```
import praw
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.model_selection import train_test_split
import joblib
import warnings
```

```
WARNING:tensorflow:From C:\Users\Hp\anaconda3\lib\site-packages\keras\
src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is
deprecated. Please use
tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

```
warnings.filterwarnings("ignore")
nltk.download("stopwords")
nltk.download("wordnet")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Hp\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Hp\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
True
```

```
# Authentication Function
def authenticate_reddit(client_id, client_secret, user_agent):
    try:
        reddit = praw.Reddit(
            client_id=client_id,
            client_secret=client_secret,
            user_agent=user_agent,
        )
        print("Authenticated successfully!")
        return reddit
    except Exception as e:
        print("Authentication failed:", e)
        return None

# Data Scraping
def scrape_reddit_data(reddit, subreddit_name, keyword=None,
limit=100):
```

```
    try:
        subreddit = reddit.subreddit(subreddit_name)
        posts = subreddit.search(keyword, limit=limit) if keyword else
subreddit.hot(limit=limit)
        data = []

        for post in posts:
            post.comments.replace_more(limit=0)
            comments = " ".join([comment.body for comment in
post.comments.list()])
            data.append({
                "Title": post.title,
                "Comments": comments,
                "Score": post.score,
                "Sentiment Text": post.title + " " + comments,
                "Created At": pd.to_datetime(post.created_utc,
unit="s"),
            })

        return pd.DataFrame(data)
    except Exception as e:
        print("Error during scraping:", e)
        return pd.DataFrame()

# Text Preprocessing
def preprocess_text(df):
    stop_words = set(stopwords.words("english"))
    lemmatizer = WordNetLemmatizer()

    df["Cleaned Text"] = df["Sentiment Text"].str.lower()
    df["Cleaned Text"] = df["Cleaned Text"].apply(
        lambda x: " ".join([lemmatizer.lemmatize(word) for word in
x.split() if word not in stop_words])
    )
    return df

# Sentiment Analysis
def analyze_sentiment(df):
    analyzer = SentimentIntensityAnalyzer()
    df["Sentiment Score"] = df["Cleaned Text"].apply(lambda x:
analyzer.polarity_scores(x)["compound"])
    return df

# Feature Engineering
def add_features(df):
    df["Hour"] = df["Created At"].dt.hour
    df["Day"] = df["Created At"].dt.day_name()
    df["Comment Length"] = df["Comments"].str.len()
    return df
```

```
# Model Training
def train_hybrid_model(df):
    # Define target (upward movement if sentiment score > 0.2)
    df["Stock Movement"] = (df["Sentiment Score"] > 0.2).astype(int)

    # Features and target
    features = ["Sentiment Score", "Score", "Comment Length"]
    X = df[features]
    y = df["Stock Movement"]

    # Split and normalize data
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Random Forest for feature importance
    rf_model = RandomForestClassifier()
    rf_model.fit(X_train, y_train)
    print("Random Forest Report:")
    print(classification_report(y_test, rf_model.predict(X_test)))

    # LSTM for sequential data
    X_train_lstm = X_train.reshape((X_train.shape[0], 1,
X_train.shape[1]))
    X_test_lstm = X_test.reshape((X_test.shape[0], 1,
X_test.shape[1]))

    lstm_model = Sequential([
        LSTM(50, input_shape=(X_train_lstm.shape[1],
X_train_lstm.shape[2])),
        Dense(1, activation="sigmoid"),
    ])
    lstm_model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"])
    lstm_model.fit(X_train_lstm, y_train, epochs=10, batch_size=16,
verbose=1)

    print("LSTM Evaluation:")
    lstm_model.evaluate(X_test_lstm, y_test)

    return rf_model, lstm_model, scaler

# Saving Models
def save_models(rf_model, lstm_model, scaler):
    joblib.dump(rf_model, "random_forest_model.pkl")
    joblib.dump(scaler, "scaler.pkl")
    lstm_model.save("lstm_model.h5")
    print("Models saved successfully.")
```

```python
def main():
    # Reddit API credentials
    client_id = "q9C9_B3Km7S9L4rofTLUvw"
    client_secret = "fy7yX-94w8usjBQaxspmrbCb_dHD5A"
    user_agent = "AdditionalCoast770"

    # Authenticating with Reddit
    reddit = authenticate_reddit(client_id, client_secret, user_agent)

    # Scrape data from Reddit
    data = scrape_reddit_data(reddit, "StockMarket", keyword="SAIL",
limit=100)

    if not data.empty:
        # Preprocess text data
        data = preprocess_text(data)
        # Perform sentiment analysis
        data = analyze_sentiment(data)
        # Add features for prediction
        data = add_features(data)
        # Train hybrid model
        rf_model, lstm_model, scaler = train_hybrid_model(data)
        # Save trained models and scaler
        save_models(rf_model, lstm_model, scaler)

if __name__ == "__main__":
    main()
```

```
Authenticated successfully!
Random Forest Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         2
           1       1.00      1.00      1.00        12

    accuracy                           1.00        14
   macro avg       1.00      1.00      1.00        14
weighted avg       1.00      1.00      1.00        14

WARNING:tensorflow:From C:\Users\Hp\anaconda3\lib\site-packages\keras\
src\layers\rnn\lstm.py:148: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

WARNING:tensorflow:From C:\Users\Hp\anaconda3\lib\site-packages\keras\
src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/10
WARNING:tensorflow:From C:\Users\Hp\anaconda3\lib\site-packages\keras\
```

```
src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is
deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Hp\anaconda3\lib\site-packages\keras\
src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

4/4 [==============================] - 2s 5ms/step - loss: 0.6891 -
accuracy: 0.8113
Epoch 2/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6809 -
accuracy: 0.8491
Epoch 3/10
4/4 [==============================] - 0s 11ms/step - loss: 0.6727 -
accuracy: 0.9811
Epoch 4/10
4/4 [==============================] - 0s 6ms/step - loss: 0.6648 -
accuracy: 1.0000
Epoch 5/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6575 -
accuracy: 1.0000
Epoch 6/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6495 -
accuracy: 1.0000
Epoch 7/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6417 -
accuracy: 1.0000
Epoch 8/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6340 -
accuracy: 1.0000
Epoch 9/10
4/4 [==============================] - 0s 5ms/step - loss: 0.6259 -
accuracy: 1.0000
Epoch 10/10
4/4 [==============================] - 0s 4ms/step - loss: 0.6181 -
accuracy: 1.0000
LSTM Evaluation:
1/1 [==============================] - 1s 507ms/step - loss: 0.5975 -
accuracy: 1.0000
Models saved successfully.
```