Classification of Rock/Mine with KNN

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

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

```
from google.colab import files
data_to_load = files.upload()
```

```
Choose Files ROCK_OR_MINE.csv
• ROCK_OR_MINE.csv(text/csv) - 87776 bytes, last modified: 12/17/2022 - 100% done
Saving ROCK_OR_MINE.csv to ROCK_OR_MINE (1).csv
```

```
df = pd.read_csv('ROCK_OR_MINE.csv', header=None)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.28
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.61
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.12
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.44

5 rows × 61 columns

```
df.columns
```

```
Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
            17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
            34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
            51, 52, 53, 54, 55, 56, 57, 58, 59, 60],
           dtype='int64')
```

```
df = df.rename(columns={60: 'Label'})
```

```
df.columns
```

```
Index([      0,       1,       2,       3,       4,       5,       6,       7,
              8,       9,      10,      11,      12,      13,      14,      15,
             16,      17,      18,      19,      20,      21,      22,      23,
             24,      25,      26,      27,      28,      29,      30,      31,
             32,      33,      34,      35,      36,      37,      38,      39,
             40,      41,      42,      43,      44,      45,      46,      47,
             48,      49,      50,      51,      52,      53,      54,      55,
             56,      57,      58,      59, 'Label'],
      dtype='object')
```

Data Cleaning

```
df.isna().sum()
```

```
df.isna().sum().sum()
```

```
df.isna().sum().sum()
```

```
0
```

Exploratory Data Analysis

Ika Lulus yuliatin

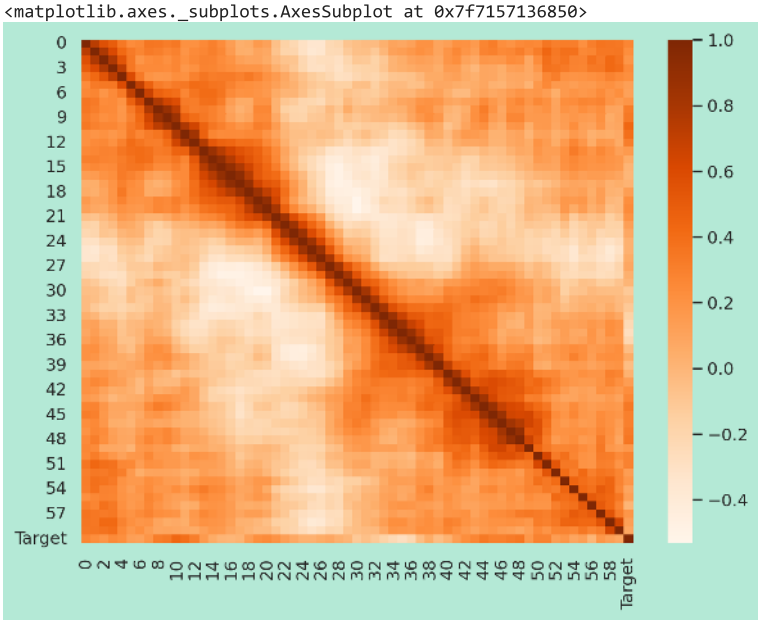
There is war is going on between two countries submarine of the country is going under the water to another country and enemy country planted some mines in the oceans mine are nothing but explosive that explodes when some object comes in contact with it and there can be rocks in the ocean so submarine needs to predict whether it is crossing mine or rock our job is to make a system that can predict whether the object beneath the submarine is a mine or a rock so how this is done is submarine uses sonar signal that sends sound and receives switchbacks so this signal in the processed to detect whether the object is a mine or it's just a rock in the ocean to predict the rock and mine

```
df['Target'] = df['Label'].map({"R":0, "M":1})
```

```
sns.set(rc={"axes.facecolor":"#b4e9d6","figure.facecolor":"#b4e9d6"})
pallet = ["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"]
cmap = colors.ListedColormap(["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"])
```

```
plt.figure(figsize=(8,6),dpi=200)
sns.heatmap(df.corr(), cmap='Oranges')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7157136850>
```

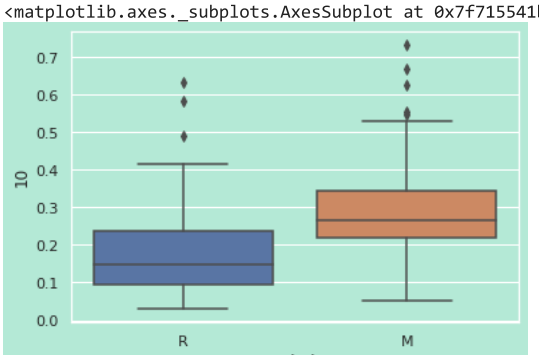


```
np.abs(df.corr()['Target']).sort_values().tail(6)
```

```
44        0.339406
9         0.341142
48        0.351312
11        0.392245
10        0.432855
Target    1.000000
Name: Target, dtype: float64
```

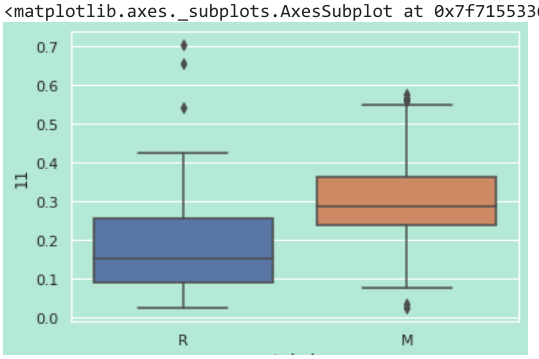
```
sns.boxplot(x=df['Label'],y=df[10])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f715541bd60>
```



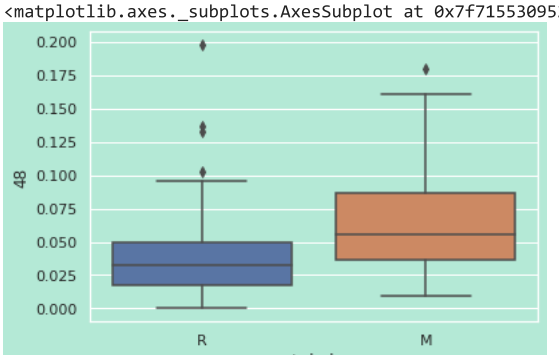

```
sns.boxplot(x=df['Label'],y=df[11])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7155336700>
```




```
sns.boxplot(x=df['Label'],y=df[48])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7155309520>
```




```
df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	ma
0	208.0	0.029164	0.022991	0.0015	0.013350	0.02280	0.035550	0.137
1	208.0	0.038437	0.032960	0.0006	0.016450	0.03080	0.047950	0.233
2	208.0	0.043832	0.038428	0.0015	0.018950	0.03430	0.057950	0.305
3	208.0	0.053892	0.046528	0.0058	0.024375	0.04405	0.064500	0.426
4	208.0	0.075202	0.055552	0.0067	0.038050	0.06250	0.100275	0.401
...	...	...	...	...	...	...	...	..
56	208.0	0.007820	0.005785	0.0003	0.003700	0.00595	0.010425	0.035

Split The Data

```
df.columns
```

```
Index([       0,        1,        2,        3,        4,        5,        6,
               7,        8,        9,       10,       11,       12,       13,
              14,       15,       16,       17,       18,       19,       20,
              21,       22,       23,       24,       25,       26,       27,
              28,       29,       30,       31,       32,       33,       34,
              35,       36,       37,       38,       39,       40,       41,
              42,       43,       44,       45,       46,       47,       48,
              49,       50,       51,       52,       53,       54,       55,
              56,       57,       58,       59,  'Label', 'Target'],
      dtype='object')
```

```
X = df.drop(['Label','Target'],axis=1)
y = df['Label']
```

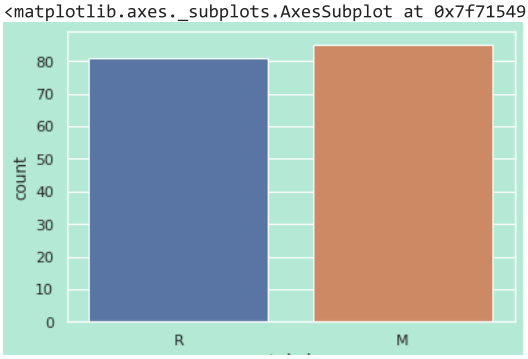
```
X.columns
```

```
Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
       54, 55, 56, 57, 58, 59],
      dtype='object')
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
sns.countplot(x=y_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71549534c0>
```




```
y_train.value_counts()
```

```
M    85
R    81
Name: Label, dtype: int64
```

Modelling with KNN

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier


scaler = StandardScaler()


knn = KNeighborsClassifier()


from sklearn.pipeline import Pipeline


pipe = Pipeline([('scaler',scaler), ('knn',knn)])


from sklearn.model_selection import GridSearchCV


parameters = ({'knn__n_neighbors': list(range(1,30)),
               'knn__weights': ['uniform', 'distance'],
               'knn__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']})


grid = GridSearchCV(estimator=pipe,param_grid=parameters,scoring='accuracy',cv=5)


grid.fit(X_train,y_train)
```

```
     GridSearchCV(cv=5,
                  estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                            ('knn', KNeighborsClassifier())]),
                  param_grid={'knn__algorithm': ['auto', 'ball_tree', 'kd_tree',
                                                 'brute'],
                              'knn__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
                                                   12, 13, 14, 15, 16, 17, 18, 19,
                                                   20, 21, 22, 23, 24, 25, 26, 27,
                                                   28, 29],
                              'knn__weights': ['uniform', 'distance']},
                  scoring='accuracy')
```

```
grid.best_estimator_
```

```
     Pipeline(steps=[('scaler', StandardScaler()),
                     ('knn', KNeighborsClassifier(n_neighbors=1))])
```

```
grid.best_estimator_.get_params()
```

```
     {'memory': None,
      'steps': [('scaler', StandardScaler()),
       ('knn', KNeighborsClassifier(n_neighbors=1))],
      'verbose': False,
      'scaler': StandardScaler(),
      'knn': KNeighborsClassifier(n_neighbors=1),
      'scaler__copy': True,
      'scaler__with_mean': True,
      'scaler__with_std': True,
      'knn__algorithm': 'auto',
      'knn__leaf_size': 30,
      'knn__metric': 'minkowski',
      'knn__metric_params': None,
      'knn__n_jobs': None,
      'knn__n_neighbors': 1,
      'knn__p': 2,
      'knn__weights': 'uniform'}
```

Cross Validation Results

```
cv_results = pd.DataFrame(grid.cv_results_)


cv_results.info()
```

```
     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 232 entries, 0 to 231
     Data columns (total 16 columns):
      #   Column                 Non-Null Count  Dtype
     ---  ------                 --------------  -----
      0   mean_fit_time          232 non-null    float64
```

```
 1   std_fit_time            232 non-null    float64
 2   mean_score_time         232 non-null    float64
 3   std_score_time          232 non-null    float64
 4   param_knn__algorithm    232 non-null    object
 5   param_knn__n_neighbors  232 non-null    object
 6   param_knn__weights      232 non-null    object
 7   params                  232 non-null    object
 8   split0_test_score       232 non-null    float64
 9   split1_test_score       232 non-null    float64
 10  split2_test_score       232 non-null    float64
 11  split3_test_score       232 non-null    float64
 12  split4_test_score       232 non-null    float64
 13  mean_test_score         232 non-null    float64
 14  std_test_score          232 non-null    float64
 15  rank_test_score         232 non-null    int32
dtypes: float64(11), int32(1), object(4)
memory usage: 28.2+ KB
```

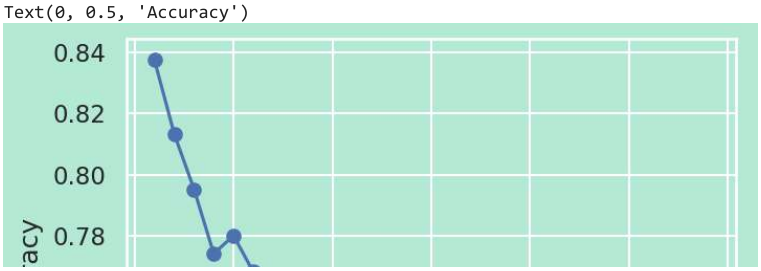
```
cv_score = cv_results.groupby('param_knn__n_neighbors').agg('mean')['mean_test_score']
```

```
cv_score
```

```
param_knn__n_neighbors
1     0.837433
2     0.813280
3     0.795187
4     0.774064
5     0.780036
6     0.768093
7     0.753119
8     0.737879
9     0.731729
10    0.713725
11    0.734759
12    0.722638
13    0.749911
14    0.734848
15    0.749733
16    0.744029
17    0.729144
18    0.723084
19    0.738146
20    0.726114
21    0.735294
22    0.720143
23    0.723084
24    0.708111
25    0.711230
26    0.696078
27    0.717112
28    0.714171
29    0.720143
Name: mean_test_score, dtype: float64
```

```
plt.figure(figsize=(5,4),dpi=150)
plt.plot(range(1,30),cv_score,'o-')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
```

```
Text(0, 0.5, 'Accuracy')
```



Final Model Evaluation

```
y_pred = grid.predict(X_test)
```

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

```
confusion_matrix(y_test,y_pred)
```

```
array([[24,  2],
       [ 2, 14]])
```

```
print(classification_report(y_test,y_pred))
```

```
                  precision    recall  f1-score   support

               M       0.92      0.92      0.92        26
               R       0.88      0.88      0.88        16

        accuracy                           0.90        42
       macro avg       0.90      0.90      0.90        42
    weighted avg       0.90      0.90      0.90        42
```

```
accuracy_score(y_test,y_pred)
```

```
0.9047619047619048
```

CONCLUSION

1. The best parameters of KNN estimator in this model are n_neighbors = 1, weights = 'uniform', and algorithm = 'auto'.
2. The model can predict the unseen data (X_test) quite good, with accuracy of 90.48%