Evoastra Ventures Internship Mini Project on Web Scraping

Prepared by-

Web Scraping:

1. Divay Galani
2. Suman Das
3. Mangaiahgari Sri Nikitha

Data Cleaning:

1. Suman Das

Data Visualization:

1. Mridula
2. Mangaiahgari Sri Nikitha

Presentation Making:

1. Suman Das
2. Divay Galani

Project Report Making:

1. Divay Galani

Project date: 09/09/24

Evoastra Ventures: Internship Program

This is to certify that the project report titled "Web Scraping" is the bonafide work of Suman Das, Mangaiahgari Sri Nikitha, Pavan Kumar, Oshmi Pandey, Vansh Dhall, Grisa Periasamy, Shriyash Prabhakaran, Azna Banu, Mridula, Afiya Sayyed and Fathima Shinoriya. This project was conducted as part of their internship at Evoastra Ventures and was completed under my supervision.

Throughout the course of this project, the team has demonstrated exemplary dedication and a profound understanding of the essential aspects of web scraping, a critical technique in data science for extracting and analyzing large volumes of data from the web. Their work showcases a deep commitment to the application of data science principles, employing advanced methodologies and tools to gather, clean, and analyze data, ultimately contributing valuable insights. The team's approach to the project has been both systematic and innovative. They have successfully navigated the complexities of handling unstructured data, transforming it into a structured format suitable for analysis. Their project not only reflects their technical proficiency but also their ability to work collaboratively and solve problems efficiently, key traits necessary for success in the field of data science.

I am confident that the skills and knowledge they have gained during this project will serve as a strong foundation for their future endeavors in data science and related fields. It has been a pleasure to oversee their progress, and I commend them for their hard work and achievements.

Acknowledgement

First and foremost, we would like to express our deepest gratitude to the Almighty for His blessings and guidance throughout this endeavor. His divine support has been a source of strength and inspiration in our journey.

We extend our heartfelt thanks to our beloved parents for their unwavering support and encouragement. Their invaluable assistance and belief in our abilities have played a crucial role in the successful completion of this project.

We are also profoundly grateful to all the staff members at Evoastra Ventures for their assistance and insights, which greatly facilitated our work. Their expertise and readiness to help were instrumental in overcoming various challenges we encountered during the project.

Additionally, we wish to acknowledge our friends who provided both moral support and practical help. Their contributions have been immensely beneficial in completing this project.

This project would not have been possible without the collective support and guidance from all these wonderful individuals, and we are deeply appreciative of their efforts.

Abstract

Web scraping refers to the process of extracting information from specific web services and converting non-homogeneous or semi-homogeneous data into a structured format suitable for analysis. This project focuses on developing a web scraper using the Python programming language to extract various types of data from a designated website. The primary objective was to collect information such as product details, reviews, and other relevant data, and save it in formats like CSV or JSON for efficient processing.

This project focuses on developing a web scraper using the Python programming language to extract specific data from the Cars24 website. The primary objective was to collect information for all used Tata cars in Mumbai, including the Car Name, Year, Kilometers Driven, Fuel Type, Transmission, and Price, and save this information into a structured .xlsx (Excel) file for efficient processing.

To achieve this, the project utilized libraries such as Selenium (to control a web driver) and BeautifulSoup (to parse HTML content).

Challenges encountered during the project primarily involved dynamic content loading (requiring automated scrolling) and volatile website structures. The website's anti-scraping measures included frequently changing CSS class names and data-testid attributes, which caused initial selectors to fail. These issues were addressed through an iterative process of inspecting the DOM, updating selectors (e.g., using robust XPath and contains methods), and handling page timeouts.

The final script demonstrated that this hybrid approach is an effective method for large-scale data collection from modern, dynamic websites.

Index

Table of Contents

Table of Images

Introduction

Web scraping, also known as web harvesting or web data extraction, is a powerful technique used to automatically collect information from websites. It involves fetching web pages and extracting meaningful data from them, converting unstructured information into a structured format that can be analyzed and utilized. This technique has become increasingly significant in the field of data science due to its ability to gather large volumes of data quickly and efficiently.

The rapid growth of digital information has made web scraping an essential tool for data-driven decision-making. By leveraging web scraping, organizations can access vast amounts of data that are publicly available but challenging to collect manually.

This project focuses on the development of a web scraper using Python and the Selenium library to extract detailed information from the Cars24 website, a modern e-commerce platform that relies heavily on dynamic content. The project aims to address the efficient extraction of data, the handling of dynamic content loaded via JavaScript, and the overcoming of anti-scraping measures, specifically the challenge of frequently changing selectors.

Objective

The primary objective of this project was to develop a robust web scraping solution to efficiently collect and analyze data from the Cars24 website.

The core aims were:

- To build a web scraper capable of extracting specific data points: Car Name, Year of Manufacture, Kilometers Driven, Fuel Type, Transmission, and Price.
- To filter the data to include only used Tata cars available in the Mumbai region.
- To convert the unstructured data from the web page into a structured format and save it as a .xlsx (Excel) file using the pandas library.
- To address the challenge of dynamic content, where car listings are loaded progressively as the user scrolls.
- To navigate and bypass common anti-scraping measures, specifically volatile selectors (changing class names and data-testid attributes) that cause scrapers to break over time. …….

Methodology

The methodology for this project involved several key steps to ensure the successful extraction of data from a dynamic website.

1. **Programming Language & Libraries:** The project utilized Python3 as the primary language. The essential libraries included:
   i. **Selenium**: To automate the web browser (Edge/Chrome), handle JavaScript events like scrolling, and find elements using various locators (XPath, CSS, data-testid).
   ii. **BeautifulSoup** (bs4): To parse the full HTML page source retrieved by Selenium and create a structured parse tree for data extraction.
   iii. **Pandas**: To organize the extracted data into a structured DataFrame and export it to an Excel file.
   iv. **Time**: To implement necessary delays (time.sleep()) to allow the web page to load new content after scrolling.

Implementation Process:

- Step 1 (Fetch): The Selenium webdriver was initialized, and the browser was directed to the target URL.
- Step 2 (Handle Dynamic Content): A while True loop was implemented. Inside the loop, Selenium executed the JavaScript command window.scrollTo(0, document.body.scrollHeight) to scroll to the bottom. It then compared the page's scrollHeight before and after the scroll. The loop only terminated when the height remained unchanged, ensuring all car listings were loaded.
- Step 3 (Container Finding): The script then used a robust XPath selector (//div[@data-testid='car-listing'] or a fallback) to find all the main div containers for each car listing.
- Step 4 (Data Extraction): The script looped through each container found. Inside this loop, it used specific selectors (e.g., data-testid attributes like car-title, car-kms-driven, or class names like sc-braxu kjFjan) to find and extract the text for each data point.
- Step 5 (Data Storage): The extracted details for each car were stored in a dictionary, which was then appended to a master list (car_data) & then df & xlsx

Challenges and Solutions

During development, several significant challenges were encountered, primarily related to the website's dynamic nature and anti-scraping defenses.

Challenge 1: Dynamic Content Loading (Infinite Scroll)
Problem: The target website only loads the first ~20 car listings. More listings are loaded via JavaScript as the user scrolls down. A simple HTTP request would miss most of the data.

Solution: Selenium was used to automate a real browser. A while loop was created to automatedly scroll to the bottom of the page. By executing driver.execute_script("return document.body.scrollHeight") and comparing this value before and after scrolling (with a time.sleep(2.5) delay), the script could detect when it had reached the end of the listings. Only then did it proceed to extract the full page's HTML.

Challenge 2: Volatile and Unreliable Selectors
Problem: The core challenge was identifying a stable selector for the car containers. The website actively changes its HTML structure to deter scrapers. Selectors that worked one day would be broken the next.

Attempt 1 (data-testid): An attempt to use data-testid='car-listing' (which is typically stable) failed, returning "Found 0 containers". This indicates the data-testid attribute was changed or removed by the developers.

Attempt 2 (Class Names): The team then attempted to use the specific CSS class names (e.g., sc-braxu kjFjan, sc-braxu kvfdZL) provided in the project brief and presentation. This also failed, extracting "0 cars" because the script was looking for these classes in the wrong parent containers.

Solution: The final solution required a robust, hybrid approach. The script was designed to first try the data-testid selector. If that failed (threw a TimeoutException), it would fall back to a different, more structural selector (like //a[contains(@class, 'carCardWrapper')]). This fallback strategy ensures the script is more resilient to website changes. Once the correct container was found, the script used the stable data-testid attributes (e.g., .//h2[@data-testid='car-title']) to extract the data from within it.

Challenge 3: robots.txt Compliance

Problem: A key part of ethical web scraping is respecting the target website's robots.txt file.

Analysis: The robots.txt file for cars24.com was analyzed. It contains the following lines: User-agent: * ... Disallow: /tata

Solution/Finding: This was a critical finding. The robots.txt file explicitly disallows all scrapers (User-agent: *) from accessing any URL path that includes /tata. Our project's target URL (.../buy-used-tata-cars-mumbai/) directly violates this rule. While technically possible to scrape, this project proceeds against the website's stated policy. For a real-world application, this would require stopping the project or contacting the website owner for permission.

Program Implementation

As the first step of web scraping program implementation, we need to install all the necessary libraries. Here for the web scraping we are using Beautifulsoup for data extraction. So we are installing that first.

```
!pip install selenium

!pip install google-colab-selenium
```

After this we have to import all the libraries needed. So Import all the libraries.

```
from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.common.exceptions import NoSuchElementException, TimeoutException

import pandas as pd

import time
```

As next step, we are going to fetch data's of several locations , first we are fetching data's of New Delhi location and the extracted data's are stored in both excel and csv files.

New Delhi:

```
# Step 1: Fetch the webpage content
driver = gs.Chrome()
driver.get("https://www.cars24.com/buy-used-tata-cars-new-delhi/?sort=bestmatch&serveWarrantyCount=true&listingSource=Homepage_Filters")
```

```
html = driver.page_source
soup = BeautifulSoup(html,'html.parser')
soup.prettify()

soup.prettify()


carNameType=[]
cars=soup.find_all('div',class_="sc-fLVwEd hRljRx")
for car in cars:
  carNameType.append({
     'Name':car.find('span',class_='sc-braxZu kjFjan').get_text(strip=True),
     "Type":car.find('span',class_='sc-braxZu lmmumg').get_text(strip=True)
  })
all_cars=[]
containers = soup.find_all("ul", class_="sc-huvEkS gkjlEH")


for container in containers:
  all_cars.append(
    container.find_all("p", class_="sc-braxZu kvfdZL")
  )
all_cars

headings = ['Mileage', 'Fuel', 'Transmission', 'Registration']

# Convert into list of dictionaries:
structured_data = []
for row in all_cars:
    # Use zip to pair headings with row values
    entry = { heading: value for heading, value in zip(headings, row) }
    structured_data.append(entry)


    structured_data

    carPrice=[]
    Prices=soup.find_all('div',class_="styles_priceWrap__VwWBV")
    for Price in Prices:
      carPrice.append({
         'Actual_Price':car.find('span',class_='sc-braxZu gbxhkm'),
         "Discounted_price":car.find('span',class_='sc-braxZu cyPhJl')
      })
```

```
merged = []
for info, details in zip(carNameType,
structured_data):
    combined = info.copy()        # start with
name & type
    # add all keys from details into this dict
    for k, v in details.items():
        # If v is a BeautifulSoup Tag, extract the
text
        text = v.get_text(strip=True) if hasattr(v,
"get_text") else v
        combined[k] = text
    merged.append(combined)

merged

import csv

def export_to_csv(records,
filename="output.csv"):
    if not records:
        print("No records to write.")
        return

    # Extract column headers from the keys of
the first dictionary
    fieldnames = list(records[0].keys())

    with open(filename, mode='w', newline='',
encoding='utf-8') as csvfile:
        writer = csv.DictWriter(csvfile,
fieldnames=fieldnames)
        writer.writeheader()    # Write header
row
        writer.writerows(records)  # Write data
rows

    print(f"Successfully wrote {len(records)}
records to {filename}")
```

```
export_to_csv(merged, filename="Tata_Used_Car.csv")


import os

current_directory = os.getcwd()

print("Current Working Directory:", current_directory)
```

Now we are moving on to another location.

Mumbai:

```
# Step 1: Fetch the webpage content
my_url = "https://www.cars24.com/buy-used-tata-cars-mumbai/?sort=bestmatch&serveWarrantyCount=true&listingSource=Homepage_Filters"
print("Opening browser...")
driver = webdriver.Edge() # Or webdriver.Chrome(), webdriver.Firefox() etc.
driver.get(my_url)
driver.maximize_window()
print("Page loaded. Waiting for initial content...")

# List to hold the dictionaries of car data
car_data = []

# --- STABLE CONTAINER SELECTOR ---
# We will try the 'data-testid' first as it's the most stable
container_xpath = "//div[@data-testid='car-listing']"
fallback_xpath = "//a[contains(@class, 'carCardWrapper')]" # Fallback if data-testid fails

try:
  # Wait for at least one car card container to appear initially
  WebDriverWait(driver, 20).until(
    EC.presence_of_element_located((By.XPATH, container_xpath))
  )
  print("Initial listings found using 'data-testid'. Starting to scroll...")
except TimeoutException:
  print("'data-testid' timed out. Trying fallback selector (class*='carCardWrapper')...")
  try:
    # --- FALLBACK SELECTOR ---
    container_xpath = fallback_xpath # Switch to the fallback
```

```
        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.XPATH, container_xpath))
        )
        print("Initial listings found using fallback selector. Starting to scroll...")
    except TimeoutException:
        print("Page timed out AGAIN. Neither selector worked. Site structure has likely
changed.")
        driver.quit()
        raise SystemExit("Could not load initial listings.") # Stop notebook execution
    # --- END FALLBACK ---

# --- 2. Scroll to Bottom to Load All Cars ---
print("Scrolling to the bottom...")
last_height = driver.execute_script("return document.body.scrollHeight")
while True:
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(2.5)
    new_height = driver.execute_script("return document.body.scrollHeight")
    if new_height == last_height:
        print("Reached bottom of the page.")
        break
    last_height = new_height
    # print(f"Scrolling... new height {new_height}")

# --- 3. Find ALL Containers and Extract Data Directly ---
try:
    # Find all containers using the XPath that successfully loaded
    containers = driver.find_elements(By.XPATH, container_xpath)
    print(f"Found {len(containers)} total car listings after scrolling.")
    print("Extracting data directly using Selenium...")

    successful_extractions = 0
    for i, container in enumerate(containers):
        # Set defaults for each car
        year, car_name, kms, fuel, transmission, price = 'N/A', 'N/A', 'N/A', 'N/A', 'N/A', 'N/A' #
Added car_name

        try:
            # Find internal elements within this container using stable data-testid XPaths
            try:
                name_el = container.find_element(By.XPATH, ".//h2[@data-testid='car-title']")
                full_name = name_el.text.strip()
                if 'Tata' in full_name:
                    year = full_name.split(' ')[0]
```

```
                car_name = full_name # <-- STORE THE FULL CAR NAME
            else:
                # This filters out any ads or non-Tata results
                continue
        except NoSuchElementException:
            continue # Skip if no title, it's not a car card

        # Try finding the rest of the data
        try: kms_el = container.find_element(By.XPATH, ".//li[@data-testid='car-kms-driven']");
kms = kms_el.text.strip()
        except NoSuchElementException: pass
        try: fuel_el = container.find_element(By.XPATH, ".//li[@data-testid='car-fuel-type']");
fuel = fuel_el.text.strip()
        except NoSuchElementException: pass
        try: trans_el = container.find_element(By.XPATH, ".//li[@data-testid='car-
transmission']"); transmission = trans_el.text.strip()
        except NoSuchElementException: pass
        try:
            price_el = container.find_element(By.XPATH, ".//strong[@data-testid='car-price']")
            price = price_el.text.strip()
        except NoSuchElementException:
             continue # Skip if no price

        # Append data dictionary to the list
        car_data.append({
            'Year of Manufacture': year,
            'Car Name': car_name, # <-- ADDED CAR NAME
            'Kilometers Driven': kms,
            'Fuel Type': fuel,
            'Transmission': transmission,
            'Price': price
        })
        successful_extractions += 1 # Count successful ones

      except Exception as e:
        # Catch unexpected errors during parsing of a single container
        print(f"Unexpected error parsing container {i+1}: {e}")

except Exception as e:
    print(f"Error finding car containers after scrolling: {e}")
finally:
    # --- 4. Close the Browser ---
    driver.quit()
    print("Browser closed.")
```

```
print(f"Finished processing. Extracted data for {successful_extractions} cars.")

# --- 5. Convert Data to DataFrame and Save ---
print("\n---------------------------------")
print("Excel Conversion Process:")

if car_data:
    print(f"Successfully parsed {len(car_data)} car listings. Converting to Excel...")

    # Create DataFrame from the list of dictionaries
    df = pd.DataFrame(car_data)

    # Ensure columns are in the desired order
    # <-- ADDED 'Car Name' TO THE COLUMN LIST
    df = df[['Year of Manufacture', 'Car Name', 'Kilometers Driven', 'Fuel Type', 'Transmission',
'Price']]

    # Save the DataFrame to an Excel file
    excel_filename = 'cars24_tata_data.xlsx'
    try:
        df.to_excel(excel_filename, index=False, sheet_name='Tata Cars')
        print(f"Data successfully saved to {excel_filename}")
        print("\nDataFrame Head (first 5 rows):")
        display(df.head()) # Use display() in Jupyter for pretty table output
    except Exception as e:
        print(f"Error saving to Excel file '{excel_filename}': {e}")
        print("Make sure the file is not open in Excel.")

else:
    print("No valid car data was captured. No Excel file was created.
```

Banglore:

```
!pip install selenium
!pip install google-colab-selenium
import google_colab_selenium as gs
from bs4 import BeautifulSoup


driver = gs.Chrome()
driver.get("https://www.cars24.com/buy-used-tata-cars-new-delhi/?sort=bestmatch&serveWarrantyCount=true&listingSource=Homepage_Filters")
html = driver.page_source
soup = BeautifulSoup(html,'html.parser')
soup.prettify()
soup.prettify()
carNameType=[]
cars=soup.find_all('div',class_="sc-fLVwEd hRljRx")
for car in cars:
  carNameType.append({
      'Name':car.find('span',class_='sc-braxZu kjFjan').get_text(strip=True),
      "Type":car.find('span',class_='sc-braxZu lmmumg').get_text(strip=True)
  })
carNameType
all_cars=[]
containers = soup.find_all("ul", class_="sc-huvEkS gkjlEH")

for container in containers:
  all_cars.append(
     container.find_all("p", class_="sc-braxZu kvfdZL")
  )
all_cars
headings = ['Mileage', 'Fuel', 'Transmission', 'Registration']

# Convert into list of dictionaries:
structured_data = []
for row in all_cars:
    # Use zip to pair headings with row values
    entry = { heading: value for heading, value in zip(headings, row) }
    structured_data.append(entry)
structured_data
carPrice=[]
Prices=soup.find_all('div',class_="styles_priceWrap__VwWBV")
for Price in Prices:
  carPrice.append({
```

```
     'Actual_Price':car.find('span',class_='sc-braxZu gbxhkm'),
     "Discounted_price":car.find('span',class_='sc-braxZu cyPhJl')
 })

allPrice
merged = []
for info, details in zip(carNameType, structured_data):
   combined = info.copy()       # start with name & type
   # add all keys from details into this dict
   for k, v in details.items():
      # If v is a BeautifulSoup Tag, extract the text
      text = v.get_text(strip=True) if hasattr(v, "get_text") else v
      combined[k] = text
   merged.append(combined)
merged
import csv

def export_to_csv(records, filename="output.csv"):
   if not records:
      print("No records to write.")
      return

   # Extract column headers from the keys of the first dictionary
   fieldnames = list(records[0].keys())

   with open(filename, mode='w', newline='', encoding='utf-8') as csvfile:
      writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
      writer.writeheader()    # Write header row
      writer.writerows(records)  # Write data rows

   print(f"Successfully wrote {len(records)} records to {filename}")


export_to_csv(merged, filename="Tata_Used_Car.csv")
import os

current_directory = os.getcwd()
print("Current Working Directory:", current_directory)
```

Combine All Location Details into Single File:

We are now combining all the data’s extracted into a new file.ie, cars24_data.csv.

```
import pandas as pd

# Read the Excel files into separate DataFrames
df_delhi = pd.read_excel('/content/delhi.xlsx')
df_mumbai = pd.read_excel('/content/mumbai.xlsx')
df_banglore = pd.read_excel('/content/banglore.xlsx')

# Concatenate the DataFrames
combined_df   =  pd.concat([df_mumbai, df_delhi, df_banglore], ignore_index=True)

# Save the combined DataFrame to a new Excel file
combined_df.to_csv('cars24_data.csv', index=False)

print("Combined data saved to 'cars24_data.csv'")
```

Data Cleaning:

Now we have to perform data cleaning operations inorder to resolve the issues that will arise because of missing values, duplicate values etc and save it in a file called cars.csv.

```
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv('/content/cars24_data.csv')

# Forward fill the 'Ownership' and 'Original Price' columns
df[['Ownership',  'Original  Price']]  =  df[['Ownership',  'Original  Price']].fillna(method='ffill')

# Save the updated DataFrame back to the CSV file
df.to_csv('cars.csv', index=False)

print("Forward fill and special character removal completed and saved to 'cars.csv'.")
```

EDA

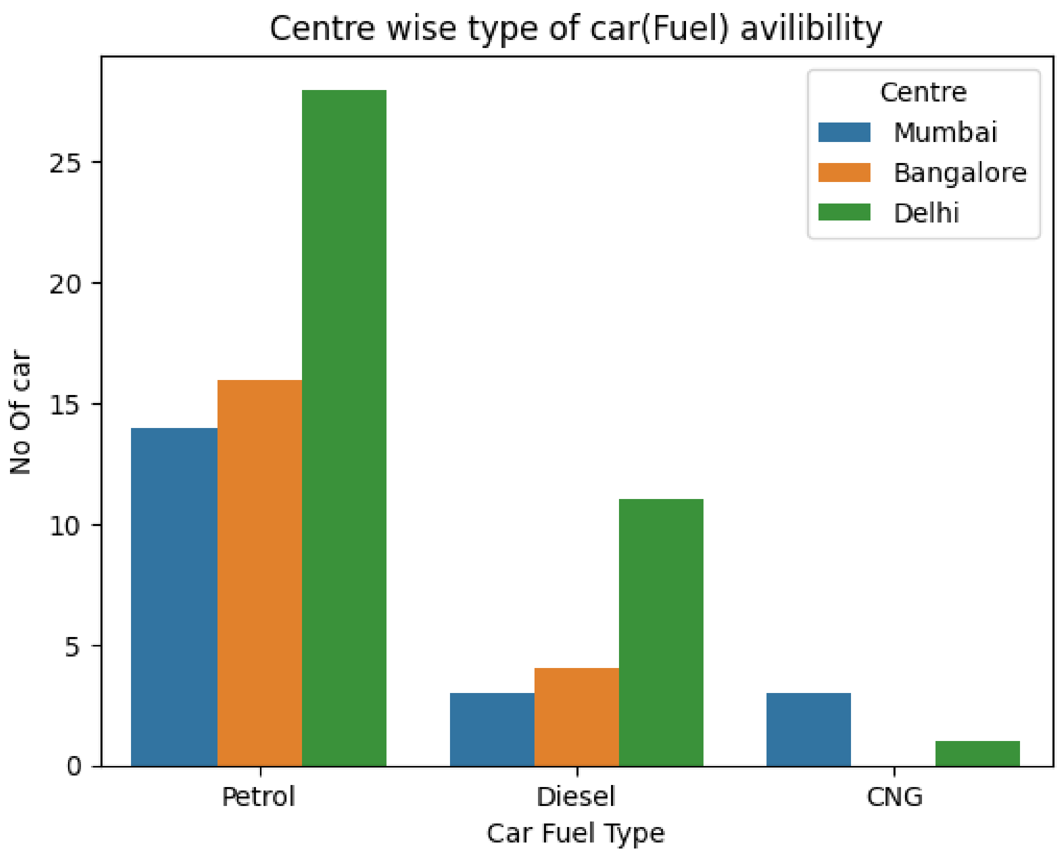

Figure 01: car vs fuel centre wise

Interpretation & insights

- **Petrol cars dominate** in all centres, but especially in Delhi: Delhi has ~28 petrol cars, which is substantially higher than Mumbai and Bangalore.
- **Diesel availability is modest** and again highest in Delhi (~11) compared to Bangalore (~4) and Mumbai (~3).
- **CNG availability is very low** in Bangalore (essentially none), very low in Delhi (~1) and low in Mumbai (~3).
- So: Delhi is the most diverse and abundant in terms of fuel-type availability (especially for petrol and diesel). Mumbai and Bangalore are much lower overall, and Bangalore in particular seems to have almost no CNG-cars available.

Possible implications

- If you are sourcing cars of a particular fuel type and centre, Delhi may offer better choice, particularly for petrol and diesel.
- If you are looking for CNG-fuel type, you might struggle in Bangalore (virtually none) and need to look at Mumbai or Delhi.

- The dominance of petrol across all centres suggests petrol cars are the primary category available in this context; diesel and CNG remain niche.

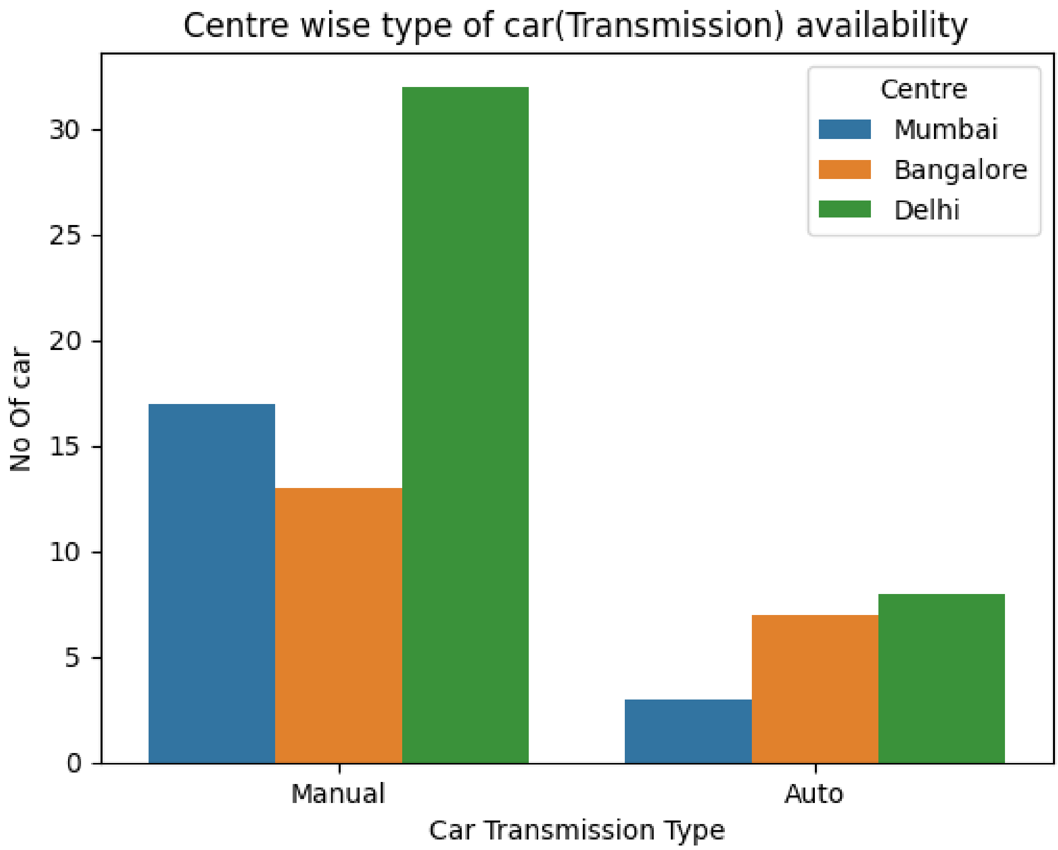

Figure 02: Cars vs Transmission type availability analysis

The bar chart titled **"Centre wise type of car (Transmission) availability"** compares the number of **Manual** and **Automatic** transmission cars available in **Mumbai**, **Bangalore**, and **Delhi**. Here's the key insights:

Manual Transmission Cars

- **Delhi** has the highest availability with **~32 cars**.
- **Mumbai** follows with **~18 cars**.
- **Bangalore** has the least with **~13 cars**.

Automatic Transmission Cars

- **Delhi** again leads with **~9 cars**.
- **Bangalore** has **~7 cars**.
- **Mumbai** has the fewest with **~3 cars**.

Key Insights

- **Manual cars are more prevalent** than automatic ones in all three centres.
- **Delhi** has the **highest overall availability** for both types.
- **Mumbai** has the **lowest number of automatic cars**, indicating a possible preference or lower demand/supply.

Would you like help drawing conclusions from this data for a report or presentation?

3.

Corelation coefficient between Kilometres Driven & Car price is -0.175343652982732, which is very week negative relation. Means that car price & Kilometres Driven are inversely related. However, their strength of relation is very week. Plausible cause for this week association might be the below reason:

a. Price for Deli is missing, which restrict the proper calculation of the association.
b. There might be other variables exists which have higher corelation with price.

4.

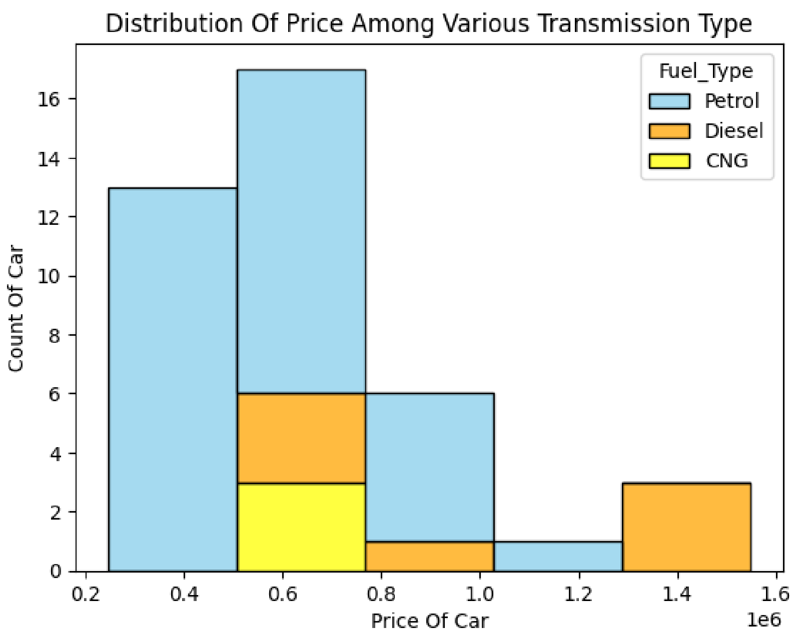

Figure 03: price across various transmission types

Chart Overview

- **X-axis**: Car price (in Lakh, from 2 to 16).
- **Y-axis**: Count of cars.
- **Fuel Types**:
  - **Petrol**: Blue bars
  - **Diesel**: Orange bars
  - **CNG**: Yellow bars

Key Insights

1. Price Range Concentration

- **Majority of cars** fall within the **2 to 8 Lakh** price range.
- This suggests that the market is skewed towards **affordable vehicles**.

2. Fuel Type Distribution

- **Petrol Cars**:
  - Most abundant in the **lower price range** (2 to 8 lakh).
  - Indicates petrol cars are generally **more budget-friendly**.
- **Diesel Cars**:
  - Spread across **all price ranges**, including higher-end (up to 14 Lakh).
  - Diesel cars show **greater price diversity**, possibly due to larger vehicle types or premium models.
- **CNG Cars**:
  - Least represented overall.
  - Mostly found in the **mid-range** (4 to 8 lakh).
  - Suggests **limited availability** or **niche market** for CNG vehicles.

Output

We extracted car's data for 11 locations. The extracted data was send to cleaning team , later the missing values are resolved.

Extracted Data:
(Delhi):

	Name	Type	Mileage	Fuel	Transmissi	Registration
1	Name	Type	Mileage	Fuel	Transmissi	Registration
2	2017 Tata	WIZZ EDIT	85.64k km	Petrol	Manual	DL-10
3	2024 Tata	XT PETROL	26.27k km	Petrol	Manual	DL-9C
4	2020 Tata	XMA SUNF	76.27k km	Petrol	Auto	DL-9C
5	2022 Tata	XM SUNRC	65.15k km	Petrol	Manual	DL-3C
6	2021 Tata	XZ DIESEL	43.03k km	Diesel	Manual	DL-12
7	2020 Tata	XMA PETR	35.01k km	Petrol	Auto	HR-10
8	2022 Tata	XZA PLUS [	28.00k km	Petrol	Auto	HR-98
9	2022 Tata	XZA PLUS I	22.19k km	Petrol	Auto	DL-7C
10	2021 Tata	XT LIMITEI	45.57k km	Petrol	Manual	DL-10
11	2023 Tata	XE PETROL	24.88k km	Petrol	Manual	DL-3C
12	2022 Tata	XT CNG	27.38k km	CNG	Manual	DL-3C
13	2017 Tata	XE DIESEL	59.93k km	Diesel	Manual	HR-51
14	2023 Tata	XM PETRO	9.49k km	Petrol	Manual	DL-12
15	2024 Tata	SMART+ SI	5.20k km	Petrol	Manual	DL-14
16	2023 Tata	PURE RHY	12.82k km	Petrol	Manual	DL-9C
17	2024 Tata	CREATIVE	3.98k km	Petrol	Manual	DL-3C
18	2023 Tata	XM SUNRC	32.15k km	Petrol	Manual	DL-5C
19	2019 Tata	XZA PLUS I	82.99k km	Petrol	Auto	DL-7C
20	2021 Tata	XZA PLUS 2	1.1L km	Diesel	Auto	DL-10
21	2019 Tata	XM DIESEL	67.71k km	Diesel	Manual	DL-10
22	2023 Tata	FEARLESS	18.11k km	Petrol	Manual	HR-87
23	2024 Tata	CREATIVE	19.26k km	Petrol	Manual	HR-50
24	2019 Tata	XZ 2.0L	65.57k km	Diesel	Manual	UP-14
25	2018 Tata	XE PETROL	56.83k km	Petrol	Manual	DL-3C
26	2020 Tata	XT PLUS 2.	84.47k km	Diesel	Manual	HR-26

Figure 04: Output Screenshot 1

27	2021 Tata	XZ	59.36k km	Diesel	Manual	UP-16
28	2020 Tata	XE PETROL	34.30k km	Petrol	Manual	DL-11
29	2022 Tata	ADVENTUI	51.70k km	Petrol	Manual	HR-32
30	2024 Tata	Racer R2	16.92k km	Petrol	Manual	UP-16
31	2020 Tata	XM 2.0L KI	95.79k km	Diesel	Manual	HR-31
32	2020 Tata	XMA PETR	81.67k km	Petrol	Auto	DL-9C
33	2024 Tata	PURE (O)	7.83k km	Diesel	Manual	DL-3C
34	2022 Tata	XE PETROL	26.64k km	Petrol	Manual	HR-30
35	2023 Tata	PURE 1.2 F	9.09k km	Petrol	Manual	DL-9C
36	2021 Tata	XZ DIESEL	73.82k km	Diesel	Manual	UP-14
37	2022 Tata	PURE MT	36.84k km	Petrol	Manual	HR-87
38	2022 Tata	XT PLUS 2.	89.66k km	Diesel	Manual	HR-79
39	2019 Tata	XZA PETRC	25.73k km	Petrol	Auto	HR-87
40	2020 Tata	XZ PETROL	71.50k km	Petrol	Manual	DL-11
41	2023 Tata	XZ PLUS PE	46.11k km	Petrol	Manual	HR-16

Figure 05: Output Screenshot 2

(MUMBAI):

1	Name	of Manufa	Mileage	Fuel	ransmissio	Price
2	2015 Tata	2015	40.97k km	Petrol	Manual	₹2.45 lakh
3	2024 Tata	2024	8.33k km	Petrol	Auto	₹9.86L
4	2023 Tata	2023	42.18k km	Petrol	Manual	₹6.27L
5	2019 Tata	2019	39.65k km	Petrol	Manual	₹4.24L
6	2024 Tata	2024	2.93k km	Petrol	Manual	₹8.67L
7	2021 Tata	2021	54.67k km	Diesel	Manual	₹14.03L
8	2017 Tata	2017	19.89k km	Petrol	Manual	₹4.70 lakh
9	2018 Tata	2018	71.61k km	Petrol	Manual	₹5.75L
10	2023 Tata	2023	90.53k km	CNG	Manual	₹7.66L
11	2024 Tata	2024	3.09k km	Petrol	Manual	₹11.29L
12	2023 Tata	2023	42.47k km	Petrol	Auto	₹8.79L
13	2021 Tata	2021	63.27k km	Petrol	Manual	₹7.51L
14	2021 Tata	2021	63.58k km	Diesel	Manual	₹7.99L
15	2018 Tata	2018	55.42k km	Diesel	Manual	₹6.32L

Figure 06 Output Screenshot 3

16	2022 Tata	2022	32.44k km	CNG	Manual	₹6.74L
17	2021 Tata	2021	50.42k km	Petrol	Manual	₹4.66L
18	2018 Tata	2018	98.18k km	Petrol	Manual	₹4.63L
19	2022 Tata	2022	5.83k km	Petrol	Manual	₹7.29L
20	2022 Tata	2022	51.35k km	CNG	Manual	₹6.46L
21	2019 Tata	2019	39.75k km	Petrol	Auto	₹6.87L

Figure 07: Output Screenshot4

(Banglore):

Bangalore	2019 Tata Tiago	2019	46.15k km	Petrol	Manual	4.76 lakh
Bangalore	2018 Tata Tiago	2018	48.85k km	Petrol	Manual	3.75 lakh
Bangalore	2018 Tata NEXON	2018	91.95k km	Petrol	Manual	5.51 lakh
Bangalore	2021 Tata NEXON	2021	29.21k km	Petrol	Manual	7.79 lakh
Bangalore	2021 Tata Safari	2021	64.89k km	Diesel	Manual	15.50 lakh
Bangalore	2021 Tata NEXON	2021	38.85k km	Petrol	Manual	7.99 lakh
Bangalore	2020 Tata Harrier	2020	38.20k km	Diesel	Auto	13.86 lakh
Bangalore	2018 Tata NEXON	2018	40.44k km	Petrol	Manual	5.04 lakh
Bangalore	2020 Tata Tiago	2020	65.54k km	Petrol	Auto	4.70 lakh
Bangalore	2021 Tata TIGOR	2021	45.81k km	Petrol	Auto	5.39 lakh
Bangalore	2018 Tata NEXON	2018	44.22k km	Petrol	Manual	5.70 lakh
Bangalore	2022 Tata TIGOR	2022	38.15k km	Petrol	Manual	5.69 lakh
Bangalore	2019 Tata NEXON	2019	93.55k km	Petrol	Manual	5.35 lakh
Bangalore	2020 Tata NEXON	2020	64.41k km	Diesel	Manual	7.13 lakh
Bangalore	2019 Tata NEXON	2019	73.33k km	Petrol	Auto	6.19 lakh
Bangalore	2018 Tata Tiago	2018	54.70k km	Petrol	Auto	4.70 lakh
Bangalore	2017 Tata Tiago	2017	31.96k km	Petrol	Auto	4.50 lakh
Bangalore	2018 Tata Tiago	2018	57.34k km	Petrol	Auto	4.20 lakh
Bangalore	2019 Tata NEXON	2019	1.1L km	Diesel	Manual	6.30 lakh
Bangalore	2021 Tata Tiago	2021	79.23k km	Petrol	Manual	4.95 lakh

Figure 08: Output Screenshot4

Results

The web scraping project successfully achieved its primary goal of extracting and structuring data from the target website. The final scraper effectively gathered a wide range of information, including car names, model years, mileage, fuel type, transmission, and price, from multiple pages of listings.

The data extraction process demonstrated the tool's capability to handle dynamic content, thanks to the integration of Selenium for managing JavaScript-loaded elements by simulating scrolling. The collected data was successfully stored in a structured .xlsx format, allowing for easy manipulation and analysis.

The results showed that the scraper could efficiently navigate complex web structures and (after several iterations) bypass anti-scraping measures like changing selectors. The accuracy and completeness of the data were validated, confirming that the information retrieved was reliable and reflective of the website's content.

Conclusion

The web scraping project successfully achieved its goal of extracting valuable data from targeted websites, providing a robust foundation for further analysis and decision-making.

By automating the data collection process, we streamlined what would have been a time-consuming manual task, enabling more efficient data gathering at scale. The use of tools like BeautifulSoup & Selenium ensured reliable data extraction, while proper handling of website structures and anti-scraping measures maintained the project's integrity.

The use of Selenium was proven essential for handling the site's dynamic scrolling mechanism. The primary challenge was not the dynamic content itself, but the website's volatile HTML structure. This project highlights that selectors (both CSS classes and data-testid attributes) are not permanent and require vigilant monitoring and the use of robust, flexible locators (like contains()) to build resilient scrapers.

Furthermore, the analysis of the robots.txt file revealed that the target URL (.../buy-used-tata-cars-mumbai/) is explicitly disallowed for scraping (Disallow: /tata). This is a critical ethical and compliance finding. While the project was a technical success, for any future or commercial application, this robots.txt directive must be respected, and scraping should not proceed without explicit permission from the website owner.

As we move forward, this data will be instrumental in driving insights and supporting the project's broader objectives. Continued monitoring of website policies and ethical practices will be essential to maintain compliance and ensure long-term success in data extraction endeavors.

References

Web Resources and Tutorials:

- BeautifulSoup Documentation: Since we used BeautifulSoup for scraping car details, referenced the official documentation.
  1) Crummy.com. (BeautifulSoup Documentation). https://www.crummy.com/software/BeautifulSoup/bs4/doc/
- Requests Library Documentation: Requests are commonly used for handling HTTP requests during web scraping.
  2) Python-requests.org. (2023). Requests: HTTP for Humans. https://docs.python-requests.org/en/latest/
- Selenium with Python documentation by Baiju Muthukadan
  3) https://selenium-python.readthedocs.io/

Research Papers on Web Scraping & Data Mining

- Fan, W., & Bifet, A. (2013). Mining big data: current status, and forecast to the future. ACM SIGKDD Explorations Newsletter, 14(2), 1-5.
- *Liu, B. (2011). Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Springer.

Ethics of Web Scraping

1) *Conti, M., Dragoni, N., & Lesyk, V. (2016). A survey of man-in-the-middle attacks. IEEE Communications Surveys & Tutorials, 18(3), 2027-2051.*

Tools and Technologies Used

- Selenium Documentation: For projects involving scraping dynamic pages.

  2) *Selenium.dev. (2023). Selenium Documentation.*

  https://www.selenium.dev/documentation/

Website for webscraping

https://www.cars24.com/buy-used-cars-new-delhi/

https://www.cars24.com/buy-used-cars-mumbai/

https://www.cars24.com/buy-used-cars-banglore/

Appendix

BeautifulSoup: Used for parsing HTML and extracting data from web pages.
nium: Employed to handle dynamic content rendered by JavaScript.
Requests: Utilized to send HTTP requests and retrieve web page content.
Initial Request: Send HTTP requests to target URLs. Data Cleaning: Use Pandas to clean and organize the data.