In [1]:
```python
# Import pandas
```

In [2]:
```python
# Create a series of three different colours
```

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

In [4]:
```python
# Create a series of three different car types and view it
```

In [5]:
```python
# Combine the Series of cars and colours into a DataFrame
```

In [6]:
```python
# Import "../data/car-sales.csv" and turn it into a DataFrame
```

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]:
```python
# Export the DataFrame you created to a .csv file
```

In [8]:
```python
# Find the different datatypes of the car data DataFrame
```

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

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

What does it show you?

In [11]:
```python
# Create a Series of different numbers and find the mean of them
```

In [12]:
```python
# Create a Series of different numbers and find the sum of them
```

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

In [14]:
```python
# Find the length of the car sales DataFrame
```

In [15]:
```python
# Show the first 5 rows of the car sales DataFrame
```

In [16]:
```python
# Show the first 7 rows of the car sales DataFrame
```

In [17]:
```python
# Show the bottom 5 rows of the car sales DataFrame
```

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

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

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]:
```python
# Select the "Odometer (KM)" column from the car sales DataFrame
```

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

In [22]:
```python
# Select the rows with over 100,000 kilometers on the Odometer
```

In [23]:
```python
# Create a crosstab of the Make and Doors columns
```

In [24]:
```python
# Group columns of the car sales DataFrame by the Make column and find the average
```

In [25]:
```python
# Import Matplotlib and create a plot of the Odometer column
# Don't forget to use %matplotlib inline
```

In [26]:
```python
# Create a histogram of the Odometer column using hist()
```

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

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

You might want to search for "how to convert a pandas string columb 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]:
```python
# Remove the punctuation from price column
```

In [29]:
```
# Check the changes to the price column
```

In [30]:
```
# Remove the two extra zeros at the end of the price column
```

In [31]:
```
# Check the changes to the Price column
```

In [32]:
```
# Change the datatype of the Price column to integers
```

In [33]:
```
# Lower the strings of the Make column
```

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 [34]:
```
# Make lowering the case of the Make column permanent
```

In [35]:
```
# Check the car sales DataFrame
```

Notice how the Make column stays lowered after reassigning.

Now let's deal with missing data.

In [36]:
```
# Import the car sales DataFrame with missing data ("../data/car-sales-missing-data.csv")


# Check out the new DataFrame
```

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

Let's try fill them.

In [37]:
```
# Fill the Odometer column missing values with the mean of the column inplace
```

In [38]:
```
# View the car sales missing DataFrame and verify the changes
```

In [39]:
```
# Remove the rest of the missing data inplace
```

In [40]:
```
# Verify the missing values are removed by viewing the DataFrame
```

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

In [41]:
```
# Create a "Seats" column where every row has a value of 5
```

In [42]:
```
# 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 length
# as the DataFrame
```

In [43]:
```
# Create a column which represents the price of a car per kilometer
# Then view the DataFrame
```

In [44]:
```
# Remove the last column you added using .drop()
```

In [45]:
```
# Shuffle the DataFrame using sample() with the frac parameter set to 1
# Save the the shuffled DataFrame to a new variable
```

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 [46]:
```
# Reset the indexes of the shuffled DataFrame
```

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

In [47]:
```
# Change the Odometer values from kilometers to miles using a Lambda function
# Then view the DataFrame
```

In [48]:
```
# Change the title of the Odometer (KM) to represent miles instead of kilometers
```