```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA


# Load the datase
df = pd.read_csv('/content/sonar data.csv')



df.head()
```

	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.
0	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.
1	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.
2	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.
3	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.
4	0.0286	0.0453	0.0277	0.0174	0.0384	0.0990	0.1201	0.1833	0.2105	0.3039	...	0.0045	0.0014	0.0038	0.0013	0.0089	0.0057	0.

5 rows × 61 columns

```
print("Shape of the dataset:", data.shape)
```

```
Shape of the dataset: (208, 61)
```

```
print(data.describe())
```

```
                0           1           2           3           4           5   \
count  208.000000  208.000000  208.000000  208.000000  208.000000  208.000000
mean     0.029164    0.038437    0.043832    0.053892    0.075202    0.104570
std      0.022991    0.032960    0.038428    0.046528    0.055552    0.059105
min      0.001500    0.000600    0.001500    0.005800    0.006700    0.010200
25%      0.013350    0.016450    0.018950    0.024375    0.038050    0.067025
50%      0.022800    0.030800    0.034300    0.044050    0.062500    0.092150
75%      0.035550    0.047950    0.057950    0.064500    0.100275    0.134125
max      0.137100    0.233900    0.305900    0.426400    0.401000    0.382300

                6           7           8           9   ...          50  \
count  208.000000  208.000000  208.000000  208.000000  ...  208.000000
mean     0.121747    0.134799    0.178003    0.208259  ...    0.016069
std      0.061788    0.085152    0.118387    0.134416  ...    0.012008
min      0.003300    0.005500    0.007500    0.011300  ...    0.000000
25%      0.080900    0.080425    0.097025    0.111275  ...    0.008425
50%      0.106950    0.112100    0.152250    0.182400  ...    0.013900
75%      0.154000    0.169600    0.233425    0.268700  ...    0.020825
max      0.372900    0.459000    0.682800    0.710600  ...    0.100400

               51          52          53          54          55          56  \
count  208.000000  208.000000  208.000000  208.000000  208.000000  208.000000
mean     0.013420    0.010709    0.010941    0.009290    0.008222    0.007820
std      0.009634    0.007060    0.007301    0.007088    0.005736    0.005785
min      0.000800    0.000500    0.001000    0.000600    0.000400    0.000300
25%      0.007275    0.005075    0.005375    0.004150    0.004400    0.003700
50%      0.011400    0.009550    0.009300    0.007500    0.006850    0.005950
75%      0.016725    0.014900    0.014500    0.012100    0.010575    0.010425
max      0.070900    0.039000    0.035200    0.044700    0.039400    0.035500

               57          58          59
count  208.000000  208.000000  208.000000
mean     0.007949    0.007941    0.006507
std      0.006470    0.006181    0.005031
min      0.000300    0.000100    0.000600
25%      0.003600    0.003675    0.003100
50%      0.005800    0.006400    0.005300
75%      0.010350    0.010325    0.008525
max      0.044000    0.036400    0.043900

[8 rows x 60 columns]
```

2 **Data** Preparation

```
X = df.iloc[:, :-1]  # Features
y = df.iloc[:, -1]   # Target
```

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

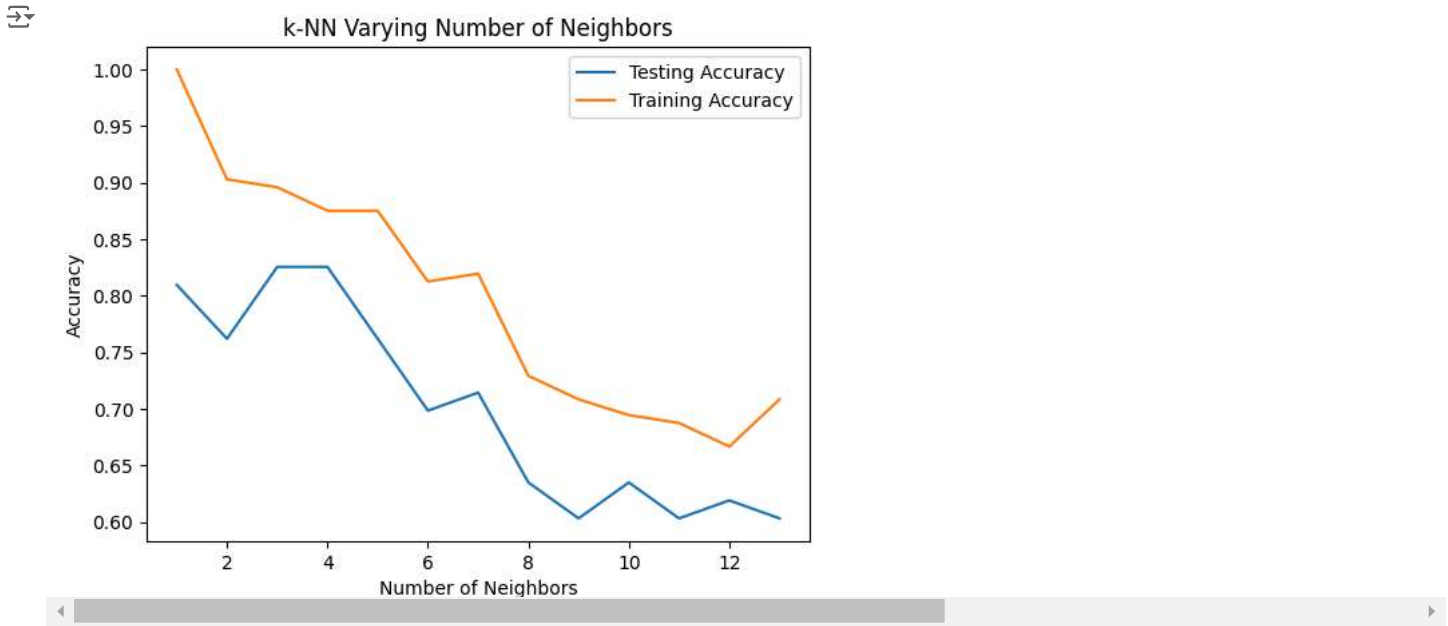
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3 Model Development

```
from sklearn.neighbors import KNeighborsClassifier
neighbors = np.arange(1, 14)
train_accuracy = np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))
```

```
for i, k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)
```

```
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training Accuracy')
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.title('k-NN Varying Number of Neighbors')
plt.legend()
plt.show()
```




```
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred_logistic = model.predict(X_test)
```

```
# Initialize PCA and reduce the number of components
pca = PCA(n_components=10)  # Adjust the number of components as needed
```

```
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train Logistic Regression model on PCA-transformed data
model_pca = LogisticRegression()
model_pca.fit(X_train_pca, y_train)

# Make predictions
y_pred_pca = model_pca.predict(X_test_pca)


# Train SVM model on original features
svm = SVC(kernel='linear')  # You can use different kernels like 'rbf' as well
svm.fit(X_train, y_train)

# Make predictions
y_pred_svm = svm.predict(X_test)
```

4 Model Evaluation

```
from sklearn.metrics import accuracy_score, confusion_matrix
print("kNN Accuracy:", accuracy_score(y_test, y_pred_knn))
```

```
kNN Accuracy: 0.7619047619047619
```

```
print("kNN Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_knn))
```

```
kNN Confusion Matrix:
[[31  5]
 [10 17]]
```

```
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logistic))
```

```
Logistic Regression Accuracy: 0.746031746031746
```

```
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_logistic))
```

```
Logistic Regression Confusion Matrix:
[[24 12]
 [ 4 23]]
```

```
# Evaluate the PCA-based model
print("PCA + Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_pca))
print("PCA + Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_pca))
```

```
PCA + Logistic Regression Accuracy: 0.7619047619047619
PCA + Logistic Regression Confusion Matrix:
[[24 12]
 [ 3 24]]
```

```
# Evaluate the SVM model
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print("SVM Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.7619047619047619
SVM Confusion Matrix:
[[26 10]
 [ 5 22]]
```



```
input_data = (0.0100,0.0171,0.0623,0.0205,0.0205,0.0368,0.1098,0.1276,0.0598,0.1264,0.0881,0.1992,0.0184,0.2261,0.1729,0.2131,0.0693,0.2281,0

#changing the input_data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

#reshape the np array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped)
print(prediction)
```

```
if (prediction[0]=='R'):
  print('The object is a Rock')
else:
  print('The object is a mine')
```

```
[1]
The object is a mine
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Logistic
  warnings.warn(
```