In [ ]:

```
# Imports

# Data tools and plotting
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
from sklearn.model_selection import train_test_split
import plotly.express as px

# Metrics
from sklearn.calibration import calibration_curve
from sklearn.metrics import accuracy_score, plot_confusion_matrix, brier_score_l

# Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

!pip install shap

import shap
```

```
Collecting shap
  Downloading https://files.pythonhosted.org/packages/85/a3/c0eab9dd6a894165e2cb
87504ff5b2710ac5ede3447d9138620b7341b6a2/shap-0.37.0.tar.gz (326kB)
     |████████████████████████████████| 327kB 5.3MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from shap) (1.18.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages
(from shap) (1.4.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-pac
kages (from shap) (0.22.2.post1)
Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages
(from shap) (1.1.4)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.6/dist-pack
ages (from shap) (4.41.1)
Collecting slicer==0.0.3
  Downloading https://files.pythonhosted.org/packages/02/a6/c708c5a0f338e99cfbcb
6288b88794525548e4fc1b8457feec2c552a81a4/slicer-0.0.3-py3-none-any.whl
Requirement already satisfied: numba in /usr/local/lib/python3.6/dist-packages
(from shap) (0.48.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-pac
kages (from scikit-learn->shap) (0.17.0)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-pac
kages (from pandas->shap) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.
6/dist-packages (from pandas->shap) (2.8.1)
Requirement already satisfied: llvmlite<0.32.0,>=0.31.0dev0 in /usr/local/lib/py
thon3.6/dist-packages (from numba->shap) (0.31.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packa
ges (from numba->shap) (50.3.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-package
s (from python-dateutil>=2.7.3->pandas->shap) (1.15.0)
Building wheels for collected packages: shap
  Building wheel for shap (setup.py) ... done
  Created wheel for shap: filename=shap-0.37.0-cp36-cp36m-linux_x86_64.whl size=
463913 sha256=ff55e6a51f543242a95f8c7ef8c104db5d1a43b134ebe15a10150ad14b4ea14c
  Stored in directory: /root/.cache/pip/wheels/df/ad/b0/aa7815ec68850d66551ef618
095eccb962c8f6022f1d3dd989
Successfully built shap
```

```
Installing collected packages: slicer, shap
Successfully installed shap-0.37.0 slicer-0.0.3
```

Problem, Hypothesis, Data, and Approach

Given sonar data of metal cylinders and of rocks, we want to classify each chirp as either coming from a rock or coming from a cylinder.

We hypothesise that there will be a couple of "key" frequencies that resonate differently with the cylinders than with the rocks, and will be instrumental in differentiating them from rocks.

The data was collected from UC Irvine (DGorman, R. P., and Sejnowski, T. J. (1988). "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets" in Neural Networks, Vol. 1, pp. 75-89), and includes recordings from a representative range of angles (90 degrees for cylinders, which are cylindrically symmetrical, and 180 degrees for rocks). The features have already been extracted and cleaned.

For model selection, we considered the fact that this is a classification task, and therefore decided to compare various classification models: Random Forest Classifier, K-Nearest-Neighbors Classifier, and Support Vector Classifier (SVC). We compare the performance of these models, measuring their classification accuracy, and visualizing their confusion matrices and calibration curves to understand their suseptibility to misclassifications.

Finally, we analyse the models with SHAP to gain some takeaways, and explore a potential reason as to why SHAP found particular frequencies to be useful.

Read in the data, split to train and test sets

In [ ]:

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/undo

# column 60 is the label
X_train,X_test,y_train,y_test = train_test_split(df.iloc[:,:60],df[60])#(df.drop
```

Initial Visualization

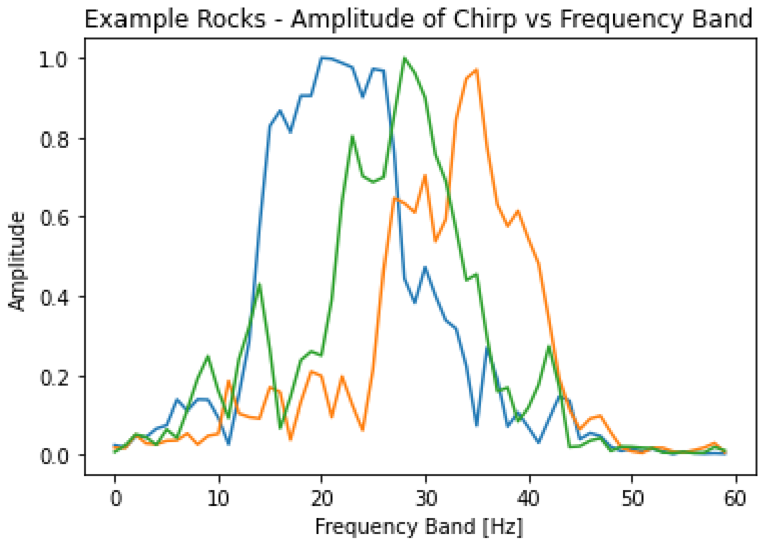
Example Plots of individual samples

The results are not very helpful but we can already see general differences between rocks and cylinders around frequency bands 10 and 45

Rocks

In [ ]:

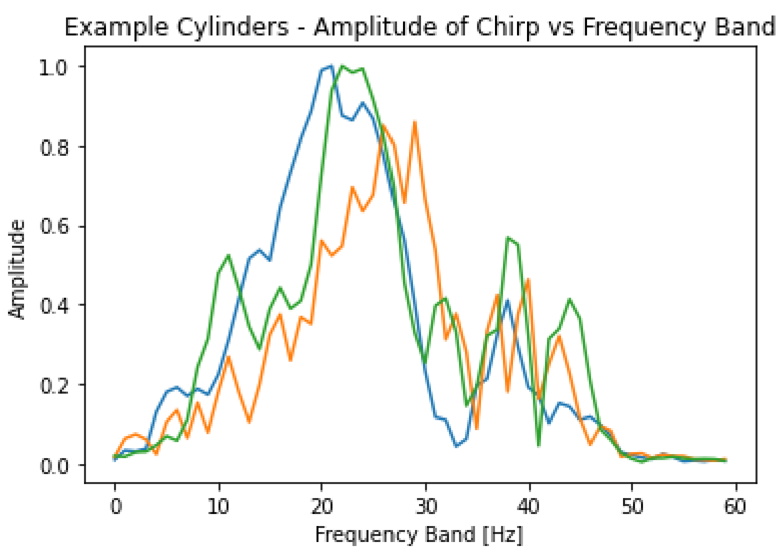
```
# shuffle the Rock samples and plot the first 3
for i, row in df[df[60]=="R"].drop(columns=[60]).sample(frac=1.).head(3).iterrow
  row.plot(title="Example Rocks - Amplitude of Chirp vs Frequency Band",xlabel="
```



Cylinders

In [ ]:

```
for i, row in df[df[60]=="M"].drop(columns=[60]).sample(frac=1.).head(3).iterrow
  row.plot(title="Example Cylinders - Amplitude of Chirp vs Frequency Band",xlab
```



Additional Visualization using t-SNE, a dimensionality reduction technique

There are not perfectly clear clusters, but the distribution is not totally random. Rocks tend to be closer to rocks and cylinders to cylinders in frequency-band space. That's simply what this visualization shows.

In [ ]:

```
# quick visualization using TSNE
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
vis = pd.DataFrame(tsne.fit_transform(df.drop(columns=[60])));vis["label"] = df[
px.scatter(vis,x=0,y=1,color="label",hover_data=["label"])
```

Modeling the Data, Evaluating the Models

Useful Functions

In [ ]:

```
def train_model(model, name):
  # given a newly initialized model, fit it to the data,
  # make a confusion matrix and measure its accuracy

  # fit the model to the data, and have it predict on the test set data
  model.fit(X_train,y_train)
  model_pred = model.predict(X_test)

  #accuracy and precision metrics
  acc = accuracy_score(y_test, model_pred); print("Accuracy Achieved: ", acc)
  print("Precision of "+name+" for Class R :", precision_score(y_test,model_pred
  print("Precision of "+name+" for Class M :", precision_score(y_test,model_pred

  # Generate confusion matrix plot
  plt.figure()
  plt.title(name+" Confusion Matrix")
  model_ax = plt.axes()
  plot_confusion_matrix(model,X_test,y_test, ax=model_ax)

def generate_calibration_curve(model,X,y, n_bins=8):
```

```
    probabilities = model.predict_proba(X)[:,1] # probability of positive
    # generate the calibration curve
    fraction_of_positives, mean_predicted_value = calibration_curve(y, probabiliti
    # and plot it
    plt.figure()
    plt.title("Cablibration Curve - Predicting that the Object is a Rock")
    plt.ylabel("Fraction of positives")
    plt.xlabel("Predicted Probability of being a Rock")
    plt.plot(mean_predicted_value, fraction_of_positives,"s-")
    plt.plot([0,1], [0,1], "--")

    #Measure the Brier score, a measure of calibration
    print("Brier score loss: ", brier_score_loss(y, probabilities))

def shap_it(model, X_train, n=None):
    shap.initjs()
    if n:
      # choose n random indices if specified, otherwise all
      ix = np.random.choice(X_train.index.tolist(),n)
      X_shap = X_train.loc[ix,:]
    else:
      X_shap = X_train
    # pass in the predict_proba method
    explainer = shap.KernelExplainer(model.predict_proba, X_shap)
    shap_values = explainer.shap_values(X_shap)
    #visualize the shap summary
    summary = shap.summary_plot(shap_values, X_shap)
    # and then the "line plot"
    line = shap.force_plot(explainer.expected_value[0], shap_values[0][0], X_shap.
    return summary, line
```

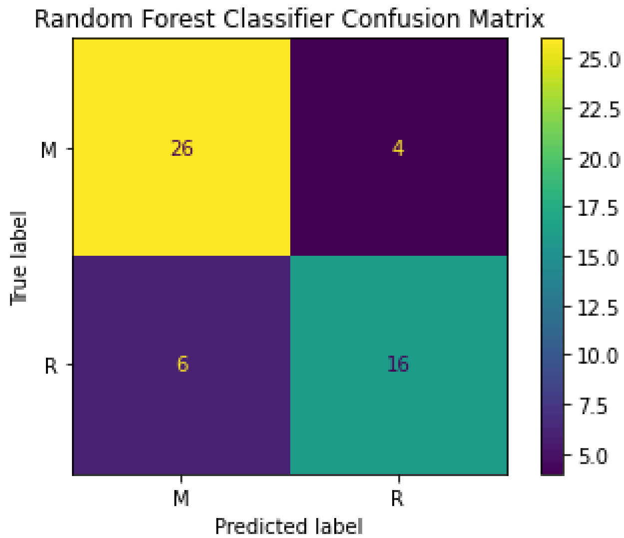
Random Forest Classifier

In [ ]:

```
rf = RandomForestClassifier()
train_model(rf, "Random Forest Classifier")
```

```
Accuracy Achieved:  0.8076923076923077
Precision of Random Forest Classifier for Class R : 0.8
Precision of Random Forest Classifier for Class M : 0.8125
```

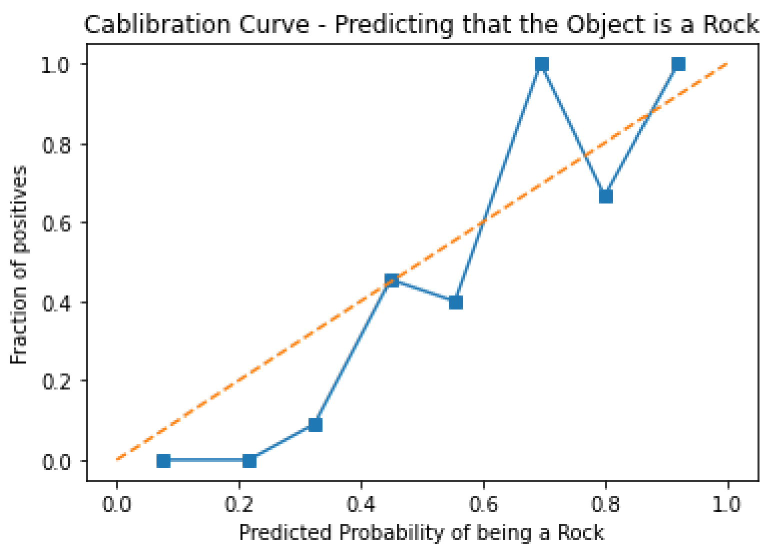
```
Adding an axes using the same arguments as a previous axes currently reuses the
earlier instance.  In a future version, a new instance will always be created an
d returned.  Meanwhile, this warning can be suppressed, and the future behavior
ensured, by passing a unique label to each axes instance.
```



In [ ]:

```
generate_calibration_curve(rf, X_test, y_test)
```

```
Brier score loss:  0.13330576923076926
```



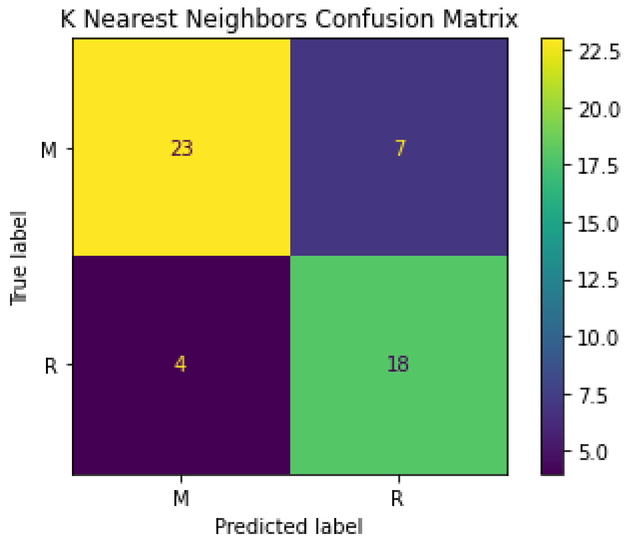
K Nearest Neighbors Classifier

In [ ]:

```
kn = KNeighborsClassifier() # default is 5 neighbors
train_model(kn, "K Nearest Neighbors")
```

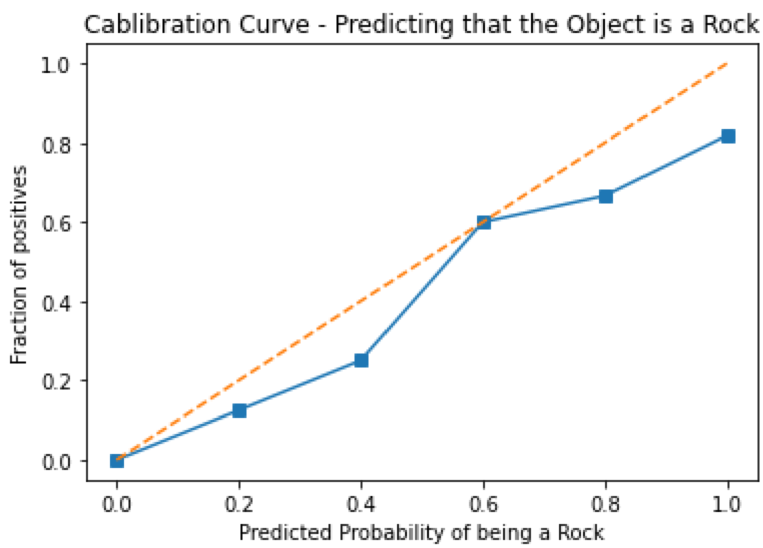
```
Accuracy Achieved:  0.7884615384615384
Precision of K Nearest Neighbors for Class R : 0.72
Precision of K Nearest Neighbors for Class M : 0.8518518518518519
```

```
Adding an axes using the same arguments as a previous axes currently reuses the
earlier instance.  In a future version, a new instance will always be created an
d returned.  Meanwhile, this warning can be suppressed, and the future behavior
ensured, by passing a unique label to each axes instance.
```



In [ ]:
```
generate_calibration_curve(kn, X_test, y_test, n_bins=8)
```

```
Brier score loss:  0.1692307692307692
```

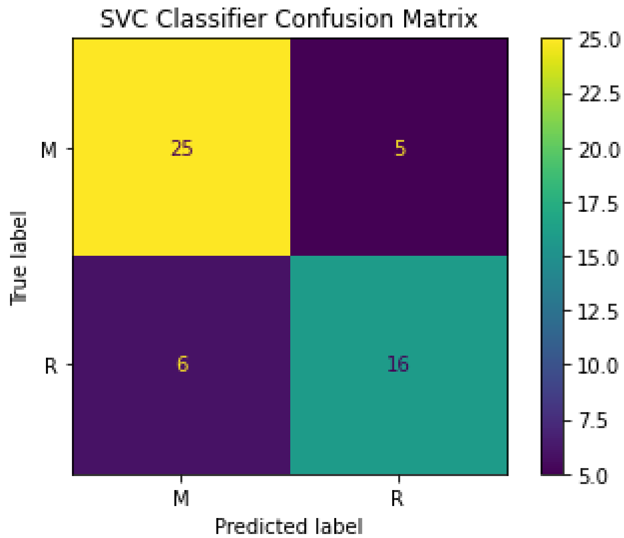


Support Vector Classifier

In [ ]:
```
# Train the SVC model, enabling probability so that we can make a calibration cu
svc = SVC(probability=True)
train_model(svc,"SVC Classifier")
```

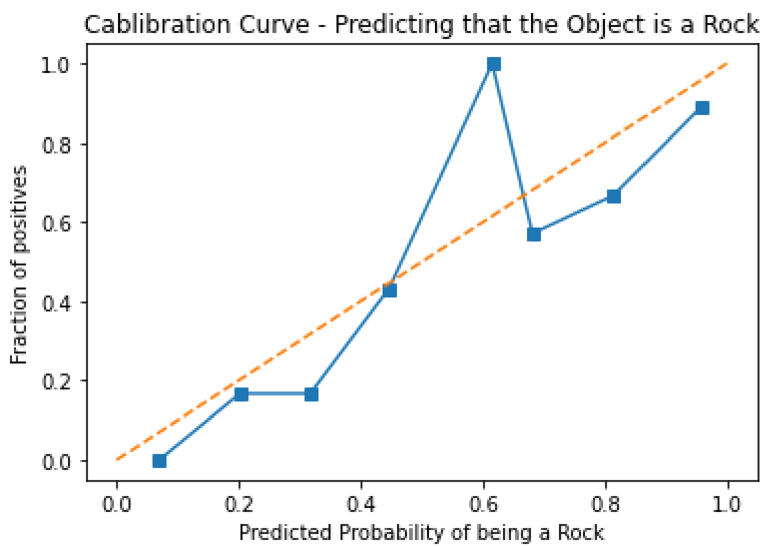
```
Accuracy Achieved:  0.7884615384615384
Precision of SVC Classifier for Class R : 0.7619047619047619
Precision of SVC Classifier for Class M : 0.8064516129032258
```

```
Adding an axes using the same arguments as a previous axes currently reuses the
earlier instance.  In a future version, a new instance will always be created an
d returned.  Meanwhile, this warning can be suppressed, and the future behavior
ensured, by passing a unique label to each axes instance.
```



In [ ]:
```
generate_calibration_curve(svc, X_test, y_test, n_bins=8)
```

```
Brier score loss:  0.15672580028154084
```

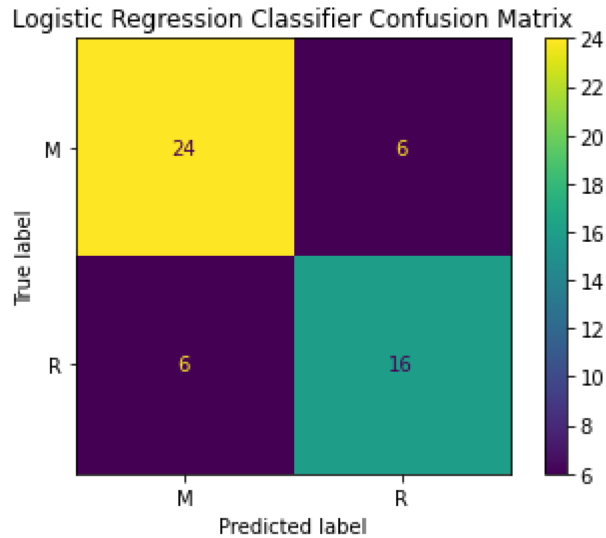


Logistic Regression Classifier

In [ ]:
```
# The results are bad
lr = LogisticRegression()
train_model(lr,"Logistic Regression Classifier")
```

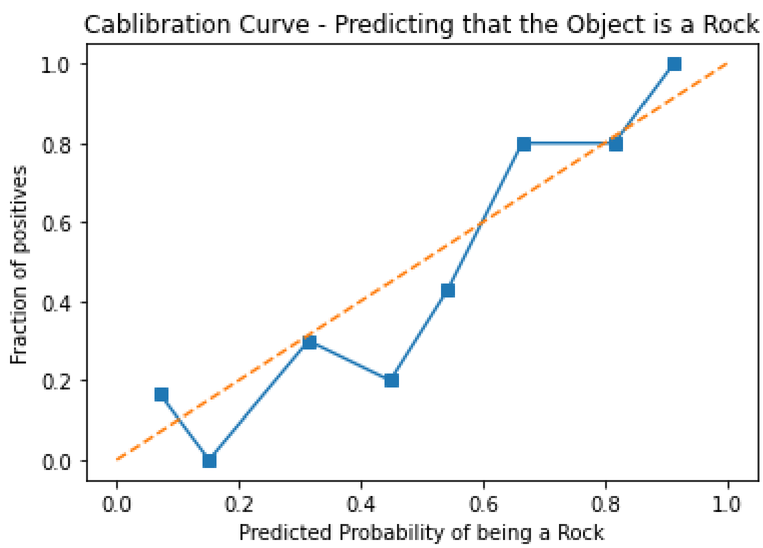
```
Accuracy Achieved:  0.7692307692307693
Precision of Logistic Regression Classifier for Class R : 0.7272727272727273
Precision of Logistic Regression Classifier for Class M : 0.8
```

```
Adding an axes using the same arguments as a previous axes currently reuses the
earlier instance.  In a future version, a new instance will always be created an
d returned.  Meanwhile, this warning can be suppressed, and the future behavior
ensured, by passing a unique label to each axes instance.
```



In [ ]:
```
generate_calibration_curve(lr, X_test, y_test, n_bins=8)
```

```
Brier score loss:  0.16885857656734973
```



Analysis, Interpretation, and Knowledge Extraction

Shap Analysis

Which frequencies are most useful for discerning rock from cylinder?

Random Forest is our selected model due to its high performance and high interpretability , but we will also perform Shapley Analysis on the other models for supporting evidence and to see if they depend on similar frequencies

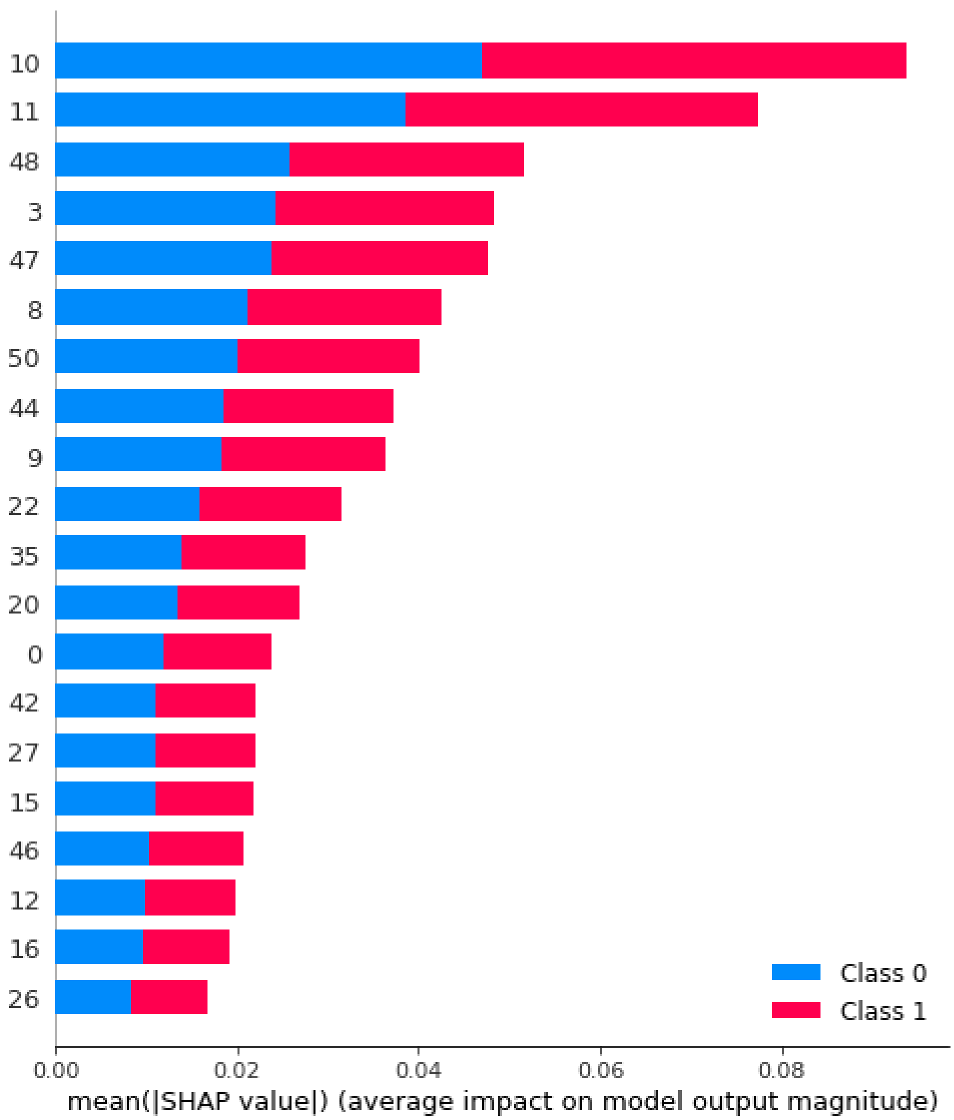
Note that each model tends to identify many frequency bands in the following ranges as important:

- 9-12
- 18-22
- 34-36
- 45-50

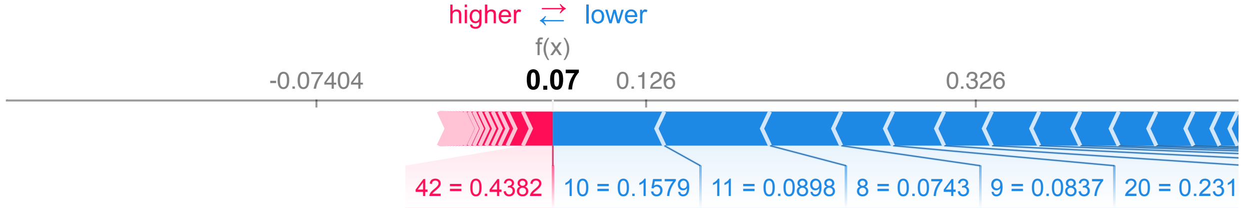
Random Forest explained by SHAP:

In [ ]:

```
# We can use the TreeExplainer on the Random Forest model, which is our selected
shap.initjs()
# Explain the model's predictions using SHAP for the first 100 data points
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_train)
# Visualize the SHAP summary for the first 100 data points
shap.summary_plot(shap_values, X_train)
#Look at the first data point using a force plot
shap.force_plot(explainer.expected_value[0], shap_values[0][:1], X_train[:1])
```



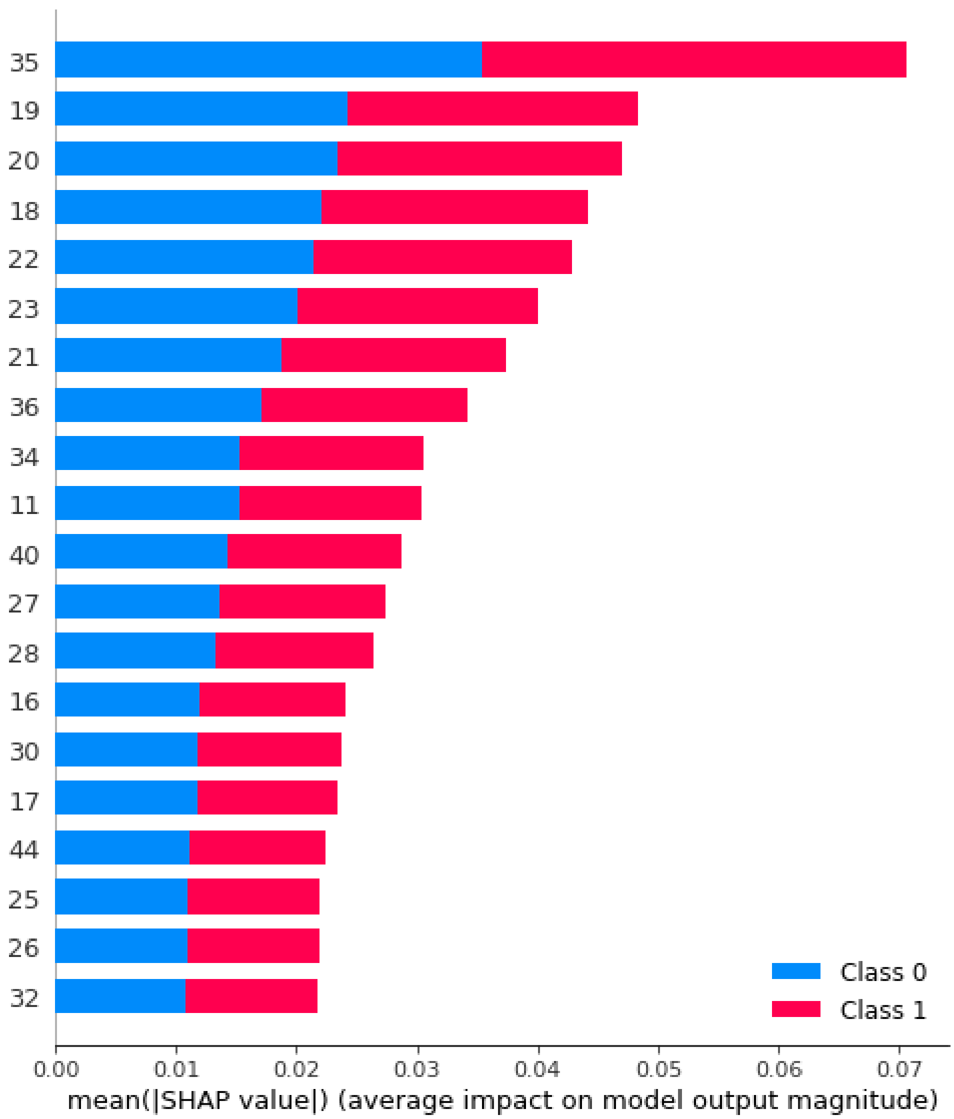
Out[ ]:



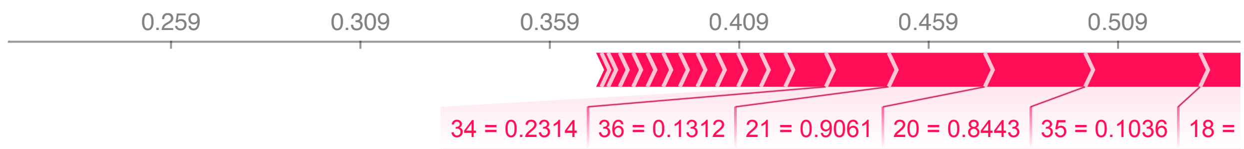
In [ ]:

```
# Use SHAP on KNN classifier
summary, line = shap_it(kn,X_train)
summary
line
```

```
Using 156 background data samples could cause slower run times. Consider using s
hap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K sam
ples.
```



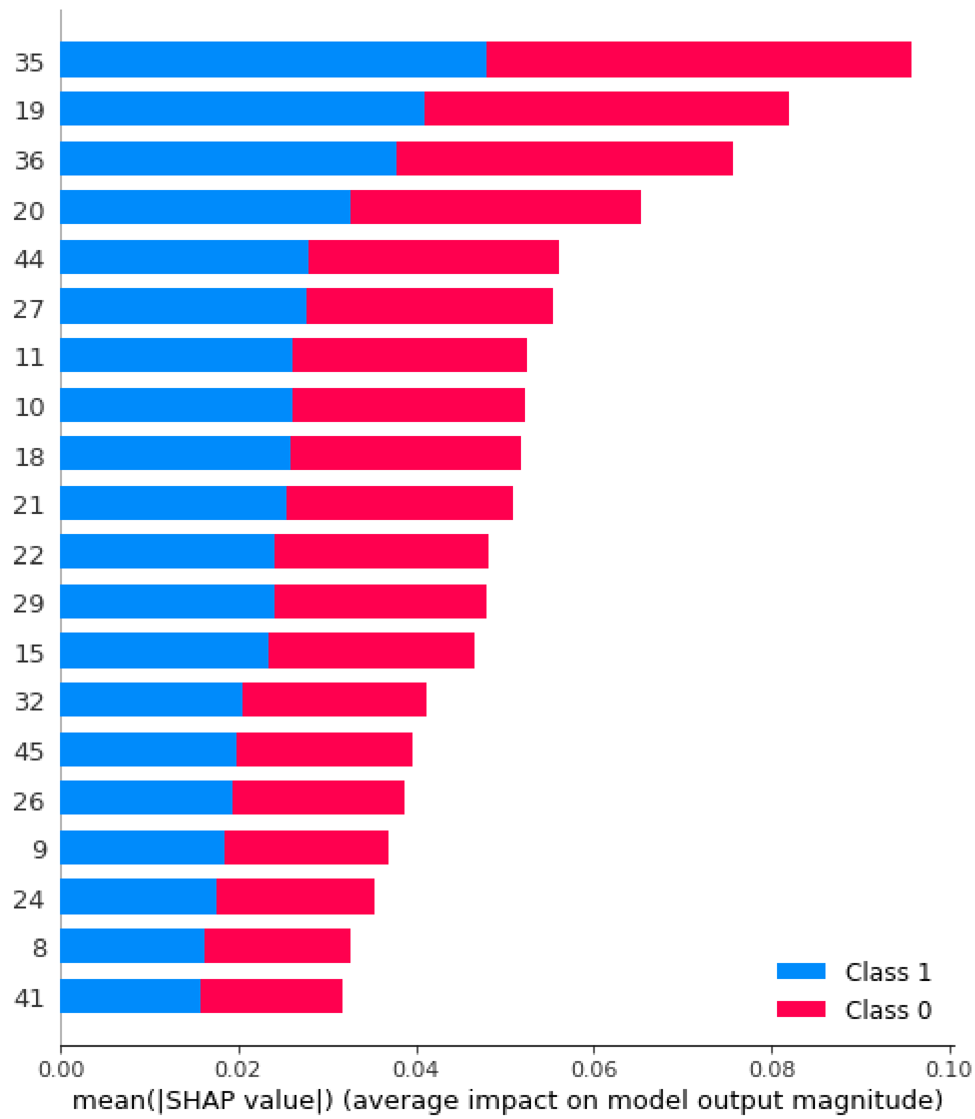
Out[ ]:



In [ ]:

```
# Use SHAP on SVC
summary, line = shap_it(svc,X_train)
summary
line
```

```
Using 156 background data samples could cause slower run times. Consider using s
hap.sample(data, K) or shap.kmeans(data, K) to summarize the background as K sam
ples.
```



Out[ ]:

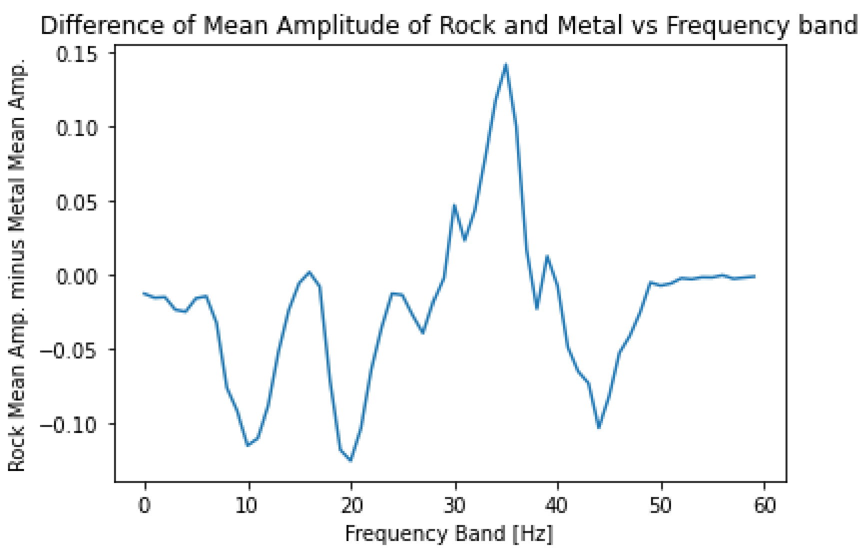
Differences in Mean Amplitudes per Frequency Band between Rocks and Cylinders

The peaks and valleys of this chart, which indicate the frequency bands in which the difference between the means of the Rocks' amplitudes and the Cylinders' amplitudes is the greatest, line up very closely with the frequency band ranges identified as important through shapley analysis

In [ ]:

```
# Take all Rock samples, take the mean over the samples, do the same for the cyl
(df[df[60]=="R"].mean(axis=0) - df[df[60]=="M"].mean(axis=0)).plot(title = "Diff
                                                                    xlabel="Frequ
```

Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fbb842dc2e8>



Histograms of Amplitudes of Specific Key Frequencies

In the plot above, observe that there are large mean differences around frequency band **10, 20, 33-35, 44-48** . These frequencies are also marked as important by the SHAP analysis. Lets plot histograms of these frequencies for rocks and for cylinders, and see if there is a discernable difference.

Note that the peaks often are in different places, or that ther is at least a significant amount of non-overlap between the histograms.

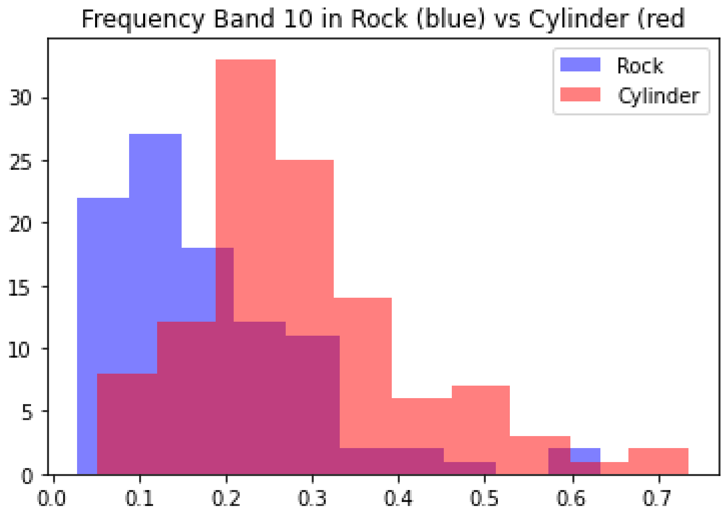
In [ ]:

```
def compare_freq(band):
  # extract the column we're interested in, separately for samples of each class
```

```
    r = df[df[60]=="R"][band]
    m = df[df[60]=="M"][band]
    # make overlapping histograms
    plt.hist(r,alpha=0.5,color="blue")
    plt.hist(m,alpha=0.5, color = "red")
    # add a legend
    plt.legend(["Rock", "Cylinder"])
    plt.title("Frequency Band "+str(band)+" in Rock (blue) vs Cylinder (red")
    plt.show()
```

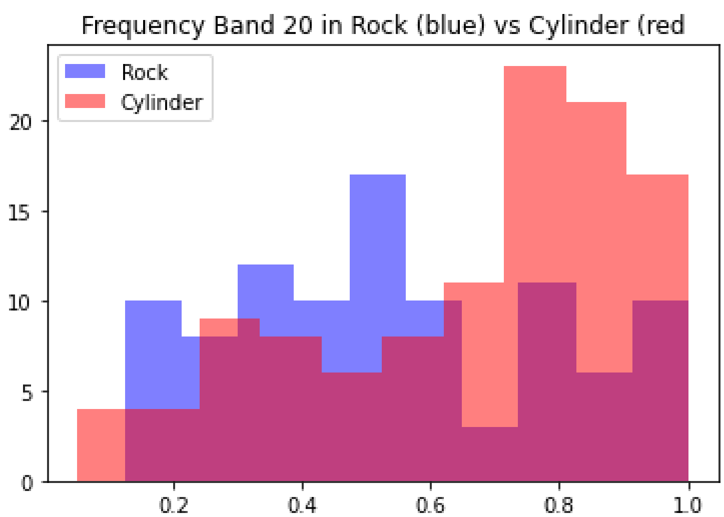
In [ ]:

```
compare_freq(10)
```



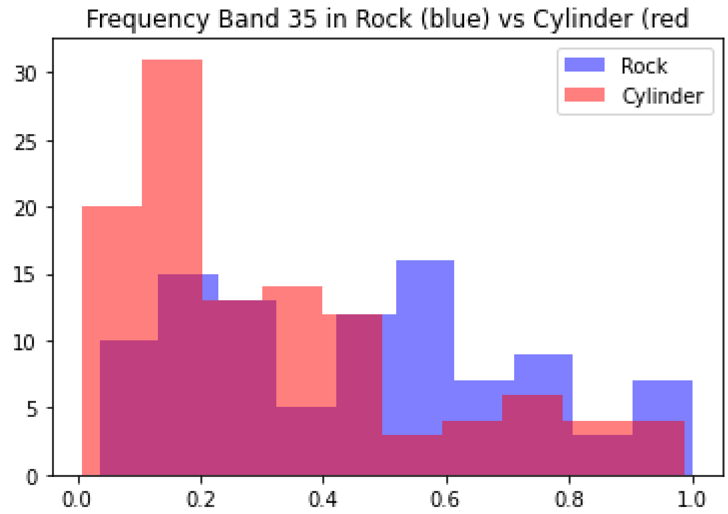
In [ ]:

```
compare_freq(20)
```

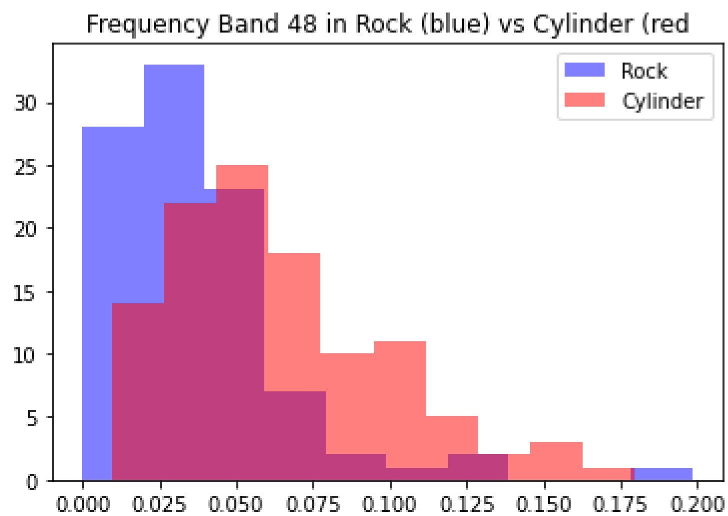


In [ ]:

```
compare_freq(35)
```



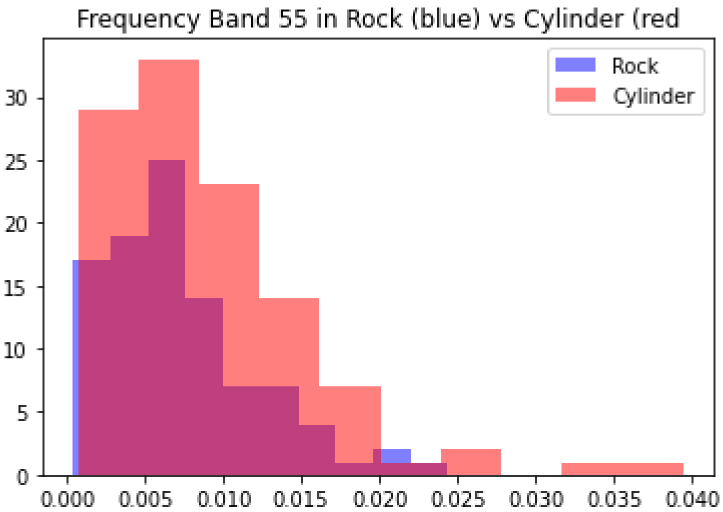
In [ ]:
```
compare_freq(48)
```



Compare to useless frequency bands

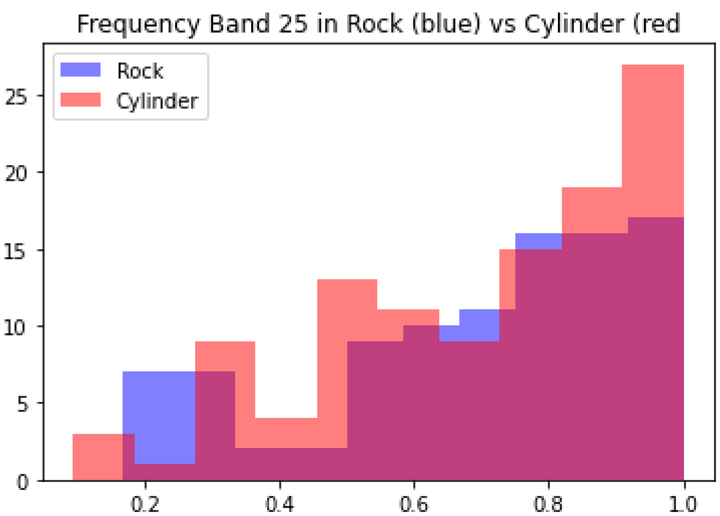
The peaks occur in the same place! This is probably why these frequencies don't supply very much useful information to the models, and are not identified as important by SHAP.

In [ ]:
```
compare_freq(55)
```



In [ ]:

```
compare_freq(25)
```



Conclusion

The data support our hypothesis that certain frequencies (for example, bands 10, 20, 35, 48) are key distinguishers between chirps coming from rocks and chirps coming from cylinders. This is evidenced by 1. SHAP analysis of well-performing machine learning models, and 2. By the different means and distributions of the amplitudes in these frequency bands in rocks vs. cylinders.

In [ ]: