In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/dock
er-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list a
ll files under the input directory

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

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved ou
tside of the current session
```

```
/kaggle/input/indian-subcontinent-earthquake-data-2000-to-2024/Earthquakes.c
sv
```

In [2]:

```
df=pd.read_csv("/kaggle/input/indian-subcontinent-earthquake-data-2000-to-2024/E
arthquakes.csv")
```

In [3]:

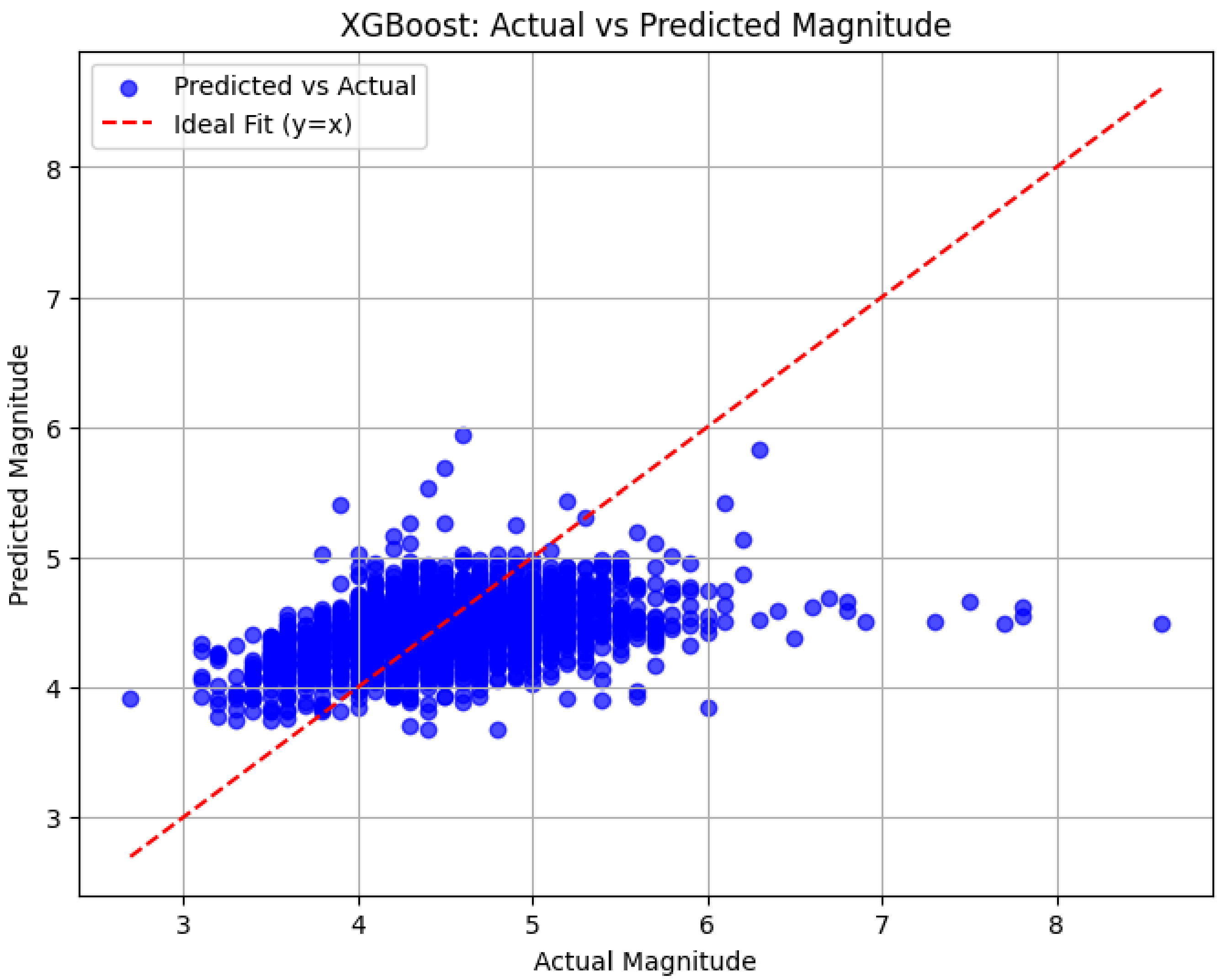
```
from sklearn.model_selection import train_test_split

features=['latitude', 'longitude', 'depth']
target='mag'

X = df[features]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
state=42)
```

In [4]:

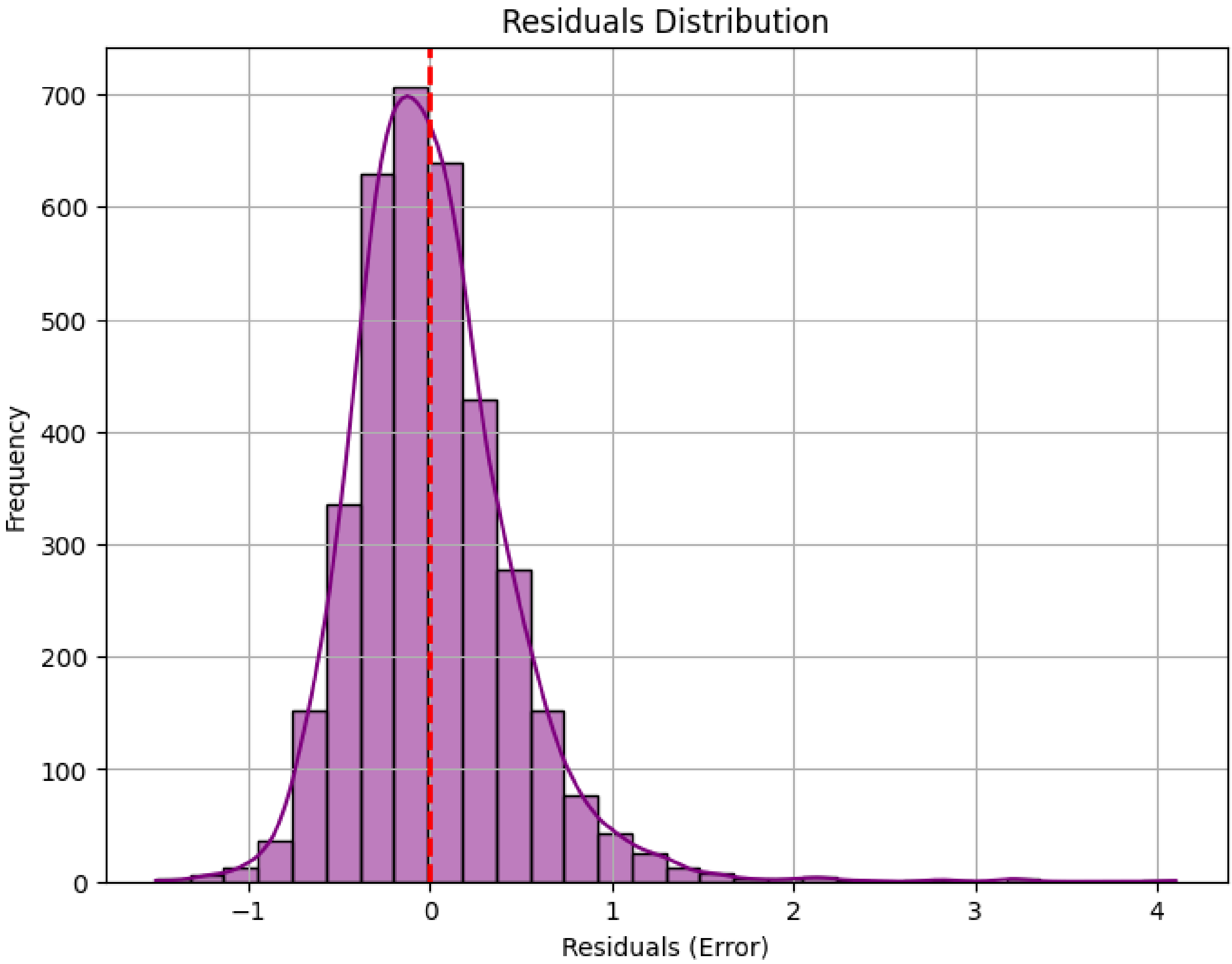
```python
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Train the XGBoost model
model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R-squared Score

# Calculate Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)  # Ensures accuracy is non-negative

# Print Scores
print(f'Mean Absolute Error (MAE): {mae:.4f}')
print(f'Mean Squared Error (MSE): {mse:.4f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.4f}')
print(f'R-squared Score (R²): {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Actual vs Predicted Magnitude
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("XGBoost: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Residuals Plot
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
```

```python
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
plt.grid()
plt.show()

# Plot 3: Feature Importance
plt.figure(figsize=(8, 6))
xgb.plot_importance(model, importance_type='weight', title='Feature Importance')
plt.show()
```

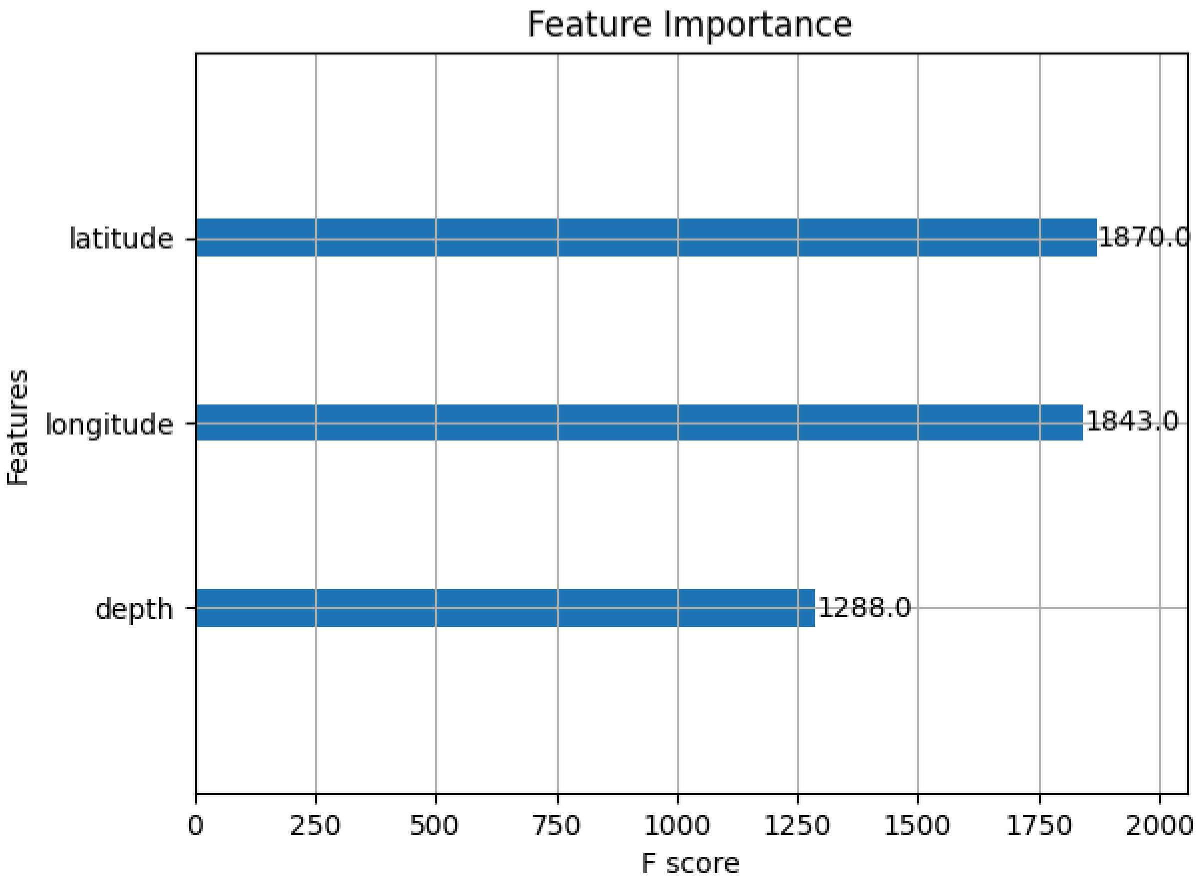
```
Mean Absolute Error (MAE): 0.3235
Mean Squared Error (MSE): 0.1925
Root Mean Squared Error (RMSE): 0.4387
R-squared Score (R²): 0.1415
Accuracy: 92.76%
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



<Figure size 800x600 with 0 Axes>

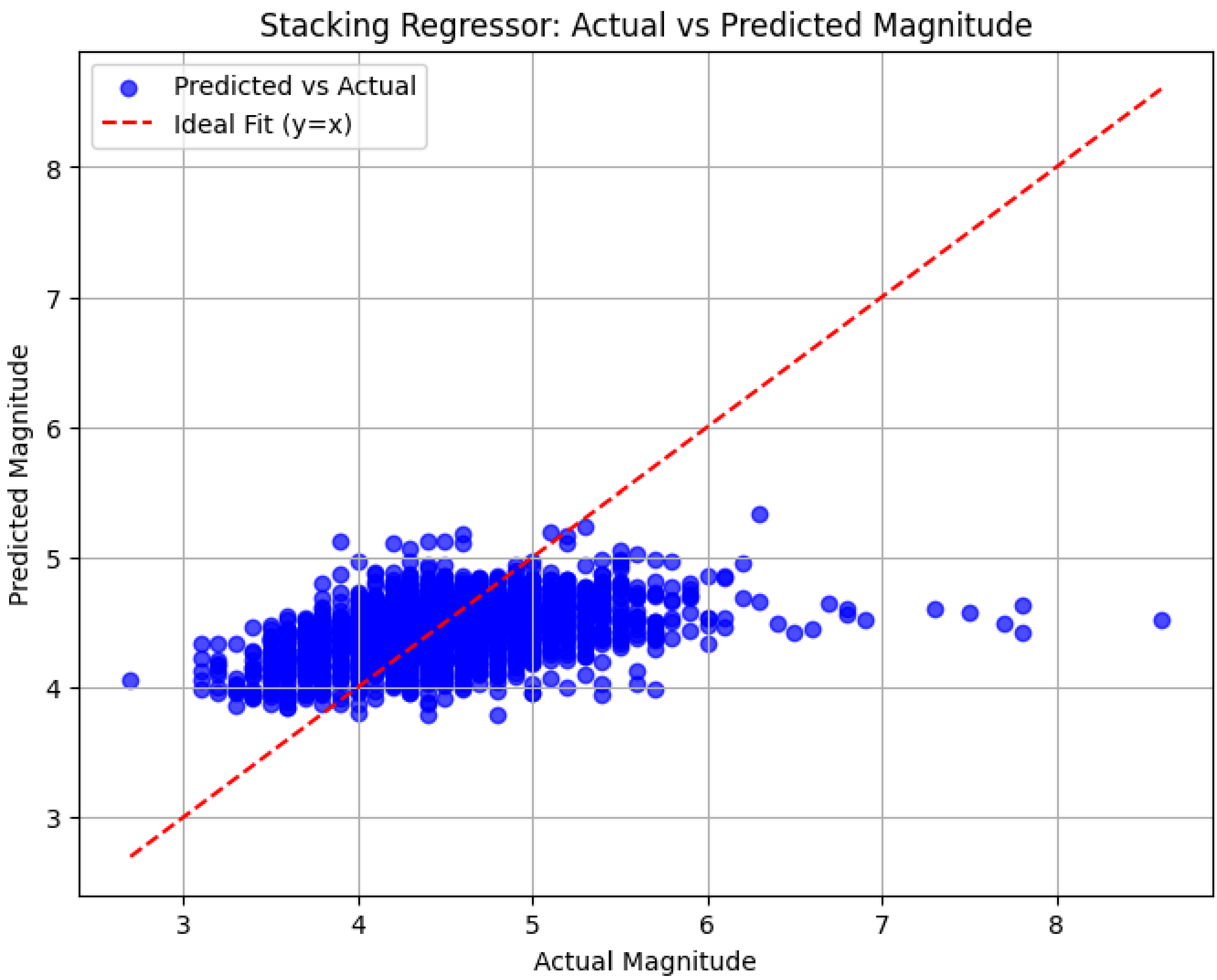

Feature Importance
latitude
longitude
depth
1870.0
1843.0
1288.0
Features
0
250
500
750
1000
1250
1500
1750
2000
F score

In [5]:

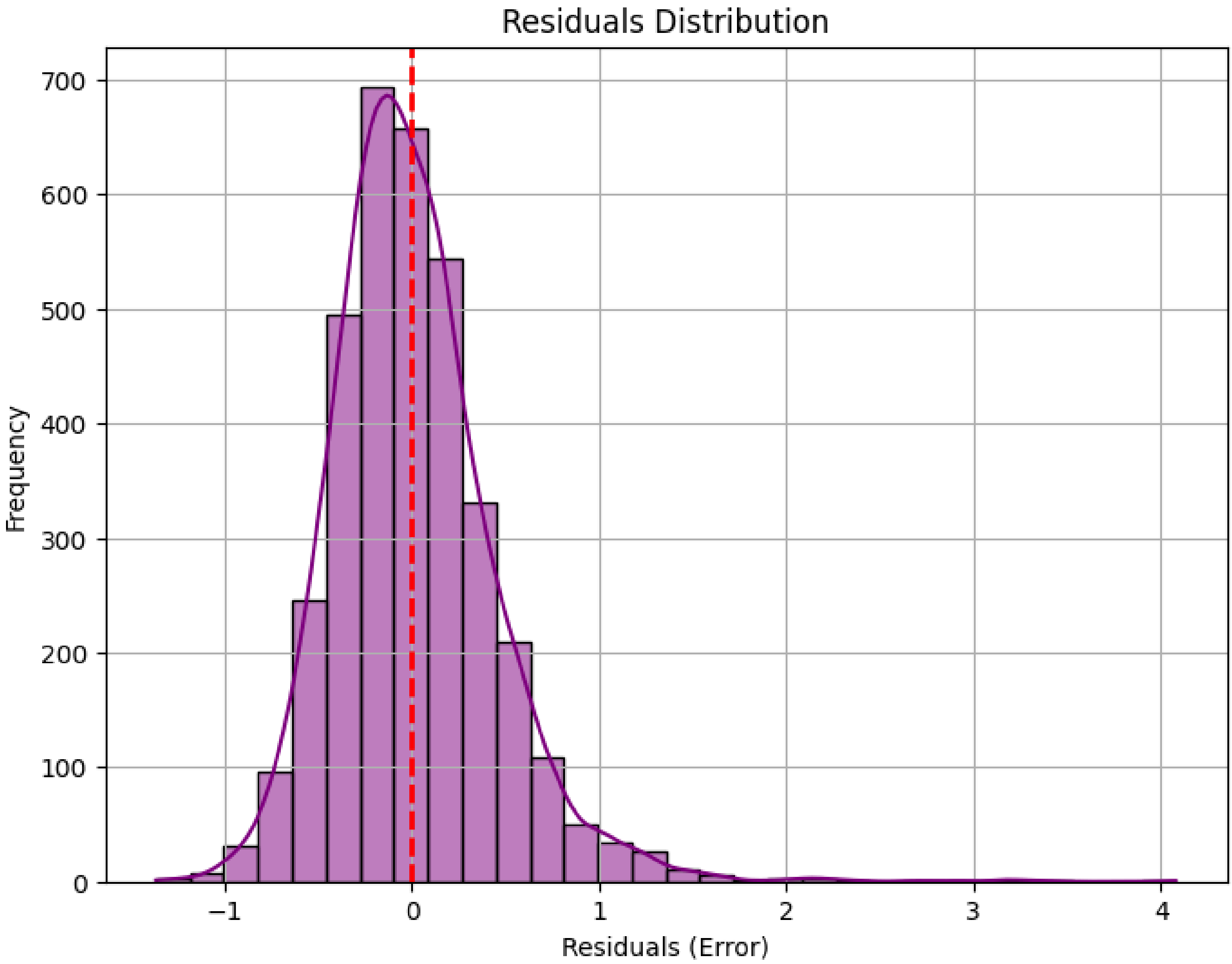
```
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define base models
base_models = [
    ('dt', DecisionTreeRegressor(max_depth=5)),
    ('rf', RandomForestRegressor(n_estimators=100, random_state=42)),
    ('svr', SVR(kernel='rbf'))
]

# Meta model
meta_model = Ridge(alpha=1.0)

# Create Stacking Regressor
stacking_regressor = StackingRegressor(estimators=base_models, final_estimator=m
eta_model)

# Train the model
stacking_regressor.fit(X_train, y_train)

# Make predictions
y_pred = stacking_regressor.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)

# Print Scores
print(f'MAE: {mae:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R² Score: {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Actual vs Predicted
```

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actua
l')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', li
nestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("Stacking Regressor: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Residuals
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
plt.grid()
plt.show()

# Plot 3: Error Spread
plt.figure(figsize=(8, 6))
plt.scatter(range(len(y_test)), residuals, alpha=0.5, color='green')
plt.axhline(0, color='red', linestyle='dashed')
plt.xlabel("Test Samples")
plt.ylabel("Residuals")
plt.title("Error Spread Across Test Samples")
plt.grid()
plt.show()
```

```
MAE: 0.3228
MSE: 0.1908
RMSE: 0.4369
R² Score: 0.1488
Accuracy: 92.78%
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Residuals Distribution
Frequency
700
600
500
400
300
200
100
0
−1
0
1
2
3
4
Residuals (Error)

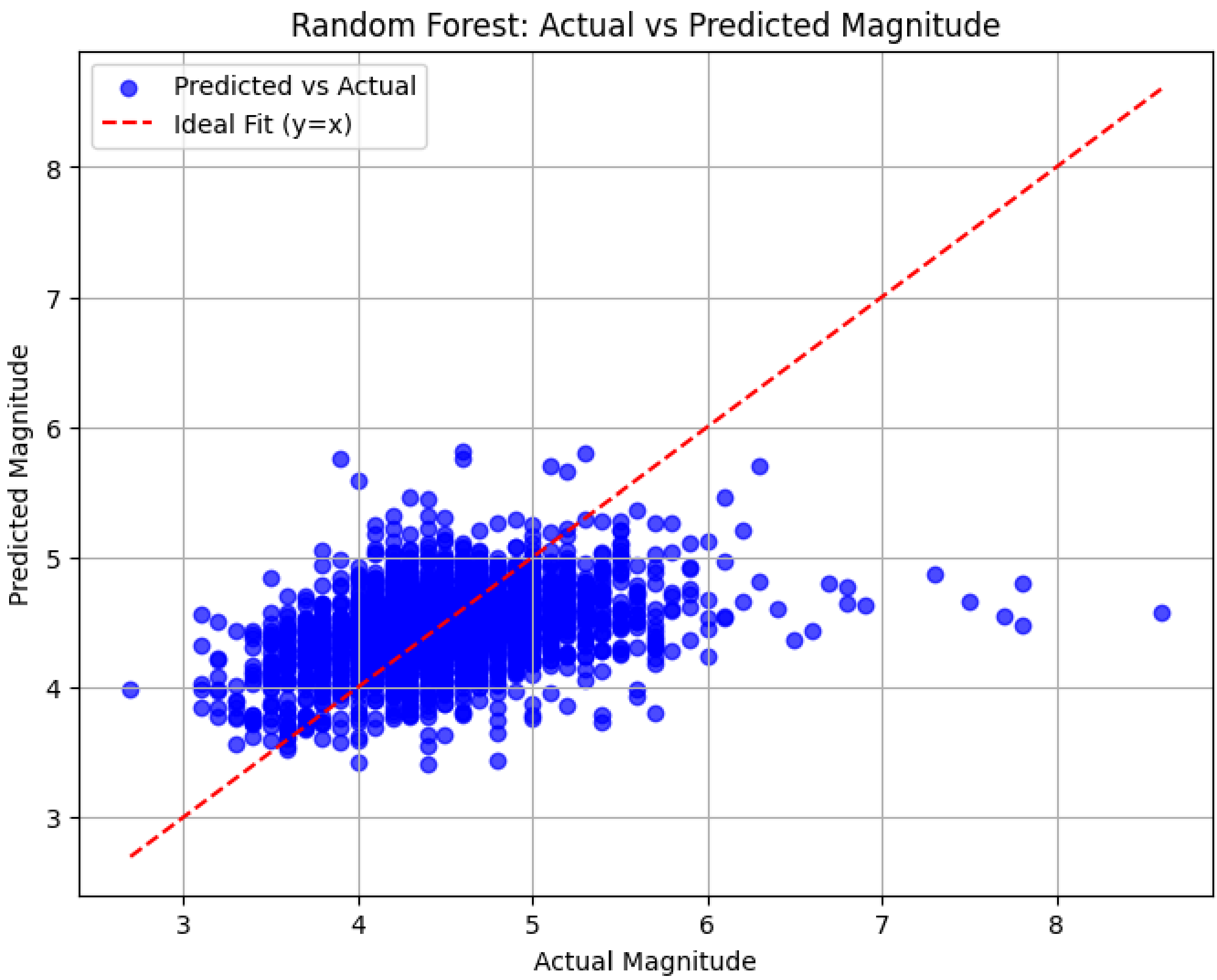
Error Spread Across Test Samples
Residuals
Test Samples
4
3
2
1
0
−1
0
500
1000
1500
2000
2500
3000
3500

In [6]:

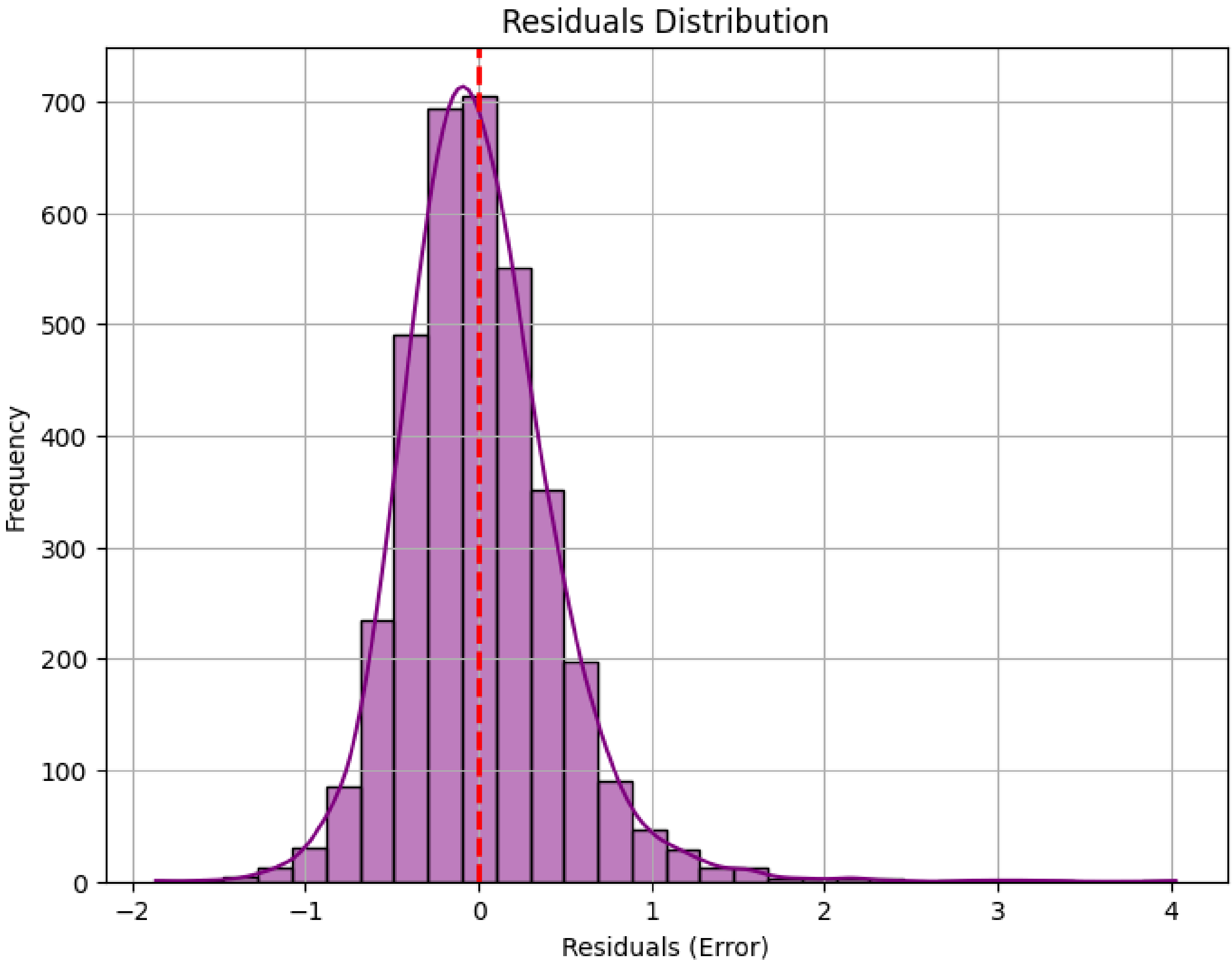
```python
# Define and train the Random Forest model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)
rf_regressor.fit(X_train, y_train)

# Make predictions
y_pred = rf_regressor.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)

# Print Scores
print(f'MAE: {mae:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R² Score: {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("Random Forest: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Residuals
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
plt.grid()
plt.show()
```

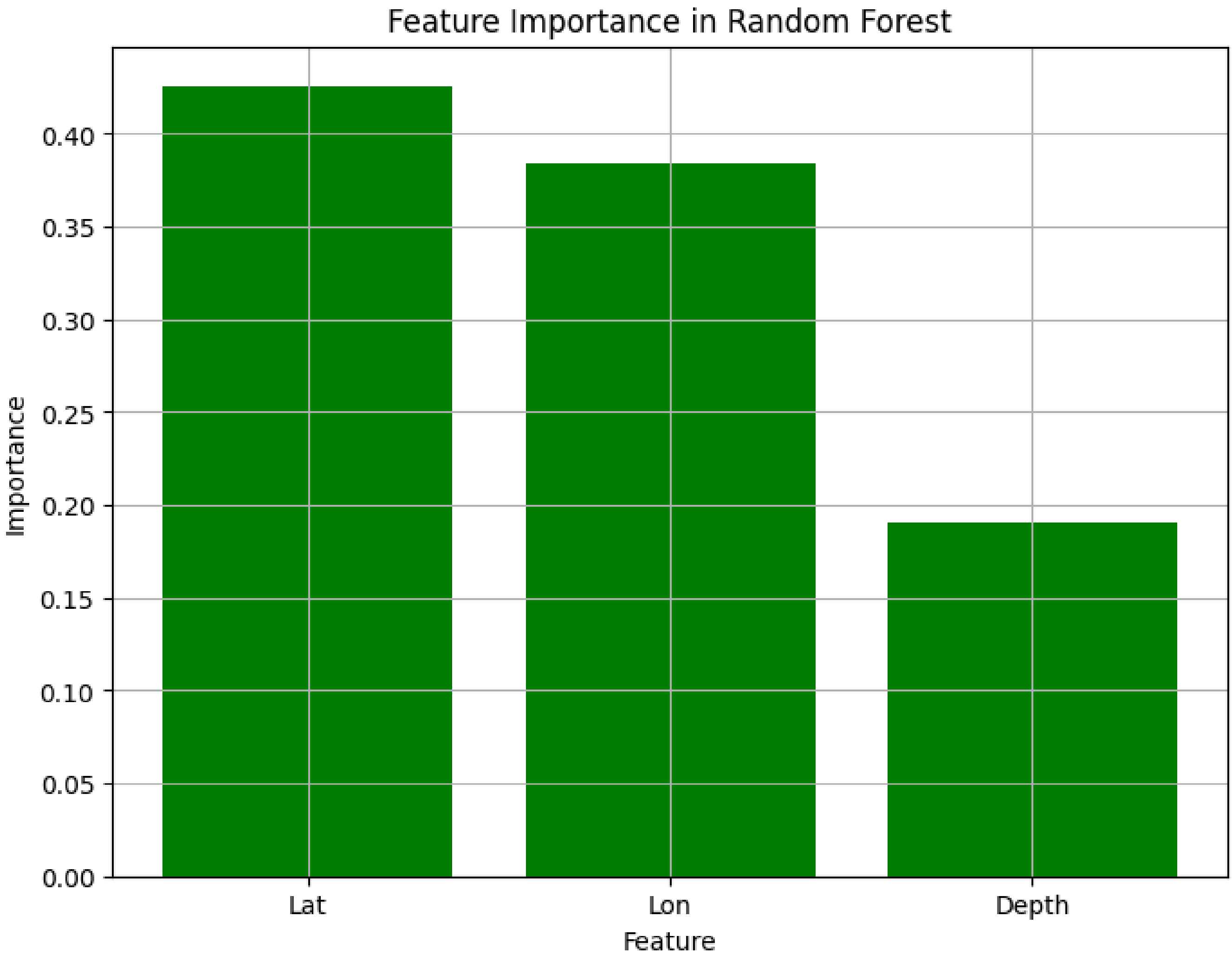
```python
# Plot 3: Feature Importance
plt.figure(figsize=(8, 6))
importances = rf_regressor.feature_importances_
plt.bar(range(len(importances)), importances, tick_label=["Lat", "Lon", "Dept
h"], color='green')
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance in Random Forest")
plt.grid()
plt.show()
```

```
MAE: 0.3333
MSE: 0.2025
RMSE: 0.4501
R² Score: 0.0966
Accuracy: 92.54%
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Residuals Distribution
Frequency
Residuals (Error)
0
100
200
300
400
500
600
700
−2
−1
0
1
2
3
4


Feature Importance in Random Forest
Importance
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Lat
Lon
Depth
Feature

In [7]:

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Reshape input for LSTM (samples, timesteps, features)
X_train_lstm = np.reshape(X_train.values, (X_train.shape[0], 1, X_train.shape
[1]))
X_test_lstm = np.reshape(X_test.values, (X_test.shape[0], 1, X_test.shape[1]))

# Define the LSTM model
model = Sequential([
    LSTM(50, return_sequences=True, input_shape=(1, X_train.shape[1])),
    Dropout(0.2),
    LSTM(50, return_sequences=False),
    Dropout(0.2),
    Dense(25, activation='relu'),
    Dense(1)  # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='mse')

# Train the model
history = model.fit(X_train_lstm, y_train, epochs=50, batch_size=16, validation_
data=(X_test_lstm, y_test), verbose=1)

# Make predictions
y_pred = model.predict(X_test_lstm).flatten()

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)

# Print Scores
print(f'MAE: {mae:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R² Score: {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Loss Curve
```

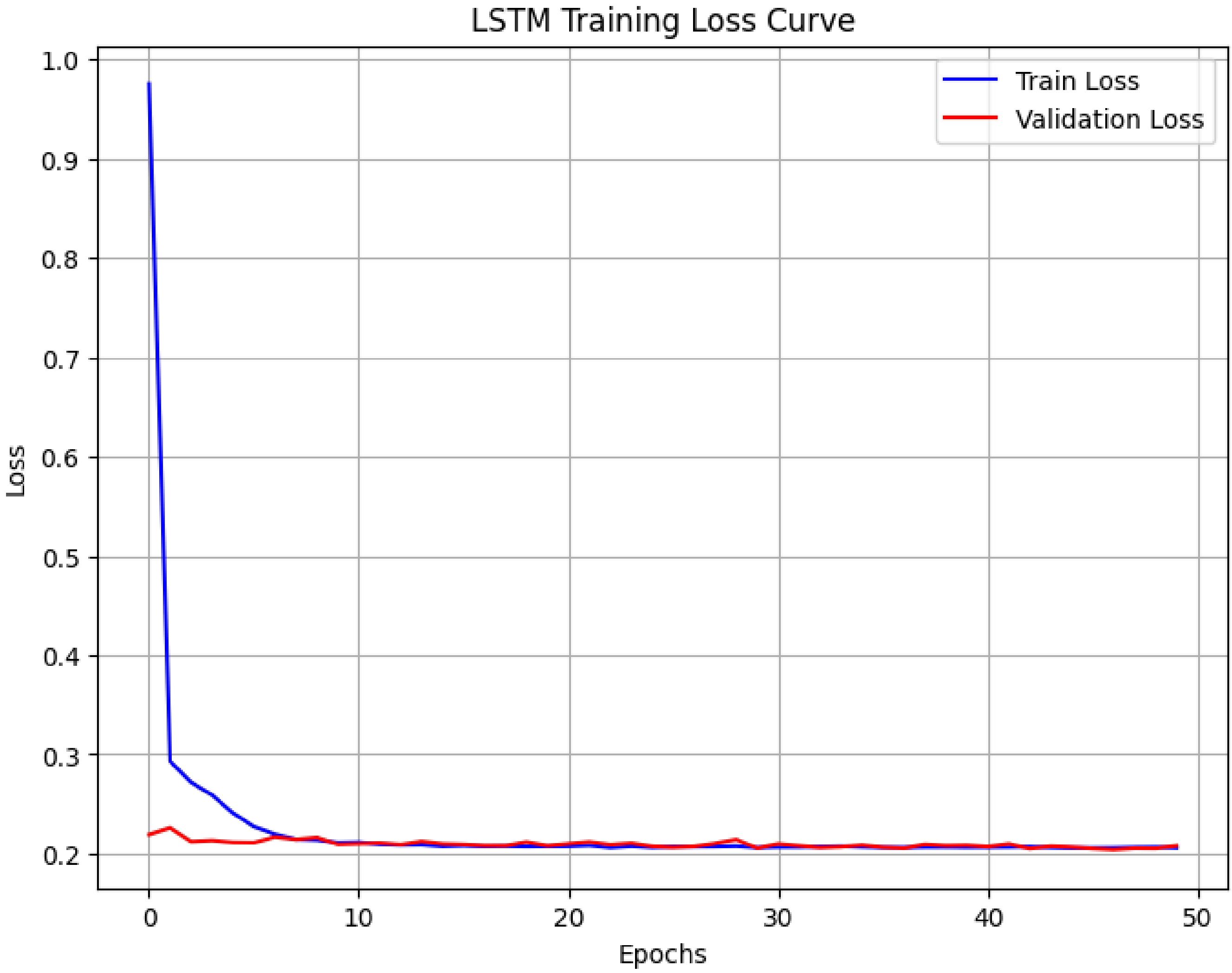
```
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='Train Loss', color='blue')
plt.plot(history.history['val_loss'], label='Validation Loss', color='red')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("LSTM Training Loss Curve")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actua
l')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', li
nestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("LSTM: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 3: Residuals
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
plt.grid()
plt.show()
```

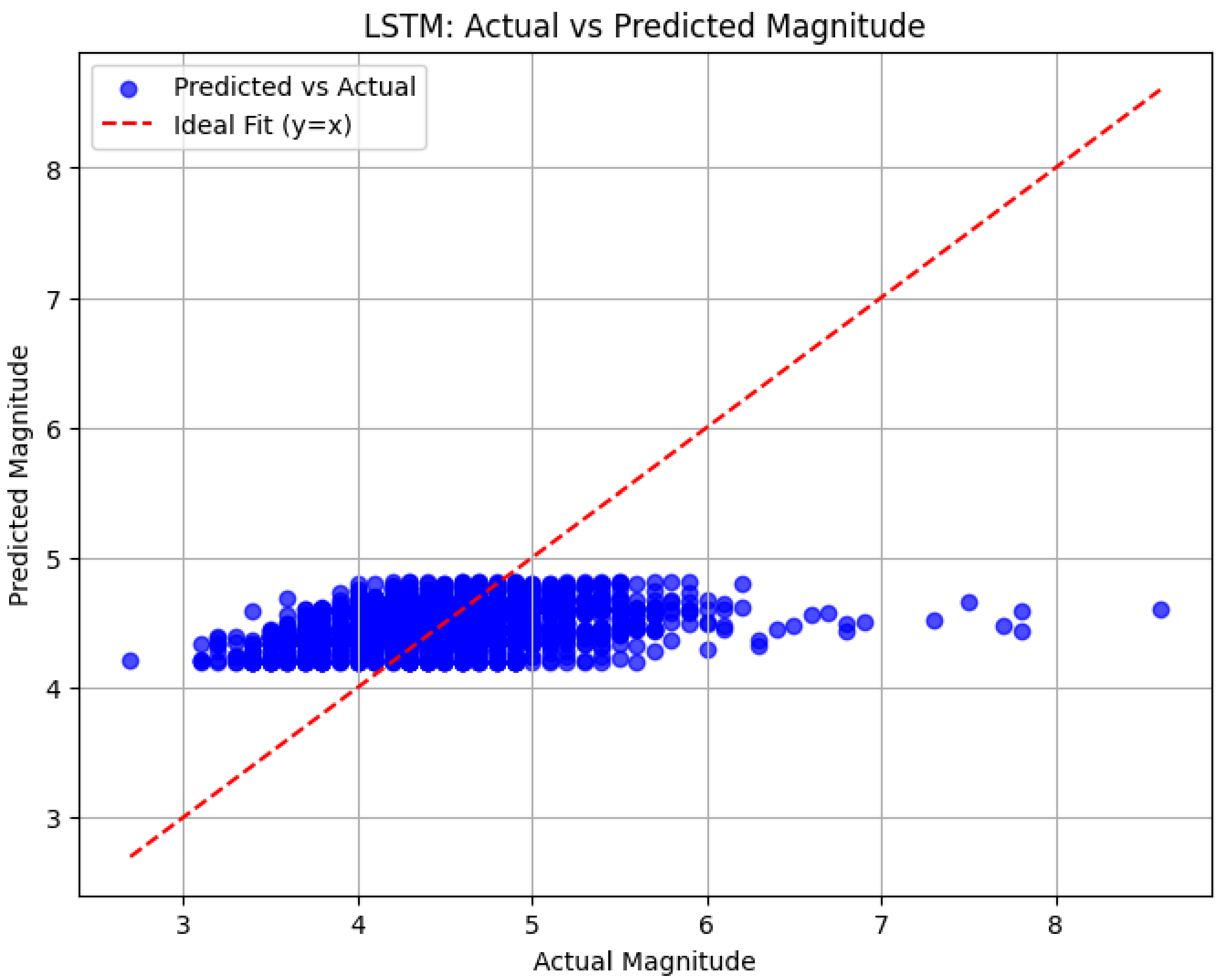
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: Use
rWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

```
Epoch 1/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - loss: 3.0251 - val_loss: 0.2192
Epoch 2/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2932 - val_loss: 0.2260
Epoch 3/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2729 - val_loss: 0.2121
Epoch 4/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2610 - val_loss: 0.2132
Epoch 5/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2492 - val_loss: 0.2112
Epoch 6/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2312 - val_loss: 0.2110
Epoch 7/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2174 - val_loss: 0.2168
Epoch 8/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2184 - val_loss: 0.2142
Epoch 9/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2122 - val_loss: 0.2163
Epoch 10/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2084 - val_loss: 0.2093
Epoch 11/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2110 - val_loss: 0.2099
Epoch 12/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2001 - val_loss: 0.2107
Epoch 13/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2097 - val_loss: 0.2091
Epoch 14/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2053 - val_loss: 0.2123
Epoch 15/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2067 - val_loss: 0.2098
Epoch 16/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2063 - val_loss: 0.2092
Epoch 17/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2049 - val_loss: 0.2081
Epoch 18/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2081 - val_loss: 0.2081
Epoch 19/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2124 - val_loss: 0.2116
Epoch 20/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2081 - val_loss: 0.2080
Epoch 21/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2095 - val_loss: 0.2099
Epoch 22/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2063 - val_loss: 0.2118
Epoch 23/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2097 - val_loss: 0.2090
```

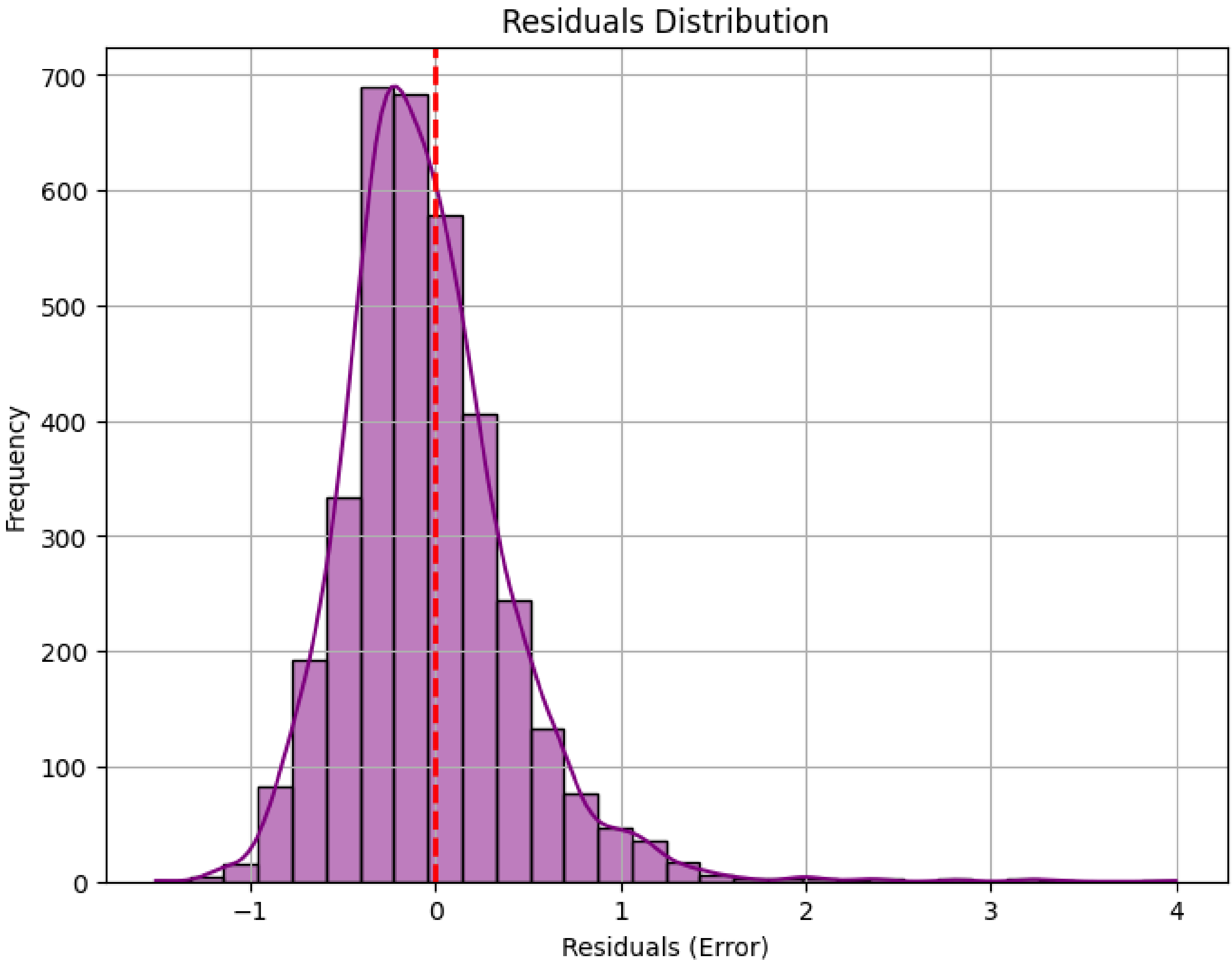
```
Epoch 24/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2057 - val_loss: 0.2105
Epoch 25/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2001 - val_loss: 0.2073
Epoch 26/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2018 - val_loss: 0.2062
Epoch 27/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2026 - val_loss: 0.2072
Epoch 28/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2124 - val_loss: 0.2103
Epoch 29/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2088 - val_loss: 0.2142
Epoch 30/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2047 - val_loss: 0.2054
Epoch 31/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2092 - val_loss: 0.2098
Epoch 32/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2080 - val_loss: 0.2079
Epoch 33/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2104 - val_loss: 0.2060
Epoch 34/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.1996 - val_loss: 0.2068
Epoch 35/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2070 - val_loss: 0.2086
Epoch 36/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2063 - val_loss: 0.2063
Epoch 37/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2076 - val_loss: 0.2054
Epoch 38/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.2055 - val_loss: 0.2093
Epoch 39/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2035 - val_loss: 0.2081
Epoch 40/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2120 - val_loss: 0.2084
Epoch 41/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2054 - val_loss: 0.2070
Epoch 42/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2071 - val_loss: 0.2099
Epoch 43/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2077 - val_loss: 0.2050
Epoch 44/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2038 - val_loss: 0.2074
Epoch 45/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2021 - val_loss: 0.2065
Epoch 46/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2088 - val_loss: 0.2051
Epoch 47/50
```

```
891/891 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - loss: 0.2086 - val_loss: 0.2040
Epoch 48/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2043 - val_loss: 0.2054
Epoch 49/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2057 - val_loss: 0.2052
Epoch 50/50
891/891 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2058 - val_loss: 0.2078
112/112 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
MAE: 0.3420
MSE: 0.2078
RMSE: 0.4559
R² Score: 0.0730
Accuracy: 92.35%
```






```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

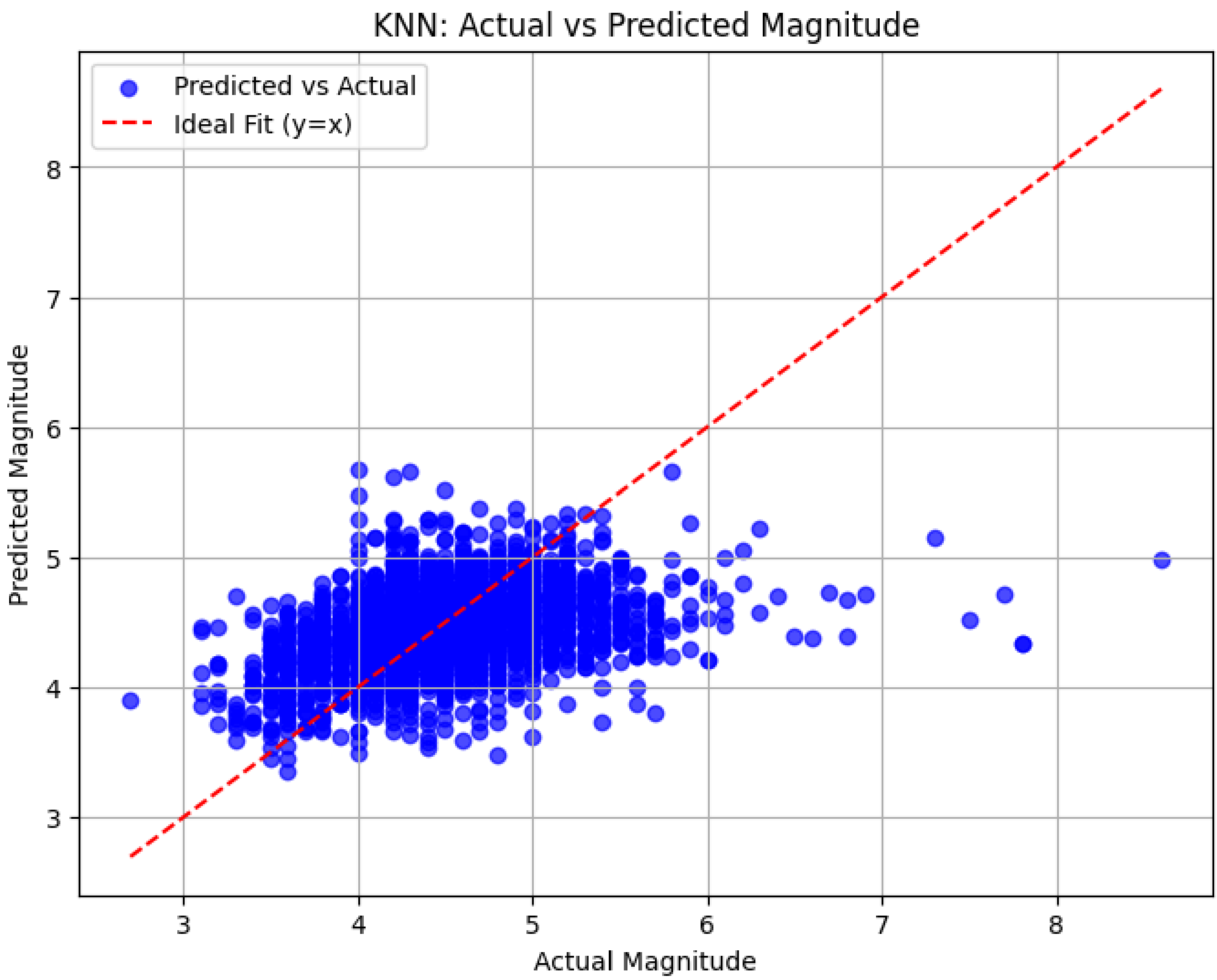

Residuals Distribution
Frequency
700
600
500
400
300
200
100
0
−1
0
1
2
3
4
Residuals (Error)

In [8]:

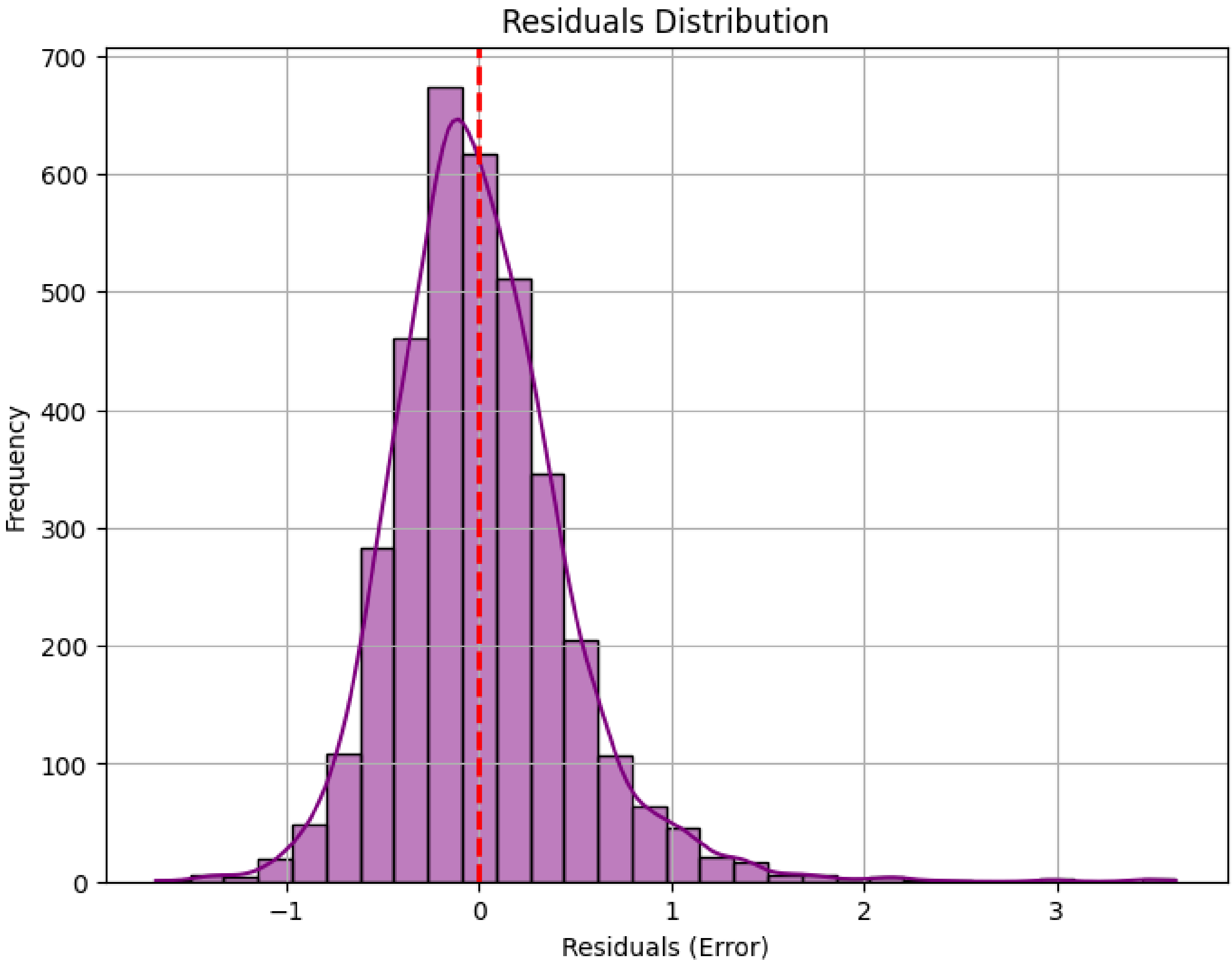
```
from sklearn.neighbors import KNeighborsRegressor

# Define and train the KNN model
knn_regressor = KNeighborsRegressor(n_neighbors=5)  # Using k=5
knn_regressor.fit(X_train, y_train)

# Make predictions
y_pred = knn_regressor.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)

# Print Scores
print(f'MAE: {mae:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R² Score: {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actua
l')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', li
nestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("KNN: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Residuals Distribution
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
plt.title("Residuals Distribution")
```

```
plt.grid()
plt.show()

# Plot 3: Error vs Neighbors
neighbors = list(range(1, 20))
errors = []

for k in neighbors:
    knn = KNeighborsRegressor(n_neighbors=k)
    knn.fit(X_train, y_train)
    pred_k = knn.predict(X_test)
    errors.append(mean_absolute_error(y_test, pred_k))

plt.figure(figsize=(8, 6))
plt.plot(neighbors, errors, marker='o', linestyle='dashed', color='green')
plt.xlabel("Number of Neighbors (k)")
plt.ylabel("MAE")
plt.title("KNN: MAE vs Number of Neighbors")
plt.grid()
plt.show()
```

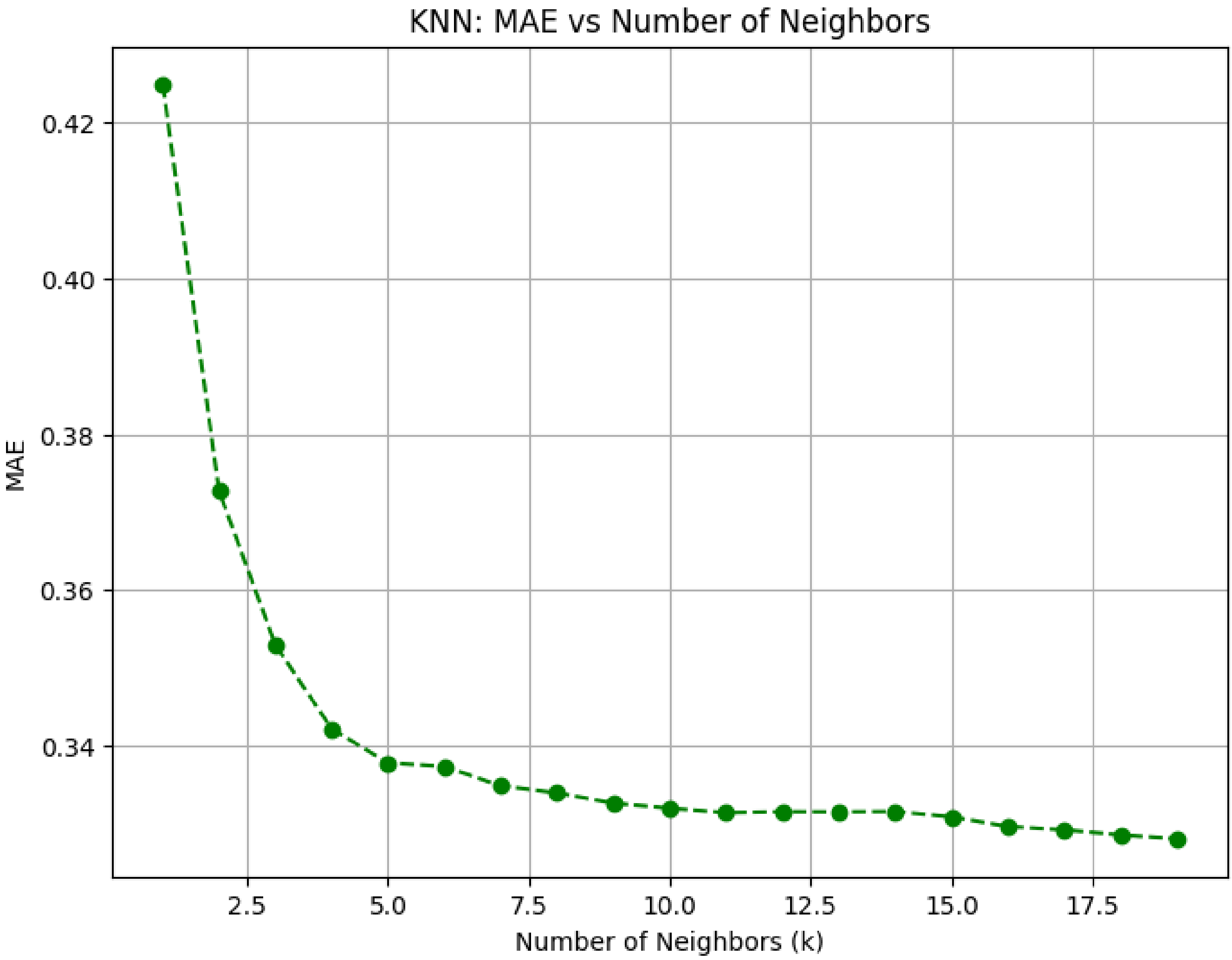
```
MAE: 0.3379
MSE: 0.2089
RMSE: 0.4571
R² Score: 0.0681
Accuracy: 92.44%
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Residuals Distribution
Frequency
0
100
200
300
400
500
600
700
−1
0
1
2
3
Residuals (Error)

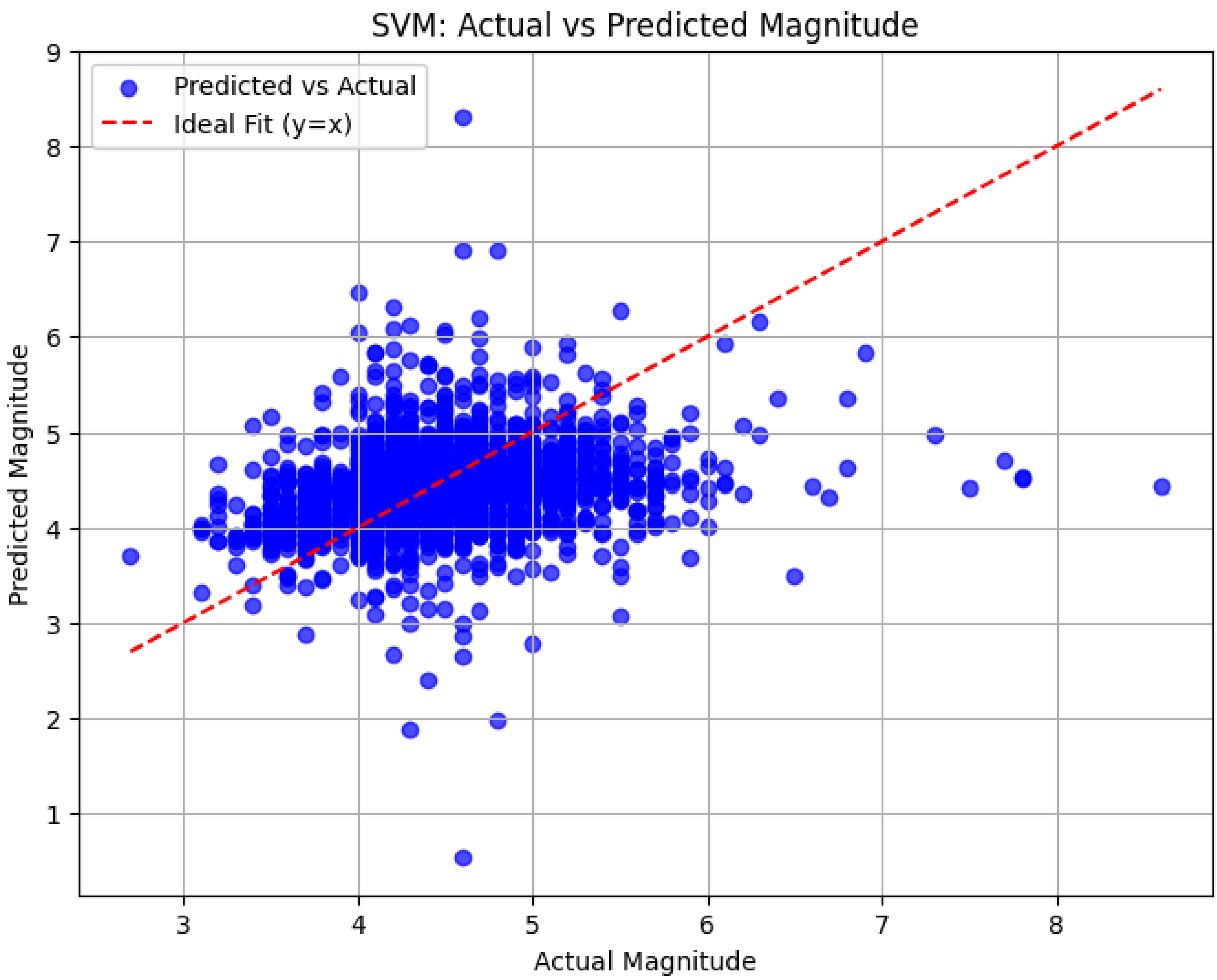

KNN: MAE vs Number of Neighbors
0.42
0.40
0.38
0.36
0.34
MAE
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Neighbors (k)

In [9]:

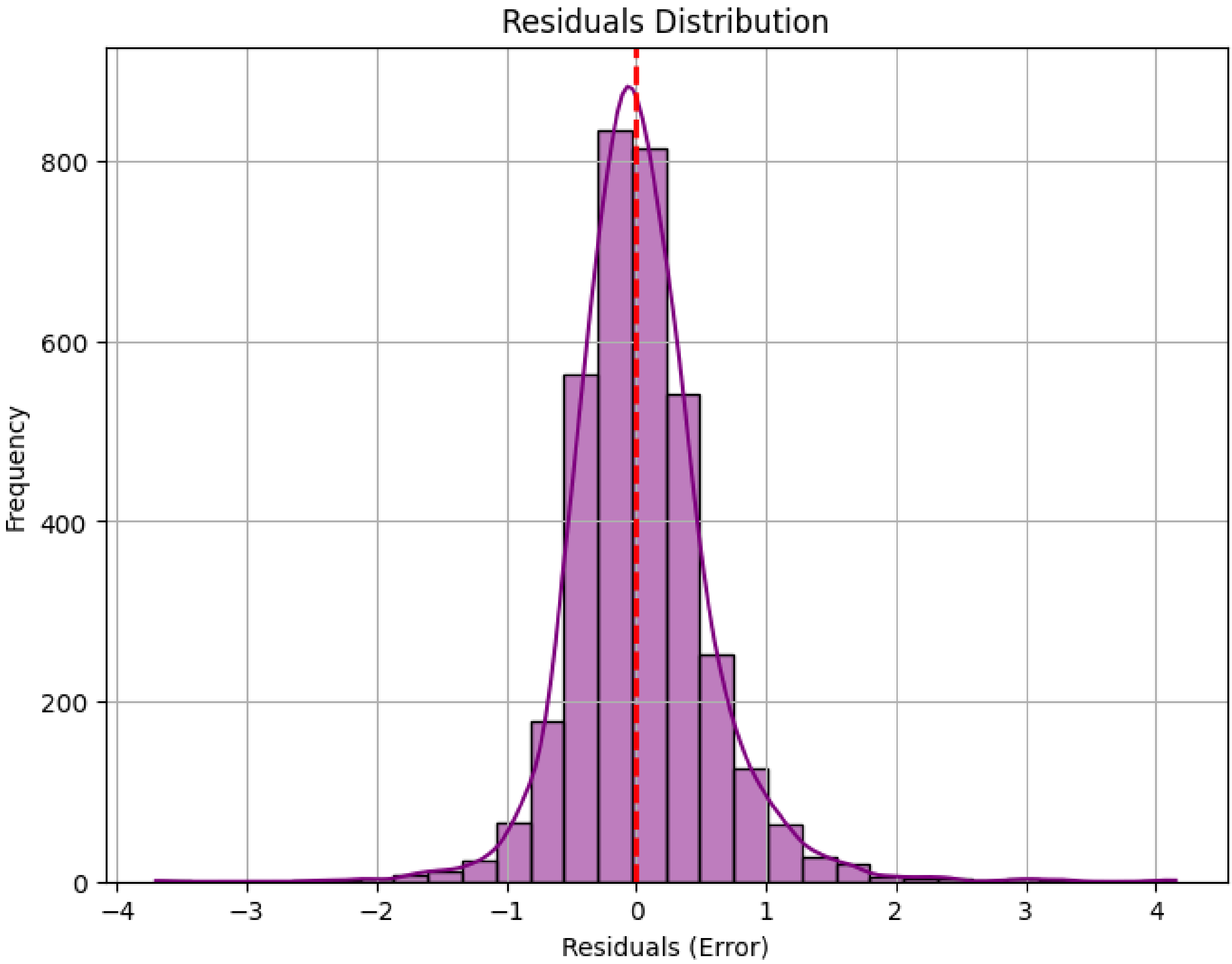
```
from sklearn.svm import SVR

# Define and train the SVM model
svm_regressor = SVR(kernel='rbf', C=100, gamma=0.1)  # RBF kernel for non-linear data
svm_regressor.fit(X_train, y_train)

# Make predictions
y_pred = svm_regressor.predict(X_test)

# Calculate metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5  # Root Mean Squared Error
r2 = r2_score(y_test, y_pred)  # R² Score

# Accuracy in Percentage
accuracy = max(0, (1 - (mae / y_test.mean())) * 100)

# Print Scores
print(f'MAE: {mae:.4f}')
print(f'MSE: {mse:.4f}')
print(f'RMSE: {rmse:.4f}')
print(f'R² Score: {r2:.4f}')
print(f'Accuracy: {accuracy:.2f}%')

# Plot 1: Actual vs Predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.7, color='blue', label='Predicted vs Actual')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed', label='Ideal Fit (y=x)')
plt.xlabel("Actual Magnitude")
plt.ylabel("Predicted Magnitude")
plt.title("SVM: Actual vs Predicted Magnitude")
plt.legend()
plt.grid()
plt.show()

# Plot 2: Residuals Distribution
residuals = y_test - y_pred
plt.figure(figsize=(8, 6))
sns.histplot(residuals, bins=30, kde=True, color='purple')
plt.axvline(0, color='red', linestyle='dashed', linewidth=2)
plt.xlabel("Residuals (Error)")
plt.ylabel("Frequency")
```

```
plt.title("Residuals Distribution")
plt.grid()
plt.show()
```

```
MAE: 0.3759
MSE: 0.2785
RMSE: 0.5278
R² Score: -0.2422
Accuracy: 91.59%
```




```
/usr/local/lib/python3.10/dist-packages/seaborn/_oldcore.py:1119: FutureWarn
ing: use_inf_as_na option is deprecated and will be removed in a future vers
ion. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Residuals Distribution
Frequency
0
200
400
600
800
−4
−3
−2
−1
0
1
2
3
4
Residuals (Error)