Pandas

Pandas

Pandas is an open source Python package

- widely used for **data science/data analysis and machine learning tasks**.
- offers data structures and operations to manipulate numerical tables and time series.
- can import data from various file formats such as CSV and JSON
- built on top of other packages such as Numpy

Pandas in Jupyter Notebook

Launch a new Jupyter Notebook

- type the following commands to check that Pandas is already installed in your Anaconda/Jupyter Notebook

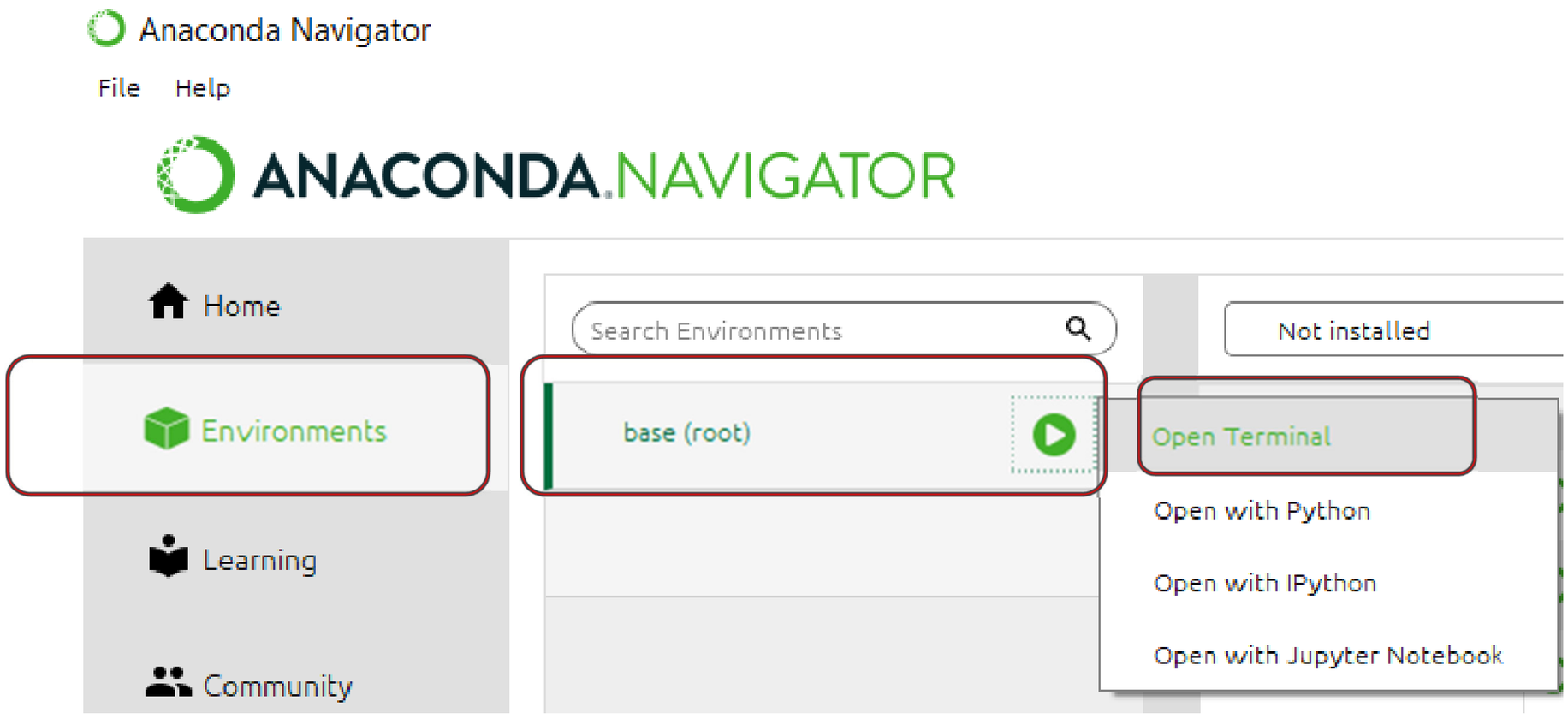
```
In [1]:  import pandas as pd

In [2]:  pd.__version__

Out[2]:  '1.3.4'
```

Aside: Installation of Pandas

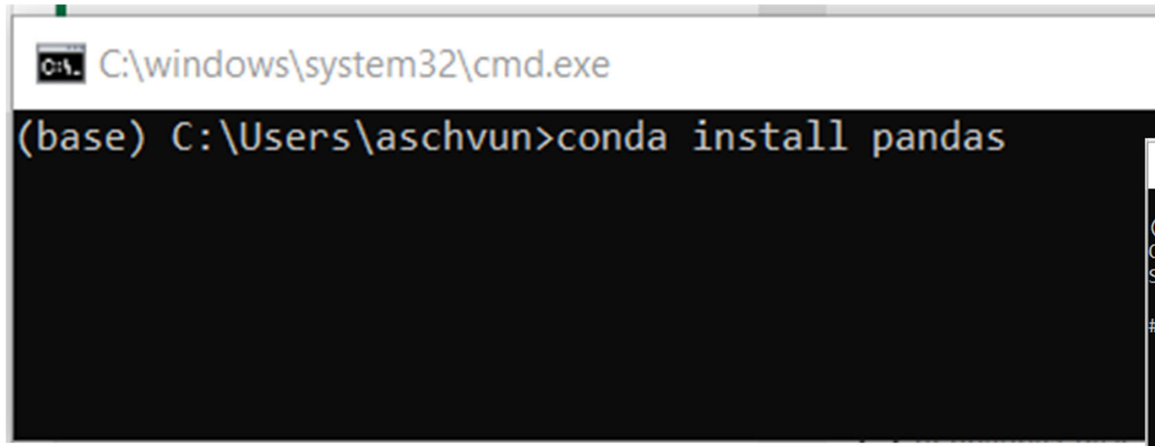
To install the Pandas library using Navigator

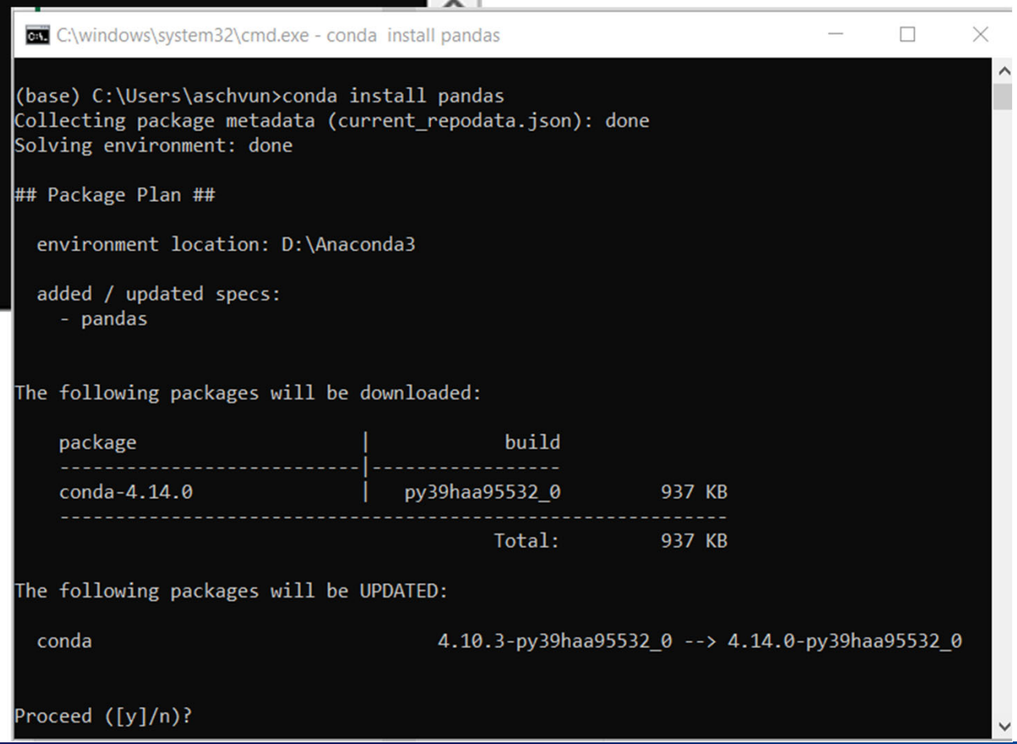


Aside: Installation of Pandas

In the terminal

- type “conda install pandas”





DataFrame

Pandas stores data using a 2-D data structure with rows and columns

- known as DataFrame

Finger Exercise

Creating a DataFrame (from a Dictionary based ‘Database’)

```
Database_cars = {
  "Brand": ["Toyota", "Mazada", "VW", "BMW", "Honda"],
  "Model": ["Corolla", "CX5", "Jetta", "X3", "Civic"],
  "Year": [2018, 2020, 2019, 2018, 2021],
  "COE": [31000, 25000, 28000, 33000, 45000],
  "Cat": ["A", "B", "A", "B", "A"],
  "Price": [51000, 73000, 70000, 98000, 81000]
}
```

```
Cars = pd.DataFrame(Database_car) #load data into a DataFrame object
```

Displaying the DataFrame

To display the DataFrame created

Finger Exercise

```
Cars = pd.DataFrame(Database_car) #load data into a DataFrame object
```

```
print(Cars)
```

```
    Brand    Model  Year    COE Cat  Price
0  Toyota  Corolla  2018  31000   A  51000
1  Mazada      CX5  2020  25000   B  73000
2      VW    Jetta  2019  28000   A  70000
3     BMW       X3  2018  33000   B  98000
4   Honda    Civic  2021  45000   A  81000
```

Locating Entries

You can locate an entry by row using the `loc` method

Finger Exercise:

```
# entries in first row
print(Cars.loc[0])
```

```
Brand      Toyota
Model     Corolla
Year         2018
COE         31000
Cat             A
Price       51000
Name: 0, dtype: object
```

	Brand	Model	Year	COE	Cat	Price
0	Toyota	Corolla	2018	31000	A	51000
1	Mazada	CX5	2020	25000	B	73000
2	VW	Jetta	2019	28000	A	70000
3	BMW	X3	2018	33000	B	98000
4	Honda	Civic	2021	45000	A	81000

Locating Entries

You can locate an entry by row: `loc` method

Finger Exercise

```
# entries in third row
print(Cars.loc[2])
```

```
Brand       VW
Model    Jetta
Year      2019
COE      28000
Cat          A
Price    70000
Name: 2, dtype: object
```

```
    Brand    Model  Year    COE Cat  Price
0  Toyota  Corolla  2018  31000   A  51000
1  Mazada      CX5  2020  25000   B  73000
2      VW    Jetta  2019  28000   A  70000
3     BMW       X3  2018  33000   B  98000
4   Honda    Civic  2021  45000   A  81000
```

```
print(Cars.loc[[2, 4]])
```

```
    Brand  Model  Year    COE Cat  Price
2      VW  Jetta  2019  28000   A  70000
4   Honda  Civic  2021  45000   A  81000
```

Series of Entry

You can also use Pandas to create (series of) lists from the database:
Series method

```
Database_cars = {
  "Brand": ["Toyota", "Mazada", "VW", "BMW", "Honda"],
  "Model": ["Corolla", "CX5", "Jetta", "X3", "Civic"],
  "Year": [2018, 2020, 2019, 2018, 2021],
  "COE": [31000, 25000, 28000, 33000, 45000],
  "Cat": ["A", "B", "A", "B", "A"],
  "Price": [51000, 73000, 70000, 98000, 81000]
}
```

Finger Exercise:

```
Cars_lists = pd.Series(Database_cars)
print(Cars_lists)
```

```
Brand       [Toyota, Mazada, VW, BMW, Honda]
Model       [Corolla, CX5, Jetta, X3, Civic]
Year          [2018, 2020, 2019, 2018, 2021]
COE      [31000, 25000, 28000, 33000, 45000]
Cat                          [A, B, A, B, A]
Price    [51000, 73000, 70000, 98000, 81000]
dtype: object
```

Access the list in Pandas Series

You can then access the lists in the series

Finger Exercise:

```
print(pd.Series(Database_cars)[0])
```

```
['Toyota', 'Mazada', 'VW', 'BMW', 'Honda']
```

```
print(pd.Series(Database_cars)[3])
```

```
[31000, 25000, 28000, 33000, 45000]
```

```
print(pd.Series(Database_cars)[1:3])
```

```
Model    [Corolla, CX5, Jetta, X3, Civic]
Year       [2018, 2020, 2019, 2018, 2021]
dtype: object
```

```
Database_cars = {
  "Brand": ["Toyota", "Mazada", "VW", "BMW", "Honda"],
  "Model": ["Corolla", "CX5", "Jetta", "X3", "Civic"],
  "Year": [2018, 2020, 2019, 2018, 2021],
  "COE": [31000, 25000, 28000, 33000, 45000],
  "Cat": ["A", "B", "A", "B", "A"],
  "Price": [51000, 73000, 70000, 98000, 81000]
}
```

Recap - CSV File

In practice, it is more likely that you will load your data sets from a file

- Data sets are commonly stored as a CSV (Comma-separated values) file

A CSV file is a text file that uses a comma to separate values (in each line)

- each line in the file is a data record
- each record consists of one or more fields, separated by commas

```
1  Brand,Year,COE,Cat
2  Volvo, 2012, 40744, B
3  Ford, 2007, 38403, B
4  Honda, 2018, 93099, B
5  GMC, 2017, 51073, B
6  VW, 2014, 46771, A
7  Toyota, 2000, 97010, B
8  Volvo, 2019, 72859, B
9  Tesla, 2022, 46315, B
10 Volvo, 2003, 48109, B
11 Ford, 2001, 68804, A
12 VW, 2013, 49381, B
13 Nissan, 2013, 65209, A
14 Honda, 2017, 38134, B
15 Tesla, 2011, 47726, A
16 Ford, 2010, 30651, A
```

Reading CSV File

To load a CSV file into a DataFrame: read_csv()

```
df = pd.read_csv('Cars.csv')
```

To display the DataFrame to console-friendly output : to_string

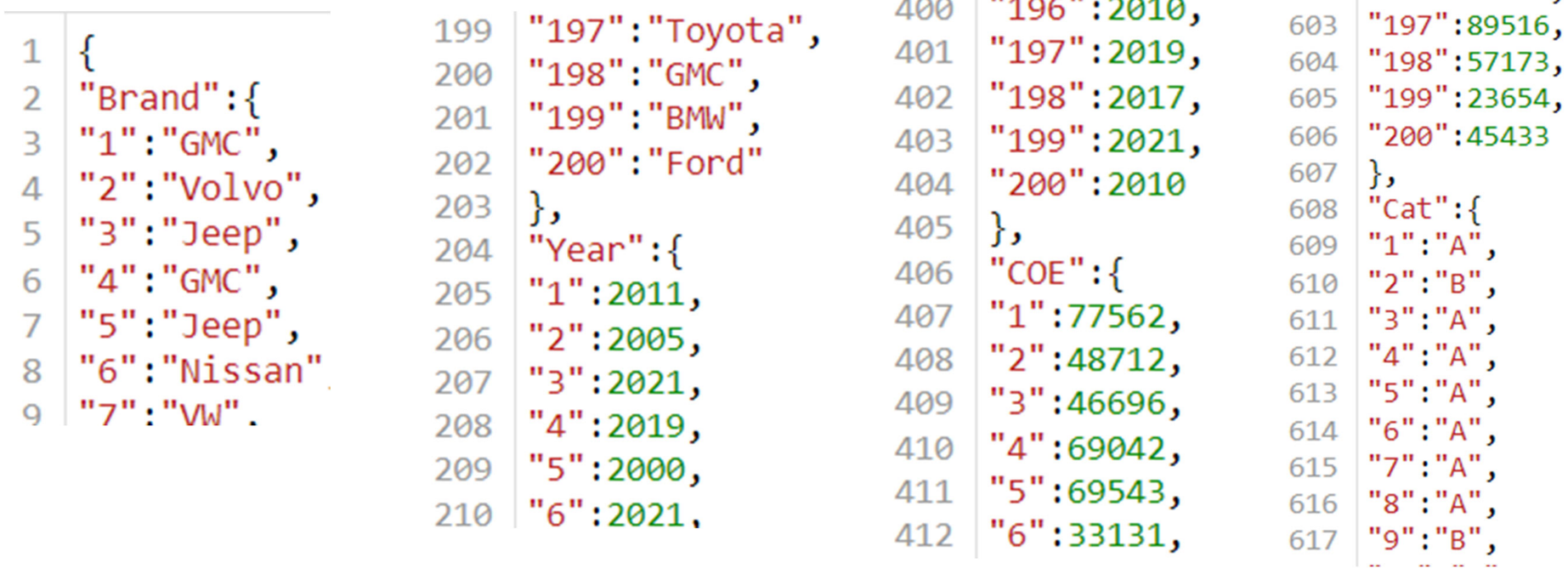
```
print(df.to_string())
```

```
    Brand  Year    COE Cat
0   Volvo  2012  40744   B
1    Ford  2007  38403   B
2   Honda  2018  93099   B
3     GMC  2017  51073   B
4      VW  2014  46771   A
5  Toyota  2000  97010   B
6   Volvo  2019  72859   B
7   Tesla  2022  46315   B
8   Volvo  2003  48109   B
9    Ford  2001  68804   A
10     VW  2013  49381   B
11 Nissan  2013  65209   A
12  Honda  2017  38134   B
```

Recap: JSON File

Big data sets are usually stored as a JSON (JavaScript Object Notation) file

- consists of attribute-values pairs and arrays



Reading JSON File

To load a JSON file into a Pandas DataFrame: read_json()

```
df = pd.read_json('Cars.json')
```

```
print(df.to_string())
```

	Brand	Year	COE	Cat
1	GMC	2011.0	77562.0	A
2	Volvo	2005.0	48712.0	B
3	Jeep	2021.0	46696.0	A
4	GMC	2019.0	69042.0	A
5	Jeep	2000.0	69543.0	A
6	Nissan	2021.0	33131.0	A
7	VW	2004.0	79464.0	A
8	BMW	2004.0	32030.0	A
9	Nissan	2013.0	45242.0	B
10	GMC	2016.0	89946.0	NaN
11	VW	2019.0	98119.0	A
12	Volvo	2007.0	47119.0	B
13	Honda	2019.0	42189.0	B
14	Volvo	2002.0	90204.0	A

NaN = ‘Not a number’

- undefined entry
- probably due to missing entry

Examine the Entries

Shows the first 5 rows and last 5 rows of the DataFrame

CSV

```
print(df)
```

```
      Brand  Year    COE Cat
0     Volvo  2012  40744   B
1      Ford  2007  38403   B
2     Honda  2018  93099   B
3       GMC  2017  51073   B
4        VW  2014  46771   A
..      ...   ...    ...  ..
195   Honda  2021  89704   A
196   Volvo  2010  69027   A
197     GMC  2004  43849   B
198  Toyota  2011  37264   A
199   Tesla  2020  35093   A

[200 rows x 4 columns]
```

JSON

```
print(df)
```

```
      Brand    Year       COE  Cat
1       GMC  2011.0   77562.0    A
2     Volvo  2005.0   48712.0    B
3      Jeep  2021.0   46696.0    A
4       GMC  2019.0   69042.0    A
5      Jeep  2000.0   69543.0    A
..      ...     ...       ...  ...
197  Toyota  2019.0   89516.0    B
198     GMC  2017.0   57173.0    B
199     BMW  2021.0   23654.0    A
200    Ford  2010.0   45433.0  NaN
201     NaN     NaN       NaN    A

[201 rows x 4 columns]
```

`head()` and `tail()` methods

To have a quick overview of the Dataframe: **`head()` and `tail()`** methods

```
print(df.head(10))
```

	Brand	Year	COE	Cat
0	Volvo	2012	40744	B
1	Ford	2007	38403	B
2	Honda	2018	93099	B
3	GMC	2017	51073	B
4	VW	2014	46771	A
5	Toyota	2000	97010	B
6	Volvo	2019	72859	B
7	Tesla	2022	46315	B
8	Volvo	2003	48109	B
9	Ford	2001	68804	A

```
print(df.tail(10))
```

	Brand	Year	COE	Cat
190	Ford	2001	37253	B
191	Volvo	2005	62525	A
192	GMC	2002	53611	A
193	Nissan	2013	41440	B
194	GMC	2002	97235	A
195	Honda	2021	89704	A
196	Volvo	2010	69027	A
197	GMC	2004	43849	B
198	Toyota	2011	37264	A
199	Tesla	2020	35093	A

`info()` method

Getting information about the data set: `info()`

CSV

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Brand   200 non-null    object
 1   Year    200 non-null    int64 
 2   COE     200 non-null    int64 
 3   Cat     200 non-null    object
dtypes: int64(2), object(2)
memory usage: 6.4+ KB
None
```

JSON

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201 entries, 1 to 201
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Brand   200 non-null    object 
 1   Year    200 non-null    float64
 2   COE     200 non-null    float64
 3   Cat     181 non-null    object 
dtypes: float64(2), object(2)
memory usage: 7.9+ KB
None
```

(Note :Some entries with no values)

`isin()` method

Checking for specific value(s): `isin()`

```
df['Brand'].isin(['GMC'])
```

```
1       True
2      False
3      False
4       True
5      False
       ...
197    False
198     True
199    False
200    False
201    False
Name: Brand, Length: 201, dtype: bool
```

```
print(df.to_string())
```

	Brand	Year	COE	Cat
1	GMC	2011.0	77562.0	A
2	Volvo	2005.0	48712.0	B
3	Jeep	2021.0	46696.0	A
4	GMC	2019.0	69042.0	A
5	Jeep	2000.0	69543.0	A
6	Nissan	2021.0	33131.0	A
7	VW	2004.0	79464.0	A
8	BMW	2004.0	32030.0	A
9	Nissan	2013.0	45242.0	B
10	GMC	2016.0	89946.0	NaN
11	VW	2019.0	98119.0	A
12	Volvo	2007.0	47119.0	B
13	Honda	2019.0	42189.0	B
14	Volvo	2002.0	90204.0	A

`isin()` method

Filtered for specific value(s): `isin()`

```
df[df['Brand'].isin(['GMC'])]
```

	Brand	Year	COE	Cat
1	GMC	2011.0	77562.0	A
2	Volvo	2005.0	48712.0	B
3	Jeep	2021.0	46696.0	A
4	GMC	2019.0	69042.0	A
5	Jeep	2000.0	69543.0	A
6	Nissan	2021.0	33131.0	A
7	VW	2004.0	79464.0	A
8	BMW	2004.0	32030.0	A
9	Nissan	2013.0	45242.0	B
10	GMC	2016.0	89946.0	NaN
11	VW	2019.0	98119.0	A
12	Volvo	2007.0	47119.0	B
13	Honda	2019.0	42189.0	B
14	Volvo	2002.0	90204.0	A

	Brand	Year	COE	Cat
1	GMC	2011.0	77562.0	A
4	GMC	2019.0	69042.0	A
10	GMC	2016.0	89946.0	NaN
20	GMC	2008.0	42227.0	NaN
24	GMC	2006.0	41729.0	A
35	GMC	2003.0	62779.0	B
45	GMC	2019.0	72109.0	A
52	GMC	2007.0	57751.0	B
61	GMC	2014.0	45189.0	B
84	GMC	2013.0	79460.0	A
86	GMC	2005.0	28675.0	B
87	GMC	2014.0	40255.0	A
94	GMC	2004.0	89171.0	B
134	GMC	2017.0	58555.0	B
135	GMC	2014.0	61186.0	B
136	GMC	2000.0	38796.0	A
145	GMC	2005.0	31469.0	A
153	GMC	2013.0	47876.0	B
154	GMC	2018.0	21247.0	B
187	GMC	2012.0	61498.0	B
192	GMC	2016.0	20334.0	B
198	GMC	2017.0	57173.0	B

`isna()` method

To check which entries has NaN entries (in the JSON file): `isna()` (or `isnull()`)

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 201 entries, 1 to 201
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Brand   200 non-null    object 
 1   Year    200 non-null    float64
 2   COE     200 non-null    float64
 3   Cat     181 non-null    object 
dtypes: float64(2), object(2)
memory usage: 7.9+ KB
None
```

```
nan_values = df[df.isna().any(axis=1)]
print (nan_values)
```

	Brand	Year	COE	Cat
10	GMC	2016.0	89946.0	NaN
20	GMC	2008.0	42227.0	NaN
30	Tesla	2005.0	23145.0	NaN
40	Nissan	2006.0	35057.0	NaN
50	BMW	2022.0	42506.0	NaN
60	Toyota	2001.0	85123.0	NaN
70	Toyota	2013.0	49827.0	NaN
80	VW	2001.0	84749.0	NaN
90	BMW	2021.0	55382.0	NaN
100	Volvo	2009.0	54084.0	NaN
110	Tesla	2016.0	28553.0	NaN
120	Honda	2010.0	33041.0	NaN
130	BMW	2009.0	25391.0	NaN
140	Jeep	2016.0	47339.0	NaN
150	Tesla	2015.0	82138.0	NaN
160	BMW	2021.0	45254.0	NaN
170	Toyota	2003.0	21333.0	NaN
180	Tesla	2003.0	43256.0	NaN
190	Ford	2014.0	84997.0	NaN
200	Ford	2010.0	45433.0	NaN
201	NaN	NaN	NaN	A

Cleaning Data – `dropna()` method

Empty cells can cause problem when we analyze the data

- to remove the rows that contains empty cell(s) : `dropna()`

```
new_df = df.dropna()
print(new_df.to_string())
```

```
      Brand    Year       COE Cat
1       GMC  2011.0  77562.0   A
2     Volvo  2005.0  48712.0   B
3      Jeep  2021.0  46696.0   A
4       GMC  2019.0  69042.0   A
5      Jeep  2000.0  69543.0   A
6    Nissan  2021.0  33131.0   A
7        VW  2004.0  79464.0   A
8       BMW  2004.0  32030.0   A
9    Nissan  2013.0  45242.0   B
11       VW  2019.0  98119.0   A
12    Volvo  2007.0  47119.0   B
13    Honda  2019.0  42189.0   B
14    Volvo  2002.0  90204.0   A
15    Tesla  2018.0  48349.0   B
```

```
      Brand    Year       COE  Cat
1       GMC  2011.0  77562.0    A
2     Volvo  2005.0  48712.0    B
3      Jeep  2021.0  46696.0    A
4       GMC  2019.0  69042.0    A
5      Jeep  2000.0  69543.0    A
6    Nissan  2021.0  33131.0    A
7        VW  2004.0  79464.0    A
8       BMW  2004.0  32030.0    A
9    Nissan  2013.0  45242.0    B
10      GMC  2016.0  89946.0  NaN
11       VW  2019.0  98119.0    A
12    Volvo  2007.0  47119.0    B
13    Honda  2019.0  42189.0    B
14    Volvo  2002.0  90204.0    A
15    Tesla  2018.0  48349.0    B
16   Nissan  2016.0  30995.0    A
17   Toyota  2015.0  24552.0    A
18    Volvo  2022.0  30227.0    B
```

Cleaning Data – `fillna()` method

Empty cells can cause problem when we analyze the data

- fill empty cells with new default values: `fillna()` with specific column

```
df = pd.read_csv('Cars.csv')
print(df.to_string())
```

```
    Brand  Year    COE  Cat
0   Volvo  2012  40744    B
1    Ford  2007  38403    B
2   Honda  2018  93099    B
3     GMC  2017  51073    B
4      VW  2014  46771  NaN
5  Toyota  2000  97010    B
6   Volvo  2019  72859    B
7   Tesla  2022  46315    B
8   Volvo  2003  48109  NaN
9    Ford  2001  68804    A
10     VW  2013  49381    B
11 Nissan  2013  65209    A
12  Honda  2017  38134    B
13  Tesla  2011  47726  NaN
14   Ford  2010  30651    A
15  Honda  2003  81746    B
16  Volvo  2021  82735    A
```

```
nan_values = df[df.isnull().any(axis=1)]
print (nan_values)
```

```
    Brand  Year      COE  Cat
4      VW  2014  46771.0  NaN
8   Volvo  2003  48109.0  NaN
13  Tesla  2011  47726.0  NaN
27    NaN  2018  92140.0    B
81    BMW  2008      NaN    A
```

```
df["Cat"]=df["Cat"].fillna('A')
```

```
print(df.loc[[4, 8, 13]])
```

```
    Brand  Year      COE Cat
4      VW  2014  46771.0   A
8   Volvo  2003  48109.0   A
13  Tesla  2011  47726.0   A
```

Replacing Wrong Data

For obvious wrong data (e.g. typo)

- replace it with another value

```
df = pd.read_csv('Cars.csv')
print(df.to_string())
```

```
16    Volvo  2021  82735    A
17   Nissan  2002  62360    B
18      GMC  2022  25732    A
19   Toyota  2010  34647    B
20       VW  2002  71148    B
21    Tesla  2012  56188    A
22      GMC  1009  70138    A
23    Honda  2015  83285    B
24      GMC  2013  58722    B
25      GMC  2013  58722    B
26    Honda  2004  97517    B
27     Ford  2018  92140    B
28       VW  2015  58221    B
29      GMC  2001  23711    A
30     Ford  2012  34368    A
31   Nissan  2010  47178    A
32    Volvo  2021  87538    A
33   Nissan  2022  78178    A
```

```
df.loc[22, 'Year'] = 2009
```

```
print(df.to_string())
```

```
19   Toyota  2010  34647    B
20       VW  2002  71148    B
21    Tesla  2012  56188    A
22      GMC  2009  70138    A
23    Honda  2015  83285    B
24      GMC  2013  58722    B
25      GMC  2013  58722    B
26    Honda  2004  97517    B
27     Ford  2018  92140    B
28       VW  2015  58221    B
```

```
#Replace all value that is less than 40000 to 40000!!
for x in df.index:
  if df.loc[x, "COE"] < 40000:
    df.loc[x, "COE"] = 40000
```

Checking Duplicated Data

To check for duplicate data: duplicated() method

```
df = pd.read_csv('Cars.csv')
print(df.to_string())
```

```
22      GMC  1009  70138     A
23    Honda  2015  83285     B
24      GMC  2013  58722     B
25      GMC  2013  58722     B
26    Honda  2004  97517     B
27     Ford  2018  92140     B
28       VW  2015  58221     B
```

```
# Check for duplicate
duplicateRows = df[df.duplicated()]
```

```
duplicateRows
```

	Brand	Year	COE	Cat
25	GMC	2013	58722	B

Cleaning Duplicated Data

To remove duplicate data: drop_duplicated() method

```
df = pd.read_csv('Cars.csv')
print(df.to_string())
```

```
22      GMC  1009  70138    A
23    Honda  2015  83285    B
24      GMC  2013  58722    B
25      GMC  2013  58722    B
26    Honda  2004  97517    B
27     Ford  2018  92140    B
28       VW  2015  58221    B
```

```
df.drop_duplicates(inplace = True)
```

```
print(df.to_string())
```

```
19   Toyota  2010  34647    B
20       VW  2002  71148    B
21    Tesla  2012  56188    A
22      GMC  2009  70138    A
23    Honda  2015  83285    B
24      GMC  2013  58722    B
26    Honda  2004  97517    B
27     Ford  2018  92140    B
28       VW  2015  58221    B
```

Data Analysis

To find mean value of a column: mean() method

```
df["COE"].mean()
```

```
59347.79899497487
```

To find Median value of a column: median() method

```
df["Year"].median()
```

```
2011.0
```

Data Analysis

To count frequency of occurrence: value_counts() method

```
df1 = df['Brand'].value_counts()
```

```
print(df1)
```

```
Honda      27
Nissan     27
Volvo      26
GMC        23
VW         21
Toyota     21
Ford       20
Tesla      19
BMW        15
Name: Brand, dtype: int64
```

```
df2 = df['Cat'].value_counts()
```

```
print(df2)
```

```
B     106
A      90
Name: Cat, dtype: int64
```

Finally – Exporting to files

To save/export the (cleaned) DataFrame to a CSV file:

- use to_csv()

```
df.to_csv('Cars_cleaned.csv')
```

To save/export the (cleaned) DataFrame to a JSON file

- use to_json()

```
df.to_json('Cars_cleaned.json')
```

Exploratory Data Analysis: with Seaborn for Data Visualization

Seaborn

A data visualization library built on top of matplotlib

- a high-level interface for drawing attractive and informative statistical graphics
  - allowing you to create complex visualizations using a simple API
  - provides simple codes to visualize complex statistical plots
  - offers multiple choices for plotting styles
  - integrated with the functionality provided by the Pandas's data frame
    - commonly used with Pandas for data analysis purpose

Pandas in Jupyter Notebook

Launch a new Jupyter Notebook

- type the following commands to import the necessary library for this exercise

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Datasets

There are numerous datasets packages in Seaborn

```
sns.get_dataset_names () # show the dataset available in Seaborn
```

```
['anagrams',
 'anscombe',
 'attention',
 'brain_networks',
 'car_crashes',
 'diamonds',
 'dots',
 'dowjones',
 'exercise',
 'flights',
 'fmri',
 'geyser',
 'glue',
 'healthexp',
 'iris',
 'mpg',
 'penguins',
 'planets',
 'seaice',
 'taxis',
 'tips',
 'titanic']
```

‘tips’ Dataset

This is a very small dataset

- with 244 rows and 7 variables
- the tips received by a single waiter in a single restaurant over a few months

i. the **tip** received in dollars
ii. the **bill** in dollars
iii. the **sex** of the bill payer
iv. whether there were **smokers** in the party
v. the **day** of the week
vi. the **time** of day
vii. the **size** of the party

Loading the Dataset

We can load the dataset directly in Seaborn using its load_dataset function

- which is stored in Pandas dataframe format by default

```
df = sns.load_dataset ('tips')
```

Alternatively, you can load its csv file from its url (in github) using Pandas

```
csv_url = 'https://raw.githubusercontent.com/mwaskom/seaborn-data/master/tips.csv'
df =  pd.read_csv(csv_url)
```

Dataset Contents

Viewing the dataset

```
df.head ()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
df.tail ()
```

	total_bill	tip	sex	smoker	day	time	size
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

```
df.index
```

```
RangeIndex(start=0, stop=244, step=1)
```

Dataset Checking

Check the data

- any missing value(s)?

```
print(df.isna().any())
```

```
total_bill    False
tip           False
sex           False
smoker        False
day           False
time          False
size          False
dtype: bool
```

Data types

Check the datatyp

```
df.dtypes
```

```
total_bill       float64
tip              float64
sex             category
smoker          category
day             category
time            category
size               int64
dtype: object
```

- three numerical columns and 4 non-numerica columns.

i. Sex is a binary category type (male/female)
ii. Smoker is a binary category type (Yes/No)
iii. Day is a category type (Thur to Sun)
iv. Time is a binary category type (Lunch/Dinner)

View Data Entries

Sort the data

- by tip amount

```
df.sort_values(by='tip').head()
```

	total_bill	tip	sex	smoker	day	time	size
67	3.07	1.00	Female	Yes	Sat	Dinner	1
236	12.60	1.00	Male	Yes	Sat	Dinner	2
92	5.75	1.00	Female	Yes	Fri	Dinner	2
111	7.25	1.00	Female	No	Sat	Dinner	1
0	16.99	1.01	Female	No	Sun	Dinner	2

- By total bill amount

```
df.sort_values(by='total_bill', ascending = False).head(3)
```

	total_bill	tip	sex	smoker	day	time	size
170	50.81	10.00	Male	Yes	Sat	Dinner	3
212	48.33	9.00	Male	No	Sat	Dinner	4
59	48.27	6.73	Male	No	Sat	Dinner	4

Exploratory Data Analysis

Exploratory Data Analysis (non-graphical using Pandas functions)

Summary statistics of

```
df.describe()
```

	total_bill	tip	size
count	244.000000	244.000000	244.000000
mean	19.785943	2.998279	2.569672
std	8.902412	1.383638	0.951100
min	3.070000	1.000000	1.000000
25%	13.347500	2.000000	2.000000
50%	17.795000	2.900000	2.000000
75%	24.127500	3.562500	3.000000
max	50.810000	10.000000	6.000000

Exploratory Data Analysis

Summary statistics of the category values

```
df.describe(include=['category'])
```

	sex	smoker	day	time
count	244	244	244	244
unique	2	2	4	2
top	Male	No	Sat	Dinner
freq	157	151	87	176

Some common Statistics

- Average Bills

```
print(f"Average of total_bill = {df['total_bill'].mean():6.4}")
```

- Median of Bills

```
print(f"Median of total_bill = {df['total_bill'].quantile(q=0.5):4.2f}")
```

- Average Tips

```
print(f"Average of tip = {df['tip'].mean():.2f}")
```

- Median Tips

```
print(f"Median of tip = {df['tip'].quantile(q=0.5):.2f}")
```

More Statistics

- Variance

```
print(f"Variance of total_bill = {df['total_bill'].var():.2f}")
```

- Standard deviation

```
print(f"Standard Deviation of total_bill = {df['total_bill'].std():.2f}")
```

- Minimum and maximum tips

```
print(f"Min and Max tip = {df['tip'].min()} and {df['tip'].max()}")
```

Seaborn Package for Data Visualization

We will now use seaborn package to create some visualisations of the dataset

Countplot

- show the number of occurrences in a category of a variable

```
fig, ax = plt.subplots() # using matplotlib to set up a plot
ax.set_title("Count of tables served by Day") # add a title
```

```
days=["Thur", "Fri", "Sat","Sun"]
sns.countplot(x ="day",hue="time", order=days, data =df) #use sns
```

```
days=["Thur", "Fri", "Sat","Sun"]
sns.countplot(x ="day",hue="time", order=days, data =df, palette=["green","yellow"])
```

Seaborn Package for Data Visualization

Countplot

- show the number of occurrences in a category of a variable

```
fig, axes = plt.subplots(1, 2, figsize=(15, 4)) # set up 1 by 2 plot with size of 15 by 4
days=["Thur", "Fri", "Sat","Sun"]

sns.countplot(x =("day"), hue="size",data =df,  order=days, ax=axes[0])
axes[0].set_title("Tables served by Day and by party size");

sns.countplot(x ="day",hue="sex", order=days, palette=dict(Female="pink", Male="lightblue"),
              data =df, ax=axes[1]) #use sns
axes[1].set_title("Bill Payer")
```

Seaborn Package for Data Visualization

Histplot

- shows the distribution of a single quantitative variable (number of occurrences) in a category of a variable

```
fig, ax = plt.subplots()
ax.set_title("Histogram of Total Bill amounts")

sns.histplot(df['total_bill'], color="lightgreen", bins=30)

# add a vertical line at the mean
ax.axvline(df['total_bill'].mean(), color='red', linewidth=2, linestyle="--")

# add a vertical line at the median
ax.axvline(df['total_bill'].quantile(q=0.5), color='blue', linewidth=2, linestyle=":")
```

Seaborn Package for Data Visualization

Histplot

```
fig, ax = plt.subplots()
ax.set_title("Histogram of  Tip amounts");

sns.histplot(df['tip'], kde = True,  color="blue", bins=30)

ax.axvline(df['tip'].mean(), color='red', linewidth=2, linestyle="--")

ax.axvline(df['tip'].quantile(q=0.5), color='yellow', linewidth=2, linestyle=":")
```

Seaborn Package for Data Visualization

Boxplot

- shows the central tendency and outliers
- bounded by a box representing the 1st quartile and 3rd quartile
- line drawn through the box represents the median.

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

sns.boxplot(y=df['total_bill'], ax=axes[0])
axes[0].set_title("Boxplot of Total Bill amount")

sns.boxplot(y=df['tip'], ax=axes[1], color="green")
axes[1].set_title("Boxplot of Tips amounts");
```

Seaborn Package for Data Visualization

Boxplot

- By day

```
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

sns.boxplot(x="day",y="total_bill" ,data=df, order=days, ax=axes[0])
axes[0].set_title("Boxplot of total bill amount by day")

gender_pal=dict(Female="pink",Male="skyblue")
sns.boxplot(x="day",y="total_bill" ,hue="sex",data=df,
            palette=gender_pal,order=days, ax=axes[1])
axes[1].set_title("Total bill amount by Male/Female")
```

Seaborn Package for Data Visualization

Pairplot

- show all pairwise relationships of the (variables) in a dataset.

```
sns.pairplot(df)
```

Note: the diagonal entries show the histogram/distribution of the entries

- Can be more specific:

```
sns.pairplot(df, hue="sex", palette=gender_pal)
```

'hue' map use the variable in data to map plot aspects to different colors.