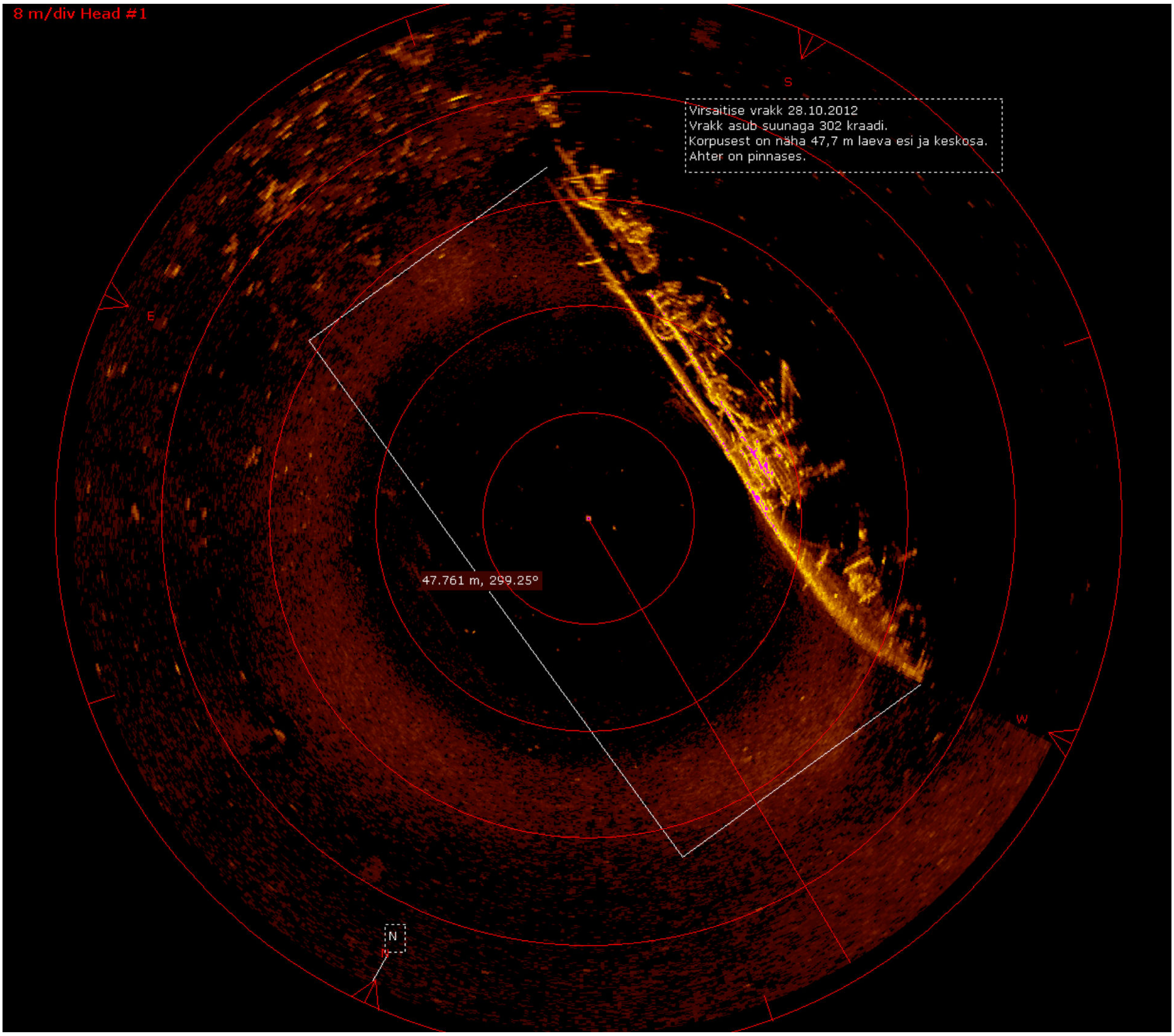
The Sonar Data

Detecting a Rock or a Mine

Sonar (sound navigation ranging) is a technique that uses sound propagation (usually underwater, as in submarine navigation) to navigate, communicate with or detect objects on or under the surface of the water, such as other vessels.



The data set contains the response metrics for 60 separate sonar frequencies sent out against a known mine field (and known rocks). These frequencies are then labeled with the known object they were beaming the sound at (either a rock or a mine).

Our main goal is to create a machine learning model capable of detecting the difference between a rock or a mine based on the response of the 60 separate sonar frequencies.

Data Source: [https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks) (https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks))](https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks))

In [1]:

```
#Importing important modules
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
#data import
df = pd.read_csv('../DATA/sonar.all-data.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	Freq_1	Freq_2	Freq_3	Freq_4	Freq_5	Freq_6	Freq_7	Freq_8	Freq_9	Freq_10	...	Fre
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.(
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.(
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.(
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.(
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.(

5 rows × 61 columns

In [4]:

```
df.shape
```

Out[4]:

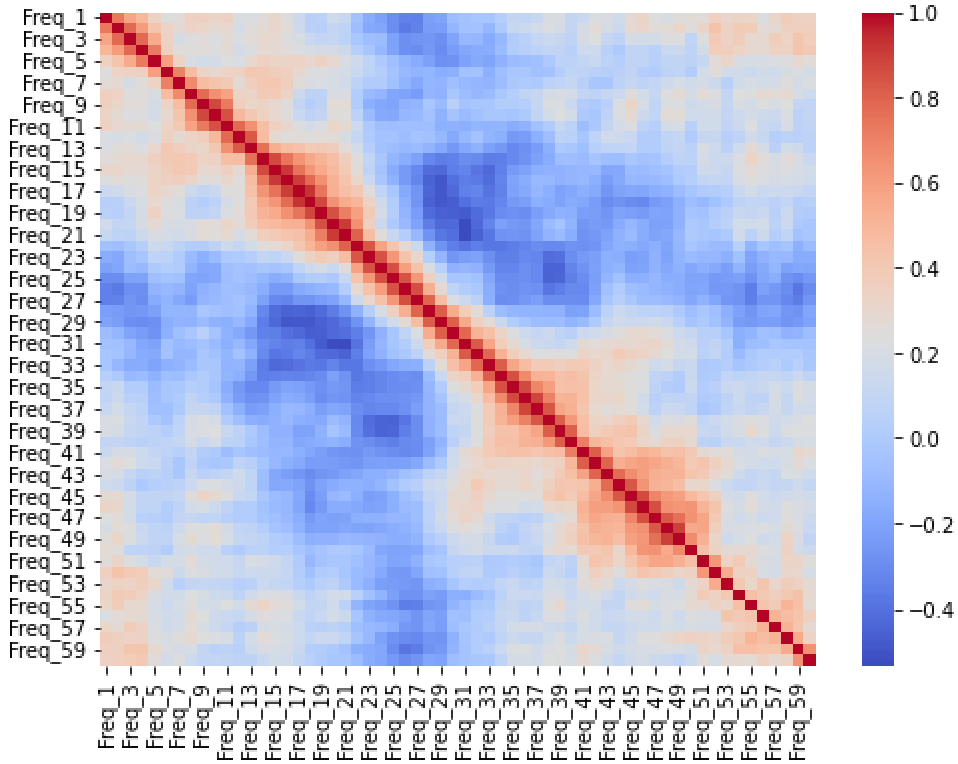
```
(208, 61)
```

In [5]:

```
#Data observation.Observing correlation between various features
dff = df.corr()
plt.figure(figsize=(8,6))
sns.heatmap(dff,cmap='coolwarm')
```

Out[5]:

```
<AxesSubplot:>
```



In [6]:

```
#Changing string data to numeric for evaluation
df['Target'] = df['Label'].map({'R':0,'M':1})
```

In [7]:

```
df
```

Out[7]:

	Freq_1	Freq_2	Freq_3	Freq_4	Freq_5	Freq_6	Freq_7	Freq_8	Freq_9	Freq_10	...	F
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	
...	...	...	...	...	...	...	...	...	...	...	...	
203	0.0187	0.0346	0.0168	0.0177	0.0393	0.1630	0.2028	0.1694	0.2328	0.2684	...	
204	0.0323	0.0101	0.0298	0.0564	0.0760	0.0958	0.0990	0.1018	0.1030	0.2154	...	
205	0.0522	0.0437	0.0180	0.0292	0.0351	0.1171	0.1257	0.1178	0.1258	0.2529	...	
206	0.0303	0.0353	0.0490	0.0608	0.0167	0.1354	0.1465	0.1123	0.1945	0.2354	...	
207	0.0260	0.0363	0.0136	0.0272	0.0214	0.0338	0.0655	0.1400	0.1843	0.2354	...	

208 rows × 62 columns

In [8]:

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

Out[8]:

```
Freq_45    0.339406
Freq_10    0.341142
Freq_49    0.351312
Freq_12    0.392245
Freq_11    0.432855
Target     1.000000
Name: Target, dtype: float64
```

In [9]:

```
#Importing SKLearn for data splitting
from sklearn.model_selection import train_test_split
```

In [10]:

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

In [11]:

```
y = df['Label']
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=
```

In [13]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [14]:

```
model = KNeighborsClassifier()
```

In [15]:

```
from sklearn.preprocessing import StandardScaler
```

In [16]:

```
scaler = StandardScaler()
```

In [17]:

```
operations = [('scaler',scaler),('model',model)]
```

In [18]:

```
#Creating a pipeline to determine best values which give good result with minimal error
from sklearn.pipeline import Pipeline
```

In [19]:

```
pipe = Pipeline(operations)
```

In [20]:

```
from sklearn.model_selection import GridSearchCV
```

In [21]:

```
k_values = list(range(1,30))
```

In [22]:

```
param_grid = {'model__n_neighbors':k_values}
```

In [23]:

```
classi = GridSearchCV(pipe,param_grid,cv=5,scoring='accuracy')
```

In [24]:

```
#Obtaining the best parameters from various given parameter choices
classi.fit(X_train,y_train)
```

Out[24]:

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                       ('model', KNeighborsClassifier())]),
             param_grid={'model__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 1
0,
                                                 11, 12, 13, 14, 15, 16, 17,
18,
                                                 19, 20, 21, 22, 23, 24, 25,
26,
                                                 27, 28, 29]},
             scoring='accuracy')
```

In [25]:

```
#All the best parameter value among the provided ones
classi.best_estimator_.get_params()
```

Out[25]:

```
{'memory': None,
 'steps': [('scaler', StandardScaler()),
  ('model', KNeighborsClassifier(n_neighbors=1))],
 'verbose': False,
 'scaler': StandardScaler(),
 'model': KNeighborsClassifier(n_neighbors=1),
 'scaler__copy': True,
 'scaler__with_mean': True,
 'scaler__with_std': True,
 'model__algorithm': 'auto',
 'model__leaf_size': 30,
 'model__metric': 'minkowski',
 'model__metric_params': None,
 'model__n_jobs': None,
 'model__n_neighbors': 1,
 'model__p': 2,
 'model__weights': 'uniform'}
```

In [26]:

```
classi.cv_results_['mean_test_score']
```

Out[26]:

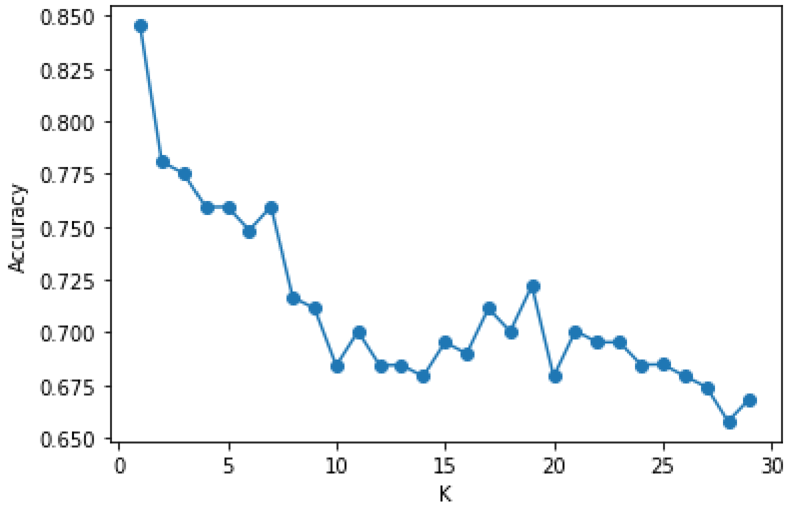
```
array([0.84537696, 0.78065434, 0.77524893, 0.75917496, 0.75931721,
       0.74822191, 0.75945946, 0.71664296, 0.7113798 , 0.68421053,
       0.70042674, 0.68435277, 0.68449502, 0.67908962, 0.69530583,
       0.68990043, 0.7113798 , 0.70042674, 0.72204836, 0.67908962,
       0.70071124, 0.69530583, 0.69530583, 0.68463727, 0.68477952,
       0.67923186, 0.67411095, 0.65775249, 0.6685633 ])
```

In [27]:

```
#Determining which K value to choose with minimal processing and maximum output
scores = classi.cv_results_['mean_test_score']
plt.plot(k_values,scores,'o-')
plt.xlabel("K")
plt.ylabel("Accuracy")
```

Out[27]:

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



In [28]:

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

In [29]:

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

In [30]:

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

Out[30]:

```
array([[12,  1],
       [ 1,  7]], dtype=int64)
```

In [31]:

```
#Recall and accuracy quite good
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           M       0.92      0.92      0.92        13
           R       0.88      0.88      0.88         8

    accuracy                           0.90        21
   macro avg       0.90      0.90      0.90        21
weighted avg       0.90      0.90      0.90        21
```