# EDA_Lab

May 4, 2022

## 1 Exploratory Data Analysis

Estimated time needed: **30** minutes

Exploratory Data Analysis (EDA) is the crucial process of using summary statistics and graphical representations to perform preliminary investigations on data to uncover patterns, detect anomalies, test hypotheses, and verify assumptions.

In this notebook, we will learn some interesting and useful data exploration techniques that can be applied to explore any geographical data.

### 1.1 Objectives

*After completing this lab you will be able to:*

- Do Data Wrangling
- Do Data Filtering
- Plot with plotly.express
- Produce choropleth map

---

### 1.2 Setup

For this lab, we will be using the following libraries:

- `pandas` for managing the data.
- `plotly.express` for visualizing the data.
- `json` for reading json file formats.

### 1.3 Installing Required Libraries

The following required modules are pre-installed in the Skills Network Labs environment. However, if you run this notebook commands in a different Jupyter environment (e.g. Watson Studio or Ananconda) you will need to install these libraries by removing the `#` sign before `!mamba` in the code cell below.

[ ]:
```
# All Libraries required for this lab are listed below. The libraries␣
↪pre-installed on Skills Network Labs are commented.
# !mamba install -qy pandas==1.3.4 numpy==1.21.4 seaborn==0.9.0 matplotlib==3.5.
↪0 scikit-learn==0.20.1
```

```
# Note: If your environment doesn't support "!mamba install", use "!pip install"
```

[1]:
```
import pandas as pd
import plotly.express as px
import datetime
import requests
import json
```

### 1.4 Reading and understanding our data

The dataset in this lab is Monthly average retail prices for gasoline and fuel oil, by geography . It is available through Statistics Canada and includes monthly average gasoline price (Cents per Litre), of major Canadian Cities, starting from 1979 until recent.

Another dataset, canada_provinces.geojson, contains the mapping information of all Canadian Provinces. It will be used in our analysis to produce a choropleth map.

Let's read the data into *pandas* dataframe and look at the first 5 rows using the `head()` method.

[2]:
```
gasoline = pd.read_csv("https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBM-ML0232EN-SkillsNetwork/asset/18100001.csv")
gasoline.head()
```

[2]:

| | REF_DATE | GEO | DGUID | Type of fuel | UOM | UOM_ID | SCALAR_FACTOR | SCALAR_ID | VECTOR | COORDINATE | VALUE | STATUS | SYMBOL | TERMINATED | DECIMALS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Jan-79 | St. John's, Newfoundland and Labrador | 2011S0503001 | Regular unleaded gasoline at full service fill… | Cents per litre | 57 | units | 0 | v735046 | 2.1 | 26.0 | NaN | NaN | t | 1 |
| 1 | Jan-79 | Charlottetown and Summerside, Prince Edward Is… | NaN | Regular unleaded gasoline at full service fill… | Cents per litre | 57 | units | 0 | v735056 | 3.1 | 24.6 | NaN | NaN | t | 1 |
| 2 | Jan-79 | Halifax, Nova Scotia | 2011S0503205 | Regular unleaded gasoline at full service fill… | Cents per litre | 57 | units | 0 | v735057 | 4.1 | 23.4 | NaN | NaN | t | 1 |
| 3 | Jan-79 | Saint John, New Brunswick | 2011S0503310 | Regular unleaded gasoline at full service fill… | Cents per litre | 57 | units | 0 | v735058 | 5.1 | 23.2 | NaN | NaN | t | 1 |
| 4 | Jan-79 | Québec, Quebec | 2011S0503421 | Regular unleaded gasoline at full service fill… | Cents per litre | 57 | units | 0 | v735059 | 6.1 | 22.6 | NaN | NaN | t | 1 |

```
3          t          1
4          t          1
```

Let's find out how many entries there are in our dataset, using `shape` function.

[3]:
```
gasoline.shape
```

[3]:
```
(41942, 15)
```

Using `info` function, we will take a look at our types of data.

[4]:
```
gasoline.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41942 entries, 0 to 41941
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   REF_DATE        41942 non-null  object 
 1   GEO             41942 non-null  object 
 2   DGUID           39451 non-null  object 
 3   Type of fuel    41942 non-null  object 
 4   UOM             41942 non-null  object 
 5   UOM_ID          41942 non-null  int64  
 6   SCALAR_FACTOR   41942 non-null  object 
 7   SCALAR_ID       41942 non-null  int64  
 8   VECTOR          41942 non-null  object 
 9   COORDINATE      41942 non-null  float64
 10  VALUE           41942 non-null  float64
 11  STATUS          0 non-null      float64
 12  SYMBOL          0 non-null      float64
 13  TERMINATED      16564 non-null  object 
 14  DECIMALS        41942 non-null  int64  
dtypes: float64(4), int64(3), object(8)
memory usage: 4.8+ MB
```

Using `columns` method, we will print all the column names.

[5]:
```
gasoline.columns
```

[5]:
```
Index(['REF_DATE', 'GEO', 'DGUID', 'Type of fuel', 'UOM', 'UOM_ID',
       'SCALAR_FACTOR', 'SCALAR_ID', 'VECTOR', 'COORDINATE', 'VALUE', 'STATUS',
       'SYMBOL', 'TERMINATED', 'DECIMALS'],
      dtype='object')
```

Below, we will check for any missing values.

[6]:
```
gasoline.isnull().sum()
```

[6]:
```
REF_DATE             0
GEO                  0
DGUID             2491
Type of fuel         0
UOM                  0
UOM_ID               0
SCALAR_FACTOR        0
SCALAR_ID            0
VECTOR               0
COORDINATE           0
VALUE                0
STATUS           41942
SYMBOL           41942
TERMINATED       25378
DECIMALS             0
dtype: int64
```

## 1.5 Data Wrangling

### 1.5.1 Selecting and renaming the columns of interest

Below, we are filtering our data, by selecting only the relevant columns. Also, we are using the `rename()` method to change the name of the columns.

[7]:
```
data = (gasoline[['REF_DATE','GEO','Type of fuel','VALUE']]).
 ↪rename(columns={"REF_DATE" : "DATE", "Type of fuel" : "TYPE"})
data.head()
```

[7]:

| | DATE | GEO | TYPE | VALUE |
|---|---|---|---|---|
| 0 | Jan-79 | St. John's, Newfoundland and Labrador | Regular unleaded gasoline at full service fill… | 26.0 |
| 1 | Jan-79 | Charlottetown and Summerside, Prince Edward Is… | Regular unleaded gasoline at full service fill… | 24.6 |
| 2 | Jan-79 | Halifax, Nova Scotia | Regular unleaded gasoline at full service fill… | 23.4 |
| 3 | Jan-79 | Saint John, New Brunswick | Regular unleaded gasoline at full service fill… | 23.2 |
| 4 | Jan-79 | Québec, Quebec | Regular unleaded gasoline at full service fill… | 22.6 |

### 1.5.2 Splitting the columns

The `str.split()` function splits the string records, by a 'comma', with `n=1` split, and Expend=True , returns a dataframe. Below, we are splitting 'GEO' into 'City' and 'Province'.

[8]:
```
data[['City', 'Province']] = data['GEO'].str.split(',', n=1, expand=True)
```

[9]:
```
data.head()
```

[9]:
```
      DATE                                                GEO  \
0  Jan-79              St. John's, Newfoundland and Labrador
1  Jan-79  Charlottetown and Summerside, Prince Edward Is…
2  Jan-79                                 Halifax, Nova Scotia
3  Jan-79                            Saint John, New Brunswick
4  Jan-79                                       Québec, Quebec

                                              TYPE  VALUE  \
0  Regular unleaded gasoline at full service fill…   26.0
1  Regular unleaded gasoline at full service fill…   24.6
2  Regular unleaded gasoline at full service fill…   23.4
3  Regular unleaded gasoline at full service fill…   23.2
4  Regular unleaded gasoline at full service fill…   22.6

                           City                    Province
0                    St. John's   Newfoundland and Labrador
1  Charlottetown and Summerside         Prince Edward Island
2                       Halifax                  Nova Scotia
3                    Saint John                New Brunswick
4                        Québec                       Quebec
```

### 1.5.3 Changing to *datetime* format

If we scroll up to our `gasoline.info()` section, we can find that 'REF_DATE' is an object type. To be able to filter by day, month, or year, we need to change the format from object type to *datetime*. Pandas function `to_datetime()` transforms to date time format. Also, we need to specify the format of *datetime* that we need. In our case, `format='%b-%y'` means that it will split into the name of a month and year. `str.slice(stop=3)` splits and outputs the first 3 letters of a month. For more information on how to transform to *datetime*, please visit this pandas documentation. Also, this web page contains more information on *datetime* formats.

[10]:
```
data['DATE'] = pd.to_datetime(data['DATE'], format='%b-%y')
data['Month'] = data['DATE'].dt.month_name().str.slice(stop=3)
data['Year'] = data['DATE'].dt.year
```

[11]:
```
data.head()
```

[11]:
```
        DATE                                                GEO  \
0 1979-01-01              St. John's, Newfoundland and Labrador
1 1979-01-01  Charlottetown and Summerside, Prince Edward Is…
2 1979-01-01                                 Halifax, Nova Scotia
3 1979-01-01                            Saint John, New Brunswick
4 1979-01-01                                       Québec, Quebec

                                              TYPE  VALUE  \
```

```
0  Regular unleaded gasoline at full service fill…   26.0
1  Regular unleaded gasoline at full service fill…   24.6
2  Regular unleaded gasoline at full service fill…   23.4
3  Regular unleaded gasoline at full service fill…   23.2
4  Regular unleaded gasoline at full service fill…   22.6
```

| | City | Province | Month | Year |
|---|---|---|---|---|
| 0 | St. John's | Newfoundland and Labrador | Jan | 1979 |
| 1 | Charlottetown and Summerside | Prince Edward Island | Jan | 1979 |
| 2 | Halifax | Nova Scotia | Jan | 1979 |
| 3 | Saint John | New Brunswick | Jan | 1979 |
| 4 | Québec | Quebec | Jan | 1979 |

The `describe()` function provides statistical information about the numeric variables. Since we only have the 'VALUE' variable that we want statistical information on, we will filter it by `data.VALUE.describe()` function.

[12]:
```
data.VALUE.describe()
# can also use  data['VALUE'].describe()
```

[12]:
```
count    41942.000000
mean        84.784858
std         31.492697
min         18.300000
25%         58.200000
50%         79.200000
75%        110.900000
max        191.600000
Name: VALUE, dtype: float64
```

Now, it is useful to know what is inside our categorical variables. We will use `unique().tolist()` functions to print out all of our 'GEO' colunm.

[13]:
```
data.GEO.unique().tolist()
# can also use  data['GEO'].unique().tolist()
```

[13]:
```
["St. John's, Newfoundland and Labrador",
 'Charlottetown and Summerside, Prince Edward Island',
 'Halifax, Nova Scotia',
 'Saint John, New Brunswick',
 'Québec, Quebec',
 'Montréal, Quebec',
 'Ottawa-Gatineau, Ontario part, Ontario/Quebec',
 'Toronto, Ontario',
 'Thunder Bay, Ontario',
 'Winnipeg, Manitoba',
 'Regina, Saskatchewan',
 'Saskatoon, Saskatchewan',
```

```
 'Edmonton, Alberta',
 'Calgary, Alberta',
 'Vancouver, British Columbia',
 'Victoria, British Columbia',
 'Whitehorse, Yukon',
 'Yellowknife, Northwest Territories']
```

### 1.6 Exercise 1

In this exercise, print out all categories in 'TYPE' column.

[15]:
```
# Enter your code and run the cell
data['TYPE'].unique().tolist()
```

[15]:
```
['Regular unleaded gasoline at full service filling stations',
 'Regular unleaded gasoline at self service filling stations',
 'Premium unleaded gasoline at full service filling stations',
 'Premium unleaded gasoline at self service filling stations',
 'Diesel fuel at full service filling stations',
 'Household heating fuel',
 'Diesel fuel at self service filling stations']
```

Solution (Click Here)

    <code>

data.TYPE.unique().tolist()

### 1.7 Data Filtering

This section will introduce you to some of the most common filtering techniques when working with pandas dataframes.

#### 1.7.1 Filtering with logical operators

We can use the logical operators on column values to filter rows. First, we specify the name of our dataFrame, then, square brackets to select the name of the column, double 'equal' sign, '==' to select the name of a row group, in single or double quotation marks. If we want to exclude some entries (e.g. some locations), we would use the 'equal' and 'exclamation point' signs together, '=!'. We can also use '</>', '<=/>=' signs to select numeric information.

Let's select the Calgary, Alberta data to see all the information.

[16]:
```
calgary = data[data['GEO'] == 'Calgary, Alberta']
calgary
```

[16]:
```
            DATE               GEO  \
13    1979-01-01  Calgary, Alberta
28    1979-02-01  Calgary, Alberta
43    1979-03-01  Calgary, Alberta
```

```
58    1979-04-01  Calgary, Alberta
73    1979-05-01  Calgary, Alberta
…          …                 …
41855 2021-09-01  Calgary, Alberta
41856 2021-09-01  Calgary, Alberta
41923 2021-10-01  Calgary, Alberta
41924 2021-10-01  Calgary, Alberta
41925 2021-10-01  Calgary, Alberta

                                                      TYPE  VALUE     City  \
13     Regular unleaded gasoline at full service fill…   18.7  Calgary
28     Regular unleaded gasoline at full service fill…   18.9  Calgary
43     Regular unleaded gasoline at full service fill…   18.9  Calgary
58     Regular unleaded gasoline at full service fill…   19.1  Calgary
73     Regular unleaded gasoline at full service fill…   19.2  Calgary
…                                                        …    …       …
41855  Premium unleaded gasoline at self service fill…  156.6  Calgary
41856       Diesel fuel at self service filling stations  125.1  Calgary
41923  Regular unleaded gasoline at self service fill…  140.8  Calgary
41924  Premium unleaded gasoline at self service fill…  164.4  Calgary
41925       Diesel fuel at self service filling stations  138.3  Calgary

       Province Month  Year
13      Alberta   Jan  1979
28      Alberta   Feb  1979
43      Alberta   Mar  1979
58      Alberta   Apr  1979
73      Alberta   May  1979
…         …   …   …
41855   Alberta   Sep  2021
41856   Alberta   Sep  2021
41923   Alberta   Oct  2021
41924   Alberta   Oct  2021
41925   Alberta   Oct  2021

[2109 rows x 8 columns]
```

Now, let's select year 2000.

[17]:
```
sel_years = data[data['Year'] ==  2000]
sel_years
```

[17]:
```
            DATE                                    GEO  \
16168 2000-01-01  St. John's, Newfoundland and Labrador
16169 2000-01-01  St. John's, Newfoundland and Labrador
16170 2000-01-01  St. John's, Newfoundland and Labrador
16171 2000-01-01  St. John's, Newfoundland and Labrador
```

```
16172 2000-01-01  St. John's, Newfoundland and Labrador
…          …                                       …
17579 2000-12-01     Yellowknife, Northwest Territories
17580 2000-12-01     Yellowknife, Northwest Territories
17581 2000-12-01     Yellowknife, Northwest Territories
17582 2000-12-01     Yellowknife, Northwest Territories
17583 2000-12-01     Yellowknife, Northwest Territories

                                                    TYPE  VALUE         City  \
16168  Regular unleaded gasoline at full service fill…   78.0   St. John's
16169  Regular unleaded gasoline at self service fill…   74.9   St. John's
16170  Premium unleaded gasoline at full service fill…   84.5   St. John's
16171  Premium unleaded gasoline at self service fill…   81.3   St. John's
16172       Diesel fuel at full service filling stations   69.2   St. John's
…                                                     …    …           …
17579  Premium unleaded gasoline at full service fill…   92.6  Yellowknife
17580  Premium unleaded gasoline at self service fill…   95.4  Yellowknife
17581       Diesel fuel at full service filling stations   81.9  Yellowknife
17582       Diesel fuel at self service filling stations   78.9  Yellowknife
17583                           Household heating fuel   58.8  Yellowknife

                         Province Month  Year
16168   Newfoundland and Labrador   Jan  2000
16169   Newfoundland and Labrador   Jan  2000
16170   Newfoundland and Labrador   Jan  2000
16171   Newfoundland and Labrador   Jan  2000
16172   Newfoundland and Labrador   Jan  2000
…                             …   …    …
17579       Northwest Territories   Dec  2000
17580       Northwest Territories   Dec  2000
17581       Northwest Territories   Dec  2000
17582       Northwest Territories   Dec  2000
17583       Northwest Territories   Dec  2000

[1416 rows x 8 columns]
```

### 1.7.2 Filtering by multiple conditions

There are many alternative ways to perform filtering in pandas. We can also use '|' ('or') and '&' (and) to select multiple columns and rows.

For example, let us select Toronto and Edmonton locations.

[18]:
```
mult_loc = data[(data['GEO'] == "Toronto, Ontario") | (data['GEO'] ==␣
 ↪"Edmonton, Alberta")]
mult_loc
```

[18]:
```
            DATE                GEO  \
7     1979-01-01   Toronto, Ontario
12    1979-01-01  Edmonton, Alberta
22    1979-02-01   Toronto, Ontario
27    1979-02-01  Edmonton, Alberta
37    1979-03-01   Toronto, Ontario
…            …                 …
41903 2021-10-01   Toronto, Ontario
41904 2021-10-01   Toronto, Ontario
41920 2021-10-01  Edmonton, Alberta
41921 2021-10-01  Edmonton, Alberta
41922 2021-10-01  Edmonton, Alberta

                                                    TYPE  VALUE      City  \
7      Regular unleaded gasoline at full service fill…   23.0   Toronto
12     Regular unleaded gasoline at full service fill…   18.3  Edmonton
22     Regular unleaded gasoline at full service fill…   23.2   Toronto
27     Regular unleaded gasoline at full service fill…   18.5  Edmonton
37     Regular unleaded gasoline at full service fill…   23.2   Toronto
…                                                    …    …       …
41903       Diesel fuel at self service filling stations  141.3   Toronto
41904                             Household heating fuel  148.0   Toronto
41920  Regular unleaded gasoline at self service fill…  138.3  Edmonton
41921  Premium unleaded gasoline at self service fill…  159.6  Edmonton
41922       Diesel fuel at self service filling stations  134.7  Edmonton

       Province Month  Year
7       Ontario   Jan  1979
12      Alberta   Jan  1979
22      Ontario   Feb  1979
27      Alberta   Feb  1979
37      Ontario   Mar  1979
…         …   …   …
41903   Ontario   Oct  2021
41904   Ontario   Oct  2021
41920   Alberta   Oct  2021
41921   Alberta   Oct  2021
41922   Alberta   Oct  2021

[4600 rows x 8 columns]
```

Alternatively, we can use `isin` method to select multiple locations.

[19]:
```
cities = ['Calgary', 'Toronto', 'Edmonton']
CTE = data[data.City.isin(cities)]
CTE
```

[19]:

```
            DATE                GEO  \
7     1979-01-01   Toronto, Ontario
12    1979-01-01  Edmonton, Alberta
13    1979-01-01   Calgary, Alberta
22    1979-02-01   Toronto, Ontario
27    1979-02-01  Edmonton, Alberta
…            …                 …
41921 2021-10-01  Edmonton, Alberta
41922 2021-10-01  Edmonton, Alberta
41923 2021-10-01   Calgary, Alberta
41924 2021-10-01   Calgary, Alberta
41925 2021-10-01   Calgary, Alberta

                                                      TYPE  VALUE      City  \
7      Regular unleaded gasoline at full service fill…   23.0   Toronto
12     Regular unleaded gasoline at full service fill…   18.3  Edmonton
13     Regular unleaded gasoline at full service fill…   18.7   Calgary
22     Regular unleaded gasoline at full service fill…   23.2   Toronto
27     Regular unleaded gasoline at full service fill…   18.5  Edmonton
…                                                      …    …        …
41921  Premium unleaded gasoline at self service fill…  159.6  Edmonton
41922       Diesel fuel at self service filling stations  134.7  Edmonton
41923  Regular unleaded gasoline at self service fill…  140.8   Calgary
41924  Premium unleaded gasoline at self service fill…  164.4   Calgary
41925       Diesel fuel at self service filling stations  138.3   Calgary

       Province Month  Year
7       Ontario   Jan  1979
12      Alberta   Jan  1979
13      Alberta   Jan  1979
22      Ontario   Feb  1979
27      Alberta   Feb  1979
…         …   …   …
41921   Alberta   Oct  2021
41922   Alberta   Oct  2021
41923   Alberta   Oct  2021
41924   Alberta   Oct  2021
41925   Alberta   Oct  2021

[6709 rows x 8 columns]
```

### 1.8 Exercise 2 a

In this exercise, please use the examples shown above, to select the data that shows the price of the 'household heating fuel', in Vancouver, in 1990.

[24]:
```
# Enter your code below and run the cell
van_price = data[(data['City'] == 'Vancouver') & (data['Year'] == 1990) &␣
 ↪(data['TYPE'] == 'Household heating fuel')]
van_price
```

[24]:

| | DATE | GEO | TYPE | VALUE | \ |
|---|---|---|---|---|---|
| 2192 | 1990-01-01 | Vancouver, British Columbia | Household heating fuel | 32.4 | |
| 2304 | 1990-02-01 | Vancouver, British Columbia | Household heating fuel | 33.7 | |
| 2416 | 1990-03-01 | Vancouver, British Columbia | Household heating fuel | 34.0 | |
| 2528 | 1990-04-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | |
| 2640 | 1990-05-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | |
| 2752 | 1990-06-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | |
| 2864 | 1990-07-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | |
| 2976 | 1990-08-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | |
| 3088 | 1990-09-01 | Vancouver, British Columbia | Household heating fuel | 36.7 | |
| 3200 | 1990-10-01 | Vancouver, British Columbia | Household heating fuel | 41.8 | |
| 3312 | 1990-11-01 | Vancouver, British Columbia | Household heating fuel | 42.7 | |
| 3424 | 1990-12-01 | Vancouver, British Columbia | Household heating fuel | 45.7 | |

| | City | Province | Month | Year |
|---|---|---|---|---|
| 2192 | Vancouver | British Columbia | Jan | 1990 |
| 2304 | Vancouver | British Columbia | Feb | 1990 |
| 2416 | Vancouver | British Columbia | Mar | 1990 |
| 2528 | Vancouver | British Columbia | Apr | 1990 |
| 2640 | Vancouver | British Columbia | May | 1990 |
| 2752 | Vancouver | British Columbia | Jun | 1990 |
| 2864 | Vancouver | British Columbia | Jul | 1990 |
| 2976 | Vancouver | British Columbia | Aug | 1990 |
| 3088 | Vancouver | British Columbia | Sep | 1990 |
| 3200 | Vancouver | British Columbia | Oct | 1990 |
| 3312 | Vancouver | British Columbia | Nov | 1990 |
| 3424 | Vancouver | British Columbia | Dec | 1990 |

Solution (Click Here)

    <code>

exercise2a = data[( data[‘Year’] == 1990) & (data[‘TYPE’] == “Household heating fuel”) & (data[‘City’]==‘Vancouver’)] exercise2a

### 1.9 Exercise 2 b

In this exercise, please select the data that shows the price of the ‘household heating fuel’, in Vancouver, in the years of 1979 and 2021.

[26]:
```
# Enter your code below and run the cell
ex_2b = data[(data['City'] == 'Vancouver') & (data['TYPE'] == 'Household␣
 ↪heating fuel') & (data['Year'] == 1990) | (data['Year'] == 2021)]
ex_2b
```

[26]:

| | DATE | GEO | TYPE | VALUE | City | Province | Month | Year |
|---|---|---|---|---|---|---|---|---|
| 2192 | 1990-01-01 | Vancouver, British Columbia | Household heating fuel | 32.4 | Vancouver | British Columbia | Jan | 1990 |
| 2304 | 1990-02-01 | Vancouver, British Columbia | Household heating fuel | 33.7 | Vancouver | British Columbia | Feb | 1990 |
| 2416 | 1990-03-01 | Vancouver, British Columbia | Household heating fuel | 34.0 | Vancouver | British Columbia | Mar | 1990 |
| 2528 | 1990-04-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | Vancouver | British Columbia | Apr | 1990 |
| 2640 | 1990-05-01 | Vancouver, British Columbia | Household heating fuel | 34.5 | Vancouver | British Columbia | May | 1990 |
| … | … | … | … | … | … | … | … | … |
| 41937 | 2021-10-01 | Whitehorse, Yukon | Household heating fuel | 140.6 | Whitehorse | Yukon | Oct | 2021 |
| 41938 | 2021-10-01 | Yellowknife, Northwest Territories | Regular unleaded gasoline at self service fill… | 150.6 | Yellowknife | Northwest Territories | Oct | 2021 |
| 41939 | 2021-10-01 | Yellowknife, Northwest Territories | Premium unleaded gasoline at self service fill… | 166.1 | Yellowknife | Northwest Territories | Oct | 2021 |
| 41940 | 2021-10-01 | Yellowknife, Northwest Territories | Diesel fuel at self service filling stations | 149.8 | Yellowknife | Northwest Territories | Oct | 2021 |
| 41941 | 2021-10-01 | Yellowknife, Northwest Territories | Household heating fuel | 130.7 | Yellowknife | Northwest Territories | Oct | 2021 |

[702 rows x 8 columns]

Solution (Click Here)

    <code>

exercise2b = data[( data[‘Year’] <= 1979) | ( data[‘Year’] == 2021) & (data[‘TYPE’] == “Household heating fuel”) & (data[‘City’]==‘Vancouver’)] exercise2b

Hint (Click Here)

    <code>

If we use '&' operator between the two years, it will return an empty data frame. This is because there was no data for the 'household heating fuel, in Vancouver, in 1979. Using 'or' operator is suitable because either one of two years that contains any information on 'household heating fuel' in Vancouver.

### 1.9.1 Filtering using `groupby()` method

The role of `groupby()` is to analyze data by some categories. The simplest call is by a column name. For example, let's use the 'GEO' column and `ngroups` function to calculate the number of groups (cities, provinces) in 'GEO' column.

[27]:
```
geo = data.groupby('GEO')
geo.ngroups
```

[27]:
```
18
```

Most commonly, we use `groupby()` to split the data into groups,this will apply some function to each of the groups (e.g. mean, median, min, max, count), then combine the results into a data structure. For example, let's select the 'VALUE' column and calculate the mean of the gasoline prices per year. First, we specify the 'Year" column, following by the 'VALUE' column, and the `mean()` function.

[28]:
```
group_year = data.groupby(['Year'])['VALUE'].mean()
group_year
```

[28]:
```
Year
1979     23.604444
1980     28.068750
1981     38.002604
1982     44.701563
1983     47.904688
1984     50.442708
1985     53.899479
1986     48.405208
1987     49.758333
1988     49.217188
1989     51.700000
1990     55.048735
1991     56.527041
1992     54.633832
1993     54.334734
1994     54.247899
1995     56.177451
1996     58.134110
1997     59.182062
1998     56.247246
1999     58.743362
```

```
2000     72.207839
2001     72.403107
2002     70.312147
2003     75.541667
2004     82.960452
2005     96.328743
2006    101.209393
2007    105.258263
2008    123.340678
2009     96.969068
2010    106.369845
2011    126.790607
2012    130.380085
2013    129.677273
2014    133.169203
2015    110.366908
2016    101.790821
2017    112.852657
2018    129.408575
2019    125.776329
2020    107.617150
2021    133.990580
Name: VALUE, dtype: float64
```

### 1.10 Exercise 3 a

In the cell below, please use `groupby()` method to group by the maximum value of gasoline prices, for each month.

[29]:
```
# Enter your code below and run the cell
exercise3a = data.groupby(['Month'])['VALUE'].max()
```

[30]:
```
exercise3a
```

[30]:
```
Month
Apr    187.8
Aug    188.3
Dec    158.5
Feb    168.0
Jan    162.0
Jul    191.6
Jun    183.2
Mar    171.8
May    189.3
Nov    162.2
Oct    184.9
Sep    179.3
```

```
Name: VALUE, dtype: float64
```

Solution (Click Here)

    <code>

exercise3a = data.groupby(['Month'])['VALUE'].max()

### 1.11 Exercise 3 b

In the cell below, please use `groupby()` method to group by the median value of gasoline prices, for each year and each city.

[31]:
```
# Enter your code below and run the cell
ex_3b = data.groupby(['Year', 'City'])['VALUE'].median()
ex_3b
```

[31]:
```
Year  City
1979  Calgary                          19.15
      Charlottetown and Summerside     25.45
      Edmonton                         18.70
      Halifax                          24.00
      Montréal                         23.25
                                       ...
2021  Vancouver                       151.10
      Victoria                        148.80
      Whitehorse                      142.25
      Winnipeg                        127.70
      Yellowknife                     138.65
Name: VALUE, Length: 751, dtype: float64
```

Solution (Click Here)

    <code>

exercise3b = data.groupby(['Year', 'City'])['VALUE'].median()

Hint (Click Here)

    <code>

We can also reset the index of the new data output, by using `reset_index()`, and round up the output values to 2 decimal places.

exercise3b = data.groupby(['Year', 'City'])['VALUE'].median().reset_index(name ='Value').round(2)

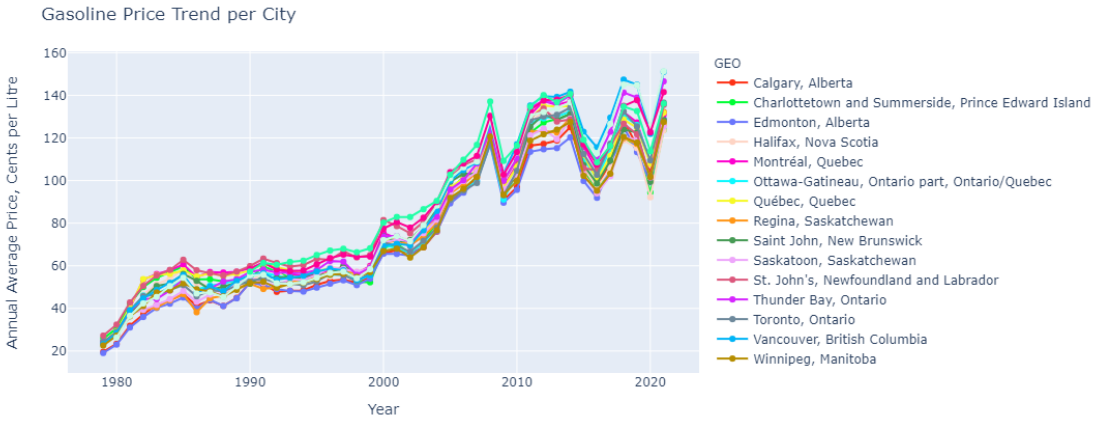
### 1.12 Visualizing the data with *pandas* plotly.express

The *plotly.express* library (usually imported as px) contains functions that can create entire figures at once. *plotly.express* is a built-in part of the *plotly* library, and makes creation of most common figures very easy. For more information on *plotly.express*, please refer to this documentation.

Here, we will plot the prices of gasoline in all cities during 1979 - 2021.

[32]:
```
price_bycity = data.groupby(['Year', 'GEO'])['VALUE'].mean().reset_index(name␣
↪='Value').round(2)
```

[34]:
```
fig = px.line(price_bycity
                    ,x='Year', y = "Value",
                    color = "GEO", color_discrete_sequence=px.colors.qualitative.
↪Light24)
fig.update_traces(mode='markers+lines')
fig.update_layout(
    title="Gasoline Price Trend per City",
    xaxis_title="Year",
    yaxis_title="Annual Average Price, Cents per Litre")
fig.show()
```




Here, we will plot the average monthly prices of gasoline in Toronto for the year of 2021.

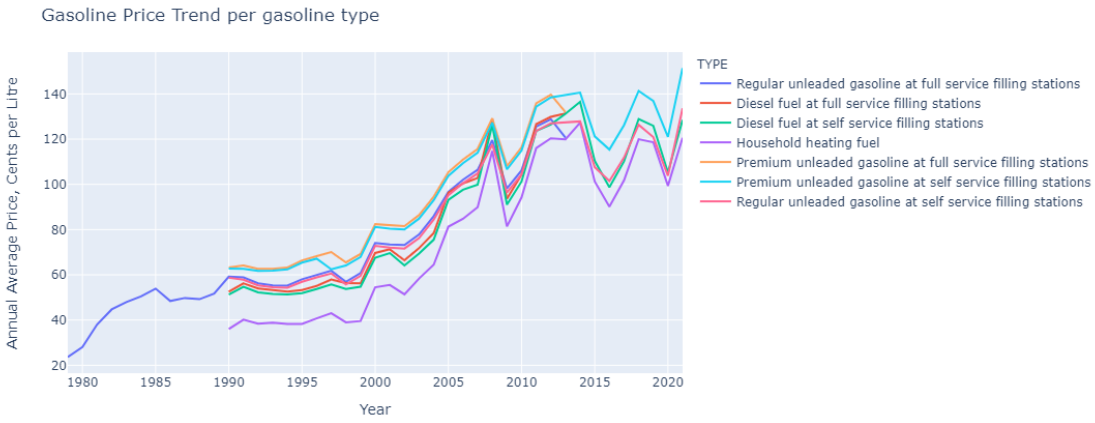
[35]:
```
mon_trend = data[(data['Year'] ==  2021) & (data['GEO'] == "Toronto, Ontario")]
group_month = mon_trend.groupby(['Month'])['VALUE'].mean().reset_index().
↪sort_values(by="VALUE")
```

[36]:
```
fig = px.line(group_month,
                    x='Month', y = "VALUE")
fig.update_traces(mode='markers+lines')
fig.update_layout(
    title="Toronto Average Monthly Gasoline Price in 2021",
    xaxis_title="Month",
    yaxis_title="Monthly Price, Cents per Litre")
fig.show()
```


## 1.13 Exercise 4

In the cell below, use *plotly.express* or other libraries, to plot the annual average gasoline price, per year, per gasoline type.

[40]:
```
# Enter your code below and run the cell
gas_df = data.groupby(['Year', 'TYPE'])['VALUE'].mean().reset_index()
fig = px.line(gas_df, x = 'Year' , y= 'VALUE', color = 'TYPE')
fig.update_layout(
    title="Gasoline Price Trend per gasoline type",
    xaxis_title="Year",
    yaxis_title="Annual Average Price, Cents per Litre")
fig.show()
```




Solution (Click Here)

    <code>

type_gas = data.groupby([‘Year’, ‘TYPE’])[‘VALUE’].mean().reset_index(name =‘Type’).round(2) fig = px.line(type_gas, x=‘Year’, y = “Type”, color = “TYPE”, color_discrete_sequence=px.colors.qualitative.Light24) fig.update_traces(mode=‘markers+lines’) fig.update_layout( title=“Fuel Type Price Trend”, xaxis_title=“Year”, yaxis_title=“Annual Average Price, Cents per Litre”) fig.show()

We can also use the animated time frame to show the trend of gasoline prices over time.

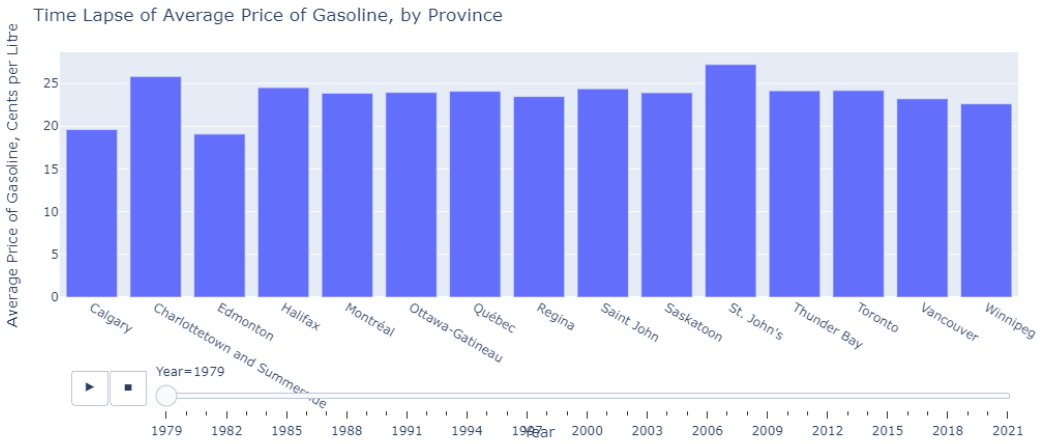
[41]:
```
bycity = data.groupby(['Year', 'City'])['VALUE'].mean().reset_index(name␣
↪='Value').round(2)
bycity.head()
```

[41]:

| | Year | City | Value |
|---|---|---|---|
| 0 | 1979 | Calgary | 19.61 |
| 1 | 1979 | Charlottetown and Summerside | 25.82 |
| 2 | 1979 | Edmonton | 19.08 |
| 3 | 1979 | Halifax | 24.52 |
| 4 | 1979 | Montréal | 23.86 |

[42]:
```
fig = px.bar(bycity,
             x='City', y = "Value", animation_frame="Year")
fig.update_layout(
    title="Time Lapse of Average Price of Gasoline, by Province",
    xaxis_title="Year",
    yaxis_title="Average Price of Gasoline, Cents per Litre")

fig.show()
```




Another way to display the distribution of average gasoline prices in Canadian Provinces is by plotting a map. We will use 2021 year to display the average gasoline price in all Canadian

Provinces. First, we select the year.

[43]:
```
one_year = data[data['Year'] == 2021]
one_year.head()
```

[43]:
```
            DATE                                                GEO  \
41252 2021-01-01                St. John's, Newfoundland and Labrador
41253 2021-01-01                St. John's, Newfoundland and Labrador
41254 2021-01-01                St. John's, Newfoundland and Labrador
41255 2021-01-01                St. John's, Newfoundland and Labrador
41256 2021-01-01  Charlottetown and Summerside, Prince Edward Is…

                                                    TYPE  VALUE  \
41252  Regular unleaded gasoline at self service fill…  124.8
41253  Premium unleaded gasoline at self service fill…  130.6
41254       Diesel fuel at self service filling stations  126.7
41255                             Household heating fuel   89.8
41256  Regular unleaded gasoline at self service fill…  109.1

                              City                    Province Month  Year
41252                   St. John's   Newfoundland and Labrador   Jan  2021
41253                   St. John's   Newfoundland and Labrador   Jan  2021
41254                   St. John's   Newfoundland and Labrador   Jan  2021
41255                   St. John's   Newfoundland and Labrador   Jan  2021
41256  Charlottetown and Summerside        Prince Edward Island   Jan  2021
```

Then, we group by the 'Province' and the 'mean' values of gasoline prices per each province. We also need to index each province with province id.

[44]:
```
geodata =  one_year.groupby('Province')['VALUE'].mean().reset_index(name␣
 ↪='Average Gasoline Price').round(2)

provinces={' Newfoundland and Labrador':5,
 ' Prince Edward Island':8,
 ' Nova Scotia':2,
 ' New Brunswick':7,
 ' Quebec':1,
 ' Ontario':11,
 ' Ontario part, Ontario/Quebec':12,
 ' Manitoba':10,
 ' Saskatchewan':3,
 ' Alberta':4,
 ' British Columbia':6,
 ' Yukon':9,
 ' Northwest Territories':13
}
geodata['ProvinceID']=geodata['Province'].map(provinces)
display(geodata)
```

| | Province | Average Gasoline Price | ProvinceID |
|---|---|---|---|
| 0 | Alberta | 130.48 | 4 |
| 1 | British Columbia | 151.17 | 6 |
| 2 | Manitoba | 127.48 | 10 |
| 3 | New Brunswick | 128.35 | 7 |
| 4 | Newfoundland and Labrador | 135.54 | 5 |
| 5 | Northwest Territories | 136.13 | 13 |
| 6 | Nova Scotia | 123.54 | 2 |
| 7 | Ontario | 140.85 | 11 |
| 8 | Ontario part, Ontario/Quebec | 135.79 | 12 |
| 9 | Prince Edward Island | 123.80 | 8 |
| 10 | Quebec | 131.44 | 1 |
| 11 | Saskatchewan | 125.89 | 3 |
| 12 | Yukon | 141.50 | 9 |

Here, we are linking each province by its specified 'provinceID' with another dataset, 'canada_provinces.geojson', containing all the mapping information for plotting our provinces.

First, we need to download the Canadian Provinces dataset from IBM cloud storage, using the `requests.get()` function.

[45]:
```
geo = requests.get("https://cf-courses-data.s3.us.cloud-object-storage.
↪appdomain.cloud/IBM-ML0232EN-SkillsNetwork/asset/canada_provinces.geojson")
```

Next, we will load the file as a string, using `json.loads()` function.

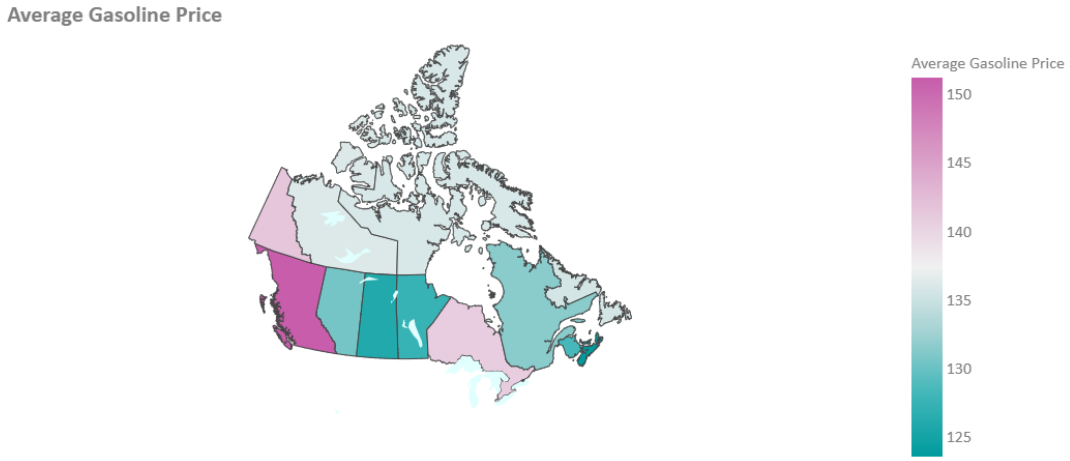
[46]:
```
mp = json.loads(geo.text)

fig = px.choropleth(geodata,
                    locations="ProvinceID",
                    geojson=mp,
                    featureidkey="properties.cartodb_id",
                    color="Average Gasoline Price",
                    color_continuous_scale=px.colors.diverging.Tropic,
                    scope='north america',
                    title='<b>Average Gasoline Price </b>',
                    hover_name='Province',
                    hover_data={
                        'Average Gasoline Price' : True,
                        'ProvinceID' : False
                    },

                    locationmode='geojson-id',
                    )
fig.update_layout(
    showlegend=True,
    legend_title_text='<b>Average Gasoline Price</b>',
    font={"size": 16, "color": "#808080", "family" : "calibri"},
    margin={"r":0,"t":40,"l":0,"b":0},
```

```
    legend=dict(orientation='v'),
    geo=dict(bgcolor='rgba(0,0,0,0)', lakecolor='#e0fffe')
)

#Show Canada only
fig.update_geos(showcountries=False, showcoastlines=False,
                showland=False, fitbounds="locations",
                subunitcolor='white')
fig.show()
```




## 1.14 Exercise 5

In this exercise, experiment with different color scales to make the visualization easier to read. Some suggestions are provided in the "Hint" section. Simply copy the above code and replace 'px.colors.diverging.Tropic', with any other color scales. For example, the sequential color scales are appropriate for most continuous data, but in some cases it can be helpful to use a diverging or cyclical color scale. Diverging color scales are appropriate for the continuous data that has a natural midpoint. For more information on *plotly* colors, please visit this plotly documentation web page.

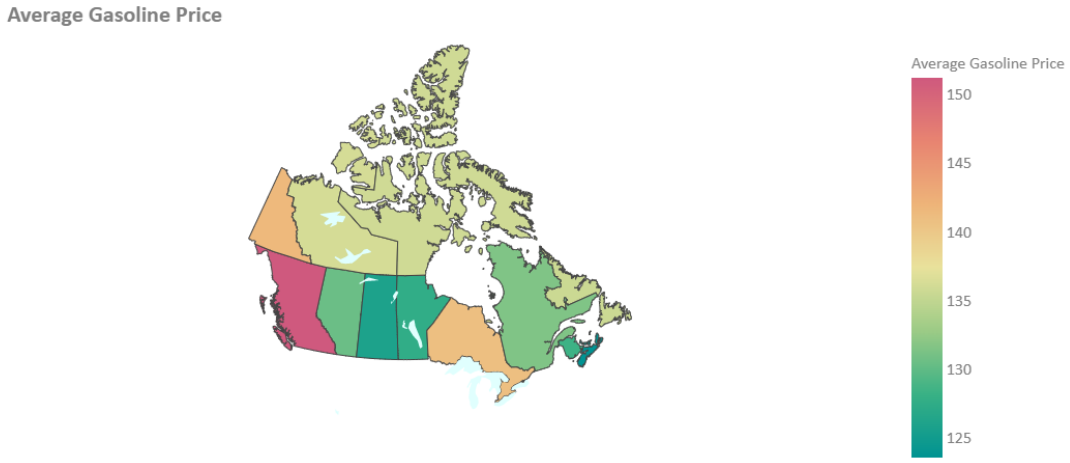
[47]:
```
# Enter your code and run the cell
mp = json.loads(geo.text)

fig = px.choropleth(geodata,
                    locations="ProvinceID",
                    geojson=mp,
                    featureidkey="properties.cartodb_id",
                    color="Average Gasoline Price",
                    color_continuous_scale=px.colors.diverging.Temps,
                    scope='north america',
                    title='<b>Average Gasoline Price </b>',
```

```
                    hover_name='Province',
                    hover_data={
                        'Average Gasoline Price' : True,
                        'ProvinceID' : False
                    },

                    locationmode='geojson-id',
                    )
fig.update_layout(
    showlegend=True,
    legend_title_text='<b>Average Gasoline Price</b>',
    font={"size": 16, "color": "#808080", "family" : "calibri"},
    margin={"r":0,"t":40,"l":0,"b":0},
    legend=dict(orientation='v'),
    geo=dict(bgcolor='rgba(0,0,0,0)', lakecolor='#e0fffe')
)

#Show Canada only
fig.update_geos(showcountries=False, showcoastlines=False,
                showland=False, fitbounds="locations",
                subunitcolor='white')
fig.show()
```




Hint (Click Here)

```
    <code>
px.colors.diverging.Tropic
px.colors.diverging.Temps
px.colors.sequential.Greens
px.colors.sequential.Reds
```

# 2 Congratulations! - You have completed the lab

## 2.1 Author

Svitlana Kramar

## 2.2 Change Log

| Date (YYYY-MM-DD) | Version | Changed By | Change Description |
|---|---|---|---|
| 2022-01-18 | 0.1 | Svitlana K. | Added Introduction |