VarsityPro – Data Analyst Internship Assessment

Aim:

To create a Python script that automates web scraping using Selenium, performs data cleaning and manipulation, and establishes a basic data pipeline.

Tools Used:

1. Jupyter Notebook
2. Command Prompt (Bash)

Libraries Used:

1. **Selenium**: Selenium is a web testing library used for automating browser interactions, allowing for dynamic web page interaction and testing.
2. **bs4**: Beautiful Soup is a Python library for pulling data out of HTML and XML files, providing tools to navigate and search the parse tree.
3. **Pandas**: Pandas is a powerful data manipulation and analysis library for Python, providing data structures like DataFrame for efficiently handling and analyzing structured data.

Code and Explanation:

```
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd

# Function to scrape the page using Selenium
def scrape_page(url):
    driver = webdriver.Chrome()
    driver.get(url)

    # Wait for the page to load (you might need to adjust the
time)
    driver.implicitly_wait(10)

    # Get the page source
    page_source = driver.page_source

    driver.quit()

    return page_source

# Function to extract paragraphs from HTML
def extract_paragraphs(html):
    soup = BeautifulSoup(html, 'html.parser')
```

```
    paragraphs = [p.get_text() for p in soup.find_all('p')]
    return paragraphs

# Scraping the Wikipedia page
url = "https://en.wikipedia.org/wiki/Nikola_Tesla"
page_source = scrape_page(url)

# Extracting paragraphs
paragraphs = extract_paragraphs(page_source)
```

This Python code scrapes the material from the Nikola Tesla Wikipedia page using the Selenium and BeautifulSoup libraries. The scrape_page method uses Selenium to automate the process of obtaining the page source, enabling JavaScript content to be rendered dynamically. The method extract_paragraphs receives the HTML source it has gotten, uses BeautifulSoup to parse the HTML, and extracts the text content from each paragraph (<p>) element on the page. The last paragraphs, which include the text for additional processing or analysis, are saved in the paragraphs variable.

```
#DATA CLEANING (DUPLICATES,NA VALUES)
# Create a Pandas DataFrame
df = pd.DataFrame({"Text": paragraphs})

# Basic data cleaning
df['Text'] = df['Text'].str.strip()

# Handling missing values
df.dropna(inplace=True)

# Removing duplicates
df.drop_duplicates(inplace=True)

# Saving cleaned data to a CSV file
df.to_csv('cleaned_data.csv', index=False)
```

This Python code demonstrates a basic data cleaning process using Pandas for a DataFrame named df, which contains text data from paragraphs. The str.strip() method is applied to remove leading and trailing whitespaces from the 'Text' column, addressing potential inconsistencies. To handle missing values, the dropna() method is used to remove any rows with null values, ensuring data completeness. Duplicates are then removed using the drop_duplicates() method, eliminating redundant entries. Finally, the cleaned DataFrame is saved to a CSV file named 'cleaned_data.csv'. This code ensures the text data is properly formatted, free of missing values and duplicates, ready for further analysis or use.

```
# Saving plain text to a text file
with open('plain_text.txt', 'w', encoding='utf-8') as file:
    file.write('\n'.join(paragraphs))

# Saving HTML text to a text file
with open('html_text.html', 'w', encoding='utf-8') as file:
    file.write(page_source)
```

The first file, 'plain_text.txt', stores the plain text content of the webpage by joining the paragraphs with newline characters. The second file, 'html_text.html', captures the raw HTML source code of the webpage, preserving its structure.

```
# Extracting hyperlinks
soup = BeautifulSoup(page_source, 'html.parser')
links = soup.find_all('a', href=True)

# Ensure paragraphs and links have the same length
min_length = min(len(paragraphs), len(links))
paragraphs = paragraphs[:min_length]
links = links[:min_length]

# Create a DataFrame with 'Text' and 'Link' columns
links_df = pd.DataFrame({
    'Text': paragraphs,
    'Link': [link['href'] for link in links]
})
links_df.to_csv('links_data.csv', index=False)
```

This Python code snippet uses BeautifulSoup to parse the HTML source code of a webpage (page_source) and extract all hyperlinks (<a> tags with 'href' attribute) from the page. It then ensures that the number of paragraphs (paragraphs) and the number of extracted links are the same by truncating the longer of the two lists. Finally, it creates a Pandas DataFrame named links_df with two columns: 'Text', containing the paragraph text, and 'Link', containing the corresponding hyperlinks. This DataFrame is designed to store and organize the extracted information for further analysis or presentation, aligning each paragraph with its associated hyperlink. The dataframe is then saved into a csv file called ‘links_data.csv’.

Data Pipeline:

```
# scrape_data.py

import requests
from bs4 import BeautifulSoup
```

```
# Function to scrape data from a website
def scrape_data(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract paragraphs and page_source based on your
scraping logic
    paragraphs = [paragraph.text for paragraph in
soup.find_all('p')]
    page_source = str(soup)

    return paragraphs, page_source

if __name__ == "__main__":
    # URL of the website to scrape
    target_url = "https://en.wikipedia.org/wiki/Nikola_Tesla"

    # Scraping data
    scraped_paragraphs, scraped_page_source =
scrape_data(target_url)

    # Save the scraped data to files
    with open('scraped_text.txt', 'w', encoding='utf-8') as
file:
        file.write('\n'.join(scraped_paragraphs))

    with open('scraped_html.html', 'w', encoding='utf-8') as
file:
        file.write(scraped_page_source)

import re
import pandas as pd
from bs4 import BeautifulSoup

def clean_data(text):
    # Example cleaning: Removing non-alphanumeric characters
    cleaned_text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    return cleaned_text

def manipulate_data(cleaned_text, html_text):
    soup = BeautifulSoup(html_text, 'html.parser')
    links = [(link.text.strip(), link['href']) for link in
soup.find_all('a', href=True)]

    # Create a DataFrame
    df = pd.DataFrame(links, columns=['Text', 'Link'])

    return df

if __name__ == "__main__":
```

```
    # Read scraped data from files
    with open('scraped_text.txt', 'r', encoding='utf-8') as
file:
        scraped_text = file.read()

    with open('scraped_html.html', 'r', encoding='utf-8') as
file:
        scraped_html = file.read()

    # Step 1: Clean data
    cleaned_text = clean_data(scraped_text)

    # Save cleaned text to a file
    with open('cleaned_text.txt', 'w', encoding='utf-8') as
file:
        file.write(cleaned_text)

    # Step 2: Manipulate data and create DataFrame
    df = manipulate_data(cleaned_text, scraped_html)

    # Save DataFrame to a CSV file
    df.to_csv('hyperlinks_data.csv', index=False)

    print("Data cleaning and manipulation completed.")

Bash command for running the script:

python clean_and_manipulate_data.py
```

This data pipeline is a two-step process for web scraping and cleaning/manipulating data. The first script, scrape_data.py, uses the requests library and BeautifulSoup to scrape text content and HTML source code from the Wikipedia page on Nikola Tesla. The extracted data is then saved into two separate text files, 'scraped_text.txt' and 'scraped_html.html'. The second script, clean_and_manipulate_data.py, reads the scraped data from these files, cleans the text by removing non-alphanumeric characters, and then manipulates it by extracting hyperlinks. The resulting information is stored in a Pandas DataFrame with two columns, 'Text' and 'Link'. Finally, this DataFrame is saved to a CSV file named 'hyperlinks_data.csv'. The pipeline ensures that the scraped data is cleaned and organized for further analysis or presentation. To execute the pipeline, run the command "python clean_and_manipulate_data.py" in a bash terminal.