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 [ ]:

```
# Import pandas
import pandas as pd
import numpy as np
```

In [ ]:

```
# Create a series of three different colours
series1 = pd.Series(["White","Red","Black"])
```

In [ ]:

```
# View the series of different colours
series1
```

Out[ ]:

```
0    White
1      Red
2    Black
dtype: object
```

In [ ]:

```
# Create a series of three different car types and view it
series2 = pd.Series(["Toyota","Hyundai","Tata"])
```

In [ ]:

```
# Combine the Series of cars and colours into a DataFrame
cars_colours = pd.DataFrame({"Cars":series2,"Colours":series1})
cars_colours
```

Out[ ]:

	Cars	Colours
0	Toyota	White
1	Hyundai	Red
2	Tata	Black

In [ ]:

```
# Import "../data/car-sales.csv" and turn it into a DataFrame
df = pd.read_csv("car-sales.csv")
df.head()
```

Out[ ]:

	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

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 [ ]:

```
# Export the DataFrame you created to a .csv file
df.to_csv("car_sales_copy.csv")
```

In [ ]:

```
# Find the different datatypes of the car data DataFrame
df.dtypes
```

Out[ ]:

```
Make             object
Colour           object
Odometer (KM)     int64
Doors             int64
Price            object
dtype: object
```

In [ ]:

```
# Describe your current car sales DataFrame using describe()
df.describe()
```

Out[ ]:

	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 [ ]:

```
# Get information about your DataFrame using info()
df.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 [ ]:

```
# Create a Series of different numbers and find the mean of them
mean_data = pd.Series([1,3,5,7,9,2,4,6,8,10])
mean_data.mean()
```

Out[ ]:

```
5.5
```

In [ ]:

```
# Create a Series of different numbers and find the sum of them
sum_data = pd.Series([5,3,15,21,36,9,19,24,76])
sum_data.sum()
```

Out[ ]:

```
208
```

In [ ]:

```
# List out all the column names of the car sales DataFrame
df.columns
```

Out[ ]:

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

In [ ]:

```
# Find the length of the car sales DataFrame
len(df)
```

Out[ ]:

```
10
```

In [ ]:

```
# Show the first 5 rows of the car sales DataFrame
df.head()
```

Out[ ]:

	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 [ ]:

```
# Show the first 7 rows of the car sales DataFrame
df.head(7)
```

Out[ ]:

	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 [ ]:

```
# Show the bottom 5 rows of the car sales DataFrame
df.tail()
```

Out[ ]:

	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 [ ]:

```
# Use .loc to select the row at index 3 of the car sales DataFrame
df.loc[3:3,:]
```

Out[ ]:

	Make	Colour	Odometer (KM)	Doors	Price
3	BMW	Black	11179	5	$22,000.00

In [ ]:

```
# Use .iloc to select the row at position 3 of the car sales DataFrame
df.iloc[3:4,:]
```

Out[ ]:

	Make	Colour	Odometer (KM)	Doors	Price
3	BMW	Black	11179	5	$22,000.00

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.

They are the same because the index of this dataframe starst from 0, just as the default indexing would.

In [ ]:

```
# Select the "Odometer (KM)" column from the car sales DataFrame
odo = df["Odometer (KM)"]
```

In [ ]:

```
# Find the mean of the "Odometer (KM)" column in the car sales DataFrame
odo.mean()
```

Out[ ]:

```
78601.4
```

In [ ]:

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

Out[ ]:

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

In [ ]:

```
# Create a crosstab of the Make and Doors columns
cross_tab = pd.crosstab(df["Make"],df["Doors"])
cross_tab
```

Out[ ]:

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

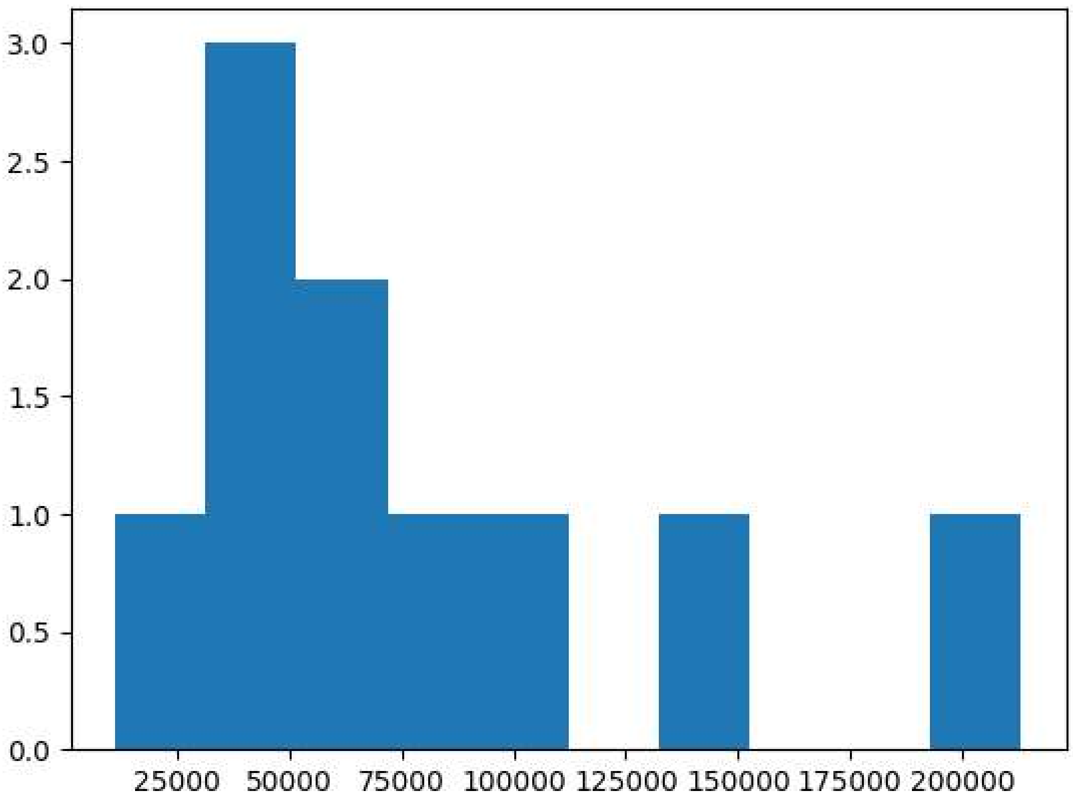
```
In [ ]: # Group columns of the car sales DataFrame by the Make column and find the average
        car_grp = df.groupby(["Make"])["Odometer (KM)"]
        car_grp.mean()
```

```
Out[ ]: Make
        BMW        11179.000000
        Honda      62778.333333
        Nissan    122347.500000
        Toyota     85451.250000
        Name: Odometer (KM), dtype: float64
```

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

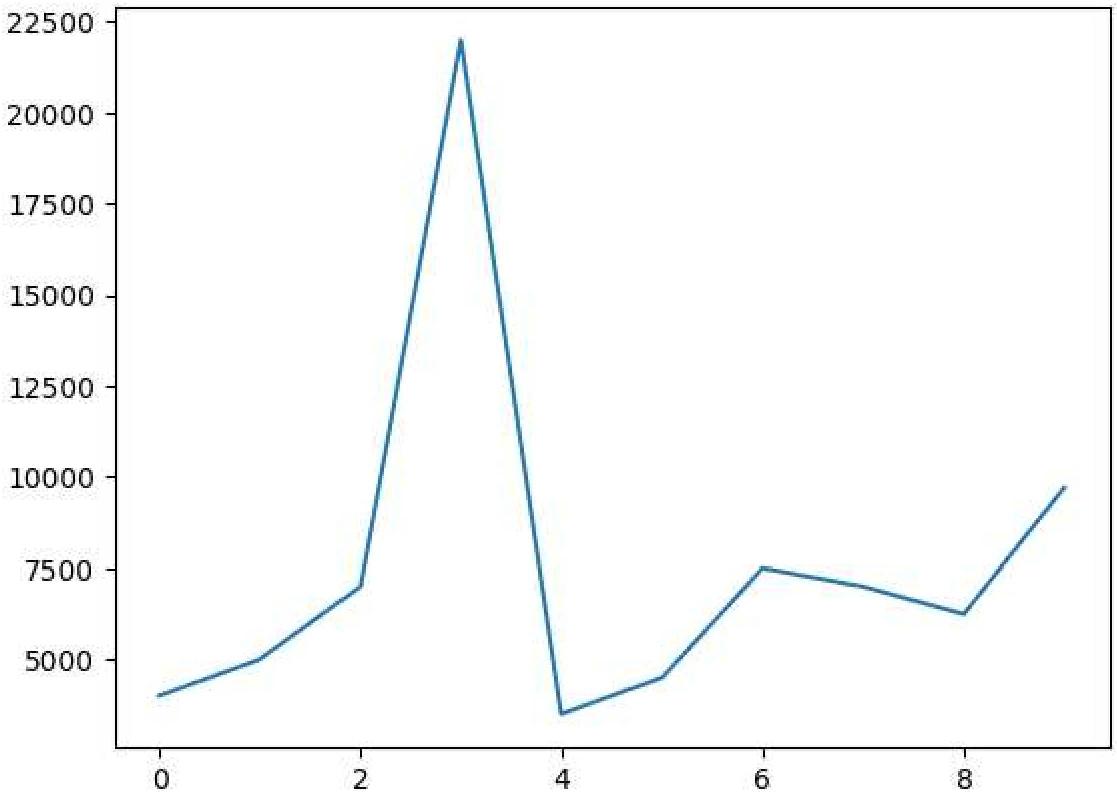
```
In [ ]: # Create a histogram of the Odometer column using hist()
        plt.hist(x = df["Odometer (KM)"])
```

```
Out[ ]: (array([1., 3., 2., 1., 1., 0., 1., 0., 0., 1.]),
         array([ 11179. ,  31370.6,  51562.2,  71753.8,  91945.4, 112137. ,
                132328.6, 152520.2, 172711.8, 192903.4, 213095. ]),
         <BarContainer object of 10 artists>)
```



```
In [ ]: # Try to plot the Price column using plot()
        df['Price'].plot()
```

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



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 [ ]:

```
# Remove the punctuation from price column
# use str.replace("[\$\,\.]", "")
# refer the link to study Punctuation replcemenet https://blog.enterprisedna.co/python-remove-punctuation-from-string/

df["Price"] = df['Price'].str.replace("$","")
df["Price"] = df['Price'].str.replace(",","")
df["Price"] = df['Price'].str.replace(".","")
```

In [ ]:

```
# Check the changes to the price column
df['Price']
```

Out[ ]:

```
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 [ ]:

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

In [ ]:

```
# Check the changes to the Price column
df["Price"]
```

Out[ ]:

```
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 [ ]:

```
# Change the datatype of the Price column to integers
df['Price'] = df['Price'].astype(int)
```

In [ ]:

```
# Lower the strings of the Make column
df["Make"].str.lower()
df.head()
```

Out[ ]:

	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

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 [ ]:

```
# Make lowering the case of the Make column permanent
df["Make"] = df["Make"].str.lower()
```

In [ ]:

```
# Check the car sales DataFrame
df.head()
```

Out[ ]:

	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

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.