입사평가를 위한 데이터 분석

지원자 안효준

환경설정, 데이터 입출력, 패키지 불러오기

In [2]:
```
import pandas as pd


import time
import warnings

warnings.filterwarnings('ignore')
pd.options.display.max_columns = 999
```

In [3]:
```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [4]:
```
train.columns
```

Out[4]:
```
Index(['ID', 'Date', 'Temperature', 'Humidity', 'Operator', 'Measure1',
       'Measure2', 'Measure3', 'Measure4', 'Measure5', 'Measure6', 'Measure7',
       'Measure8', 'Measure9', 'Measure10', 'Measure11', 'Measure12',
       'Measure13', 'Measure14', 'Measure15', 'Hours Since Previous Failure',
       'Failure', '?Date.year', '?Date.month', '?Date.day-of-month',
       '?Date.day-of-week', '?Date.hour', '?Date.minute', '?Date.second'],
      dtype='object')
```

In [5]:
```
test.columns
```

Out[5]:
```
Index(['ID', 'Date', 'Temperature', 'Humidity', 'Operator', 'Measure1',
       'Measure2', 'Measure3', 'Measure4', 'Measure5', 'Measure6', 'Measure7',
       'Measure8', 'Measure9', 'Measure10', 'Measure11', 'Measure12',
       'Measure13', 'Measure14', 'Measure15', 'Hours Since Previous Failure',
       '?Date.year', '?Date.month', '?Date.day-of-month', '?Date.day-of-week',
       '?Date.hour', '?Date.minute', '?Date.second'],
      dtype='object')
```

Failure 변수 파악

In [6]:
```
train.Failure.value_counts()
```

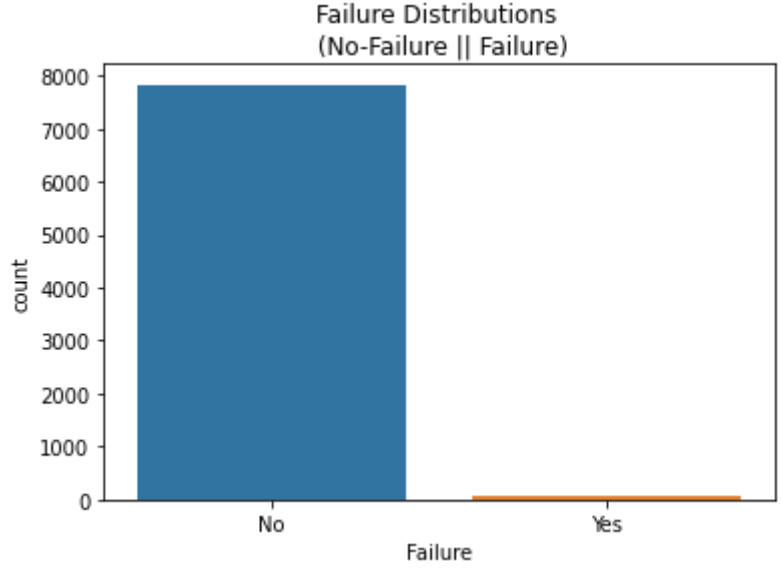
Out[6]:
```
No    7830
Yes     75
Name: Failure, dtype: int64
```

In [7]:
```
print('Ratio of Failure', round(
   train.Failure.value_counts()[1]/len(train) * 100, 2), '%')
```

```
Ratio of Failure 0.95 %
```

In [8]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.countplot('Failure', data=train)
plt.title('Failure Distributions \n (No-Failure || Failure)', fontsize=12);
```



라벨 인코더 사용

- No : 0, Yes : 1

In [9]:
```
# 라벨 인코더 사용

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(train['Failure'])
train['failure'] = le.transform(train['Failure'])
train.head(2)
```

Out[9]:

	ID	Date	Temperature	Humidity	Operator	Measure1	Measure2	Measure3	Measure4	Measure5	Measure6	Measure7	Meas
0	1	01-01-2016 00:00	67	82	Operator1	291	1	1	1041	846	334	706	1086
1	2	01-01-2016 01:00	68	77	Operator1	1180	1	1	1915	1194	637	1093	524

In [10]:
```
train.failure.value_counts()
```

Out[10]:
```
0   7830
1     75
Name: failure, dtype: int64
```

Operator 별 fail 비율 파악

In [11]:

```
op = train[['Operator', 'failure']]
failure_op = op.groupby(['Operator', 'failure']).agg({'Operator': 'count'})
op_count = op.groupby(['Operator']).agg('count')
op_failure = failure_op.div(op_count, level='Operator') * 100
failure_op.div(op_count, level='Operator')
failure_op
# op.groupby('Operator').Failure.value_counts()
```

Out[11]:

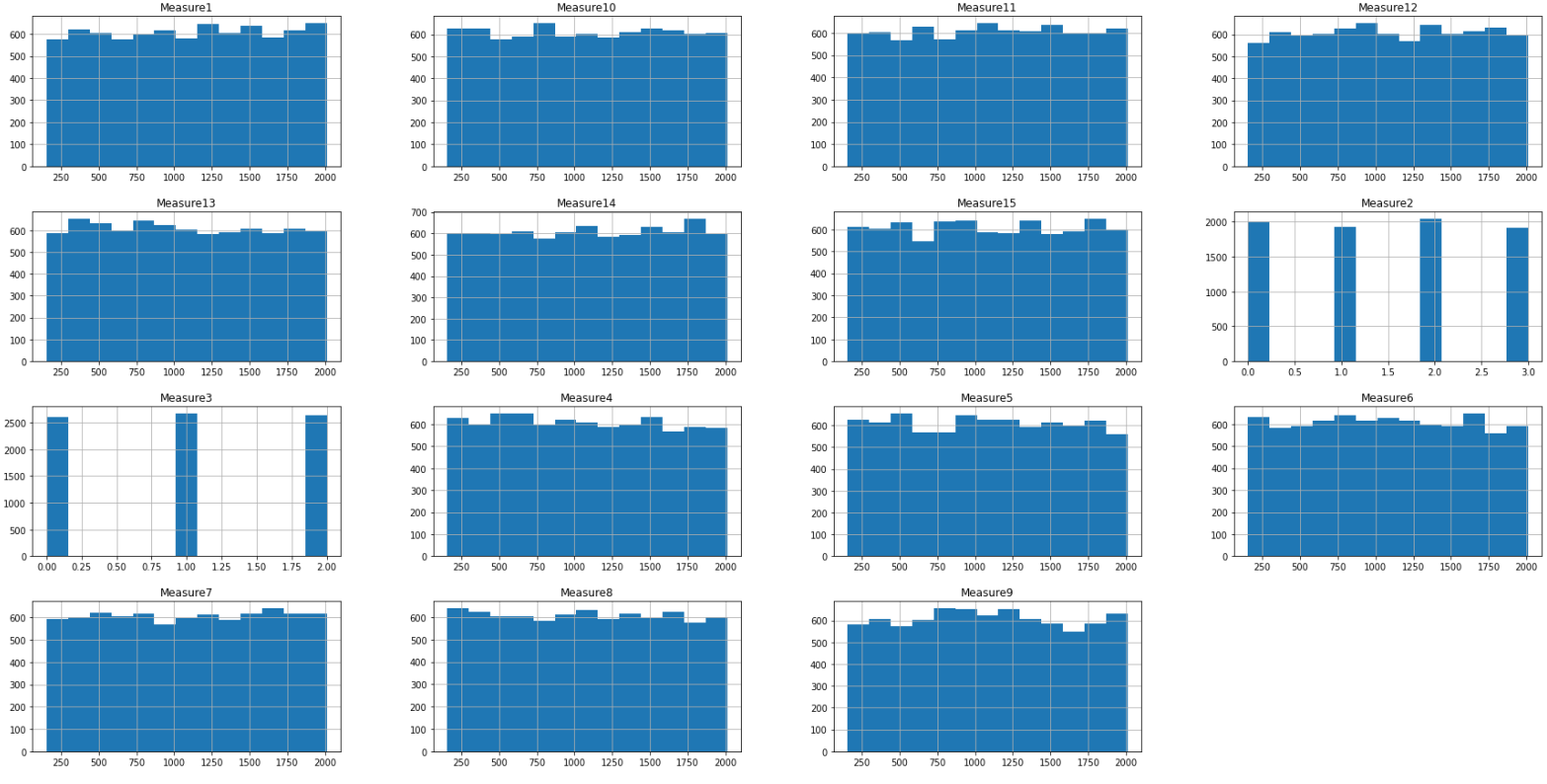
		Operator
Operator	**failure**	
Operator1	**0**	870
	1	10
Operator2	**0**	1734
	1	19
Operator3	**0**	871
	1	9
Operator4	**0**	869
	1	11
Operator5	**0**	875
	1	5
Operator6	**0**	875
	1	5
Operator7	**0**	866
	1	14
Operator8	**0**	870
	1	2

훈련 데이터 변수 분리

In [12]:
```
features = train.loc[:, 'Temperature':'Measure15']
features = features.drop(['Operator'], axis=1)
failure = train[['failure']]

features.shape, failure.shape
```

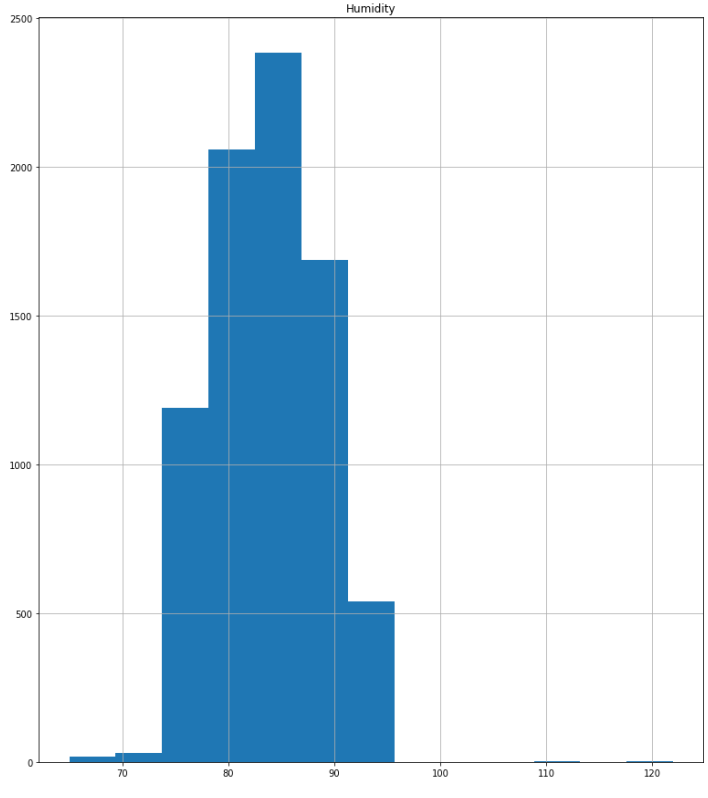
Out[12]:
```
((7905, 17), (7905, 1))
```

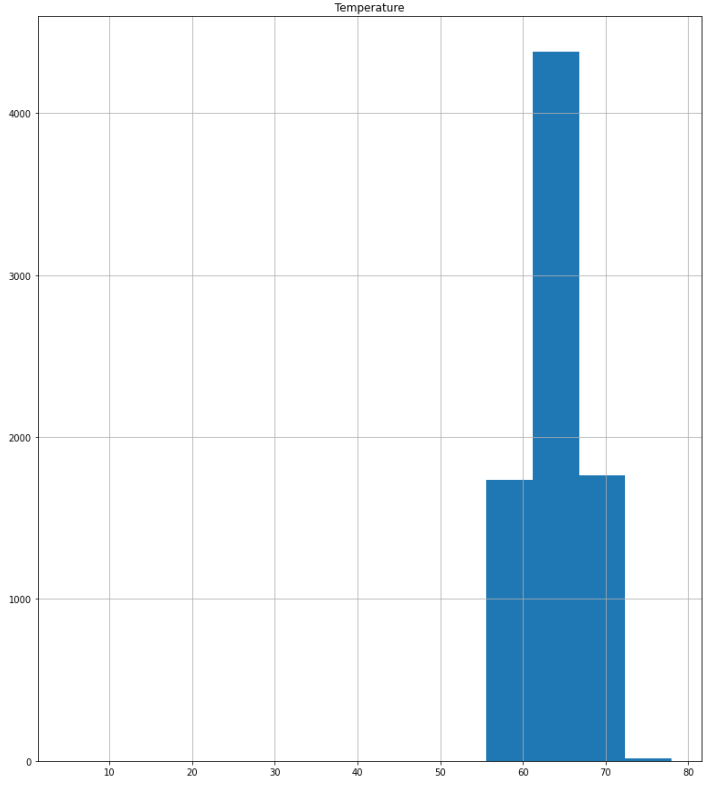
In [13]:
```
features.loc[:, 'Measure1':'Measure15'
        ].hist(bins=13, figsize=(30, 15));
```



In [14]:
```
features[['Temperature', 'Humidity']
    ].hist(bins=13, figsize=(30, 15))
```

Out[14]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f88dce0b1d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f88dccc6a90>]],
      dtype=object)
```





트레인 테스트 데이터 분리

- train : test = 8:2
- failure -> stratify 적용, Failure 비율 유지

In [15]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    features, failure, test_size=0.2, random_state=13, stratify=failure)
```

In [16]:
```
# 트레인 데이터 불균형 확인
import numpy as np

np.unique(y_train, return_counts=True)
```

Out[16]:
```
(array([0, 1]), array([6264,   60]))
```

In [17]:
```
tmp = np.unique(y_train, return_counts=True)[1]
print(round(tmp[1]/len(y_train) * 100, 2), '%')
```

```
0.95 %
```

In [18]:
```
# 테스트 데이터 불균형 확인

np.unique(y_test, return_counts=True)
```

Out[18]:
```
(array([0, 1]), array([1566,   15]))
```

In [19]:
```
tmp = np.unique(y_test, return_counts=True)[1]
print(round(tmp[1]/len(y_test) * 100, 2), '%')
```

```
0.95 %
```

예측 모델 생성

- 함수 선언
    - 평가 지표, confusion matrix

In [20]:
```
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score)

def get_clf_eval(y_test, pred):
    acc = accuracy_score(y_test, pred)
    pre = precision_score(y_test, pred)
    re = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    acu = roc_auc_score(y_test, pred)

    return acc, pre, re, f1, acu
```

In [21]:
```
from sklearn.metrics import confusion_matrix

def print_clf_eval(y_test, pred):
    confusion = confusion_matrix(y_test, pred)
    acc, pre, re, f1, auc = get_clf_eval(y_test, pred)
    print('confusion matrix')
    print(confusion)
    print('Accuracy: {0:.4f}, precision: {1:.4f}'.format(acc, pre))
    print('Recall: {0:.4f}, F1: {1:.4f}, AUC: {2:.4f}'.format(re, f1, auc))
```

Logistic

In [22]:

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(random_state=34, solver='liblinear')
lr_clf.fit(X_train, y_train)
lr_pred = lr_clf.predict(X_test)

print_clf_eval(y_test, lr_pred)
```

```
confusion matrix
[[1565    1]
 [   5   10]]
Accuracy: 0.9962, precision: 0.9091
Recall: 0.6667, F1: 0.7692, AUC: 0.8330
```

DecisionTree

In [23]:
```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier(random_state=34, max_depth=4)
dt_clf.fit(X_train, y_train)
dt_pred = dt_clf. predict(X_test)

print_clf_eval(y_test, dt_pred)
```

```
confusion matrix
[[1560    6]
 [   1   14]]
Accuracy: 0.9956, precision: 0.7000
Recall: 0.9333, F1: 0.8000, AUC: 0.9648
```

RandomForestClassifier

In [24]:
```
from sklearn.ensemble import RandomForestClassifier

rf_clf = RandomForestClassifier(random_state=13, n_jobs=-1, n_estimators=100)
rf_clf.fit(X_train, y_train)
rf_pred = rf_clf.predict(X_test)

print_clf_eval(y_test, rf_pred)
```

```
confusion matrix
[[1562    4]
 [   2   13]]
Accuracy: 0.9962, precision: 0.7647
Recall: 0.8667, F1: 0.8125, AUC: 0.9321
```

GBM

In [25]:
```
from sklearn.ensemble import GradientBoostingClassifier

start_time = time.time()

gb_clf = GradientBoostingClassifier(random_state=34)
gb_clf.fit(X_train, y_train)
gb_pred = gb_clf.predict(X_test)

print_clf_eval(y_test, gb_pred)
print('fit_time : ', time.time() - start_time)
```

```
confusion matrix
[[1557    9]
 [   1   14]]
Accuracy: 0.9937, precision: 0.6087
Recall: 0.9333, F1: 0.7368, AUC: 0.9638
fit_time :  1.6211180686950684
```

In [26]:
```
gb_clf.n_estimators_
```

Out[26]:
```
100
```

GBM Grid Search

- 그리드 서치 사용

In [27]:
```
# 그리드 서치

from sklearn.model_selection import GridSearchCV

params = {
    'n_estimators': [100, 200, 300, 400, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1]
}

start_time = time.time()
grid = GridSearchCV(gb_clf, param_grid=params, cv=5, verbose=1, n_jobs=-1)
grid.fit(X_train, y_train)
print('Fit time: ', time.time() - start_time)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:   19.8s
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:  1.1min finished

Fit time:  66.94274616241455
```

In [28]:
```
grid.best_score_
grid.best_estimator_
```

Out[28]:
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=200,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=34, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [29]:
```
print_clf_eval(y_test, grid.best_estimator_.predict(X_test))
```

```
confusion matrix
[[1562    4]
 [   2   13]]
Accuracy: 0.9962, precision: 0.7647
Recall: 0.8667, F1: 0.8125, AUC: 0.9321
```

XGBoost

In [30]:
```
from xgboost import XGBClassifier

start_time = time.time()
xgb = XGBClassifier(n_estimators=200, learning_rate=0.1, max_depth=3)
xgb.fit(X_train, y_train)
print('Fit time: ', time.time() - start_time)
```

```
Fit time:  0.24648642539978027
```

In [31]:
```
print_clf_eval(y_test, xgb.predict(X_test))
```

```
confusion matrix
[[1562    4]
 [   1   14]]
Accuracy: 0.9968, precision: 0.7778
Recall: 0.9333, F1: 0.8485, AUC: 0.9654
```

LGBMClassifier

In [32]:
```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(
    n_estimators=1000, num_leaves=128, n_jobs=-1, boost_from_average=False)
lgbm_clf.fit(X_train, y_train)
lgbm_pred = lgbm_clf.predict(X_test)

print_clf_eval(y_test, lgbm_pred)
```

```
confusion matrix
[[1565    1]
 [   4   11]]
Accuracy: 0.9968, precision: 0.9167
Recall: 0.7333, F1: 0.8148, AUC: 0.8663
```

모델 결과 정리

- 함수 선언
    - 개별 모델 결과
    - 모델 결과 수집

In [33]:
```
def get_result(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    pred = model.predict(X_test)

    return get_clf_eval(y_test, pred)
```

In [34]:
```
def get_result_pd(models, model_names, X_train, y_train, X_test, y_test):
    col_names = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']
    tmp = []

    for model in models:
        tmp.append(get_result(model, X_train, y_train, X_test, y_test))

    return pd.DataFrame(tmp, columns=col_names, index=model_names)
```

In [35]:
```
import time

models = [lr_clf, dt_clf, rf_clf, lgbm_clf, xgb, gb_clf]
model_names = ['LogisticReg', 'DecisionTree', 'RandomForest',
               'LightGBM', 'XGBoost', 'GradientBoosting']

start_time = time.time()
results = get_result_pd(models, model_names, X_train, y_train, X_test, y_test)

print('Fit time: ', time.time() - start_time)
normal_result = results.sort_values('accuracy', ascending=False)
normal_result
```

```
Fit time:  3.190579652786255
```

Out[35]:

	accuracy	precision	recall	f1	roc_auc
LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
LogisticReg	0.996205	0.909091	0.666667	0.769231	0.833014
RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793

In [36]:
```
results.sort_values('precision', ascending=False)
```

Out[36]:

	accuracy	precision	recall	f1	roc_auc
LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
LogisticReg	0.996205	0.909091	0.666667	0.769231	0.833014
XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793

In [37]:
```
results.sort_values('recall', ascending=False)
```

Out[37]:

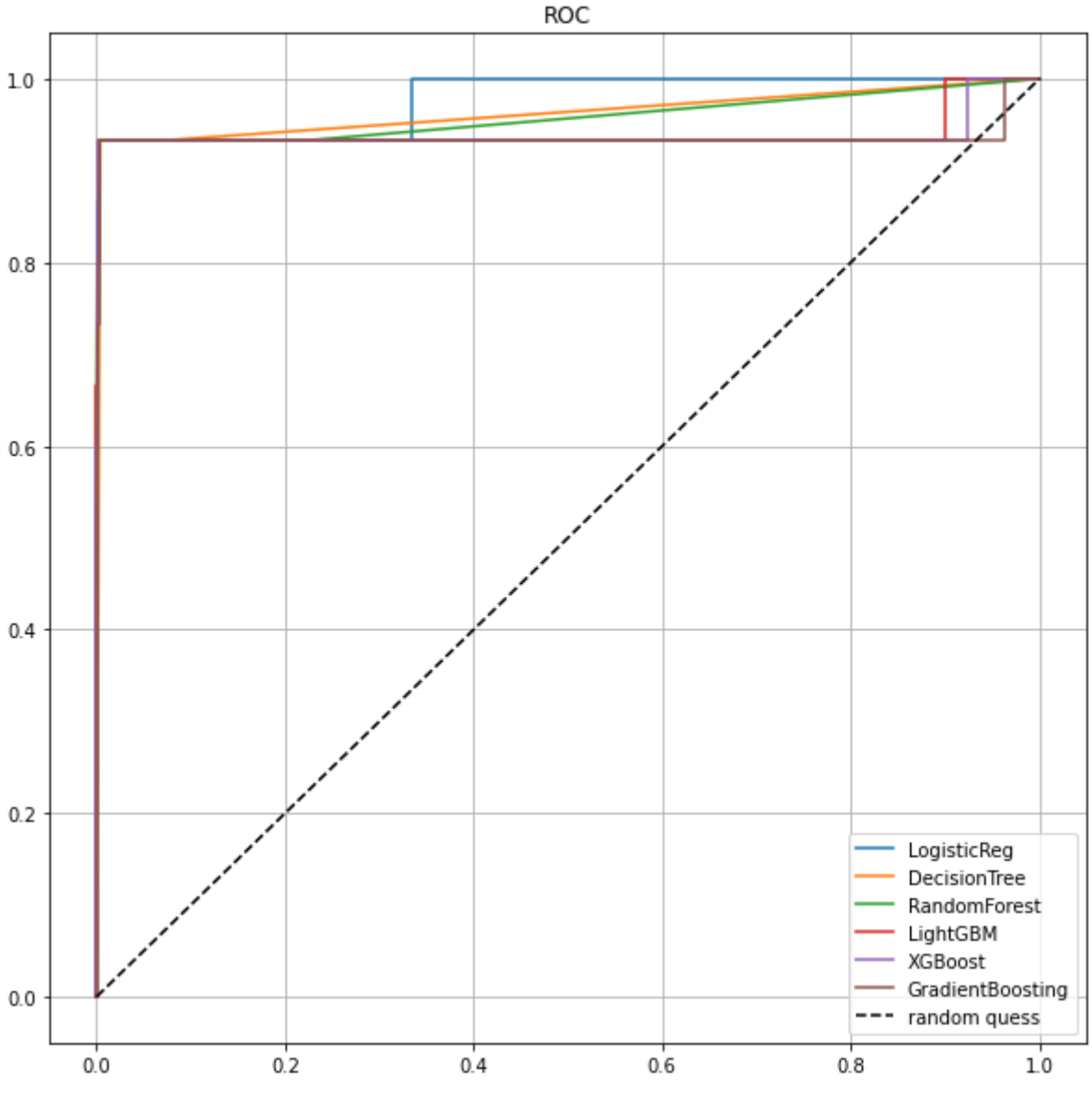
	accuracy	precision	recall	f1	roc_auc
DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793
RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
LogisticReg	0.996205	0.909091	0.666667	0.769231	0.833014

ROC 커브

In [38]:

```
from sklearn.metrics import roc_curve


def draw_roc_curve(models, model_names, X_test, y_test):
    plt.figure(figsize=(10, 10))

    for model in range(len(models)):
        pred = models[model].predict_proba(X_test)[:, 1]
        fpr, tpr, thresholds = roc_curve(y_test, pred)
        plt.plot(fpr, tpr, label=model_names[model])

    plt.plot([0, 1], [0, 1], 'k--', label='random quess')
    plt.title('ROC')
    plt.legend()
    plt.grid()
    plt.show()
```

In [39]:
```
draw_roc_curve(models, model_names, X_test, y_test)
```



Five-cross Validation

In [40]:

```
from sklearn.model_selection import KFold, cross_val_score, KFold

results = []
cv_result = []


def cross_model(models):
    for model in models:
        kfold = KFold(n_splits=5, random_state=34, shuffle=True)
        cv_results = cross_val_score(
            model, X_train, y_train, cv=kfold, scoring='accuracy')
        results.append(cv_results)

    return results
```

In [41]:
```
# 결과 정리 - 모델명 + 결과
cross_result = cross_model(models)
```

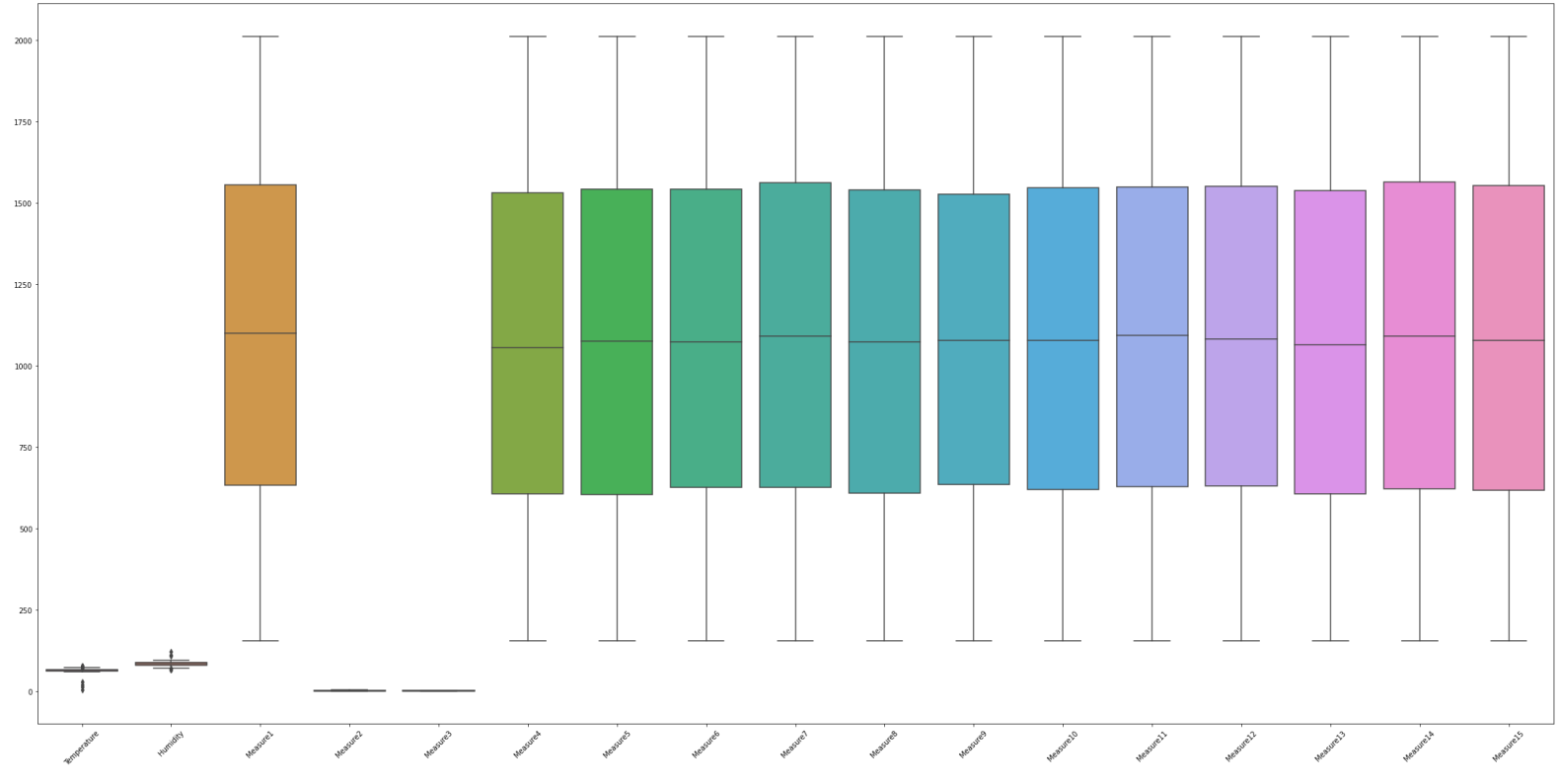
In [42]:
```
cross_total_result = pd.DataFrame(cross_result, index=model_names)
# five cross 평균치
cross_total_result['mean'] = cross_total_result.mean(axis=1)
# 평균 높은 순으로 정렬
cross_total_result.sort_values('mean', ascending=False)
```

Out[42]:

	0	1	2	3	4	mean
RandomForest	0.995257	0.994466	0.996838	0.996047	0.998418	0.996205
DecisionTree	0.995257	0.992885	0.997628	0.996047	0.998418	0.996047
XGBoost	0.996047	0.992885	0.997628	0.995257	0.998418	0.996047
LogisticReg	0.995257	0.992095	0.997628	0.996838	0.996835	0.995731
GradientBoosting	0.996838	0.992095	0.996047	0.995257	0.996835	0.995415
LightGBM	0.993676	0.992095	0.997628	0.995257	0.997627	0.995257

이상치 확인

In [44]:
```
plt.figure(figsize=(30, 15))
sns.boxplot(data=features)
plt.xticks(rotation=45)
plt.tight_layout()
```



스케일링

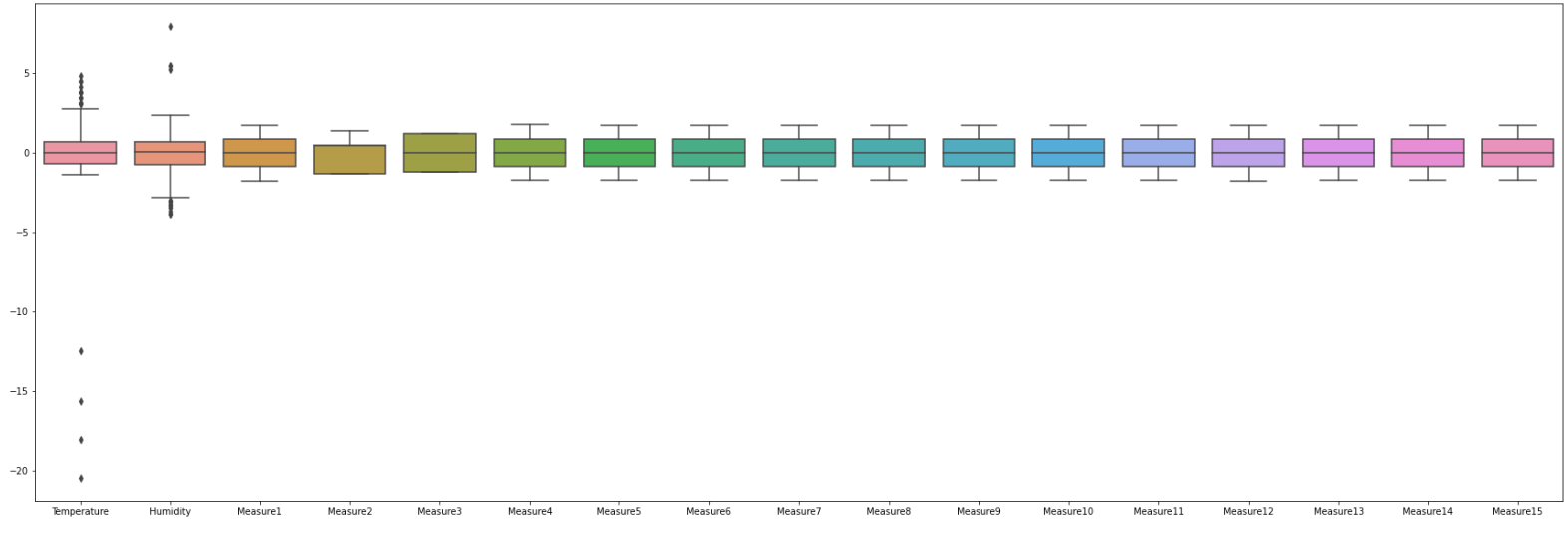
- MinMaxScaler
- StandardScalre

In [45]:
```
from sklearn.preprocessing import MinMaxScaler, StandardScaler

MMS = MinMaxScaler()
SS = StandardScaler()

MMS.fit(features)
SS.fit(features)

features_MMS = MMS.transform(features)
features_SS = SS.transform(features)
```

In [46]:
```
features_MMS_pd = pd.DataFrame(features_MMS, columns=features.columns)
features_SS_pd = pd.DataFrame(features_SS, columns=features.columns)
```

In [48]:
```
# standard scaler
import seaborn as sns
plt.figure(figsize=(30, 10))
sns.boxplot(data=features_SS_pd)
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f88ddc89c90>
```



In [49]:
```
# MinMaxScaler 데이터 나누고 결과 확인
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   features_MMS_pd, failure, test_size=0.2, random_state=13, stratify=failure)

start_time = time.time()
results = get_result_pd(models, model_names, X_train, y_train, X_test, y_test)

print('Fit time: ', time.time() - start_time)
mm_result = results.sort_values('accuracy', ascending=False)
mm_result
```

```
Fit time:  3.5407588481903076
```

Out[49]:

	accuracy	precision	recall	f1	roc_auc
LightGBM	0.997470	0.923077	0.800000	0.857143	0.899681
XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793
LogisticReg	0.990512	0.000000	0.000000	0.000000	0.500000

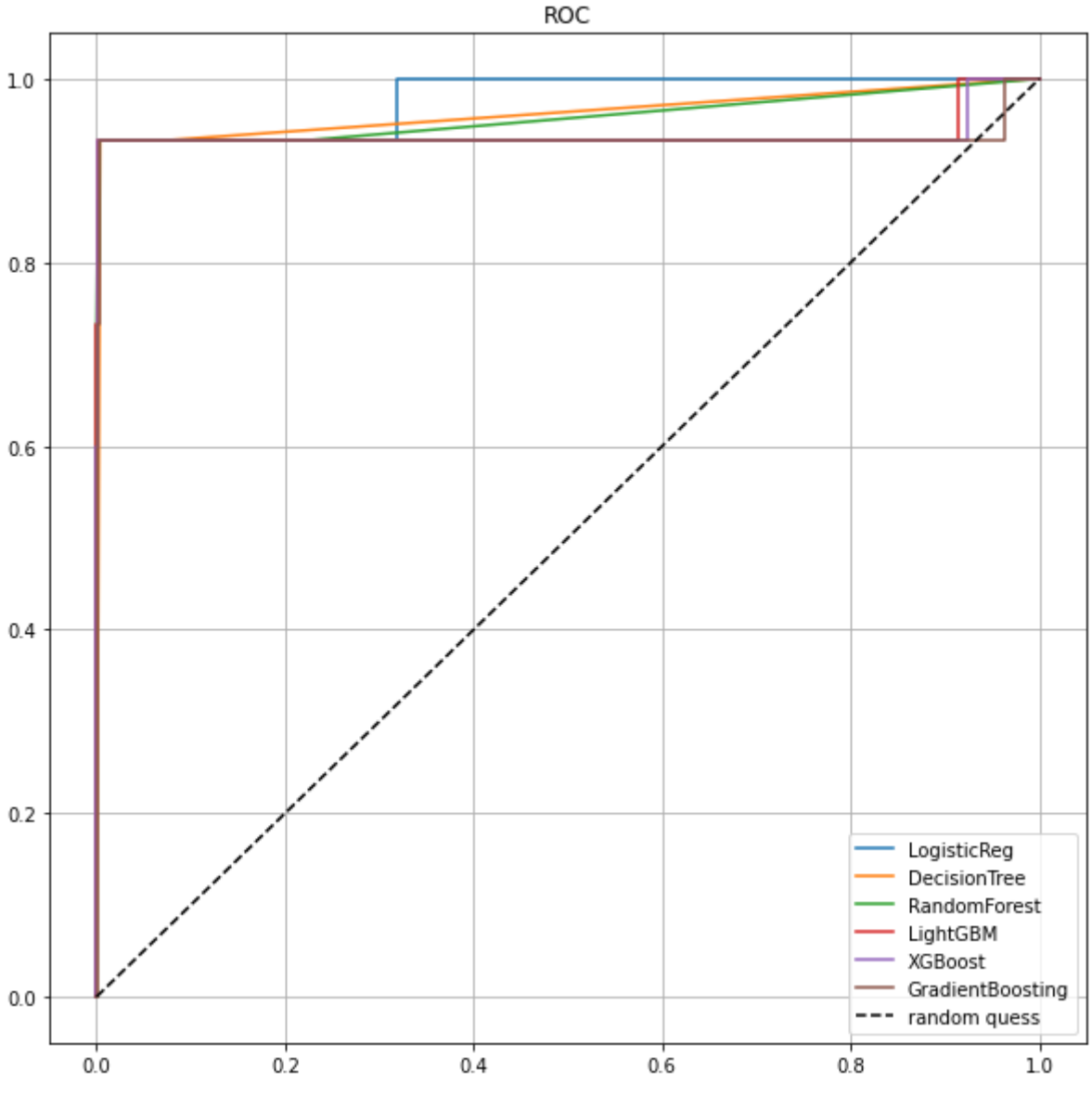
In [50]:
```
# StandardScaler 데이터 나누고 결과 확인
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   features_SS_pd, failure, test_size=0.2, random_state=13, stratify=failure)

start_time = time.time()
results = get_result_pd(models, model_names, X_train, y_train, X_test, y_test)

print('Fit time: ', time.time() - start_time)
ss_result = results.sort_values('accuracy', ascending=False)
ss_result
```

```
Fit time:  3.2248706817626953
```

Out[50]:

	accuracy	precision	recall	f1	roc_auc
LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
LogisticReg	0.995572	0.900000	0.600000	0.720000	0.799681
DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793

In [51]:
```
draw_roc_curve(models, model_names, X_test, y_test)
```



Over & Under Sampling

- Fail 데이터가 적어 불균형성을 해소해보기 위해 시도

In [52]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=34)
X_train_over, y_train_over = smote.fit_sample(X_train, y_train)
```

In [53]:
```
X_train.shape, y_train.shape
```

Out[53]: ((6324, 17), (6324, 1))

In [54]:
```
X_train_over.shape, y_train_over.shape
```

Out[54]: ((12528, 17), (12528, 1))

In [55]:
```
# fail 증폭
print(np.unique(y_train, return_counts=True))
print(np.unique(y_train_over, return_counts=True))
```

```
(array([0, 1]), array([6264,   60]))
(array([0, 1]), array([6264, 6264]))
```

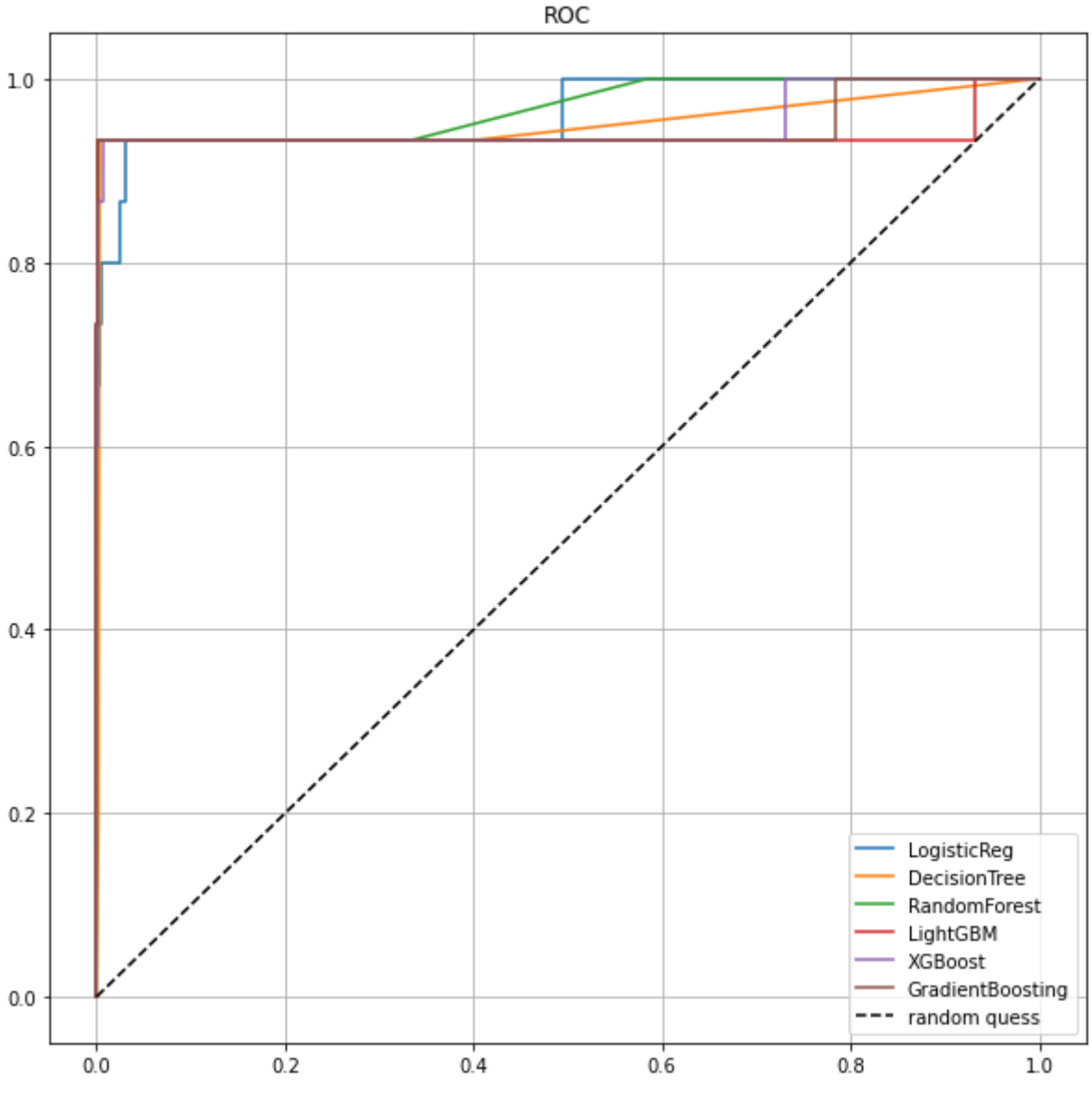
In [56]:
```
import time

models = [lr_clf, dt_clf, rf_clf, lgbm_clf, xgb, gb_clf]
model_names = ['LogisticReg', 'DecisionTree', 'RandomForest',
          'LightGBM', 'XGBoost', 'GradientBoosting']

start_time = time.time()
results = get_result_pd(models, model_names, X_train_over,
                  y_train_over, X_test, y_test)

print('Fit time: ', time.time() - start_time)
ou_result = results.sort_values('accuracy', ascending=False)
ou_result
```

```
Fit time:  6.487576484680176
```

Out[56]:

	accuracy	precision	recall	f1	roc_auc
RandomForest	0.997470	0.823529	0.933333	0.875000	0.965709
LightGBM	0.996837	0.777778	0.933333	0.848485	0.965390
GradientBoosting	0.996837	0.777778	0.933333	0.848485	0.965390
DecisionTree	0.996205	0.736842	0.933333	0.823529	0.965070
XGBoost	0.994940	0.684211	0.866667	0.764706	0.931418
LogisticReg	0.901961	0.083333	0.933333	0.153005	0.917497

In [57]:
```
draw_roc_curve(models, model_names, X_test, y_test)
```



테스트 적용

In [58]:
```
pd.concat([normal_result, ss_result, mm_result, ou_result], axis=0,
          keys=['normal', 'standard', 'MinMax', 'Balancing'])
```

Out[58]:

		accuracy	precision	recall	f1	roc_auc
normal	LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
	XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
	LogisticReg	0.996205	0.909091	0.666667	0.769231	0.833014
	RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
	DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
	GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793
standard	LightGBM	0.996837	0.916667	0.733333	0.814815	0.866347
	XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
	RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
	LogisticReg	0.995572	0.900000	0.600000	0.720000	0.799681
	DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
	GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793
MinMax	LightGBM	0.997470	0.923077	0.800000	0.857143	0.899681
	XGBoost	0.996837	0.777778	0.933333	0.848485	0.965390
	RandomForest	0.996205	0.764706	0.866667	0.812500	0.932056
	DecisionTree	0.995572	0.700000	0.933333	0.800000	0.964751
	GradientBoosting	0.993675	0.608696	0.933333	0.736842	0.963793
	LogisticReg	0.990512	0.000000	0.000000	0.000000	0.500000
Balancing	RandomForest	0.997470	0.823529	0.933333	0.875000	0.965709
	LightGBM	0.996837	0.777778	0.933333	0.848485	0.965390
	GradientBoosting	0.996837	0.777778	0.933333	0.848485	0.965390
	DecisionTree	0.996205	0.736842	0.933333	0.823529	0.965070
	XGBoost	0.994940	0.684211	0.866667	0.764706	0.931418
	LogisticReg	0.901961	0.083333	0.933333	0.153005	0.917497

Normal xgboost 모델 선택

In [59]:
```
features_test = test[['Temperature', 'Humidity', 'Measure1',
                      'Measure2', 'Measure3', 'Measure4', 'Measure5', 'Measure6', 'Measure7',
                      'Measure8', 'Measure9', 'Measure10', 'Measure11', 'Measure12',
                      'Measure13', 'Measure14', 'Measure15']]
```

In [61]:
```
test['Failure'] = xgb.predict(features_test)
```

In [63]:
```
test.loc[test.Failure == 0, 'Failure'] = 'No'
test.loc[test.Failure == 1, 'Failure'] = 'Yes'
```

결과

- 선택한 훈련 모델을 기반으로 테스트 데이터로 테스트한 결과 모두 No로 분류함

In [64]:
```
test.Failure.value_counts()
```

Out[64]:
```
No    879
Name: Failure, dtype: int64
```

In [65]:
```
result_test = test[['ID', 'Failure']]
```

In [66]:
```
result_test.to_csv('submisson_fin.csv')
```