**Assignment 1**

1. **Code**

# IMPORTING NECESSARY LIBRARIES

import time

import pandas as pd

import requests

from bs4 import BeautifulSoup

import plotly.express as px

import numpy as np

import json

with open("./IRL\_ADM1.json", "r") as fp:

  ireland\_regions\_geo = json.load(fp)

# Fix the geojson map

from geojson\_rewind import rewind

ireland\_regions\_geo = rewind(ireland\_regions\_geo,rfc7946=False)

basic\_url = "https://www.cars.ie/used-cars?page="

PAGES\_TO\_DOWNLOAD = 50

car\_url = []

car\_name = []

car\_price = []

car\_make = []

car\_model = []

car\_year = []

car\_county = []

car\_odometer = []

car\_fuel\_type = []

car\_color = []

car\_engine\_size = []

car\_transmission = []

car\_body\_type = []

car\_total\_prev\_owner = []

car\_total\_doors = []

car\_tax\_expiry = []

car\_nct\_expiry = []

car\_dealer\_name = []

car\_dealer\_address = []

car\_dealer\_phone\_num = []

car\_dealer\_franchise = []

dealer=[]

index = 1

# SIMPLE INFO RETRIEVAL

try:

    for page in range(PAGES\_TO\_DOWNLOAD + 1):

        if page == 0:

            continue

        else:

            url = basic\_url + str(page)

            webpage\_content = requests.get(url).text

            print("Requesting page num = " + str(page)+"...")

            soup = BeautifulSoup(webpage\_content, "html.parser")

            tables = soup.find\_all("div", class\_="car-listing-inner")

            # retrieves basic info about car e.g. price, make, model, car's page url, year of manufacturing, etc..

            for element in tables:

                if element == '\n':

                    continue

                a\_href\_element = element.a

                href = "http://www.cars.ie" + a\_href\_element['href']

                car\_url.append(href)

                info\_blocks = element.find\_all("h3", class\_="greenText")

                info\_block = info\_blocks[0]

                car\_name.append(info\_block.text)

                car\_make.append(str(info\_block.text).split()[0])

                car\_model.append(str(info\_block.text).split()[1])

                info\_blocks = element.find\_all("p", class\_="greenText price BP")

                info\_block = info\_blocks[0]

                car\_price.append(info\_block.text)

                info\_blocks = element.find\_all("div", class\_="col-xs-10")

                info\_block = info\_blocks[0]

                car\_year.append(str(info\_block.text).split()[0])

                info\_blocks = element.find\_all("p", class\_="text-right")

                info\_block = info\_blocks[0]

                car\_county.append(str(info\_block.text).split()[0])

finally:

  print("Finsihed requesting all pages")

# FURTHER INFO RETRIEVAL

# loops through all the car urls and stores information in relevant lists for later use

for url in car\_url:

    try:

        # The following url have to be skipped, as car info is not available on website but it is still shown in car listing

        print("Processing...Car #"+str(index))

        page = requests.get(url)

        page\_content = page.text

        s = BeautifulSoup(page\_content, "html.parser")

        car\_tables = s.find\_all("div", class\_="stripped-table")

        car\_table = car\_tables[0]

        info\_blocks = car\_table.find\_all("div", class\_="row")

        info\_block = info\_blocks[0]

        car\_odometer.append(str(info\_block.text).split()[3])

        info\_block = info\_blocks[1]

        car\_fuel\_type.append(str(info\_block.text).split()[2])

        info\_block = info\_blocks[2]

        car\_color.append(str(info\_block.text).split()[1])

        info\_block = info\_blocks[3]

        car\_engine\_size.append(str(info\_block.text).split()[2])

        info\_block = info\_blocks[4]

        car\_transmission.append(str(info\_block.text).split()[1])

        info\_block = info\_blocks[5]

        car\_body\_type.append(str(info\_block.text).split()[2])

        info\_block = info\_blocks[6]

        car\_total\_prev\_owner.append(str(info\_block.text).split()[1])

        info\_block = info\_blocks[7]

        car\_total\_doors.append(str(info\_block.text).split()[1])

        info\_block = info\_blocks[8]

        car\_tax\_expiry.append(str(info\_block.text).split()[2])

        info\_block = info\_blocks[9]

        car\_nct\_expiry.append(str(info\_block.text).split()[2])

        index += 1

    except Exception:

        print("Could not process the data of following car --> "+str(url))

        continue

# SAVING DATA TO CSV FILE

print("Starting writing to csv file...")

a = {'Name': car\_name, 'Price': car\_price, 'Make': car\_make, 'Model': car\_model,

     'Engine Size': car\_engine\_size, 'Fuel Type': car\_fuel\_type,

    'Odometer': car\_odometer, 'Transmission': car\_transmission,

    'Body Type': car\_body\_type, 'Manufacturing Year': car\_year, 'County': car\_county,

    'Doors': car\_total\_doors, 'Color': car\_color, 'Owners': car\_total\_prev\_owner, 'Tax Expiry': car\_tax\_expiry,

    'NCT Expiry': car\_nct\_expiry, 'URL': car\_url}

df = pd.DataFrame.from\_dict(a, orient='index')

df = pd.DataFrame.transpose(df)

df.drop(df[df["Odometer"].isna()].index, inplace=True)

df.to\_csv("CarsIE.csv")

print("Finished writing to csv file")

print(df)

# PREPROCESSIG

def name\_preproc(car\_name):

    cName = str(car\_name).capitalize()

    return cName

def price\_preproc(car\_price):

    cPrice = str(car\_price).replace(",", "")

    if "€" in cPrice:

        cPrice = cPrice.replace("€", "")

        cPrice = int(cPrice)

    if cPrice == "POA":

        return 0

    return cPrice

def make\_preproc(car\_make):

    cMake = str(car\_make).capitalize()

    return cMake

def model\_preproc(car\_model):

    cModel = str(car\_model).capitalize()

    return cModel

def engine\_size\_preproc(car\_engine\_size):

    cEngSize = float(car\_engine\_size)

    return cEngSize

def fuel\_type\_preproc(car\_fuel\_type):

    cFuelType = str(car\_fuel\_type).capitalize()

    return cFuelType

def odometer\_preproc(car\_odometer):

    cOdometer = str(car\_odometer).replace(",", "")

    return cOdometer

def transmission\_preproc(car\_transmission):

    cTransmission = str(car\_transmission).capitalize()

    return cTransmission

def body\_type\_preproc(car\_body\_type):

    cBodyType = str(car\_body\_type).capitalize()

    return cBodyType

def manufacturing\_year\_preproc(car\_manufacturing\_year):

    cYear = int(car\_manufacturing\_year)

    if cYear == "NaN":

        cYear = df['Manufacturing Year'].fillna(0).astype(int)

    return cYear

def county\_preproc(car\_county):

    cCounty = str(car\_county).capitalize()

    return cCounty

def doors\_preproc(car\_doors):

    if type(car\_doors) is float:

        return 0

    if car\_doors == "-":

        return 0

    return int(car\_doors)

def color\_preproc(car\_color):

    if car\_color == "-":

        return 'Other'

    return car\_color

def owners\_preproc(car\_owners):

    if type(car\_owners) is float:

        return 0

    if car\_owners == "-":

        return 0

    return int(car\_owners)

def tax\_expiry\_preproc(car\_tax\_expiry):

    if car\_tax\_expiry == "-":

        return '-'

    return car\_tax\_expiry

def nct\_expiry\_preproc(car\_nct\_expiry):

    if car\_nct\_expiry == "-":

        return '-'

    return car\_nct\_expiry

# applying the preproc to csv file

df = pd.read\_csv("CarsIE.csv")

name = df['Name'].apply(name\_preproc)

price = df['Price'].apply(price\_preproc)

make = df['Make'].apply(make\_preproc)

model = df['Model'].apply(model\_preproc)

engine\_size = df['Engine Size'].apply(engine\_size\_preproc)

fuel\_type = df['Fuel Type'].apply(fuel\_type\_preproc)

odometer = df['Odometer'].apply(odometer\_preproc)

transmission = df['Transmission'].apply(transmission\_preproc)

body\_type = df['Body Type'].apply(body\_type\_preproc)

year = df['Manufacturing Year'].apply(manufacturing\_year\_preproc)

county = df['County'].apply(county\_preproc)

doors = df['Doors'].apply(doors\_preproc)

color = df['Color'].apply(color\_preproc)

owners = df['Owners'].apply(owners\_preproc)

tax\_expiry = df['Tax Expiry'].apply(tax\_expiry\_preproc)

nct\_expiry = df['NCT Expiry'].apply(nct\_expiry\_preproc)

df = pd.DataFrame({"Name": name, "Price": price, "Make": make, "Model": model,

                   "Engine": engine\_size, "Fuel": fuel\_type, "Odometer": odometer,

                   "Transmission": transmission, "Body": body\_type, "Year": year,

                   "County": county, "Doors": doors, "Color": color,

                   "Owners": owners, "Tax-Expiry": tax\_expiry, "NCT-Expiry": nct\_expiry

                   })

df.to\_csv("preprocData\_CarsIE.csv")

df = pd.read\_csv("preprocData\_CarsIE.csv", usecols=['Year', 'Price'])

# Draw a Scatter Chart for Year verses Price

fig = px.scatter(df, x='Year', y='Price', title='Prices of Cars Per Year')

fig.update\_xaxes(range=[1990, 2024])

fig.update\_yaxes(range=[-5000, 50000])

fig.show()

avg\_car\_price\_df = df.groupby("Year")["Price"].mean()

# Draw a Bar Chart to show the relationship between Year and Average Price

fig = px.bar(avg\_car\_price\_df, title='Average Car Prices Per Year', labels=dict(index="Year", value="Average Price €"))

fig.update\_xaxes(range=[1990, 2024])

fig.update\_yaxes(range=[-5000, 50000])

fig.show()

# Draw a box chart for for Year verses Price

fig = px.box(df, x='Year', y='Price', title='Prices of Cars Per Year')

fig.update\_xaxes(range=[1900, 2024])

fig.update\_yaxes(range=[-5000, 50000])

fig.show()

# Create a Scatter Facet to show the relationship between Year verse Average Price for different auto/manual gearbox, different Manufacturers , different Door Numbers and use mileage to change the scatter marker size

newDF = pd.read\_csv("preprocData\_CarsIE.csv", usecols=['Year', 'Price', 'Odometer', 'Make', 'Fuel', 'Doors', 'County'])

fig = px.scatter(newDF, x="Year", y="Price", color="Doors", facet\_col="Make", facet\_row="Fuel", size="Odometer", template="plotly\_dark", title="Relationship between Year verse Average Price for different gearbox, different manufacturers, different Door Numbers", width=6000, height=1000)

fig.update\_xaxes(range=[1990, 2024])

fig.update\_yaxes(range=[-5000, 50000])

fig.show()

# Use df.pivot\_table function to aggregate the average price for the cars of different Engine Types, different Manufacturers, different Door Numbers, and different Year info

dfp = newDF.pivot\_table(values="Price", index=["Year", "Fuel", "Make", "Doors"], aggfunc="mean").reset\_index()

dfp.sort\_values(["Year", ], inplace=True)

# Calculate the average price for each group and then draw a Line Facet plot

fig = px.line(dfp, x="Year", y="Price", color="Fuel", facet\_col="Make", facet\_row="Doors", template="plotly\_dark")

fig.update\_xaxes(range=[1990, 2024], showticklabels=True)

fig.update\_yaxes(range=[-5000, 50000])

fig.update\_layout(width=6000, height=1000)

fig.show()

# Using plotly.px.choropleth to create a geolocation chart to show the average car price in different counties, versus different years.

fig = px.choropleth(

    newDF,

    geojson=ireland\_regions\_geo,

    locations="County",

    color="Price",

    color\_continuous\_scale="reds",

    featureidkey="properties.NAME",

    range\_color=(0, df["Price"].max()),

    scope="europe",

    animation\_frame="Year",

    fitbounds="geojson",

    title="Map exhibiting the average car price in different counties, over the years. "

)

fig.update\_geos(visible=False)

fig.show()

1. **Graphs**
   1. A Scatter Chart for Year verses Price

Chart, histogram

Description automatically generated

* 1. A Bar Chart to show the relationship between Year and Average Price

Chart

Description automatically generated with low confidence

* 1. Chart, histogram

     Description automatically generatedA box chart for Year verses Price
  2. A Scatter Facet to show the relationship between Year verse Average Price for different auto/manual gearbox, different Manufacturers , different Door Numbers and use mileage to change the scatter marker size

Background pattern

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

* 1. A Line Facet plot

Graphical user interface, application

Description automatically generated

A picture containing electronics, night

Description automatically generated

* 1. Map exhibiting the average car price in different counties, over the years.

Chart, scatter chart

Description automatically generated

* 1. Pie Facet

1. **Conclusion**

For this assignment, I used plotly and pandas to visualize the data that I collected from scraping a website ([www.cars.ie](http://www.cars.ie)). I used the plotly and pandas to create different types of graphs such as Scatter chart which shows the Prices of cars per year. As per my understanding every graphs requires some sort of dataframe. In many cases the dataframe could be created using a pandas function called *pandas.DataFrame()*. However, in some cases, such as when showing the relationship between Year and Average Price using Bar chart. A new dataframe was created using the *dataframe.groupby()* and after this the average was calculated using the *mean()* function. When creating dataframe, we can specify which columns we want to include in the dataframe. To show the graph on screen, I used *fig.show()* function to display the graphs for the user to see. The plotly library provides functions that we can use to customize the layout of the graphs, update the x-axis and the y-axis. I used these functions to customize the graphs and their look and feel.

The assignment can be found on [this GitHub repository](https://github.com/MehadND/data-viz-4). Also some additional files such as the csv files can also be found on the repository.