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)
```

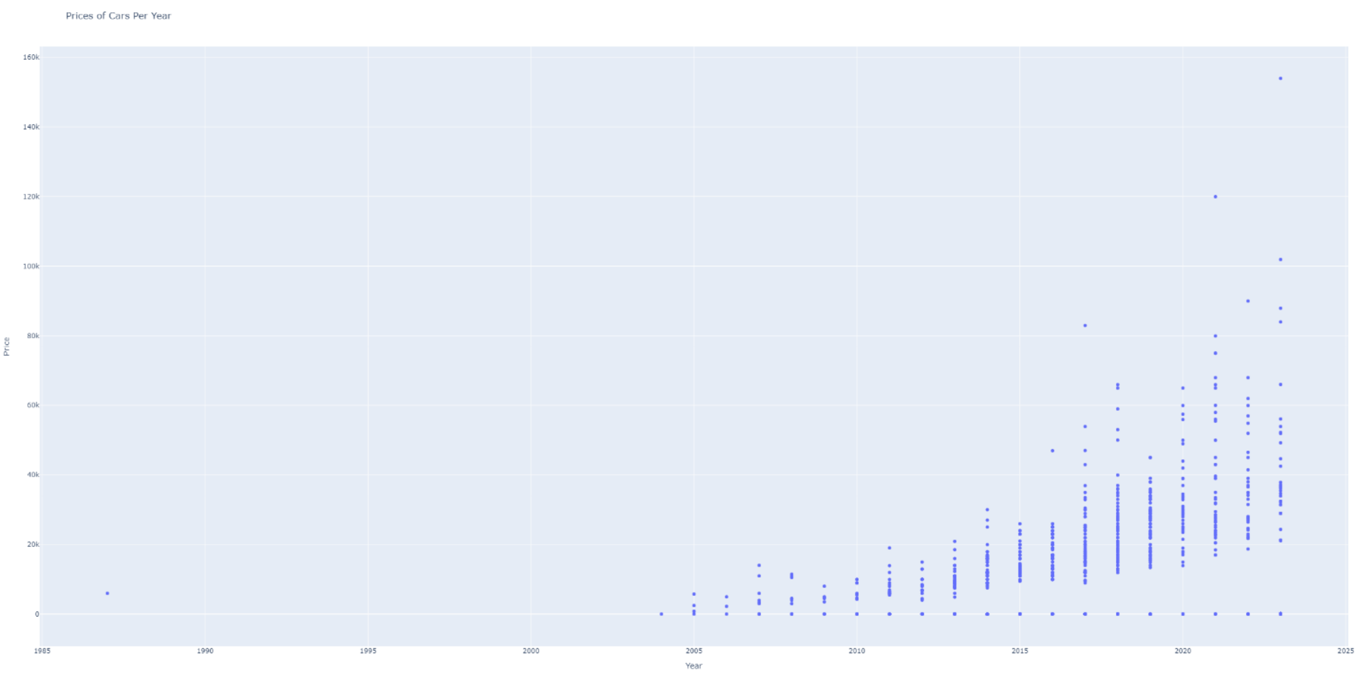
```
def tax_expiry_preproc(car_tax_expiry):
    if car_tax_expiry == "-":
        return '-'

    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
```

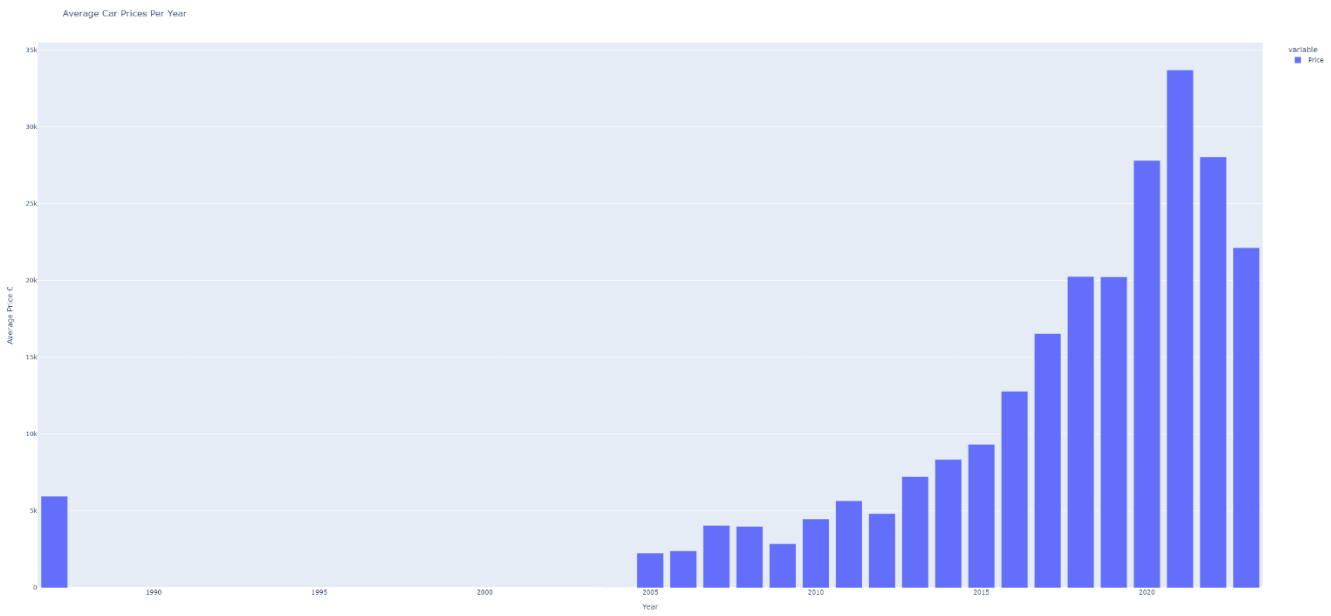
```
fig = px.line(dfp, x="Year", y="Price", color="Fuel", facet_col="Make",
facet_row="Doors", template="plotly_dark")

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()
```

2. Graphs

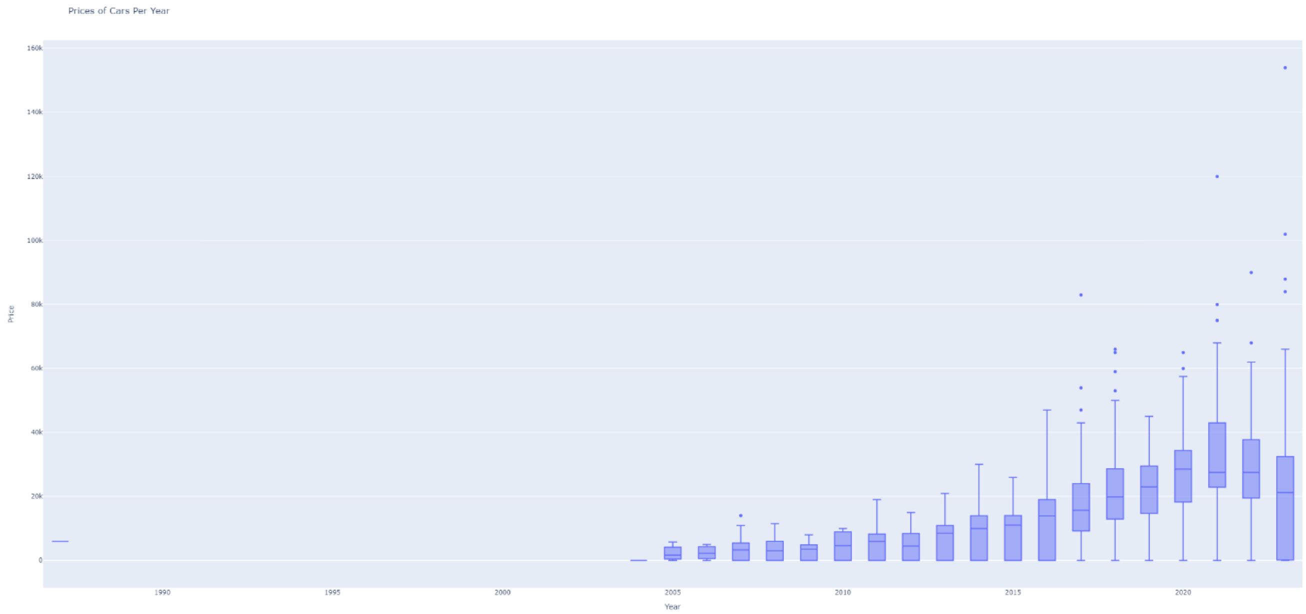
a. A Scatter Chart for Year verses Price



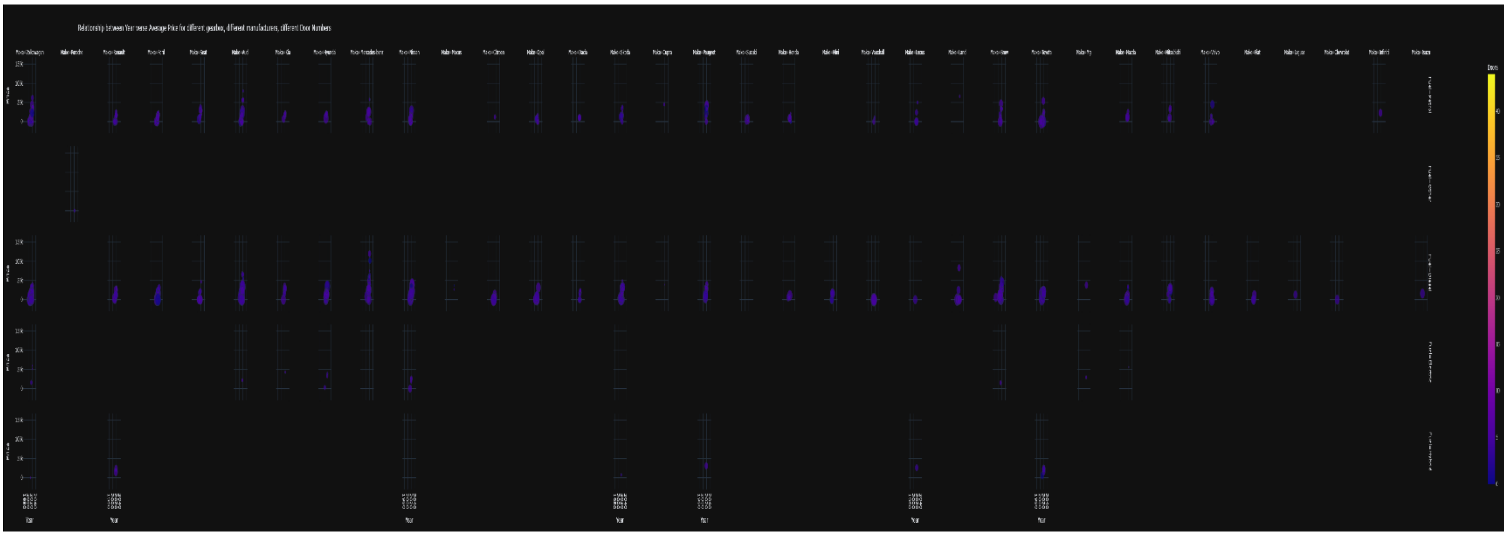
b. A Bar Chart to show the relationship between Year and Average Price

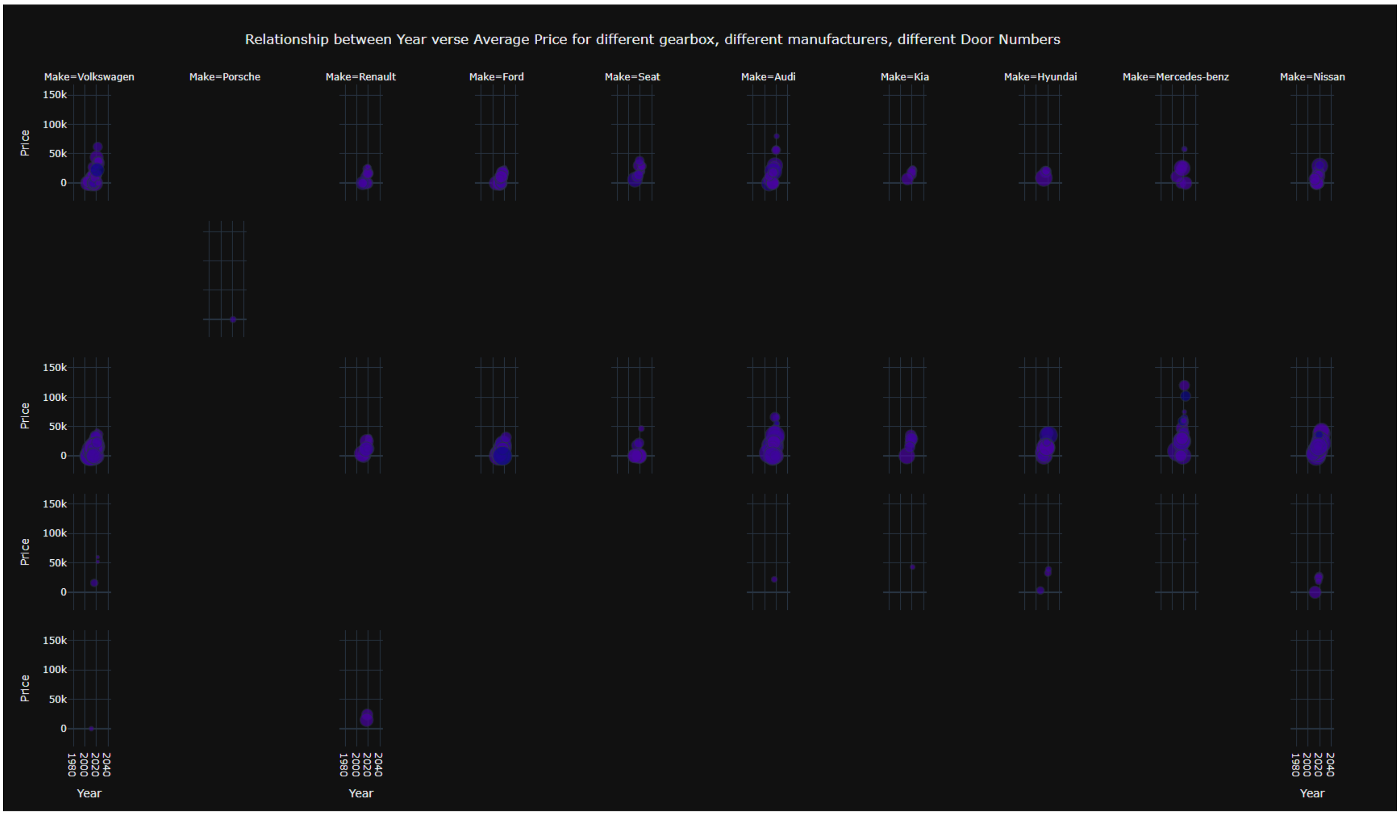


c. A box chart for Year verses Price

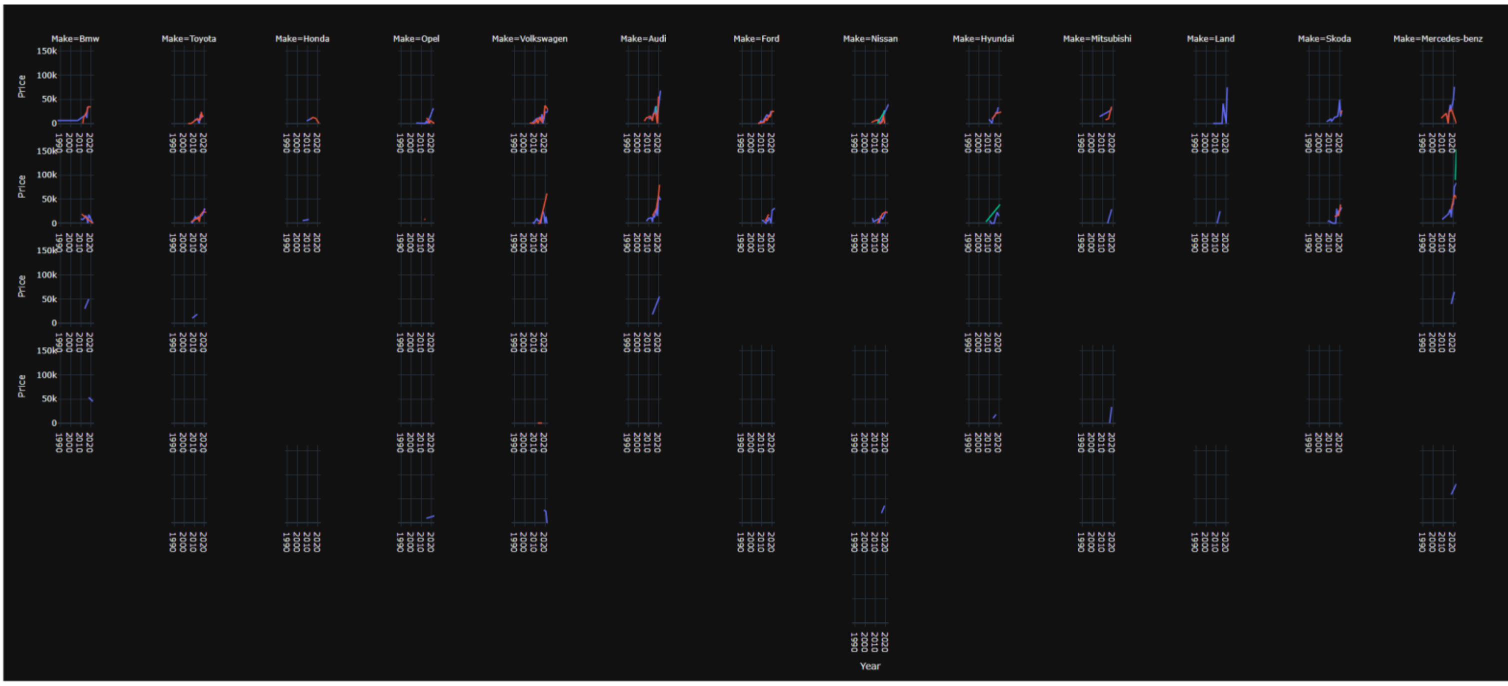


d. A Scatter Facet to show the relationship between Year verse Average Price for different auto/manual gearbox, different Manufacturers , different Door Numbers and use mileage to change the scatter marker size

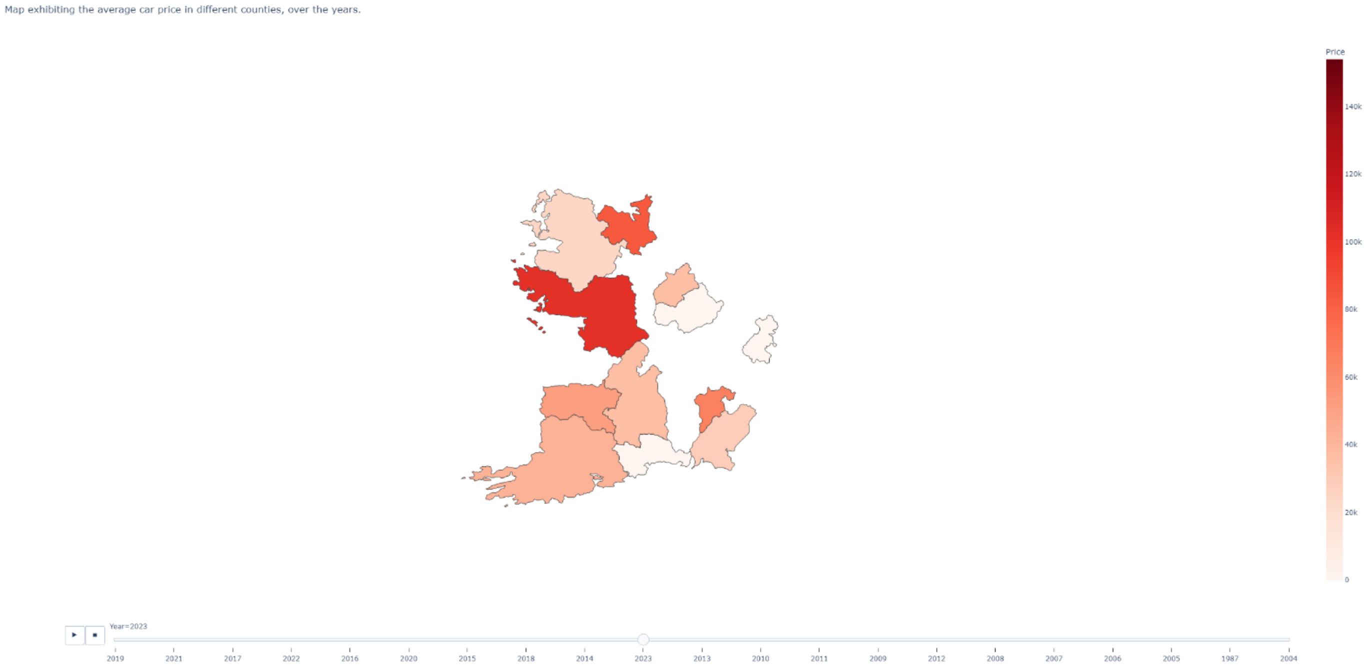




e. A Line Facet plot



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



g. Pie Facet

3. Conclusion

For this assignment, I used plotly and pandas to visualize the data that I collected from scraping a website (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. Also some additional files such as the csv files can also be found on the repository.