Detect Product Faults with a Smart Quality System

In this process we will follow the mentioned guidelines mentioned in the **use case understanding and planning**:

- Analytics Objective: To predict wheather the ball bearings should be replaced or not. For this we will try to generate features by statsitical understanding and then use this features to map decision space for classification tasks.
- Output of the model: Prediction by binary values '0' or '1'.

Understand the data

In [10]:

```
# import libraries
import numpy as np      # numpy for numerical analysis
import pandas as pd     # for data wrangling and cleaning
import os               #. system files access
import glob             #. for finding files in folder along with 'os' package

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

In [3]:

```
# read the quality csy
df_product_quality = pd.read_csv('product_quality_log.csv').iloc[:,1:]

# get a glimpse of it
df_product_quality.head()
```

Out[3]:

	machine_id	product_id	quality
0	Printer F0815	P3.2.500	OK
1	Printer F0815	P3.2.501	OK
2	Printer F0815	P3.2.502	OK
3	Printer F0815	P3.2.503	OK
4	Printer F0815	P3.2.504	OK

In [4]:

```
# similarly read the production csv
df_production = pd.read_csv('production_log.csv').iloc[:,1:]
df_production.head()
```

Out[4]:

	timestamp	product_id
0	2021-05-17_08-12-48	P3.2.500
1	2021-05-17_08-12-51	P3.2.501
2	2021-05-17_08-12-54	P3.2.502
3	2021-05-17_08-12-57	P3.2.503

	timestamp	product_id
4	2021-05-17_08-13-00	P3.2.504

In [5]:
```
print(df_product_quality.shape)
print(df_production.shape)
```

```
(1656, 3)
(1656, 2)
```

In [6]:
```
## lets merge these two colus
merged_df = pd.merge(df_product_quality,df_production, on='product_id')   # primary
merged_df.tail()
```

Out[6]:

	machine_id	product_id	quality	timestamp
1651	Printer F0815	P3.2.2151	nOK	2021-05-17_09-35-21
1652	Printer F0815	P3.2.2152	nOK	2021-05-17_09-35-24
1653	Printer F0815	P3.2.2153	nOK	2021-05-17_09-35-27
1654	Printer F0815	P3.2.2154	nOK	2021-05-17_09-35-30
1655	Printer F0815	P3.2.2155	nOK	2021-05-17_09-35-33

Generating features

> For statistical modelling we need features which needs to be generated by analysing the data. The features which we will consider over here are:

Stats features

1. **Mean** value of sensor 1 and sensor 2
2. **Median** value of sensor 1 and sensor 2
3. **Standard Deviation** value of sensor 1 and sensor 2
4. **Min and max** value of sensor 1 and sensor 2

In [7]:
```
# getting all sensor data fropm files
get_files =[f for f in glob.glob("vibrationdata/*")]
```

In [8]:
```
def compute_stats_features(df, array):
    '''
    We will calculate statistical features of the data from both the sensors

    '''
    df['sensor_data_1_mean'] = np.mean(np.array(lines).astype(np.float),axis=0)[0]
    df['sensor_data_2_mean'] = np.mean(np.array(lines).astype(np.float),axis=0)[1]

    df['sensor_data_1_median'] = np.median(np.array(lines).astype(np.float),axis=0)[
    df['sensor_data_2_median'] = np.median(np.array(lines).astype(np.float),axis=0)[

    df['sensor_data_1_stdev'] =  np.std(np.array(lines).astype(np.float),axis=0)[0]
    df['sensor_data_2_stdev'] =  np.std(np.array(lines).astype(np.float),axis=0)[1]

    df['sensor_data_1_min'] =    np.min(np.array(lines).astype(np.float),axis=0)[0]
```

```
    df['sensor_data_2_min'] =    np.min(np.array(lines).astype(np.float),axis=0)[1]

    df['sensor_data_1_max'] =    np.max(np.array(lines).astype(np.float),axis=0)[0]
    df['sensor_data_2_max'] =    np.max(np.array(lines).astype(np.float),axis=0)[1]


    return df
```

In [11]:

```
%%time
d_stats = pd.DataFrame()  # append in empty df
i = 0
for f in get_files:
    with open(f) as file:
        lines = [line.rstrip('\n').replace('\t',',').split(',') for line in file]
        d_stats = d_stats.append(compute_stats_features(merged_df[merged_df['timesta
        i = i + 10
        if i % 10 == 0:
            print('Done processing for {} files'.format{i})
```

```
CPU times: user 11min 38s, sys: 22.2 s, total: 12min
Wall time: 12min 24s
```

In [12]:

```
# we get the features now
d_stats.head()
```

Out[12]:

	machine_id	product_id	quality	timestamp	sensor_data_1_mean	sensor_data_2_mean	sensor_
1655	Printer F0815	P3.2.2155	nOK	2021-05-17_09-35-33	-0.118210	-0.118192	
653	Printer F0815	P3.2.1153	OK	2021-05-17_08-45-27	-0.116194	-0.116286	
120	Printer F0815	P3.2.620	OK	2021-05-17_08-18-48	-0.117007	-0.116786	
84	Printer F0815	P3.2.584	OK	2021-05-17_08-17-00	-0.117463	-0.117262	
594	Printer F0815	P3.2.1094	OK	2021-05-17_08-42-30	-0.117361	-0.117486	

Time features

> We will now look at time interval during which these data was generated. We will use **rolling mean** function over the time window of **5 minutes** to compute sensor mean values.

In [13]:

```
# get time and date and sort it
d_stats['date'] = [i.split('_')[0] for i in d_stats['timestamp']]
d_stats['time'] = [i.split('_')[1].replace('-',':') for i in d_stats['timestamp']]
```

```
d_stats['timestamp'] = pd.to_datetime(d_stats['date'] + ' ' + d_stats['time'])

# drop and sort time values
d_stats.drop(columns=['machine_id', 'date', 'time'], inplace=True)
d_stats = d_stats.sort_values(by=['timestamp']).reset_index(drop=True)

d_stats.set_index('timestamp', inplace=True)
```

In [14]:

```
# get the rolling mean value
d_stats['rollingmeanVal_1'] = d_stats.rolling('5T').sensor_data_1_mean.mean()
d_stats['rollingmeanVal_2'] = d_stats.rolling('5T').sensor_data_2_mean.mean()
```

Visualisation

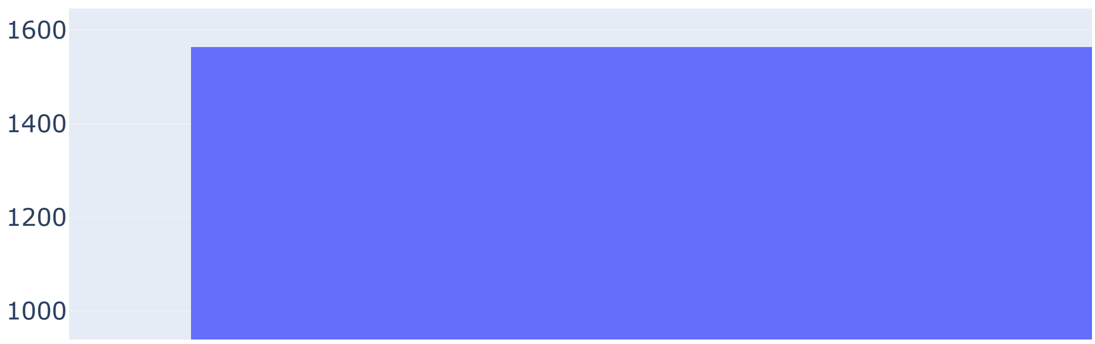
In [ ]:

```
import plotly.express as px # viz package
```

Histogram of quality

In [28]:

```
# Here we use a column with categorical data
fig = px.histogram(d_stats, x="quality")
fig.show()
```

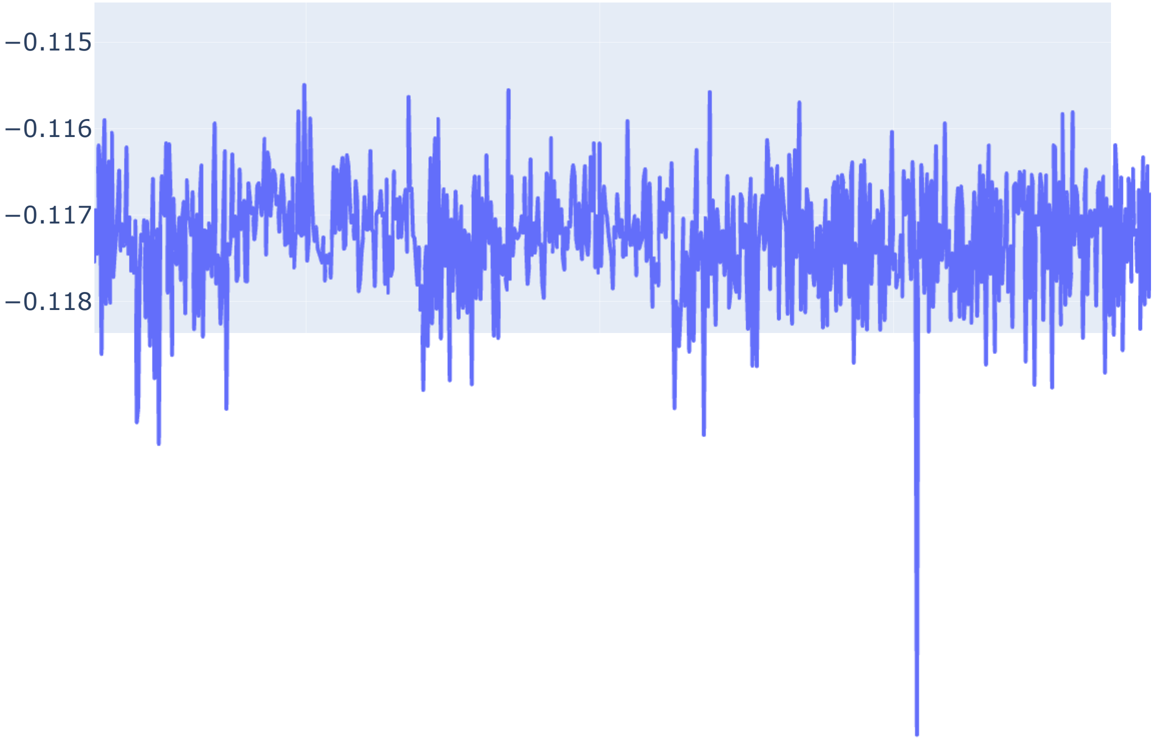


Observation: **

Mean plot

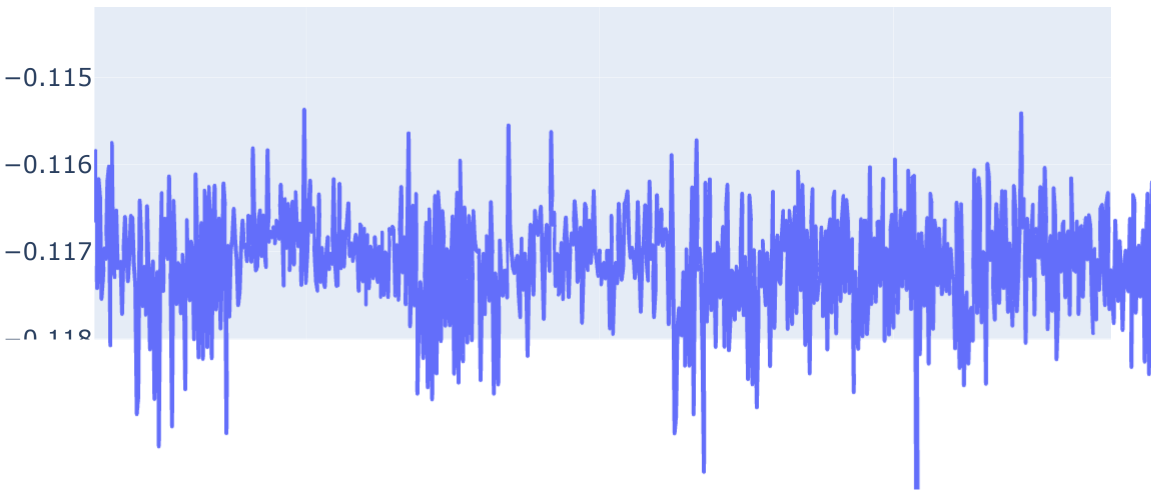
In [34]:

```
fig = px.line(y=d_stats['sensor_data_1_mean'], x=d_stats.index)
fig.show()
```



In [35]:

```
fig = px.line(y=d_stats['sensor_data_2_mean'], x=d_stats.index)
fig.show()
```



Observation: **

Time and Quality

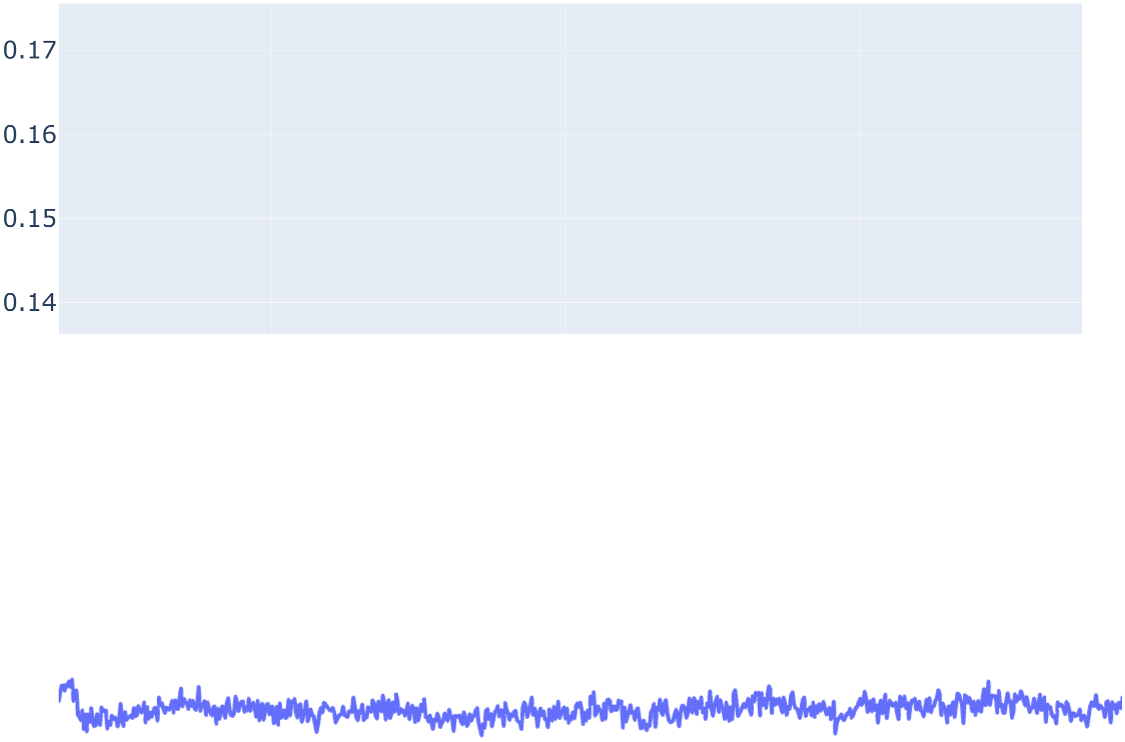
In [42]:

```
fig = px.scatter(y=d_stats['quality'], x=d_stats.index)
fig.show()
```

Standard deviation

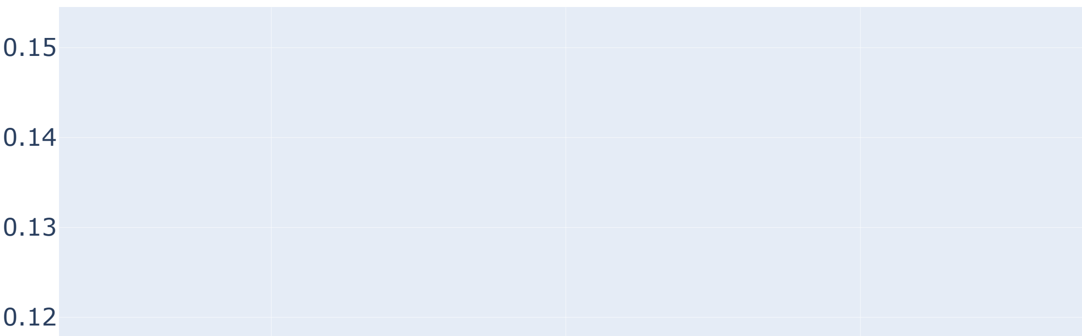
In [45]:

```
fig = px.line(y=d_stats['sensor_data_1_stdev'], x=d_stats.index)
fig.show()
```



In [46]:

```
fig = px.line(y=d_stats['sensor_data_2_stdev'], x=d_stats.index)
fig.show()
```



Observation: **

Modelling

> For predicting we are simply doing a classification task with the help of ML algorithms. For this purpose we will use 3 types of algorithms mainly:

1. **Logistic Regression**
2. **Decision Trees**
3. **Random Forest**

In [17]:
```
# binary encode quality for classification
d_stats['quality_code'] = d_stats['quality'].map({'OK':1, 'nOK':0})    # binary enco
```

In [19]:
```
# get the values
X = d_stats.drop(columns = ['product_id','quality','quality_code']).values
y = d_stats['quality_code'].values

print(X.shape, y.shape)
```

```
(1656, 12) (1656,)
```

In [22]:
```
from sklearn.model_selection import train_test_split      # for splitting the data
from sklearn import linear_model                          # for logistic regression
from sklearn.ensemble import RandomForestClassifier       # for Random Forest
from sklearn.tree import DecisionTreeClassifier           # for Decision tree
from sklearn.model_selection import GridSearchCV          # for searching in hyperpa
from sklearn.metrics import f1_score, precision_score, confusion_matrix, recall_scor

class classification:

  def __init__(self, X, y):

       self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y,


  def calc_metrics_class(self):

        precision = precision_score(self.pred, self.y_test)
        recall = recall_score(self.pred,self.y_test)
        f1 = f1_score(self.pred,self.y_test)
        accuracy = accuracy_score(self.pred,self.y_test)
        print("precision", precision, '\n', "recall", recall, '\n', "f1", f1, '\n',
```

```python
    def logistic_regression(self):


        # Create logistic regression
        print("Performing modelling for Logistic Regression")
        logistic = linear_model.LogisticRegression(max_iter = 1000)

        # Create regularization penalty space
        param_grid = {
                      'penalty' : ['l1', 'l2'],
                      }

        # Create hyperparameter options and fot it into grid search
        grid_model = GridSearchCV(estimator=logistic, param_grid=param_grid, cv=5, ver

        # Fit the model and find best hyperparams
        grid_model.fit(self.X_train,self.y_train)
        print("Best parameters =", grid_model.best_params_)

        # Fit the model with best params
        model_clf = logistic.set_params(**grid_model.best_params_)
        model_clf.fit(self.X_train, self.y_train)
        self.pred = model_clf.predict(self.X_test)
        self.calc_metrics_class()

    def random_forest(self):
        print("Performing modelling for Random forest")
        rf_model = RandomForestClassifier(random_state=1)
        param_grid = {
                      'n_estimators': [50],
                      'max_features': [0.9],
                      'min_samples_split': [3]
                      }
        grid_model = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_j
        grid_model.fit(self.X_train,self.y_train)
        print("Best parameters =", grid_model.best_params_)
        model_clf = rf_model.set_params(**grid_model.best_params_)
        model_clf.fit(self.X_train, self.y_train)
        self.pred = model_clf.predict(self.X_test)
        self.calc_metrics_class()

    def decision_tree(self):
        print("Performing modelling for decision tree")
        #create a dictionary of all values we want to test
        param_grid = { 'criterion':['gini'],'max_depth': [20]}
        # decision tree model
        dtree_model=DecisionTreeClassifier()
        #use gridsearch to test all values
        grid_model = GridSearchCV(dtree_model, param_grid, cv=5, n_jobs=-1)
        grid_model.fit(self.X_train,self.y_train)
        print("Best parameters =", grid_model.best_params_)
        model_clf = dtree_model.set_params(**grid_model.best_params_)
        model_clf.fit(self.X_train, self.y_train)
        self.pred = model_clf.predict(self.X_test)
        self.calc_metrics_class()
```

In [23]:

```python
c = classification(X,y)
c.logistic_regression()
```

```
Performing modelling for Logistic Regression
Best parameters = {'penalty': 'l2'}
precision 1.0
```

```
 recall 0.9427710843373494
 f1 0.9705426356589147
 accuracy 0.9427710843373494
```

In [24]:

```
c.decision_tree()
```

```
Performing modelling for decision tree
Best parameters = {'criterion': 'gini', 'max_depth': 20}
precision 0.9840255591054313
 recall 0.9655172413793104
 f1 0.9746835443037976
 accuracy 0.9518072289156626
```

In [25]:

```
c.random_forest()
```

```
Performing modelling for Random forest
Best parameters = {'max_features': 0.9, 'min_samples_split': 3, 'n_estimators': 50}
precision 0.9840255591054313
 recall 0.9716088328075709
 f1 0.9777777777777777
 accuracy 0.9578313253012049
```

In [ ]:

```

```