# Pandas Practice

This notebook is dedicated to practicing different tasks with pandas. The solutions are available in a solutions notebook, however, you should always try to figure them out yourself first.

It should be noted there may be more than one different way to answer a question or complete an exercise.

Exercises are based off (and directly taken from) the quick introduction to pandas notebook.

Different tasks will be detailed by comments or text.

For further reference and resources, it's advised to check out the pandas documentation.

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

In [2]:
```
# Create a series of three different colours
colors = pd.Series(["Red","Blue","Yellow"])
```

In [3]:
```
# View the series of different colours
colors
```

Out[3]:
```
0       Red
1      Blue
2    Yellow
dtype: object
```

In [4]:
```
# Create a series of three different car types and view it
car_types = pd.Series(["Toyota","Mercedes","Honda"])
car_types
```

Out[4]:
```
0      Toyota
1    Mercedes
2       Honda
dtype: object
```

In [5]:
```
# Combine the Series of cars and colours into a DataFrame
car_data = pd.DataFrame({"Car type: ": car_types, "Color ": colors})
car_data
```

Out[5]:

| | Car type: | Color |
|---|---|---|
| **0** | Toyota | Red |
| **1** | Mercedes | Blue |
| **2** | Honda | Yellow |

In [6]:
```
# Import "../data/car-sales.csv" and turn it into a DataFrame
car_sales = pd.read_csv("data/car-sales.csv")
car_sales
```

Out[6]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | $4,000.00 |
| **1** | Honda | Red | 87899 | 4 | $5,000.00 |
| **2** | Toyota | Blue | 32549 | 3 | $7,000.00 |
| **3** | BMW | Black | 11179 | 5 | $22,000.00 |
| **4** | Nissan | White | 213095 | 4 | $3,500.00 |
| **5** | Toyota | Green | 99213 | 4 | $4,500.00 |
| **6** | Honda | Blue | 45698 | 4 | $7,500.00 |
| **7** | Honda | Blue | 54738 | 4 | $7,000.00 |
| **8** | Toyota | White | 60000 | 4 | $6,250.00 |
| **9** | Nissan | White | 31600 | 4 | $9,700.00 |

**Note:** Since you've imported `../data/car-sales.csv` as a DataFrame, we'll now refer to this DataFrame as 'the car sales DataFrame'.

In [7]:
```
# Export the DataFrame you created to a .csv file
car_sales.to_csv("data/exported-car-sales.csv",index=False)
```

In [8]:
```
# Find the different datatypes of the car data DataFrame
car_sales.dtypes
```

Out[8]:
```
Make             object
Colour           object
Odometer (KM)     int64
Doors             int64
Price            object
dtype: object
```

In [9]:
```
# Describe your current car sales DataFrame using describe()
car_sales.describe()
```

Out[9]:

| | Odometer (KM) | Doors |
|---|---|---|
| **count** | 10.000000 | 10.000000 |
| **mean** | 78601.400000 | 4.000000 |
| **std** | 61983.471735 | 0.471405 |
| **min** | 11179.000000 | 3.000000 |
| **25%** | 35836.250000 | 4.000000 |
| **50%** | 57369.000000 | 4.000000 |
| **75%** | 96384.500000 | 4.000000 |
| **max** | 213095.000000 | 5.000000 |

In [10]:
```
# Get information about your DataFrame using info()
car_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   Make            10 non-null     object
 1   Colour          10 non-null     object
 2   Odometer (KM)   10 non-null     int64 
 3   Doors           10 non-null     int64 
 4   Price           10 non-null     object
dtypes: int64(2), object(3)
memory usage: 528.0+ bytes
```

What does it show you?

In [11]:
```
# Create a Series of different numbers and find the mean of them
number_mean = pd.Series([1,2,3,4,5])
number_mean.mean()
```

Out[11]:
```
3.0
```

In [12]:
```
# Create a Series of different numbers and find the sum of them
number_sum = pd.Series([1,2,3,4,5])
number_sum.sum()
```

Out[12]:
```
15
```

In [13]:
```
# List out all the column names of the car sales DataFrame
car_sales.columns
```

Out[13]:
```
Index(['Make', 'Colour', 'Odometer (KM)', 'Doors', 'Price'], dtype='object')
```

In [14]:
```
# Find the length of the car sales DataFrame
len(car_sales)
```

Out[14]:
```
10
```

In [15]:
```
# Show the first 5 rows of the car sales DataFrame
car_sales.head(5)
```

Out[15]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | $4,000.00 |
| **1** | Honda | Red | 87899 | 4 | $5,000.00 |
| **2** | Toyota | Blue | 32549 | 3 | $7,000.00 |
| **3** | BMW | Black | 11179 | 5 | $22,000.00 |
| **4** | Nissan | White | 213095 | 4 | $3,500.00 |

In [16]:
```
# Show the first 7 rows of the car sales DataFrame
car_sales.head(7)
```

Out[16]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | $4,000.00 |
| **1** | Honda | Red | 87899 | 4 | $5,000.00 |
| **2** | Toyota | Blue | 32549 | 3 | $7,000.00 |
| **3** | BMW | Black | 11179 | 5 | $22,000.00 |
| **4** | Nissan | White | 213095 | 4 | $3,500.00 |
| **5** | Toyota | Green | 99213 | 4 | $4,500.00 |
| **6** | Honda | Blue | 45698 | 4 | $7,500.00 |

In [17]:
```
# Show the bottom 5 rows of the car sales DataFrame
car_sales.tail(5)
```

Out[17]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **5** | Toyota | Green | 99213 | 4 | $4,500.00 |
| **6** | Honda | Blue | 45698 | 4 | $7,500.00 |
| **7** | Honda | Blue | 54738 | 4 | $7,000.00 |
| **8** | Toyota | White | 60000 | 4 | $6,250.00 |
| **9** | Nissan | White | 31600 | 4 | $9,700.00 |

In [18]:
```
# Use .loc to select the row at index 3 of the car sales DataFrame
car_sales.loc[3]
```

Out[18]:
```
Make                  BMW
Colour              Black
Odometer (KM)       11179
Doors                   5
Price            $22,000.00
Name: 3, dtype: object
```

In [19]:
```
# Use .iloc to select the row at position 3 of the car sales DataFrame
car_sales.iloc[3]
```

Out[19]:
```
Make                  BMW
Colour              Black
Odometer (KM)       11179
Doors                   5
Price            $22,000.00
Name: 3, dtype: object
```

Notice how they're the same? Why do you think this is?

Check the pandas documentation for .loc and .iloc. Think about a different situation each could be used for and try them out.

In [20]:
```
# Select the "Odometer (KM)" column from the car sales DataFrame
car_sales["Odometer (KM)"]
```

Out[20]:
```
0    150043
1     87899
2     32549
3     11179
4    213095
5     99213
6     45698
7     54738
8     60000
9     31600
Name: Odometer (KM), dtype: int64
```

In [21]:
```
# Find the mean of the "Odometer (KM)" column in the car sales DataFrame
car_sales["Odometer (KM)"].mean()
```

Out[21]:
```
78601.4
```

In [22]:
```
# Select the rows with over 100,000 kilometers on the Odometer
car_sales[car_sales["Odometer (KM)"] > 100000]
```

Out[22]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | $4,000.00 |
| **4** | Nissan | White | 213095 | 4 | $3,500.00 |

In [23]:
```
# Create a crosstab of the Make and Doors columns
pd.crosstab(car_sales["Make"],car_sales["Doors"])
```

Out[23]:

| Doors | 3 | 4 | 5 |
|---|---|---|---|
| **Make** | | | |
| **BMW** | 0 | 0 | 1 |
| **Honda** | 0 | 3 | 0 |
| **Nissan** | 0 | 2 | 0 |
| **Toyota** | 1 | 3 | 0 |

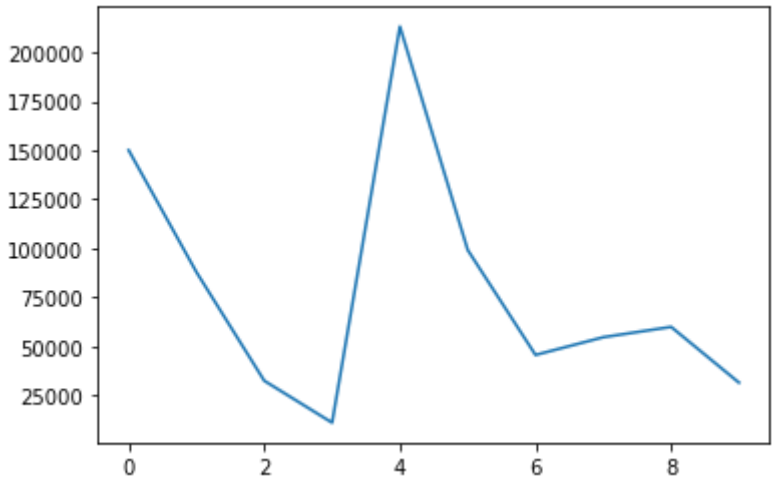
In [24]:
```
# Group columns of the car sales DataFrame by the Make column and find the average
car_sales.groupby(["Make"]).mean()
```

Out[24]:

| | Odometer (KM) | Doors |
|---|---|---|
| **Make** | | |
| **BMW** | 11179.000000 | 5.00 |
| **Honda** | 62778.333333 | 4.00 |
| **Nissan** | 122347.500000 | 4.00 |
| **Toyota** | 85451.250000 | 3.75 |

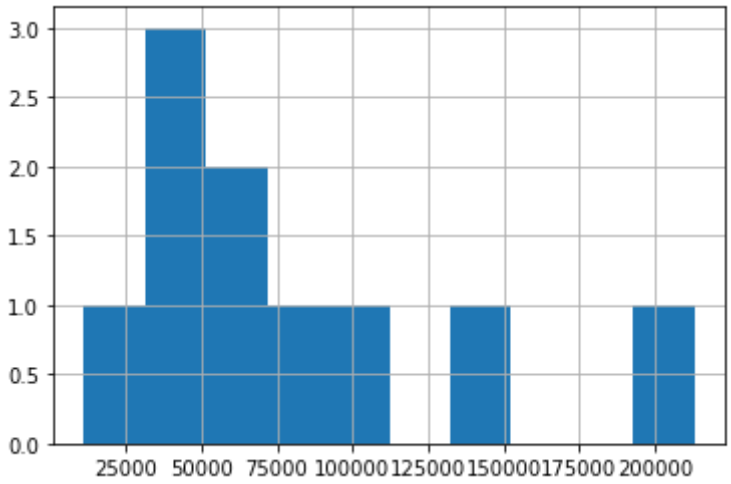
In [25]:
```
# Import Matplotlib and create a plot of the Odometer column
# Don't forget to use %matplotlib inline
%matplotlib inline
import matplotlib.pyplot as plt
car_sales["Odometer (KM)"].plot()
```

Out[25]:
```
<AxesSubplot:>
```




In [26]:
```
# Create a histogram of the Odometer column using hist()
car_sales["Odometer (KM)"].hist()
```

Out[26]:
```
<AxesSubplot:>
```




In [27]:
```
# Try to plot the Price column using plot()
car_sales["Price"].plot()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [27], in <cell line: 2>()
      1 # Try to plot the Price column using plot()
----> 2 car_sales["Price"].plot()

File c:\ml-courses-pratice\env\lib\site-packages\pandas\plotting\_core.py:972, in
PlotAccessor.__call__(self, *args, **kwargs)
    969             label_name = label_kw or data.columns
    970             data.columns = label_name
--> 972 return plot_backend.plot(data, kind=kind, **kwargs)

File c:\ml-courses-pratice\env\lib\site-packages\pandas\plotting\_matplotlib\__ini
t__.py:71, in plot(data, kind, **kwargs)
     69         kwargs["ax"] = getattr(ax, "left_ax", ax)
     70 plot_obj = PLOT_CLASSES[kind](data, **kwargs)
---> 71 plot_obj.generate()
     72 plot_obj.draw()
     73 return plot_obj.result

File c:\ml-courses-pratice\env\lib\site-packages\pandas\plotting\_matplotlib\core.
py:327, in MPLPlot.generate(self)
    325 def generate(self):
    326     self._args_adjust()
--> 327     self._compute_plot_data()
    328     self._setup_subplots()
    329     self._make_plot()

File c:\ml-courses-pratice\env\lib\site-packages\pandas\plotting\_matplotlib\core.
py:506, in MPLPlot._compute_plot_data(self)
    504 # no non-numeric frames or series allowed
    505 if is_empty:
--> 506     raise TypeError("no numeric data to plot")
    508 self.data = numeric_data.apply(self._convert_to_ndarray)

TypeError: no numeric data to plot
```

Why didn't it work? Can you think of a solution?

You might want to search for "how to convert a pandas string column to numbers".

And if you're still stuck, check out this Stack Overflow question and answer on turning a price column into integers.

See how you can provide the example code there to the problem here.

In [28]:
```
car_sales
```

Out[28]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | $4,000.00 |
| **1** | Honda | Red | 87899 | 4 | $5,000.00 |
| **2** | Toyota | Blue | 32549 | 3 | $7,000.00 |
| **3** | BMW | Black | 11179 | 5 | $22,000.00 |
| **4** | Nissan | White | 213095 | 4 | $3,500.00 |
| **5** | Toyota | Green | 99213 | 4 | $4,500.00 |
| **6** | Honda | Blue | 45698 | 4 | $7,500.00 |
| **7** | Honda | Blue | 54738 | 4 | $7,000.00 |
| **8** | Toyota | White | 60000 | 4 | $6,250.00 |
| **9** | Nissan | White | 31600 | 4 | $9,700.00 |

In [29]:
```
# Remove the punctuation from price column
car_sales["Price"] = car_sales["Price"].str.replace('[^\w\s]','')
```

```
C:\Users\Nguyen Minh Thuy\AppData\Local\Temp\ipykernel_9120\3923282592.py:2: FutureWarning: The default value of regex will change from True to False in a future version.
  car_sales["Price"] = car_sales["Price"].str.replace('[^\w\s]','')
```

In [30]:
```
# Check the changes to the price column
car_sales["Price"]
```

Out[30]:
```
0     400000
1     500000
2     700000
3    2200000
4     350000
5     450000
6     750000
7     700000
8     625000
9     970000
Name: Price, dtype: object
```

In [31]:
```
# Remove the two extra zeros at the end of the price column
car_sales["Price"] = car_sales["Price"].str[:-2]
```

In [32]:
```
# Check the changes to the Price column
car_sales["Price"]
```

Out[32]:
```
0     4000
1     5000
2     7000
3    22000
4     3500
5     4500
6     7500
7     7000
8     6250
9     9700
Name: Price, dtype: object
```

In [33]:
```
# Change the datatype of the Price column to integers
car_sales["Price"].astype(int)
```

Out[33]:
```
0     4000
1     5000
2     7000
3    22000
4     3500
5     4500
6     7500
7     7000
8     6250
9     9700
Name: Price, dtype: int32
```

In [34]:
```
# Lower the strings of the Make column
car_sales["Make"].str.lower()
```

Out[34]:
```
0    toyota
1     honda
2    toyota
3       bmw
4    nissan
5    toyota
6     honda
7     honda
8    toyota
9    nissan
Name: Make, dtype: object
```

If you check the car sales DataFrame, you'll notice the Make column hasn't been lowered.

How could you make these changes permanent?

Try it out.

In [35]:
```
car_sales
```

Out[35]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4 | 4000 |
| **1** | Honda | Red | 87899 | 4 | 5000 |
| **2** | Toyota | Blue | 32549 | 3 | 7000 |
| **3** | BMW | Black | 11179 | 5 | 22000 |
| **4** | Nissan | White | 213095 | 4 | 3500 |
| **5** | Toyota | Green | 99213 | 4 | 4500 |
| **6** | Honda | Blue | 45698 | 4 | 7500 |
| **7** | Honda | Blue | 54738 | 4 | 7000 |
| **8** | Toyota | White | 60000 | 4 | 6250 |
| **9** | Nissan | White | 31600 | 4 | 9700 |

In [36]:
```
# Make lowering the case of the Make column permanent
car_sales["Make"] = car_sales["Make"].str.lower()
```

In [37]:
```
# Check the car sales DataFrame
car_sales
```

Out[37]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| **0** | toyota | White | 150043 | 4 | 4000 |
| **1** | honda | Red | 87899 | 4 | 5000 |
| **2** | toyota | Blue | 32549 | 3 | 7000 |
| **3** | bmw | Black | 11179 | 5 | 22000 |
| **4** | nissan | White | 213095 | 4 | 3500 |
| **5** | toyota | Green | 99213 | 4 | 4500 |
| **6** | honda | Blue | 45698 | 4 | 7500 |
| **7** | honda | Blue | 54738 | 4 | 7000 |
| **8** | toyota | White | 60000 | 4 | 6250 |
| **9** | nissan | White | 31600 | 4 | 9700 |

Notice how the Make column stays lowered after reassigning.

Now let's deal with missing data.

In [38]:
```
# Import the car sales DataFrame with missing data ("../data/car-sales-missing-data
car_sales_missing = pd.read_csv("data/car-sales-missing-data.csv")

# Check out the new DataFrame
car_sales_missing
```

Out[38]:

| | Make | Colour | Odometer | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | $4,000 |
| **1** | Honda | Red | 87899.0 | 4.0 | $5,000 |
| **2** | Toyota | Blue | NaN | 3.0 | $7,000 |
| **3** | BMW | Black | 11179.0 | 5.0 | $22,000 |
| **4** | Nissan | White | 213095.0 | 4.0 | $3,500 |
| **5** | Toyota | Green | NaN | 4.0 | $4,500 |
| **6** | Honda | NaN | NaN | 4.0 | $7,500 |
| **7** | Honda | Blue | NaN | 4.0 | NaN |
| **8** | Toyota | White | 60000.0 | NaN | NaN |
| **9** | NaN | White | 31600.0 | 4.0 | $9,700 |

Notice the missing values are represented as `NaN` in pandas DataFrames.

Let's try fill them.

In [39]:
```
# Fill the Odometer column missing values with the mean of the column inplace
car_sales_missing["Odometer"].fillna(car_sales_missing["Odometer"].mean())
```

Out[39]:
```
0    150043.000000
1     87899.000000
2     92302.666667
3     11179.000000
4    213095.000000
5     92302.666667
6     92302.666667
7     92302.666667
8     60000.000000
9     31600.000000
Name: Odometer, dtype: float64
```

In [40]:
```
# View the car sales missing DataFrame and verify the changes
car_sales_missing
```

Out[40]:

| | Make | Colour | Odometer | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | $4,000 |
| **1** | Honda | Red | 87899.0 | 4.0 | $5,000 |
| **2** | Toyota | Blue | NaN | 3.0 | $7,000 |
| **3** | BMW | Black | 11179.0 | 5.0 | $22,000 |
| **4** | Nissan | White | 213095.0 | 4.0 | $3,500 |
| **5** | Toyota | Green | NaN | 4.0 | $4,500 |
| **6** | Honda | NaN | NaN | 4.0 | $7,500 |
| **7** | Honda | Blue | NaN | 4.0 | NaN |
| **8** | Toyota | White | 60000.0 | NaN | NaN |
| **9** | NaN | White | 31600.0 | 4.0 | $9,700 |

In [41]:
```
# Remove the rest of the missing data inplace
car_sales_missing.dropna(inplace=True)
```

In [42]:
```
# Verify the missing values are removed by viewing the DataFrame
car_sales_missing
```

Out[42]:

| | Make | Colour | Odometer | Doors | Price |
|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | $4,000 |
| **1** | Honda | Red | 87899.0 | 4.0 | $5,000 |
| **3** | BMW | Black | 11179.0 | 5.0 | $22,000 |
| **4** | Nissan | White | 213095.0 | 4.0 | $3,500 |

We'll now start to add columns to our DataFrame.

In [43]:
```
# Create a "Seats" column where every row has a value of 5
car_sales_missing["Seats"] = 5
car_sales_missing
```

Out[43]:

| | Make | Colour | Odometer | Doors | Price | Seats |
|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | $4,000 | 5 |
| **1** | Honda | Red | 87899.0 | 4.0 | $5,000 | 5 |
| **3** | BMW | Black | 11179.0 | 5.0 | $22,000 | 5 |
| **4** | Nissan | White | 213095.0 | 4.0 | $3,500 | 5 |

In [44]:
```
# Create a column called "Engine Size" with random values between 1.3 and 4.5
# Remember: If you're doing it from a Python list, the list has to be the same leng
# as the DataFrame
import numpy as np
car_sales_missing['Engine Size'] = np.random.uniform(1.3,4.5, size=len(car_sales_m
car_sales_missing
```

Out[44]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | $4,000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | $5,000 | 5 | 2.74 |
| **3** | BMW | Black | 11179.0 | 5.0 | $22,000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | $3,500 | 5 | 1.53 |

In [45]:
```
# convert Price column to int and remove $
car_sales_missing["Price"] = car_sales_missing["Price"].str.replace('[\$\,\.]','')
car_sales_missing
```

```
C:\Users\Nguyen Minh Thuy\AppData\Local\Temp\ipykernel_9120\3551647596.py:2: Futur
eWarning: The default value of regex will change from True to False in a future ve
rsion.
  car_sales_missing["Price"] = car_sales_missing["Price"].str.replace
('[\$\,\.]','').astype(int)
```

Out[45]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 |
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 |

In [46]:
```
# Create a column which represents the price of a car per kilometer
# Then view the DataFrame
car_sales_missing["Price per KM"] = round(car_sales_missing["Price"] / car_sales_m
car_sales_missing
```

Out[46]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size | Price per KM |
|---|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 | 0.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 | 0.06 |
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 | 1.97 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 | 0.02 |

In [47]:
```
# Remove the last column you added using .drop()
car_sales_missing = car_sales_missing.drop("Price per KM", axis = 1)
```

```
car_sales_missing
```

Out[47]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 |
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 |

In [48]:
```
car_sales_missing
```

Out[48]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 |
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 |

In [49]:
```
# Shuffle the DataFrame using sample() with the frac parameter set to 1
# Save the the shuffled DataFrame to a new variable
car_sales_shuffle = car_sales_missing.sample(frac = 1)
car_sales_shuffle
```

Out[49]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 |
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 |

Notice how the index numbers get moved around. The `sample()` function is a great way to get random samples from your DataFrame. It's also another great way to shuffle the rows by setting `frac=1`.

In [50]:
```
# Reset the indexes of the shuffled DataFrame
#car_sales_shuffle.reset_index(drop=True,inplace=True)
car_sales_shuffle.reset_index()
car_sales_shuffle
```

Out[50]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **3** | BMW | Black | 11179.0 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 213095.0 | 4.0 | 3500 | 5 | 1.53 |
| **0** | Toyota | White | 150043.0 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 87899.0 | 4.0 | 5000 | 5 | 2.74 |

Notice the index numbers have been changed to have order (start from 0).

In [54]:
```
# Change the Odometer values from kilometers to miles using a Lambda function
# Then view the DataFrame
```

```
car_sales_shuffle["Odometer"] = car_sales_shuffle["Odometer"].apply(lambda x: x/1.6
car_sales_shuffle
```

Out[54]:

| | Make | Colour | Odometer | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **3** | BMW | Black | 6986.875 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 133184.375 | 4.0 | 3500 | 5 | 1.53 |
| **0** | Toyota | White | 93776.875 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 54936.875 | 4.0 | 5000 | 5 | 2.74 |

In [55]:

```
# Change the title of the Odometer (KM) to represent miles instead of kilometers
car_sales_shuffle.rename(columns = {'Odometer':'Odometer (Miles)'}, inplace = True)
car_sales_shuffle
```

Out[55]:

| | Make | Colour | Odometer (Miles) | Doors | Price | Seats | Engine Size |
|---|---|---|---|---|---|---|---|
| **3** | BMW | Black | 6986.875 | 5.0 | 22000 | 5 | 1.55 |
| **4** | Nissan | White | 133184.375 | 4.0 | 3500 | 5 | 1.53 |
| **0** | Toyota | White | 93776.875 | 4.0 | 4000 | 5 | 2.03 |
| **1** | Honda | Red | 54936.875 | 4.0 | 5000 | 5 | 2.74 |

# Extensions

For more exercises, check out the pandas documentation, particularly the 10-minutes to pandas section.

One great exercise would be to retype out the entire section into a Jupyter Notebook of your own.

Get hands-on with the code and see what it does.

The next place you should check out are the top questions and answers on Stack Overflow for pandas. Often, these contain some of the most useful and common pandas functions. Be sure to play around with the different filters!

Finally, always remember, the best way to learn something new to is try it. Make mistakes. Ask questions, get things wrong, take note of the things you do most often. And don't worry if you keep making the same mistake, pandas has many ways to do the same thing and is a big library. So it'll likely take a while before you get the hang of it.