Python Basics - Individual

Graded

Student

Anand Patel

Total Points

84.5 / 100 pts

Autograder Score

3.0 / 5.0

Failed Tests

test_data_dir (test_files.TestFiles) (0/1)
test_list_files (test_files.TestFiles) (2/3)

Passed Tests

test_zip (test_files.TestFiles) (1/1)

Question 2

1.2 - for loop — **1** / 1 pt

✔ **– 0 pts** Correct

Question 3

1.3 - verify solution — **1** / 1 pt

✔ **– 0 pts** Correct

Question 4

1.4 - efficiency check — **1** / 1 pt

✔ **– 0 pts** Correct

Question 5

1.5 - discussion — **0.5** / 1 pt

✔ **– 0.5 pts** Missing discussion of design details.

Question 6

1.6 - design, implement, verify (less efficient) — **2** / 3 pts

✔ **– 1 pt** Verification unclear.

Question 7

1.7 - discussion (less efficient) — **1** / 1 pt

✔ **– 0 pts** Good discussion.

Question 8

1.8 - Comparing Implementations — **1** / 2 pts

✔ **– 1 pt** Minimal explanation provided.

Question 9

2.0.1 - Import — **2** / 2 pts

✔ **– 0 pts** Correct

Question 10

2.0.2 - Load — **1** / 1 pt

✔ **– 0 pts** Correct

Question 11

2.0.3 - Display data — **1** / 1 pt

✔ **– 0 pts** Correct

Question 12

2.0.4 - Display data information **1** / 1 pt

✔ **− 0 pts** Correct

Question 13

2.1.1 - Remove columns **1** / 2 pts

✔ **− 1 pt** No statement

Question 14

2.1.2 - Rename columns **1** / 2 pts

✔ **− 1 pt** No statement

Question 15

2.1.3 - Remove duplicated rows **3** / 3 pts

✔ **− 0 pts** Correct

Question 16

2.1.4 - Null values **2** / 2 pts

✔ **− 0 pts** Correct

Question 17

2.1.5 - Handling missing values **3** / 3 pts

✔ **− 0 pts** Correct

Question 18

2.1.6 - Reasoning **0** / 3 pts

✔ **− 3 pts** have not answered the question

Question 19

2.2.1 - Add HP_Type **2** / 2 pts

✔ **− 0 pts** Correct

Question 20

2.2.2 - Add Price_class **2** / 2 pts

✔ **− 0 pts** Correct

Question 21

2.2.3 - Save modified data **1** / 1 pt

✔ **− 0 pts** Correct

Question 22

2.3.1 - Mean price **1** / 1 pt

✔ **− 0 pts** Correct

Question 23

2.3.2 - Count cars by year **1** / 1 pt

✔ **− 0 pts** Correct

Question 24

2.3.3 - Unique Makes **1** / 1 pt

✔ **− 0 pts** Correct

Question 25

2.3.4 - Fuel efficiency **1.5** / 2 pts

✔ **− 0.5 pts** Report the number of unique vehicle makers.

Question 26

2.3.5 - Cross-tabulation **2** / 2 pts

✔ **− 0 pts** Correct

Question 27

2.3.6 - Isolating specific features **1** / 1 pt

✔ **− 0 pts** Correct

Question 28

2.3.7 - Isolating specific features **2** / 2 pts

✔ **− 0 pts** Correct

Question 29

2.3.8 - Counting specific features **2** / 2 pts

✔ **− 0 pts** Correct

Question 30

2.3.9 - Grouping data **1.5** / 2 pts

✔ **− 0.5 pts** You should display the values as integers

Question 31

2.3.10 - Reporting extreme values **1** / 1 pt

✔ **− 0 pts** Correct

Question 32

3.1 - Prepare data **5** / 5 pts

✔ **− 0 pts** Correct formulations.

Question 33

3.2 - Define a Tournament class **26** / 30 pts

✔ **− 2 pts** A more optimal selection of cars could be guaranteed found by implementation of the classical 0/1 knapsack problem (as opposed to fractional or greedy solution).

✔ **− 1 pt** Winners should be permitted to `_purchase_inventory` again between matches.

✔ **− 1 pt** Missing full set of complete docstrings.

Question 34

3.3 - Execute your Tournament class **5** / 5 pts

✔ **− 0 pts** Good execution demonstration.

Question 35

3.4 - Compare the results of multiple Tournament executions **4** / 5 pts

✔ **+ 4 pts** This could be written more generally, to operate on any number of Tournaments. You might even overload a comparison operator to compare instances of Tournaments.

Autograder Results

test_data_dir (test_files.TestFiles) (0/1)

Test Failed: False is not true : Missing data directory.

test_list_files (test_files.TestFiles) (2/3)

Found *CS2PP22_CW1.ipynb
Missing *CS2PP22_CW1.html
Found *cardata_modified.csv

test_zip (test_files.TestFiles) (1/1)

Submitted Files

▾ CS2PP22_CW1.ipynb Download

Front page of the student's submission (the following are compulsory):

Module Code: CS2PP22

Assignment report Title: CW1 Individual

Date (when the work completed): 16/02/2024

Actual hrs spent for the assignment: 20

CS2PP22 Programming in Python for Data Science

Instructions:

- Write Python code to perform each of the following sub-tasks.
  - Try to follow the PEP 8 – Style Guide for Python Code: https://peps.python.org/pep-0008
  - Function and variable type annotations are not required.

- Some parts of this assignment may require further self-study of Python documentations or other resources.

- You may also refer to other documentation/self-study resources, such as those suggested in the Lecture Notes or a multitude of other resources

- Blank code and markdown cells are provided for each sub-task, however you will likely need to create additional cells to provide further explana

Items to be submitted: 1. A modified version of this Jupyter notebook file (`.ipynb`) - This is to be submitted already **fully executed** i

4. `cardata_modified.csv`

6. $*Optional*: Anyfunctionsandclassescreatedfor this Task may be written in a separate`.py`module file and`import`ed to this space.T$

10 marks

- A **network** or **graph**, G, consists of a set of **nodes** (or vertices), V, and edges, E.

- An edge is a pair of nodes (a,b) denoting the nodes connected by the edge. $$ G = (V,E) $$ `![CompareNetworks.png](images/CompareNetworks.png)` - Networks can be represented in various different structures.

- We can represent networks using a simple list of all the nodes and all the edges in the network:

$$V = a, b, c, d, e$$

Edges.png

- Another data structure that we could use is to build a **neighbour list** for every node.

- For every node, x, we maintain a list of all the neighbours, y. `![Neighbours.png](images/Neighbours.png)`

Data Reference:

The file `data/dolphins.tsv` contains a representation of a social network dataset where dolphins have links between them if they frequently associated with one another.

- Taken from the Koblenz Network Collection by the University of Koblenz–Landau. Dolphins network dataset – KONECT, April 2017. http://konect.cc/networks/dolphins/.

To read in the *tab separated* network file, we need to read each line in the file. To do this we just treat the file object as an iterator.

- This is an example of using a context manager: https://book.pythontips.com/en/latest/context_managers.html

- Here `line` will be a string for each line in the file. The `.split(x)` method splits a string into a list of substrings for each occurrence of the character `x` (in this case the tab: `\t`). This code reads the file and places the data in an edge list representation: ```python # Create an empty edge set edges = set() with open('data/dolphins.tsv', 'r') as file: for line in file: a, b = line.split('\t') e = (int(a), int(b)) edges.add(e) ``` --- **Instructions:** 1. Begin with the resulting `edges` variable.$$$$ 2. [*1 mark*] Use a `for` loop to populate a `dict` that will contain the **neighbour list** network representation.$$$$ 3. [*1 mark*] **Verify** that your solution matches that for nodes 55, 2, and 20 (shown below).$$$$ 4. [*1 mark*] With your entire neighbour list code in a single Jupyter notebook cell, use the `timeit` Magic command to report the performance of your code, executing **19 runs** of **2000 loops** each. - There should be **no network data output from this cell**.$$$$ 5. [*1 mark*] Discuss your solution, describing the Python constructs/tools you have used, and the design of your implementation.$$$$ 6. [*3 marks*] Design, implement, and verify the results of a **second, less efficient** implementation and record its equivalent performance metrics as above.$$$$ 7. [*1 mark*] Discuss your **second** solution, describing the Python constructs/tools you have used, and the design of your less efficient implementation.$$$$ 8. [*2 marks*] Explain why the efficiency differs between your two implementations. --- There will be many nodes in the final dataset. The result should have the form: ```python {55: {2, 7, 8, 14, 20, 42, 58}, 2: {18, 20, 27, 28, 29, 37, 42, 55}, 20: {2, 8, 31, 55},... ``` This shows that dolphin `55` is frequently associated with dolphins `2, 7, 8, 14, 20, 42, and 58`.

In [8]:
```
# Create an empty edge set
edges = set()

with open('dolphins.tsv', 'r') as file:
    for line in file:
        a, b = line.split('\t')
        e = (int(a), int(b))
        edges.add(e)
```

In [31]:
```
%%timeit -r 19 -n 2000
network = {}
for edge in edges:
    a, b = edge
    if a not in network:
        network[a] = set()
    if b not in network:
        network[b] = set()
    network[a].add(b)
    network[b].add(a)
```

```
59.6 µs ± 3.96 µs per loop (mean ± std. dev. of 19 runs, 2000 loops each)
```

In [36]:
```
# Verifying my more efficient solution
nodes = [55, 2, 20]
for node in nodes:
  print(node, network[node])
```

```
55 {2, 7, 8, 42, 14, 20, 58}
2 {37, 42, 18, 20, 55, 27, 28, 29}
20 {8, 2, 31, 55}
```

In [32]:
```
```

Discussion: This approach takes advantage of the speed of dictionaries and the uniqueness property of sets so that each neighbor is listed once for each node This prevents redundant/uncessesary checks

In [38]:
```
%%timeit -r 19 -n 2000
network = {}
for a, b in edges:
    if isinstance(network.get(a), set):
        network[a].add(b)
    else:
        network[a] = {b}
    if isinstance(network.get(b), set):
        network[b].add(a)
```

```
    else:
        network[b] = {a}
```

83.3 µs ± 3.73 µs per loop (mean ± std. dev. of 19 runs, 2000 loops each)

In [ ]:

```

```

Discussion: dict.get is less efficient than the first solution as it will always return a value as it does a check if the key exists within the set and isinstance does a type check of the value. These 2 extra steps for each edge will increase the overhead compared to the first solution

---

Data Reference:

- Car Features and MSRP Data: `cardata.csv`

$-Thisdatasetincludescarfeaturessuchasmake, model, year, andenginetype, asscrapedfromEdmundsandTwitter.Itisoftenusedto$

- Source: https://www.kaggle.com/datasets/CooperUnion/cardataset

$-Each**row**correspondstoasinglekindofvehicle.$

- The **columns** correspond to:

Column	Description
Make	Car maker
Model	Car model
Year	Car year (Marketing)
Engine Fuel Type	Type of engine fuel category
Engine HP	Engine horsepower (HP)
Engine Cylinders	Number of engine cylinders
Transmission Type	Type of transmission category
Driven_Wheels	Drive wheel category
Number of Doors	Number of doors
Market Category	Market category
Vehicle Size	Vehicle size category
Vehicle Style	Vehicle style category
highway MPG	Highway fuel efficiency in miles per gallon

Column	Description
city mpg	City fuel efficiency in miles per gallon
Popularity	Twitter-based popularity metric
MSRP	Manufacturer suggested retail price (USD)

2.0. Analysis Preparation

5 Marks

2.0.1. Import 2 marks

Import the libraries that will be used in your solution.

In [3]:

```
import numpy as np
import pandas as pd
```

2.0.2. Load data 1 mark

Locate the data file `cardata.csv` within the zipped file you have downloaded from Blackboard (under `./data/Task1/`) and read the data from file to a `pandas` `DataFrame`.

In [4]:

```
cars = pd.read_csv('cardata.csv')
```

2.0.3. Display data 1 mark

Display the first 5 rows of the `DataFrame`.

In [5]:

```
cars.head()
```

Out [5]:

```
  Make       Model  Year             Engine Fuel Type  Engine HP  \
0  BMW  1 Series M  2011  premium unleaded (required)      335.0   
1  BMW    1 Series  2011  premium unleaded (required)      300.0   
2  BMW    1 Series  2011  premium unleaded (required)      300.0   
3  BMW    1 Series  2011  premium unleaded (required)      230.0   
4  BMW    1 Series  2011  premium unleaded (required)      230.0   

   Engine Cylinders Transmission Type     Driven_Wheels  Number of Doors  \
0               6.0            MANUAL  rear wheel drive              2.0   
1               6.0            MANUAL  rear wheel drive              2.0   
2               6.0            MANUAL  rear wheel drive              2.0   
3               6.0            MANUAL  rear wheel drive              2.0   
4               6.0            MANUAL  rear wheel drive              2.0   

                         Market Category Vehicle Size Vehicle Style  \
0  Factory Tuner,Luxury,High-Performance      Compact         Coupe   
1                     Luxury,Performance      Compact   Convertible   
2                Luxury,High-Performance      Compact         Coupe   
3                     Luxury,Performance      Compact         Coupe   
4                                 Luxury      Compact   Convertible   

   highway MPG  city mpg  Popularity   MSRP  
0           26        19        3916  46135  
1           28        19        3916  40650  
2           28        20        3916  36350  
3           28        18        3916  29450  
4           28        18        3916  34500  
```

2.0.4. Display data information 1 mark

In one or more code cells, display the following information of the `DataFrame`:

- number of rows
- number of columns
- column names and data types

Following the display of this information, use a Markdown cell to write:

- one sentence summarising the numbers of rows and columns,
- and one sentence describing the unique data types.

In [6]:

```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               11914 non-null  object 
 1   Model              11914 non-null  object 
 2   Year               11914 non-null  int64  
 3   Engine Fuel Type   11911 non-null  object 
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object 
 7   Driven_Wheels      11914 non-null  object 
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object 
 10  Vehicle Size       11914 non-null  object 
 11  Vehicle Style      11914 non-null  object 
 12  highway MPG        11914 non-null  int64  
 13  city mpg           11914 non-null  int64  
 14  Popularity         11914 non-null  int64  
 15  MSRP               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

This `DataFrame` contains 11,914 rows and 16 columns.

The data types present include strings (recorded as `object`), 64-bit integers (`int64`), and 64-bit floating point values (`float64`).

2.1. Data Cleaning

15 Marks

2.1.1. Remove columns 2 marks

Drop the following columns from the `DataFrame`:

- Engine Fuel Type
- Market Category
- Number of Doors
- Vehicle Size

Then, **display** the resulting `DataFrame` and **write a statement**, explaining why one might choose to exclude these features.

In [7]:

```
cars = cars.drop(columns=['Engine Fuel Type', 'Market Category','Number of Doors', 'Vehicle Size'])
cars
```

Out [7]:

```
      Make       Model  Year  Engine HP  Engine Cylinders  \
0      BMW  1 Series M  2011      335.0               6.0   
1      BMW    1 Series  2011      300.0               6.0   
2      BMW    1 Series  2011      300.0               6.0   
3      BMW    1 Series  2011      230.0               6.0   
```

```
4         BMW    1 Series  2011      230.0              6.0
...        ...        ...   ...        ...              ...
11909    Acura        ZDX  2012      300.0              6.0
11910    Acura        ZDX  2012      300.0              6.0
11911    Acura        ZDX  2012      300.0              6.0
11912    Acura        ZDX  2013      300.0              6.0
11913  Lincoln     Zephyr  2006      221.0              6.0

      Transmission Type      Driven_Wheels  Vehicle Style  highway MPG  \
0                MANUAL   rear wheel drive          Coupe           26
1                MANUAL   rear wheel drive    Convertible           28
2                MANUAL   rear wheel drive          Coupe           28
3                MANUAL   rear wheel drive          Coupe           28
4                MANUAL   rear wheel drive    Convertible           28
...                 ...                ...            ...          ...
11909         AUTOMATIC    all wheel drive  4dr Hatchback           23
11910         AUTOMATIC    all wheel drive  4dr Hatchback           23
11911         AUTOMATIC    all wheel drive  4dr Hatchback           23
11912         AUTOMATIC    all wheel drive  4dr Hatchback           23
11913         AUTOMATIC  front wheel drive          Sedan           26

       city mpg  Popularity   MSRP
0            19        3916  46135
1            19        3916  40650
2            20        3916  36350
3            18        3916  29450
4            18        3916  34500
...         ...         ...    ...
11909        16         204  46120
11910        16         204  56670
11911        16         204  50620
11912        16         204  50920
11913        17          61  28995

[11914 rows x 12 columns]
```

2.1.2. Rename columns 2 marks

Rename the following columns:

Column	New name
Engine HP	HP
Engine Cylinders	Cylinders
Transmission Type	Transmission
Driven_Wheels	Drive Mode
highway MPG	MPG-H
city mpg	MPG-C
MSRP	Price

Then, **display** the resulting `DataFrame` and **write a statement**, discussing why one might change the names in this way.

In [8]:

```
cars = cars.rename(columns = {'Engine HP':'HP','Engine Cylinders':'Cylinders','Transmission Type':'Transmission','Driven_Wheels':'Drive Mode','highway MPG':'MPG-H','city mpg':'MPG-C','MSRP':'Price'})
cars
```

Out [8]:
```
         Make       Model  Year     HP  Cylinders Transmission  \
0         BMW  1 Series M  2011  335.0        6.0       MANUAL
1         BMW    1 Series  2011  300.0        6.0       MANUAL
2         BMW    1 Series  2011  300.0        6.0       MANUAL
3         BMW    1 Series  2011  230.0        6.0       MANUAL
4         BMW    1 Series  2011  230.0        6.0       MANUAL
...       ...         ...   ...    ...        ...          ...
11909   Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11910   Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11911   Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11912   Acura         ZDX  2013  300.0        6.0    AUTOMATIC
11913  Lincoln     Zephyr  2006  221.0        6.0    AUTOMATIC

              Drive Mode  Vehicle Style  MPG-H  MPG-C  Popularity  Price
0        rear wheel drive          Coupe     26     19        3916  46135
1        rear wheel drive    Convertible     28     19        3916  40650
2        rear wheel drive          Coupe     28     20        3916  36350
3        rear wheel drive          Coupe     28     18        3916  29450
4        rear wheel drive    Convertible     28     18        3916  34500
...                  ...            ...    ...    ...         ...    ...
11909     all wheel drive  4dr Hatchback     23     16         204  46120
11910     all wheel drive  4dr Hatchback     23     16         204  56670
11911     all wheel drive  4dr Hatchback     23     16         204  50620
11912     all wheel drive  4dr Hatchback     23     16         204  50920
11913   front wheel drive          Sedan     26     17          61  28995

[11914 rows x 12 columns]
```

2.1.3. Remove duplicated rows 3 marks

Use an `f-string` to `print()` the number of irrelevant duplicate rows (count repeats only, not both originals and repeats).

Drop the duplicated rows, retain the resulting `DataFrame`, and show the resulting number of rows in the new `DataFrame`.

In [9]:
```
print(f"Duplicate rows: {cars.duplicated().sum()}")
cars = cars.drop_duplicates()
print(f"Rows after dropping duplicates: {len(cars)}")
```

```
Duplicate rows: 803
Rows after dropping duplicates: 11111
```

2.1.4. Null values 2 marks

Report the number of null values remaining in each column.

In [10]:
```
null_counts = cars.isnull().sum()

print(null_counts)
```

```
Make              0
Model             0
Year              0
HP               69
Cylinders        30
Transmission      0
Drive Mode        0
Vehicle Style     0
MPG-H             0
MPG-C             0
Popularity        0
Price             0
dtype: int64
```

2.1.5. Handling missing values 3 marks

- For the HP column, replace missing values with the column mean.
- Drop all other rows that still contain missing values.
- Display the final number of rows in the dataframe.

In [11]:

```
hp_mean = np.mean(cars.HP)
cars = cars.fillna(value={"HP": hp_mean})
cars.isnull().sum()
```

Out [11]:

```
Make              0
Model             0
Year              0
HP                0
Cylinders        30
Transmission      0
Drive Mode        0
Vehicle Style     0
MPG-H             0
MPG-C             0
Popularity        0
Price             0
dtype: int64
```

In [12]:

```
cars = cars.dropna()

print('Rows: ', cars.shape[0])
```

```
Rows:  11081
```

2.1.6. Reasoning 3 marks

In the previous step, we eliminated entries with missing data from the dataset. If this dataset were to be used to train a machine learning model to predict vehicle prices, what is a potential drawback of this approach? Describe an alternative approach and any related caveats of which we should be aware.

2.2. Creating New Columns

5 Marks

2.2.1. Add HP_Type 2 marks

Using an implementation of **list comprehension**, create a new column, HP_Type, such that,

- if a car's HP is greater than or equal to 300, HP_Type is 'high'.
- Otherwise, HP_Type is 'low'.

Then, display the resulting DataFrame.

In [13]:

```
cars['HP_Type'] = ['high' if hp >= 300 else 'low' for hp in cars['HP']]
cars
```

Out [13]:

```
        Make      Model  Year     HP  Cylinders Transmission  \
0        BMW  1 Series M  2011  335.0        6.0       MANUAL
1        BMW   1 Series  2011  300.0        6.0       MANUAL
2        BMW   1 Series  2011  300.0        6.0       MANUAL
3        BMW   1 Series  2011  230.0        6.0       MANUAL
4        BMW   1 Series  2011  230.0        6.0       MANUAL
...      ...        ...   ...    ...        ...          ...
11909  Acura        ZDX  2012  300.0        6.0    AUTOMATIC
11910  Acura        ZDX  2012  300.0        6.0    AUTOMATIC
11911  Acura        ZDX  2012  300.0        6.0    AUTOMATIC
11912  Acura        ZDX  2013  300.0        6.0    AUTOMATIC
```

```
11913  Lincoln     Zephyr  2006  221.0       6.0    AUTOMATIC

            Drive Mode  Vehicle Style  MPG-H  MPG-C  Popularity  Price  \
0      rear wheel drive         Coupe     26     19        3916  46135
1      rear wheel drive   Convertible     28     19        3916  40650
2      rear wheel drive         Coupe     28     20        3916  36350
3      rear wheel drive         Coupe     28     18        3916  29450
4      rear wheel drive   Convertible     28     18        3916  34500
...                 ...           ...    ...    ...         ...    ...
11909    all wheel drive  4dr Hatchback     23     16         204  46120
11910    all wheel drive  4dr Hatchback     23     16         204  56670
11911    all wheel drive  4dr Hatchback     23     16         204  50620
11912    all wheel drive  4dr Hatchback     23     16         204  50920
11913  front wheel drive         Sedan     26     17          61  28995

      HP_Type
0        high
1        high
2        high
3         low
4         low
...       ...
11909    high
11910    high
11911    high
11912    high
11913     low

[11081 rows x 13 columns]
```

2.2.2. Add `Price_class` 2 marks

Using an implementation of a **function**, create a new column, `Price_class`, such that it becomes equal to:

- 'high', if `Price` is greater than or equal to 50,000
- 'mid', if `Price` is between 30,000 (inclusive) and 50,000 (exclusive), and
- 'low', if `Price` is below 30,000.

Then, display the resulting `DataFrame`.

In [14]:

```python
def price_category(price):
    if price >= 50000:
        return 'high'
    elif price >= 30000:
        return 'mid'
    else:
        return 'low'

cars['Price_class'] = cars['Price'].apply(price_category)
cars
```

Out [14]:

```
          Make       Model  Year     HP  Cylinders Transmission  \
0          BMW  1 Series M  2011  335.0        6.0       MANUAL
1          BMW    1 Series  2011  300.0        6.0       MANUAL
2          BMW    1 Series  2011  300.0        6.0       MANUAL
3          BMW    1 Series  2011  230.0        6.0       MANUAL
4          BMW    1 Series  2011  230.0        6.0       MANUAL
...        ...         ...   ...    ...        ...          ...
11909    Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11910    Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11911    Acura         ZDX  2012  300.0        6.0    AUTOMATIC
11912    Acura         ZDX  2013  300.0        6.0    AUTOMATIC
11913  Lincoln      Zephyr  2006  221.0        6.0    AUTOMATIC

            Drive Mode  Vehicle Style  MPG-H  MPG-C  Popularity  Price  \
0      rear wheel drive         Coupe     26     19        3916  46135
1      rear wheel drive   Convertible     28     19        3916  40650
2      rear wheel drive         Coupe     28     20        3916  36350
3      rear wheel drive         Coupe     28     18        3916  29450
4      rear wheel drive   Convertible     28     18        3916  34500
```

```
...                ...         ...   ...   ...       ...    ...
11909   all wheel drive  4dr Hatchback    23    16       204  46120
11910   all wheel drive  4dr Hatchback    23    16       204  56670
11911   all wheel drive  4dr Hatchback    23    16       204  50620
11912   all wheel drive  4dr Hatchback    23    16       204  50920
11913  front wheel drive          Sedan    26    17        61  28995

      HP_Type Price_class
0        high         mid
1        high         mid
2        high         mid
3         low         low
4         low         mid
...       ...         ...
11909    high         mid
11910    high        high
11911    high        high
11912    high        high
11913     low         low

[11081 rows x 14 columns]
```

2.2.3 Save modified data 1 mark

Save the modified `DataFrame` to a new comma-separated file called `cardata_modified.csv` to be stored under the `data/Task1/` directory.

Do not include the row indices in the file.

In [15]:

```
cars.to_csv('cardata_modified.csv', index=False)
```

2.3. Exploratory Data Analysis

15 Marks

2.3.1. Mean price 1 mark

Find the mean `Price` of all vehicles. Report the solution rounded to 2 decimal places.

In [16]:

```
round(cars['Price'].mean(), 2)
```

Out [16]:

```
41957.25
```

2.3.2. Count cars by year 1 mark

Report the number of cars in each `Year`.

In [17]:

```
cars['Year'].value_counts()
```

Out [17]:

```
2015    2048
2016    2046
2017    1598
2014     542
2009     356
2012     350
2007     334
2013     325
2008     322
2011     279
2010     276
2003     238
2004     235
2005     213
2002     205
```

```
2006    194
2001    168
1997    162
1993    159
1998    144
1992    127
1994    125
2000    115
1995    114
1999    114
1996    113
1991    102
1990     77
Name: Year, dtype: int64
```

2.3.3. Unique Makes [1 mark]

Report a list of unique values of `Make`, sorted in ascending alphabetical order.

In [18]:

```
sorted(list(set(cars.Make)))
```

Out [18]:

```
['Acura',
 'Alfa Romeo',
 'Aston Martin',
 'Audi',
 'BMW',
 'Bentley',
 'Bugatti',
 'Buick',
 'Cadillac',
 'Chevrolet',
 'Chrysler',
 'Dodge',
 'FIAT',
 'Ferrari',
 'Ford',
 'GMC',
 'Genesis',
 'HUMMER',
 'Honda',
 'Hyundai',
 'Infiniti',
 'Kia',
 'Lamborghini',
 'Land Rover',
 'Lexus',
 'Lincoln',
 'Lotus',
 'Maserati',
 'Maybach',
 'Mazda',
 'McLaren',
 'Mercedes-Benz',
 'Mitsubishi',
 'Nissan',
 'Oldsmobile',
 'Plymouth',
 'Pontiac',
 'Porsche',
 'Rolls-Royce',
 'Saab',
 'Scion',
 'Spyker',
 'Subaru',
 'Suzuki',
 'Tesla',
 'Toyota',
 'Volkswagen',
 'Volvo']
```

2.3.4. Fuel efficiency 2 marks

Report the number of unique vehicle makers.

Display the mean `MPG-H` and mean `MPG-C` for each `Model` of each `Make` in a `MultiIndex` `DataFrame`. Report the values rounded to 2 decimal places.

Verify that the expected number of vehicle makers are represented in the resulting `DataFrame`.

In [19]:

```
cars.Make.unique().size
mean_mpg = cars.groupby(['Make', 'Model'])[['MPG-H', 'MPG-C']].mean().round(2)
mean_mpg
```

Out [19]:

```
                MPG-H  MPG-C
Make  Model
Acura CL        26.78  17.00
      ILX       35.12  24.62
      ILX Hybrid  38.00  39.00
      Integra   28.17  21.83
      Legend    23.19  16.00
...               ...    ...
Volvo V90       23.00  16.00
      XC        23.00  17.00
      XC60      27.13  19.87
      XC70      27.50  20.17
      XC90      25.29  20.47

[923 rows x 2 columns]
```

2.3.5. Cross-tabulation 2 marks

Display in a single `DataFrame` the number of car entries for each `Drive Mode` and `Transmission` combination, with a `Total` column (showing the sum of the rows) and a `Total` row (showing the sum of the columns) in the `margins`.

In [20]:

```
pd.crosstab(index=cars['Drive Mode'], columns=cars['Transmission'], margins=True, margins_name='Total')
```

Out [20]:

```
Transmission       AUTOMATED_MANUAL  AUTOMATIC  DIRECT_DRIVE  MANUAL  UNKNOWN  \
Drive Mode
all wheel drive                 198       1897            11     202        0
four wheel drive                  0        979             0     291        2
front wheel drive               231       2861            36    1209        4
rear wheel drive                124       2101            11     918        6
Total                           553       7838            58    2620       12

Transmission       Total
Drive Mode
all wheel drive     2308
four wheel drive    1272
front wheel drive   4341
rear wheel drive    3160
Total              11081
```

2.3.6. Isolating specific features 1 mark

Display data corresponding to the `Year` 2017 for Porsche's Macan model.

In [21]:

```
cars[(cars['Year'] == 2017) & (cars['Make'] == 'Porsche') & (cars['Model'] == 'Macan')]
```

Out [21]:

```
         Make  Model  Year     HP  Cylinders      Transmission  \
6630  Porsche  Macan  2017  360.0        6.0  AUTOMATED_MANUAL
6631  Porsche  Macan  2017  340.0        6.0  AUTOMATED_MANUAL
6632  Porsche  Macan  2017  400.0        6.0  AUTOMATED_MANUAL
6633  Porsche  Macan  2017  252.0        4.0  AUTOMATED_MANUAL
```

```
       Drive Mode Vehicle Style  MPG-H  MPG-C  Popularity  Price HP_Type  \
6630  all wheel drive       4dr SUV     23     17        1715  67200    high
6631  all wheel drive       4dr SUV     23     17        1715  54400    high
6632  all wheel drive       4dr SUV     23     17        1715  76000    high
6633  all wheel drive       4dr SUV     25     20        1715  47500     low

     Price_class
6630        high
6631        high
6632        high
6633         mid
```

2.3.7. Isolating specific features [2 marks]

Display data corresponding to the `Year` 2015 for makers FIAT and Scion that have entries with automatic `Transmission` in a **single** `DataFrame`.

In [22]:

```
cars[(cars['Year'] == 2015) & ((cars['Make'] == 'FIAT') | (cars['Make'] == 'Scion')) & (cars.Transmission == 'AUTOMATIC') ]
```

Out [22]:

```
        Make Model  Year     HP  Cylinders Transmission         Drive Mode  \
572     FIAT  500L  2015  160.0        4.0    AUTOMATIC  front wheel drive
4897   Scion  FR-S  2015  200.0        4.0    AUTOMATIC   rear wheel drive
4898   Scion  FR-S  2015  200.0        4.0    AUTOMATIC   rear wheel drive
5962   Scion    iQ  2015   94.0        4.0    AUTOMATIC  front wheel drive
10312  Scion    tC  2015  179.0        4.0    AUTOMATIC  front wheel drive
10313  Scion    tC  2015  179.0        4.0    AUTOMATIC  front wheel drive
11548  Scion    xB  2015  158.0        4.0    AUTOMATIC  front wheel drive
11549  Scion    xB  2015  158.0        4.0    AUTOMATIC  front wheel drive

      Vehicle Style  MPG-H  MPG-C  Popularity  Price HP_Type Price_class
572           Wagon     30     22         819  24695     low         low
4897          Coupe     34     25         105  31090     low         mid
4898          Coupe     34     25         105  26000     low         low
5962  2dr Hatchback     37     36         105  15665     low         low
10312 2dr Hatchback     31     23         105  20360     low         low
10313 2dr Hatchback     31     23         105  24340     low         low
11548         Wagon     28     22         105  18070     low         low
11549         Wagon     28     22         105  19685     low         low
```

2.3.8. Counting specific features [2 marks]

For BMWs with `Price` greater than 30,000, report the number of cars for each `Transmission` category.

In [23]:

```
cars[(cars['Make'] == 'BMW') & (cars['Price'] > 30000)].groupby('Transmission').size()
```

Out [23]:

```
Transmission
AUTOMATED_MANUAL     18
AUTOMATIC           240
DIRECT_DRIVE          4
MANUAL               51
dtype: int64
```

2.3.9. Grouping data [2 marks]

For 2017 data, show the minimum and maximum `Price`, as well as minimum and maximum `HP` for each `Make`.

Display the values as **integers** in a **single** `DataFrame`.

In [24]:

```
cars[(cars['Year'] == 2017)].groupby(['Make'])[['Price', 'HP']].agg([np.min, np.max])
```

Out [24]:

```
              Price            HP
                min    max   min      max
Make
```

Acura	27990	156000	201.0	573.000000
Audi	31200	189900	186.0	610.000000
BMW	33100	137000	170.0	600.000000
Buick	21065	49625	138.0	310.000000
Cadillac	34595	97795	265.0	640.000000
Chevrolet	13000	92395	98.0	650.000000
Chrysler	21995	45270	184.0	300.000000
Dodge	20995	118795	173.0	707.000000
FIAT	15990	31800	101.0	252.588752
Ford	14130	68996	120.0	526.000000
GMC	24070	71665	182.0	420.000000
Genesis	41400	54550	311.0	420.000000
Honda	15990	47070	130.0	280.000000
Hyundai	17150	41150	128.0	293.000000
Infiniti	33950	53300	208.0	400.000000
Kia	14165	45700	138.0	290.000000
Land Rover	37695	62500	240.0	240.000000
Lexus	31250	89380	134.0	467.000000
Lincoln	32720	76645	240.0	380.000000
Lotus	91900	91900	400.0	400.000000
Maserati	71600	145500	345.0	523.000000
Mazda	17845	30695	146.0	184.000000
Mercedes-Benz	32400	247900	177.0	621.000000
Mitsubishi	12995	31695	78.0	224.000000
Nissan	11990	109990	109.0	565.000000
Porsche	47500	200400	252.0	580.000000
Subaru	18395	39995	148.0	305.000000
Toyota	15250	84325	106.0	381.000000
Volkswagen	17895	60195	150.0	292.000000
Volvo	33950	57200	240.0	316.000000

2.3.10. Reporting extreme values 1 mark

Determine which `Make` has the highest mean `Price`. Display this `Make` and its corresponding mean `Price` (rounded to 2 decimal places).

In [25]:

```
cars.groupby('Make')['Price'].mean().sort_values().tail(1).round(2)
```

Out [25]:

```
Make
Bugatti    1757223.67
Name: Price, dtype: float64
```

40 Marks

Bracket

Design a `Tournament` class to represent **teams** of car efficiency enthusiasts that can collect an **inventory** of fuel efficient cars by finding sponsorship from car makers.

Once **sponsors** are arranged and teams are formed, teams will **compete** by facing off head-to-head (pairwise, bracket-style) until only one `Tournament.champion` prevails.

Competitors perform by driving each car in their inventory the distance that **1 gallon** of fuel will permit them to travel in **highway** situations.

The **winning team** of a match will have collectively driven their vehicles the furthest. That is, the sum of the `MPG-H` ratings in their inventory will be the **score** that decides the winner.

3.1 Prepare data 5 marks

Begin with the data you have saved in the `cardata_modified.csv` file.

Read in and modify this data so that:

- The `Price`s are rounded to the **nearest $100**.
- Only cars made **after the** `Year` **2000** are retained in the `DataFrame`.
- Only car `Make`rs with **more than 55 entries** in the dataset are retained in the `DataFrame`.

Teams will be able to choose cars for their inventory from this modified `DataFrame`.

In [26]:

```
cars = pd.read_csv('cardata_modified.csv')
cars['Price'] = cars['Price'].round(-2)
cars = cars[cars['Year'] > 2000]

cars = cars[cars.groupby('Make').Make.transform('count') > 55]
cars.Make.value_counts()
```

Out [26]:

```
Chevrolet        959
Ford             693
Toyota           576
Volkswagen       545
Nissan           486
GMC              427
Dodge            414
Honda            410
Cadillac         381
Mazda            321
BMW              314
Infiniti         312
Suzuki           306
Audi             284
Mercedes-Benz    271
Hyundai          243
Kia              227
Subaru           210
Acura            210
Volvo            187
Lexus            186
Mitsubishi       175
Buick            166
Chrysler         159
Lincoln          152
Pontiac          141
Land Rover       123
Porsche          123
Aston Martin      91
Saab              78
Bentley           74
Ferrari           69
FIAT              62
Scion             60
Oldsmobile        57
Name: Make, dtype: int64
```

3.2. Define a `Tournament` class 30 marks

Your are to translate **these requirements** (with consideration of their usage, defined in **3.3** and **3.4**) into working code.

Overall: Coding efficiency and structure, including comments and docstrings, where appropriate, will contribute to the mark in th

- The class is initialised with:
  - a `DataFrame` of **cars**, as designed above
    - **No missing values** are permitted.
  - a tournament **name**
  - an **optional number of competing teams**, defaulting to 16.
    - If the input number of teams is not an integer, `raise` the appropriate kind of exception, and a message saying, "The number of teams must
    - Also, `assert` that the value is positive and non-zero.
    - Ensure that this number is a **perfect square**.
  - Include sensible object representation dunder methods (i.e. `__repr__` and `__str__`).

- There is a method to `generate_sponsors`. - The method, by default, **randomly selects a sponsor** for each team from the available list of `Mak `generate_teams`. - This method should simply populate a list of `Team`s in the `Tournament` class.$$$$ - The teams are members of the `Team` cl

- There is a method to `buy_cars`.

- This method will allow the `Team`s to each purchase their initial inventory.

$-Thereisamethodto`_purchase_inventory`. - Thisinternalmethodtakes1argument : asingle`Team`object. - Withtheinformationpro$

- There is a method to `hold_event` (i.e., execute the tournament competition process).
  - Cycle through the pairwise matches, keeping track of `Team` performance metrics.
    - The teams will compete in a head to head match and either continue on or be **eliminated**.
  - After each match, allocate a **financial prize** to the winning `Team`. You can decide how to implement this; perhaps the prize increases in every
  - After awarding the prize, allow the `Team` to `_purchase_inventory` again (increasing the number of cars in their inventory) before the next match.
    - Newly purchased cars can be duplicates of members of the `Team`'s existing inventory, but only one any kind of car is permitted to be purch
  - At the end of the tournament event, record the `Tournament.champion` `Team`.

In [49]:

```python
import pandas as pd
import numpy as np
from random import choice, randint, sample

class Tournament:
    """Tournement class"""
    class Team:
        def __init__(self, sponsor, budget):
            self.sponsor = sponsor
            self.budget = budget
            self.inventory = []
            self.active = True
            self.win_record = []

        def __str__(self):
            return f"Team sponsored by {self.sponsor} with budget £{self.budget} and {len(self.inventory)} cars."

    def __init__(self, cars, name, num_teams=16):
        if not isinstance(num_teams, int):
            raise ValueError("The number of teams must be an integer.")
        assert num_teams > 0, "Number of teams must be positive and non-zero."
        assert (num_teams & (num_teams - 1) == 0) and num_teams != 0, "Number of teams must be a perfect square."

        self.cars = cars
        self.name = name
        self.num_teams = num_teams
        self.sponsors = []
        self.budgets = []

    def __repr__(self):
        return f"Tournament(name={self.name}, num_teams={self.num_teams})"

    def __str__(self):
        return f"Tournament: {self.name} with {self.num_teams} teams competing."

    def generate_sponsors(self, specified_sponsors=None, low=100000, high=500000, incr=100):
        """
        Randomly selects sponsors for each team and random budgets between 100k and 500k.
        """
        makers = self.cars['Make'].unique()
        num_specified = len(specified_sponsors) if specified_sponsors else 0
        if num_specified < self.num_teams:
            random_sponsors = sample(list(makers), self.num_teams - num_specified)
            self.sponsors = specified_sponsors + random_sponsors if specified_sponsors else random_sponsors
        else:
            self.sponsors = specified_sponsors[:self.num_teams]

        self.budgets = [randint(low // incr, high // incr) * incr for _ in range(self.num_teams)]

    def generate_teams(self):
        """
        Generates the team objects within the tournament
        """
        self.teams = [self.Team(sponsor, budget) for sponsor, budget in zip(self.sponsors, self.budgets)]

    def buy_cars(self):
        """
        Method for teams to purchase their initial inventory based on their budget
        """
```

```python
        for team in self.teams:
            self._purchase_inventory(team)

    def _purchase_inventory(self, team):
        """
        Internal method to buy cars from available cars
        """
        available_cars = self.cars[self.cars['Make'] == team.sponsor].copy()
        available_cars.sort_values(by='MPG-H', ascending=False, inplace=True)

        for _, car in available_cars.iterrows():
            if car['Price'] <= team.budget:
                team.inventory.append(car)
                team.budget -= car['Price']

    def show_win_record(self):
        if self.teams is not None:
            for team in self.teams:
                print(f"{team.sponsor}: {team.win_record}")
        else:
            print("Teams not yet generated or competition not held.")

    def show_teams(self):
        """
        Displays information about teams in the tournament including inventory size, budget and sponsor
        """
        if not self.teams:
            print("No teams have been generated yet.")
            return

        print(f"{'Team Sponsor':<20} {'Budget':<10} {'Inventory Size':<15}")
        for team in self.teams:
            sponsor = team.sponsor
            budget = f"${team.budget}"
            inventory_size = len(team.inventory)
            print(f"{sponsor:<20} {budget:<10} {inventory_size:<15}")

    def hold_event(self):
        """
        Executes the tournament and determines a champion
        """
        for team in self.teams:
            team.win_record = []

        round_teams = self.teams
        while len(round_teams) > 1:
            winners = []
            for i in range(0, len(round_teams), 2):
                team1, team2 = round_teams[i], round_teams[i+1]

                score1 = sum(car['MPG-H'] for car in team1.inventory)
                score2 = sum(car['MPG-H'] for car in team2.inventory)

                if score1 > score2:
                    winner, loser = team1, team2
                else:
                    winner, loser = team2, team1

                winner.win_record.append('W')
                loser.win_record.append('L')

                winners.append(winner)
            round_teams = winners

        self.champion = round_teams[0]

        print(f"The champion is {self.champion.sponsor} with a total MPG-H of {sum(car['MPG-H'] for car in self.champion.inventory)}")
```

3.3. Execute your `Tournament` class [5 marks]

- The process of building and executing the stages associated with the tournament will look like this:

```
t1 = Tournament(car, "The First Folks")
t1.generate_sponsors()
t1.generate_teams()
t1.buy_cars()
t1.hold_event()
print(f'The champion of {t1.name} Tournament is the {t1.champion}')
```

```
...
The champion of The First Folks Tournament is the Team sponsored by Nissan with $32000 available and 32 cars.
```

- Produce some visual (e.g., printed or plotted) record of the `Tournament` matches by invoking the `show_win_record` method:

```
t1.show_win_record()
```

```
...

        Scion: ['W    ', 'W    ', 'L    ']
       Suzuki: ['L    ']
       Subaru: ['W    ', 'L    ']
   Land Rover: ['L    ']
       Nissan: ['W    ', 'W    ', 'W    ', 'W    ']
        Dodge: ['L    ']
        Lexus: ['L    ']
         Saab: ['W    ', 'L    ']
Mercedes-Benz: ['L    ']
        Buick: ['W    ', 'L    ']
   Volkswagen: ['W    ', 'W    ', 'L    ']
   Oldsmobile: ['L    ']
      Hyundai: ['W    ', 'W    ', 'W    ', 'L    ']
     Infiniti: ['L    ']
     Cadillac: ['L    ']
          BMW: ['W    ', 'L    ']
```

In [55]:

```
t1 = Tournament(cars, "SuperLeaded League")
t1.generate_sponsors()
t1.generate_teams()
t1.buy_cars()
t1.hold_event()
print(f'The champion of {t1.name} Tournament is the {t1.champion}')
t1.show_win_record()
```

```
The champion is Nissan with a total MPG-H of 1410
The champion of SuperLeaded League Tournament is the Team sponsored by Nissan with budget £1200 and 20 cars.
Lincoln: ['L']
Lexus: ['W', 'L']
Land Rover: ['L']
Infiniti: ['W', 'W', 'L']
GMC: ['L']
Kia: ['W', 'W', 'W', 'L']
BMW: ['L']
FIAT: ['W', 'L']
Volkswagen: ['L']
Mitsubishi: ['W', 'L']
Nissan: ['W', 'W', 'W', 'W']
Dodge: ['L']
Cadillac: ['L']
Chrysler: ['W', 'W', 'L']
Audi: ['W', 'L']
Saab: ['L']
```

In [56]:

```
t1.show_teams()
```

```
Team Sponsor        Budget     Inventory Size
Lincoln             $11800     3
Lexus               $21900     5
Land Rover          $2100      3
Infiniti            $13600     9
```

```
GMC            $1900     9
Kia            $4700     10
BMW            $3300     5
FIAT           $800      11
Volkswagen     $100      12
Mitsubishi     $10100    25
Nissan         $1200     20
Dodge          $12000    25
Cadillac       $24200    9
Chrysler       $12200    18
Audi           $7000     4
Saab           $20100    10
```

2389

3.4. Compare the results of multiple `Tournament` executions [5 marks]

- Lastly, **execute 2** `Tournaments` in full.
- Compare the performance of their `.champion`s and produce a ranked representation of the different `Tournaments`. For example, when we rank tournaments by the score of their overall champion, we might produce:

```
Tournament ranking:
Position    Name              Sponsor         Score
1           The Other Group   Chevrolet       2044
2           The First Folks   Nissan          1737
```

In [57]:

```python
tournament1 = Tournament(cars, name="The First Folks")
tournament1.generate_sponsors()
tournament1.generate_teams()
tournament1.buy_cars()
tournament1.hold_event()

tournament2 = Tournament(cars, name="SuperLeaded League")
tournament2.generate_sponsors()
tournament2.generate_teams()
tournament2.buy_cars()
tournament2.hold_event()


champion1_score = sum(car['MPG-H'] for car in tournament1.champion.inventory)
champion2_score = sum(car['MPG-H'] for car in tournament2.champion.inventory)

tournaments = [
    {"Position": 1, "Name": tournament1.name, "Sponsor": tournament1.champion.sponsor, "Score": champion1_score},
    {"Position": 2, "Name": tournament2.name, "Sponsor": tournament2.champion.sponsor, "Score": champion2_score}
]

tournaments_sorted = sorted(tournaments, key=lambda x: x["Score"], reverse=True)

for i, tournament in enumerate(tournaments_sorted, start=1):
    tournament["Position"] = i

print("Tournament ranking:")
print(f"{'Position':<12} {'Name':<25} {'Sponsor':<25} {'Score':<5}")
for tournament in tournaments_sorted:
    print(f"{tournament['Position']:<12} {tournament['Name']:<25} {tournament['Sponsor']:<25} {tournament['Score']:<5}")
```

```
The champion is Nissan with a total MPG-H of 1286
The champion is Toyota with a total MPG-H of 763
Tournament ranking:
Position     Name                      Sponsor                   Score
1            The First Folks           Nissan                    1286
2            SuperLeaded League        Toyota                    763
```

In [ ]:

cardata_modified.csv Download

1	Large file hidden. You can download it using the button above.