TARUMT

PCB QUALITY PREDICTION

NATHAN TING
TEO SHI HAN
YEE CHOOI LI
KHOO CHONG EE

Table of Contents

1.0 Business Understanding

1.1 Company Background

Hotayi Electronic (M) Sdn. Bhd. is a global Electronics Manufacturing Services (EMS) company established in Malaysia since 1992. The company manufactures products which serve specific market segment categories such as automotive, consumer, communication, optical, storage devices, industrial equipment, DRAM modules. The company also carries out Printed Circuit Board Assembly (PCBA) processes for PCBs to be used in electronic appliances. To inspect the quality of the assembled PCBs, Hotayi Electronic has 3D Automated Optical Inspection (AOI) machines which scan the PCB for defects (e.g. dimension). A PCB which is too small (negative dimension values) or too big (positive dimension values) might not fit into the mechanical box assembly, and even worse when it needs to be fitted into compact multi-part groups such as industrial applications.

1.2 Project Brief

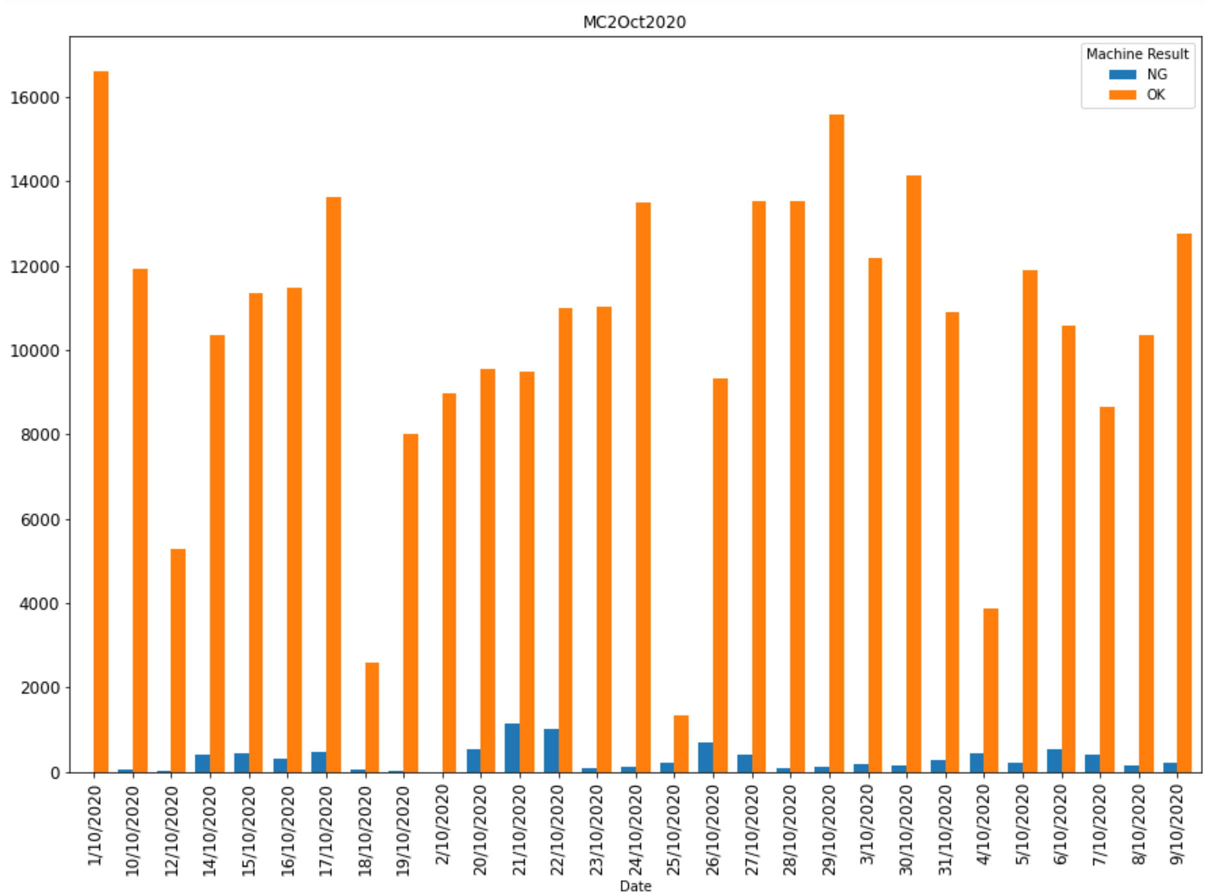
The primary objective of this project is to predict the quality of PCBs with an accuracy of 90% and greater. This objective is important in minimizing human error during the quality check of PCBs, since properly trained machine learning models will be able to determine the PCB quality more accurately and quickly. The project is deemed successful when the prediction accuracy is 90% and greater. The data will be tested with 3 different machine learning models: K-Nearest Neighbors, Gaussian Naive Bayes and Random Forest Classifier. These models will be fitted with different instances of datasets such as filling missing values with mean, modifying outliers and even justifiably excluding some data points to ensure the accuracy in the prediction. The performance of each model (accuracy, mean-squared error, recall, precision and F1 score) will be compared to determine the best model in this project.

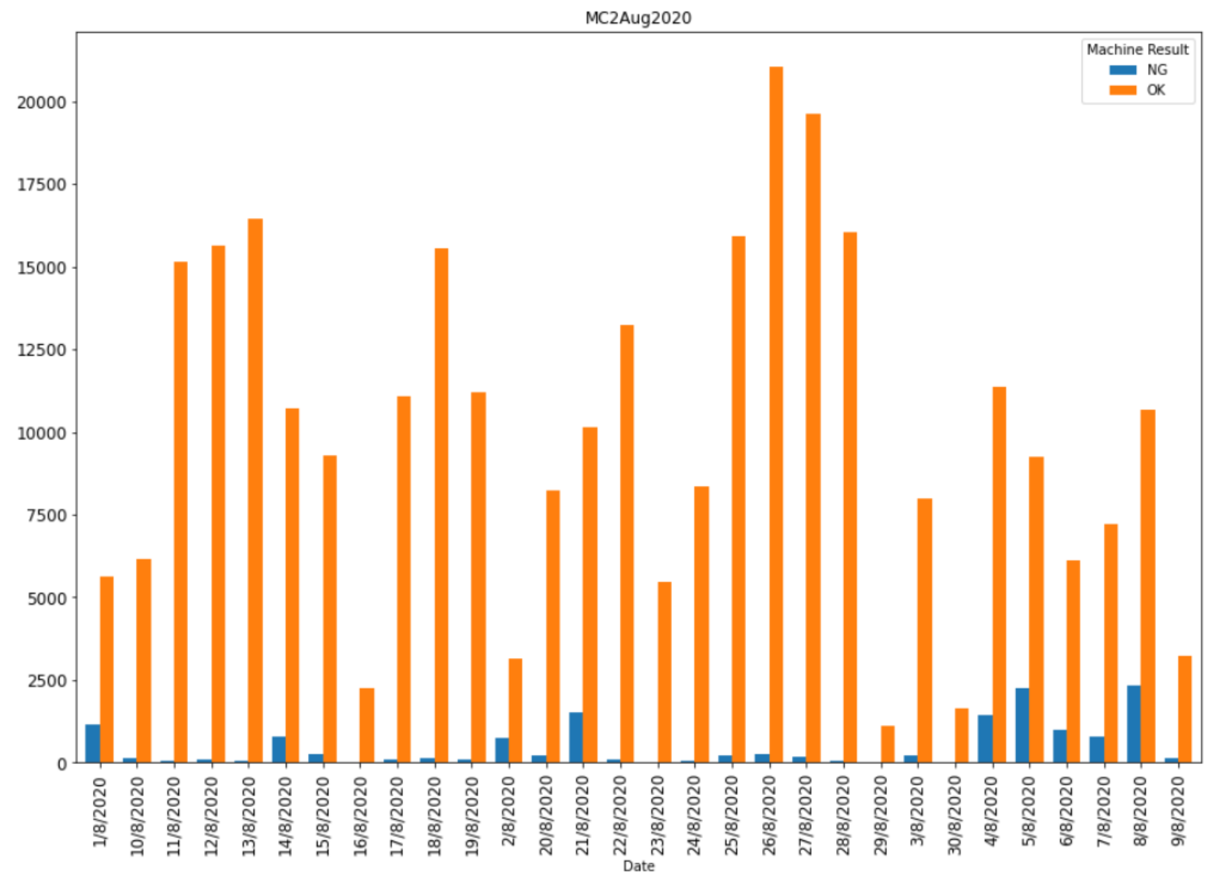
2.0 Data Understanding

In this project, 2 datasets from Hotayi Electronic (M) Sdn Bhd were used in the form of csv files: MC2 October 2020 (MC2Oct2020_csv.csv) and MC2 August 2020 (MC2Aug2020_csv.csv). The decision was made to have one main dataset for training and testing the machine learning models, whereas the other would act as a supporting dataset to make a more accurate choice for the model to be selected for deployment at the end of the project.

2.1 Target and Features

The MC2Oct2020_csv csv file contains 312389 rows x 13 columns, with the 13th day omitted due to compilation issues. The MC2August2020_csv csv file contains 313338 rows x 13 columns. In both csv files, 'Machine Result' is discretely classified into 'OK' and 'NG' for every date in the dataset, so 'Machine Result' will become the target, with the continuous data columns '(um)Point1' to '(um)Point5' becoming the features.




MC2Aug2020
Machine Result
NG
OK
20000
17500
15000
12500
10000
7500
5000
2500
0
1/8/2020
10/8/2020
11/8/2020
12/8/2020
13/8/2020
14/8/2020
15/8/2020
16/8/2020
17/8/2020
18/8/2020
19/8/2020
2/8/2020
20/8/2020
21/8/2020
22/8/2020
23/8/2020
24/8/2020
25/8/2020
26/8/2020
27/8/2020
28/8/2020
29/8/2020
3/8/2020
30/8/2020
4/8/2020
5/8/2020
6/8/2020
7/8/2020
8/8/2020
9/8/2020
Date

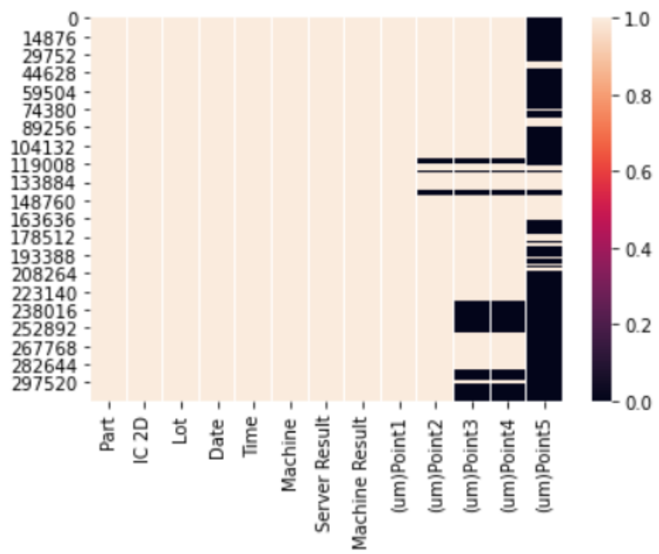
2.2 Missing Values

Both datasets had missing values, all of them being in the features columns.

MC2Oct2020 had the most missing values in ‘(um)Point 5’ (243,872), followed by ‘(um)Point 4’ (56,736), ‘(um)Point 3’ (56,736) and finally ‘(um)Point 2’ (8,886).

In [5]:
```
sns.heatmap(data=MC1csv.notnull())
```

Out[5]:
```
<AxesSubplot:>
```



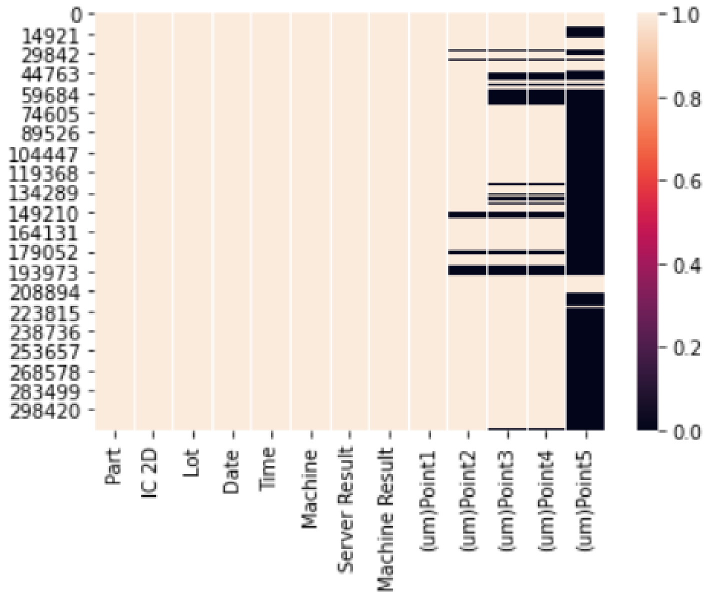
In [11]:
```
MC1csv.isnull().sum()
```

Out[11]:
```
Part                   0
IC 2D                  0
Lot                    0
Date                   0
Time                   0
Machine                0
Server Result          0
Machine Result         0
(um)Point1             0
(um)Point2          8886
(um)Point3         56736
(um)Point4         56736
(um)Point5        243872
dtype: int64
```

MC2Aug2020 had the most missing values in '(um)Point 5' (265,549), followed by '(um)Point 4' (46,331), '(um)Point 3' (46,331) and finally '(um)Point 2' (21,847).

```
In [8]: sns.heatmap(data=MC2csv.notnull())

Out[8]: <AxesSubplot:>
```



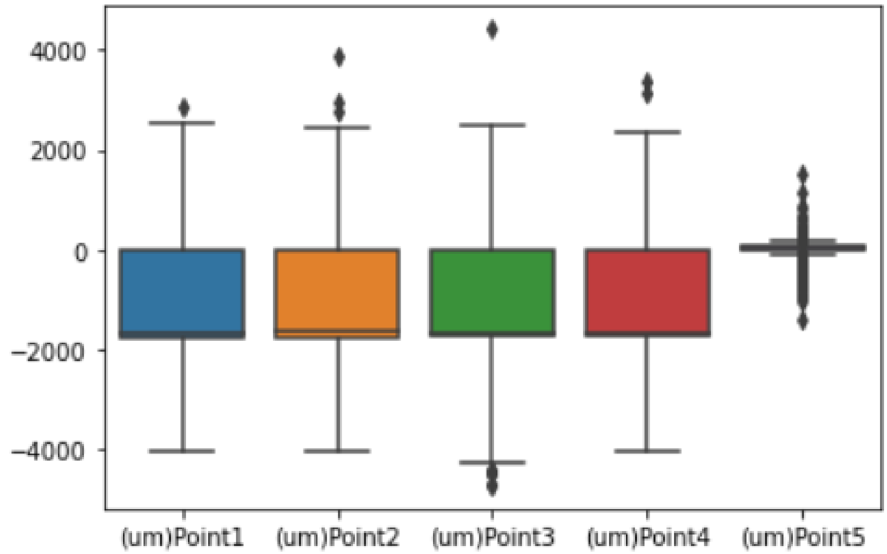

```
In [9]: MC2csv.isnull().sum()

Out[9]: Part                   0
        IC 2D                  0
        Lot                    0
        Date                   0
        Time                   0
        Machine                0
        Server Result          0
        Machine Result         0
        (um)Point1             0
        (um)Point2         21847
        (um)Point3         46331
        (um)Point4         46331
        (um)Point5        265549
        dtype: int64
```

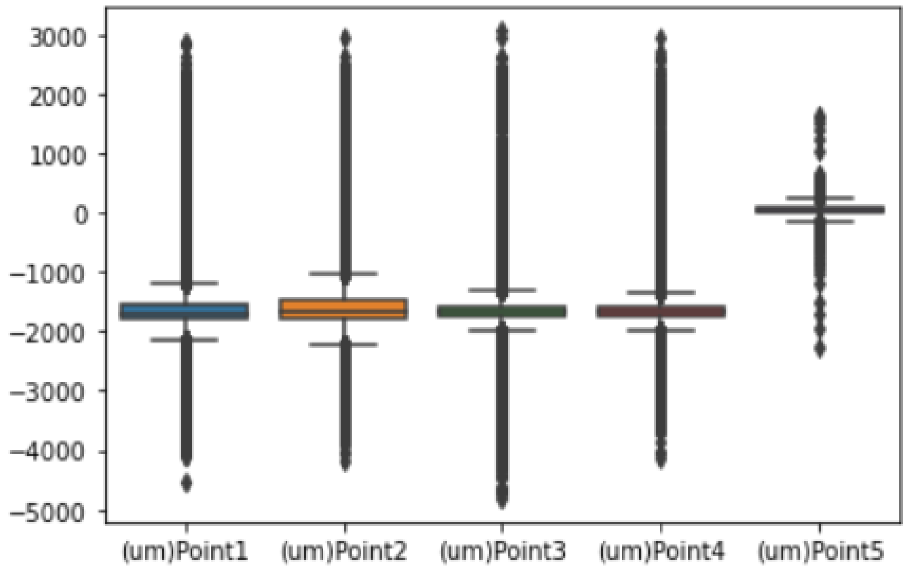
Since '(um)Point 5' had the most missing values in both datasets, an initial decision was made to train and test the models with and without '(um)Point 5' to test the accuracy of the models.

2.3 Data Dispersion

Boxplots of the feature columns of MC2Oct2020 and MC2Aug2020 were plotted to have a look at how the data is distributed in the columns.



MC2Oct2020 Boxplot



MC2Aug2020 Boxplot

MC2Oct2020 dataset had much better data dispersion and a lot less outliers across all 5 features when compared to MC2Aug2020, but the ‘(um)Point5’ plot was bad in both dataset plots, consisting mostly of outliers. Hence, it was decided that MC2Oct2020 would be the main dataset (imported using pandas.read_csv as MC1csv), whereas MC2Aug2020 would be the supporting dataset (imported using pandas.read_csv as MC2csv).

2.4 Features Correlation

The correlation score between the feature columns were checked as further data understanding.

Out[9]:

	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5
(um)Point1	1.000000	0.954770	0.943452	0.942839	-0.042108
(um)Point2	0.954770	1.000000	0.968513	0.973653	-0.085128
(um)Point3	0.943452	0.968513	1.000000	0.982353	0.002315
(um)Point4	0.942839	0.973653	0.982353	1.000000	0.426310
(um)Point5	-0.042108	-0.085128	0.002315	0.426310	1.000000

MC2Oct2020

Out[18]:

	(um)Point1	(um)Point2	(um)Point3	(um)Point4	(um)Point5
(um)Point1	1.000000	0.950796	0.927812	0.897565	0.077741
(um)Point2	0.950796	1.000000	0.913608	0.898278	0.015676
(um)Point3	0.927812	0.913608	1.000000	0.949675	-0.248482
(um)Point4	0.897565	0.898278	0.949675	1.000000	0.193515
(um)Point5	0.077741	0.015676	-0.248482	0.193515	1.000000

MC2Aug2020

In both datasets, all feature columns, except '(um)Point 5', had strong correlation with each other. This further supported the decision to train and test the models with and without '(um)Point 5' to test the accuracy of the models.

3.0 Data Preparation

3.1 Dropping Unwanted Columns

As noted before, the features are columns '(um)Point 1' to '(um)Point 5', whereas the target is column 'Machine Result'. Hence, remaining columns were dropped since they were not needed.

```
MC1 = MC1csv.drop(columns=['Part', 'IC 2D','Lot','Date','Time','Machine','Server Result'], axis=1
```

3.2 Label-Encoding The Target

As noted before, the target column 'Machine Result' is discretely classified into 'OK' and 'NG'. However, for machine learning models, the target needs to be in numeric form (float type). Hence, LabelEncoder() was imported and used to encode 'OK' into 1 and 'NG' into 0.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
MC1['Machine Result']=le.fit_transform(MC1['Machine Result'])
```

3.3 Locating Outliers, Modifying Outliers and Filling Missing Values

Rather than directly removing the outliers, the decision was made to modify the outliers to see if that could help with appropriating the dataset for training and testing. However, firstly, the outliers needed to be located. This was done by using the Interquartile Range Rule.

Interquartile Range (IQR) = 3rd Quartile - 1st Quartile
The upper limit of the dataset = 3rd Quartile + (1.5xIQR)
The lower limit of the dataset = 1st Quartile - (1.5xIQR)

Any value beyond the upper and lower limit of the dataset would be considered as outliers.
Once these outliers were found, they were converted into NaN values using numpy.nan so they could be filled in along with the missing values later on.

It was decided that the 3 most common methods of filling missing values would be used:

- Fill in missing values with Mean
- Fill in missing values with Median
- Fill in missing values with a constant, in this case, 0

```
Q1MC1 = MC1.quantile(0.25)
Q3MC1 = MC1.quantile(0.75)
IQR1 = Q3MC1-Q1MC1

MC1mean = MC1.mean()
MC1median = MC1.median()

MC1[(MC1<(Q1MC1 - 1.5 * IQR1))|(MC1>(Q3MC1 + 1.5 * IQR1))]=np.nan
MC1Mean = MC1.fillna(MC1mean)
MC1Median = MC1.fillna(MC1median)
MC1Zero = MC1.fillna(0)
```

3.4 Splitting The Dataset

Each modified instance of the dataset was split into a training set and a testing set with a 75:25 ratio by using sklearn.model_selection.train_test_split.

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, random_state=1)
```

3.5 Normalizing The Data

Normalizing the data in the features columns [‘(um)Point 1’ to ‘(um)Point 5] using MinMaxScaler() converts the numeric values to a common scale from 0 to 1, without distorting differences in the ranges of values or losing information.

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
Xtrain = scaler.fit_transform(Xtrain)
Xtest = scaler.transform(Xtest)
```

3.6 Model Hyperparameter Tuning

For each model trained and tested with each modified instance of the dataset, their model hyperparameters were also tuned using GridSearchCV to get the best possible accuracy for each model.

The models that were hyperparameter-tuned were:

- **K-Nearest Neighbors**

```
from sklearn.model_selection import GridSearchCV
parameters = [{'weights': ['uniform', 'distance'],'n_neighbors': list(np.arange(1, 100, 1))}]
grid = GridSearchCV(knn, parameters)
grid.fit(Xtrain, ytrain)
print(grid.best_estimator_)
```

- **Gaussian Naive Bayes**

```
from sklearn.model_selection import GridSearchCV
parameters = [{'var_smoothing': np.logspace(0,-9, num=100) }]
grid = GridSearchCV(gnb, parameters)
grid.fit(Xtrain, ytrain)
print(grid.best_estimator_)
```

- **Random Forest Classifier**

```
from sklearn.model_selection import GridSearchCV
parameters = [{'n_estimators':list(np.arange(1, 100, 1)) }]
grid = GridSearchCV(rfc, parameters)
grid.fit(Xtrain, ytrain)
print(grid.best_estimator_)
```

4.0 Data Modelling

As noted before, the training set and testing set were splitted with a 75:25 ratio. The models chosen to be used in this project were:

1. K-Nearest Neighbors
2. Gaussian Naive Bayes
3. Random Forest Classifier

The datasets were first fitted into the models which had default hyperparameters to see how extensive the hyperparameter tuning needed to be. Then, after tuning the hyperparameters (as noted in 3.6 Hyperparameter Tuning), the datasets were fitted into the models again. The accuracy, recall and F1 scores were noted, along with the ability of the models to classify the features into ‘OK’ or ‘NG’.

It was noted that with the missing values filled with median in all instances of datasets MC1 and MC2, all three models were unable to classify the features into 1 (‘OK’) and 0 (‘NG’), even after tuning the hyperparameters. Hence, displayed below are the results from datasets which were filled with mean and 0 respectively:

1. MC1 (MC2Oct2020) without Point 5, missing values filled with mean

Out[66]: MC1DropP5Mean

	KNN	GNB	RFC
Accuracy	0.988220	0.976350	0.987989
Mean-squared Error	0.011780	0.023650	0.012011
Precision	0.988996	0.976252	0.989389
Recall	0.998999	1.000000	0.998354
F1	0.993972	0.987983	0.993851

2. MC1 (MC2Oct2020) without Point 5, missing values filled with 0

Out[56]: MC1DropP5Zero

	KNN	GNB	RFC
Accuracy	0.988386	0.976734	0.988130
Mean-squared Error	0.011614	0.023266	0.011870
Precision	0.989112	0.976654	0.989531
Recall	0.999052	0.999974	0.998354
F1	0.994057	0.988176	0.993923

3. MC2 (MC2Aug2020) without Point 5, missing values filled with mean

Out[103]: MC2DropP5Mean

	KNN	GNB	RFC
Accuracy	0.954975	0.947597	0.955129
Mean-squared Error	0.045025	0.052403	0.044871
Precision	0.958002	0.954673	0.957879
Recall	0.996538	0.992233	0.996845
F1	0.976890	0.973091	0.976974

4. MC2 (MC2Aug2020) without Point 5, missing values filled with 0

Out[80]: MC2DropP5Zero

	KNN	GNB	RFC
Accuracy	0.955371	0.828238	0.955244
Mean-squared Error	0.044629	0.171762	0.044756
Precision	0.958207	0.986642	0.957990
Recall	0.996738	0.831386	0.996845
F1	0.977093	0.902385	0.977031

5. MC1 (MC2Oct2020) with Point 5, missing values filled with mean

Out[32]: MC1Mean

	KNN	GNB	RFC
Accuracy	0.987311	0.975326	0.988258
Mean-squared Error	0.012689	0.024674	0.011742
Precision	0.988883	0.975310	0.989993
Recall	0.998169	0.999934	0.998011
F1	0.993505	0.987469	0.993986

6. MC1 (MC2Oct2020) with Point 5, missing values filled with 0

Out[38]: MC1Zero

	KNN	GNB	RFC
Accuracy	0.987183	0.975364	0.988297
Mean-squared Error	0.012817	0.024636	0.011703
Precision	0.988933	0.975482	0.990083
Recall	0.997985	0.999789	0.997959
F1	0.993438	0.987486	0.994005

7. MC2 (MC2Aug2020) with Point 5, missing values filled with mean

Out[121]:

MC2Mean

	KNN	GNB	RFC
Accuracy	0.955984	0.954924	0.956916
Mean-squared Error	0.044016	0.045076	0.043084
Precision	0.962007	0.954924	0.961589
Recall	0.993129	1.000000	0.994613
F1	0.977320	0.976943	0.977822

Note: Gaussian Naive Bayes was unable to classify 'OK' and 'NG' after tuning its hyperparameters

8. MC2 (MC2Aug2020) with Point 5, missing values filled with 0

Out[28]:

MC2Zero

	KNN	GNB	RFC
Accuracy	0.956214	0.835131	0.956980
Mean-squared Error	0.043786	0.164869	0.043020
Precision	0.962135	0.985427	0.961484
Recall	0.993236	0.839768	0.994800
F1	0.977438	0.906785	0.977858

5.0 Evaluation

As seen above, the accuracy, mean-squared error, precision, recall and F1 score between datasets with missing values filled with mean and datasets with missing values filled with 0 were relatively similar.

For datasets without Point 5 (with respect to MC1 as our main dataset), having missing values filled with 0 resulted in the best performance across all 3 models, with Random Forest Classifier being the best performing model.

```
After tuning RandomForestClassifier
Mean squared error: 0.011869702169069631
Accuracy score: 0.9881302978309303
Precision score:  0.9895308461478213
Recall score:  0.9983537251906386
F1 score:  0.9939227062641361
              precision    recall  f1-score   support

           0       0.92      0.63      0.75      2169
           1       0.99      1.00      0.99     75929

    accuracy                           0.99     78098
   macro avg       0.95      0.81      0.87     78098
weighted avg       0.99      0.99      0.99     78098
```

In contrast, Gaussian Naive Bayes was the worst performing model.

```
After tuning GaussianNB
Mean squared error: 0.023316858306230634
Accuracy score: 0.9766831416937694
Precision score:  0.976603297918864
Recall score:  0.9999736596030502
F1 score:  0.988150317227916
              precision    recall  f1-score   support

           0       0.99      0.16      0.28      2169
           1       0.98      1.00      0.99     75929

    accuracy                           0.98     78098
   macro avg       0.99      0.58      0.63     78098
weighted avg       0.98      0.98      0.97     78098
```

For datasets with Point 5, having missing values filled with mean also resulted in the best performance across all 3 models. However, Gaussian Naive Bayes was unable to classify 0 ('NG') and 1 ('OK') for MC2 with Point 5 filled with mean. Hence, with emphasis on this limitation (with respect to MC1 as our main dataset), having missing values filled with 0 was deemed more appropriate across both datasets with Point 5, with Random Forest Classifier also being the best performing model.

```
After tuning RandomForestClassifier
Mean squared error: 0.011703244641348049
Accuracy score: 0.9882967553586519
Precision score:  0.9900827094194661
Recall score:  0.9979586192363918
F1 score:  0.9940050635568206
              precision    recall  f1-score   support

           0       0.90      0.65      0.76      2169
           1       0.99      1.00      0.99     75929

    accuracy                           0.99     78098
   macro avg       0.95      0.82      0.87     78098
weighted avg       0.99      0.99      0.99     78098
```

In contrast, Gaussian Naive Bayes was also the worst performing model.

```
After tuning GaussianNB
Mean squared error: 0.024674127378421982
Accuracy score: 0.975325872621578
Precision score:  0.9753102278858259
Recall score:  0.9999341490076256
F1 score:  0.9874687042757274
              precision    recall  f1-score   support

           0       0.98      0.11      0.20      2169
           1       0.98      1.00      0.99     75929

    accuracy                           0.98     78098
   macro avg       0.98      0.56      0.60     78098
weighted avg       0.98      0.98      0.97     78098
```

6.0 Deployment

For deployment, the Random Forest Classifier model will be serialized and saved into a file using the Pickle operation. This file can later be loaded to deserialize the model to be used for predicting machine results of ‘OK’ and ‘NG’.

```
In [24]: from sklearn.preprocessing import LabelEncoder
         le = LabelEncoder()
         X = MC1Zero.drop('Machine Result', axis=1)
         y = le.fit_transform(MC1Zero['Machine Result'])

         from sklearn.model_selection import train_test_split
         import pickle
         Xtrain, Xtest, ytrain, ytest = train_test_split(X, y,random_state=1)

         from sklearn.ensemble import RandomForestClassifier
         rfc = RandomForestClassifier(n_estimators=51)
         rfc.fit(Xtrain, ytrain)

         # save the model to disk
         filename = 'deployed_model.sav'
         pickle.dump(deployed, open(filename, 'wb'))
```

```
In [25]: # load the model from disk
         deployed = pickle.load(open(filename, 'rb'))
```

The model will be available when the user logs in to the system. Since the performance between datasets without Point 5 and with Point 5 is relatively similar, the prediction model will take all 5 points as input values. Hence, the user can interact with the model by inputting values of (um)Point1, (um)Point2, (um)Point3, (um)Point4 and (um)Point5. Then, the model will print out the predicted Machine Result based on the inputted values.

1. Prompt the user for Point 1, Point 2, Point 3, Point 4 and Point 5 values. If any of the inputs have null values, they will be replaced with 0.

```
In [13]: def prediction (P1, P2, P3, P4, P5):
             if (P1==''):
                 P1=0
             if (P2==''):
                 P2=0
             if (P3==''):
                 P3=0
             if (P4==''):
                 P4=0
             if (P5==''):
                 P5=0

             prediction = deployed.predict([[P1,P2,P3,P4,P5]])

             if (prediction==[0]):
                 predicted='NG'
             else:
                 predicted='OK'

             return predicted
```

2. Fit the inputted values into the selected model, which is Random Forest Classifier with n_estimators=51.

```
In [14]: def main():

             P1=input("Enter value of Point 1(um): " )
             P2=input("Enter value of Point 2(um): " )
             P3=input("Enter value of Point 3(um): " )
             P4=input("Enter value of Point 4(um): " )
             P5=input("Enter value of Point 5(um): " )

             result=prediction(P1,P2,P3,P4,P5)
             print("Predicted Machine Result: " , result)
             return
```

3. Display the predicted result.

```
In [16]: main()

Enter value of Point 1(um): 34.3
Enter value of Point 2(um): 9.8
Enter value of Point 3(um): 349.6
Enter value of Point 4(um): 47.4
Enter value of Point 5(um): 25.7
Predicted Machine Result:  NG
```

7.0 Conclusion

In this project, in terms of handling missing values and modifying outliers, filling missing values with 0 was the best method, whereas filling missing values with median was the worst method. In terms of machine learning models, Random Forest Classifier was the best performing model, whereas Gaussian Naive Bayes performed the worst. This was most probably because Gaussian Naive Bayes assumes that the features were independent of each other, which they were clearly not. It was also evident that dropping Point 5 from the datasets did not jeopardize the performance of the models. By including all 5 points as input options, it gave more flexibility because any 5 points inputted by the user could produce predicted results of up to 99% accuracy.

This assignment was a good opportunity to practise what was learnt in the Data Science course in a real-life situation. It involved data compilation, data exploration, data pre-processing and data modelling with non-optimized datasets and involved quite a bit of experimenting in terms of models to be used. It is hoped that the deployment method of the selected machine learning model can be further enhanced in the form of an application for users to enter different point values to predict machine results.

8.0 References

1. n.d. (2017) *CAPABILITY.* HOTAYI Electronic. Available from: http://www.hotayi.com/capability [Accessed: 28 August 2021]
2. n.d. (n.d.) *Automatic optical inspection, AOI systems.* Electronics Notes. Available from: https://www.electronics-notes.com/articles/test-methods/automatic-automated-test-ate/aoi-optical-inspection.php [Accessed: 28 August 2021]
3. Hommer, n.d. (2021) *10 Common PCB Quality Problems You Should Know.* WellPCB. Available from: https://www.wellpcb.com/news/10-common-pcb-quality-problems.html [Accessed: 28 August 2021]
4. Marsja, E. (2020) *Pandas Count Occurrences in Column – i.e. Unique Values.* n.d. Available from: https://www.marsja.se/pandas-count-occurrences-in-column-unique-values/ [Accessed: 10 August 2021]
5. fuglede, n.d. (2017) *Pandas side-by-side stacked bar plot.* Stack Overflow. Available from: https://stackoverflow.com/questions/47494557/pandas-side-by-side-stacked-bar-plot [Accessed: 15 August 2021]
6. Seppänen, J.K. (2009) *Controlling bars width in matplotlib with per-month data.* Stack Overflow. Available from: https://stackoverflow.com/questions/886716/controlling-bars-width-in-matplotlib-with-per-month-data [Accessed: 15 August 2021]
7. Maku, n.d. (2019) *How to add table title in python preferably with pandas.* Stack Overflow. Available from: https://stackoverflow.com/questions/57958432/how-to-add-table-title-in-python-preferably-with-pandas [Accessed: 15 August 2021]
8. Solomon, B. (2018) *Correlation coefficient of two columns in pandas dataframe with .corr().* Stack Overflow. Available from: https://stackoverflow.com/questions/49350445/correlation-coefficient-of-two-columns-in-pandas-dataframe-with-corr [Accessed: 6 August 2021]
9. Sharma, N. (2018) *Ways to Detect and Remove the Outliers.* Towards Data Science. Available from: https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba [Accessed: 6 August 2021]
10. Saeed, H. (2021) *Python: How to Handle Missing Data in Pandas Dataframe.* Stack Abuse. Available from: https://stackabuse.com/python-how-to-handle-missing-dataframe-values-in-pandas/ [Accessed: 6 August 2021]
11. Rajput-Ji, n.d. (2021) *How to drop one or multiple columns in Pandas Dataframe.* GeeksforGeeks. Available from: https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/ [Accessed: 6 August 2021]
12. xiaoharper *et.al.* (2019) *Normalize Data.* Microsoft. Available from: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data [Accessed: 8 August 2021]
13. Naviani, A. (2018) *Understanding Random Forest Classifiers in Python.* Datacamp. Available from: https://www.datacamp.com/community/tutorials/random-forests-classifier-python [Accessed: 10 August 2021]

14. Kumar, A. (2020) *Accuracy, Precision, Recall and F1-Score - Python Examples.* Data Analytics. Available from: https://vitalflux.com/accuracy-precision-recall-f1-score-python-example/ [Accessed: 10 August 2021]
15. n.d. (2020) *sklearn.metrics.classification_report*. scikit-learn developers. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html [Accessed: 10 August 2021]
16. Shovalt, n.d. (2017) *UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.* Stack Overflow. Available from: https://stackoverflow.com/questions/43162506/undefinedmetricwarning-f-score-is-ill-defined-and-being-set-to-0-0-in-labels-wi [Accessed: 12 August 2021]
17. Brownlee, J. (2016) *Save and Load Machine Learning Models in Python with scikit-learn.* Machine Learning Mastery. Available from: https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/ [Accessed: 27 August 2021]
18. Prabakaran, S. (2018) *How Naive Bayes Algorithm Works? (with example and full code).* Machine Learning Plus. Available from: https://www.machinelearningplus.com/predictive-modeling/how-naive-bayes-algorithm-works-with-example-and-full-code/ [Accessed: 30 August 2021]

9.0 Appendices

STUDENT'S DECLARATION OF ORIGINALITY

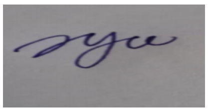
By submitting this assignment, I declare that this submitted work is free from all forms of plagiarism and for all intents and purposes is my own properly derived work. I understand that I have to bear the consequences if I fail to do so.

Assignment Submission

Course Code:	BACS3013
Course Title:	Data Science
Signature:	
Name of Student:	Nathan Ting Chow Yew
Student ID:	20WMR12222
Date:	04/09/2021

Course Code:	BACS3013
Course Title:	Data Science
Signature:	
Name of Student:	Khoo Chong Ee
Student ID:	20WMR12982
Date:	04/09/2021

Course Code:	BACS3013
Course Title:	Data Science
Signature:	
Name of Student:	Teo Shi Han
Student ID:	20WMR12044
Date:	04/09/2021

Course Code:	BACS3013
Course Title:	Data Science
Signature:	
Name of Student:	Yee Chooi Li
Student ID:	20WMR12498
Date:	04/09/2021