Assignment 1

Download auto_mpg.txt and store inside 'tabular' folder created inside of 'data' directory. Add this notebook to your python codebase.

The **purpose** of this assignmnet is to become familiar with core Python (*Part 0*), numpy and Pandas basics (*Part 1*), and handling data (*Part 2*).

Tanay Nistala 2022-02-01

Part 0

The goal of Part 0 is to

- Practice problems based on core Python
- Gain better understanding of work-flows controlled by conditional statements

Resources for python

Here are some of the best resources for Python on the web.

Learning resources

- Interactive Python (http://interactivepython.org/) An online book that includes embedded live excercises. Fun!
- Dive Into Python (http://www.diveintopython.net/) An excellent, thorough book.
- tutorial point (http://www.tutorialspoint.com/python/index.htm) A resource that is useful when you want an explanation of one concept, rather than a whole chapter.

Reference resources

Typically, if you have a question about python, you can find an answer by using google. The following sites will usually have the best answer.

- Official python documentation (https://docs.python.org/3/library/)
- Quick Reference from Tutorial Point (http://www.tutorialspoint.com/python/python_quick_guide.htm)

Sample. Notice the printout below the solution. Notice it is self-documenting, including problem definition and solution (i.e., source code), along with result (i.e., printout).

In [1]:

```python
val = 2
li = [2, 3, 4, 5]
if val in li:
    print('Found value', val, 'in list')
else:
    print('Value', val, 'not found in list')
if 6 in li:
    print('Found value', 6, 'in list')
else:
    print('Value', 6, 'not found in list')
print('List items:', li)
```

```
Found value 2 in list
Value 6 not found in list
List items: [2, 3, 4, 5]
```

0.1) Describe the 4 core Python containers (note the keyword core, i.e., not numpy arrays or other container types that are included in Python Packages).

a) What are characteristics of each?

lists (li=[]), tuples (tu=()), strings (st=""), dictionaries (dic={})-- each are containers with various characteristics.

Lists and tuples can store any type, and can be made up of various different types. Both preserve order (as do strings), with the difference being lists are mutable, while tuples and strings are immutable.

Strings are made up of sequences of characters.

Dictionaries are key-value pairs, where values are accessed via indexing with key. Keys must be unique and are immutable, while values can be of any type and are mutable.

Each container is accessed using square brackets, with indices for tuples, lists, and strings (i.e., ordered) and keys for dictionaries (i.e., unordered).

b) Instantiate each with 0 elements (i.e., empty), and show adding a single element to each.

In [2]:

```
# instantiate empty containers
li = []
tu = ()
st = ""
di = {}

# add single elements
li.append(1)
tu = tu + (1,)
st += "1"
di[1] = 1
```

c) Provide 1 or more use cases for each.

Lists are useful for storing ordered data that can have elements added and removed at any time, such as storing a list of objects used for a model. Tuples' immutability makes them useful for storing constant structures, such as storing origin/reference coordinates. Strings are used for anything involving text, like storing textual data. Dictionaries are useful for organizing an object's properties and normalizing them across many objects. They can be used for parsing JSON files into a Python-native datatype.

0.2) Write a program that takes in a positive number (in some variable, say `i`) and computes the sum of all the number between 0 and that number (inclusive).

a) Do it using a for loop

In [3]:

```
i = 10
```

In [4]:

```
def sum_forloop(n):
    """
    Function that returns the sum of all numbers in range [0, n].
    :param n: upper limit of summation, which is positive integer greater t
    :return: Sum from i=0 to i=n.
    """
    if type(n) is not int or n < 0:
        # check input meets conditions
        return None

    sum = 0
    for i in range(n+1):
        sum += i
    return sum

print('Sum using list using for loop', sum_forloop(i))
```

```
Sum using list using for loop 55
```

b) Do it in one line using the function `sum` and list comprehension.

In [5]:

```python
print("Sum using list comprehension", sum([i for i in range(1, i+1)]))
```

```
Sum using list comprehension 55
```

0.3) Create a lookup table for your class schedule, with the CRN as keys and the name of class as the value. Loop over the dictionary and print out the CRN and course name (single line per class).

In [6]:

```python
classList = {
    "CS0023"    : "Game Design",
    "CS0052"    : "Natural Language Processing",
    "CS0135"    : "Machine Learning",
    "CS0150-02" : "Deep Graph Learning",
    "CS0150-09" : "Computer Vision",
    "UEP0173"   : "Transportation Planning",
}

for classCRN in classList:
    print(classCRN, classList[classCRN])
```

```
CS0023 Game Design
CS0052 Natural Language Processing
CS0135 Machine Learning
CS0150-02 Deep Graph Learning
CS0150-09 Computer Vision
UEP0173 Transportation Planning
```

0.4) Create an empty list. Then, copy the for-loop from previous excercise such that the program prompts you to input the time of the day (as type sting, and using military time would allow for AM and PM to be omitted). These times are to be stored in empty dictionay using the same keys (i.e., CRN->time class starts)

In [7]:

```python
classSchedule = {}
for classCRN in classList:
    classSchedule[classCRN] = input(f"What time is {classList[classCRN]}? "
```

```
What time is Game Design? 1800
What time is Natural Language Processing? 1800
What time is Machine Learning? 1030
What time is Deep Graph Learning? 1500
What time is Computer Vision? 1630
What time is Transportation Planning? 1330
```

0.5 Write a Python program to convert temperatures to and from Celsius, Fahrenheit.

$$\frac{c}{5} = \frac{f-32}{9},$$

where c is the temperature in Celsius and f is the temperature in Fahrenheit.

Test code: 60°C is 140 in Fahrenheit 45°F is 7 in Celsius

In [8]:

```
def fahrenheit2celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

def celsius2fahrenheit(celsius):
    return celsius * 9/5 + 32


temp_c = 60
temp_f = 45

temp_c_out = fahrenheit2celsius(temp_f)
temp_f_out = celsius2fahrenheit(temp_c)

print("{} deg. F is {} deg. C".format(temp_f, temp_c_out))
print("{} deg. C is {} deg. F".format(temp_c, temp_f_out))
```

```
45 deg. F is 7.222222222222222 deg. C
60 deg. C is 140.0 deg. F
```

0.6 Write a Python program to construct the following pattern, using a nested for loop.

O

O X

O X O

O X O X

O X O X O

O X O X

O X O

O X

O

In [9]:

```
symbols = ('O', 'X')

for lineNum in range(-4, 5):
    for char in range(5 - abs(lineNum)):
        print(symbols[char % 2], end=' ')
    print()
```

```
O
O X
O X O
O X O X
O X O X O
O X O X
O X O
O X
O
```

0.7 Write a Python program that reads two integers representing a month and day and prints the season for that month and day. Go to the editor Expected Output:

Input the month (e.g. January, February etc.): july
Input the day: 31
Season is summer

In [10]:

```python
months = ['january', 'february', 'march', 'april', 'may', 'june', 'july', '
month = months.index(input("Enter month: ").lower()) + 1
day = int(input("Enter day: "))

seasons = ['winter', 'spring', 'summer', 'autumn']
season = seasons[(month % 12) // 3]

print("The season is", season)
```

```
Enter month: july
Enter day: 31
The season is summer
```

0.8 Implement repeats(), as specified in doc-string. Then call on variables a and b below. Print True if repeated, else, print False.

In [11]:

```python
a = [1, 3, 1, 6, 3, 5, 5, 2]
b = [1, 2, 3, 3, 4, 5, 6, 7, 8, 9]

def repeated_val(xs, val=5):
    """Function to search whether 'val' is repeated in sequence.

    :param xs:      List of items to search
    :param val:     Val being searched (default = 5)

    :return:        True if repeated 'val' and neighbors, i.e., [..., 'val'
    """

    return any([xs[index] == val and xs[index+1] == val for index in range(

print("list 'a' repeats 5:", repeated_val(a))
print("list 'a' repeats 6:", repeated_val(a, val=6))
print("list 'b' repeats 5:", repeated_val(b))
```

```
list 'a' repeats 5: True
list 'a' repeats 6: False
list 'b' repeats 5: False
```

Part 1

The goal in this part is to

- understand basic functionality of numpy and pandas
- learn how to use numpy and pandas to solve common coding tasks

- understand these packages to process real-world data

Import other libraries, such that numpy library is called by with np and pandas with pd

In [12]:

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

a) Numpy Basics

*Make sure to leave random seeds in each cell so that the outputs match the expected answer.

1)

Create a 10x10 array with random values and find the minimum and maximum values

In [13]:

```
np.random.seed(123)

arr = np.random.rand(10, 10)

print("Maximum", np.max(arr))
print("Minimum", np.min(arr))
```

```
Maximum 0.9953584820340174
Minimum 0.01612920669501683
```

2)

Extract the integer part of array Z using 5 different numpy methods

In [14]:

```
np.random.seed(123)
Z = np.random.uniform(0, 10, 10)

# THESE RETURN A PURE INTEGER

# Method 1
intArr1 = np.int_(Z)

# Method 2
intArr2 = Z.astype(np.int32)

# Method 3
intArr3 = np.asarray(Z, dtype=np.int32)

# Method 4
intArr4 = np.array([int(x) for x in Z])

# THIS RETURNS A TRUNCATED FLOAT

# Method 6
intArr6 = np.trunc(Z)
```

Create a vector of size 20 with values spanning (0, 1), i.e., 0 and 1 are excluded.

In [15]:

```
vec = np.linspace(0, 1, 22)[1:-1]
```

Create a random vector of size 15 and sort it

In [16]:

```
np.random.seed(123)

vec = np.sort(np.random.rand(15))
```

Consider two random array A anb B, check if they are equal

In [17]:

```
np.random.seed(123)
A = np.random.randint(0, 2, 5)
B = np.random.randint(0, 2, 5)
```

In [18]:

```
print((A==B).all())
```

```
False
```

matplotlib is the plotting library which pandas' plotting functionality is built upon, and it is usually aliased to plt.

%matplotlib inline tells the notebook to show plots inline, instead of creating them in a separate window.

plt.style.use('ggplot') is a style theme that most people find agreeable, based upon the styling of R's ggplot package.

See the documentation https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html) if you get stuck!

Make an array immutable (read-only)

In [19]:

```
Z = np.zeros(10)
Z.flags.writeable = False
```

What if we want to plot multiple things? Pandas allows you to pass in a matplotlib Axis object for plots, and plots will also return an Axis object.

Make a bar plot of monthly revenue with a line plot of monthly advertising spending (numbers in millions)

Create a structured array representing a position (x,y) and a color (r,g,b). Instantiate structured array's values to be all zeros (though same method for other values as well).

In [20]:

```
width = 256
height = 256

arr = np.zeros((height, width, 3))
```

Considering a four dimensions array, how to get sum over the last two axis at once?

In [21]:

```
np.random.seed(123)
A = np.random.randint(0, 10, (3, 4, 3, 4))

sum = A.sum(axis=(-1, -2))
```

Considering a (w,h,3) image of (dtype=ubyte), compute the number of unique colors

In [22]:

```
np.random.seed(123)
w, h = 16, 16
I = np.random.randint(0, 2, (h, w, 3)).astype(np.ubyte)
print(I.shape)
```

```
(16, 16, 3)
```

In [23]:

```
flattenedImage = I.reshape((h*w, 3))
uniqueColors = np.unique(flattenedImage, axis=0)
len(uniqueColors)
```

Out[23]:

```
8
```

How to accumulate elements of a vector (X) to an array (F) based on an index list (I)?

In [24]:
```
X = [1, 2, 3, 4, 5, 6]
I = [1, 3, 9, 3, 4, 1]
F = np.bincount(I, X)
print(F)
```

```
[0. 7. 0. 6. 5. 0. 0. 0. 0. 3.]
```

In [ ]:
```

```

How to read the following file?

In [25]:
```
fpath = os.path.join("data", "tabular", "missing.dat")
file = open(fpath, "r")
lines = file.readlines()
```

Convert a vector of ints into a matrix binary representation

In [26]:
```
I = np.array([0, 1, 2, 3, 15, 16, 32, 64, 128])
B = ((I.reshape(-1,1) & (2**np.arange(8))) != 0).astype(int)
```

Given a two dimensional array, how to extract unique rows?

In [27]:
```
Z = np.random.randint(0, 2, (6, 3))
print(Z)

uniqueRows = np.unique(Z, axis=0)
```

```
[[0 1 0]
 [0 0 0]
 [0 1 1]
 [0 0 0]
 [0 1 1]
 [0 1 0]]
```

Pandas

Made-up data representing animals and trips to vet

In [28]:
```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'ca
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'n

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

Create a DataFrame df from this dictionary data which has the index labels.

In [29]:

```
df = pd.DataFrame(data, index=labels)
```

Display a summary of the basic information about this DataFrame and its data.

In [30]:

```
df
```

Out[30]:

```
   animal  age  visits  priority
a     cat  2.5       1       yes
b     cat  3.0       3       yes
c   snake  0.5       2        no
d     dog  NaN       3       yes
e     dog  5.0       2        no
f     cat  2.0       3        no
g   snake  4.5       1        no
h     cat  NaN       1       yes
i     dog  7.0       2        no
j     dog  3.0       1        no
```

Return the first 3 rows of the DataFrame df.

In [31]:

```
df.head(3)
```

Out[31]:

```
   animal  age  visits  priority
a     cat  2.5       1       yes
b     cat  3.0       3       yes
c   snake  0.5       2        no
```

Select just the 'animal' and 'age' columns from the DataFrame df.

In [32]:

```
df[["animal", "age"]]
```

Out[32]:

	animal	age
a	cat	2.5
b	cat	3.0
c	snake	0.5
d	dog	NaN
e	dog	5.0
f	cat	2.0
g	snake	4.5
h	cat	NaN
i	dog	7.0
j	dog	3.0

Change the age in row 'f' to 1.5.

In [33]:

```
df.loc[["f"], ["age"]] = 1.5
```

Calculate the mean age for each different animal in df.

In [34]:

```
df["age"].mean()
```

Out[34]:

```
3.375
```

In the 'animal' column, change the 'snake' entries to 'python'.

In [35]:

```
df.loc[df["animal"] == "snake", "animal"] = "python"
```

For each animal type and each number of visits, find the mean age. In other words, each row is an animal, each column is a number of visits and the values are the mean ages (hint: use a pivot table).

In [36]:

```
pd.pivot_table(df, values="age", index=["animal"], columns=["visits"], aggf
```

Out[36]:

visits	**1**	**2**	**3**
animal			
cat	2.5	NaN	2.25
dog	3.0	6.0	NaN
python	4.5	0.5	NaN

Given a DataFrame, subtract the row mean from each element in the row?

In [37]:

```
# a 5x3 frame of float values
df_floats = pd.DataFrame(np.random.random(size=(5, 3)))
df_floats.sub(df_floats.mean(axis=1), axis=0)
```

Out[37]:

	0	1	2
0	0.117260	-0.103754	-0.013506
1	-0.072643	0.137904	-0.065261
2	0.318477	-0.071004	-0.247473
3	0.213906	-0.404635	0.190729
4	-0.368056	0.295641	0.072415

Series and Datetimeindex

Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers. Let's call this Series s.

In [38]:

```
businessDays = pd.date_range(start="2015-01-01", end="2016-01-01", freq="B"
s = pd.Series(np.random.rand(len(businessDays)), index=businessDays)
```

Find the sum of the values in s for every Wednesday.

In [39]:

```
s[s.index.weekday == 3].sum()
```

Out[39]:

```
30.56137413754528
```

For each calendar month in s, find the mean of values.

In [40]:

```
s.groupby(s.index.month).mean()
```

Out[40]:

```
1     0.525470
2     0.411958
3     0.482855
4     0.465519
5     0.586244
6     0.459178
7     0.499775
8     0.579787
9     0.509425
10    0.545982
11    0.496362
12    0.549904
dtype: float64
```

For each group of four consecutive calendar months in s, find the date on which the highest value occurred.

```
In [41]: s.groupby(pd.Grouper(freq="4M")).idxmax()
```

```
Out[41]: 2015-01-31   2015-01-27
         2015-05-31   2015-04-03
         2015-09-30   2015-06-10
         2016-01-31   2015-11-18
         Freq: 4M, dtype: datetime64[ns]
```

Cleaning Data

The DataFrame to use in the following puzzles:

```
In [42]: df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_Stoc
                                        'Budapest_PaRis', 'Brussels_londOn'],
                            'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
                            'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                                 'Airline': ['KLM(!)', '<Air France> (12)', '(British Air
                                        '12. Air France', '"Swiss Air"']})

         df.head()
```

Out[42]:

	From_To	FlightNumber	RecentDelays	Airline
0	LoNDon_paris	10045.0	[23, 47]	KLM(!)
1	MAdrid_miLAN	NaN	[]	<Air France> (12)
2	londON_StockhOlm	10065.0	[24, 43, 87]	(British Airways.)
3	Budapest_PaRis	NaN	[13]	12. Air France
4	Brussels_londOn	10085.0	[67, 32]	"Swiss Air"

Some values in the the FlightNumber column are missing. These numbers are meant to increase by 10 with each row so 10055 and 10075 need to be put in place. Fill in these missing numbers and make the column an integer column (instead of a float column).

```
In [43]: df["FlightNumber"] = df["FlightNumber"].interpolate().astype(int)
```

The From_To column would be better as two separate columns! Split each string on the underscore delimiter _ to give a new temporary DataFrame with the correct values. Assign the correct column names to this temporary DataFrame.

```
In [44]: df[["From", "To"]] = df["From_To"].str.split("_", expand=True)
```

Notice how the capitalisation of the city names is all mixed up in this temporary DataFrame. Standardise the strings so that only the first letter is uppercase (e.g. "londON" should become "London".)

In [45]:

```
df[["From", "To"]] = df[["From", "To"]].applymap(lambda x: x.capitalize())
```

Delete the From_To column from df and attach the temporary DataFrame from the previous questions.

In [46]:

```
df = df.drop("From_To", axis=1)
```

Plotting

Pandas is integrated with the plotting library matplotlib, and makes plotting DataFrames very user-friendly! Plotting in a notebook environment usually makes use of the following boilerplate:

matplotlib is the plotting library which pandas' plotting functionality is built upon, and it is usually aliased to plt.

%matplotlib inline tells the notebook to show plots inline, instead of creating them in a separate window.

plt.style.use('ggplot') is a style theme that most people find agreeable, based upon the styling of R's ggplot package.

See the documentation [https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html)](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.plot.html) if you get stuck!

In [47]:

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
```

In [48]:

```
df = pd.DataFrame({"xs":[1,5,2,8,1], "ys":[4,2,1,9,6]})
```

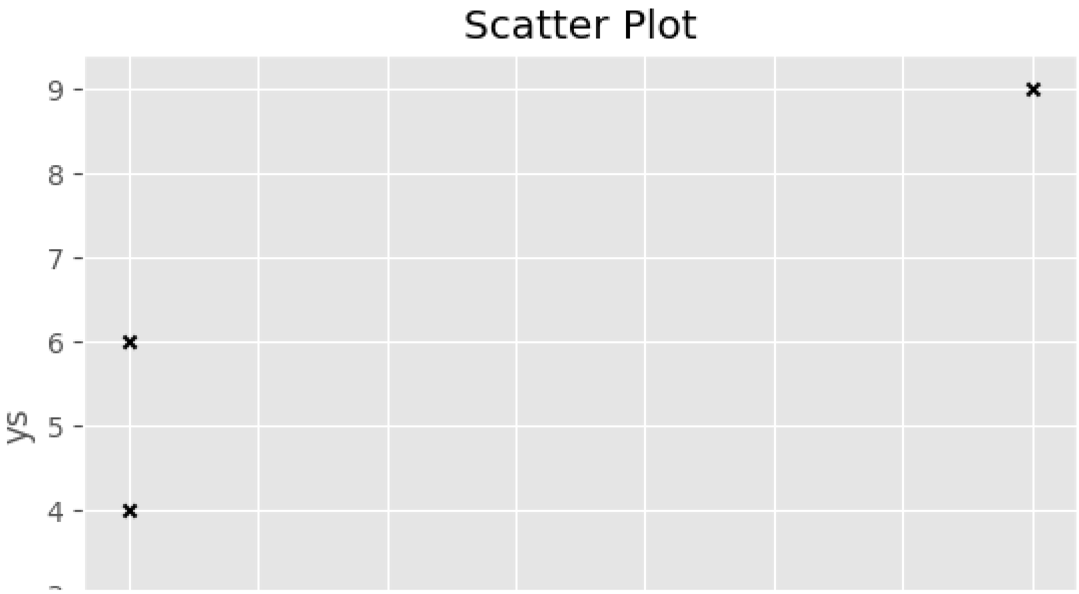
1.31)

For starters, make a scatter plot of this random data, but use black X's instead of the default markers. Add title "Scatter Plot" to the plot. Use df from previous cell.

NOTE: Don't forget to add [any] title and axes labels

```
In [49]: df.plot(kind="scatter", x="xs", y="ys", title="Scatter Plot", marker="x", c
```

```
Out[49]: <AxesSubplot: title={'center': 'Scatter Plot'}, xlabel='xs', ylabel='ys'>
```



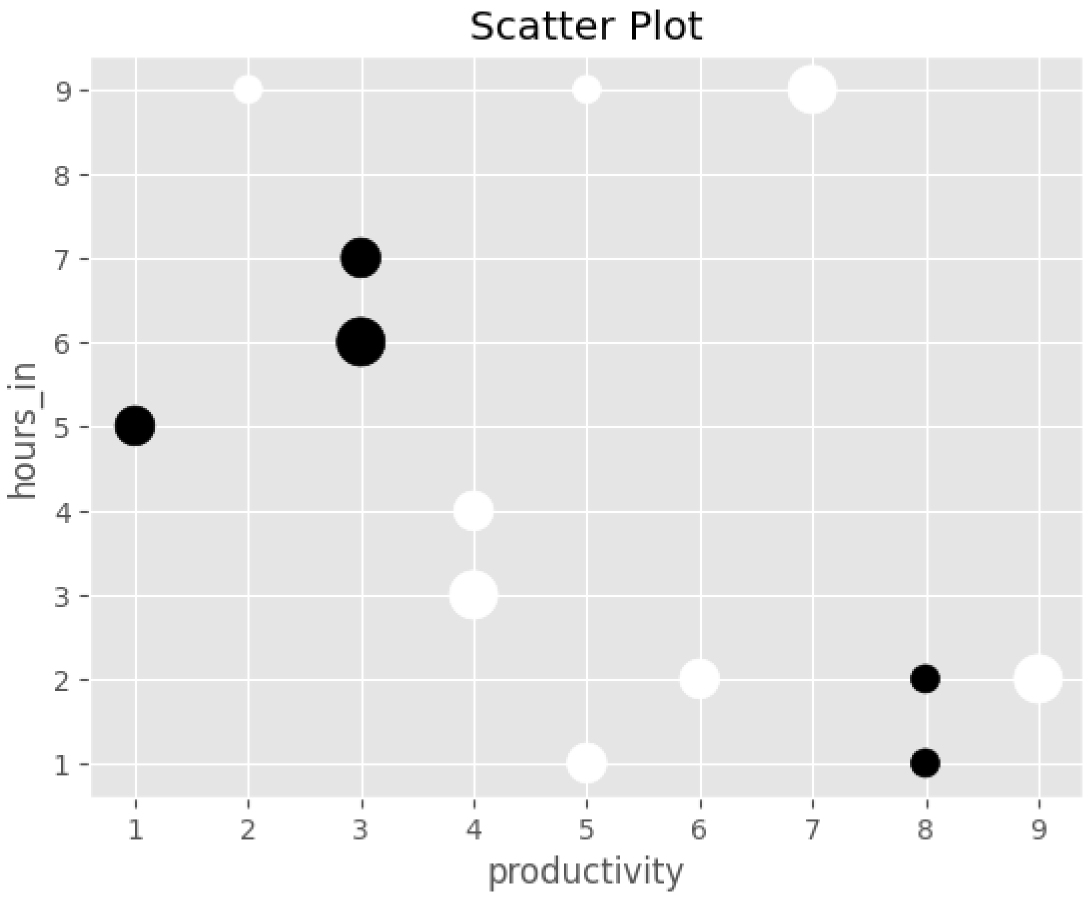
Columns in your DataFrame can also be used to modify colors and sizes. Bill has been keeping track of his performance at work over time, as well as how good he was feeling that day, and whether he had a cup of coffee in the morning. Make a plot which incorporates all four features of this DataFrame.

(Hint: If you're having trouble seeing the plot, try multiplying the Series which you choose to represent size by 10 or more)

In [50]:
```
df2 = pd.DataFrame({"productivity":[5,2,3,1,4,5,6,7,8,3,4,8,9],
                   "hours_in"    :[1,9,6,5,3,9,2,9,1,7,4,2,2],
                   "happiness"   :[2,1,3,2,3,1,2,3,1,2,2,1,3],
                   "caffeinated" :[0,0,1,1,0,0,0,0,1,1,0,1,0]})

df2.plot(kind="scatter", x="productivity", y="hours_in", title="Scatter Plo
```

Out[50]:
```
<AxesSubplot: title={'center': 'Scatter Plot'}, xlabel='productivity', yl
abel='hours_in'>
```



1.33)

What if we want to plot multiple things? Pandas allows you to pass in a matplotlib Axis object for plots, and plots will also return an Axis object.

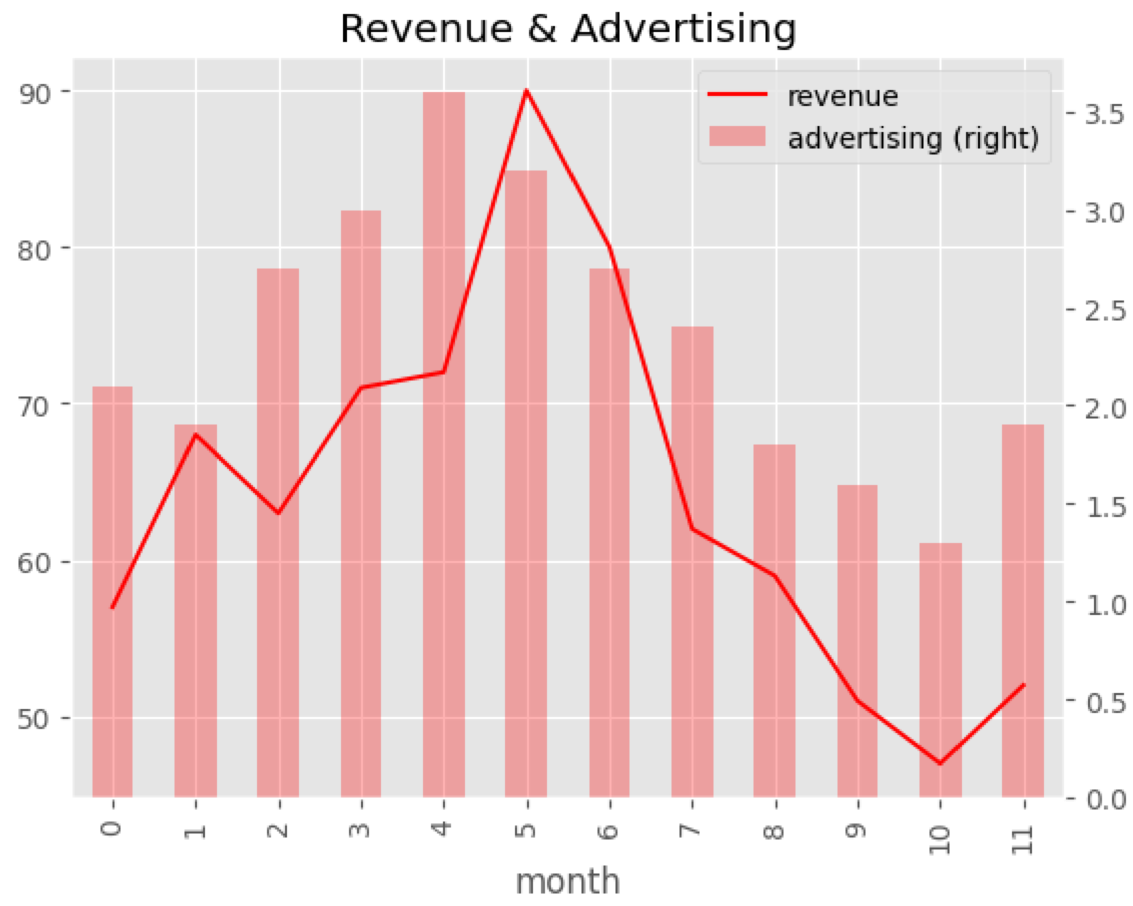
Make a bar plot of monthly revenue with a line plot of monthly advertising spending (numbers in millions)

- Two plots should be in one figure
- Make sure that the y-axis scales of 2 plots are different
- Be sure to include legend

```
In [51]: df_to_plot = pd.DataFrame({"revenue":[57,68,63,71,72,90,80,62,59,51,47,52],
                                    "advertising":[2.1,1.9,2.7,3.0,3.6,3.2,2.7,2.4,1.8,1.6,1
                                    "month":range(12)
                                   })

         axis = df_to_plot.plot(kind="line", x="month", y="revenue", title="Revenue
         df_to_plot.plot(kind="bar", x="month", y="advertising", ax=axis, secondary_
```

```
Out[51]: <AxesSubplot: >
```



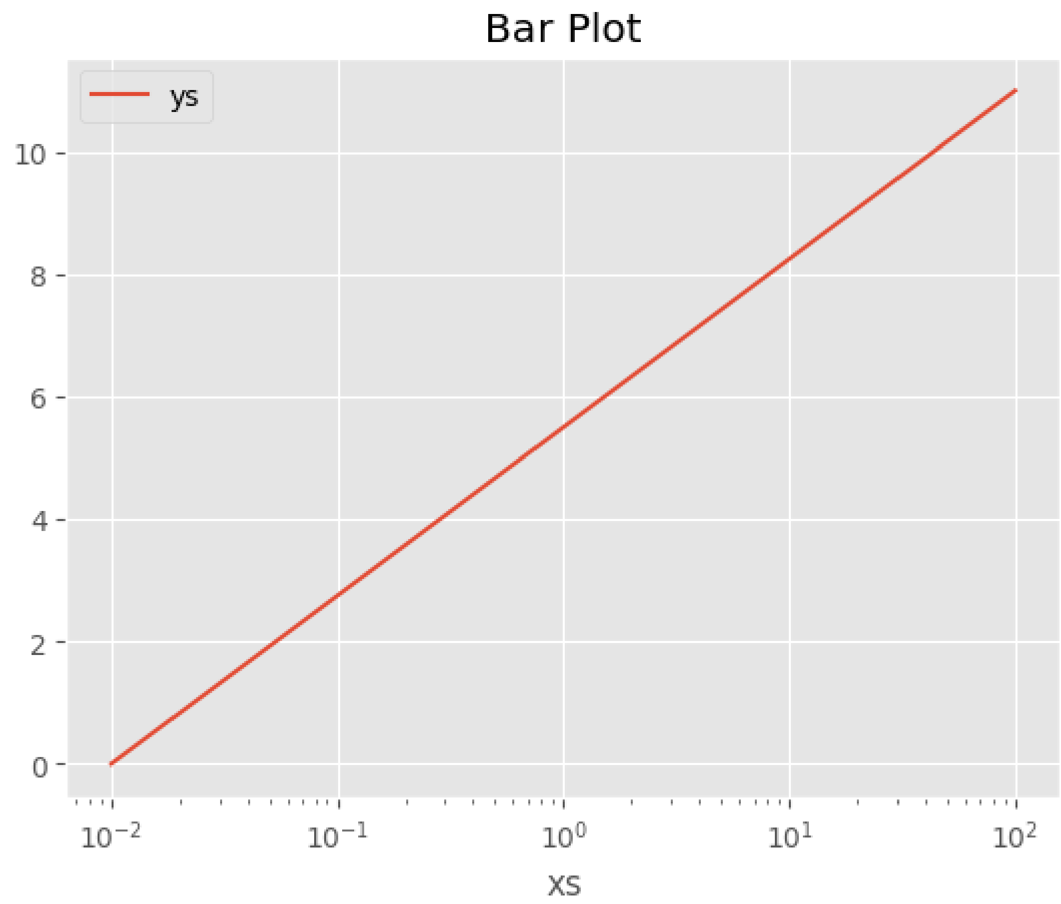
1.33)

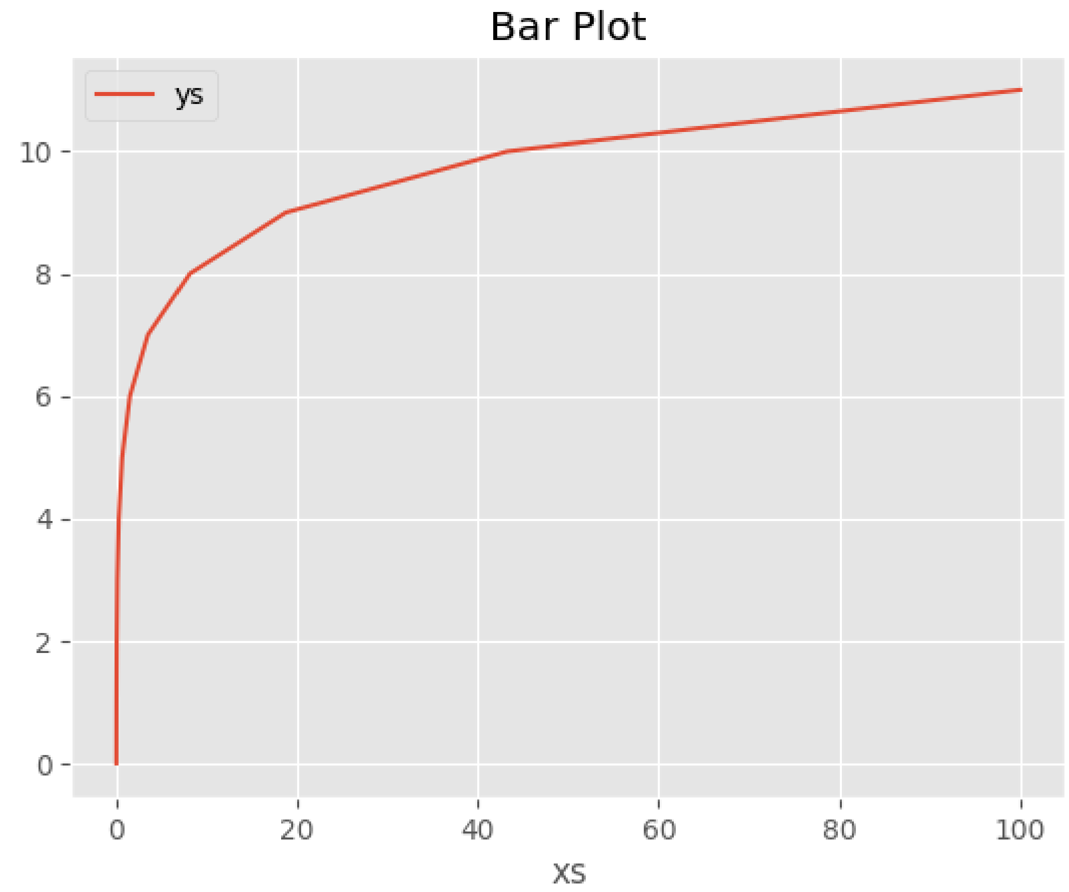
What if we want to put the x-axis in a different scale? Create two line plots with xs as x-axis and ys as y-axis. First plot uses log scaling on x-axis, and the second plot uses default scaling on x-axis.

```
In [52]: df3 = pd.DataFrame({"xs":np.logspace(-2, 2, base=10, num=12),
                             "ys":range(12)
                            })

         df3.plot(kind="line", x="xs", y="ys", title="Bar Plot", logx=True)
         df3.plot(kind="line", x="xs", y="ys", title="Bar Plot")

Out[52]: <AxesSubplot: title={'center': 'Bar Plot'}, xlabel='xs'>
```





Matrix Manipulations

Lets first create a matrix and perform some manipulations of it.

Using numpy's matrix data structure, define the following matricies:

$$A = \begin{bmatrix} 3 & 5 & 9 \\ 3 & 3 & 4 \\ 5 & 9 & 17 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

After this solve the matrix equation:

$$Ax = B$$

In [53]:

```
A = np.matrix([ [3, 5, 9],
                [3, 3, 4],
                [5, 9, 17] ])

B = np.matrix([ [2], [1], [4] ])

np.linalg.solve(A, B)
```

Out[53]:

```
matrix([[ 1.],
        [-2.],
        [ 1.]])
```

Now write three functions for matrix multiply $C = AB$ in each of the following styles:

1. By using nested for loops to impliment the naive algorithm ($C_{ij} = \sum_{k=0}^{m-1} A_{ik} B_{kj}$)
2. Using numpy's built in martrix multiplication

Both methods should have the same answer

In [54]:

```
np.matrix([ [np.sum([ A[i, k] * B[k, j] for k in range(A.shape[1]) ])] for
```

Out[54]:

```
matrix([[47],
        [25],
        [87]])
```

In [55]:

```
np.matmul(A, B)
```

Out[55]:

```
matrix([[47],
        [25],
        [87]])
```

Part 2

Getting used to the data

In [56]:

```
# Reads text file and uses '|' as separator
auto = pd.read_table('data/tabular/auto_mpg.txt', sep='|')
auto.head()
```

Out[56]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino

Answer the following questions about the data:

a) What is the shape of the data?

In [57]:

```
auto.shape
```

Out[57]:

```
(392, 9)
```

b) How many rows and columns are there?

In [58]:

```
rows = auto.shape[0]
cols = auto.shape[1]
```

c) What variables are available?

In [59]:

```
auto.columns
```

Out[59]:

```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'car_name'],
      dtype='object')
```

d) What are the ranges for the values in each numeric column?

In [60]:
```
numericCols = auto[auto.select_dtypes(include=np.number).columns]
numericCols.max() - numericCols.min()
```

Out[60]:
```
mpg               37.6
cylinders          5.0
displacement     387.0
horsepower       184.0
weight          3527.0
acceleration      16.8
model_year        12.0
origin             2.0
dtype: float64
```

e) What is the average value for each column? Does that differ significantly from the median?

In [62]:
```
auto.agg(['mean', 'median'])
```

```
/var/folders/9j/ddzkzm7x2bv5txk3k7qblr_80000gn/T/ipykernel_12810/23389185
0.py:1: FutureWarning: ['car_name'] did not aggregate successfully. If an
y error is raised this will raise in a future version of pandas. Drop the
se columns/ops to avoid this warning.
  auto.agg(['mean', 'median'])
```

Out[62]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	
mean	23.445918	5.471939	194.41199	104.469388	2977.584184	15.541327	75.979592	1.5
median	22.750000	4.000000	151.00000	93.500000	2803.500000	15.500000	76.000000	1.0

Answer the following questions about the data:

a) Which 5 cars get the best gas mileage?

In [63]:
```
auto.sort_values(by="mpg", ascending=False).head(5)
```

Out[63]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
320	46.6	4	86.0	65	2110	17.9	80	3	mazda glc
327	44.6	4	91.0	67	1850	13.8	80	3	honda civic 1500 gl
323	44.3	4	90.0	48	2085	21.7	80	2	vw rabbit c (diesel)
388	44.0	4	97.0	52	2130	24.6	82	2	vw pickup
324	43.4	4	90.0	48	2335	23.7	80	2	vw dasher (diesel)

b) Which 5 cars with more than 4 cylinders get the best gas mileage?

In [64]:

```
auto[auto.cylinders > 4].sort_values(by="mpg", ascending=False).head(5)
```

Out[64]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
381	38.0	6	262.0	85	3015	17.0	82	1	oldsmobile cutlass ciera (diesel)
325	36.4	5	121.0	67	2950	19.9	80	2	audi 5000s (diesel)
330	32.7	6	168.0	132	2910	11.4	80	3	datsun 280-zx
355	30.7	6	145.0	76	3160	19.6	81	2	volvo diesel
304	28.8	6	173.0	115	2595	11.3	79	1	chevrolet citation

c) Which 5 cars get the worst gas mileage?

In [65]:

```
auto.sort_values(by="mpg", ascending=True).head(5)
```

Out[65]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
28	9.0	8	304.0	193	4732	18.5	70	1	hi 1200d
26	10.0	8	307.0	200	4376	15.0	70	1	chevy c20
25	10.0	8	360.0	215	4615	14.0	70	1	ford f250
27	11.0	8	318.0	210	4382	13.5	70	1	dodge d200
123	11.0	8	350.0	180	3664	11.0	73	1	oldsmobile omega

d) Which 5 cars with 4 or fewer cylinders get the worst gas mileage?

In [66]:

```
auto[auto.cylinders <= 4].sort_values(by="mpg", ascending=True).head(5)
```

Out[66]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	car_name
110	18.0	3	70.0	90	2124	13.5	73	3	maxda rx3
75	18.0	4	121.0	112	2933	14.5	72	2	volvo 145e (sw)
119	19.0	4	121.0	112	2868	15.5	73	2	volvo 144ea
70	19.0	3	70.0	97	2330	13.5	72	3	mazda rx2 coupe
111	19.0	4	122.0	85	2310	18.5	73	1	ford pinto

Part 4 Use groupby and aggregations to explore the relationships between mpg and the other variables. Which variables seem to have the greatest effect on mpg? Some examples of things you might want to look at are:

- What is the mean mpg for cars for each number of cylindres (i.e. 3 cylinders, 4 cylinders, 5 cylinders, etc)?
- Did mpg rise or fall over the years contained in this dataset?
- What is the mpg for the group of lighter cars vs the group of heaver cars? Note: Be creative in the ways in which you divide up the data. You are trying to create segments of the data using logical filters and comparing the mpg for each segment of the data.

```
In [67]: auto.groupby("cylinders").agg({"mpg": "mean"})
```

```
Out[67]:
                          mpg
          cylinders
                  3  20.550000
                  4  29.283920
                  5  27.366667
                  6  19.973494
                  8  14.963107
```

Let's now look how MPG has changed over time, while also considering how specific groups have changed-- look at low, mid, and high power cars based upon their horsepower and see how these groups have changed over time.

In his data, he called the original dataset 'auto'.

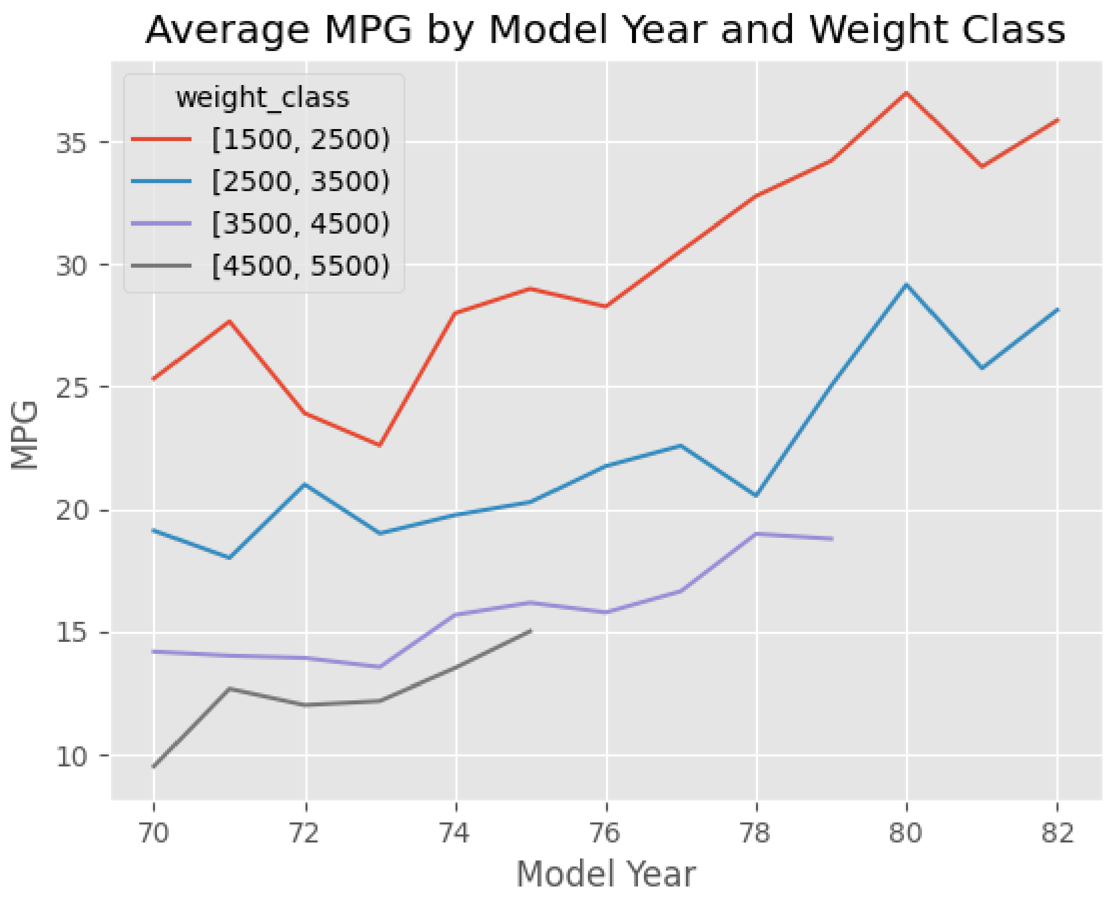
Now to look at how efficency has changed over time based on power and weight classes, two things that we know play a large role in gas mileage. First, we create a table of efficiency by power class and year.

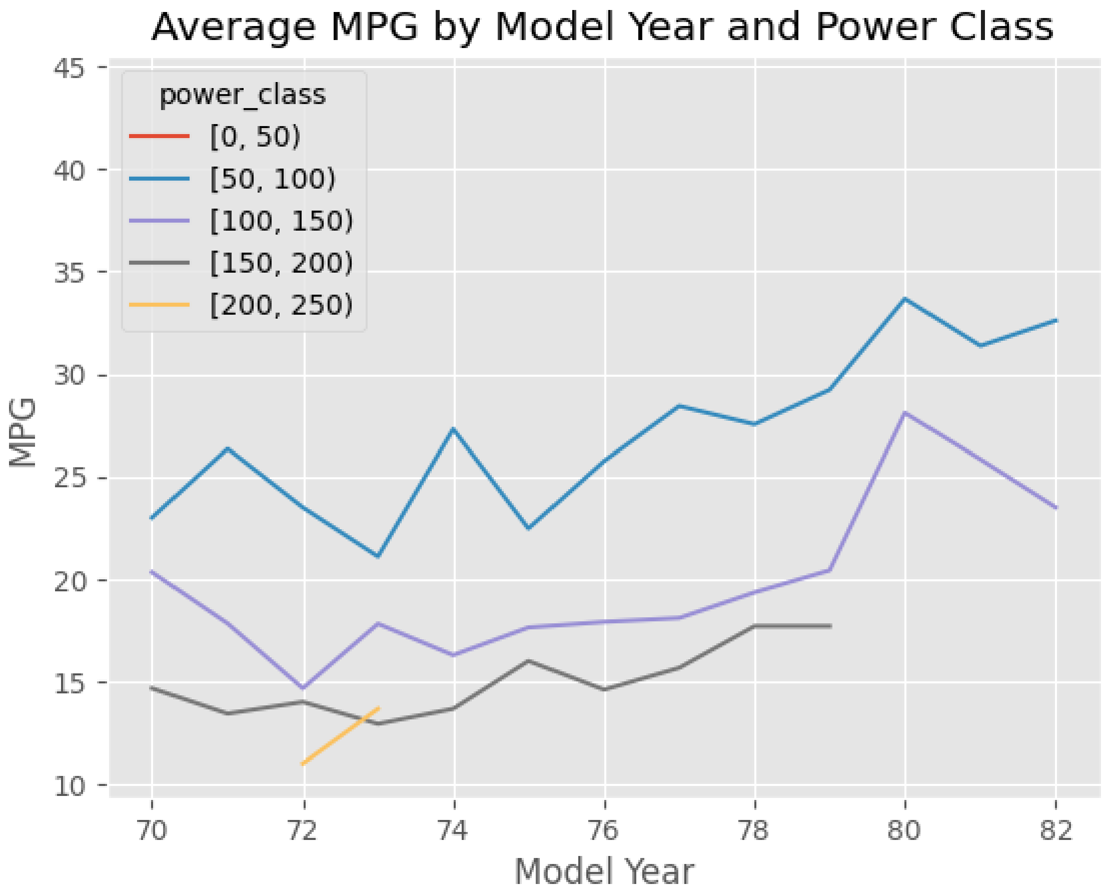
```
In [68]: auto["power_class"] = pd.cut(auto.horsepower, range(0, 300, 50), right=Fals
         auto["weight_class"] = pd.cut(auto.weight, range(1500, 6500, 1000), right=F

         pivot = auto.pivot_table(index="model_year", columns=["weight_class", "powe

         auto.pivot_table(index="model_year", columns=["weight_class"], values="mpg"
         auto.pivot_table(index="model_year", columns=["power_class"], values="mpg",
```

```
Out[68]: <AxesSubplot: title={'center': 'Average MPG by Model Year and Power Clas
         s'}, xlabel='Model Year', ylabel='MPG'>
```




Average MPG by Model Year and Power Class
power_class
[0, 50)
[50, 100)
[100, 150)
[150, 200)
[200, 250)
MPG
45
40
35
30
25
20
15
10
70
72
74
76
78
80
82
Model Year

In [ ]: