Data Analysis Practical Test E15: Test Notebook

COMMON: Load Package

Question No. 1

주어진 데이터 파일은 특정 철강사의 제품코드, 불량코드, 그리고 공정 과정에서 발생한 데이터를 담고 있다.
해당 데이터를 이용하여 다음 문제의 답변을 작성하시오.

제공 데이터 파일: E15Q1_data_raw.csv

- 1-24번 컬럼: Analog Data
- 25번 컬럼: 제품코드 (Binary)
- 26번 컬럼: 불량코드 (Integer with range 1 to 7)

① EDA를 실시하여 결과값을 제시하고, 상관분석을 시행하여 변수 선택 및 파생 변수 생성과정을 풀이하시오.

In [3]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [4]:

```
raw = pd.read_csv('DataAnalPrac-master/DataAnalPrac-master/E15/data/E15Q1_data_raw.csv', engine = 'p
data = raw.copy()
```

데이터의 기본 정보 확인하기

In [5]:

```
data.head()
```

Out[5]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perimeter	S
0	42	50	270900	270944	267	17	44	
1	645	651	2538079	2538108	108	10	30	
2	829	835	1553913	1553931	71	8	19	
3	853	860	369370	369415	176	13	45	
4	1289	1306	498078	498335	2409	60	260	

5 rows × 27 columns

In [6]:

```
data.describe()
```

Out[6]:

	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_
count	1941.000000	1941.000000	1.941000e+03	1.941000e+03	1941.000000	1941.000000	19
mean	571.136012	617.964451	1.650685e+06	1.650739e+06	1893.878413	111.855229	
std	520.690671	497.627410	1.774578e+06	1.774590e+06	5168.459560	301.209187	4
min	0.000000	4.000000	6.712000e+03	6.724000e+03	2.000000	2.000000	
25%	51.000000	192.000000	4.712530e+05	4.712810e+05	84.000000	15.000000	
50%	435.000000	467.000000	1.204128e+06	1.204136e+06	174.000000	26.000000	
75%	1053.000000	1072.000000	2.183073e+06	2.183084e+06	822.000000	84.000000	
max	1705.000000	1713.000000	1.298766e+07	1.298769e+07	152655.000000	10449.000000	181

8 rows × 27 columns

In [7]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1941 entries, 0 to 1940
Data columns (total 27 columns):
X_Minimum                 1941 non-null int64
X_Maximum                 1941 non-null int64
Y_Minimum                 1941 non-null int64
Y_Maximum                 1941 non-null int64
Pixels_Areas              1941 non-null int64
X_Perimeter               1941 non-null int64
Y_Perimeter               1941 non-null int64
Sum_of_Luminosity         1941 non-null int64
Minimum_of_Luminosity     1941 non-null int64
Maximum_of_Luminosity     1941 non-null int64
Length_of_Conveyer        1941 non-null int64
Steel_Plate_Thickness     1941 non-null int64
Edges_Index               1941 non-null float64
Empty_Index               1941 non-null float64
Square_Index              1941 non-null float64
Outside_X_Index           1941 non-null float64
Edges_X_Index             1941 non-null float64
Edges_Y_Index             1941 non-null float64
Outside_Global_Index      1941 non-null float64
LogOfAreas                1941 non-null float64
Log_X_Index               1941 non-null float64
Log_Y_Index               1941 non-null float64
Orientation_Index         1941 non-null float64
Luminosity_Index          1941 non-null float64
SigmoidOfAreas            1941 non-null float64
SteelType                 1941 non-null int64
Fault                     1941 non-null int64
dtypes: float64(13), int64(14)
memory usage: 409.5 KB
```

결측치 확인

In [9]:

```
data.isnull().sum()
```

Out[9]:

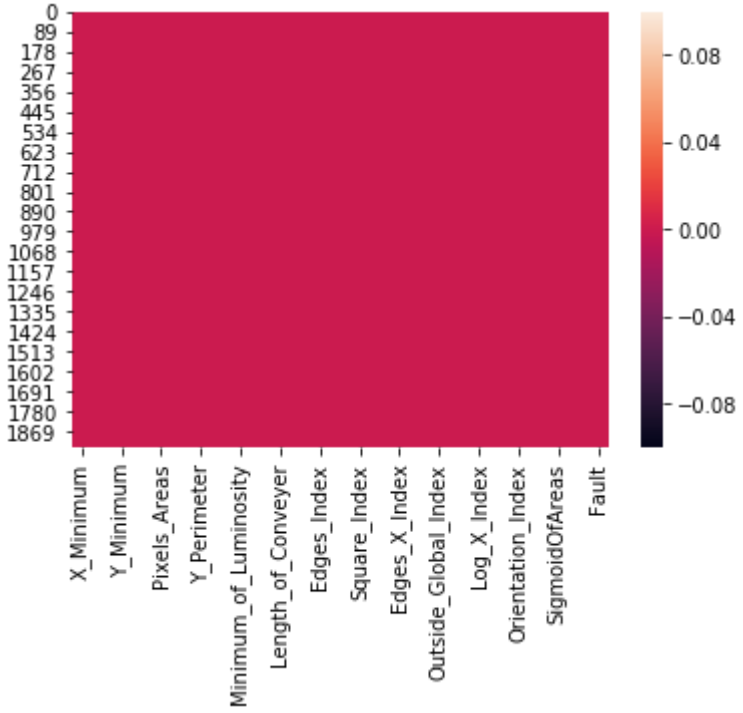
```
X_Minimum               0
X_Maximum               0
Y_Minimum               0
Y_Maximum               0
Pixels_Areas            0
X_Perimeter             0
Y_Perimeter             0
Sum_of_Luminosity       0
Minimum_of_Luminosity   0
Maximum_of_Luminosity   0
Length_of_Conveyer      0
Steel_Plate_Thickness   0
Edges_Index             0
Empty_Index             0
Square_Index            0
Outside_X_Index         0
Edges_X_Index           0
Edges_Y_Index           0
Outside_Global_Index    0
LogOfAreas              0
Log_X_Index             0
Log_Y_Index             0
Orientation_Index       0
Luminosity_Index        0
SigmoidOfAreas          0
SteelType               0
Fault                   0
dtype: int64
```

In [11]:

```
sns.heatmap(data.isnull())
```

Out[11]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c7febc12e8>



Boxplot을 통한 outlier

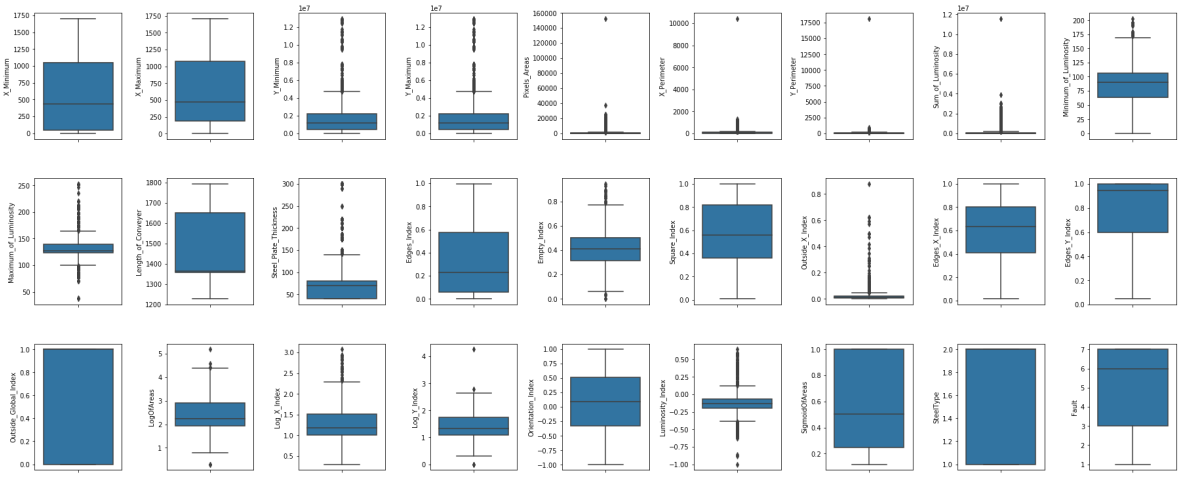
In [18]:

```
for k, v in data.items():
    print(k)
```

```
X_Minimum
X_Maximum
Y_Minimum
Y_Maximum
Pixels_Areas
X_Perimeter
Y_Perimeter
Sum_of_Luminosity
Minimum_of_Luminosity
Maximum_of_Luminosity
Length_of_Conveyer
Steel_Plate_Thickness
Edges_Index
Empty_Index
Square_Index
Outside_X_Index
Edges_X_Index
Edges_Y_Index
Outside_Global_Index
LogOfAreas
Log_X_Index
Log_Y_Index
Orientation_Index
Luminosity_Index
SigmoidOfAreas
SteelType
Fault
```

In [24]:

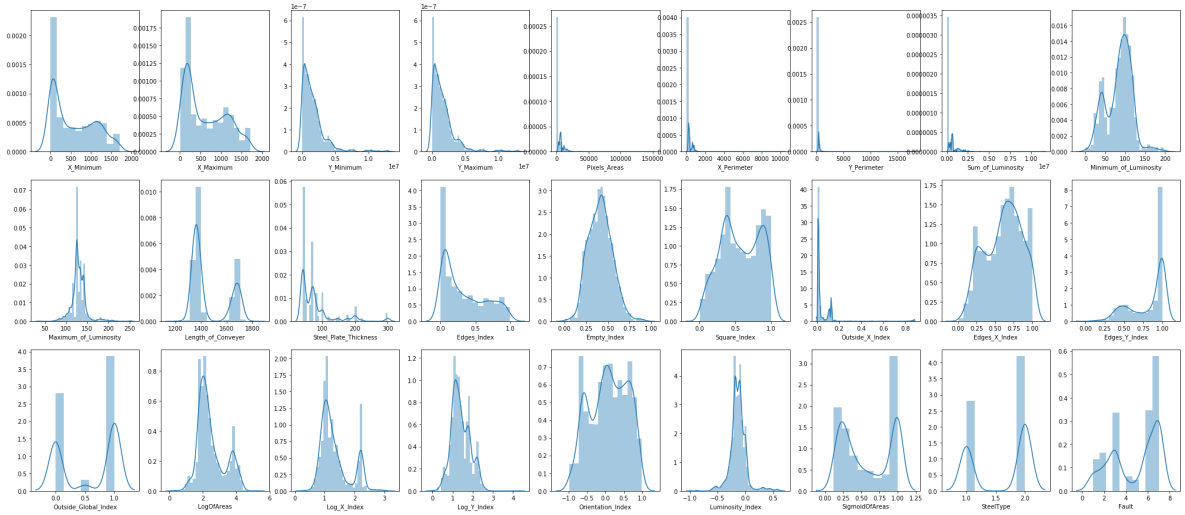
```
f, ax = plt.subplots(3, 9 , figsize = (25,10))
idx = 0
for i in range(0,3):
    for j in range(0,9):
        sns.boxplot(data.iloc[:,idx], orient='v', ax = ax[i][j])
        idx += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Check Distribution

In [25]:

```
f,ax = plt.subplots(3, 9, figsize = (35,15))
k = 0
for i in range(0,3):
    for j in range(0,9):
        sns.distplot(data[data.columns[k]], ax = ax[i,j])
        k += 1
```



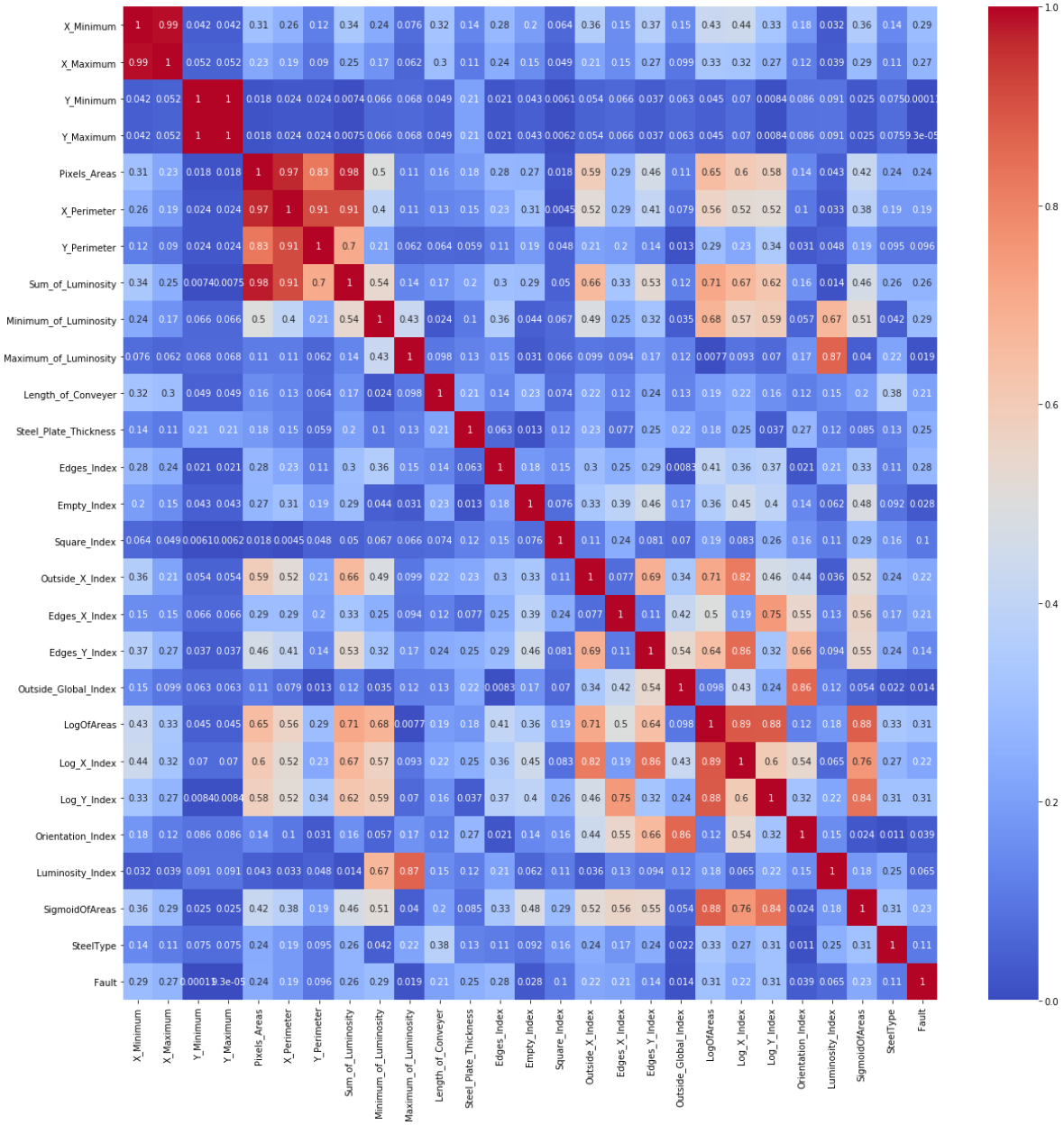
Check Correalation

In [31]:

```
f, ax = plt.subplots(1,1, figsize = (20,20))
sns.heatmap(data.corr().abs(), cmap = 'coolwarm', annot = True, ax = ax)
```

Out[31]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c78743a0b8>



(X_Maximum, X_Minimum), (Y_Maximum, Y_Minimum), (Pixels_Areas, X_Perimeter, Y_Perimeter, Sum_of_Luminosity), (LogOfAreas, Log_X_Index, Log_Y_Index)에 다중공선성 의심,
Fault와 상관계수가 가장 높은 하나씩의 변수를 제외하고 모두제거

In [32]:

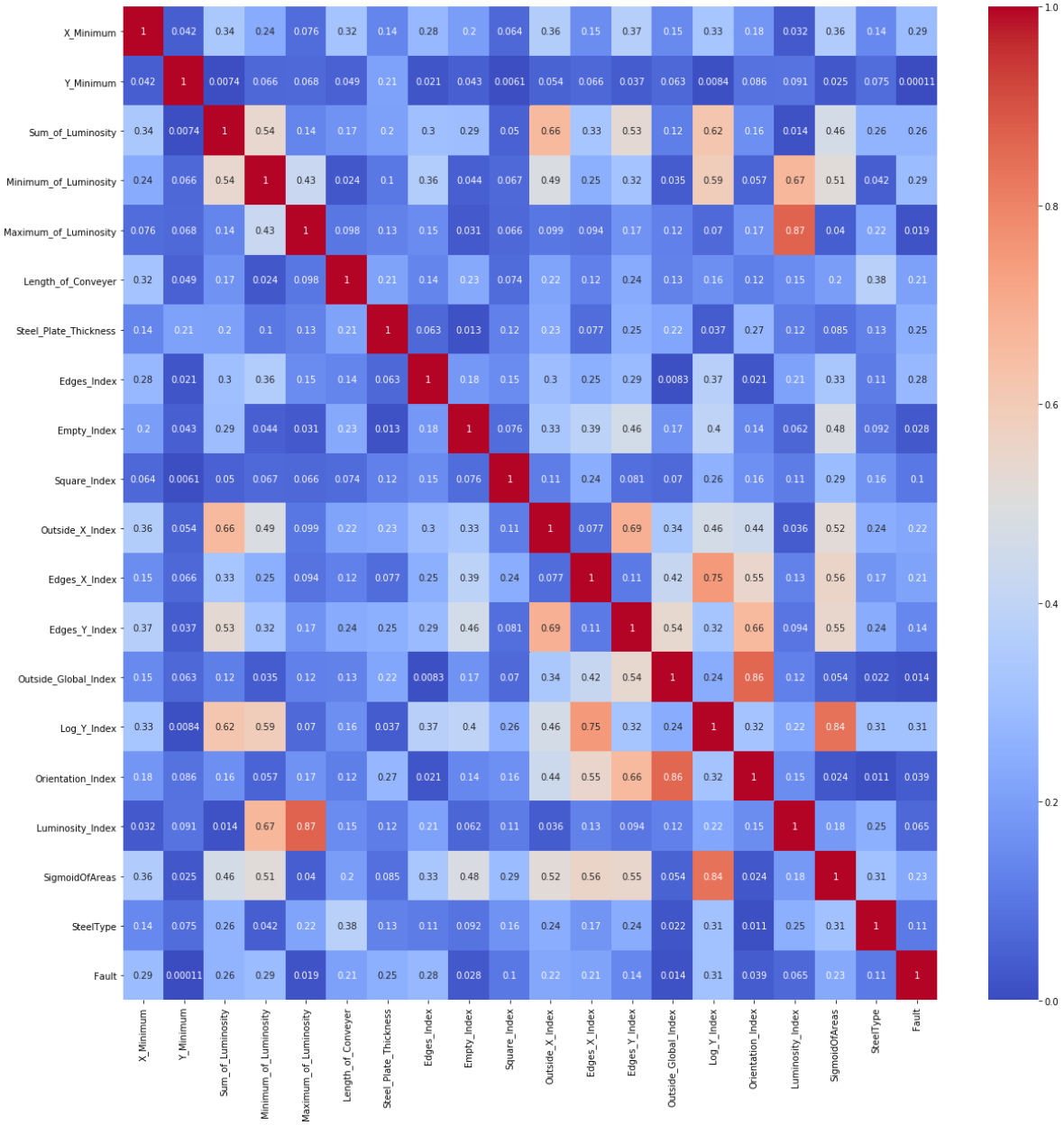
```
data2 = data.drop(['X_Maximum', 'Y_Maximum', 'Pixels_Areas', 'X_Perimeter', 'Y_Perimeter', 'LogOfAre
```

In [33]:

```
f, ax = plt.subplots(1,1, figsize = (20,20))
sns.heatmap(data2.corr().abs(), cmap = 'coolwarm', annot = True, ax = ax)
```

Out[33]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c785409940>



In [34]:

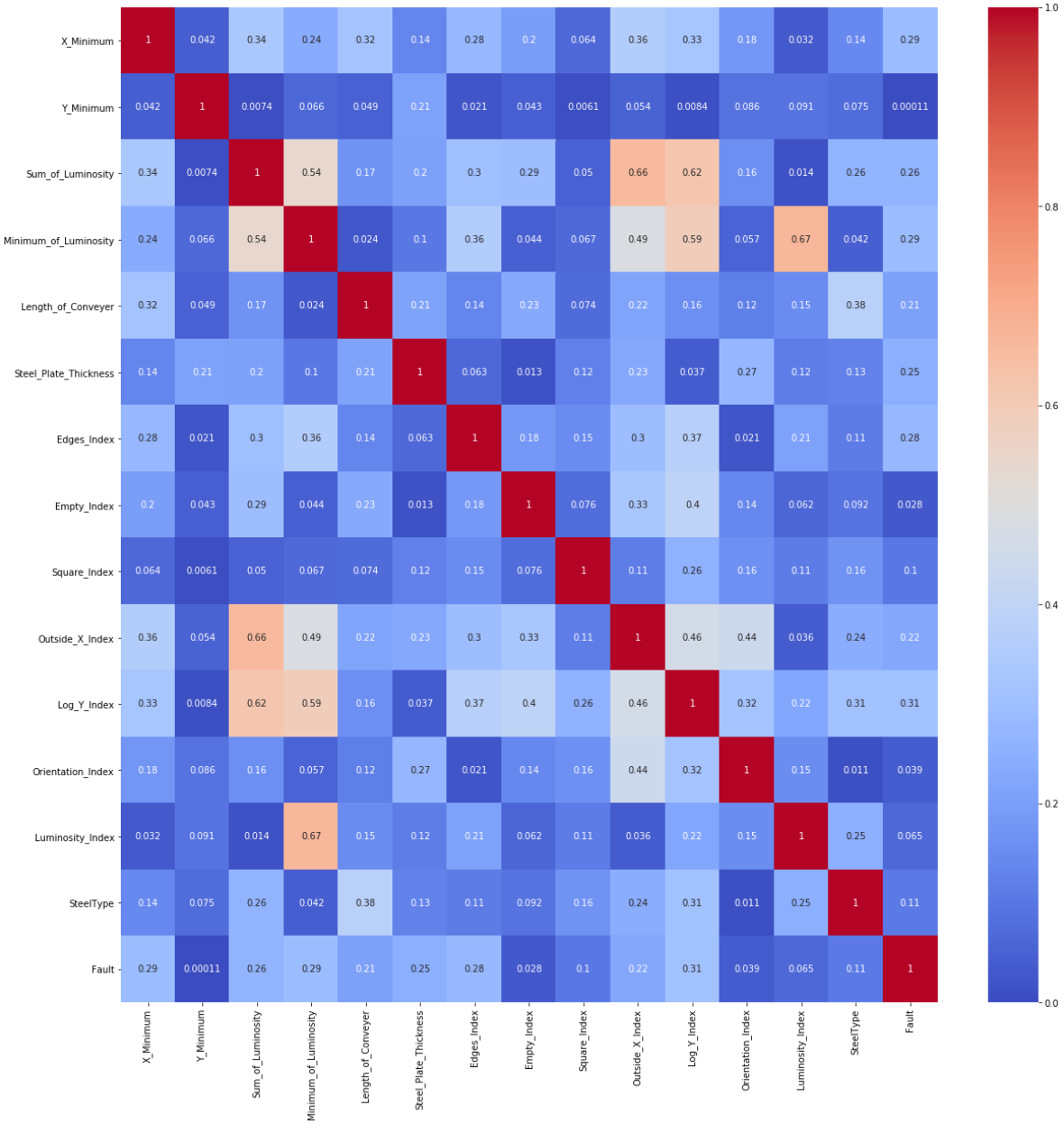
```
data2 = data2.drop(['Maximum_of_Luminosity', 'Outside_Global_Index', 'SigmoidOfAreas','Edges_X_Inde
```

In [35]:

```
f, ax = plt.subplots(1,1, figsize = (20,20))
sns.heatmap(data2.corr().abs(), cmap = 'coolwarm', annot = True, ax = ax)
```

Out[35]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c78536c668>



-제거된 데이터를 토대로 PCA 수행 및, PCA변수 추출

In [38]:

```
for_pca_data =x data[['Maximum_of_Luminosity', 'Outside_Global_Index', 'SigmoidOfAreas','Edges_X_Inc
```

In [41]:

```
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
```

In [43]:

```
for_pca_data = pd.DataFrame(std.fit_transform(for_pca_data))
```

```
C:₩Anaconda₩lib₩site-packages₩sklearn₩preprocessing₩data.py:645: DataConversionWarni
ng: Data with input dtype int64, float64 were all converted to float64 by StandardSc
aler.
  return self.partial_fit(X, y)
C:₩Anaconda₩lib₩site-packages₩sklearn₩base.py:464: DataConversionWarning: Data with
input dtype int64, float64 were all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [52]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components = 6)
pca.fit(for_pca_data)
```

Out[52]:

```
PCA(copy=True, iterated_power='auto', n_components=6, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [54]:

```
pca.explained_variance_ratio_[0:6].sum()
```

Out[54]:

```
0.9304411604757159
```

In [59]:

```
pca_components = pd.DataFrame(pca.fit_transform(for_pca_data), columns=['pca1','pca2','pca3','pca4'
```

In [60]:

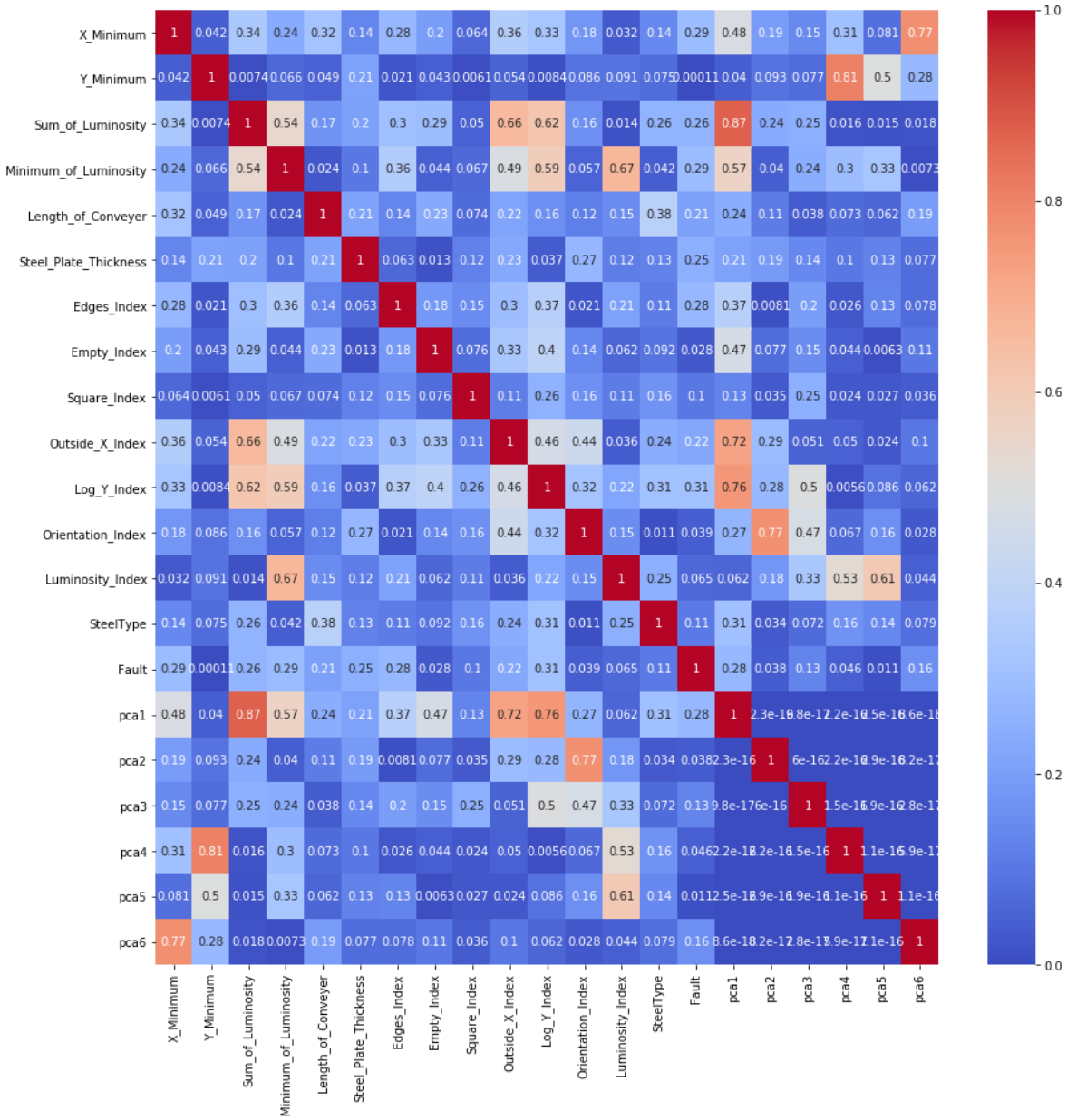
```
data3 = pd.concat([data2, pca_components], axis = 1)
```

In [64]:

```
f, ax = plt.subplots(1,1, figsize = (15,15))
sns.heatmap(data3.corr().abs(), cmap = 'coolwarm', annot =True, ax= ax)
```

Out[64]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c787251978>



In [66]:

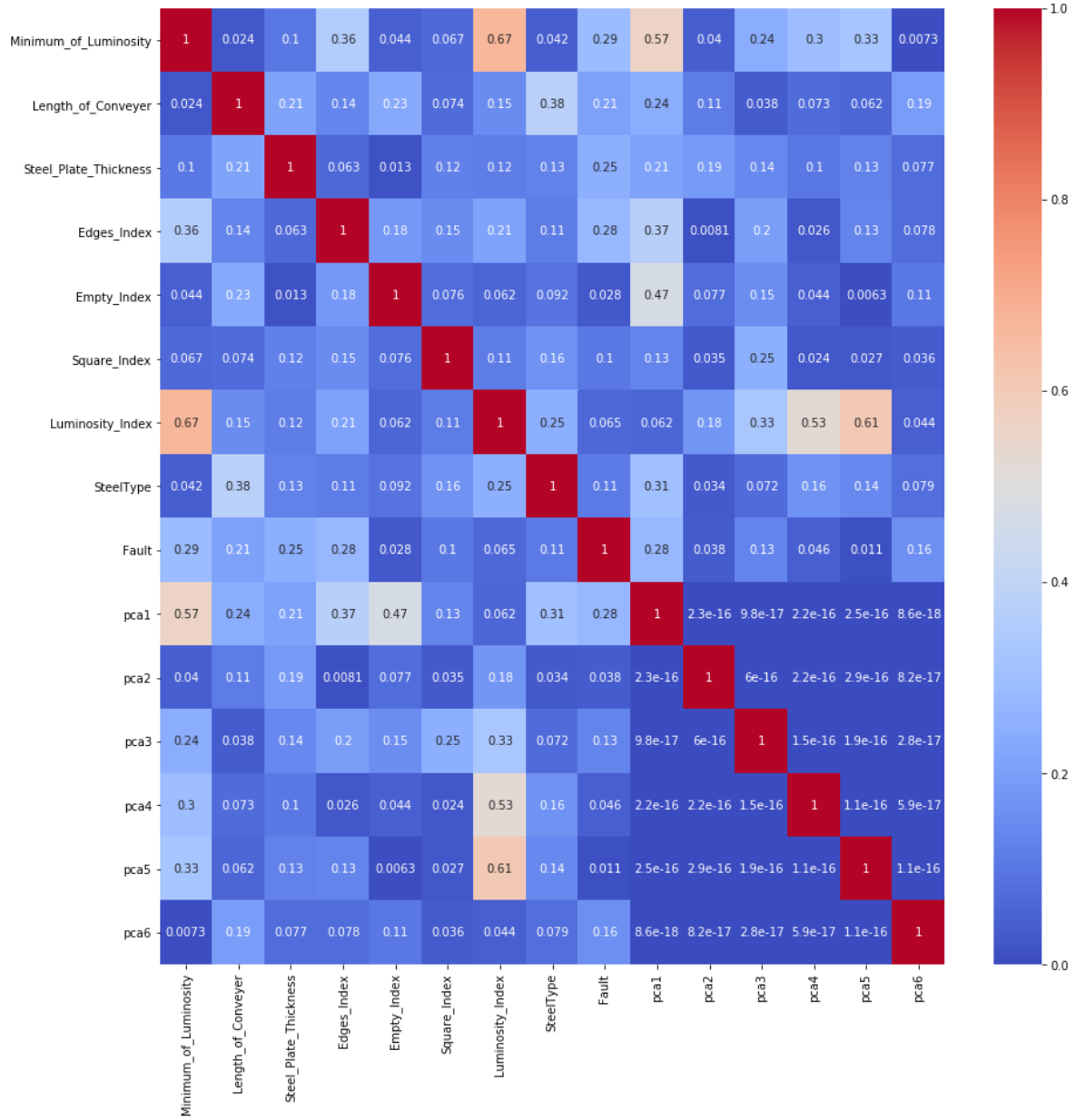
```
data4 = data3.drop(['X_Minimum','Y_Minimum','Sum_of_Luminosity','Orientation_Index', 'Log_Y_Index',
```

In [67]:

```
f, ax = plt.subplots(1,1, figsize = (15,15))
sns.heatmap(data4.corr().abs(), cmap = 'coolwarm', annot =True, ax= ax)
```

Out[67]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c788f883c8>



이제 공선성이 매우 강한 친구들은 보이지않는다.

② 전체 데이터를 Train, Validataion, Test 용도로 분할하고 시각화하시오. (각 비율: 50%, 30%, 20%)

In [71]:

```
X = data4.drop('Fault', axis = 1)
y = data4['Fault']
```

In [76]:

```
train_X.shape, train_Y.shape, test_X.shape, test_Y.shape
```

Out[76]:

```
((970, 14), (971, 14), (970,), (971,))
```

In [153]:

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split(X, y, test_size = 0.5, stratify = y)
valid_X, test_X, valid_Y, test_Y = train_test_split(test_X, test_Y, test_size= 0.4, stratify = test_
```

In [82]:

```
print(train_X.shape, valid_X.shape, test_X.shape)
```

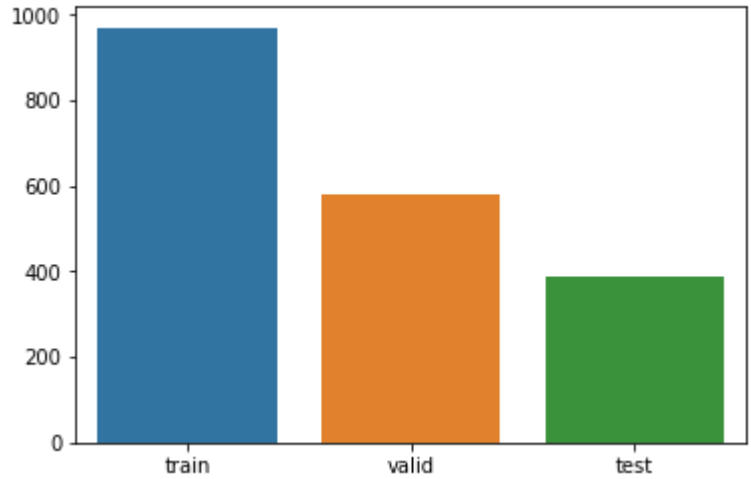
```
(970, 14) (582, 14) (389, 14)
```

In [83]:

```
sns.barplot(x = ['train','valid','test'], y = [970, 582, 389])
```

Out[83]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c786ba55f8>
```

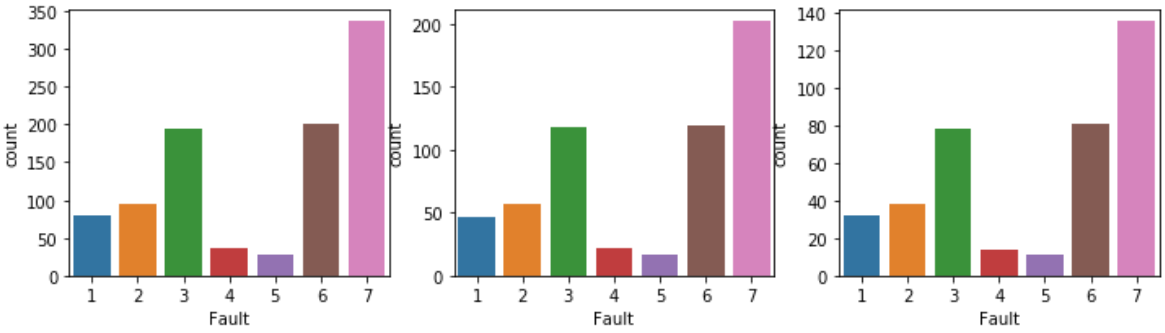


In [85]:

```
f, ax = plt.subplots(1,3, figsize = (12, 3))
sns.countplot(train_Y, ax = ax[0])
sns.countplot(valid_Y, ax = ax[1])
sns.countplot(test_Y, ax = ax[2])
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c7892f8438>
```



---

③ 불량코드 1에 대하여, Logistic Regression을 활용하여 이항분류 모형을 생성하시오.

생성한 모형에 대한 최적의 Cut-Off Value를 선정 후, Confusion Matrix를 제시하시오. (반드시 시각화와 통계량을 포함시킬 것)

In [101]:

```
train_Y.replace([2,3,4,5,6,7], 0, inplace =  True)
valid_Y.replace([2,3,4,5,6,7], 0, inplace =  True)
test_Y.replace([2,3,4,5,6,7], 0, inplace =  True)
```

In [102]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(train_X, train_Y)
```

```
C:₩Wanaconda₩lib₩site-packages₩sklearn₩linear_model₩logistic.py:433: FutureWarning: D
efault solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this w
arning.
  FutureWarning)
```

Out[102]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [118]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_auc_score
```

In [112]:

```
y_pred = lr.predict(valid_X)
print(accuracy_score(valid_Y, y_pred))
print(classification_report(valid_Y, y_pred))
```

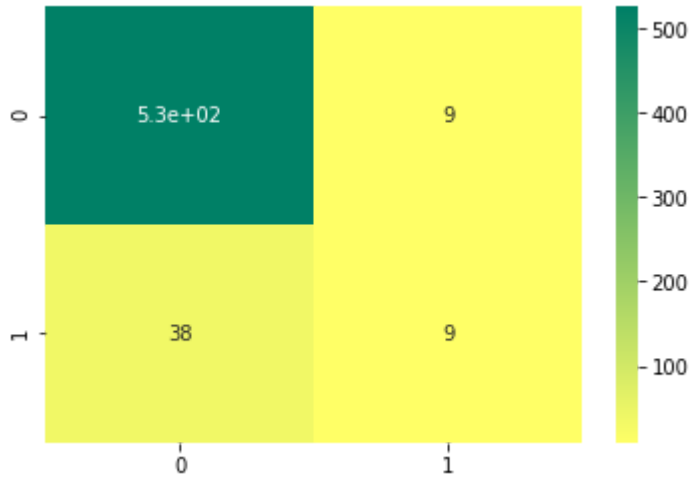
```
0.9192439862542955
              precision    recall  f1-score   support

           0       0.93      0.98      0.96       535
           1       0.50      0.19      0.28        47

   micro avg       0.92      0.92      0.92       582
   macro avg       0.72      0.59      0.62       582
weighted avg       0.90      0.92      0.90       582
```

In [115]:

```
sns.heatmap(confusion_matrix(valid_Y, y_pred), annot = True, cmap = 'summer_r')
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c78b046400>
```



In [105]:

```
lr.predict_proba(valid_X)
```

Out[105]:

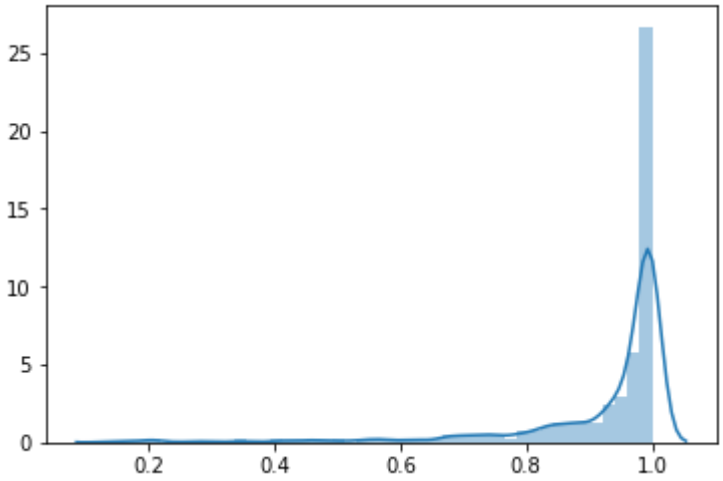
```
array([[0.74396493, 0.25603507],
       [0.99744919, 0.00255081],
       [0.78764647, 0.21235353],
       ...,
       [0.79574669, 0.20425331],
       [0.97497305, 0.02502695],
       [0.40090049, 0.59909951]])
```

In [106]:

```
sns.distplot(lr.predict_proba(valid_X)[:,0])
```

Out[106]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c78a5af898>
```



In [94]:

```
pred_proba = lr.predict_proba(valid_X)[:,0]
```

In [122]:

```
ranges = np.arange(0.0, 1.0, 0.01)
accuracy_ls = []
auc_ls  = []
for r in ranges:
    y_pred = np.where(pred_proba > r, 1, 0)
    accuracy_ls.append(accuracy_score(valid_Y, y_pred))
    auc_ls.append(roc_auc_score(valid_Y, y_pred))
```

In [123]:

```
print(max(accuracy_ls), accuracy_ls.index(max(accuracy_ls)), ranges[ accuracy_ls.index(max(accuracy_
```

```
0.9243986254295533 56 0.56
```

In [125]:

```
print(max(auc_ls), auc_ls.index(max(auc_ls)), ranges[auc_ls.index(max(auc_ls))])
```

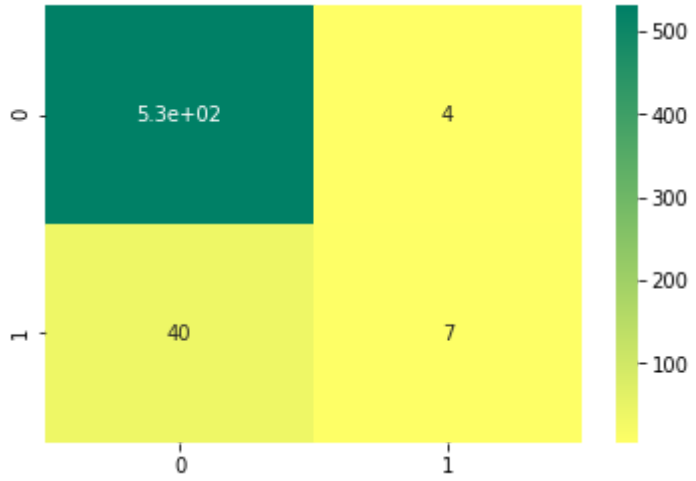
```
0.8695963412209186 11 0.11
```

In [126]:

```
y_pred = np.where(pred_proba > 0.56, 1, 0)
sns.heatmap(confusion_matrix(valid_Y, y_pred), annot = True, cmap = 'summer_r')
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c78b1c5a90>
```

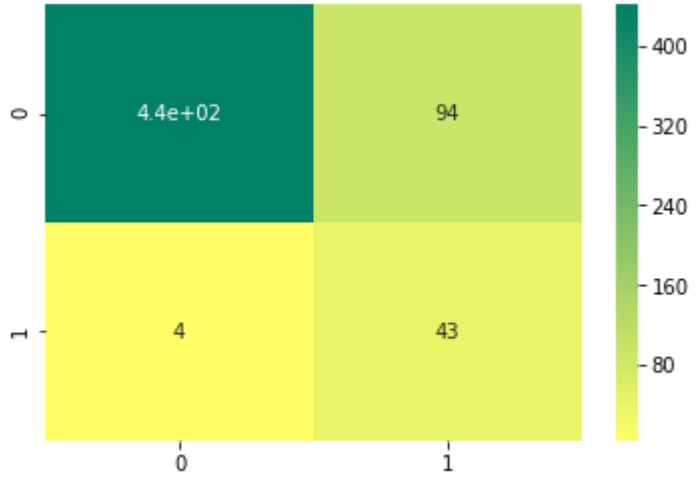


In [127]:

```
y_pred = np.where(pred_proba > 0.11, 1, 0)
sns.heatmap(confusion_matrix(valid_Y, y_pred), annot = True, cmap = 'summer_r')
```

Out[127]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c78b061630>
```

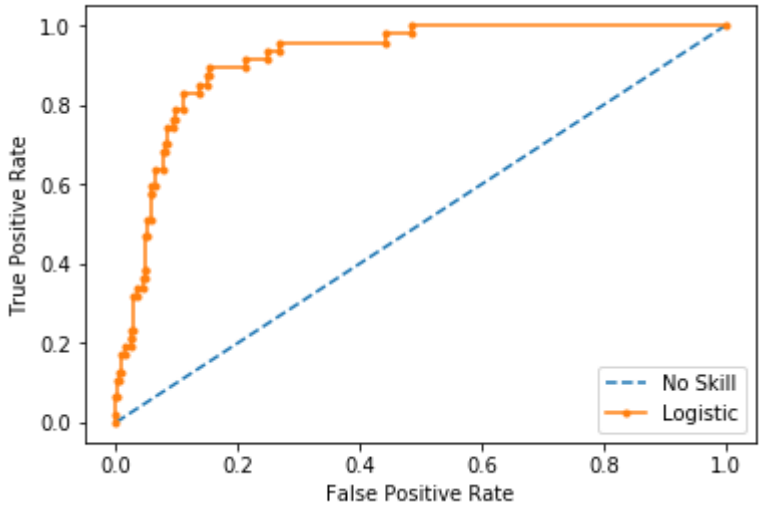


In [131]:

```
yhat = lr.predict_proba(valid_X)
# retrieve just the probabilities for the positive class
pos_probs = yhat[:, 1]
```

In [133]:

```
from sklearn.metrics import roc_curve
plt.plot([0, 1], [0, 1], linestyle='--', label='No Skill')
# calculate roc curve for model
fpr, tpr, _ = roc_curve(valid_Y, pos_probs)
# plot model roc curve
plt.plot(fpr, tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```



In [148]:

```
y_pred = np.where(lr.predict_proba(test_X)[:,1]>0.11, 1, 0)
print(accuracy_score(test_Y, y_pred), roc_auc_score(test_Y, y_pred))
```

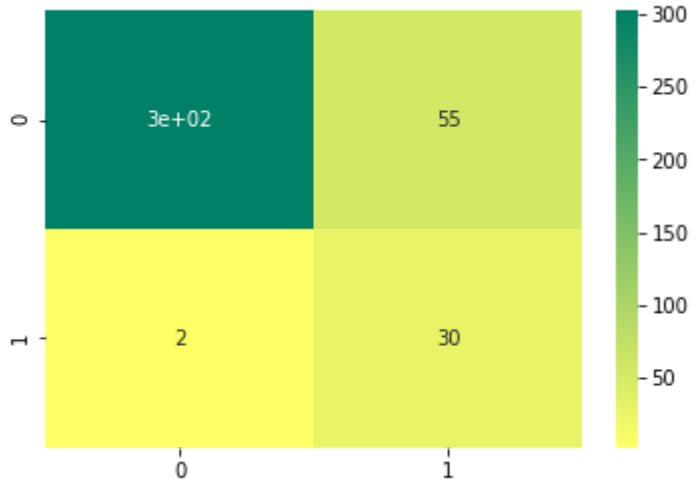
0.8534704370179949 0.89171918767507

In [150]:

```
sns.heatmap(confusion_matrix(test_Y, y_pred), cmap = 'summer_r', annot= True)
```

Out[150]:

<matplotlib.axes._subplots.AxesSubplot at 0x1c78b8c4ac8>



---

④ Logistic Regression을 제외하고 SVM을 포함하여 3가지 다항 분류 모형을 만들어 Precision과 Sensitivity(TPR)를 제시하시오.

또한 모델향상과정과 최적화 과정을 통해 Confusion Matrix를 도출하시오.

In [154]:

```
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [155]:

```
svc = SVC()
rf =  RandomForestClassifier()
knn = KNeighborsClassifier()
```

In [157]:

```
rf.fit(train_X, train_Y)
print(classification_report(valid_Y, rf.predict(valid_X)))
print(accuracy_score(valid_Y, rf.predict(valid_X)))
```

```
              precision    recall  f1-score   support

           1       0.62      0.43      0.51        47
           2       0.88      0.88      0.88        57
           3       0.95      0.97      0.96       118
           4       0.91      0.91      0.91        22
           5       1.00      0.75      0.86        16
           6       0.66      0.71      0.68       120
           7       0.70      0.72      0.71       202

   micro avg       0.77      0.77      0.77       582
   macro avg       0.82      0.77      0.79       582
weighted avg       0.77      0.77      0.77       582

0.7680412371134021
```

In [158]:

```
svc.fit(train_X, train_Y)
print(classification_report(valid_Y, svc.predict(valid_X)))
print(accuracy_score(valid_Y, svc.predict(valid_X)))
```

```
C:₩Anaconda₩lib₩site-packages₩sklearn₩svm₩base.py:196: FutureWarning: The default va
lue of gamma will change from 'auto' to 'scale' in version 0.22 to account better fo
r unscaled features. Set gamma explicitly to 'auto' or 'scale' to avoid this warnin
g.
  "avoid this warning.", FutureWarning)
              precision    recall  f1-score   support

           1       0.75      0.13      0.22        47
           2       0.83      0.79      0.81        57
           3       0.99      0.87      0.93       118
           4       0.91      0.91      0.91        22
           5       0.86      0.75      0.80        16
           6       0.80      0.39      0.53       120
           7       0.56      0.90      0.69       202

   micro avg       0.71      0.71      0.71       582
   macro avg       0.81      0.68      0.70       582
weighted avg       0.76      0.71      0.69       582

0.711340206185567
```

In [160]:

```
knn.fit(train_X, train_Y)
print(classification_report(valid_Y, knn.predict(valid_X)))
print(accuracy_score(valid_Y, knn.predict(valid_X)))
```

```
              precision    recall  f1-score   support

           1       0.30      0.23      0.26        47
           2       0.64      0.98      0.77        57
           3       0.88      0.97      0.92       118
           4       0.78      0.95      0.86        22
           5       0.53      1.00      0.70        16
           6       0.58      0.53      0.56       120
           7       0.64      0.51      0.57       202

   micro avg       0.66      0.66      0.66       582
   macro avg       0.62      0.74      0.66       582
weighted avg       0.65      0.66      0.65       582

0.6615120274914089
```

default에 대해 가장 성능이 좋은 randomforetst에 대해서 gridsearch를 수행 하겠습니다.

In [163]:

```
from sklearn.model_selection import GridSearchCV
n_estimators = [100,500,1000]
max_depth = [5,10]
min_samples_split = [5,10]
param_grid = {'n_estimators':n_estimators, 'max_depth': max_depth, 'min_samples_split' :min_samples_
grid = GridSearchCV(rf, param_grid=param_grid, cv = 5, verbose = 1)
```

In [165]:

```
grid.fit(train_X, train_Y)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:  2.4min finished
```

Out[165]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion
='gini',
            max_depth=None, max_features='auto', 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=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [100, 500, 1000], 'max_depth': [5, 10], 'min_samp
les_split': [5, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [166]:

```
grid.best_params_
```

Out[166]:

```
{'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 1000}
```

In [167]:

```
y_pred = grid.best_estimator_.predict(valid_X)
print(classification_report(valid_Y, y_pred))
print(accuracy_score(valid_Y, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.79      0.32      0.45        47
           2       0.91      0.89      0.90        57
           3       0.98      0.96      0.97       118
           4       0.91      0.91      0.91        22
           5       1.00      0.88      0.93        16
           6       0.72      0.59      0.65       120
           7       0.66      0.84      0.74       202

   micro avg       0.78      0.78      0.78       582
   macro avg       0.85      0.77      0.79       582
weighted avg       0.79      0.78      0.77       582

0.7800687285223368
```

---

⑤ 상기 ③번과 ④번 4가지 모형 중 1가지를 선택하여 최적의 클러스터링 개수(단일집단~5개)를 제시하시오.

또한 군집분석을 이용한 모형성능 향상 과정을 수행하여, 성능향상 전후의 F1 Score와 모형 평가 결과를 제시하시오.

In [171]:

```
from sklearn.cluster import KMeans
```

In [172]:

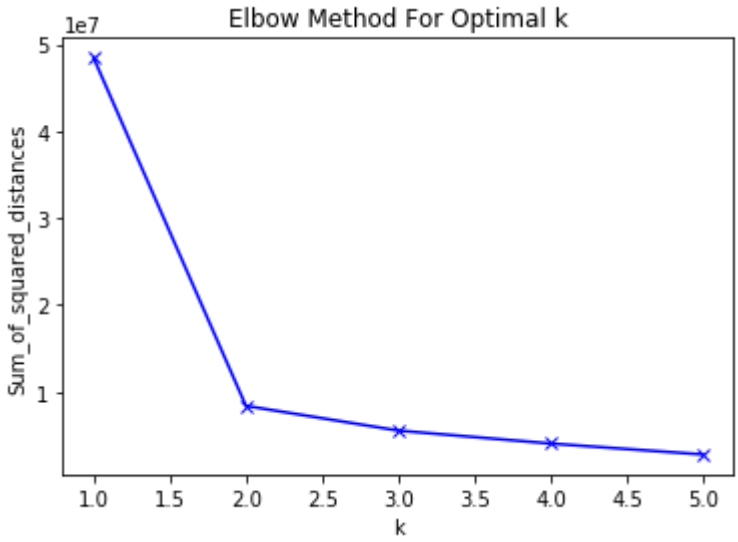
```
ranges = range(1,6)
```

In [182]:

```
for_scree = []
for r in ranges:
    km = KMeans(n_clusters=r)
    km.fit(X)
    for_scree.append(km.inertia_)
```

In [183]:

```
plt.plot(ranges, for_scree, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



In [187]:

```
km = KMeans(n_clusters=2)
km.fit(X)
```

Out[187]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

In [194]:

```
X2 = pd.concat([X,pd.DataFrame(km.labels_, columns = ['clu1'])], axis = 1)
```

In [208]:

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split(X2, y, test_size = 0.5, stratify = y)
valid_X, test_X, valid_Y, test_Y = train_test_split(test_X, test_Y, test_size= 0.4, stratify = test_
```

In [209]:

```
from sklearn.model_selection import GridSearchCV
n_estimators = [100,500,1000]
max_depth = [5,10]
min_samples_split = [5,10]
param_grid = {'n_estimators':n_estimators, 'max_depth': max_depth, 'min_samples_split' :min_samples_
grid = GridSearchCV(rf, param_grid=param_grid, cv = 5, verbose = 1)
```

In [210]:

```
grid.fit(train_X, train_Y)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:  1.3min finished
```

Out[210]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion
='gini',
            max_depth=None, max_features='auto', 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=10, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [100, 500, 1000], 'max_depth': [5, 10], 'min_samp
les_split': [5, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=1)
```

In [211]:

```
y_pred = grid.best_estimator_.predict(valid_X)
print(classification_report(valid_Y, y_pred))
print(accuracy_score(valid_Y, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.81      0.47      0.59        47
           2       0.88      0.91      0.90        57
           3       0.97      0.95      0.96       118
           4       1.00      0.91      0.95        22
           5       1.00      0.62      0.77        16
           6       0.72      0.65      0.68       120
           7       0.66      0.79      0.72       202

   micro avg       0.78      0.78      0.78       582
   macro avg       0.86      0.76      0.80       582
weighted avg       0.79      0.78      0.78       582

0.7783505154639175
```

In [212]:

```
y_pred = grid.best_estimator_.predict(test_X)
print(classification_report(test_Y, y_pred))
print(accuracy_score(test_Y, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.72      0.41      0.52        32
           2       0.91      0.79      0.85        38
           3       0.97      0.92      0.95        78
           4       1.00      0.86      0.92        14
           5       1.00      0.73      0.84        11
           6       0.72      0.70      0.71        81
           7       0.68      0.83      0.75       135

   micro avg       0.78      0.78      0.78       389
   macro avg       0.86      0.75      0.79       389
weighted avg       0.79      0.78      0.78       389

0.781491002570694
```

In [213]:

```
sns.heatmap(confusion_matrix(test_Y, y_pred), annot = True, cmap = 'summer_r')
```

Out[213]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c78b807400>
```