```
pip install google-auth google-auth-oauthlib google-auth-httplib2 googleapiclient pandas numpy nltk scikit-learn textblob
```

```
Requirement already satisfied: google-auth in /usr/local/lib/python3.11/dist-packages (2.38.0)
Requirement already satisfied: google-auth-oauthlib in /usr/local/lib/python3.11/dist-packages (1.2.1)
Requirement already satisfied: google-auth-httplib2 in /usr/local/lib/python3.11/dist-packages (0.2.0)
ERROR: Could not find a version that satisfies the requirement googleapiclient (from versions: none)
ERROR: No matching distribution found for googleapiclient
```

```
pip install google-api-python-client
```

```
Requirement already satisfied: google-api-python-client in /usr/local/lib/python3.11/dist-packages (2.164.0)
Requirement already satisfied: httplib2<1.dev0,>=0.19.0 in /usr/local/lib/python3.11/dist-packages (from google-api-python-client) (0.22
Requirement already satisfied: google-auth!=2.24.0,!=2.25.0,<3.0.0.dev0,>=1.32.0 in /usr/local/lib/python3.11/dist-packages (from google
Requirement already satisfied: google-auth-httplib2<1.0.0,>=0.2.0 in /usr/local/lib/python3.11/dist-packages (from google-api-python-cli
Requirement already satisfied: google-api-core!=2.0.*,!=2.1.*,!=2.2.*,!=2.3.0,<3.0.0.dev0,>=1.31.5 in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.11/dist-packages (from google-api-python-client) (4.1.1)
Requirement already satisfied: googleapis-common-protos<2.0.0,>=1.56.2 in /usr/local/lib/python3.11/dist-packages (from google-api-core!
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<7.0.0,>=3.19.5 in /usr/l
Requirement already satisfied: proto-plus<2.0.0,>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from google-api-core!=2.0.*,!=2.1.*
Requirement already satisfied: requests<3.0.0,>=2.18.0 in /usr/local/lib/python3.11/dist-packages (from google-api-core!=2.0.*,!=2.1.*,!
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from google-auth!=2.24.0,!=2.25.0,<3.0
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.11/dist-packages (from google-auth!=2.24.0,!=2.25.0,<3.0.
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.11/dist-packages (from google-auth!=2.24.0,!=2.25.0,<3.0.0.dev0,>
Requirement already satisfied: pyparsing!=3.0.0,!=3.0.1,!=3.0.2,!=3.0.3,<4,>=2.4.2 in /usr/local/lib/python3.11/dist-packages (from http
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.11/dist-packages (from pyasn1-modules>=0.2.1->google-auth!
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-api-core!=2
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-api-c
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests<3.0.0,>=2.18.0->google-api-c
```

```python
import os
import re
import time
import pandas as pd
import numpy as np
import googleapiclient.discovery
import googleapiclient.errors
from textblob import TextBlob
import random

# Set up YouTube API
API_KEY = "AIzaSyDNupwVZOeuuxYE9FD7VYWAUuSAX-DTQ84"
youtube = googleapiclient.discovery.build("youtube", "v3", developerKey=API_KEY)

# CSV file to store data
DATASET_FILE = "youtube_fake_news_data.csv"

# List of search queries to get diverse data
SEARCH_QUERIES = ["breaking news", "political news", "health updates", "climate change", "viral news"]

# Number of videos to fetch per query
VIDEOS_PER_QUERY = 50


### Helper Functions ###

def search_videos(query, max_results=50):
    """Fetches video IDs based on a search query."""
    request = youtube.search().list(
        q=query,
        part="id",
        type="video",
        maxResults=max_results
    )
    response = request.execute()

    return [item["id"]["videoId"] for item in response.get("items", [])]


def get_video_details(video_id):
    """Fetches video metadata (title, description, stats)."""
    request = youtube.videos().list(
        part="snippet,statistics",
        id=video_id
    )
    response = request.execute()

    if not response.get("items"):
        return None

    video = response["items"][0]
    return {
        "video_id": video_id,
        "title": video["snippet"]["title"],
        "description": video["snippet"]["description"],
        "likes": int(video["statistics"].get("likeCount", 0)),
        "dislikes": int(video["statistics"].get("dislikeCount", 1)),  # Avoid zero division
        "comment_count": int(video["statistics"].get("commentCount", 0))
    }


def get_video_comments(video_id, max_comments=50):
    """Fetches top comments from a video."""
    comments = []
    try:
        request = youtube.commentThreads().list(
            part="snippet",
            videoId=video_id,
            maxResults=max_comments
        )
        response = request.execute()

        for item in response["items"]:
            comment_text = item["snippet"]["topLevelComment"]["snippet"]["textOriginal"]
            comments.append(comment_text)

    except Exception:
        pass  # Skip videos with disabled comments

    return comments


def check_lack_of_citations(description):
    """Detects if the description contains credible sources (links)."""
    return 0 if re.search(r'https?://\S+', description) else 1  # 0 = Real, 1 = Fake


def check_spam_comments(comments):
    """Calculates spam comment ratio based on common spam keywords."""
    spam_keywords = ["click here", "subscribe", "buy now", "discount", "visit", "win free"]
    spam_count = sum(1 for comment in comments if any(word in comment.lower() for word in spam_keywords))
    return spam_count / max(len(comments), 1)


def analyze_comment_sentiment(comments):
    """Performs sentiment analysis on comments."""
    total_sentiment = sum(TextBlob(comment).sentiment.polarity for comment in comments)
    return total_sentiment / max(len(comments), 1)


def like_dislike_ratio(likes, dislikes):
    """Computes the like-to-dislike ratio."""
    return likes / dislikes if dislikes > 0 else likes


### Data Collection Pipeline ###
def scrape_videos_and_save():
    """Scrapes videos and appends them to the dataset."""
    all_data = []

    for query in SEARCH_QUERIES:
        print(f"🔍 Searching for videos on: {query}")
        video_ids = search_videos(query, VIDEOS_PER_QUERY)

        for video_id in video_ids:
            print(f"⏳ Processing video: {video_id}...")

            video_details = get_video_details(video_id)
            if not video_details:
                continue  # Skip if no details
            comments = get_video_comments(video_id)

            row = {
                "video_id": video_id,
                "lack_of_citations": check_lack_of_citations(video_details["description"]),
                "spam_comment_ratio": check_spam_comments(comments),
                "comment_sentiment": analyze_comment_sentiment(comments),
                "like_dislike_ratio": like_dislike_ratio(video_details["likes"], video_details["dislikes"]),
                "label": random.choice([0, 1])  # Temporary random labeling
            }

            all_data.append(row)

    df = pd.DataFrame(all_data)

    if os.path.exists(DATASET_FILE):
        df.to_csv(DATASET_FILE, mode='a', header=False, index=False)
    else:
        df.to_csv(DATASET_FILE, index=False)

    print("\n✅ Data saved successfully!\n")


### Run Data Collection ###
scrape_videos_and_save()
```

```
🔍 Searching for videos on: breaking news
📹 Processing video: ELXL9i9-FXg...
📹 Processing video: sr121jPdv18...
📹 Processing video: az2q1T6pwc0...
📹 Processing video: jr2QLcZbnE8...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: 7DoW68U2ZNk...
📹 Processing video: VIuP2SzVINA...
📹 Processing video: Te_aJ0ASj54...
📹 Processing video: 0lURdVBCBo8...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: bCoI4JsD0rs...
📹 Processing video: d8y3anhieho...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: jzfxuYj7cEQ...
📹 Processing video: IJEUUcfsXhQ...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: MVCu8dvZ9Gg...
📹 Processing video: ItJpsqA2xJY...
📹 Processing video: YB6hNo2vKOE...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: eFEOwiWdRAY...
📹 Processing video: D6XlQZ8StEc...
📹 Processing video: miFpe38knH4...
📹 Processing video: Ydj353Y8vwU...
📹 Processing video: q5UtCf61Yes...
📹 Processing video: l4ayas-7ql4...
📹 Processing video: XkjIbaxMHJk...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: n31tSjK1yok...
📹 Processing video: Wjp8LwSSAY8...
📹 Processing video: 6X3oFn9wUmY...
📹 Processing video: IeAmyqttkS0...
📹 Processing video: k3BSEWkOtr8...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
📹 Processing video: 37zUjt2ASqw...
📹 Processing video: 0PqxeNQeQL8...
📹 Processing video: EtpxCWRKcAs...
📹 Processing video: sW8y1t0sKzk...
📹 Processing video: dVeyqbRAW1o...
📹 Processing video: ZlKMjpB9aj4...
📹 Processing video: 3J-zSPGcn50...
📹 Processing video: st5KJ-iy-fI...
📹 Processing video: UO3Vk3jsAe4...
📹 Processing video: 1R3S5XkmocY...
📹 Processing video: u5MPVX_jn2E...
📹 Processing video: 2mUmjDHi460...
📹 Processing video: RT2S3Y0x4Ig...
📹 Processing video: 6ReKSiyhNs8...
📹 Processing video: 2RaS0B2Xv1g...
📹 Processing video: 5cwD0oeYb9w...
```

```
Processing video: dduefDMgoUs...
Processing video: -rkKtSDtQMY...
WARNING:googleapiclient.http:Encountered 403 Forbidden with reason "commentsDisabled"
Processing video: WwLin2JwqrI...
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
df = pd.read_csv("youtube_fake_news_data_updated.csv")

# Select features and target variable
X = df[["lack_of_citations", "spam_comment_ratio", "comment_sentiment", "like_dislike_ratio"]]
y = df["label"]

# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Normalize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the model using Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)

# Evaluate the model
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Model Accuracy: 95.95%
Classification Report:
               precision    recall  f1-score   support

           0       0.89      1.00      0.94        25
           1       1.00      0.94      0.97        49

    accuracy                           0.96        74
   macro avg       0.95      0.97      0.96        74
weighted avg       0.96      0.96      0.96        74
```

```
import googleapiclient.discovery
import re
import numpy as np
from textblob import TextBlob

# YouTube API Setup
API_KEY = "AIzaSyDNupwVZOeuuxYE9FD7VYWAUuSAX-DTQ84"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

# Initialize YouTube API client
youtube = googleapiclient.discovery.build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION, developerKey=API_KEY)

# Function to check citation presence in description
def check_citations(description):
    urls = re.findall(r'(https?://\S+)', description)
    return 0 if urls else 1  # 0 = Has citations, 1 = No citations (potential fake news)

# Function to get video details
def get_video_data(video_id):
    request = youtube.videos().list(
        part="snippet,statistics",
        id=video_id
    )
    response = request.execute()

    if "items" not in response or len(response["items"]) == 0:
        return None  # Video not found
        snippet = response["items"][0]["snippet"]
        stats = response["items"][0]["statistics"]

        description = snippet.get("description", "")
        likes = int(stats.get("likeCount", 0))
        dislikes = int(stats.get("dislikeCount", 1))  # Avoid division by zero

        # Compute features
        lack_of_citations = check_citations(description)
        like_dislike_ratio = round(likes / dislikes, 2)

        return lack_of_citations, like_dislike_ratio

    # Function to get comments and analyze sentiment & spam ratio
    def get_comment_data(video_id):
        request = youtube.commentThreads().list(
            part="snippet",
            videoId=video_id,
            maxResults=50
        )
        response = request.execute()

        comments = []
        spam_count = 0

        for item in response.get("items", []):
            comment = item["snippet"]["topLevelComment"]["snippet"]["textDisplay"]
            comments.append(comment)

            # Check for spam (e.g., repetitive phrases, links)
            if re.search(r'(https?://\S+|subscribe|click here|free|giveaway)', comment, re.IGNORECASE):
                spam_count += 1

        # Compute sentiment score
        sentiments = [TextBlob(c).sentiment.polarity for c in comments]
        comment_sentiment = np.mean(sentiments) if sentiments else 0

        # Compute spam comment ratio
        spam_comment_ratio = round(spam_count / max(len(comments), 1), 2)

        return spam_comment_ratio, comment_sentiment

    # Function to predict fake or real news
    def predict_video_authenticity(video_id, model, scaler):
        video_data = get_video_data(video_id)
        comment_data = get_comment_data(video_id)

        if not video_data or not comment_data:
            return "Error: Unable to fetch video data."

        lack_of_citations, like_dislike_ratio = video_data
        spam_comment_ratio, comment_sentiment = comment_data

        # Prepare feature array
        features = np.array([[lack_of_citations, spam_comment_ratio, comment_sentiment, like_dislike_ratio]])
        features_scaled = scaler.transform(features)

        # Predict
        prediction = model.predict(features_scaled)
        return "Fake News" if prediction[0] == 0 else "Real News"

    # Example Usage:
    video_id = "BN-C5N60u_M"
    result = predict_video_authenticity(video_id, model, scaler)
    print(f"Prediction for Video {video_id}: {result}")
```

```
Prediction for Video BN-C5N60u_M: Real News
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Standard
  warnings.warn(
```

Start coding or generate with AI.