In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score,precision_score,recall_score,roc_auc_score,precis
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense,Dropout
import keras
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
import datetime
import calendar
from scipy.stats import chi2_contingency
```

In [2]:
```
def load_data():
    """
    load_data - function loads the data from into dataframe.
                Also datetime column is split into day, month and day of the week colu
                if any of these columns have any relation with response variable i.e.

    returns -  dataframe after loading the data
    """
    path = os.getcwd()
    device_data = pd.read_csv(path+"\\Data\\device_failure.csv",encoding='ISO-8859-1')


    device_data["date"] = pd.to_datetime(device_data["date"])
    device_data["day"] = device_data["date"].apply(lambda x:x.day)
    device_data["month"] = device_data["date"].apply(lambda x:x.month)
    device_data["day_of_week"]=device_data["date"].apply(lambda x: calendar.day_name[x

    return device_data
```

In [143…
```
device_data = load_data()
```

Below plot tells you the data is highly imbalance. Also there are no missing values.

In [4]:
```
# device_data.describe()
device_data.isnull().sum()
```

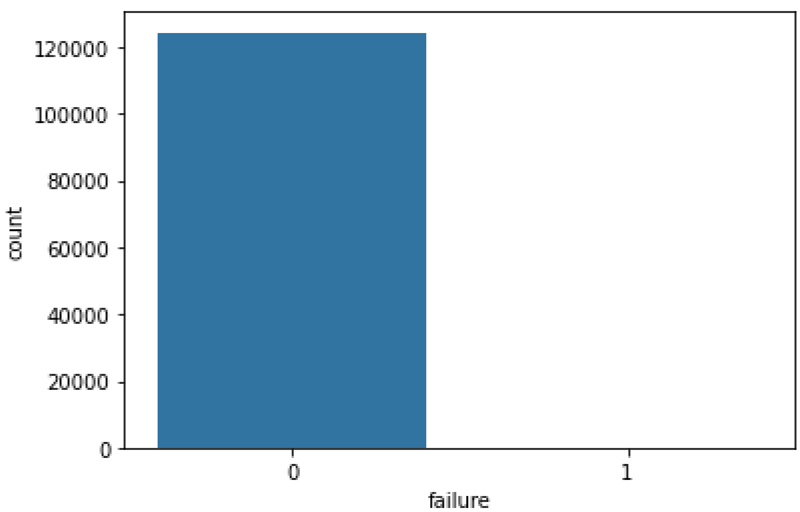
Out[4]:
```
date            0
device          0
failure         0
attribute1      0
attribute2      0
attribute3      0
attribute4      0
attribute5      0
attribute6      0
attribute7      0
attribute8      0
attribute9      0
day             0
month           0
day_of_week     0
dtype: int64
```

Exploratory analysis

In [5]:
```
sns.countplot(device_data["failure"])
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit k
eyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[5]:
```
<AxesSubplot:xlabel='failure', ylabel='count'>
```



In [6]:
```
print(device_data["failure"].count())
print(device_data[device_data["failure"]==1]["failure"].count())
```

```
124494
106
```

As shown above failure data points are 0.084% of the total data

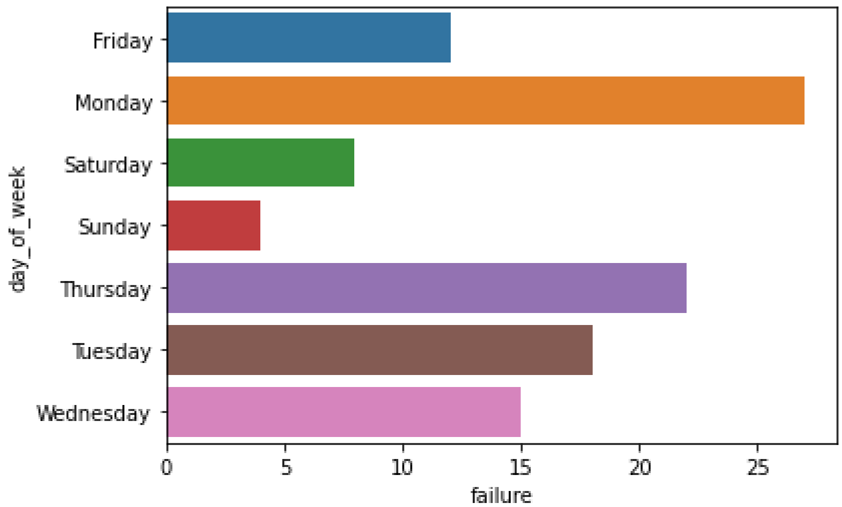
Below, Checking the relationship between respose variable "feature" and other categorical varibles like day_of_week ,month and day of month

In [7]:
```
df_fail=device_data[device_data["failure"]==1]
df_fail = df_fail.groupby(["day_of_week"])["failure"].count().reset_index()
```

```
sns.barplot(df_fail["failure"],df_fail["day_of_week"])

# df_pass=device_data[device_data["failure"]==0]
# sns.barplot(device_data["day_of_week"],device_data["failure"])
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```

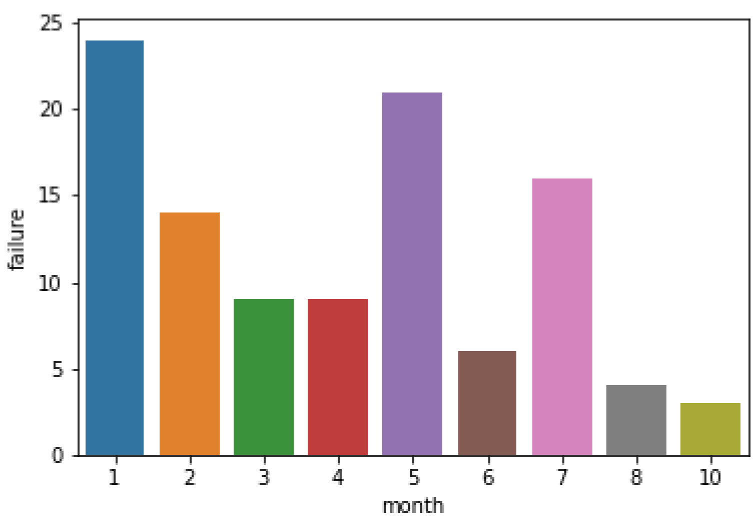
Out[7]: <AxesSubplot:xlabel='failure', ylabel='day_of_week'>



In [8]:
```
df_fail=device_data[device_data["failure"]==1]
df_fail = df_fail.groupby(["month"])["failure"].count().reset_index()
sns.barplot(df_fail["month"],df_fail["failure"])
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[8]: <AxesSubplot:xlabel='month', ylabel='failure'>

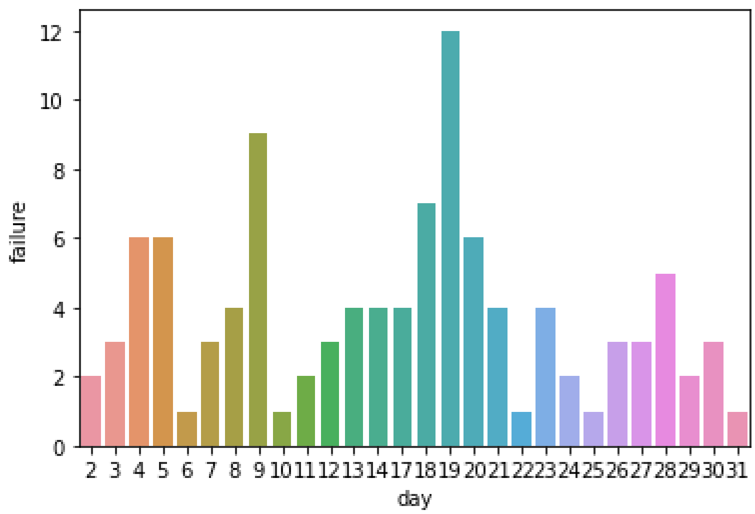


In [9]:
```
df_fail=device_data[device_data["failure"]==1]
df_fail = df_fail.groupby(["day"])["failure"].count().reset_index()
```

```
sns.barplot(df_fail["day"],df_fail["failure"])
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only va
lid positional argument will be `data`, and passing other arguments without an explic
it keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[9]:
```
<AxesSubplot:xlabel='day', ylabel='failure'>
```



Statistical tests for variables day,month,day_of_week

In [14]:
```
def chi_sqr_p_value(df,col1,col2):
    """
    chi_sqr_p_value - This function calculates chi_square value between 2 categorical
    parameters:
                df - input dataframe
                col1 - 1st categorical variable
                col2 - 2nd categorical variable

    returns -  p-value using chi square method indicating the relation between 2 categ
            if p-value > 0.05 Null hypothesis is rejected
               p-value < 0.05 Null hypothesis will be failed to reject.
    """

    dataset_table = pd.crosstab(device_data[col1],device_data[col2])
    observed_values = dataset_table.values
    p_val = chi2_contingency(dataset_table)[1]
    return p_val
```

In [11]:
```
print(chi_sqr_p_value(device_data,"failure","day_of_week"))
print(chi_sqr_p_value(device_data,"failure","month"))
print(chi_sqr_p_value(device_data,"failure","day"))
```

```
0.0003450624767265159
0.0009510068724008692
0.0006876251055513708
```

Chi square states

Null hypothesis - two categorical variables are independent in the given population.

Alternate hypothesis - Two categorical variables are not dependent by chance.There is a association.

Since the p-value of all the above variables is less than 0.05, we fail to reject Null hypothesis.

Above graphs and chi-square statistical tests fail to confirm relation of these variables with response variable "failure"

In [148...

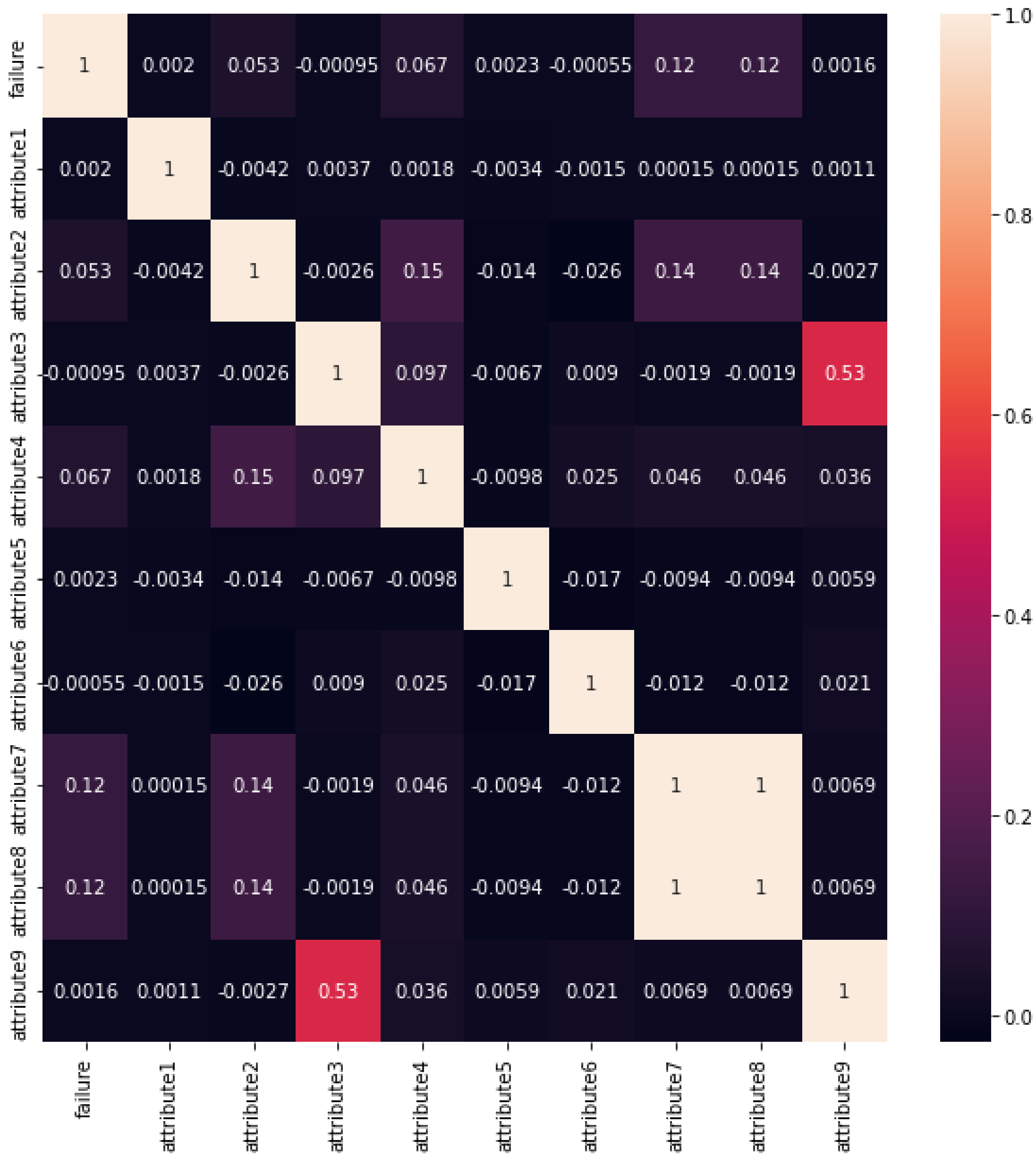
```
device_data.drop(["month","day_of_week","day"],axis=1,inplace=True)
```

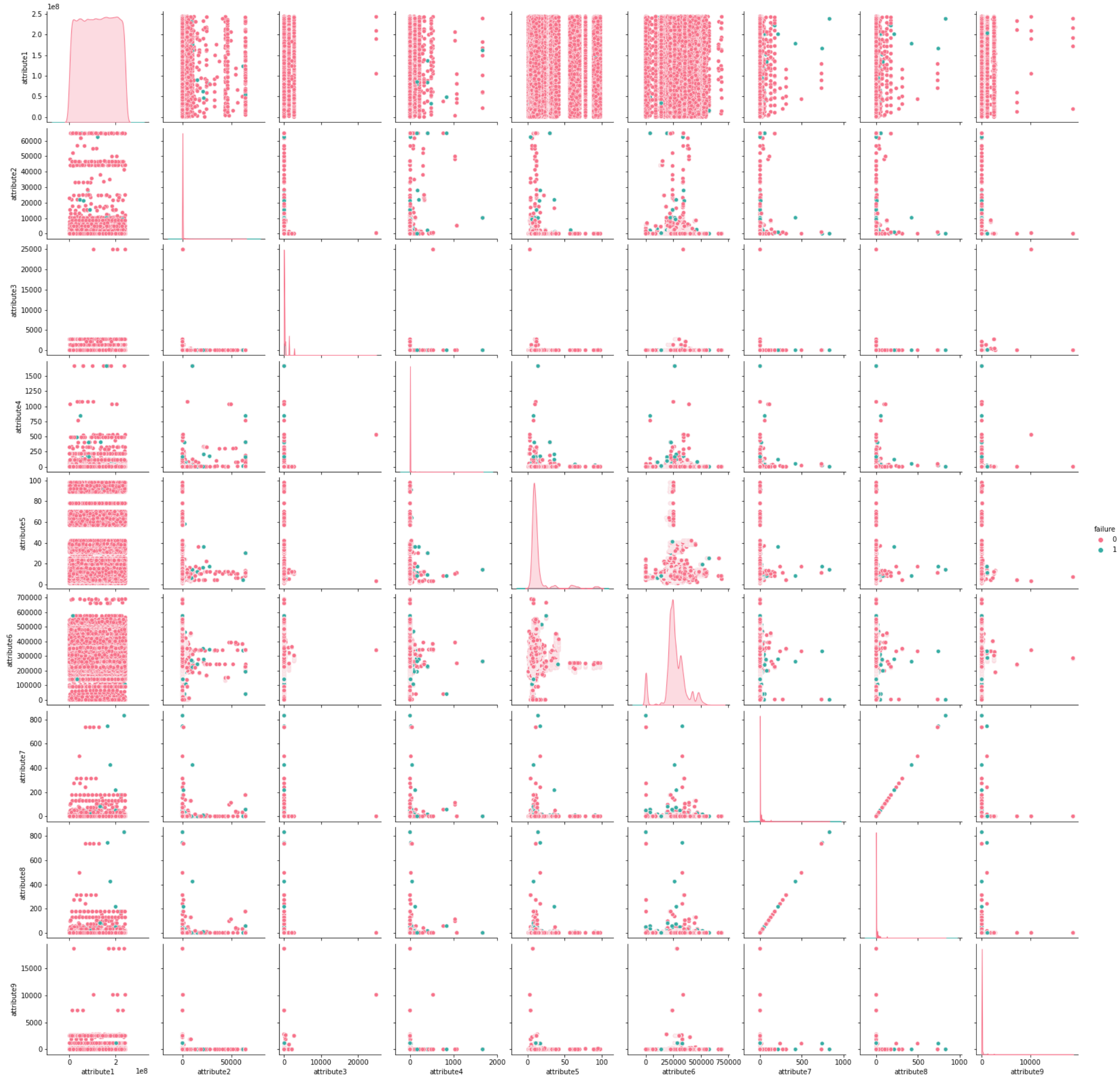
In [13]:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
#Maintenance_bp=pd.DataFrame(Maintenance_bp,index=[0, 1, 2, 3,4,5,6,7,8,9])
device_data_corr=device_data.corr()
sns.heatmap(device_data_corr, annot=True)

sns.pairplot(device_data, hue='failure', palette='husl')
```

Out[13]:

```
<seaborn.axisgrid.PairGrid at 0x1c12169b820>
```





Sampling the data

In [149…

```
scalar = MinMaxScaler()
obj_scaler = scalar.fit(device_data.iloc[:,3:])
# device_data_scaled = pd.DataFrame(obj)
```

In [150…

```
device_data_features = device_data.iloc[:,2:]
labels = device_data.iloc[:,2:3]
X_train_org,X_Unseen_features,y_train_org,y_Unseen_labels = train_test_split(device_da
X_Unseen_features.drop(["failure"],axis=1,inplace=True)

X_train = X_train_org.copy(deep=True)
y_train = y_train_org.copy(deep=True)
X_train.drop(["failure"],axis=1,inplace=True)
```

In above code data is split into X_train and X_unseen_data.

Model will be trained on train data and 10-fold cross validation will be done to check robustness of the model.

However performance of the model is measured against the unseen data that is splitted here.

In [19]:

```
def scale_data(data):
```

```
    scaled_data = pd.DataFrame(obj_scaler.transform(data))
    return scaled_data


def NN_model(X_train,y_train,X_test,y_test,eps,metric):
    model=Sequential()
    model.add(Dense(9,input_dim=9,activation="relu"))
    model.add(Dropout(0.2))
    model.add(Dense(15,activation="relu"))
    model.add(Dense(1,activation="sigmoid"))
    model.compile(loss="binary_crossentropy",optimizer="adam",metrics=[metric])

    model.fit(X_train,y_train,epochs=eps)
    preds=model.predict(X_test)
    print(preds)
#     preds=np.round(preds)
#     print(classification_report(y_test,preds))
    return model
```

In [159…

```
def argmax_class(data):
    return np.argmax(data),data[np.argmax(data)]

def post_process_output(result,threshold):
    """
    post_process_output - This function aims to do the post-processing of the model ou
                          returns both the classes with their probabilities.
    parameters :
                model - trained model after cross validation
                X_unseen - Unseen feature data
                y_unseen - Unseen label data
                threshold - confidence above which the prediction is considered as Tru

    """

    df_output_org = pd.DataFrame(result,columns=["pass","failure"])#.apply(argmax_clas
    df_output = df_output_org.T.apply(argmax_class).T
    df_output = df_output.rename(columns={"pass":"status","failure":"proba"})
    df_output["status"]=np.where(df_output["proba"]<threshold,0,1)
    return df_output["status"],df_output_org


def cross_validation_output(features_train,labels_train):
    model =  RandomForestClassifier(n_estimators=100,
                                class_weight='balanced')
    cv_results  = cross_val_score(model,features_train,labels_train
                ,scoring="f1"
                ,cv=10
               )

    model.fit(features_train,labels_train)
    return cv_results,model


def model_output_unseen_data(model,X_unseen,y_unseen,threshold):
    """
    model_output_unseen_data - To check the output of model trained in previous step c
    parameters :
```

```
                model - trained model after cross validation
                X_unseen - Unseen feature data
                y_unseen - Unseen label data
                threshold - confidence above which the prediction is considered as Tru

    returns -None

    """
    pred_on_test = model.predict_proba(X_unseen)
    pred_on_test,df_output_org = post_process_output(pred_on_test,threshold)
    print("F1 score on unseen data is %0.2f \n\n" % (f1_score(y_unseen,pred_on_test)))
    print(classification_report(y_unseen,pred_on_test))
    print(confusion_matrix(y_unseen,pred_on_test))

    return
```

Train base model without scaling

10 fold cross validation using f1 as scoring

```
Algo - Randomforestclassifier
parameter : n_estimators = 100, class_weights="balance"
```

In [21]:

```
result,rf_model = cross_validation_output(X_train,y_train["failure"].values)
print("Mean f1 score of %0.2f " % (result.mean()))
```

```
Mean f1 score of 0.00
```

In [22]:

```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       1.00      0.00      0.00     24881
           1       0.00      1.00      0.00        18

    accuracy                           0.00     24899
   macro avg       0.50      0.50      0.00     24899
weighted avg       1.00      0.00      0.00     24899

[[   14 24867]
 [    0    18]]
```

Objective - Predict the device failure while minimizing false positives and false negatives

As per objective, we will be using F1 score to judge the model performance.

Base model stats : </br> Cross validation F1 score = 0 </br> F1 score on unseen data = 0 </br> Confusion Matrix </br> True Negative : 14 </br> False Positive : 24867 </br> False Negative : 0 </br> True Positive : 18 </br>

The model performance is not good as there are a lot of False positives present even though all the TP cases are correctly identified

In [24]:
```
# modl = NN_model(X_sub_train,y_sub_train,X_val,y_val,10,keras.metrics.Recall())

# Above,trained and tested neural network model but not good enough result
```

Undersampling

Without scaling

In [25]:
```
df_failure=X_train_org[X_train_org["failure"]==1]
df_pass=X_train_org[X_train_org["failure"]==0]
X_under_sample = df_pass.sample(n=df_failure.shape[0])
```

In [26]:
```
new_train_data = pd.concat([X_under_sample,df_failure],axis=0)
X_train=new_train_data.drop(["failure"],axis=1)
y_train=new_train_data["failure"]
```

In [27]:
```
result,rf_model = cross_validation_output(X_train,y_train.values)
print("Mean f1 score of %0.2f" % (result.mean()))
```

```
Mean f1 score of 0.77
```

In [33]:
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       1.00      0.34      0.50     24881
           1       0.00      0.94      0.00        18

    accuracy                           0.34     24899
   macro avg       0.50      0.64      0.25     24899
weighted avg       1.00      0.34      0.50     24899

[[ 8368 16513]
 [    1    17]]
```

Undersampling without scaling approach stats : </br> Cross validation F1 score = 0.77 </br> F1 score on unseen data = 0 </br> Confusion Matrix </br> True Negative : 8368 </br> False Positive : 16513 </br> False Negative : 1 </br> True Positive : 17 </br>

The model performance is better than base model but not good as there are a lot of </br> False positives present even though most of the TP cases are correctly identified

Oversampling

Without scaling

In [34]:
```
df_failure_oversample=df_failure.sample(df_pass.shape[0],replace=True)
```

In [35]:
```
new_train_data = pd.concat([df_failure_oversample,df_pass],axis=0)
X_train=new_train_data.drop(["failure"],axis=1)
y_train=new_train_data["failure"]
```

In [36]:
```
result,rf_model = cross_validation_output(X_train,y_train.values)
print("Mean f1 score of %0.2f" % (result.mean()))
```

```
Mean f1 score of 1.00
```

In [40]:
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       1.00      0.00      0.00     24881
           1       0.00      1.00      0.00        18

    accuracy                           0.00     24899
   macro avg       0.50      0.50      0.00     24899
weighted avg       1.00      0.00      0.00     24899

[[   13 24868]
 [    0    18]]
```

Oversampling without scaling approach stats : </br> Cross validation F1 score = 1 </br> F1 score on unseen data = 0 </br> Confusion Matrix </br> True Negative : 13 </br> False Positive : 24868 </br> False Negative : 0 </br> True Positive : 18 </br>

With oversampling without scaling approach, even though the cross validation score is almost perfect (1.0)</br> and it has correctly predicted all the failure cases but the performance of the model on unseen data is poor.</br> As you can see there are 24868 false positive which is not a good sign.So we cannot accept this model.

Oversampling

With scaling

In [41]:
```
X_train=scale_data(X_train)
```

In [42]:
```
result,rf_model = cross_validation_output(X_train,y_train.values)
print("Mean f1 score of %0.2f" % (result.mean()))
```

```
Mean f1 score of 1.00
```

In [44]:
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\sklearn\base.py:443: UserWarning: X has feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       0.99      0.01      0.02     24881
           1       0.00      0.89      0.00        18

    accuracy                           0.01     24899
   macro avg       0.50      0.45      0.01     24899
weighted avg       0.99      0.01      0.02     24899

[[  231 24650]
 [    2    16]]
```

Oversampling with scaling approach stats : </br> Cross validation F1 score = 1 </br> F1 score on unseen data = 0 </br> Confusion Matrix </br>   True Negative : 231 </br>   False Positive : 24650 </br>   False Negative : 2 </br>   True Positive : 16 </br>

With oversampling with scaling approach, even though the cross validation score is almost perfect (0.89)</br> and it has correctly predicted most of the failure cases on unseen data but the precision on unseen data is poor.</br> As you can see there are 24650 which is not a good sign.So we cannot accept this model.

SMOTE

In [45]:
```
from imblearn.over_sampling import SMOTE
```

In [46]:
```
sm = SMOTE(sampling_strategy="minority")
# X_train_sm,y_train_sm  = .fit_sample(X_train.iloc[:,1:],X_train.iloc[:,0:1])
X_train_sm,y_train_sm = sm.fit_resample(X_train,y_train)
```

In [47]:
```
result,rf_model = cross_validation_output(X_train_sm,y_train_sm.values)
print("Mean  f1 score of %0.2f" % (result.mean()))
```

```
Mean  f1 score of 1.00
```

In [49]:
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\sklearn\base.py:443: UserWarning: X has feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       0.99      0.01      0.02     24881
           1       0.00      0.89      0.00        18

    accuracy                           0.01     24899
   macro avg       0.50      0.45      0.01     24899
weighted avg       0.99      0.01      0.02     24899

[[  230 24651]
 [    2    16]]
```

SMOTE without scaling approach stats : </br> Cross validation F1 score = 1 </br> F1 score on unseen data = 0 </br> Confusion Matrix </br> True Negative : 230 </br> False Positive : 24651 </br> False Negative : 2 </br> True Positive : 16 </br>

with SMOTE without scaling,the cross validation score is very good (1)</br> and it has correctly predicted most of the failure cases on unseen data </br>but on unseen data but the precision on unseen data is poor.As you can see there are 24651 false positive which is not a good sign. </br>So we cannot accept this model.

With scaling

In [52]:
```
X_train_sm = scale_data(X_train_sm)
# X_val = scale_data(X_val)
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X doe
s not have valid feature names, but MinMaxScaler was fitted with feature names
  warnings.warn(
```

In [53]:
```
result,rf_model = cross_validation_output(X_train_sm,y_train_sm.values)
print("Mean  f1 score of %0.2f" % (result.mean()))
```

```
Mean  f1 score of 0.87
```

In [54]:
```
X_Unseen_features = scale_data(X_Unseen_features)
```

In [55]:
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
F1 score on unseen data is 0.00


              precision    recall  f1-score   support

           0       0.99      0.01      0.02     24881
           1       0.00      0.89      0.00        18

    accuracy                           0.01     24899
   macro avg       0.50      0.45      0.01     24899
weighted avg       0.99      0.01      0.02     24899

[[  238 24643]
 [    2    16]]
```

In [ ]:

Undersampling

With scaling

In [151…

```
df_failure=X_train_org[X_train_org["failure"]==1]
df_pass=X_train_org[X_train_org["failure"]==0]
X_under_sample = df_pass.sample(n=df_failure.shape[0])
```

In [152…

```
new_train_data = pd.concat([X_under_sample,df_failure],axis=0)
X_train=new_train_data.drop(["failure"],axis=1)
y_train=new_train_data["failure"]
```

In [153…

```
X_train=scale_data(X_train)
```

In [154…

```
result,rf_model = cross_validation_output(X_train,y_train.values)
print("Mean f1 score of %0.2f" % (result.mean()))
```

```
Mean f1 score of 0.84
```

In [157…

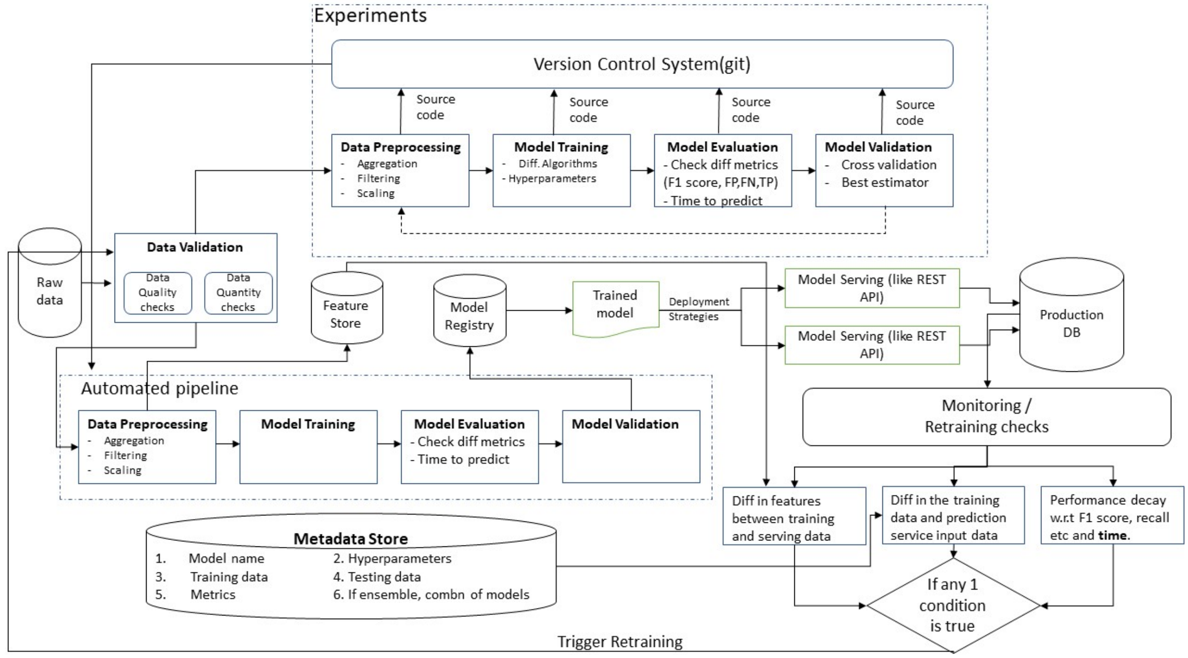
```
model_output_unseen_data(rf_model,X_Unseen_features,y_Unseen_labels["failure"].values,
```

```
C:\Users\vishalra\Anaconda3\lib\site-packages\sklearn\base.py:443: UserWarning: X has
feature names, but RandomForestClassifier was fitted without feature names
  warnings.warn(
F1 score on unseen data is 0.01


              precision    recall  f1-score   support

           0       1.00      0.92      0.96     24881
           1       0.01      0.67      0.01        18

    accuracy                           0.92     24899
   macro avg       0.50      0.79      0.48     24899
weighted avg       1.00      0.92      0.96     24899

[[22876  2005]
 [    6    12]]
```

Undersampling with scaling approach </br> Cross validation F1 score = 0.84 </br> F1 score on unseen data = 0.01 </br> Confusion Matrix </br> True Negative : 22876 </br> False Positive : 2005 </br> False Negative : 6 </br> True Positive : 12 </br>

Finally above approach can be selected because </br> F1 score on cross validation is pretty good(0.84) and also </br> its performance on unseen data is very good compared to other </br> approaches we tried.The false positives are just 2005 and </br> the recall is acceptable as 12 out of 18 cases were correctly predicted on unseen data.

MLOps Architecture Design and Documentation



Above design can be summed up into 3 parts

1. Versioning
2. Automation & Deployment
3. Monitoring / Retraining

Versioning

The idea behind the versioning is to keep track of Data,ML model and code base. The reasons for this are :

1. In production environment, if significant changes in the nature of the data are observed, we might need to retrain the model
2. Model may also be needed to retrain based on new approaches. State of the art methods and different designs.
3. Model performance may degrade over the time so it may need to retrain over latest data.

In all the above scenarios doing the versioning helps us keeping track of what had been done at any point in time. Furthermore, if there comes a time when we want to troubleshoot a particular version of the model, we may need training data and testing data on top of metrics to zero down the root cause.

Automation

The speed of delivering new model is highly dependent on the level of automation present for ML workflows. The objective here is to automate the complete ML-workflow steps without any manual intervention i.e.CI/CD.

Development and experimentation : This is where I would try and experiment different ideas to achieve the ML objective.

Continuous Integration (CI) : Once the source code is built, few test cases will be run.Upon confirming the expected output, packages, executables, and artifacts will be stored into respective stores(Feature store,metadata store & model registry).

Continuous Delivery (CD) : The artifacts produced by previous stage(CI) will be deployed to the target environment using differnt CD tools.This will conclude the new implementation of the model on target environment.

There are multiple deployment strategies that can serve the purpose.Depending upon use case the right deployment strategy is selected. For the given predictive maintenance problem, I would start with Shallow or Canary deployment strategy.

Monitoring / Retraining

Once the model is deployed it needs to be monitored for different parameters.

The performance of the model is essentialy depends on the nature of the data consumed by model to make predictions.

1. Consider a scenario for predictive maintenance problem where it has been observed that distribution of attribute 1 in production is significantly different than one on training data then we have to consider re-training the model on latest data.
2. Also there could be a scenario where a new sensor is added i.e. a new attribute10 which tells additional information about the machine then such factors need to be incorporated in model for which retraining is required.
3. Machine learning models may results into lower performance over the period of time, which is also an indication of retraining.
4. One more aspect of performance could be time complexity. There could be a use case where time required to get the predictions has to be in milliseconds or time sensitive.However for some reason time taken in production environment is more than stipulated time. In these scenario we might have to think about retraining with another algorithm which time and memory efficient.

All of these factors are covered in the design above

In [ ]:

```

```