.
8. Before prediction, the value of the day will also be converted accordingly where only the selected day of the week's value will be input as 1 while the other is input as 0.
9. The user input is then fit into the best model, which is the Random Forest Classifier with the parameter acquired from the user and hyperparameter of n_estimators = 40, criterion = 'entropy', max_depth = None, max_features = 4, min_samples_split = 2, min_samples_leaf = 1 and random_state = 0.

10. The predicted machine result is then computed and displayed to the user.

```
Enter Point1 value:55.7
Enter Point2 value:78.9
Enter Point3 value:-3.5
Enter Point4 value:52.5
Enter day value:5
- Time Value -
Enter hour of the day: 23
Enter minutes of the day: 45
Enter seconds of the day: 15
Predicted machine result:  OK
```

Figure 6.3: Output of prediction of the machine operations result.

7.0 Conclusion

In conclusion, the Random Forest Classifier is the optimal model to apply with this collection of data. Even when multiple circumstances are used, such as removing or substituting NaN values, with or without outliers, the Random Forest Classifier consistently outperforms the other models in terms of accuracy, but the Logistic Regression consistently provides the worst results. Moreover, through including the one-hot encoded day of week column and outliers when NaN values are substituting, the performance of the model is also improved obviously. This is because this strategy increases the model's versatility which can help the model be able to predict the machine results more accurately for any four points entered by the user.

The first objective has been accomplished because the deployment results demonstrate that the Random Forest Classifier is capable of correctly predicting the majority of machine outputs after training on the dataset provided by Hotayi Electronics Sdn Bhd. Thus, Hotayi Electronic Sdn Bhd may forecast machine results by feeding the Random Forest Classifier model values for Point1 to 4, the machine operation time, and day. The second objective has been successfully achieved where we are able to extract the important features including Point1, Point2, Point3, Point4, and machine results. Moreover, we successfully preprocess the data provided by Hotayi Electronics Sdn Bhd by removing the outliers and replacing the missing values using regression. All data are presented with visualization forms such as bar charts, pie charts, box plots, heat maps and line charts. The third objective was effectively accomplished because our model was evaluated and found to have a 99% accuracy after fitting and training on our processed data. As a result, we believe that our data model could assist Hotayi Electronic Sdn Bhd in gaining a better understanding of machine operations on a periodic basis and thereby optimising manufacturing machine operations.

Through this assignment, we are able to apply the knowledge gained from the lecturer and practical class to create a data model based on real data provided by Hotayi Electronic Sdn Bhd. This allows us to gain a better understanding of the entire process of handling data, analysing it, training it, and finally developing a data model successfully. Although we first had significant difficulties in comprehending and processing the data, with the guidance of our tutor, Miss Noor Aida, and weekly discussions among the team members, we were able to obtain a deeper understanding and ultimately produce a high-accuracy data model. Finally, we are appreciative that we were able to complete this task within the allotted time frame and with our best effort. In the future, we intend to study a broader variety of datasets and to augment our data model using a variety of preprocessing approaches and classifiers.

Reference

Abhishek Sharma. (2020, March 22). *Use Google Colab for Deep Learning and Machine Learning Models*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/03/google-colab-machine-learning-deep-learning/

Aunalytics. (2015, January 30). *Decision Trees: An Overview*. Aunalytics. https://www.aunalytics.com/decision-trees-an-overview/

Avinash Navlani. (2018a). *Decision Tree Classification in Python*. Datacamp. https://www.datacamp.com/community/tutorials/decision-tree-classification-python

Ayush Pant. (2019, January 22). *Introduction to Logistic Regression*. Medium; Towards Data Science. https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Brownlee, J. (2016, September 22). *K-Nearest Neighbors for Machine Learning*. Machine Learning Mastery. https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/

Christopher, A. (2021, February 3). *K-Nearest Neighbor*. Medium. https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4

Classification: ROC Curve and AUC | Machine Learning Crash Course. (2019). Google Developers. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

CTOS Data Systems Sdn Bhd. (2021). *HOTAYI ELECTRONIC (M) SDN. BHD. - Find out more!* Www.ctoscredit.com.my. https://www.ctoscredit.com.my/business/HOTAYI-ELECTRONIC-%28M%29-SDN-BHD-0245148D

Deepak Patel. (2021, February 25). *Decision Tree Algorithm for Classification : Machine Learning 101*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/02/machine-learning-101-decision-tree-algorithm-for-classification/

Deore, A. (2020, August 19). *Pickling Machine Learning Models*. Medium. https://betterprogramming.pub/pickling-machine-learning-models-aeb474bc2d78

Devi. (2020, March 29). *How to handle Missing values?* Medium. https://medium.com/analytics-vidhya/how-to-handle-missing-values-cbd03fb79ef8

Dhanasekar, S. (2021, February 21). *Random Forest Classifier- A Beginner's Guide*. Numpy Ninja. https://www.numpyninja.com/post/random-forest-classifier-a-beginner-s-guide

Donges, N. (2019). *A Complete Guide to the Random Forest Algorithm*. Built In. https://builtin.com/data-science/random-forest-algorithm

Ed Burns. (2020). *What is data preparation? - Definition from WhatIs.com*. SearchBusinessAnalytics. https://searchbusinessanalytics.techtarget.com/definition/data-preparation

Hale, J. (2020, April 7). *Don't Sweat the Solver Stuff*. Medium. https://towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451

Hotayi Electronic Sdn Bhd. (2017). *Hotayi Electronic (M) Sdn. Bhd. - Hotayi Electronic (M) Sdn. Bhd.* Www.hotayi.com. http://www.hotayi.com/

Huilgol, P. (2019, August 24). *Accuracy vs. F1-Score*. Medium. https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

Iuemon96. (2020, July 31). *Missing data imputation with fancyimpute*. GeeksforGeeks. https://www.geeksforgeeks.org/missing-data-imputation-with-fancyimpute/

Jason Brownlee. (2019, June 19). *Classification Accuracy is Not Enough: More Performance Measures You Can Use*. Machine Learning Mastery. https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

Javapoint.com. (2021). *Machine Learning Decision Tree Classification Algorithm - Javatpoint*. Www.javatpoint.com. https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

Jeremy Jordan. (2017, July 21). *Evaluating a machine learning model.* Jeremy Jordan. https://www.jeremyjordan.me/evaluating-a-machine-learning-model/

Joint Statistical Meeting. (2018). https://stefvanbuuren.name/fimd/missing-data-pattern.html. In *stefvanbuuren.name*. Chapman & Hall/CRC. https://stefvanbuuren.name/fimd/missing-data-pattern.html

Kasture, N. (2020, July 5). *Why it is important to handle missing data and 10 methods to do it.* Medium. https://medium.com/analytics-vidhya/why-it-is-important-to-handle-missing-data-and-10-methods-to-do-it-29d32ec4e6a

Liu, C. (n.d.). *Data Transformation: Standardization vs Normalization*. KDnuggets. Retrieved August 22, 2021, from https://www.kdnuggets.com/2020/04/data-transformation-standardization-normalization.html

Logistic Regression in Machine Learning - Javatpoint. (n.d.). Www.javatpoint.com. Retrieved August 23, 2021, from https://www.javatpoint.com/logistic-regression-in-machine-learning

Navlani, A. (2018b). *KNN Classification using Scikit-learn*. Datacamp. https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

pickle — Python object serialization — Python 3.7.3 documentation. (2019). Python.org. https://docs.python.org/3/library/pickle.html

Pothabattula, S. K. (2021, May 22). *A complete understanding of how the Logistic Regression can perform classification?* Medium. https://medium.com/analytics-vidhya/a-complete-understanding-of-how-the-logistic-regression-can-perform-classification-a8e951d31c76

ProjectPro. (2021). *How to optimize hyper parameters of a DecisionTree model using Grid Search in Python? - #hackerday-dezyre*. DeZyre. https://www.dezyre.com/recipes/optimize-hyper-parameters-of-decisiontree-model-using-grid-search-in-python

Raschka, S. (2021, August 29). *Why is Nearest Neighbor a Lazy Algorithm?* Dr. Sebastian Raschka. https://sebastianraschka.com/faq/docs/lazy-knn.html

Sarang Narkhede. (2018, June 26). *Understanding AUC - ROC Curve*. Medium; Towards Data Science. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

scikit-learn developers. (2019). *sklearn.neighbors.KNeighborsClassifier — scikit-learn 0.22.1 documentation*. Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Shahidul Islam Khan, & Abu Sayed Md Latiful Hoque. (2020). SICE: an improved missing data imputation technique. *Journal of Big Data*, *7*(1). https://doi.org/10.1186/s40537-020-00313-w

sklearn.linear_model.LogisticRegression — scikit-learn 0.21.2 documentation. (2014). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

sklearn.metrics.accuracy_score — scikit-learn 0.24.1 documentation. (2021). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#:~:text=Accuracy%20classification%20score.%20In%20multilabel%20classification%2C%20this%20function

sklearn.metrics.precision_score — scikit-learn 0.23.2 documentation. (2021). Scikit-Learn.org. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

Smart Vision Europe. (2021). *Redirect Notice*. Www.google.com. https://www.google.com/url?q=https://www.sv-europe.com/crisp-dm-methodology/%23three&sa=D&source=editors&ust=1629304291881000&usg=AOvVaw0Ox7HE5pAMqg6_paB5Tj4g

Spiro, R. (2020, October 30). *Hyperparameter Tuning For Random Forest - My Coding Marathon*. Rspiro9.Github.io. https://rspiro9.github.io/hyperparameter_tuning_for_random_forest

Srivignesh_R. (2021, June 21). *Streamlit Web App | Build Web Applications using Streamlit*. Analytics Vidhya.

https://www.analyticsvidhya.com/blog/2021/06/build-web-app-instantly-for-machine-learning-using-streamlit/

Stapel, E. (2019). *Interquartile Ranges (IQRs) & Outliers | Purplemath*. Purplemath. https://www.purplemath.com/modules/boxwhisk3.htm

Twin, A. (2019). *Data Mining: How Companies Use Data to Find Useful Patterns and Trends*. Investopedia. https://www.investopedia.com/terms/d/datamining.asp

Will Koehrsen. (2021). *When Accuracy Isn't Enough, Use Precision and Recall to Evaluate Your Classification Model*. Built In. https://builtin.com/data-science/precision-and-recall

Appendix

Chapter 3: Mean and Regression Accuracy
K-Nearest Neighbour

```
# KNN mean
from sklearn.neighbors import KNeighborsClassifier # 1. choose estimator

knn = KNeighborsClassifier(n_neighbors = 5) # 2. instantiate the model

# k-nearest neighbor, in coding we called it as n_neighbors

knn.fit(X_train_mean, y_train_mean) # 3. fit model to data
knn.score(X_test_mean, y_test_mean)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
predictions = knn.predict(X_test_mean)
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_mean, predictions)
# Calculate individual metrics
mse = mean_squared_error(y_test_mean, predictions)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_mean, predictions)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.96290
MSE: 0.03710
RMSE: 0.19260
```

Appendix 1: Accuracy, MSE and RMSE for KNN (missing value replaced by mean).

```
# KNN reg
from sklearn.neighbors import KNeighborsClassifier # 1. choose estimator

knn = KNeighborsClassifier(n_neighbors = 5) # 2. instantiate the model

# k-nearest neighbor, in coding we called it as n_neighbors

knn.fit(X_train_reg, y_train_reg) # 3. fit model to data
knn.score(X_test_reg, y_test_reg)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
predictions = knn.predict(X_test_reg)
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_reg, predictions)
# Calculate individual metrics
mse = mean_squared_error(y_test_reg, predictions)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_reg, predictions)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.95974
MSE: 0.04026
RMSE: 0.20065
```

Appendix 2: Accuracy, MSE and RMSE for KNN (missing value replaced by regression).

```
#Logistic regression (mean)
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression()
logreg.fit(X_train_mean, y_train_mean)

# check the score of testing set
y_pred = logreg.predict(X_test_mean)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
# predictions = knn.predict(Xtest)
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_mean, y_pred)
# Calculate individual metrics
mse = mean_squared_error(y_test_mean, y_pred)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_mean, y_pred)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.93064
MSE: 0.06936
RMSE: 0.26337
```

Appendix 3: Accuracy, MSE and RMSE for Logistic Regression(missing value replaced by mean).

```
#Logistic regression (reg)
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

logreg = LogisticRegression()
logreg.fit(X_train_reg, y_train_reg)

# check the score of testing set
y_pred = logreg.predict(X_test_reg)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_reg, y_pred)
# Calculate individual metrics
mse = mean_squared_error(y_test_reg, y_pred)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_reg, y_pred)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.91349
MSE: 0.08651
RMSE: 0.29413
```

Appendix 4: Accuracy, MSE and RMSE for Logistic Regression (missing value replaced by regression).

```
#Decision Tree (mean)
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

dt = DecisionTreeClassifier()
dt.fit(X_train_mean, y_train_mean)

# check the score of testing set
y_pred = dt.predict(X_test_mean)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
# predictions = knn.predict(Xtest)
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_mean, y_pred)
# Calculate individual metrics
mse = mean_squared_error(y_test_mean, y_pred)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_mean, y_pred)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.97789
MSE: 0.02211
RMSE: 0.14870
```

Appendix 5: Accuracy, MSE and RMSE for Decision Tree (missing value replaced by mean).

\

```
#Decision Treen (reg)
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics

dt = DecisionTreeClassifier()
dt.fit(X_train_reg, y_train_reg)

# check the score of testing set
y_pred = dt.predict(X_test_reg)

from sklearn.metrics import mean_squared_error
import math
# Get predictions
from sklearn.metrics import accuracy_score
# accuracy
accuracy_score(y_test_reg, y_pred)
# Calculate individual metrics
mse = mean_squared_error(y_test_reg, y_pred)
print("Accuracy Score: {:.5f}".format(accuracy_score(y_test_reg, y_pred)))

print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.98782
MSE: 0.01218
RMSE: 0.11037
```

Appendix 6: Accuracy, MSE and RMSE for Decision Tree (missing value replaced by regression).

```
# RF mean
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
import math

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train_mean,y_train_mean)

# prediction on test set
y_pred=clf.predict(X_test_mean)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy Score: {:.5f}".format(metrics.accuracy_score(y_test_mean, y_pred)))

mse = mean_squared_error(y_test_mean, y_pred)
print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.98655
MSE: 0.01345
RMSE: 0.11598
```

Appendix 7: Accuracy, MSE and RMSE for Random Forest (missing value replaced by mean).

```
# RF reg
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import mean_squared_error
import math

#Create a Gaussian Classifier
clf=RandomForestClassifier(n_estimators=100)

#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train_reg,y_train_reg)

# prediction on test set
y_pred=clf.predict(X_test_reg)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy Score: {:.5f}".format(metrics.accuracy_score(y_test_reg, y_pred)))

mse = mean_squared_error(y_test_reg, y_pred)
print("MSE: {0:.5f}".format(mse))
rmse = math.sqrt(mse)
print("RMSE: {0:.5f}".format(rmse))
```

```
Accuracy Score: 0.99291
MSE: 0.00709
RMSE: 0.08420
```

Appendix 8: Accuracy, MSE and RMSE for Random Forest (missing value replaced by regression).

Chapter 4
Logistic Regression

```
#Logistic regression

from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
logistic_Reg = LogisticRegression()
parameters = [{'C': list(np.arange(1000, 2000, 200)),
               'fit_intercept': [True, False],
               'tol' : [1e-5,1e-4],
               'solver' : ['newton-cg','lbfgs', 'liblinear']}]

lr_grid = GridSearchCV(logistic_Reg, parameters)
lr_grid.fit(X_train_reg, y_train_reg)

lr_acc_test = (lr_grid.score(X_test_reg, y_test_reg)) * 100
lr_acc_train = (lr_grid.score(X_train_reg, y_train_reg)) * 100

print("Best Score", lr_grid.best_score_)
print("Best Estimator for solver", lr_grid.best_estimator_.solver )
print("Best Estimator for C", lr_grid.best_estimator_.C )
print("Best Estimator for fit_intercept", lr_grid.best_estimator_.fit_intercept )
print("Best Estimator for tol", lr_grid.best_estimator_.tol )

print('LR accuracy for test set: {:.2f}%'.format(lr_acc_test))
print('LR accuracy for training set: {:.2f}%'.format(lr_acc_train))
```

```
Best Score 0.9147883354431169
Best Estimator for solver lbfgs
Best Estimator for C 1800
Best Estimator for fit_intercept False
Best Estimator for tol 1e-05
LR accuracy for test set: 91.45%
LR accuracy for training set: 91.46%
```

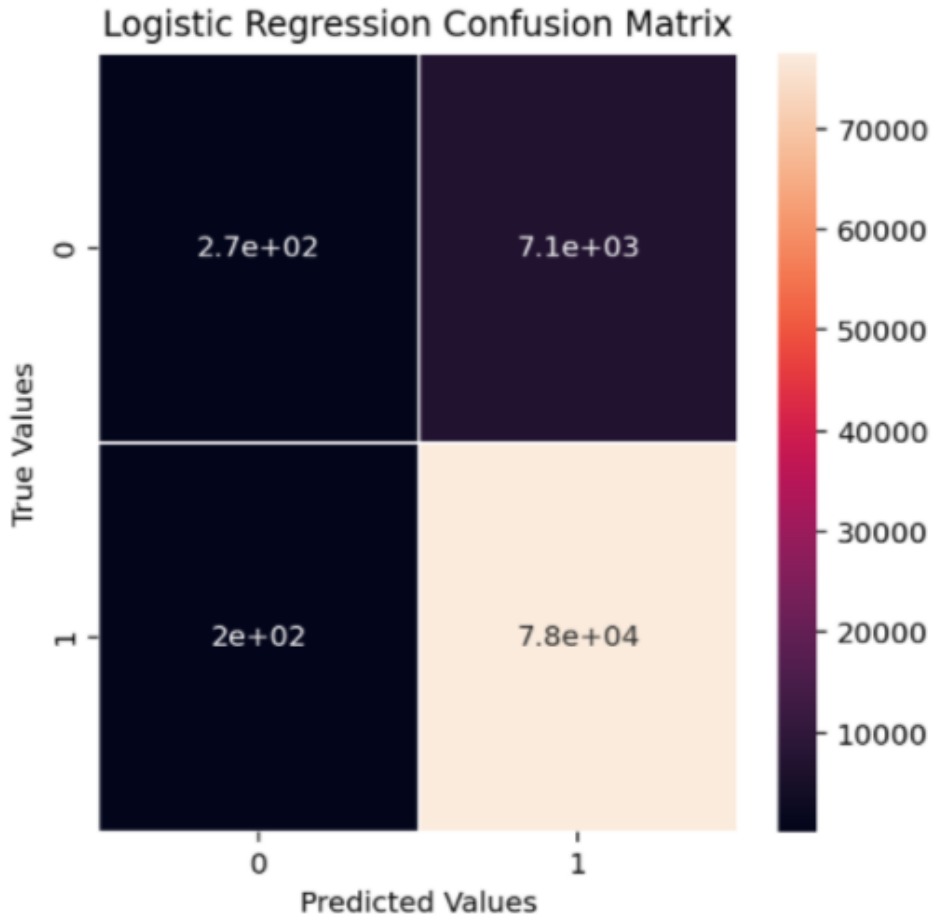
Appendix 9: GridSearchCV for parameter setting and accuracy for test and training set.

```
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix

lr_cm = confusion_matrix(y_test_reg,lr_grid.predict(X_test_reg))
lr_cm
```

```
array([[  266,  7072],
       [  196, 77515]])
```

Appendix 10: Confusion matrix.

```
# Visualizing Confusion Matrix using Heatmap

import matplotlib.pyplot as plt
plt.subplots(figsize=(5,5))
sns.heatmap(lr_cm,annot=True,linewidths=0.5)
plt.title("Logistic Regression Confusion Matrix")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()
```



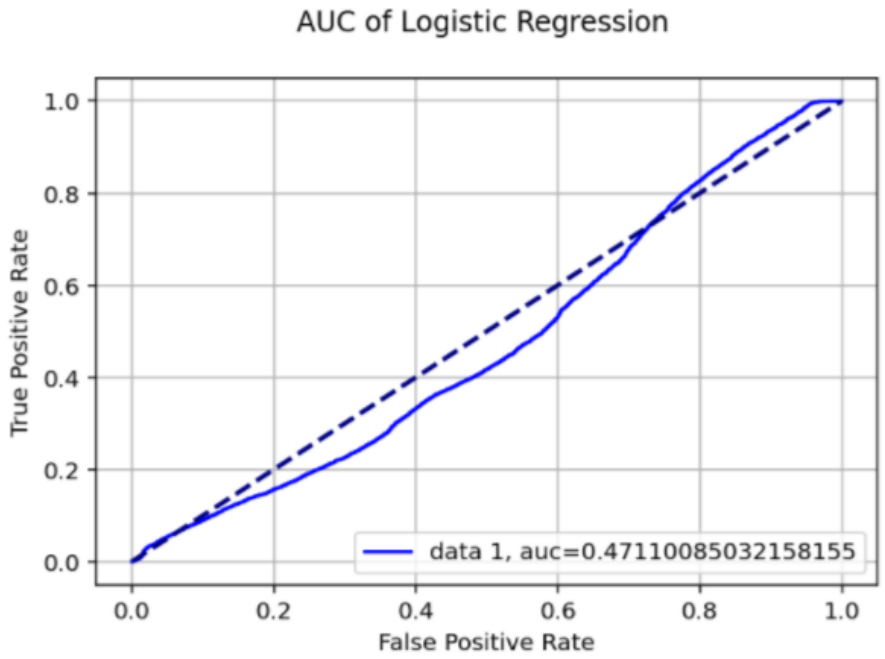
Appendix 11: Confusion matrix in heatmap.

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

y_true_lr, y_pred_lr = y_test_reg, lr_grid.predict(X_test_reg)
print(classification_report(y_true_lr, y_pred_lr))
lr_pre = precision_score(y_true_lr,y_pred_lr) * 100
lr_rec = recall_score(y_true_lr,y_pred_lr) * 100
lr_f1 = f1_score(y_true_lr,y_pred_lr) * 100
```

```
              precision    recall  f1-score   support

         0.0       0.58      0.04      0.07      7338
         1.0       0.92      1.00      0.96     77711

    accuracy                           0.91     85049
   macro avg       0.75      0.52      0.51     85049
weighted avg       0.89      0.91      0.88     85049
```

Appendix 12: Confusion report.

```
y_pred_proba =lr_grid.predict_proba(X_test_reg)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test_reg, y_pred_proba)
lr_auc = metrics.roc_auc_score(y_test_reg, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(lr_auc),color='blue')
lw = 2
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.legend(loc=4)
plt.title("AUC of Logistic Regression \n")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.grid(True)
plt.show()
```



Appendix 13: Area under graph (AUC).

Decision Tree

```
#to find the set of hyperparameters that will give the best result
from sklearn import decomposition
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier

std_slc = StandardScaler()
pca = decomposition.PCA()
dt = DecisionTreeClassifier()
pipe = Pipeline(steps=[('std_slc', std_slc),
                       ('pca', pca),
                       ('dt', dt)])
n_components = list(range(1,X_train_reg.shape[1]+1,1))

criterion = ['gini', 'entropy']
max_depth = [2,4,6,8,10,12]

parameters = dict(pca__n_components=n_components,
                  dt__criterion=criterion,
                  dt__max_depth=max_depth)

clf_GS = GridSearchCV(pipe, parameters)
clf_GS.fit(X_train_reg, y_train_reg)

print('Best Criterion:', clf_GS.best_estimator_.get_params()['dt__criterion'])
print('Best max_depth:', clf_GS.best_estimator_.get_params()['dt__max_depth'])
print('Best Number Of Components:', clf_GS.best_estimator_.get_params()['pca__n_components'])
print();
print(clf_GS.best_estimator_.get_params()['dt'])
```

```
Best Criterion: entropy
Best max_depth: 12
Best Number Of Components: 11

DecisionTreeClassifier(criterion='entropy', max_depth=12)
```

Appendix 14: GridSearchCV for parameter setting.

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion='entropy', max_depth=12)
#learning
dt.fit(X_train_reg, y_train_reg)
dt_acc_test = (dt.score(X_test_reg, y_test_reg)) * 100
dt_acc_train = (dt.score(X_train_reg, y_train_reg)) * 100
print('DT accuracy for test set: {:.2f}%'.format(dt_acc_test))
print('DT accuracy for training set: {:.2f}%'.format(dt_acc_train))

# from sklearn.tree import DecisionTreeClassifier
# dt = DecisionTreeClassifier(criterion='entropy', max_depth=12)
# #learning
# dt.fit(X_train_reg_norm, y_train_reg)
# dt_acc_test = (dt.score(X_test_reg_norm, y_test_reg)) * 100
# dt_acc_train = (dt.score(X_train_reg_norm, y_train_reg)) * 100
# print('DT accuracy for test set: {:.2f}%'.format(dt_acc_test))
# print('DT accuracy for training set: {:.2f}%'.format(dt_acc_train))
```

```
DT accuracy for test set: 97.29%
DT accuracy for training set: 97.45%
```

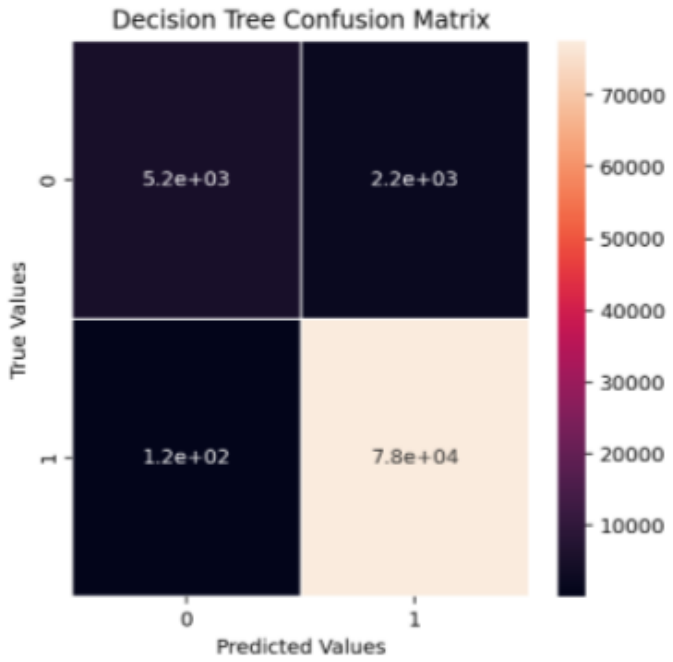
Appendix 15: Accuracy for test and training set.

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
dt_cm = confusion_matrix(y_test_reg,dt.predict(X_test_reg))
dt_cm
```

```
array([[ 5157,  2181],
       [  122, 77589]])
```

Appendix 16: Confusion matrix.

```
# Visualizing Confusion Matrix using Heatmap

import matplotlib.pyplot as plt
plt.subplots(figsize=(5,5))
sns.heatmap(dt_cm,annot=True,linewidths=0.5)
plt.title("Decision Tree Confusion Matrix")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()
```



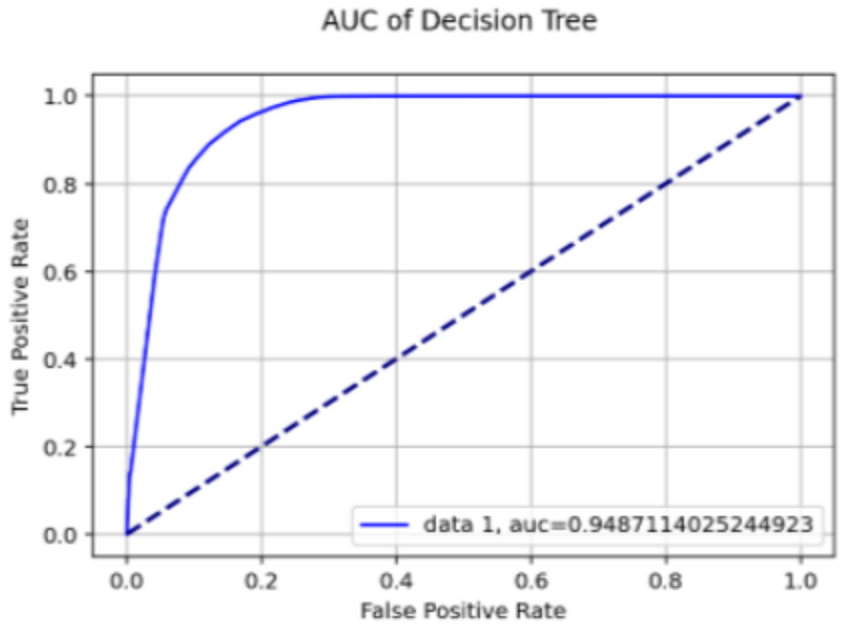
Appendix 17: Confusion matrix in heatmap.

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, f1_score, precision_score
y_true_dt, y_pred_dt = y_test_reg, dt.predict(X_test_reg)
print(classification_report(y_true_dt, y_pred_dt))
dt_pre = precision_score(y_true_dt,y_pred_dt) * 100
dt_rec = recall_score(y_true_dt,y_pred_dt) * 100
dt_f1 = f1_score(y_true_dt,y_pred_dt) * 100
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.70      0.82      7338
         1.0       0.97      1.00      0.99     77711

    accuracy                           0.97     85049
   macro avg       0.97      0.85      0.90     85049
weighted avg       0.97      0.97      0.97     85049
```

Appendix 18: Confusion report.

```
y_pred_proba = dt.predict_proba(X_test_reg)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test_reg, y_pred_proba)
dt_auc = metrics.roc_auc_score(y_test_reg, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(dt_auc),color='blue')
lw = 2
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.legend(loc=4)
plt.title("AUC of Decision Tree \n")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.grid(True)
plt.show()
```



Appendix 19: Area under graph (AUC).

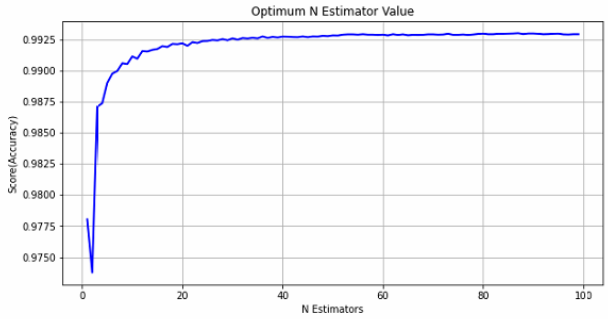
Random Forest

```
# Determine optimal parameters:
# Declare a baseline classifier:
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier()

# Create the grid parameter:
grid_rf = {'n_estimators': [20, 40],
           'criterion': ['entropy', 'gini'],
           'max_depth': [None,1,3,5,7],
           'max_features': range(1,5),
           'min_samples_split': range(2, 5),
           'min_samples_leaf': [1,3]}

# Create the grid:
gs_rf = GridSearchCV(forest, grid_rf, cv=3, n_jobs=-1)

# Fit using grid search:
gs_rf.fit(X_train_reg,y_train_reg)

# Print best accuracy and best parameters:
print('Best accuracy: %.3f' % gs_rf.best_score_)
print('\nBest params:\n', gs_rf.best_params_)
```

```
Best accuracy: 0.988

Best params:
 {'criterion': 'gini', 'max_depth': None, 'max_features': 4, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 40}
```

Appendix 20: GridSearchCV for parameter setting.

```
#1. Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

#2. Determine the Optimum n_estimators value
scores = []
for i in range(1,100):
    #RFfind = RandomForestClassifier(n_estimators = i,random_state=0)

    RFfind = RandomForestClassifier(n_estimators=i, criterion = 'entropy', max_depth=None, max_features=4, min_samples_split=2, min_samples_leaf= 1, random_state=0)

    RFfind.fit(X_train_reg,y_train_reg)
    scores.append(RFfind.score(X_test_reg,y_test_reg))

plt.figure(1, figsize=(10, 5))
plt.plot(range(1,100),scores,color="blue",linewidth=2)
plt.title("Optimum N Estimator Value")
plt.xlabel("N Estimators")
plt.ylabel("Score(Accuracy)")
plt.grid(True)
plt.show()
```



Appendix 21: Optimum N estimator value

```
#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier

rf_grid = RandomForestClassifier(n_estimators=40, criterion = 'entropy', max_depth=None, max_features=4, min_samples_split=2, min_samples_leaf= 1, random_state=0)
rf_grid.fit(X_train_reg, y_train_reg)

rf_acc_test = (rf_grid.score(X_test_reg, y_test_reg)) * 100
rf_acc_train = (rf_grid.score(X_train_reg, y_train_reg)) * 100

print('RF accuracy for test set: {:.2f}%'.format(rf_acc_test))
print('RF accuracy for training set: {:.2f}%'.format(rf_acc_train))
```

```
RF accuracy for test set: 99.27%
RF accuracy for training set: 99.99%
```

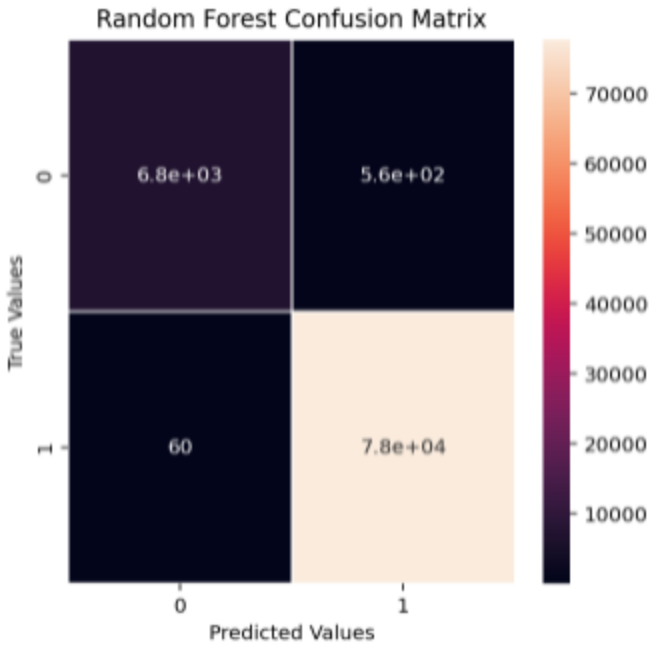
Appendix 22: Accuracy for test and training set.

```
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
rf_cm = confusion_matrix(y_test_reg,rf_grid.predict(X_test_reg))
rf_cm
```

```
array([[ 6778,   560],
       [   60, 77651]])
```

Appendix 23: Confusion matrix.

```
# Visualizing Confusion Matrix using Heatmap

import matplotlib.pyplot as plt
plt.subplots(figsize=(5,5))

sns.heatmap(rf_cm,annot=True,linewidths=0.5)

plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")

plt.show()
```



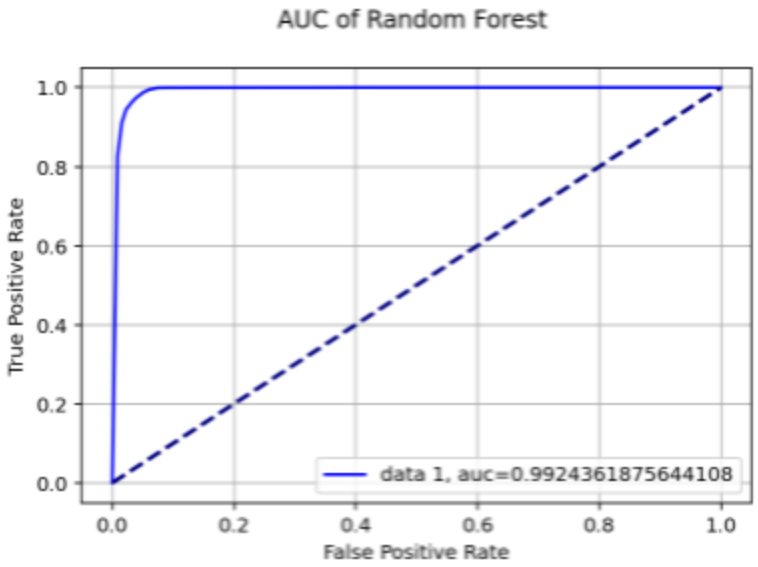
Appendix 24: Confusion matrix in heatmap.

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

y_true_rf, y_pred_rf = y_test_reg, rf_grid.predict(X_test_reg)
print(classification_report(y_true_rf, y_pred_rf))

rf_pre = precision_score(y_true_rf, y_pred_rf) * 100
rf_rec = recall_score(y_true_rf, y_pred_rf) * 100
rf_f1 = f1_score(y_true_rf, y_pred_rf) * 100
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.92      0.96      7338
         1.0       0.99      1.00      1.00     77711

    accuracy                           0.99     85049
   macro avg       0.99      0.96      0.98     85049
weighted avg       0.99      0.99      0.99     85049
```

Appendix 25: Confusion report.

```
y_pred_proba = rf_grid.predict_proba(X_test_reg)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test_reg,  y_pred_proba)
rf_auc = metrics.roc_auc_score(y_test_reg, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(rf_auc),color='blue')

lw = 2
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

plt.legend(loc=4)
plt.title("AUC of Random Forest \n")
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.grid(True)
plt.show()
```



Appendix 26: Area under graph (AUC).