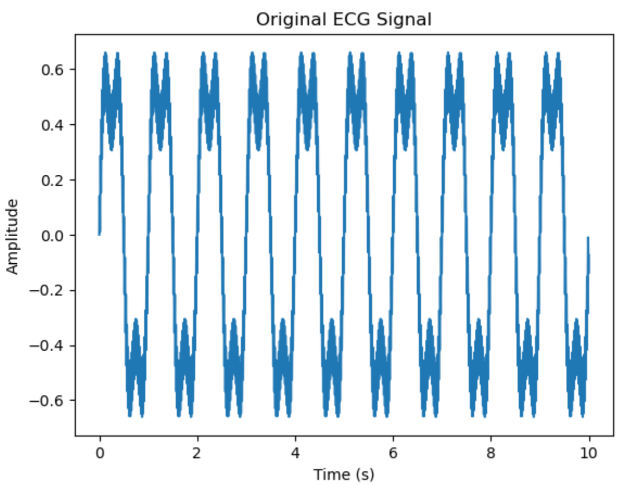
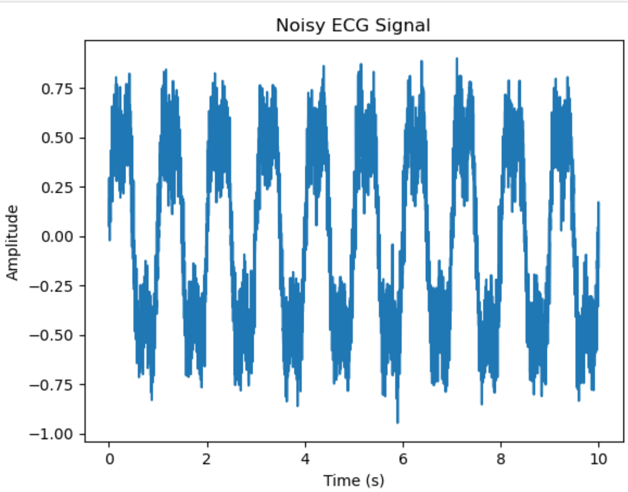
In [40]:
```
import numpy as np
import matplotlib.pyplot as plt


import pandas as pd
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, mean_squared_error
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, mean_squared_error, precision_score, recall_score, f1_score, classification_report
from sklearn.model_selection import GridSearchCV
from scipy.signal import butter, lfilter, stft
```

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(42)


fs = 500  # Sampling frequency (Hz)
t = np.arange(0, 10, 1/fs)  # Time vector for 10 seconds


ecg_signal = 0.6 * np.sin(2 * np.pi * 1 * t) + 0.2 * np.sin(2 * np.pi * 3 * t) + 0.1 * np.sin(2 * np.pi * 50 * t)

# Plot the synthetic ECG signal
plt.plot(t, ecg_signal)
plt.title('Original ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```

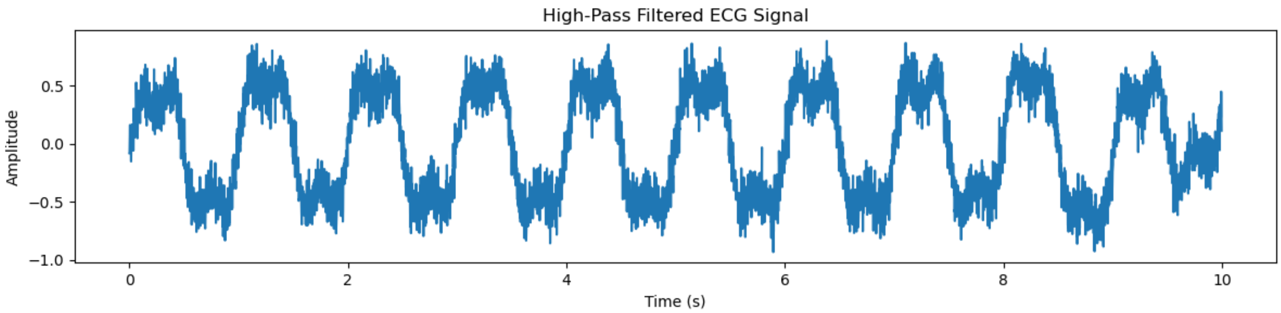


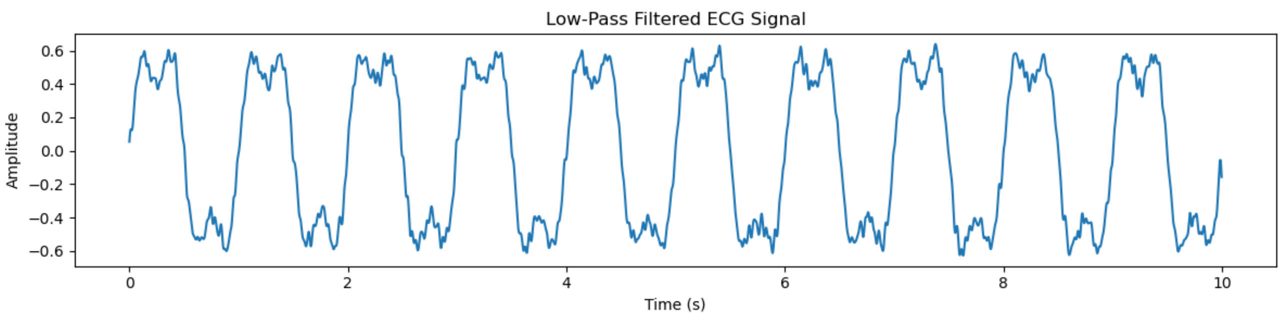
In [42]:
```
np.random.seed(42)
noise = np.random.normal(0, 0.1, ecg_signal.shape)
noisy_signal = ecg_signal + noise


plt.plot(t, noisy_signal)
plt.title('Noisy ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()
```



In [43]:
```
from scipy.signal import butter, filtfilt
np.random.seed(42)

filter(signal, cutoff=0.5, fs=500, order=5):
```

```
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_signal = filtfilt(b, a, signal)
    return filtered_signal


def lowpass_filter(signal, cutoff=30, fs=500, order=5):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_signal = filtfilt(b, a, signal)
    return filtered_signal


high_pass_signal = highpass_filter(noisy_signal)


low_pass_signal = lowpass_filter(noisy_signal)

# Plot the filtered signals
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
plt.plot(t, high_pass_signal)
plt.title('High-Pass Filtered ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 1, 2)
plt.plot(t, low_pass_signal)
plt.title('Low-Pass Filtered ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.tight_layout()
plt.show()
```



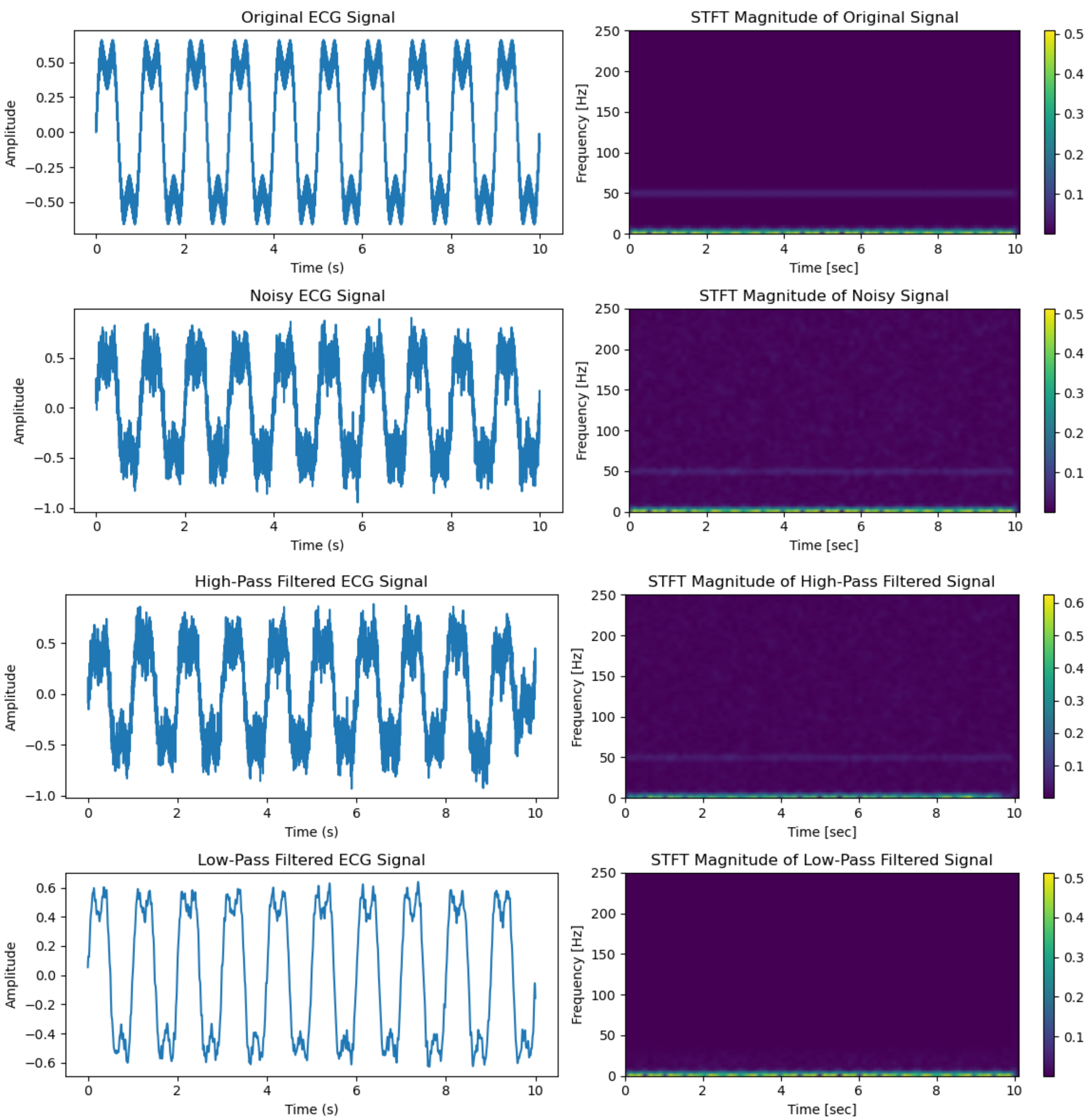


In [44]:

```
from scipy.signal import stft
np.random.seed(42)
# Apply Short-Time Fourier Transform (STFT)
f_original, t_original, Zxx_original = stft(ecg_signal, fs=fs, nperseg=128)
f_noisy, t_noisy, Zxx_noisy = stft(noisy_signal, fs=fs, nperseg=128)
f_high_pass, t_high_pass, Zxx_high_pass = stft(high_pass_signal, fs=fs, nperseg=128)
f_low_pass, t_low_pass, Zxx_low_pass = stft(low_pass_signal, fs=fs, nperseg=128)


plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.plot(t, ecg_signal)
plt.title('Original ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 2)
plt.pcolormesh(t_original, f_original, np.abs(Zxx_original), shading='gouraud')
plt.title('STFT Magnitude of Original Signal')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()


plt.subplot(2, 2, 3)
plt.plot(t, noisy_signal)
plt.title('Noisy ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 4)
plt.pcolormesh(t_noisy, f_noisy, np.abs(Zxx_noisy), shading='gouraud')
plt.title('STFT Magnitude of Noisy Signal')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
```

Loading [MathJax]/extensions/Safe.js

```python
plt.tight_layout()
plt.show()


plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
plt.plot(t, high_pass_signal)
plt.title('High-Pass Filtered ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 2)
plt.pcolormesh(t_high_pass, f_high_pass, np.abs(Zxx_high_pass), shading='gouraud')
plt.title('STFT Magnitude of High-Pass Filtered Signal')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()


plt.subplot(2, 2, 3)
plt.plot(t, low_pass_signal)
plt.title('Low-Pass Filtered ECG Signal')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')

plt.subplot(2, 2, 4)
plt.pcolormesh(t_low_pass, f_low_pass, np.abs(Zxx_low_pass), shading='gouraud')
plt.title('STFT Magnitude of Low-Pass Filtered Signal')
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.colorbar()

plt.tight_layout()
plt.show()
```



In [45]:
```
import pandas as pd
np.random.seed(42)


min_length = min(len(t), len(ecg_signal), len(noisy_signal), len(high_pass_signal), len(low_pass_signal))


t = t[:min_length]
ecg_signal = ecg_signal[:min_length]
noisy_signal = noisy_signal[:min_length]
high_pass_signal = high_pass_signal[:min_length]
low_pass_signal = low_pass_signal[:min_length]

# Extract real and imaginary parts of STFT results
stft_real_original = np.real(Zxx_original.flatten())[:min_length]
stft_imag_original = np.imag(Zxx_original.flatten())[:min_length]
stft_real_noisy = np.real(Zxx_noisy.flatten())[:min_length]
stft_imag_noisy = np.imag(Zxx_noisy.flatten())[:min_length]
stft_real_high_pass = np.real(Zxx_high_pass.flatten())[:min_length]
stft_imag_high_pass = np.imag(Zxx_high_pass.flatten())[:min_length]
stft_real_low_pass = np.real(Zxx_low_pass.flatten())[:min_length]
stft_imag_low_pass = np.imag(Zxx_low_pass.flatten())[:min_length]

# Create a DataFrame to store the signals
data = {
    'Time': t,
    'Original': ecg_signal,
    'Noisy': noisy_signal,
    'HighPassFiltered': high_pass_signal,
    'LowPassFiltered': low_pass_signal,
    'STFT_Real_Original': stft_real_original,
    'STFT_Imag_Original': stft_imag_original,
    'STFT_Real_Noisy': stft_real_noisy,
    'STFT_Imag_Noisy': stft_imag_noisy,
    'STFT_Real_HighPass': stft_real_high_pass,
    'STFT_Imag_HighPass': stft_imag_high_pass,
    'STFT_Real_LowPass': stft_real_low_pass,
    'STFT_Imag_LowPass': stft_imag_low_pass
}

df = pd.DataFrame(data)
df.to_csv('ecg_signals_stft.csv', index=False)
```

In [ ]:

In [ ]:

In [46]:
```
from sklearn.metrics import accuracy_score, mean_squared_error
np.random.seed(42)



df = pd.read_csv('ecg_signals_stft.csv')

#We made the assumption to define strong and weak signals throughout the dataset
#Will make predctions based on these random assignments and see if the model can learn.
df['Label'] = np.random.choice(['strong', 'weak'], len(df))


features = [
    'STFT_Real_Original', 'STFT_Imag_Original',
    'STFT_Real_Noisy', 'STFT_Imag_Noisy',
    'STFT_Real_HighPass', 'STFT_Imag_HighPass',
    'STFT_Real_LowPass', 'STFT_Imag_LowPass'
]
X = df[features]
y = df['Label']


y_binary = np.where(y == 'strong', 1, 0)

# Split the data into training, validation, and testing sets (60% train, 20% validation, 20% test)
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Further split training/validation set into training and validation sets (60% train, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.25, random_state=42)
y_train_bin = np.where(y_train == 'strong', 1, 0)
y_val_bin = np.where(y_val == 'strong', 1, 0)
y_test_bin = np.where(y_test == 'strong', 1, 0)


#We will standardize features throughout the code for each model to make sure we use standardized features
#It's important to make sure everything is on the same scale so the model trained can be as accurate possible

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# SVM model with hyperparameter tuning using k-fold cross-validation
param_grid = {
    'C': [0.1, 0.5, 1],
    'gamma': [1, 10, 100],
    'kernel': ['linear', 'rbf']
}

#K-fold CV
kf = KFold(n_splits=5, shuffle=True, random_state=42)

svm_model = GridSearchCV(SVC(), param_grid, cv=kf, scoring='accuracy')
svm_model.fit(X_train_scaled, y_train)


best_params = svm_model.best_params_

# Train the SVM model with the best hyperparameters on the entire training set
best_svm_model = SVC(**best_params)
best_svm_model.fit(X_train_scaled, y_train)
```

```
# Evaluate the best model on the validation set
y_val_pred = best_svm_model.predict(X_val_scaled)


SVMval_accuracy = accuracy_score(y_val, y_val_pred)
SVMval_error = 1 - SVMval_accuracy
SVMmse_val = mean_squared_error(y_val_bin, np.where(y_val_pred == 'strong', 1, 0))


print(f"Best Parameters: {best_params}")
print(f"Validation Accuracy: {SVMval_accuracy:.4f}")
print(f"Validation Error: {SVMval_error:.4f}")
print(f"Validation MSE: {SVMmse_val:.4f}")
```

```
Best Parameters: {'C': 1, 'gamma': 1, 'kernel': 'rbf'}
Validation Accuracy: 0.4980
Validation Error: 0.5020
Validation MSE: 0.5020
```

In [ ]:

In [47]:

```
np.random.seed(42)
# Set up k-fold cross-validation with 5 splits
kf = KFold(n_splits=5, shuffle=True, random_state=42)


#Training, standardizing the model for KNN for this code
#We are using different K values from 1-15 to see which KNN will have the best accuracy


best_k = None
best_accuracy = 0.0
accuracy_scores = []
mse_scores = []


for k in range(1, 16):
    accuracy_scores = []
    mse_scores = []

    # Perform k-fold cross-validation on the
    for i, j in kf.split(X_train_val):
        X_train, X_val = X_train_val.iloc[i], X_train_val.iloc[j]
        y_train, y_val = y_train_val.iloc[i], y_train_val.iloc[j]
        y_train_bin = np.where(y_train == 'strong', 1, 0)
        y_val_bin = np.where(y_val == 'strong', 1, 0)

        #Making sure features are on the same scale, will keep model as accurate as possible
        X_train_scaled = scaler.fit_transform(X_train)
        X_val_scaled = scaler.transform(X_val)

        #Initialzing the KNN model
        knn_model = KNeighborsClassifier(n_neighbors=k)




        knn_model.fit(X_train_scaled, y_train)

        # Make predictions on the validation set
        y_val_pred = knn_model.predict(X_val_scaled)


        val_accuracy = accuracy_score(y_val, y_val_pred)
        mse_val = mean_squared_error(y_val_bin, np.where(y_val_pred == 'strong', 1, 0))


        accuracy_scores.append(val_accuracy)
        mse_scores.append(mse_val)


    avg_accuracy = np.mean(accuracy_scores)
    avg_mse = np.mean(mse_scores)


    print(f"K = {k}: Average Accuracy = {avg_accuracy:.4f}, Average Validation MSE = {avg_mse:.4f}")

    # Logic to find the knn model with the highest accuracy based on the k value
    if avg_accuracy > best_accuracy:
        best_accuracy = avg_accuracy
        best_k = k


print(f"\nBest K found: {best_k} Accuracy = {best_accuracy:.4f}")

# Use the best k to train the final model on the entire training-validation set
X_train_val_scaled = scaler.fit_transform(X_train_val)
knn_best_model = KNeighborsClassifier(n_neighbors=best_k)
knn_best_model.fit(X_train_val_scaled, y_train_val)

# Evaluate the best model on the validation set
X_val_scaled = scaler.transform(X_test)
y_val_pred = knn_best_model.predict(X_val_scaled)
KNNval_accuracy = accuracy_score(y_test, y_val_pred)
KNNval_error = 1 - val_accuracy
KNNmse_val = mean_squared_error(y_test_bin, np.where(y_val_pred == 'strong', 1, 0))


print(f"\nEvaluation on Validation Set with Best K ({best_k}):")
print(f"Accuracy: {KNNval_accuracy:.4f}")
print(f"Validation Error: {KNNval_error:.4f}")
print(f"Validation MSE: {KNNmse_val:.4f}")
```

```
K = 1: Average Accuracy = 0.4900, Average Validation MSE = 0.5100
K = 2: Average Accuracy = 0.5052, Average Validation MSE = 0.4947
K = 3: Average Accuracy = 0.5050, Average Validation MSE = 0.4950
K = 4: Average Accuracy = 0.5047, Average Validation MSE = 0.4953
K = 5: Average Accuracy = 0.5123, Average Validation MSE = 0.4878
K = 6: Average Accuracy = 0.5145, Average Validation MSE = 0.4855
K = 7: Average Accuracy = 0.5142, Average Validation MSE = 0.4858
Loading [MathJax]/extensions/Safe.js e Accuracy = 0.5130, Average Validation MSE = 0.4870
```

```
K = 9: Average Accuracy = 0.5040, Average Validation MSE = 0.4960
K = 10: Average Accuracy = 0.4987, Average Validation MSE = 0.5012
K = 11: Average Accuracy = 0.5018, Average Validation MSE = 0.4982
K = 12: Average Accuracy = 0.5002, Average Validation MSE = 0.4997
K = 13: Average Accuracy = 0.5057, Average Validation MSE = 0.4942
K = 14: Average Accuracy = 0.5055, Average Validation MSE = 0.4945
K = 15: Average Accuracy = 0.5057, Average Validation MSE = 0.4942

Best K found: 6 Accuracy = 0.5145

Evaluation on Validation Set with Best K (6):
Accuracy: 0.5210
Validation Error: 0.4875
Validation MSE: 0.4790
```

In [48]:

```
np.random.seed(42)

nb_model = GaussianNB()


nb_model.fit(X_train, y_train_bin)

# Prediction for the Naive Bayes
y_val_pred = nb_model.predict(X_val)


nbval_accuracy = accuracy_score(y_val_bin, y_val_pred)
nbval_error = 1 - val_accuracy
nbmse_val = mean_squared_error(y_val_bin, y_val_pred)


print(f"Validation Accuracy: {nbval_accuracy:.4f}")
print(f"Validation Error: {nbval_error:.4f}")
print(f"Validation MSE: {nbmse_val:.4f}")
```

```
Validation Accuracy: 0.5088
Validation Error: 0.4875
Validation MSE: 0.4913
```

In [ ]:

In [49]:

```
np.random.seed(42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Logistic regression model with hyperparameter tuning using k-fold cross-validation
param_grid = {
    'C': [0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']
}

# Perform k-fold cross-validation within GridSearchCV
kf = KFold(n_splits=5, shuffle=True, random_state=42)

logreg_model = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=kf, scoring='accuracy')
logreg_model.fit(X_train_scaled, y_train)


best_params = logreg_model.best_params_

# Train the logistic regression model, we are doing it with the hyperparameter tuning
best_logreg_model = LogisticRegression(max_iter=1000, **best_params)
best_logreg_model.fit(X_train_scaled, y_train)

# Evaluate the best model on the validation set
y_val_pred = best_logreg_model.predict(X_val_scaled)


logval_accuracy = accuracy_score(y_val, y_val_pred)
logval_error = 1 - val_accuracy
logmse_val = mean_squared_error(y_val_bin, np.where(y_val_pred == 'strong', 1, 0))


print(f"Best Parameters: {best_params}")
print(f"Validation Accuracy: {logval_accuracy:.4f}")
print(f"Validation Error: {logval_error:.4f}")
print(f"Validation MSE: {logmse_val:.4f}")
```

```
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\linear_model\_sag.py:350: ConvergenceWarning: The max_iter was reached which means the coef_ did not conver
ge
  warnings.warn(
Best Parameters: {'C': 0.1, 'penalty': 'l1', 'solver': 'saga'}
Validation Accuracy: 0.4913
Validation Error: 0.4875
Validation MSE: 0.5088
```

In [50]:

```
np.random.seed(42)

# Random Forest model with hyperparameter tuning
param_grid = {
    'n_estimators': [10,30,50],
    'max_depth': [None],
    'min_samples_split': [1, 2, 5],
    'min_samples_leaf': [1, 2, 4]
}

fold cross-validation within GridSearchCV, this to find best parameters for hyperparameter tuning
```

Loading [MathJax]/extensions/Safe.js

```
kf = KFold(n_splits=5, shuffle=True, random_state=42)

rf_model = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=kf, scoring='accuracy')
rf_model.fit(X_train, y_train)


best_params = rf_model.best_params_

# Train the Random Forest model with the best hyperparameters on the entire training set
best_rf_model = RandomForestClassifier(random_state=42, **best_params)
best_rf_model.fit(X_train, y_train)


y_val_pred = best_rf_model.predict(X_val)


rfval_accuracy = accuracy_score(y_val, y_val_pred)
rfval_error = 1 - val_accuracy
rfmse_val = mean_squared_error(y_val_bin, np.where(y_val_pred == 'strong', 1, 0))


print(f"Best Parameters: {best_params}")
print(f"Validation Accuracy: {rfval_accuracy:.4f}")
print(f"Validation Error: {rfval_error:.4f}")
print(f"Validation MSE: {rfmse_val:.4f}")
```

```
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py:378: FitFailedWarning:
45 fits failed out of a total of 135.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
45 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\model_selection\_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py", line 340, in fit
    self._validate_params()
  File "C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\utils\_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_samples_split' parameter of RandomForestClassifier must be an int in the range [2, inf) or a f
loat in the range (0.0, 1.0]. Got 1 instead.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\saipa\anaconda3\Lib\site-packages\sklearn\model_selection\_search.py:952: UserWarning: One or more of the test scores are non-finite: [      nan
nan       nan 0.4903125 0.4990625 0.494375  0.5003125
 0.5040625 0.51            nan       nan       nan 0.5109375 0.511875
 0.5096875 0.5075    0.4990625 0.503125        nan       nan       nan
 0.5009375 0.500625  0.500625  0.5009375 0.500625  0.500625 ]
  warnings.warn(
Best Parameters: {'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 30}
Validation Accuracy: 0.5025
Validation Error: 0.4875
Validation MSE: 0.4975
```

In [51]:
```
data = {
    'Model': ['SVM', 'KNN', 'Naive Bayes', 'Logistic Regression', 'Random Forest'],
    'Validation Accuracy': [SVMval_accuracy, KNNval_accuracy, nbval_accuracy, logval_accuracy, rfval_accuracy],
    'Validation MSE': [SVMmse_val, KNNmse_val, nbmse_val, logmse_val, rfmse_val]
}

df = pd.DataFrame(data)


print(df)
```

```
                 Model  Validation Accuracy  Validation MSE
0                  SVM              0.49800         0.50200
1                  KNN              0.52100         0.47900
2          Naive Bayes              0.50875         0.49125
3  Logistic Regression              0.49125         0.50875
4        Random Forest              0.50250         0.49750
```

In [ ]:

In [ ]:

In [52]:
```
# Evaluate the best model on the test set, best model is the KNN model based on the results in dataframe above
X_test_scaled = scaler.transform(X_test)
y_test_pred = knn_best_model.predict(X_test_scaled)
KNNtest_accuracy = accuracy_score(y_test, y_test_pred)
KNNtest_error = 1 - KNNtest_accuracy
KNNmse_test = mean_squared_error(y_test_bin, np.where(y_test_pred == 'strong', 1, 0))

print(f"\nEvaluation on Test Set with Best K ({best_k}):")
print(f"Accuracy: {KNNtest_accuracy:.4f}")
print(f"Test MSE: {KNNmse_test:.4f}")
```

```
Evaluation on Test Set with Best K (6):
Accuracy: 0.5180
Test MSE: 0.4820
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: