Project 3 - Manufacture - Enterprise Asset Maintenance

Rahul Gupta - DSC 680 - Winter 2020

Project Drafft - Week 11

https://github.com/rahulgupta271/DSC680_Project_3_Enterprise_Asset_Maintenance

1. Business Objective & Understanding

In the industrial sector, heavy machinery downtime loses a lot of dollars both in terms of idle time wasted due to maintenance work and repair costs as well. It would be a big boost to their bottom line if the corporations will be pro-active and conduct annual maintenance activities proactively along with forecasting problems beforehand using historical evidence.

Business Goal:

The overarching aim is to establish a constructive maintenance plan that aims to anticipate potential faults in heavy machines with multiple materials. It helps firms, as mentioned earlier, by reducing overhead costs, long-term maintenance costs and optimizing production hours.

Approach:

CRISP – DM Methodology has been used to perform this supervised learning task.

2. Data Understanding

For building this Predictive Maintenance Model, the following data sources were considered.

Telemetry : Time series data from separate devices comprising of different measures such as voltage, rotation, strain and vibration readings.

Machines : Information about machines.

Failures : Records of failed components.

Maintenance : Maintenance historical records of machines involving component replacements due to regular maintenance activity or due to failures.

Errors : Historical errors thrown by the machines.

Below links has instructions to get data sets for this project.
https://gallery.azure.ai/Experiment/Predictive-Maintenance-Implementation-Guide-Data-Sets-1

Feature Variables: In each data set, below is the list of variables.

Telemetry data set has below variables:

datetime

machineID

volt

rotate

pressure

Vibration

Machines:

machineID

model

age

Errors:

datetime

machineID

errorID

Failures:

datetime

machineID

failure

Maintenance:

datetime

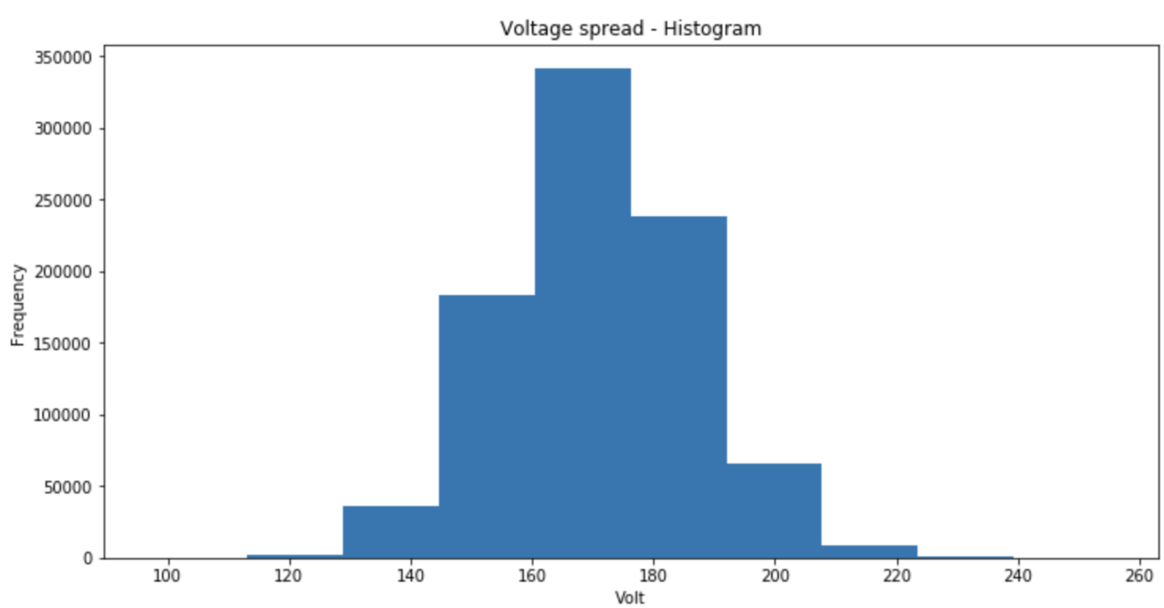
machineID

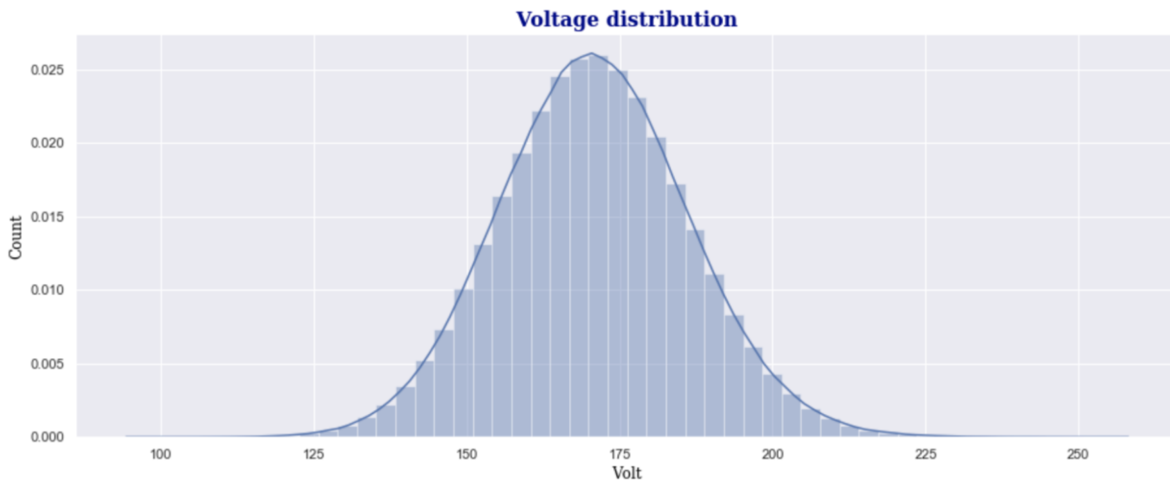
comp

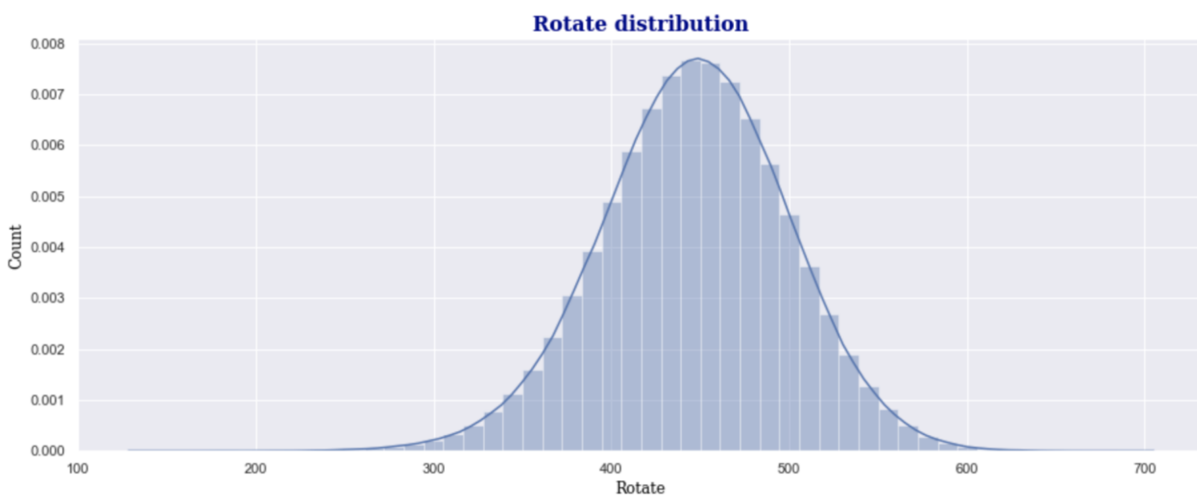
Exploratory Data Analysis:

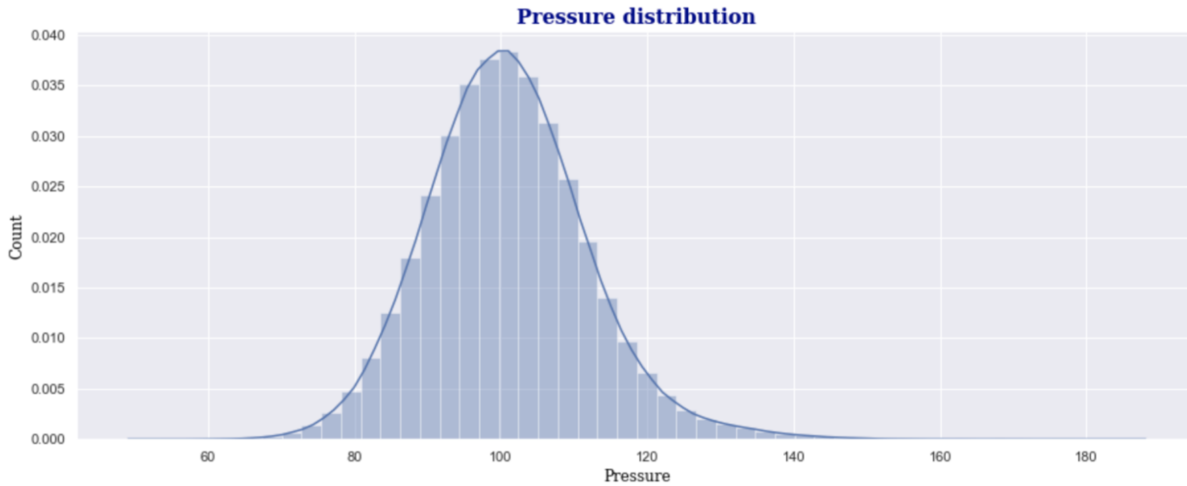
Firstly, in both of the data sets, I searched for missing values. Luckily, without any missed values, the data sets available are very clean.

So, let us visually explore the variables.

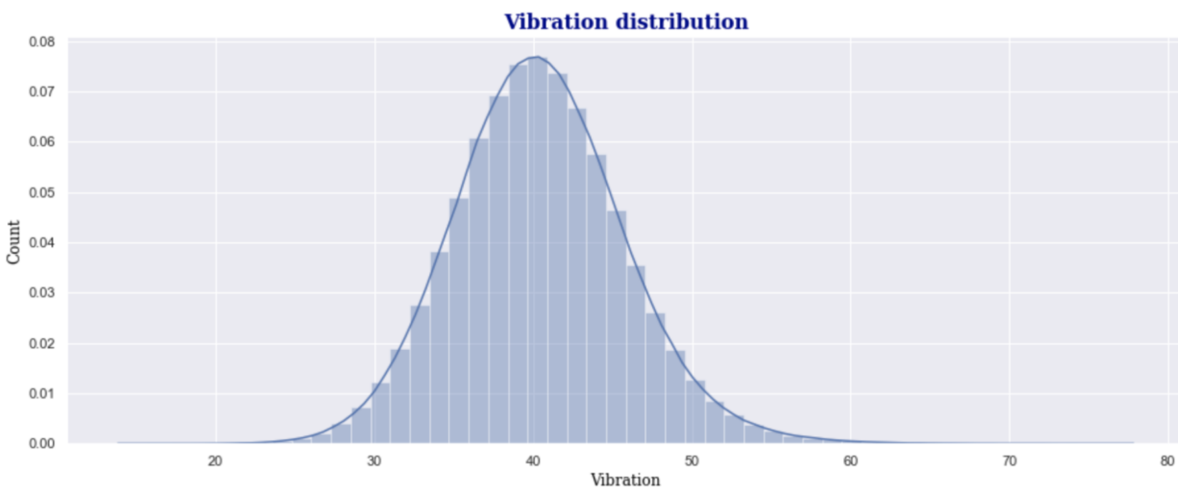



Voltage distribution
Count
0.025
0.020
0.015
0.010
0.005
0.000
100
125
150
175
200
225
250
Volt

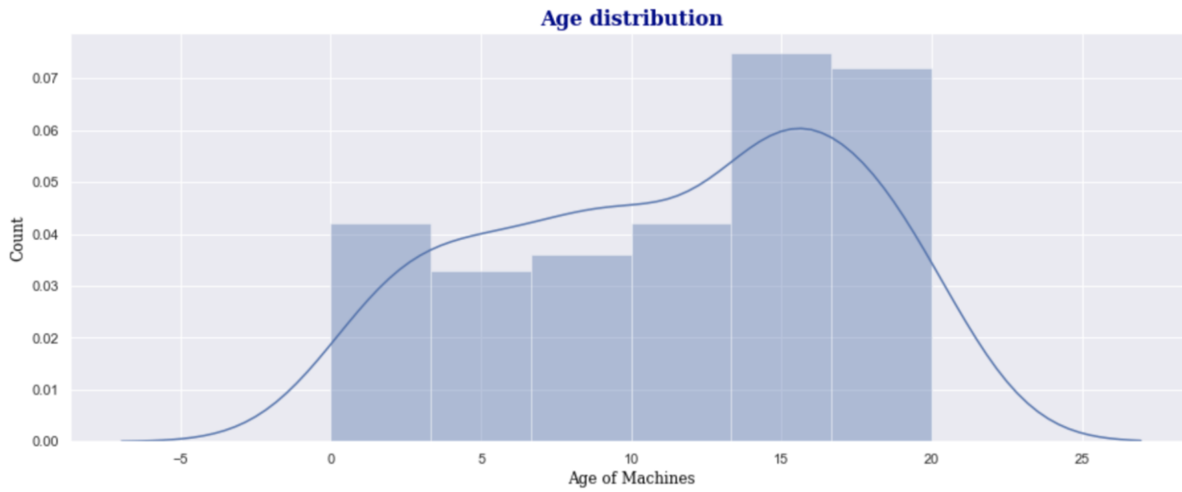

Rotate distribution
Count
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
100
200
300
400
500
600
700
Rotate

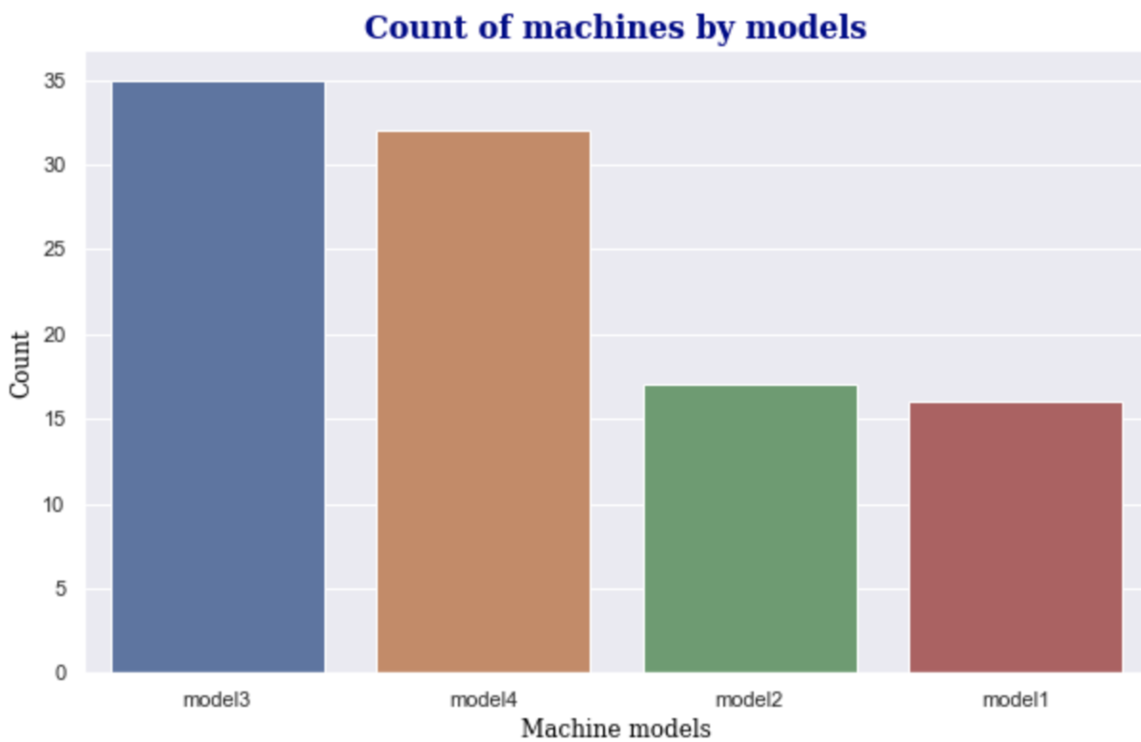


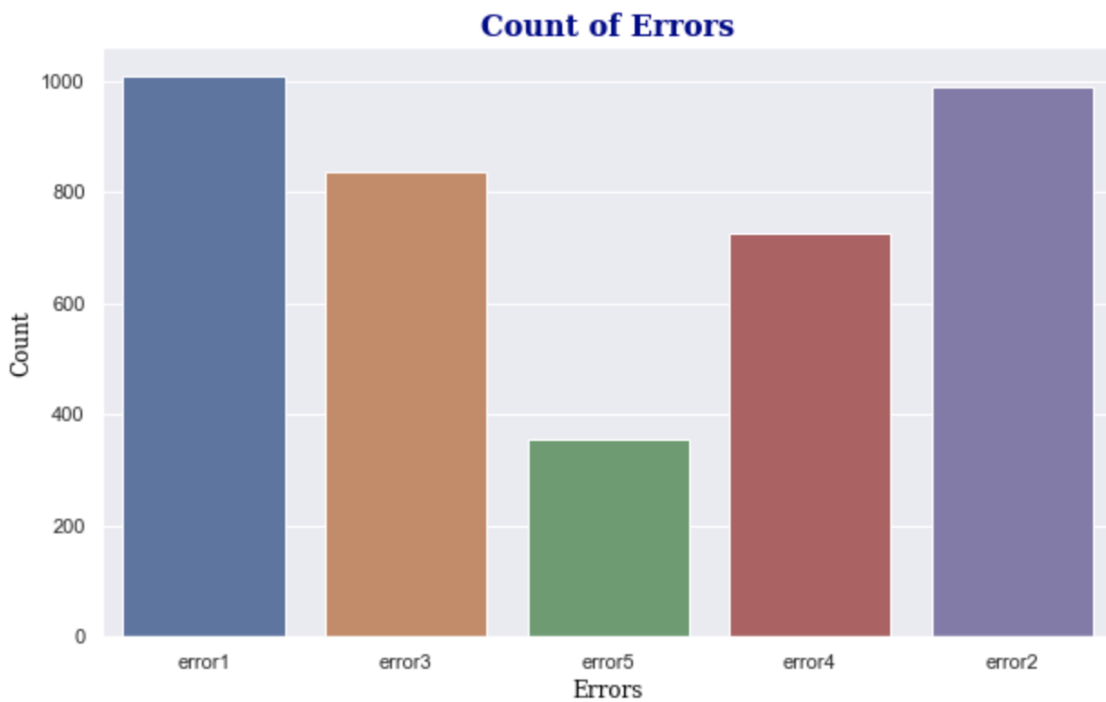
Pressure distribution
Count
Pressure
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
60
80
100
120
140
160
180

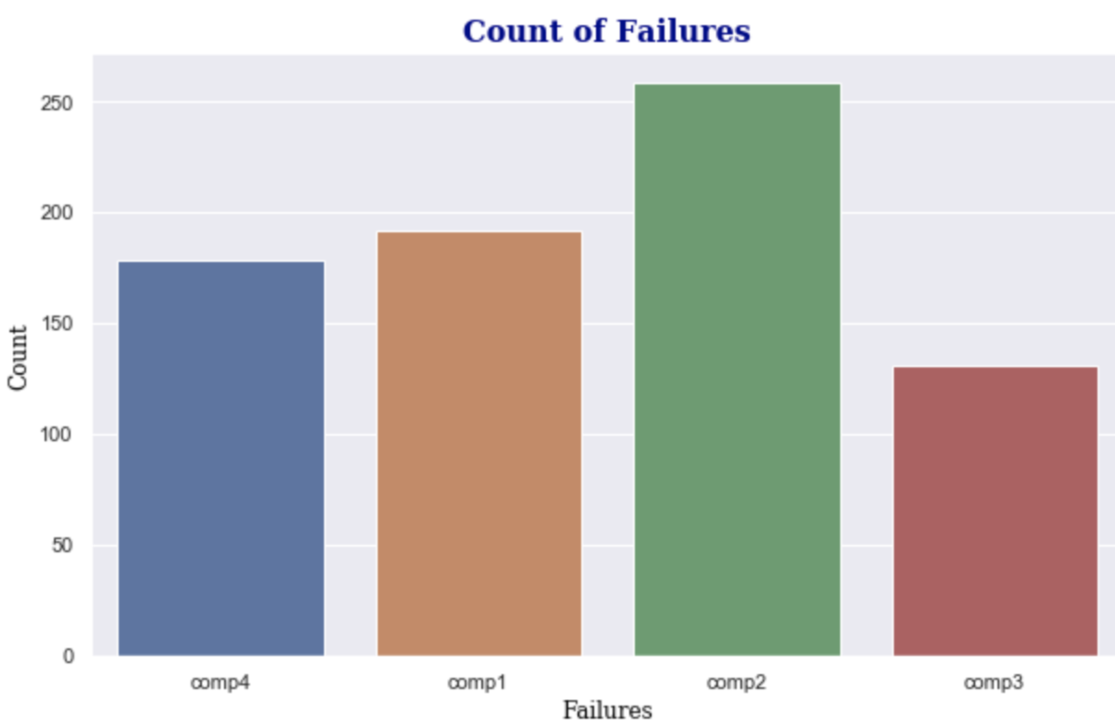


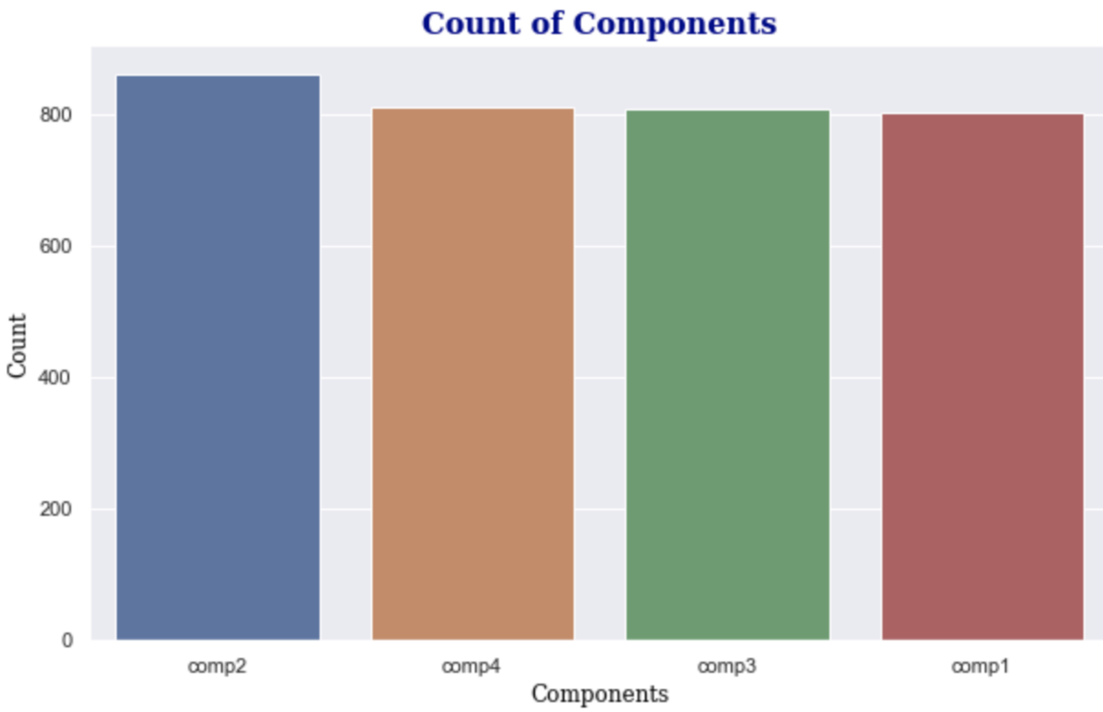
Vibration distribution
Count
Vibration
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
20
30
40
50
60
70
80


Age distribution
Count
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−5
0
5
10
15
20
25
Age of Machines


Count of machines by models
Count
35
30
25
20
15
10
5
0
model3
model4
model2
model1
Machine models


Count of Errors
Count
1000
800
600
400
200
0
error1
error3
error5
error4
error2
Errors


Count of Failures
Count
250
200
150
100
50
0
comp4
comp1
comp2
comp3
Failures



3. Data Preparation

<u>**Feature Engineering**</u>

```
# converting all date time fields into Date Time Format

errors_df['DT'] = pd.to_datetime(errors_df['datetime'])
failures_df['DT'] = pd.to_datetime(failures_df['datetime'])
maint_df['DT'] = pd.to_datetime(maint_df['datetime'])

```

Calculate mean and standard deviation of metrics for a rolling 24 hour windows

```
volt_mean_24 = pd.pivot_table(telemetry_df,
                              index='DT',
                              columns='machineID',
                              values='volt').rolling(window=24).mean().resample('3H',
                                                                  closed='left',
                                                                  label='right').first().unstack()

print(type(volt_mean_24))
```

```python
telemetry_24df.columns = ['volt_mean_24', 'volt_std_24', 'rotate_mean_24' ,'rotate_std_24','pressure_mean_24',
                          'pressure_std_24', 'vibration_mean_24', 'vibration_std_24']
telemetry_24df = telemetry_24df.reset_index()
telemetry_24df.head(10)
```

	machineID	DT	volt_mean_24	volt_std_24	rotate_mean_24	rotate_std_24	pressure_mean_24	pressure_std_24	vibration_mean_24	vibration_std_2
0	1	2015-01-01 09:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
1	1	2015-01-01 12:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2	1	2015-01-01 15:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	1	2015-01-01 18:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
4	1	2015-01-01 21:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
5	1	2015-01-02 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
6	1	2015-01-02 03:00:00	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
7	1	2015-01-02 06:00:00	169.733809	11.233120	445.179865	48.717395	96.797113	10.079880	40.385160	5.85320
8	1	2015-01-02 09:00:00	170.614862	12.519402	446.364859	48.385076	96.849785	10.171540	39.736826	6.16323

```python
telemetry_3df.columns = ['volt_mean_3', 'volt_std_3', 'rotate_mean_3' ,'rotate_std_3','pressure_mean_3',
                          'pressure_std_3', 'vibration_mean_3', 'vibration_std_3']
telemetry_3df = telemetry_3df.reset_index()
telemetry_3df.head(10)
```

machineID	DT	volt_mean_3	volt_std_3	rotate_mean_3	rotate_std_3	pressure_mean_3	pressure_std_3	vibration_mean_3
1	2015-01-01 09:00:00	170.028993	6.721032	449.533798	67.849599	94.592122	18.934956	40.893502
1	2015-01-01 12:00:00	165.443986	4.807415	425.415550	92.702671	93.315664	17.106476	39.571655
1	2015-01-01 15:00:00	162.223630	8.919370	454.923953	38.316408	106.523125	9.176711	34.799816
1	2015-01-01 18:00:00	172.355243	3.056496	423.041389	33.200513	105.491224	4.843754	40.288677
1	2015-01-01 21:00:00	160.226142	6.853823	440.413573	54.501054	95.424693	8.931082	41.776012
1	2015-01-02 00:00:00	178.513887	7.084050	440.859804	53.461091	90.827785	4.388335	44.042853
1	2015-01-02 03:00:00	167.989524	14.910036	481.173775	33.823675	94.103572	2.705111	37.755087
1	2015-01-02 06:00:00	165.002124	5.959072	456.438211	54.613403	87.673270	7.623486	41.527048
1	2015-01-02 09:00:00	193.164359	10.459846	448.652619	46.294659	101.438946	11.281152	38.133459
1	2015-01-02 12:00:00	159.676811	11.972868	430.571937	51.632304	100.242184	12.041639	37.664039

```python
errorcounts_df.columns = errorcounts_df.columns.get_level_values(0)
errorcounts_df.columns = ['machineID', 'DT', 'errorID', 'errorcount']
```

```python
errorcounts_ctdf = pd.crosstab([errorcounts_df.machineID, errorcounts_df.DT], errorcounts_df.errorID).reset_index()
errorcounts_ctdf = errorcounts_ctdf.sort_values('DT')
errorcounts_ctdf.head()
```

errorID	machineID	DT	error1	error2	error3	error4	error5
2874	81	2015-01-01 06:00:00	1	0	0	0	0
836	24	2015-01-01 06:00:00	1	0	0	0	0
2579	73	2015-01-01 06:00:00	0	0	0	1	0
1497	43	2015-01-01 07:00:00	0	0	1	0	0
2683	76	2015-01-01 08:00:00	0	0	0	0	1

```python
# summarize the errors for every 3 hours to includes errors occured in the last 24 hours

error1_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error1').rolling(window=24).sum().resample('3H',
                                                                                closed='left',
                                                                                label='right').first().unstack()

error2_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error2').rolling(window=24).sum().resample('3H',
                                                                                closed='left',
                                                                                label='right').first().unstack()

error3_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error3').rolling(window=24).sum().resample('3H',
                                                                                closed='left',
                                                                                label='right').first().unstack()

error4_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error4').rolling(window=24).sum().resample('3H',
                                                                                closed='left',
                                                                                label='right').first().unstack()

error5_count = pd.pivot_table(errorcounts_dtdf,
                              index='DT',
                              columns='machineID',
                              values='error5').rolling(window=24).sum().resample('3H',
                                                                                closed='left',
                                                                                label='right').first().unstack()
```

```
error_sum_df.columns = ['error1_count', 'error2_count', 'error3_count', 'error4_count', 'error5_count']
error_sum_df = error_sum_df.reset_index()
error_sum_df = error_sum_df.fillna(0)
error_sum_df.head(10)
```

	machineID	DT	error1_count	error2_count	error3_count	error4_count	error5_count
0	1	2015-01-01 09:00:00	0.0	0.0	0.0	0.0	0.0
1	1	2015-01-01 12:00:00	0.0	0.0	0.0	0.0	0.0
2	1	2015-01-01 15:00:00	0.0	0.0	0.0	0.0	0.0
3	1	2015-01-01 18:00:00	0.0	0.0	0.0	0.0	0.0
4	1	2015-01-01 21:00:00	0.0	0.0	0.0	0.0	0.0
5	1	2015-01-02 00:00:00	0.0	0.0	0.0	0.0	0.0
6	1	2015-01-02 03:00:00	0.0	0.0	0.0	0.0	0.0
7	1	2015-01-02 06:00:00	0.0	0.0	0.0	0.0	0.0
8	1	2015-01-02 09:00:00	0.0	0.0	0.0	0.0	0.0
9	1	2015-01-02 12:00:00	0.0	0.0	0.0	0.0	0.0

```
maint_df3 = pd.crosstab([maint_df2.machineID, maint_df2.datetime, maint_df2.DT], maint_df2.comp).reset_index()
maint_df3.head()
```

comp	machineID	datetime	DT	comp1	comp2	comp3	comp4
0	1	1/20/2015 6:00:00 AM	2015-01-20 06:00:00	1	0	1	0
1	1	1/5/2015 6:00:00 AM	2015-01-05 06:00:00	1	0	0	1
2	1	10/17/2015 6:00:00 AM	2015-10-17 06:00:00	0	1	0	1
3	1	10/2/2015 6:00:00 AM	2015-10-02 06:00:00	1	0	0	1
4	1	11/1/2015 6:00:00 AM	2015-11-01 06:00:00	0	1	0	1

```
maintcounts_dtdf_comp4 = maintcounts_dtdf[maintcounts_dtdf['comp4'] == 1.0].sort_values(['machineID', 'DT'])
maintcounts_dtdf_comp4['comp4rank'] = maintcounts_dtdf_comp4.groupby('machineID')['DT'].rank(ascending=True)
maintcounts_dtdf_comp4['comp4prevrank'] = maintcounts_dtdf_comp4['comp4rank'] + 1
maintcounts_dtdf_comp4 = maintcounts_dtdf_comp4.drop(['comp1', 'comp2', 'comp3'], axis = 1)

maintcounts_dtdf_comp4_df2 = maintcounts_dtdf_comp4


maintcounts_dtdf_comp4_lpdf = pd.merge(maintcounts_dtdf_comp4, maintcounts_dtdf_comp4_df2,
                                       left_on = ['machineID', 'comp4rank'], right_on =  ['machineID', 'comp4prevrank'],
                                       how = 'outer')

maintcounts_dtdf_comp4_lpdf = maintcounts_dtdf_comp4_lpdf.drop([ 'comp4prevrank_x','datetime_y', 'comp4_y', 'comp4rank_y
       'comp4prevrank_y'], axis = 1)


maintcounts_dtdf_comp4_lpdf.columns = ['machineID', 'DT', 'datetime_x', 'comp4_x', 'comp4rank_x', 'comp4Lastreplaceddt']

maintcounts_dtdf_comp4_lpdf['comp4Lastreplaceddt'] = maintcounts_dtdf_comp4_lpdf['comp4Lastreplaceddt']\
                                                            .fillna(method = 'bfill')
maintcounts_dtdf_comp4_lpdf.head(5)
```

	machineID	DT	datetime_x	comp4_x	comp4rank_x	comp4Lastreplaceddt
0	1	2014-07-16 06:00:00	7/16/2014 6:00:00 AM	1.0	1.0	2014-07-16 06:00:00
1	1	2015-01-05 06:00:00	1/5/2015 6:00:00 AM	1.0	2.0	2014-07-16 06:00:00
2	1	2015-02-04 06:00:00	2/4/2015 6:00:00 AM	1.0	3.0	2015-01-05 06:00:00
3	1	2015-06-19 06:00:00	6/19/2015 6:00:00 AM	1.0	4.0	2015-02-04 06:00:00
4	1	2015-09-02 06:00:00	9/2/2015 6:00:00 AM	1.0	5.0	2015-06-19 06:00:00

```
maint_summarydf['comp1_repgapdays'] = (maint_summarydf['DT'] - maint_summarydf['comp1Lastreplaceddt']).apply(lambda x: x

maint_summarydf['comp2_repgapdays'] = (maint_summarydf['DT'] - maint_summarydf['comp2Lastreplaceddt']).apply(lambda x: x
maint_summarydf['comp3_repgapdays'] = (maint_summarydf['DT'] - maint_summarydf['comp3Lastreplaceddt']).apply(lambda x: x
maint_summarydf['comp4_repgapdays'] = (maint_summarydf['DT'] - maint_summarydf['comp4Lastreplaceddt']).apply(lambda x: x

maint_summarydf.head(15)
```

	machineID	DT	datetime	comp1	comp2	comp3	comp4	comp1Lastreplaceddt	comp2Lastreplaceddt	comp3Lastreplaceddt	comp4Lastreplacedd
0	1	2015-01-01 06:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
1	1	2015-01-01 07:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
2	1	2015-01-01 08:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
3	1	2015-01-01 09:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
4	1	2015-01-01 10:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
5	1	2015-01-01 11:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
6	1	2015-01-01 12:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0
7	1	2015-01-01 13:00:00	0	0.0	0.0	0.0	0.0	2014-12-13 06:00:00	2014-06-01 06:00:00	2014-07-31 06:00:00	2014-07-16 06:00:0

Merging all datasets

```
Alldata = pd.DataFrame()

Alldata = pd.merge(telemetry_summarydf, error_sum_df, on = ['machineID', 'DT'], how = 'left')

Alldata = pd.merge(Alldata, maint_sum_df, on = ['machineID', 'DT'], how = 'left')
```

```
# merging machine meta data

Alldata = pd.merge(Alldata, machines_df, on = ['machineID'], how = 'left')
Alldata.head(15)
```

Merging Failure data set

```
# merging machine meta data

Alldata = pd.merge(Alldata, failures_df, on = ['machineID', 'DT'], how = 'left')

#Alldata.drop([], axis = 1)
Alldata.head(15)
```

4. Predictive Modelling

Model Training

Random data splitting for training and data set testing does not make sense when faults and anomalies are time series-based incidents. So, I have divided the train data and test data depending on the dates for this exercise.

```
# Create random forest classifier object
randomforest = RandomForestClassifier(random_state=1,          # for consistent results
                                      n_estimators = 100,      # number of tress in forest
                                      oob_score=True,          # OOB Score to get performance
                                      bootstrap=True,
                                      n_jobs=-1,               # for using all cores
                                      class_weight="balanced"  # for handling imbalanced classes
                                     )
```

Train and predict using the model, storing results for later

```
# Train model
model = randomforest.fit(X_train, y_train)
#models.append(model)

#Predicting the target variable - class
y_predfailure = model.predict(X_test)
#y_predfailure_results.append(y_predfailure)

# Get predicted probabilities
y_prob_failure = model.predict_proba(X_test)[:,1]
#y_probfailure_results.append(y_prob_failure)
```

5. Evaluation

The most important parameter for testing the model is recall in preventive maintenance estimation, which conveys the real number of failures expected by the model. Designed into the model here. That's around 99.8 percent. This may be attributed to a substantial portion of loss = 'no', I suspect. I am confident, with the aid of domain experts, the model could be further modified to nullify this prejudice.

```
def fn_multiclass_metrics(actual_label, predicted_label):
    """
    function that takes acutal labels and predicted labels and returns
    accuracy, auc, precision, recall and f1 scores
    average = 'weighted' for multi class classification
    """
    accuracy = accuracy_score(actual_label, predicted_label)
    precision = precision_score(actual_label, predicted_label, average = 'weighted')
    recall = recall_score(actual_label, predicted_label, average = 'weighted')
    f1 = f1_score(actual_label, predicted_label, average = 'weighted')

    return (accuracy, precision, recall, f1)
```

```
acc, prec, recall, f1 = fn_multiclass_metrics(y_test, y_pred)

acc, prec, recall, f1

```

```
(0.9989716945391568, 0.9989577896573897, 0.9989716945391568, 0.998958276507185)
```

6. Model Tuning

While I have accuracy and recall above 99 percent, the model is slightly skewed, I guess. Considering the technicalities in the generation of different feature variables in preventive maintenance, the model can be further improved. To further enhance the model, various types of classifying algorithms such as Gradient Boosting Classifier or Deep Neural Networks can be used.

7. Deployment

```
cwd = os.getcwd()
print(cwd)

projdir = os.path.dirname(cwd)
modeldir = os.path.join(projdir, 'Model')

# importing telemetry data

modelfile = os.path.join(modeldir, 'predictivemodel.pkl')


```

```
C:\Users\nrrvlkp\Documents\M\680\DSC680\DSC680-Projects\Predictive Maintenance\Code
```

```
# Save the model as a pickle in a file
joblib.dump(model, modelfile)
```

```
['C:\\Users\\nrrvlkp\\Documents\\M\\680\\DSC680\\DSC680-Projects\\Predictive Maintenance\\Model\\predictivemodel.pkl']
```

```
# Load the model from the file
tunedmodel_from_joblib = joblib.load(modelfile)

```

```
# Fitting deployed model on new data ( assume here X_train and y_train are new unseen features and targets)
deployed_model = tunedmodel_from_joblib.fit(X_train, y_train)
```

8. Conclusion

You can obtain the code for this project from my Git Hub repository.

https://github.com/rahulgupta271/DSC680_Project_3_Enterprise_Asset_Maintenance

Please visit my GITHUB Portfolio to look at my other projects.

https://rahulgupta271.github.io/

9. Assumptions

NA

10. Techniques

Used the below modules in python to accomplish this project

- pandas
- numpy
- sklearn
- matplotlib
- seaborn
- joblib
- os

11. References

1. https://gallery.azure.ai/Experiment/Predictive-MaintenanceImplementation-Guide-Data-Sets-1

2. https://limblecmms.com/blog/predictive-maintenance/3. Jake Huneycutt, May 18 2018, "Implementing a Random Forest Classification Model in Python",

3.https://docs.oracle.com/cd/E39583_01/fscm92pbr0/eng/fscm/fwkm/task_PerformingAnAssetMaintenanceCostAnalysis-677feb.html

4. cognizant.com/whitepapers/using-predictive-analytics-to-optimizeasset-maintenance-in-the-utilities-industry-codex964.pdf

5. https://www.g3pconsulting.com/en/reliability-maintenancemanagement/asset-criticality-analysis