# Assignment 2 - Deadline: Oct 9, 2024, Wed 11pm

**DSAI 510 Fall 2024**

Complete the assignment below and upload both the .ipynb file and its pdf to https://moodle.boun.edu.tr (https://moodle.boun.edu.tr) by the deadline given above. The submission page on Moodle will close automatically after this date and time.

To make a pdf, this may work: Hit CMD+P or CTRL+P, and save it as PDF. You may also use other options from the File menu.

In [1]:

```
# Run this cell first

import pandas as pd
import numpy as np

# Set the display option to show all rows scrolling with a slider
pd.set_option('display.max_rows', None)
# To disable this, run the line below:
# pd.reset_option('display.max_rows')
```

## Note:

In the problems below, if it asks, "show the number of records that are nonzero", the answer is a number; so you don't need to show the records themselves. But if it asks, "show the records with NaN", it wants you to print those records (rows) containing NAN and other entries, not asking how many such records there are. So be careful about what you're asked.

## Problem 1 (10 pts)

- Load **Electric_Vehicle_Population_Data-modified1.csv** and **Electric_Vehicle_Population_Data-modified2.csv** into pandas dataframes as df1 and df2.
- Inspect the first and last five records with `head()` and `tail()` for both dataframes.
- Use `len()` and `print()` [or `display()`] to show how many records each dataframe contains.
- Use `info()` to get a summary of both dataframes.
- Combine df1 and df2 into a new dataframe called df3 by using `concat()` and print the number of records in the new dataframe df3.
- Find and print the number of duplicate records by using `duplicated()` and `sum()`.
- Drop duplicates, save the new dataframe as dfALL and then print the number of records in dfALL.

In [3]:

```
# Break your computations into multiple cells.

df1 = pd.read_csv("Electric_Vehicle_Population_Data-modified1.csv")
df2 = pd.read_csv("Electric_Vehicle_Population_Data-modified2.csv")

display(df1.head())
display(df1.tail())
display(df2.head())
display(df1.tail())
```

| | VIN (1-10) | County | City | State | Postal Code | Model Year | Make | Model | Electric Vehicle Type |
|---|---|---|---|---|---|---|---|---|---|
| **0** | KM8K33AGXL | King | Seattle | WA | 98103 | 2020 | HYUNDAI | KONA | Battery Electric Vehicle (BEV) |
| **1** | 1C4RJYB61N | King | Bothell | WA | 98011 | 2022 | JEEP | GRAND CHEROKEE | Plug-in Hybrid Electric Vehicle (PHEV) |
| **2** | 1C4RJYD61P | Yakima | Yakima | WA | 98908 | 2023 | JEEP | GRAND CHEROKEE | Plug-in Hybrid Electric Vehicle (PHEV) |
| **3** | 5YJ3E1EA7J | King | Kirkland | WA | 98034 | 2018 | TESLA | MODEL 3 | Battery Electric Vehicle (BEV) |
| **4** | WBY7Z8C5XJ | Thurston | Olympia | WA | 98501 | 2018 | BMW | I3 | Plug-in Hybrid Electric Vehicle (PHEV) |

| | VIN (1-10) | County | City | State | Postal Code | Model Year | Make | Model | Ele Ve |
|---|---|---|---|---|---|---|---|---|---|
| **108441** | WBY8P8C55K | King | Seattle | WA | 98105 | 2019 | BMW | I3 | P H El V (F |
| **108442** | YV4H60CF3R | Pierce | Graham | WA | 98338 | 2024 | VOLVO | XC90 | P H El V (F |
| **108443** | 1FADP5CU7F | Snohomish | Monroe | WA | 98272 | 2015 | FORD | C-MAX | P H El V (F |
| **108444** | 1G1FZ6S07L | Snohomish | Bothell | WA | 98012 | 2020 | CHEVROLET | BOLT EV | B El V |
| **108445** | 5YJ3E1EB1M | Grant | Moses Lake | WA | 98837 | 2021 | TESLA | MODEL 3 | B El V |

| | VIN (1-10) | County | City | State | Postal Code | Model Year | Make | Model | Ele Ve |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1FMCU0EZ1N | Chelan | Wenatchee | WA | 98801.0 | 2022 | FORD | ESCAPE | Pl H Ele Ve (P |
| **1** | 5YJ3E1EB9K | Snohomish | Arlington | WA | 98223.0 | 2019 | TESLA | MODEL 3 | Ba Ele Ve ( |
| **2** | 5YJSA1E57N | King | Woodinville | WA | 98072.0 | 2022 | TESLA | MODEL S | Ba Ele Ve ( |
| **3** | 5YJ3E1EB4J | Snohomish | Snohomish | WA | 98290.0 | 2018 | TESLA | MODEL 3 | Ba Ele Ve ( |
| **4** | KL8CK6S00F | Whatcom | Bellingham | WA | 98225.0 | 2015 | CHEVROLET | SPARK | Ba Ele Ve ( |

| | VIN (1-10) | County | City | State | Postal Code | Model Year | Make | Model | Ele Ve |
|---|---|---|---|---|---|---|---|---|---|
| **108441** | WBY8P8C55K | King | Seattle | WA | 98105 | 2019 | BMW | I3 | P H El V (F |
| **108442** | YV4H60CF3R | Pierce | Graham | WA | 98338 | 2024 | VOLVO | XC90 | P H El V (F |
| **108443** | 1FADP5CU7F | Snohomish | Monroe | WA | 98272 | 2015 | FORD | C-MAX | P H El V (F |
| **108444** | 1G1FZ6S07L | Snohomish | Bothell | WA | 98012 | 2020 | CHEVROLET | BOLT EV | B El V |
| **108445** | 5YJ3E1EB1M | Grant | Moses Lake | WA | 98837 | 2021 | TESLA | MODEL 3 | B El V |

In [ ]:

```
# Comment your code in your own words (not GPT) unless the line is obvious.
```

In [5]:

```
print(len(df1))
print(len(df2))
```

```
108446
50484
```

In [7]:

```
df1.info()
print()
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108446 entries, 0 to 108445
Data columns (total 17 columns):
 #   Column                                             Non-Null C
ount   Dtype
---  ------                                             ----------
----   -----
```

```
 0   VIN (1-10)                                         108446 non
-null  object
 1   County                                             108446 non
-null  object
 2   City                                               108446 non
-null  object
 3   State                                              108446 non
-null  object
 4   Postal Code                                        108446 non
-null  int64
 5   Model Year                                         108446 non
-null  int64
 6   Make                                               108446 non
-null  object
 7   Model                                              108446 non
-null  object
 8   Electric Vehicle Type                              108446 non
-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  108446 non
-null  object
 10  Electric Range                                     108446 non
-null  int64
 11  Base MSRP                                          108446 non
-null  int64
 12  Legislative District                               108407 non
-null  float64
 13  DOL Vehicle ID                                     108446 non
-null  int64
 14  Vehicle Location                                   108445 non
-null  object
 15  Electric Utility                                   108446 non
-null  object
 16  2020 Census Tract                                  108446 non
-null  int64
dtypes: float64(1), int64(6), object(10)
memory usage: 14.1+ MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50484 entries, 0 to 50483
Data columns (total 17 columns):
 #   Column                                             Non-Null C
ount  Dtype
---  ------                                             ----------
----  -----
 0   VIN (1-10)                                         50484 non-
null  object
 1   County                                             50481 non-
null  object
 2   City                                               50481 non-
null  object
 3   State                                              50484 non-
null  object
 4   Postal Code                                        50481 non-
```

```
null  float64
 5   Model Year                                         50484 non-
null  int64
 6   Make                                               50484 non-
null  object
 7   Model                                              50484 non-
null  object
 8   Electric Vehicle Type                              50484 non-
null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  50484 non-
null  object
 10  Electric Range                                     50484 non-
null  int64
 11  Base MSRP                                          50484 non-
null  int64
 12  Legislative District                               50170 non-
null  float64
 13  DOL Vehicle ID                                     50484 non-
null  int64
 14  Vehicle Location                                   50478 non-
null  object
 15  Electric Utility                                   50481 non-
null  object
 16  2020 Census Tract                                  50481 non-
null  float64
dtypes: float64(3), int64(4), object(10)
memory usage: 6.5+ MB
```

In [9]:

```
df3 = pd.concat([df1,df2])

print(len(df3))
```

```
158930
```

In [11]:

```
print(df3.duplicated().sum())
```

```
8448
```

In [13]:

```
dfALL = df3.drop_duplicates()

print(len(dfALL))
```

```
150482
```

## Problem 2 (10 pts)

- Make a new dataframe, keep the columns **Model Year**, **Make**, **Model**, **Electric Range**, **Vehicle Location**, and drop all other columns.
- Change the column name **Model Year** into **Year**.
- Show the record with index number 10.
- As you see, the **Vehicle Location** shows the coordinates in the format "POINT (-122.20264 47.6785)". Here the first number (-122.xxx) is the longitude and second number is the latitude. Make two new columns **Lattitude** and **Longitude**, carry the numbers to these columns by using `str` method from pandas. Finally change the type of **Lattitude** and **Longitude** into float if they're not already. Finally, drop the column **Vehicle Location**.

In [15]:

```
# Take wanted columns as below
columns = [
#    'VIN (1-10)', 'County', 'City', 'State', 'Postal Code',
    'Model Year', 'Make', 'Model','Electric Range', 'Vehicle Locati
on',
#    'Electric Vehicle Type', 'Clean Alternative Fuel Vehicle (CAF
V) Eligibility',
#    'Base MSRP', 'Legislative District', 'DOL Vehicle ID',
#    'Electric Utility', '2020 Census Tract'
]

# Create new DataFrame from above columns
df = dfALL[columns]
```

In [17]:

```
# Use str method to change "Model Year" to "Year"
df.columns = df.columns.str.replace("Model Year", "Year")

df.columns
```

Out[17]:

```
Index(['Year', 'Make', 'Model', 'Electric Range', 'Vehicle Locatio
n'], dtype='object')
```

In [19]:

```
# .iloc[] stands to reach values by index
df.iloc[10]
```

Out[19]:

```
Year                                      2018
Make                                     TESLA
Model                                  MODEL 3
Electric Range                             215
Vehicle Location    POINT (-122.20264 47.6785)
Name: 10, dtype: object
```

In [ ]:

```
# Alternative solution to extract Longitude and Lattitude

def extract_points(input_string):

    long = np.nan
    lati = np.nan

    # Control if a POINT string is acceptible format and non-null
    if input_string and " " in str(input_string):
        # Replace the "POINT (" with nothing and ")" with nothing
        cleaned_string = str(input_string).replace("POINT (", "").r
eplace(")", "")

        # split the rest of the string and extract Longitude and La
ttitude
        cleaned_string = cleaned_string.split(" ")

        long = cleaned_string[0]
        lati = cleaned_string[1]

    # Return coordinates as below format
    return {"Longitude":float(long),"Lattitude":float(lati)}

# df["Vehicle Location"].head()

# Change the format of "Vehicle Location" column
df["Vehicle Location"] = df["Vehicle Location"].apply(extract_point
s)

# Create "Longitude" and "Lattitude" columns based on "Vehicle Loca
tion" column
for i in df["Vehicle Location"][0].keys():
    df[i] = df["Vehicle Location"].apply( lambda d: d[i])

df = df.drop(columns=['Vehicle Location'])
```

In [21]:

```
df = df.copy()

# Create Longitude and Lattitude columns and extract the data from
"Vehicle Location" column

# matches the string "POINT (" followed by two space-separated valu
es, each consisting of one or more characters that are not spaces,
# ending with a closing parenthesis.
df['Longitude'] = df['Vehicle Location'].str.extract(r'POINT \(([^
]+) ([^ ]+)\)')[0].astype(float)
df['Lattitude'] = df['Vehicle Location'].str.extract(r'POINT \(([^
]+) ([^ ]+)\)')[1].astype(float)

df = df.drop(columns=['Vehicle Location'])
```

# Problem 3 (10 pts)

- The file **EV_prices.csv** contains prices for various makes, models, and years of cars. Load this file into a dataframe. Rename the column **Model Year** to **Year**.
- We want to add a new column **Price** to our dataframe from the previous problem. This column will include the price of the car for the corresponding make, model and year if this information is available in the file **EV_prices.csv**. If not available, we'll still keep the record but the entry for price will be empty, NA, None or NaN. To achieve this, merge the dataframe from the previous problem with the dataframe containing the data from **EV_prices.csv**. Think carefully and decide if you need to merge with 'inner' or 'outer' method. At the end, we should have these columns in the merged dataframe: **Year**, **Make**, **Model**, **Electric Range**, **Latitude**, **Longitude** and **Price**. Again, the **Price** column will have numbers only for some records, but it will be empty or NaN for the rest.
- Next, show the number of records which has price information in the **Price** column. Hint: You can use a one-liner containing `len()` and `dropna()` together.

In [23]:

```
df_EV = pd.read_csv("EV_prices.csv")

print(df_EV.columns)

df_EV.columns = df_EV.columns.str.replace("Model Year", "Year")

print(df_EV.columns)
```

```
Index(['Model Year', 'Make', 'Model', 'Price'], dtype='object')
Index(['Year', 'Make', 'Model', 'Price'], dtype='object')
```

In [25]:
```
df_merged = df.merge(df_EV, on=["Year", "Make", "Model"], how="lef
t")

print(df_merged.head())
```

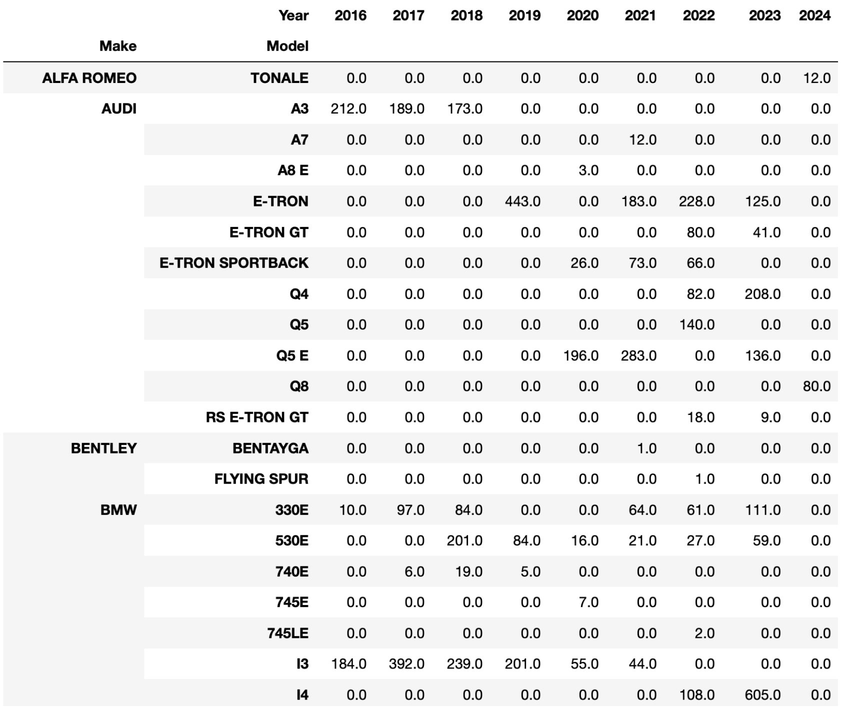
```
   Year     Make            Model  Electric Range   Longitude  Latt
itude  \
0  2020  HYUNDAI             KONA             258 -122.343010  47.6
59185
1  2022     JEEP  GRAND CHEROKEE              25 -122.205780  47.7
62405
2  2023     JEEP  GRAND CHEROKEE              25 -120.602720  46.5
96562
3  2018    TESLA          MODEL 3             215 -122.209285  47.7
11240
4  2018      BMW               I3              97 -122.896920  47.0
43535

     Price
0  22000.0
1      NaN
2      NaN
3  44000.0
4      NaN
```

In [27]:
```
print(len(df_merged[df_merged["Price"].notna()]))
```

```
19745
```

# Problem 4 (10 pts)

- Using the DataFrame from the previous problem, remove records where the **Year** column is for 2015 or earlier. Apply the format `dfmerged = dfmerged[condition]` , choosing the appropriate condition.
- Generate the table, a screenshot of which is provided below, using the `pivot_table()` method and the aggregation function `size` . The entries in the table will indicate the number of cars with the specified make, model, and year in the dataset.



| | Year | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Make** | **Model** | | | | | | | | | |
| **ALFA ROMEO** | **TONALE** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 |
| **AUDI** | **A3** | 212.0 | 189.0 | 173.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **A7** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 | 0.0 | 0.0 | 0.0 |
| | **A8 E** | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **E-TRON** | 0.0 | 0.0 | 0.0 | 443.0 | 0.0 | 183.0 | 228.0 | 125.0 | 0.0 |
| | **E-TRON GT** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 80.0 | 41.0 | 0.0 |
| | **E-TRON SPORTBACK** | 0.0 | 0.0 | 0.0 | 0.0 | 26.0 | 73.0 | 66.0 | 0.0 | 0.0 |
| | **Q4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 82.0 | 208.0 | 0.0 |
| | **Q5** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 140.0 | 0.0 | 0.0 |
| | **Q5 E** | 0.0 | 0.0 | 0.0 | 0.0 | 196.0 | 283.0 | 0.0 | 136.0 | 0.0 |
| | **Q8** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 80.0 |
| | **RS E-TRON GT** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.0 | 9.0 | 0.0 |
| **BENTLEY** | **BENTAYGA** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| | **FLYING SPUR** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **BMW** | **330E** | 10.0 | 97.0 | 84.0 | 0.0 | 0.0 | 64.0 | 61.0 | 111.0 | 0.0 |
| | **530E** | 0.0 | 0.0 | 201.0 | 84.0 | 16.0 | 21.0 | 27.0 | 59.0 | 0.0 |
| | **740E** | 0.0 | 6.0 | 19.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **745E** | 0.0 | 0.0 | 0.0 | 0.0 | 7.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **745LE** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 |
| | **I3** | 184.0 | 392.0 | 239.0 | 201.0 | 55.0 | 44.0 | 0.0 | 0.0 | 0.0 |
| | **I4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 108.0 | 605.0 | 0.0 |

- Use the `groupby()` method to create a table similar to the one above but this time entries will indicate the average latitude of the car with the specified make, model and year.

```
In [29]: df_merged = df_merged[ df_merged.Year > 2015 ]
```

In [31]:
```
df_pivot = df_merged.pivot_table(index=["Make", "Model"], columns="
Year", aggfunc="size", fill_value=0.0)

display(df_pivot.head(20))
```

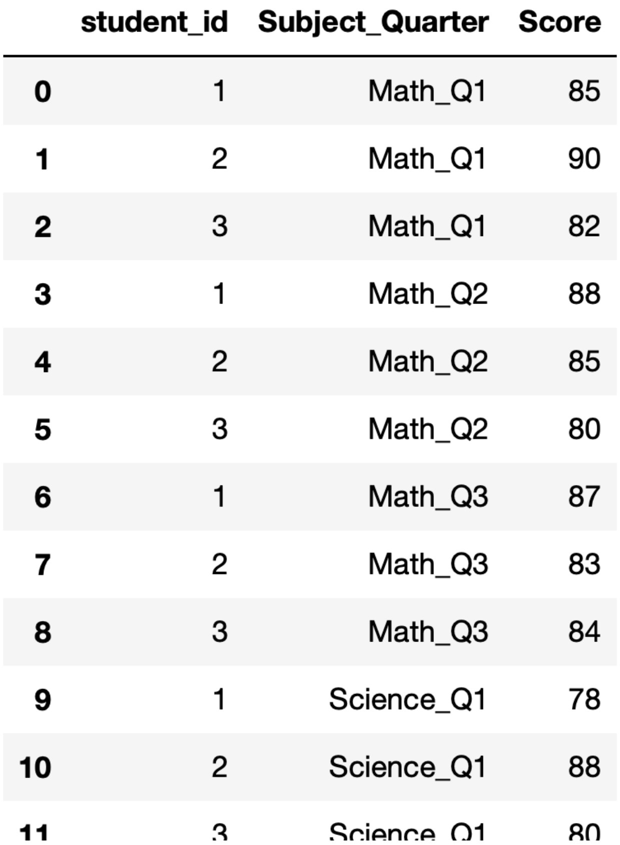
| | Year | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 | 2022 | 2023 | 2024 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Make** | **Model** | | | | | | | | | |
| **ALFA ROMEO** | **TONALE** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 |
| **AUDI** | **A3** | 212.0 | 189.0 | 173.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **A7** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 12.0 | 0.0 | 0.0 | 0.0 |
| | **A8 E** | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **E-TRON** | 0.0 | 0.0 | 0.0 | 443.0 | 0.0 | 183.0 | 228.0 | 125.0 | 0.0 |
| | **E-TRON GT** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 80.0 | 41.0 | 0.0 |
| | **E-TRON SPORTBACK** | 0.0 | 0.0 | 0.0 | 0.0 | 26.0 | 73.0 | 66.0 | 0.0 | 0.0 |
| | **Q4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 82.0 | 208.0 | 0.0 |
| | **Q5** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 140.0 | 0.0 | 0.0 |
| | **Q5 E** | 0.0 | 0.0 | 0.0 | 0.0 | 196.0 | 283.0 | 0.0 | 136.0 | 0.0 |
| | **Q8** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 80.0 |
| | **RS E-TRON GT** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 18.0 | 9.0 | 0.0 |
| **BENTLEY** | **BENTAYGA** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| | **FLYING SPUR** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| **BMW** | **330E** | 10.0 | 97.0 | 84.0 | 0.0 | 0.0 | 64.0 | 61.0 | 111.0 | 0.0 |
| | **530E** | 0.0 | 0.0 | 201.0 | 84.0 | 16.0 | 21.0 | 27.0 | 59.0 | 0.0 |
| | **740E** | 0.0 | 6.0 | 19.0 | 5.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **745E** | 0.0 | 0.0 | 0.0 | 0.0 | 7.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | **745LE** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | 0.0 | 0.0 |
| | **I3** | 184.0 | 392.0 | 239.0 | 201.0 | 55.0 | 44.0 | 0.0 | 0.0 | 0.0 |

In [33]:
```
df_groupby = df_merged[["Make","Model","Year","Lattitude"]].groupby(["Make","Model","Year"]).mean("Lattitude")

display(df_groupby.head(20))
```

| | | | Lattitude |
|---|---|---|---|
| **Make** | **Model** | **Year** | |
| **ALFA ROMEO** | **TONALE** | **2024** | 47.607409 |
| **AUDI** | **A3** | **2016** | 47.619987 |
| | | **2017** | 47.481332 |
| | | **2018** | 47.496019 |
| | **A7** | **2021** | 47.491215 |
| | **A8 E** | **2020** | 47.742747 |
| | **E-TRON** | **2019** | 47.481997 |
| | | **2021** | 47.510774 |
| | | **2022** | 47.525202 |
| | | **2023** | 47.166119 |
| | **E-TRON GT** | **2022** | 47.586789 |
| | | **2023** | 47.359299 |
| | **E-TRON SPORTBACK** | **2020** | 47.543414 |
| | | **2021** | 47.440235 |
| | | **2022** | 47.463412 |
| | **Q4** | **2022** | 47.535088 |
| | | **2023** | 47.483677 |
| | **Q5** | **2022** | 47.497235 |
| | **Q5 E** | **2020** | 47.457532 |
| | | **2021** | 47.492323 |

## Problem 5 (10 pts)

- There is a 3-row, 7-columns table whose code is given below. Use `melt()` to convert that table into this form:



| | student_id | Subject_Quarter | Score |
|---|---|---|---|
| **0** | 1 | Math_Q1 | 85 |
| **1** | 2 | Math_Q1 | 90 |
| **2** | 3 | Math_Q1 | 82 |
| **3** | 1 | Math_Q2 | 88 |
| **4** | 2 | Math_Q2 | 85 |
| **5** | 3 | Math_Q2 | 80 |
| **6** | 1 | Math_Q3 | 87 |
| **7** | 2 | Math_Q3 | 83 |
| **8** | 3 | Math_Q3 | 84 |
| **9** | 1 | Science_Q1 | 78 |
| **10** | 2 | Science_Q1 | 88 |
| **11** | 3 | Science_Q1 | 80 |

In [35]:

```
# Sample dataset
data = {
    'student_id': [1, 2, 3],
    'Math_Q1': [85, 90, 82],
    'Math_Q2': [88, 85, 80],
    'Math_Q3': [87, 83, 84],
    'Science_Q1': [78, 88, 80],
    'Science_Q2': [82, 85, 78],
    'Science_Q3': [84, 87, 83]
}
scores_df = pd.DataFrame(data)
scores_df
```

Out[35]:

| | student_id | Math_Q1 | Math_Q2 | Math_Q3 | Science_Q1 | Science_Q2 | Science_Q3 |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 85 | 88 | 87 | 78 | 82 | 84 |
| **1** | 2 | 90 | 85 | 83 | 88 | 85 | 87 |
| **2** | 3 | 82 | 80 | 84 | 80 | 78 | 83 |

In [37]:

```
# your solution goes here
melted_df = pd.melt(scores_df, id_vars=['student_id'], var_name='Subject_Quarter', value_name='Score')

print("\nMelted Data:")
display(melted_df)
```

```
Melted Data:
```

| | student_id | Subject_Quarter | Score |
|---|---|---|---|
| **0** | 1 | Math_Q1 | 85 |
| **1** | 2 | Math_Q1 | 90 |
| **2** | 3 | Math_Q1 | 82 |
| **3** | 1 | Math_Q2 | 88 |
| **4** | 2 | Math_Q2 | 85 |
| **5** | 3 | Math_Q2 | 80 |
| **6** | 1 | Math_Q3 | 87 |
| **7** | 2 | Math_Q3 | 83 |
| **8** | 3 | Math_Q3 | 84 |
| **9** | 1 | Science_Q1 | 78 |
| **10** | 2 | Science_Q1 | 88 |
| **11** | 3 | Science_Q1 | 80 |
| **12** | 1 | Science_Q2 | 82 |
| **13** | 2 | Science_Q2 | 85 |
| **14** | 3 | Science_Q2 | 78 |
| **15** | 1 | Science_Q3 | 84 |
| **16** | 2 | Science_Q3 | 87 |
| **17** | 3 | Science_Q3 | 83 |

## Problem 5 - Quality Control in a Manufacturing Plant (10 pts)

Imagine you work as a quality control analyst in a manufacturing plant that produces ball bearings. Each day, multiple batches of ball bearings are produced. To ensure the consistency and quality of the ball bearings, samples from each batch are measured to determine their diameters.

Over the course of a month, you've collected diameter data for these samples from various batches. The objective is to determine the batch consistency by measuring the standard deviation of the diameters. A lower standard deviation would indicate that the ball bearings in a batch are more consistent in size.

- Load the **ball_bearings.csv** file into a dataframe.
- Use `groupby()` to calculate the standard deviation for each batch.
- Sort the results in descending order wrt standard deviation, showing the batch with highest standard deviation at the top.

In [39]:

```
df_ball_bearings = pd.read_csv("ball_bearings.csv")

display(df_ball_bearings.head())
```

| | batch_id | diameter |
|---|---|---|
| **0** | 1 | 50.248357 |
| **1** | 1 | 49.930868 |
| **2** | 1 | 50.323844 |
| **3** | 1 | 50.761515 |
| **4** | 1 | 49.882923 |

In [41]:

```
df_grouped = df_ball_bearings.groupby(["batch_id"])["diameter"].std()

display(df_grouped)
```

```
batch_id
1      0.480014
2      0.484019
3      0.410424
4      0.556044
5      0.345405
6      0.511339
7      0.534851
8      0.451574
9      0.502297
10     0.367445
11     0.538856
12     0.593042
13     0.516218
14     0.559482
15     0.474119
16     0.319080
17     0.431002
18     0.362575
19     0.441640
20     0.577886
21     0.446483
22     0.645821
23     0.386267
24     0.608465
25     0.508577
26     0.409027
27     0.425961
28     0.435184
29     0.610599
30     0.388254
Name: diameter, dtype: float64
```

In [43]:

```
display(df_grouped.sort_values(ascending=False))
```

```
batch_id
22    0.645821
29    0.610599
24    0.608465
12    0.593042
20    0.577886
14    0.559482
4     0.556044
11    0.538856
7     0.534851
13    0.516218
6     0.511339
25    0.508577
9     0.502297
2     0.484019
1     0.480014
15    0.474119
8     0.451574
21    0.446483
19    0.441640
28    0.435184
17    0.431002
27    0.425961
3     0.410424
26    0.409027
30    0.388254
23    0.386267
10    0.367445
18    0.362575
5     0.345405
16    0.319080
Name: diameter, dtype: float64
```