

Web and Social Media Analytics Lab

1. Preprocessing text document using NLTK of Python

a. Stopword elimination

b. Stemming

c. Lemmatization

d. POS tagging

e. Lexical analysis

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag

# Ensure required NLTK data is downloaded
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('wordnet')

# (a) Stopword Elimination
def stopwords_elimination(text):
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in word_tokenize(text) if word.lower() not in stop_words]
    return filtered_words
text = "This is a sample text with stopwords."
print("Stopword Elimination:", stopwords_elimination(text))

# (b) Stemming
def stemming(text):
    ps = PorterStemmer()
    stemmed_words = [ps.stem(word) for word in word_tokenize(text)]
    return stemmed_words
text = "Running runner runs beautifully."
print("Stemming:", stemming(text))

# (c) Lemmatization
def lemmatization(text):
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = [lemmatizer.lemmatize(word) for word in word_tokenize(text)]
    return lemmatized_words
text = "Running runner runs beautifully."
print("Lemmatization:", lemmatization(text))

# (d) POS Tagging
def pos_tagging(text):
    words = word_tokenize(text)
    tagged_words = pos_tag(words)
    return tagged_words
text = "This is a sample text with POS tagging."
print("POS Tagging:", pos_tagging(text))

# (e) Lexical Analysis
def lexical_analysis(text):
    tokens = word_tokenize(text)
    tagged_tokens = pos_tag(tokens)
    return tagged_tokens
```

```
text = "This is a sample text with lexical analysis."  
print("Lexical Analysis:", lexical_analysis(text))
```

Output:

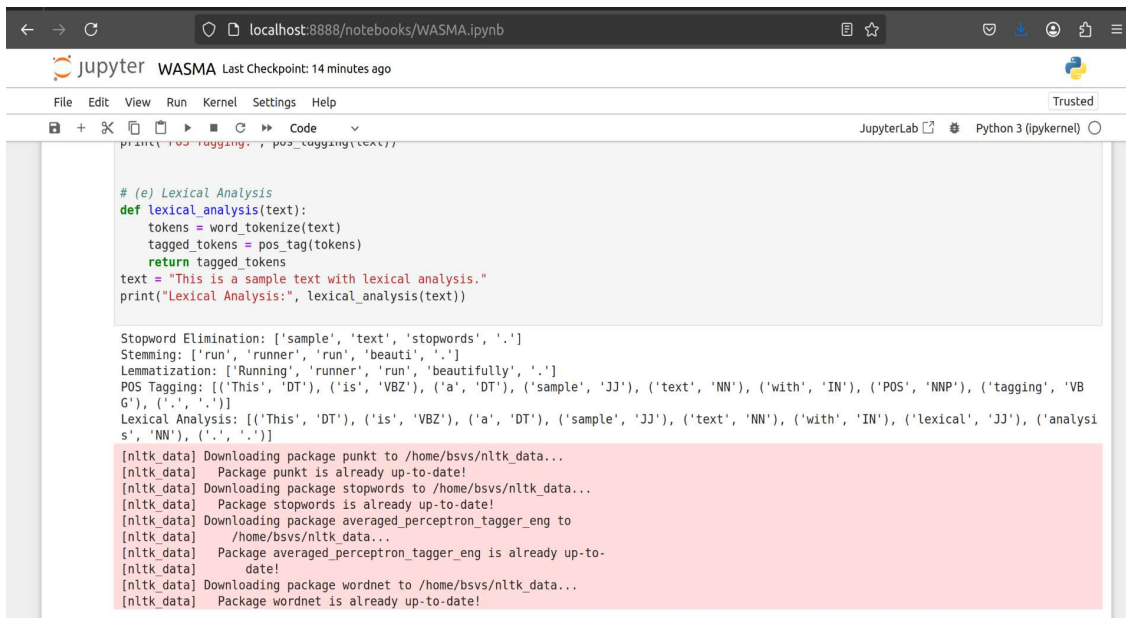
Stopword Elimination: ['sample', 'text', 'stopwords', '.']

Stemming: ['run', 'runner', 'run', 'beauti', '.']

Lemmatization: ['Running', 'runner', 'run', 'beautifully', '.']

POS Tagging: [('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'), ('with', 'IN'), ('POS', 'NNP'), ('tagging', 'VBG'), ('.', '.')]]

Lexical Analysis: [('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'), ('with', 'IN'), ('lexical', 'JJ'), ('analysis', 'NN'), ('.', '.')]]



The screenshot shows a JupyterLab window with a file named 'WASMA' and a last checkpoint 14 minutes ago. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area displays a code cell with the following Python code:

```
# (e) Lexical Analysis  
def lexical_analysis(text):  
    tokens = word_tokenize(text)  
    tagged_tokens = pos_tag(tokens)  
    return tagged_tokens  
text = "This is a sample text with lexical analysis."  
print("Lexical Analysis:", lexical_analysis(text))
```

Below the code, the output is displayed, showing the results of the lexical analysis steps:

```
Stopword Elimination: ['sample', 'text', 'stopwords', '.']  
Stemming: ['run', 'runner', 'run', 'beauti', '.']  
Lemmatization: ['Running', 'runner', 'run', 'beautifully', '.']  
POS Tagging: [('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'), ('with', 'IN'), ('POS', 'NNP'), ('tagging', 'VBG'), ('.', '.')] ]  
Lexical Analysis: [('This', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('sample', 'JJ'), ('text', 'NN'), ('with', 'IN'), ('lexical', 'JJ'), ('analysis', 'NN'), ('.', '.')] ]
```

At the bottom, there are several system messages from NLTK data:

```
[nltk data] Downloading package punkt to /home/bsvs/nltk_data...  
[nltk data] Package punkt is already up-to-date!  
[nltk data] Downloading package stopwords to /home/bsvs/nltk_data...  
[nltk data] Package stopwords is already up-to-date!  
[nltk data] Downloading package averaged_perceptron_tagger_eng to /home/bsvs/nltk_data...  
[nltk data] Package averaged_perceptron_tagger_eng is already up-to-date!  
[nltk data] Downloading package wordnet to /home/bsvs/nltk_data...  
[nltk data] Package wordnet is already up-to-date!
```

2. Sentiment analysis on customer review on products

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

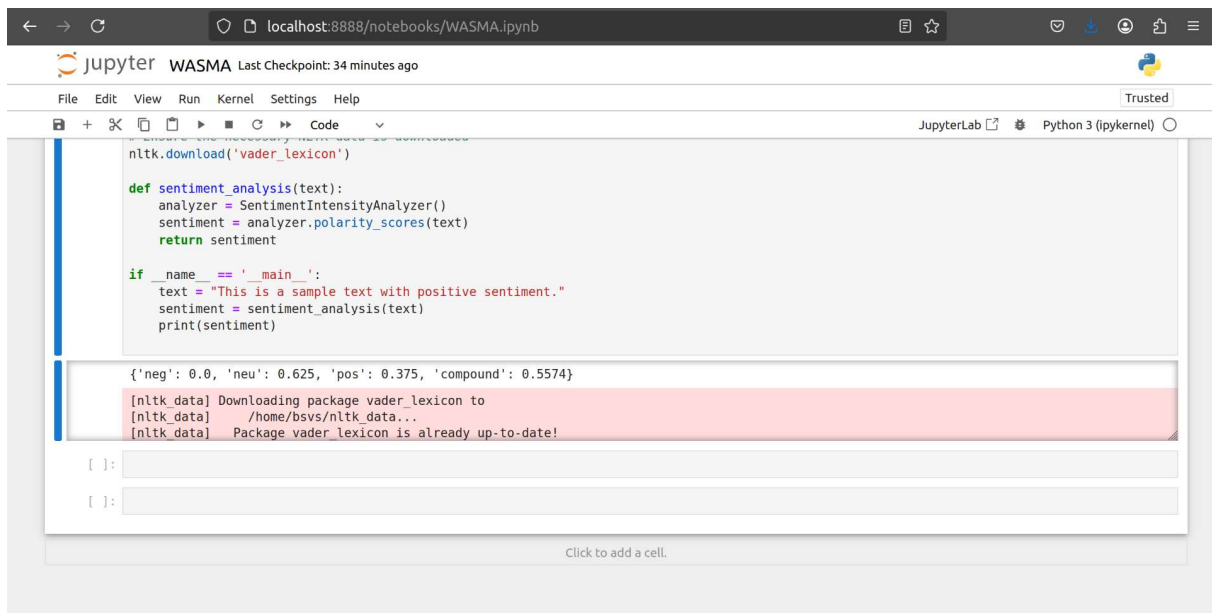
# Ensure the necessary NLTK data is downloaded
nltk.download('vader_lexicon')

def sentiment_analysis(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)
    return sentiment

if __name__ == '__main__':
    text = "This is a sample text with positive sentiment."
    sentiment = sentiment_analysis(text)
    print(sentiment)
```

Output:

```
{'neg': 0.0, 'neu': 0.625, 'pos': 0.375, 'compound': 0.5574}
```



The screenshot shows a JupyterLab interface with a notebook named 'WASMA'. The code in the notebook is as follows:

```
nltk.download('vader_lexicon')

def sentiment_analysis(text):
    analyzer = SentimentIntensityAnalyzer()
    sentiment = analyzer.polarity_scores(text)
    return sentiment

if __name__ == '__main__':
    text = "This is a sample text with positive sentiment."
    sentiment = sentiment_analysis(text)
    print(sentiment)
```

The output of the code is displayed in a red-shaded box:

```
{'neg': 0.0, 'neu': 0.625, 'pos': 0.375, 'compound': 0.5574}
```

Below the output, there are three lines of text indicating the status of the NLTK data download:

```
[nltk data] Downloading package vader_lexicon to
[nltk data] /home/bsvs/nltk_data...
[nltk data] Package vader_lexicon is already up-to-date!
```

The JupyterLab interface also shows the file explorer on the left, the command palette, and the kernel status (Python 3 (ipykernel)).

3. Web analytics

a. Web usage data (web server log data, clickstream analysis)

b. Hyperlink data

```
import requests
from bs4 import BeautifulSoup
from collections import Counter
from urllib.parse import urlparse, urljoin

def get_links(url):
    try:
        # Send a request to the webpage
        response = requests.get(url, timeout=10)
        response.raise_for_status() # Raise an error for bad responses (4xx, 5xx)

        # Parse the HTML content
        soup = BeautifulSoup(response.text, 'html.parser')

        # Extract all hyperlinks
        links = [a['href'] for a in soup.find_all('a', href=True)]

        return links
    except requests.exceptions.RequestException as e:
        print(f"Error fetching the page: {e}")
        return []

def analyze_links(url, links):
    domain = urlparse(url).netloc
    internal_links = []
    external_links = []

    for link in links:
        full_url = urljoin(url, link) # Convert relative URLs to absolute
        parsed_url = urlparse(full_url)

        if parsed_url.netloc == domain:
            internal_links.append(full_url)
        else:
            external_links.append(full_url)

    return internal_links, external_links

if __name__ == "__main__":
    url = input("Enter a website URL (including http/https): ")

    # Get all links from the page
    all_links = get_links(url)

    if not all_links:
        print("No links found on the page.")
    else:
        # Analyze internal and external links
        internal_links, external_links = analyze_links(url, all_links)

        # Count the most popular links
        link_counts = Counter(all_links)
```

The screenshot displays a JupyterLab environment with a file named 'localhost:8888/notesbooks/WASMA.ipynb' open. The interface includes a top bar with navigation icons and a 'Trusted' status indicator. The main area shows a code editor with a Python script and its output.

Code:

```
for link, count in link_counts.most_common(5):
    print(f'{link}: {count} times')

print("\nExternal Links:")
for link in external_links[:5]: # Show up to 5 external links
    print(link)
```

Output:

```
Enter a website URL (including http/https): http://facebook.com

=== Web Analytics Report ===
Total Links Found: 42
Internal Links: 18
External Links: 24

Most Popular Links (Top 5):
#: 2 times
https://www.facebook.com/recover/initiate/?privacy_mutation_token=eyJ0eXB1IjowLCJjcmVhdGhVbG90aWw1IjoxNzQzMDCyNzEzLCJjYWxsc2l0ZV9pZCI6MzgxCmJi
5MDc5NTc1OTQ2f0%3D&ars=facebook_login&next: 1 times
/r.php?entry_point=login: 1 times
/pages/create/?ref_type=registration_form: 1 times
https://www.facebook.com/: 1 times

External Links:
https://www.facebook.com/recover/initiate/?privacy_mutation_token=eyJ0eXB1IjowLCJjcmVhdGhVbG90aWw1IjoxNzQzMDCyNzEzLCJjYWxsc2l0ZV9pZCI6MzgxCmJi
5MDc5NTc1OTQ2f0%3D&ars=facebook_login&next
https://www.facebook.com/
https://hl-in.facebook.com/
https://ur-pk.facebook.com/
https://ta-in.facebook.com/
```

4. Search engine optimization- implement spamdexing

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download required NLTK data
nltk.download('stopwords')
nltk.download('punkt')

def spamdexing(text):
    stop_words = set(stopwords.words('english'))
    keywords = ['Dear', 'You', 'Won', 'Lucky Lottery'] # Targeted spam keywords

    # Tokenize the text
    words = word_tokenize(text)

    # Remove stopwords
    filtered_text = [word for word in words if word.lower() not in stop_words]

    # Perform keyword stuffing by repeating each keyword 10 times
    for keyword in keywords:
        filtered_text.extend([keyword] * 10)

    return ' '.join(filtered_text) # Convert list back to text

if __name__ == '__main__':
    text = "This is a sample text with stopwords."
    spammed_text = spamdexing(text)
    print("Spamdexed Text:\n", spammed_text)
```

Output:

Spamdexed Text:

Sample text stopwords . Dear Dear Dear Dear Dear Dear Dear Dear Dear You You You You You You
You You You You Won Won Won Won Won Won Won Won Won Won Lucky Lottery Lucky Lottery Lucky
Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky
Lottery

The screenshot shows a JupyterLab window titled "WASMA Last Checkpoint: 1 hour ago". The interface includes a top bar with navigation icons and a menu bar with options like File, Edit, View, Run, Kernel, Settings, and Help. Below the menu bar is a toolbar with various file and code execution icons. The main area displays a Python script that defines keywords, tokenizes text, removes stopwords, performs keyword stuffing by repeating each keyword 10 times, and prints the resulting spammed text. The output shows the original text with the keywords repeated multiple times.

```
keywords = ['Dear', 'You', 'Won', 'Lucky Lottery'] # Targeted spam keywords

# Tokenize the text
words = word_tokenize(text)

# Remove stopwords
filtered_text = [word for word in words if word.lower() not in stop_words]

# Perform keyword stuffing by repeating each keyword 10 times
for keyword in keywords:
    filtered_text.extend([keyword] * 10)

return '.join(filtered_text) # Convert list back to text

if __name__ == '__main__':
    text = "This is a sample text with stopwords."
    spammed_text = spamdexing(text)
    print("Spamdexed Text:\n", spammed_text)
```

Spamdexed Text:
sample text stopwords . Dear Dear Dear Dear Dear Dear Dear Dear Dear You You You You You You You You Won Won Won Won Won Won
n Won Won Won Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery Lucky Lottery L
ucky Lottery

```
[nlTK data] Downloading package stopwords to /home/bvs/nltk.data...  
[nlTK data] Package stopwords is already up-to-date!  
[nlTK data] Downloading package punkt to /home/bvs/nltk.data...  
[nlTK data] Package punkt is already up-to-date!
```

5. Use Google analytics tools to implement the following

a. Conversion Statistics

b. Visitor Profiles

Install python required libraries

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

```
from googleapiclient.discovery import build
import pandas as pd
```

```
# Your API key and channel ID
```

```
API_KEY = 'AIzaSyCisHBOLsG4o4u5k01qhe5TNxOAF5l7Dh0'
```

```
CHANNEL_ID = 'UCO_6bvpeed4bcyA3lLv_RLQ'
```

```
def get_youtube_service():
```

```
    """Builds the YouTube Data API client."""
```

```
    return build('youtube', 'v3', developerKey=API_KEY)
```

```
def get_channel_statistics(service, channel_id):
```

```
    """Fetches channel statistics."""
```

```
    request = service.channels().list(
```

```
        part='snippet,statistics',
```

```
        id=channel_id
```

```
    )
```

```
    response = request.execute()
```

```
    return response
```

```
def print_channel_stats(response):
```

```
    """Prints and formats the channel statistics."""
```

```
    stats = response['items'][0]['statistics']
```

```
    snippet = response['items'][0]['snippet']
```

```
    data = {
```

```
        'Channel Title': snippet['title'],
```

```
        'Subscribers': stats.get('subscriberCount', 'N/A'),
```

```
        'Total Views': stats.get('viewCount', 'N/A'),
```

```
        'Total Videos': stats.get('videoCount', 'N/A'),
```

```
        'Description': snippet['description'],
```

```
    }
```

```
    df = pd.DataFrame([data])
```

```
    print("\nYouTube Channel Statistics:")
```

```
    print(df)
```

```
# Get the YouTube service
```

```
youtube_service = get_youtube_service()
```

```
# Fetch and print channel statistics
```

```
response = get_channel_statistics(youtube_service, CHANNEL_ID)
```

```
print_channel_stats(response)
```

Output:

YouTube Channel Statistics:

	Channel Title	Subscribers	Total Views	Total Videos	Description
--	---------------	-------------	-------------	--------------	-------------

0	Crazy Kids	88	19742	250	
---	------------	----	-------	-----	--

← → ↻ localhost:8889/notebooks/WASMA.ipynb ☆

jupyter WASMA Last Checkpoint: 3 days ago Trusted

File Edit View Run Kernel Settings Help

JupyterLab Python 3 (ipykernel)

```
def print_channel_stats(response):
    """Prints and formats the channel statistics."""
    stats = response['items'][0]['statistics']
    snippet = response['items'][0]['snippet']
    data = {
        'Channel Title': snippet['title'],
        'Subscribers': stats.get('subscriberCount', 'N/A'),
        'Total Views': stats.get('viewCount', 'N/A'),
        'Total Videos': stats.get('videoCount', 'N/A'),
        'Description': snippet['description'],
    }
    df = pd.DataFrame([data])
    print("\nYouTube Channel Statistics:")
    print(df)

# Get the YouTube service
youtube_service = get_youtube_service()

# Fetch and print channel statistics
response = get_channel_statistics(youtube_service, CHANNEL_ID)
print_channel_stats(response)
```

YouTube Channel Statistics:

	Channel Title	Subscribers	Total Views	Total Videos	Description
0	Crazy Kids	88	19742	250	

6. Use Google analytics tools to implement the Traffic Sources.

Install python required libraries

```
!pip install google-auth google-auth-oauthlib google-auth-httplib2 google-api-python-client pandas
```

```
from google.oauth2.credentials import Credentials
```

```
from google_auth_oauthlib.flow import InstalledAppFlow
```

```
from googleapiclient.discovery import build
```

```
import pandas as pd
```

```
# Path to your OAuth 2.0 client secret file
```

```
CLIENT_SECRET_FILE = 'client_secret_585961444636-  
fl5k3fheu8mi3m0nna76r23ulc543lqa.apps.googleusercontent.com.json'
```

```
SCOPES = ['https://www.googleapis.com/auth/yt-analytics.readonly']
```

```
def get_authenticated_service():
```

```
    """Authenticate and return the YouTube Analytics API service."""
```

```
    flow = InstalledAppFlow.from_client_secrets_file(CLIENT_SECRET_FILE, SCOPES)
```

```
    credentials = flow.run_local_server(port=0)
```

```
    return build('youtubeAnalytics', 'v2', credentials=credentials)
```

```
def get_traffic_sources(service, channel_id):
```

```
    """Fetches traffic source statistics."""
```

```
    request = service.reports().query(
```

```
        ids='channel=={}'.format(channel_id),
```

```
        startDate='2024-01-01',
```

```
        endDate='2024-12-31',
```

```
        metrics='views',
```

```
        dimensions='insightTrafficSourceType',
```

```
        sort='-views'
```

```
    )
```

```
    response = request.execute()
```

```

return response

def print_traffic_sources(response):

    """Prints traffic source statistics."""

    rows = response.get('rows', [])

    columns = response.get('columnHeaders', [])

    data = []

    for row in rows:

        data.append(dict(zip([col['name'] for col in columns], row)))

    df = pd.DataFrame(data)

    print("\nYouTube Traffic Source Statistics:")

    print(df)

# Get the authenticated YouTube Analytics service

youtube_analytics_service = get_authenticated_service()

# Replace with your channel ID

CHANNEL_ID = 'UCO_6bvpeed4bcyA3iLv_RLQ'

# Fetch and print traffic source statistics

response = get_traffic_sources(youtube_analytics_service, CHANNEL_ID)

print_traffic_sources(response)

```

Output:

Please visit this URL to authorize this application:

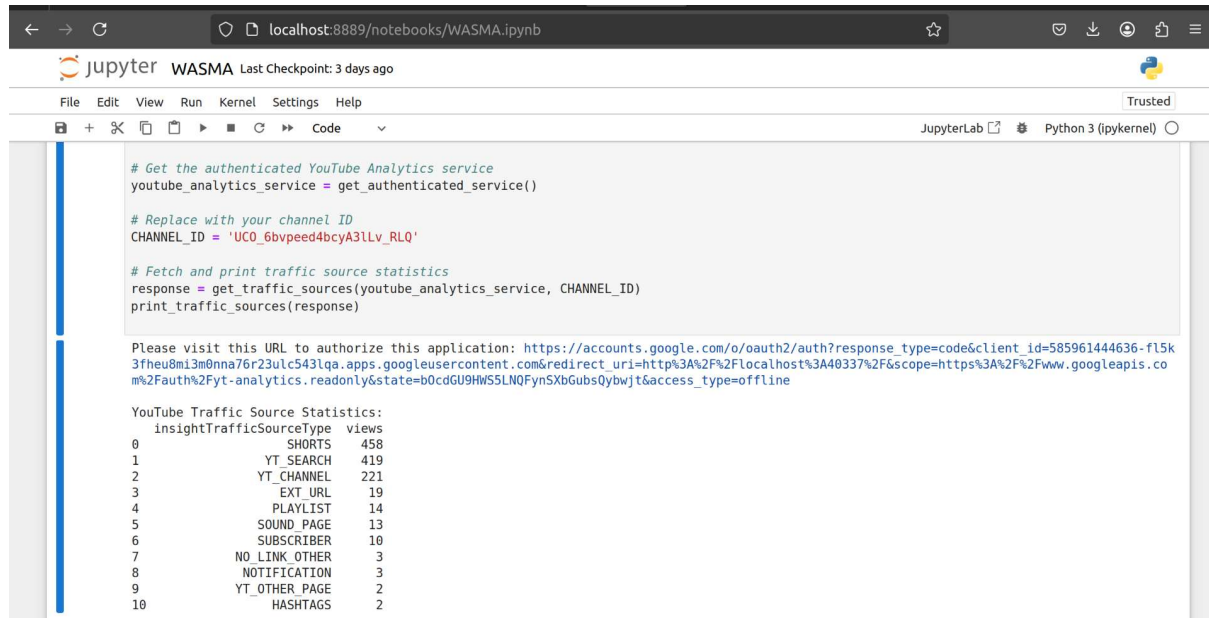
https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=585961444636-fl5k3fheu8mi3m0nna76r23ulc543lqa.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A40337%2F&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fyt-analytics.readonly&state=bOcdGU9HWS5LNQFynSXbGubsQybwjt&access_type=offline

```

YouTube Traffic Source Statistics:
insightTrafficSourceType  views
0          SHORTS      458
1        YT_SEARCH     419
2    YT_CHANNEL      221
3        EXT_URL       19
4        PLAYLIST      14
5    SOUND_PAGE       13
6    SUBSCRIBER       10
7    NO_LINK_OTHER      3

```

8 NOTIFICATION 3
9 YT_OTHER_PAGE 2
10 HASHTAGS 2



The screenshot shows a JupyterLab window with a browser address bar at `localhost:8889/notebooks/WASMA.ipynb`. The interface includes a top menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and execution. The main area displays a Python script with the following code:

```
# Get the authenticated YouTube Analytics service
youtube_analytics_service = get_authenticated_service()

# Replace with your channel ID
CHANNEL_ID = 'UC0_6bypced4bcyA3lLv_RLQ'

# Fetch and print traffic source statistics
response = get_traffic_sources(youtube_analytics_service, CHANNEL_ID)
print_traffic_sources(response)
```

Below the code, an authorization URL is displayed: `https://accounts.google.com/o/oauth2/auth?response_type=code&client_id=585961444636-fl5k3fheu8mi3m0nna76r23ulc543lqa.apps.googleusercontent.com&redirect_uri=http%3A%2F%2Flocalhost%3A40337%2F&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fyt-analytics.readonly&state=b0cdGU9HWS5LNQFynSXbGubs0ybwjt&access_type=offline`. The output of the script is a table of YouTube Traffic Source Statistics:

	insightTrafficSourceType	views
0	SHORTS	458
1	YT_SEARCH	419
2	YT_CHANNEL	221
3	EXT_URL	19
4	PLAYLIST	14
5	SOUND_PAGE	13
6	SUBSCRIBER	10
7	NO_LINK_OTHER	3
8	NOTIFICATION	3
9	YT_OTHER_PAGE	2
10	HASHTAGS	2