WTF23 DATA SCIENCE AND ARTIFICAIL INTELLIGENCE

GROUP C SUBGROUP 1

WEEKLY QUIZZ ON PYTHON- NUMPY AND PANDAS

This notebook is a compilation of some functions in pandas and numpy, what they do with examples

NUMPY FUNCTIONS

Numpy is a python library package for scientific computing. It provides a high-performance multidimensional array object, and tools for working with these arrays. This library is widely used for numerical analysis, matrix computations, and mathematical operations. In this article, we present 20 useful numpy functions along with data science and artificial intelligence applications. Let's get started! 🍀

Numpy Functions with Their Uses and Syntax

Function	Uses	Syntax
linspace()	create an array with values that are spaced linearly in a specified interval	np.linspace(start_number, stop_number, number of samples to generate)
sort()	to sort elements in an array	np.sort(array)
vstack()	stack arrays vertically	np.vstack(array1, array2,...arrayn)
hstack()	stack arrays horizontally	np.hstack(array1, array2,...arrayn
unique()	print unique values in the array	np.unique(array)
flip()	flip, or reverse, the contents of an array along an axis	np.flip(array)
ones()	create an array filled with ones	np.ones(rows, columns)
Reshape()	Gives a new shape to an array without changing its data	arr.reshape()
Flatten()	Return a copy of the array collapsed into one dimension	ndarray.flatten(order='C')
digitize()	Return the indices of the bins to which each value in input array belongs	numpy.digitize(x, bins, right=False)
Concatenate()	Join a sequence of arrays along an existing axis	np.concatenate((x,y))
Flipud()	Reverse the order of elements along axis 0(up/down)	np.flipud(a)
Repeat()	The function repeats the elements of an array. The number of repetitions is specified by the second argument repeats.	numpy.repeat(a, repeats, axis=None)
numpy.random.randint()	The function returns random integers from the interval	numpy.random.randint(low, high=None, size=None, dtype='l')

Function	Uses	Syntax
numpy.polyfit()	The function outputs a polynomial of degree deg that fits the points (x,y), minimizing the square error	numpy.polyfit(x, y, deg, rcond=None, full=False, w=None, cov=False)
numpy.polyval()	The function evaluates a polynomial at specific values	numpy.polyval(p, x)
numpy.argmax()	The function returns the indices of the maximum values along an axis.	numpy.argmax(a, axis=None, out=None)
Save()	The function is used to save the content of an array inside a text file	np.savetxt('array.txt',arr)
Load()	The function is used to load the content of an array from a text file. It takes the file name as a parameter	np.loadtxt('array.txt')
Histogram()	It is an important statistical analysis function that computes histogram values for a set of data	np.histogram(A)

Pandas Funtions with Their uses and Syntax

PANDA

Python's Pandas library is the most widely used library in Python. Because this is the data manipulation library that is necessary for every aspect of data analysis or machine learning. Even if you are working on data visualization or machine learning, some data manipulation will be there anyway. In this piece, I will list 20 Pandas functions that are necessary for everyday use to perform the regular data manipulation tasks.

Funtion	Uses	Syntax
read_csv()	To read a comma separated values(csv) file	pd.read_csv("filepath")
to_excel()	To store data to an Excel file	pd.to_excel("filepath")
nunique()	To get the total unique values of variables	dataframe.nunique()
columns()	To get the names of all the variables in a dataframe	dataframe.columns
describe()	To understand basic statistics of variables in a dataframe	dataframe.describe()
read()	This function is used to import data in various formats into the notebook area	pd.read_csv("filename")
head()	Used to display first set of rows in a dataset, it can take numbers as argument to specify number of rows to display	df.head( )
tail()	Used to display the last set of rows in a dataset. the default number of rows displayed without specifying is five rows	df.tail()
describe()	function is used to display the summary statistics of numerical columns of the dataset	df.describe()
info()	function is used to display information about the dataset columns	df.info()
shape()	Used to get the shape of the dataframe, i.e. the number of rows and columns	df.shape()
isnull().sum()	Used to check for missing values in a dataset	df.isnull().sum()
isna().sum()	Used to check for missing values in a dataset	df.isna().sum()
duplicated()	Used to check if there are duplicate values in a dataset	df.duplicated()
len()	Used to check the number of rows in a dataframe	len(df)
drop_duplicates()	Used to drop the duplicate values in a dataframe	df.drop_duplicates()

Funtion	Uses	Syntax
drop()	Used to remove columns that are not important for use in a dataframe, it takes a list of columns as argument	df.drop(arr)
rename()	Used to rename columns for better readability, it takes a dictionary format with columns to rename as arguments	df.rename(arr)
dropna()	Used to drop missing values in a dataframe	df.dropna()
nunique()	The function counts the number of unique items in a column	df.nunique()
value_counts()	The function counts the number of rows with each unique value in a column	df.value_counts()

In [1]:
```
import numpy as np
import pandas as pd
df = pd.read_csv('data.csv')
```

EXAMPLES WITH NUMPY FUNCTIONS

1: LINSPACE()

In [2]:
```
np.linspace(1, 5)
```

Out[2]:
```
array([1.        , 1.08163265, 1.16326531, 1.24489796, 1.32653061,
       1.40816327, 1.48979592, 1.57142857, 1.65306122, 1.73469388,
       1.81632653, 1.89795918, 1.97959184, 2.06122449, 2.14285714,
       2.2244898 , 2.30612245, 2.3877551 , 2.46938776, 2.55102041,
       2.63265306, 2.71428571, 2.79591837, 2.87755102, 2.95918367,
       3.04081633, 3.12244898, 3.20408163, 3.28571429, 3.36734694,
       3.44897959, 3.53061224, 3.6122449 , 3.69387755, 3.7755102 ,
       3.85714286, 3.93877551, 4.02040816, 4.10204082, 4.18367347,
       4.26530612, 4.34693878, 4.42857143, 4.51020408, 4.59183673,
       4.67346939, 4.75510204, 4.83673469, 4.91836735, 5.        ])
```

2: SORT()

In [3]:
```
prime_numbers = [11, 3, 7, 5, 2]

# sorting the list in ascending order
prime_numbers.sort()

print(prime_numbers)
```

```
[2, 3, 5, 7, 11]
```

3: VSTACK

In [4]:
```
#Get the 3-D stacked array
arr = np.array([[[1, 3], [2, 4]], [[3, 5], [5, 7]]])
arr1 = np.array([[[4, 1], [5, 7]], [[6, 8],[3, 5]]])
arr2 = np.vstack((arr, arr1))

print(arr); print(arr1); print(arr2)
```

```
[[[1 3]
  [2 4]]

 [[3 5]
  [5 7]]]
[[[4 1]
  [5 7]]

 [[6 8]
  [3 5]]]
[[[1 3]
  [2 4]]

 [[3 5]
  [5 7]]

 [[4 1]
  [5 7]]

 [[6 8]
  [3 5]]]
```

4: HSTACK

In [5]:
```
# Example 1: Use NumPy.hstack() Functions
arr = np.array([ 2, 3, 4])
arr1 = np.array([5, 6, 7])
arr2 = np.hstack((arr,arr1))

print(arr); print(arr1); print(arr2)
```

```
[2 3 4]
[5 6 7]
[2 3 4 5 6 7]
```

In [6]:
```
# Example 2: Use numpy.hstack() function to 2-d numpy arrays
arr =  np.array([[ 2, 3, 5], [ -1, -3, -5]])
arr1 = np.array([[ 6, 8, 10], [ -7, -8, -9]])
arr2 = np.hstack((arr,arr1))

print(arr); print(arr1); print(arr2)
```

```
[[ 2  3  5]
 [-1 -3 -5]]
[[ 6  8 10]
 [-7 -8 -9]]
[[ 2  3  5  6  8 10]
 [-1 -3 -5 -7 -8 -9]]
```

In [7]:
```
# Example 3: stacking arrays horizontally
arr =  np.array([[ 2, 3], [ 4, 6]])
arr1 = np.array([[ 6, 8, 10], [ 8, 10, 12]])
arr2 = np.hstack((arr,arr1))

print(arr); print(arr1); print(arr2)
```

```
[[2 3]
 [4 6]]
[[ 6  8 10]
 [ 8 10 12]]
[[ 2  3  6  8 10]
 [ 4  6  8 10 12]]
```

5: UNIQUE

In [8]:
```
list_inp = [100, 75, 100, 20, 75, 12, 75, 25]

res = np.array(list_inp)
unique_res = np.unique(res)
print("Unique elements of the list using numpy.unique():\n")
print(unique_res)
```

```
Unique elements of the list using numpy.unique():

[ 12  20  25  75 100]
```

6:FLIP

In [9]:
```
a = np.array([
    [3,5,7,9],
    [11,13,15,17],
    [9,21,23,25],
    [7,29,31,33]])

a = np.flip(a,1)
print(a)
```

```
[[ 9  7  5  3]
 [17 15 13 11]
 [25 23 21  9]
 [33 31 29  7]]
```

In [10]:
```
#Flipud: Reverse the order of elements along axis 0(up/down)

a = np.array([
    [3,5,7,9],
    [11,13,15,17],
    [9,21,23,25],
    [7,29,31,33]])

a = np.flipud(a)
print(a)
```

```
[[ 7 29 31 33]
 [ 9 21 23 25]
 [11 13 15 17]
 [ 3  5  7  9]]
```

7:ONES

In [11]:
```
#create arrays using only one and zeros
ones=np.ones((2,3))
ones
```

Out[11]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

8: RESHAPE

In [12]:
```
c = np.array([[5,10,15,20],[25,30,35,40],[45,50,55,60]])

cr = c.reshape([3,4])
print(c); print(cr)
```

```
[[ 5 10 15 20]
 [25 30 35 40]
 [45 50 55 60]]
[[ 5 10 15 20]
 [25 30 35 40]
 [45 50 55 60]]
```

9:FLATENS

In [13]:
```
## Flatens() example
a = np.array([[1,2], [3,4]])
a.flatten()
```

Out[13]:
```
array([1, 2, 3, 4])
```

In [14]:
```
a.flatten('F')
```

Out[14]:
```
array([1, 3, 2, 4])
```

10: DIGITIZE

In [15]:
```
## Digitize example
# Input array
x = np.array([0.5])

# Bins - 5 bins in total
bins = np.array([0,1,2,3])

# Digitize function - 0.5 belong to the bin 0<= 0.5 <1 - therefore returned index 1
np.digitize(x,bins)
# array([1], dtype=int64)
```

Out[15]:
```
array([1], dtype=int64)
```

In [16]:
```
# The input array can contain several inputs
x = np.array([-0.5,1,3.5])

# Digitize function
np.digitize(x,bins)
# array([0, 2, 4], dtype=int64)
```

Out[16]:
```
array([0, 2, 4], dtype=int64)
```

11: CONCATENATE

In [17]:
```
#Concatenate example

x = np.array([
     [1,2,3,14],
     [4,5,6,12],
     [7,8,9,10]
])

y = np.array([
    [5,7,15,31],
    [22,9,3,21],
    [11,13,25,19]
])

z = np.concatenate((x,y))
print(z)
```

```
[[ 1  2  3 14]
 [ 4  5  6 12]
 [ 7  8  9 10]
 [ 5  7 15 31]
 [22  9  3 21]
 [11 13 25 19]]
```

12:FLIPUD

In [18]:
```
#Flipud example

a = np.array([
    [3,5,7,9],
    [11,13,15,17],
    [9,21,23,25],
    [7,29,31,33]
])

a = np.flipud(a)
print(a)
```

```
[[ 7 29 31 33]
 [ 9 21 23 25]
 [11 13 15 17]
 [ 3  5  7  9]]
```

13: REPEAT

In [19]:
```
#Repeat example
# repeat number 3 5 times
np.repeat(3,5)
# array([3, 3, 3, 3, 3])
```

Out[19]:
```
array([3, 3, 3, 3, 3])
```

In [20]:
```
# repeat string '2015' 5 times
np.repeat('2015',5)
# array(['2015', '2015', '2015', '2015', '2015'], dtype='<U4')
```

Out[20]:
```
array(['2015', '2015', '2015', '2015', '2015'], dtype='<U4')
```

14:RANDOM.RANDINT

In [21]:
```
##numpy.random.randint example
# toss a coin
np.random.randint(2)
#1
```

Out[21]:
```
0
```

In [22]:
```
# toss a coin 5 times
np.random.randint(2,size=5)
#array([1, 1, 0, 0, 0])
```

Out[22]:
```
array([0, 0, 0, 1, 0])
```

In [23]:
```
# roll a dice
np.random.randint(1,7)
```

Out[23]:
```
4
```

In [24]:
```
#4
# roll a dice 10 times
np.random.randint(1,7,size=10)
```

Out[24]:
```
array([4, 6, 6, 6, 5, 3, 2, 4, 4, 3])
```

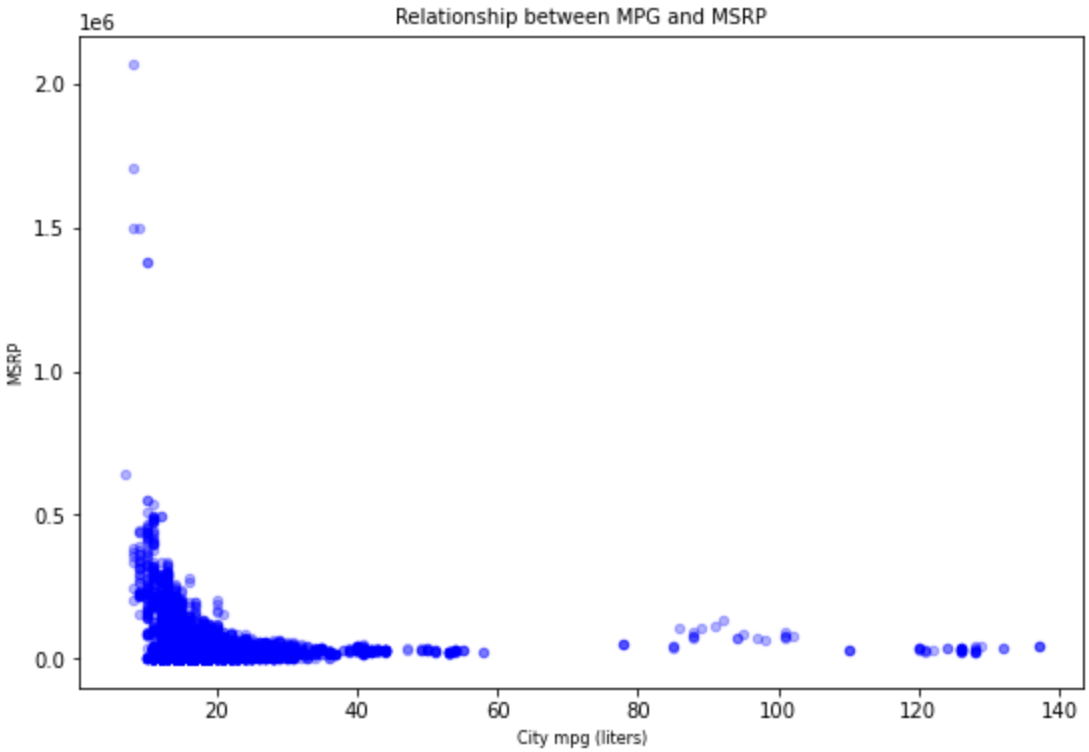
15: POLYFIT

In [25]:
```
### numpy.polyfit() examples

# read csv file
# first 5 rows
#df.head()
import matplotlib.pyplot as plt

# relation between height and weight
df.plot(kind='scatter', x='citympg',y='MSRP', color='blue',alpha=0.3, figsize=(9,6))

# title, xlabel, and ylabel
plt.title('Relationship between MPG and MSRP', size=10)
plt.xlabel('City mpg (liters)', size=8)
plt.ylabel('MSRP', size=8);
```



In [26]:
```
list(df)
```

Out[26]:
```
['Make',
 'Model',
 'Year',
 'Engine Fuel Type',
 'Engine HP',
 'Engine Cylinders',
 'Transmission Type',
 'Driven_Wheels',
 'Number of Doors',
 'Market Category',
 'Vehicle Size',
 'Vehicle Style',
 'highwayMPG',
 'citympg',
 'Popularity',
 'MSRP']
```

16: POLYVAL

In [27]:
```
### POLYVAL EXAMAPL
```

```
# Polynomial coefficients.
fit = np.polyfit(df.citympg, df.MSRP, 1)

print(fit)
#[-1054.51259208 61403.70290613]
```

```
[-1054.51259208 61403.70290613]
```

In [28]:
```
# Predict the weight - using the model weight=5.96*height-224.50
np.polyval(fit,70)
#-12412.178539604865
```

Out[28]:
```
-12412.178539604865
```

17: ARGMAX

In [29]:
```
##nUMPY ARGMAX() EXAMPLE
# numpy array
array = np.array([[1,2,3],[4,5,6]])

# index of maximum value
max_pos = np.argmax(array)

max_pos
#5
```

Out[29]:
```
5
```

18:HISTOGRAM()

In [30]:
```
### HISTOGRAM EXAMPLE
A = np.array([[3, 4, 5, 2],
              [6, 7, 2, 6]])
### HISTOGRAM EXAMPLE
np.histogram(A)
```

Out[30]:
```
(array([2, 0, 1, 0, 1, 0, 1, 0, 2, 1], dtype=int64),
 array([2. , 2.5, 3. , 3.5, 4. , 4.5, 5. , 5.5, 6. , 6.5, 7. ]))
```

EXAMPLES WITH PANDA FUNCTIONS

1 .read() function : This function is used to import data in various formats into the notebook area

In [31]:
```
#example using pd.read() to import a dataset
df = pd.read_csv("data.csv")
```

2 .head() function : Used to display first set of rows in a dataset, it can take numbers as argument to specify number of rows to display

In [32]:
```
#using .head() to display first 10 rows in df
df.head(5)
```

Out[32]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury

3.tail() function : used to display the last set of rows in a dataset. the default number of rows displayed without specifying is five rows

In [33]:
```
#example using .tail() function to display the last 5 rows in df
df.tail()
```

Out[33]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	
11909	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover
11910	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover
11911	Acura	ZDX	2012	premium unleaded (required)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover
11912	Acura	ZDX	2013	premium unleaded (recommended)	300.0	6.0	AUTOMATIC	all wheel drive	4.0	Crossover
11913	Lincoln	Zephyr	2006	regular unleaded	221.0	6.0	AUTOMATIC	front wheel drive	4.0	

4 .describe() function is used to display the summary statistics of numerical columns of the dataset

In [34]:
```
#example using .describe() function
df.describe()
```

Out[34]:

	Year	Engine HP	Engine Cylinders	Number of Doors	highwayMPG	citympg	Popularity	I
count	11914.000000	11845.00000	11884.000000	11908.000000	11914.000000	11914.000000	11914.000000	1.191400
mean	2010.384338	249.38607	5.628829	3.436093	26.637485	19.733255	1554.911197	4.059474
std	7.579740	109.19187	1.780559	0.881315	8.863001	8.987798	1441.855347	6.010910
min	1990.000000	55.00000	0.000000	2.000000	12.000000	7.000000	2.000000	2.000000
25%	2007.000000	170.00000	4.000000	2.000000	22.000000	16.000000	549.000000	2.100000
50%	2015.000000	227.00000	6.000000	4.000000	26.000000	18.000000	1385.000000	2.999500
75%	2016.000000	300.00000	6.000000	4.000000	30.000000	22.000000	2009.000000	4.223125
max	2017.000000	1001.00000	16.000000	4.000000	354.000000	137.000000	5657.000000	2.065902

5 .info() function is used to display information about the dataset columns

In [35]:
```
#example using .info()
df.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  highwayMPG         11914 non-null  int64  
 13  citympg            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
```

6 .shape attribute: used to get the shape of the dataframe, i.e. the number of rows and columns

In [36]:
```
#example using .shape
df.shape
```

Out[36]:
```
(11914, 16)
```

7.isnull().sum() or .isna().sum() : used to check for missing values in a dataset

In [37]:
```
#example using .isnull().sum()
df.isnull().sum()
```

Out[37]:
```
Make                    0
Model                   0
Year                    0
Engine Fuel Type        3
Engine HP              69
Engine Cylinders       30
Transmission Type       0
Driven_Wheels           0
Number of Doors         6
Market Category      3742
Vehicle Size            0
Vehicle Style           0
highwayMPG              0
citympg                 0
Popularity              0
MSRP                    0
dtype: int64
```

In [38]:
```
#example using .isna().sum()
df.isna().sum()
```

Out[38]:
```
Make                    0
Model                   0
Year                    0
Engine Fuel Type        3
Engine HP              69
Engine Cylinders       30
Transmission Type       0
Driven_Wheels           0
Number of Doors         6
Market Category      3742
Vehicle Size            0
Vehicle Style           0
highwayMPG              0
citympg                 0
Popularity              0
MSRP                    0
dtype: int64
```

8 .duplicated() : Used to check if there are duplicate values in a dataset

In [39]:
```
#example using .duplicated()
df[df.duplicated()] #displays all the duplicate rows in the data set
```

Out[39]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Mar
14	BMW	1 Series	2013	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury
18	Audi	100	1992	regular unleaded	172.0	6.0	MANUAL	front wheel drive	4.0	
20	Audi	100	1992	regular unleaded	172.0	6.0	MANUAL	front wheel drive	4.0	
24	Audi	100	1993	regular unleaded	172.0	6.0	MANUAL	front wheel drive	4.0	
25	Audi	100	1993	regular unleaded	172.0	6.0	MANUAL	front wheel drive	4.0	
...	...	...	...	...	...	...	...	...	...	
11481	Suzuki	X-90	1998	regular unleaded	95.0	4.0	MANUAL	four wheel drive	2.0	
11603	Volvo	XC60	2017	regular unleaded	302.0	4.0	AUTOMATIC	all wheel drive	4.0	Crossover,Luxury
11604	Volvo	XC60	2017	regular unleaded	240.0	4.0	AUTOMATIC	front wheel drive	4.0	Cro
11708	Suzuki	XL7	2008	regular unleaded	252.0	6.0	AUTOMATIC	all wheel drive	4.0	
11717	Suzuki	XL7	2008	regular unleaded	252.0	6.0	AUTOMATIC	front wheel drive	4.0	

715 rows × 16 columns

9 .len() function : used to check the number of rows in a dataframe

In [40]:
```
#example using len() to get the number of duplicate rows
len(df[df.duplicated()])
```

Out[40]:
```
715
```

10 .drop_duplicates() : used to drop the duplicate values in a dataframe

In [41]:
```
df = df.drop_duplicates()
#check the shape after dropping duplicates
df.shape
```

Out[41]:
```
(11199, 16)
```

11 .drop() : used to remove columns that are not important for use in a dataframe, it takes a list of columns as argument

In [42]:
```
df = df.drop(columns=['Number of Doors','Market Category','Vehicle Size','Vehicle Style','Popula
#check data after dropping column
df.head()
```

Out[42]:

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highwayMPG	citympg	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

12 .rename() : used to rename columns for better readability, it takes a dictionary format with columns to rename as arguments

In [43]:
```
df = df.rename(columns= {'Engine HP':'HP','Engine Cylinders':'Cylinders','Transmission Type':'Tr
df.head()
```

Out[43]:

	Make	Model	Year	HP	Cylinders	Transmission	Wheels	highwayMPG	citympg	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

13 .dropna() : used to drop missing values in a dataframe

In [44]:
```
df = df.dropna()
#check for missing values after dropping
df.isnull().sum() #no more missing values
```

Out[44]:
```
Make            0
Model           0
Year            0
HP              0
Cylinders       0
Transmission    0
Wheels          0
highwayMPG      0
citympg         0
Price           0
dtype: int64
```

14 .nunique() : counts the number of unique items in a column

In [45]:
```
df['Transmission'].nunique()
```

Out[45]:
```
5
```

15 .value_counts() : counts the number of rows with each unique value in a column

In [46]:
```
df['Transmission'].value_counts()
```

Out[46]:
```
AUTOMATIC           7900
MANUAL              2621
AUTOMATED_MANUAL     553
DIRECT_DRIVE          15
UNKNOWN               12
Name: Transmission, dtype: int64
```

CONCLUSION

There are several numpy and pandas functions used for specific purposes. A comprehensive list can be found on the documentation page

Numpy

Pandas

CONTRIBUTORS

Margaret Oluwadare

Loveth Osuagwu

Oluchi Okoro (Oluchi Oluchi)

Monsurat Onabajo

Mariam Anishere

Olubusayo Solola

Olayemi Ibiloye

Maryann Amaefula

Olayemi Oloyede

Oluwadunsin Olajide