Problem Set #3 (BDAT 1004)

By Daryoush Shabahang

August 19, 2021

Question 1 Occupations - Python with Pandas

In [161…

```
# Step 1. Import the necessary libraries
import pandas as pd

# Step 2. Import the dataset from this address.
df1 = pd.read_csv(r'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user', "|")

# Step 3. Assign it to a variable called users
# setting the index of the Data Frame to "user_id"
users = df1.set_index(['user_id'])
users.head()
```

Out[161…

	age	gender	occupation	zip_code
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213

In [162…

```
# Step 4. Discover what is the mean age per occupation

users.groupby('occupation').mean(['age'])
```

Out[162…

	age
occupation	
administrator	38.746835
artist	31.392857
doctor	43.571429
educator	42.010526
engineer	36.388060
entertainment	29.222222
executive	38.718750
healthcare	41.562500
homemaker	32.571429
lawyer	36.750000
librarian	40.000000
marketing	37.615385
none	26.555556
other	34.523810
programmer	33.121212
retired	63.071429
salesman	35.666667
scientist	35.548387

	age
occupation	
student	22.081633

In [166…

```
# Step 5. Discover the Male ratio per occupation and sort it from the most to the least

maleRatio = pd.pivot_table(users, aggfunc = 'count', index = 'occupation', values = 'age', column
```

In [168…

```
# Step 5 (cont)

# determine the total number of individuals of each occupation
total = maleRatio[['M','F']].sum(axis = 1)

maleRatio['maleRatio'] = (maleRatio['M'] / total)

maleRatio.sort_values(by = ['maleRatio'], ascending = False)
```

Out[168…

gender	F	M	maleRatio
occupation			
doctor	0	7	1.000000
engineer	2	65	0.970149
technician	1	26	0.962963
retired	1	13	0.928571
programmer	6	60	0.909091
executive	3	29	0.906250
scientist	3	28	0.903226
entertainment	2	16	0.888889
lawyer	2	10	0.833333
salesman	3	9	0.750000
educator	26	69	0.726316
student	60	136	0.693878
other	36	69	0.657143
marketing	10	16	0.615385
writer	19	26	0.577778
none	4	5	0.555556
administrator	36	43	0.544304
artist	13	15	0.535714
librarian	29	22	0.431373
healthcare	11	5	0.312500
homemaker	6	1	0.142857

In [169…

```
# Step 6. For each occupation, calculate the minimum and maximum ages

users.groupby('occupation').agg({'age':['min','max']})
```

Out[169…

	age	
	min	max
occupation		
administrator	21	70
artist	19	48
doctor	28	64
educator	23	63

	age	
	min	max
occupation		
engineer	22	70
entertainment	15	50
executive	22	69
healthcare	22	62
homemaker	20	50
lawyer	21	53
librarian	23	69
marketing	24	55
none	11	55
other	13	64
programmer	20	63
retired	51	73
salesman	18	66
scientist	23	55

In [170…

```
# Step 7. For each combination of occupation and sex, calculate the mean age

users.groupby(['occupation','gender']).agg({'age':'mean'})
```

Out[170…

		age
occupation	gender	
administrator	F	40.638889
	M	37.162791
artist	F	30.307692
	M	32.333333
doctor	M	43.571429
educator	F	39.115385
	M	43.101449
engineer	F	29.500000
	M	36.600000
entertainment	F	31.000000
	M	29.000000
executive	F	44.000000
	M	38.172414
healthcare	F	39.818182
	M	45.400000
homemaker	F	34.166667
	M	23.000000
lawyer	F	39.500000
	M	36.200000
librarian	F	40.000000
	M	40.000000
marketing	F	37.200000
	M	37.875000

		age
occupation	**gender**	
none	**F**	36.500000
	M	18.600000
other	**F**	35.472222
	M	34.028986
programmer	**F**	32.166667
	M	33.216667
retired	**F**	70.000000
	M	62.538462
salesman	**F**	27.000000
	M	38.555556
scientist	**F**	28.333333
	M	36.321429
student	**F**	20.750000
	M	22.669118
technician	**F**	38.000000
	M	32.961538

In [171…

```
# Step 8. For each occupation present the percentage of women and men

# references:
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.round.html
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html

# reusing some of the code from step 5:
maleRatio = pd.pivot_table(users, aggfunc = 'count', index = 'occupation', values = 'age', column
# determine the percentage of males -> multiple ratio by 100 to get %
maleRatio['male %'] = (maleRatio['M'] / total) * 100

# determine percentage of females
maleRatio['male %'] = (1 - (maleRatio['M'] / total)) * 100

# round percentage to the nearest tenth
# remove raw values from the table
maleRatio.round(decimals = 1).drop(columns = ['M', 'F'], axis = 0)
```

Out[171…

gender	**male %**
occupation	
administrator	45.6
artist	46.4
doctor	0.0
educator	27.4
engineer	3.0
entertainment	11.1
executive	9.4
healthcare	68.8
homemaker	85.7
lawyer	16.7
librarian	56.9
marketing	38.5
none	44.4
other	34.3

gender	male %
occupation	
programmer	9.1
retired	7.1
salesman	25.0
scientist	9.7
student	30.6

Question 2 Euro Teams - Python with Pandas

In [173…

```
# Step 1. Import the necessary libraries
import pandas as pd
import numpy as np

# Step 2. Import the dataset from this address
# Step 3. Assign it to a variable called euro12
euro12 = pd.read_csv(r'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/02_Fi

euro12.head()
```

Out[173…

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	F Conc
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	13	81.3%	41	
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	9	60.1%	53	
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	10	66.7%	25	
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	22	88.1%	43	
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	6	54.6%	36	

5 rows × 35 columns

In [174…

```
# Step 4. Select only the Goal column

euro12['Goals']
```

Out[174…

```
0      4
1      4
2      4
3      5
4      3
5     10
6      5
7      6
8      2
9      2
10     6
11     1
12     5
13    12
14     5
15     2
Name: Goals, dtype: int64
```

In [175…

```
# Step 5. How many team participated in the Euro2012?

euro12['Team'].count()
```

Out[175…

```
16
```

In [176…

```
# Step 6. What is the number of columns in the dataset?

len(euro12.columns)
```

Out[176…

```
35
```

In [177…

```
# Step 7. View only the columns Team, Yellow Cards and Red Cards and assign them to a dataframe c

discipline = euro12[['Team','Yellow Cards', 'Red Cards']]
discipline
```

Out[177…

	Team	Yellow Cards	Red Cards
0	Croatia	9	0
1	Czech Republic	7	0
2	Denmark	4	0
3	England	5	0
4	France	6	0
5	Germany	4	0
6	Greece	9	1
7	Italy	16	0
8	Netherlands	5	0
9	Poland	7	1
10	Portugal	12	0
11	Republic of Ireland	6	1
12	Russia	6	0
13	Spain	11	0
14	Sweden	7	0
15	Ukraine	5	0

In [178…

```
# Step 8. Sort the teams by Red Cards, then to Yellow Cards

discipline.sort_values(['Red Cards','Yellow Cards'], ascending = [True, True])
```

Out[178…

	Team	Yellow Cards	Red Cards
2	Denmark	4	0
5	Germany	4	0
3	England	5	0
8	Netherlands	5	0
15	Ukraine	5	0
4	France	6	0
12	Russia	6	0
1	Czech Republic	7	0
14	Sweden	7	0
0	Croatia	9	0
13	Spain	11	0
10	Portugal	12	0
7	Italy	16	0
11	Republic of Ireland	6	1
9	Poland	7	1
6	Greece	9	1

In [179…

```
# Step 9. Calculate the mean Yellow Cards given per Team

euro12.groupby('Team').agg({'Yellow Cards': 'mean'})
```

Out[179…

	Yellow Cards
Team	
Croatia	9
Czech Republic	7
Denmark	4
England	5
France	6
Germany	4
Greece	9
Italy	16
Netherlands	5
Poland	7
Portugal	12
Republic of Ireland	6
Russia	6
Spain	11
Sweden	7
Ukraine	5

In [180…

```
# Step 10. Filter teams that scored more than 6 goals

euro12.loc[euro12['Goals'] > 6]
```

Out[180…

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	Conc
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	15	93.8%	102	

2 rows × 35 columns

In [181…

```
# goalsStep 11. Select the teams that start with G

euro12[euro12['Team'].str.startswith('G')]
```

Out[181…

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Saves made	Saves-to-shots ratio	Fouls Won	F Conc
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	13	65.1%	67	

2 rows × 35 columns

In [182…

```
# Step 12. Select the first 7 columns

euro12[euro12.columns[0:7]]
```

Out[182…

	Team	Goals	Shots on	Shots off	Shooting	% Goals-to-	Total shots (inc.

			target	target	Accuracy	shots	Blocked)
0	Croatia	4	13	12	51.9%	16.0%	32
1	Czech Republic	4	13	18	41.9%	12.9%	39
2	Denmark	4	10	10	50.0%	20.0%	27
3	England	5	11	18	50.0%	17.2%	40
4	France	3	22	24	37.9%	6.5%	65
5	Germany	10	32	32	47.8%	15.6%	80
6	Greece	5	8	18	30.7%	19.2%	32
7	Italy	6	34	45	43.0%	7.5%	110
8	Netherlands	2	12	36	25.0%	4.1%	60
9	Poland	2	15	23	39.4%	5.2%	48
10	Portugal	6	22	42	34.3%	9.3%	82
11	Republic of Ireland	1	7	12	36.8%	5.2%	28
12	Russia	5	9	31	22.5%	12.5%	59
13	Spain	12	42	33	55.9%	16.0%	100
14	Sweden	5	17	19	47.2%	13.8%	39
15	Ukraine	2	7	26	21.2%	6.0%	38

In [183…

```
# Step 13. Select all columns except the last 3

euro12[euro12.columns[:-3]]
```

Out[183…

	Team	Goals	Shots on target	Shots off target	Shooting Accuracy	% Goals-to-shots	Total shots (inc. Blocked)	Hit Woodwork	Penalty goals	Penalties not scored	...	Clean Sheets	Blocks	Goa conced
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	0	10	
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	1	10	
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	1	10	
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	2	29	
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	1	7	
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	1	11	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	1	23	
7	Italy	6	34	45	43.0%	7.5%	110	2	0	0	...	2	18	
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	0	...	0	9	
9	Poland	2	15	23	39.4%	5.2%	48	0	0	0	...	0	8	
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	0	...	2	11	
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	0	...	0	23	
12	Russia	5	9	31	22.5%	12.5%	59	2	0	0	...	0	8	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	5	8	
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	0	...	1	12	
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	0	...	0	4	

16 rows × 32 columns

In [184…

```
# Step 14. Present only the Shooting Accuracy from England, Italy and Russia

euro12[euro12.Team.isin(['England','Italy','Russia'])][['Team','Shooting Accuracy']]
```

Out[184…

	Team	Shooting Accuracy
3	England	50.0%
7	Italy	43.0%
12	Russia	22.5%

Question 3 Housing - Python with Pandas

In [185…

```
# Step 1. Import the necessary libraries
import numpy as np
import pandas as pd
import random

# Step 2. Create 3 different Series, each of length 100, as follows:
# • The first a random number from 1 to 4

series1 = pd.Series(np.random.randint(1,5, size = 100))
series1
```

Out[185…

```
0      2
1      1
2      1
3      2
4      2
      ..
95     3
96     3
97     4
98     3
99     2
Length: 100, dtype: int32
```

In [186…

```
# Step 2. Create 3 differents Series, each of length 100, as follows:
# • The second a random number from 1 to 3

series2 = pd.Series(np.random.randint(1,4, size = 100))
series2
```

Out[186…

```
0      3
1      3
2      3
3      2
4      2
      ..
95     3
96     2
97     1
98     1
99     3
Length: 100, dtype: int32
```

In [187…

```
# Step 2. Create 3 differents Series, each of length 100, as follows:
# • The third a random number from 10,000 to 30,000

series3 = pd.Series(np.random.randint(10000,30001, size = 100))
series3
```

Out[187…

```
0      19542
1      28145
2      12789
3      18538
4      25725
       ...
95     11041
96     22085
97     29262
98     12849
99     28913
Length: 100, dtype: int32
```

In [188…

```
# Step 3. Create a DataFrame by joining the Series by column

housing_data_frame = pd.DataFrame({'series1': series1, 'series2': series2, 'series3': series3})
housing_data_frame
```

Out[188…

	series1	series2	series3
0	2	3	19542
1	1	3	28145
2	1	3	12789
3	2	2	18538
4	2	2	25725
...	...	...	...
95	3	3	11041
96	3	2	22085
97	4	1	29262
98	3	1	12849
99	2	3	28913

100 rows × 3 columns

In [189…

```
# Step 4. Change the name of the columns to bedrs, bathrs, price_sqr_meter

# add all three series together in a dictionary
# assign a column label to each series

housing_data_frame.columns = ["bedrs", "bathrs", "price_sqr_meter"]
housing_data_frame
```

Out[189…

	bedrs	bathrs	price_sqr_meter
0	2	3	19542
1	1	3	28145
2	1	3	12789
3	2	2	18538
4	2	2	25725
...	...	...	...
95	3	3	11041
96	3	2	22085
97	4	1	29262
98	3	1	12849
99	2	3	28913

100 rows × 3 columns

In [190…

```
# Step 5. Create a one column DataFrame with the values of the 3 Series and assign it to 'bigcolu

bigcolumn = pd.concat([series1, series2, series3])
bigcolumn
```

Out[190…

```
0         2
1         1
2         1
3         2
4         2
      ...
95    11041
96    22085
97    29262
```

```
98    12849
99    28913
Length: 300, dtype: int32
```

In [191…

```
# Step 6. Ops it seems it is going only until index 99. Is it true?

# due to the contactenation from the previous step, the indices for each series is only considere
# showing unique indices. That's why instead of showing an index of 299, it's showing us an index

# We can use these two codes to double check this:
print("The length is:", len(bigcolumn))

if (max(bigcolumn.index)==99):
    print("True, the index is 99")
else:
    print("False, the index is not 99")
```

```
The length is: 300
True, the index is 99
```

In [192…

```
# Step 7. Reindex the DataFrame so it goes from 0 to 299

# Reference:
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reset_index.html?highlight=reset_

bigcolumn.reset_index(drop = True)
```

Out[192…

```
0          2
1          1
2          1
3          2
4          2
       ...
295    11041
296    22085
297    29262
298    12849
299    28913
Length: 300, dtype: int32
```

Question 4 Wind Statistics - Python with Pandas

In [193…

```
# Step 1. Import the necessary libraries
import pandas as pd
import numpy as np
import datetime

# Step 2. Import the dataset from this address
wind_stats_data = pd.read_csv(r'https://raw.githubusercontent.com/guipsamora/pandas_exercises/mas
wind_stats_data = wind_stats_data.rename(columns = {'Yr': 'Year', 'Mo': 'Month', 'Dy': 'Day'})

wind_stats_data
```

Out[193…

	Year	Month	Day	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
0	61	1	1	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	61	1	2	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	61	1	3	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	61	1	4	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	61	1	5	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6569	78	12	27	17.58	16.96	17.62	8.08	13.21	11.67	14.46	15.59	14.04	14.00	17.21	40.08
6570	78	12	28	13.21	5.46	13.46	5.00	8.12	9.42	14.33	16.25	15.25	18.05	21.79	41.46
6571	78	12	29	14.00	10.29	14.42	8.71	9.71	10.54	19.17	12.46	14.50	16.42	18.88	29.58
6572	78	12	30	18.50	14.04	21.29	9.13	12.75	9.71	18.08	12.87	12.46	12.12	14.67	28.79
6573	78	12	31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

6574 rows × 15 columns

In [194…

```
# Step 3. Assign it to a variable called data and replace the first 3 columns by a proper datetim

wind_stats_data["Date"] = pd.to_datetime(wind_stats_data[['Year', 'Month', 'Day']].astype(str).ag
wind_stats_data = wind_stats_data.drop(columns = ['Year', 'Month', 'Day'])
column_names = ["Date", "RPT", "VAL", "ROS", "KIL","SHA", "BIR", "DUB", "CLA", "MUL", "CLO", "BEL
wind_stats_data = wind_stats_data.reindex(columns = column_names)
wind_stats_data
```

Out[194…

	Date	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
0	2061-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	2061-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	2061-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	2061-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	2061-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6569	1978-12-27	17.58	16.96	17.62	8.08	13.21	11.67	14.46	15.59	14.04	14.00	17.21	40.08
6570	1978-12-28	13.21	5.46	13.46	5.00	8.12	9.42	14.33	16.25	15.25	18.05	21.79	41.46
6571	1978-12-29	14.00	10.29	14.42	8.71	9.71	10.54	19.17	12.46	14.50	16.42	18.88	29.58
6572	1978-12-30	18.50	14.04	21.29	9.13	12.75	9.71	18.08	12.87	12.46	12.12	14.67	28.79
6573	1978-12-31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

6574 rows × 13 columns

In [196…

```
# Step 4. Year 2061? Do we really have data from this year? Create a function to fix it and apply

def correct_date(col_name):

    if col_name.year > 2000:
        year = col_name.year - 100
    else:
        year = col_name.year
    return datetime.date(year, col_name.month, col_name.day)
```

In [197…

```
# step 4 (cont) (DONE)

wind_stats_data['Date'] = wind_stats_data['Date'].apply(correct_date)
wind_stats_data
```

Out[197…

	Date	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
0	1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1	1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
2	1961-01-03	18.50	16.88	12.33	10.13	11.17	6.17	11.25	NaN	8.50	7.67	12.75	12.71
3	1961-01-04	10.58	6.63	11.75	4.58	4.54	2.88	8.63	1.79	5.83	5.88	5.46	10.88
4	1961-01-05	13.33	13.25	11.42	6.17	10.71	8.21	11.92	6.54	10.92	10.34	12.92	11.83
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6569	1978-12-27	17.58	16.96	17.62	8.08	13.21	11.67	14.46	15.59	14.04	14.00	17.21	40.08
6570	1978-12-28	13.21	5.46	13.46	5.00	8.12	9.42	14.33	16.25	15.25	18.05	21.79	41.46
6571	1978-12-29	14.00	10.29	14.42	8.71	9.71	10.54	19.17	12.46	14.50	16.42	18.88	29.58
6572	1978-12-30	18.50	14.04	21.29	9.13	12.75	9.71	18.08	12.87	12.46	12.12	14.67	28.79
6573	1978-12-31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

6574 rows × 13 columns

In [198…

```
# Step 5. Set the right dates as the index. Pay attention at the data type, it should be datetime

wind_stats_data_new = wind_stats_data.set_index("Date")
wind_stats_data_new.index.astype("datetime64[ns]")
```

Out[198…

```
DatetimeIndex(['1961-01-01', '1961-01-02', '1961-01-03', '1961-01-04',
               '1961-01-05', '1961-01-06', '1961-01-07', '1961-01-08',
               '1961-01-09', '1961-01-10',
               ...
               '1978-12-22', '1978-12-23', '1978-12-24', '1978-12-25',
               '1978-12-26', '1978-12-27', '1978-12-28', '1978-12-29',
               '1978-12-30', '1978-12-31'],
              dtype='datetime64[ns]', name='Date', length=6574, freq=None)
```

In [199…

```
# Step 6. Compute how many values are missing for each location over the entire record. They shou
# all calculations below.

wind_stats_data_new.isnull().sum()
```

Out[199…

```
RPT    6
VAL    3
ROS    2
KIL    5
SHA    2
BIR    0
DUB    3
CLA    2
MUL    3
CLO    1
BEL    0
MAL    4
dtype: int64
```

In [200…

```
# Step 7. Compute how many non-missing values there are in total.

wind_stats_data_new.count()
```

Out[200…

```
RPT    6568
VAL    6571
ROS    6572
KIL    6569
SHA    6572
BIR    6574
DUB    6571
CLA    6572
MUL    6571
CLO    6573
BEL    6574
MAL    6570
dtype: int64
```

In [201…

```
# Step 8. Calculate the mean windspeeds of the windspeeds over all the locations and all the time

wind_stats_data_new.mean()
```

Out[201…

```
RPT    12.362987
VAL    10.644314
ROS    11.660526
KIL     6.306468
SHA    10.455834
BIR     7.092254
DUB     9.797343
CLA     8.495053
MUL     8.493590
CLO     8.707332
BEL    13.121007
MAL    15.599079
dtype: float64
```

In [202…

```
# Step 9. Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and
# of the windspeeds at each location over all the days

loc_stats = pd.DataFrame()
loc_stats['min'] = wind_stats_data_new.min(axis = 0)
loc_stats['max'] = wind_stats_data_new.max(axis = 0)
loc_stats['mean'] = wind_stats_data_new.mean(axis = 0)
loc_stats['std'] = wind_stats_data_new.std(axis = 0)
loc_stats
```

Out[202…

	min	max	mean	std
RPT	0.67	35.80	12.362987	5.618413
VAL	0.21	33.37	10.644314	5.267356
ROS	1.50	33.84	11.660526	5.008450
KIL	0.00	28.46	6.306468	3.605811
SHA	0.13	37.54	10.455834	4.936125
BIR	0.00	26.16	7.092254	3.968683
DUB	0.00	30.37	9.797343	4.977555
CLA	0.00	31.08	8.495053	4.499449
MUL	0.00	25.88	8.493590	4.166872
CLO	0.04	28.21	8.707332	4.503954
BEL	0.13	42.38	13.121007	5.835037
MAL	0.67	42.54	15.599079	6.699794

In [203…

```
# Step 10. Create a DataFrame called day_stats and calculate the min, max and mean windspeed and
# of the windspeeds across all the locations at each day.

day_stats = pd.DataFrame()
day_stats['min'] = wind_stats_data_new.min(axis = 1)
day_stats['max'] = wind_stats_data_new.max(axis = 1)
day_stats['mean'] = wind_stats_data_new.mean(axis = 1)
day_stats['std'] = wind_stats_data_new.std(axis = 1)
day_stats
```

Out[203…

	min	max	mean	std
Date				
1961-01-01	9.29	18.50	13.018182	2.808875
1961-01-02	6.50	17.54	11.336364	3.188994
1961-01-03	6.17	18.50	11.641818	3.681912
1961-01-04	1.79	11.75	6.619167	3.198126
1961-01-05	6.17	13.33	10.630000	2.445356
...	...	...	...	...
1978-12-27	8.08	40.08	16.708333	7.868076
1978-12-28	5.00	41.46	15.150000	9.687857
1978-12-29	8.71	29.58	14.890000	5.756836
1978-12-30	9.13	28.79	15.367500	5.540437
1978-12-31	9.59	27.29	15.402500	5.702483

6574 rows × 4 columns

In [204…

```python
# Step 11. Find the average windspeed in January for each location.
# Treat January 1961 and January 1962 both as January.

wind_stats_data['month'] = pd.DatetimeIndex(wind_stats_data['Date']).month
january_avg = wind_stats_data.where(wind_stats_data['month'] == 1)
january_avg.loc[:,'RPT':'MAL'].mean()
```

Out[204…

```
RPT    14.847325
VAL    12.914560
ROS    13.299624
KIL     7.199498
SHA    11.667734
BIR     8.054839
DUB    11.819355
CLA     9.512047
MUL     9.543208
CLO    10.053566
BEL    14.550520
MAL    18.028763
dtype: float64
```

In [205…

```python
# Step 12. Downsample the record to a yearly frequency for each location.

wind_stats_data_new.asfreq('Y')
```

Out[205…

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
Date												
1961-12-31	9.87	7.83	7.67	3.75	5.66	3.50	10.04	3.08	5.04	3.79	8.04	14.67
1962-12-31	22.67	16.88	28.67	14.12	19.75	17.08	27.79	25.21	19.83	17.79	25.46	37.63
1963-12-31	13.88	14.42	12.12	9.25	14.33	10.67	18.29	11.96	12.04	15.37	16.79	14.09
1964-12-31	16.33	19.25	13.37	10.08	17.04	12.54	19.83	13.79	12.67	15.04	21.37	23.58
1965-12-31	13.62	13.88	12.29	6.08	12.33	7.41	9.59	10.21	7.46	12.17	15.71	16.75
1966-12-31	13.00	11.46	10.13	6.34	11.87	7.50	13.50	8.46	11.00	10.04	17.29	22.46
1967-12-31	16.88	13.75	11.34	9.08	13.54	7.71	11.75	11.83	11.83	11.75	17.25	22.63
1968-12-31	9.13	2.13	7.38	2.50	4.04	0.50	6.83	2.54	3.54	5.50	5.71	12.42
1969-12-31	14.42	13.83	27.71	7.08	12.08	10.00	14.58	11.00	12.54	7.12	11.17	17.41
1970-12-31	8.38	0.37	9.59	2.62	1.75	0.08	4.83	2.13	2.54	1.17	3.67	7.21
1971-12-31	14.88	10.50	26.08	8.46	13.50	10.04	21.04	10.25	13.54	11.34	12.12	27.33
1972-12-31	13.83	14.46	15.87	9.75	8.71	11.00	10.67	11.54	11.50	10.75	18.00	17.50
1973-12-31	10.67	10.04	6.87	1.46	6.96	5.75	3.83	6.21	4.75	6.13	12.79	15.79
1974-12-31	16.04	16.29	15.21	8.42	13.67	9.75	15.25	16.13	15.04	13.46	18.54	18.46
1975-12-31	15.59	12.33	13.42	2.37	4.08	1.17	7.08	4.25	5.91	6.34	11.38	19.55
1976-12-31	8.67	8.83	9.38	3.67	5.37	4.58	7.92	1.79	4.46	4.38	6.38	15.67
1977-12-31	15.09	7.62	8.79	7.08	10.63	7.58	15.59	11.54	12.25	9.08	14.12	19.55
1978-12-31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

In [206…

```python
# Step 13. Downsample the record to a monthly frequency for each location.

wind_stats_data_new.asfreq('M')
```

Out[206…

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
Date												
1961-01-31	24.21	19.55	16.71	11.96	14.42	10.46	14.88	8.21	10.50	9.96	12.42	13.92
1961-02-28	12.92	12.75	NaN	8.92	16.13	12.29	14.75	14.46	13.96	14.04	18.41	13.17
1961-03-31	8.96	8.04	9.13	8.50	10.75	9.54	11.92	9.59	11.25	8.54	11.96	12.21
1961-04-30	11.67	11.00	9.54	5.54	9.42	5.79	5.09	8.25	6.96	6.25	12.21	8.75
1961-05-31	7.00	9.79	12.25	4.83	8.25	5.37	6.58	9.29	6.58	7.12	11.87	10.63

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
Date												
...	...	...	...	...	...	...	...	...	...	...	...	...
1978-08-31	11.54	5.54	7.41	4.67	7.62	6.17	8.87	5.25	7.83	6.17	11.58	16.88
1978-09-30	26.75	15.63	16.54	13.37	17.58	13.13	16.92	13.79	13.46	13.79	18.91	31.88
1978-10-31	8.58	4.29	10.79	4.29	4.08	2.71	4.63	1.04	3.67	2.75	8.71	10.67
1978-11-30	15.34	4.54	14.75	3.50	4.54	4.96	7.50	2.42	4.96	3.75	4.92	11.50
1978-12-31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

In [207…

```
# Step 14. Downsample the record to a weekly frequency for each location.

wind_stats_data_new.asfreq('W')
```

Out[207…

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
Date												
1961-01-01	15.04	14.96	13.17	9.29	NaN	9.87	13.67	10.25	10.83	12.58	18.50	15.04
1961-01-08	10.96	9.75	7.62	5.91	9.62	7.29	14.29	7.62	9.25	10.46	16.62	16.46
1961-01-15	12.04	9.67	11.75	2.37	7.38	3.13	2.50	6.83	4.75	5.63	7.54	6.75
1961-01-22	9.59	5.88	9.92	2.17	6.87	5.50	9.38	7.04	6.34	7.50	10.88	9.92
1961-01-29	NaN	23.91	22.29	17.54	24.08	19.70	22.00	20.25	21.46	19.95	27.71	23.38
...	...	...	...	...	...	...	...	...	...	...	...	...
1978-12-03	21.21	21.34	17.75	11.58	16.75	14.46	17.46	15.29	15.79	17.50	21.42	25.75
1978-12-10	24.92	22.54	16.54	14.62	15.59	13.00	13.21	14.12	16.21	16.17	26.08	21.92
1978-12-17	9.87	3.21	8.04	2.21	3.04	0.54	2.46	1.46	1.29	2.67	5.00	9.08
1978-12-24	8.67	5.63	12.12	4.79	5.09	5.91	12.25	9.25	10.83	11.71	11.92	31.71
1978-12-31	20.33	17.41	27.29	9.59	12.08	10.13	19.25	11.63	11.58	11.38	12.08	22.08

940 rows × 12 columns

In [208…

```
# Step 15. Calculate the min, max and mean windspeeds and standard deviations of the windspeeds a
# for each week (assume that the first week starts on January 2 1961) for the first 52 weeks.

df = wind_stats_data_new[wind_stats_data_new.index < pd.to_datetime('1962-01-01')]
df.asfreq('W').mean()
df.asfreq('W').min()
df.asfreq('W').max()
df.asfreq('W').std()

day_stats
```

Out[208…

	min	max	mean	std
Date				
1961-01-01	9.29	18.50	13.018182	2.808875
1961-01-02	6.50	17.54	11.336364	3.188994
1961-01-03	6.17	18.50	11.641818	3.681912
1961-01-04	1.79	11.75	6.619167	3.198126
1961-01-05	6.17	13.33	10.630000	2.445356
...	...	...	...	...
1978-12-27	8.08	40.08	16.708333	7.868076
1978-12-28	5.00	41.46	15.150000	9.687857
1978-12-29	8.71	29.58	14.890000	5.756836
1978-12-30	9.13	28.79	15.367500	5.540437

	min	max	mean	std
Date				

Question 5 Food - Python with Pandas

In [225…

```
# Step 1. Import the necessary libraries
import pandas as pd
import numpy as np

# Step 2. Import the dataset from this address.
# Step 3. Assign it to a variable called chipo.
chipo = pd.read_csv(r'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv

# Step 4. See the first 10 entries
chipo.head(10)
```

Out[225…

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

In [226…

```
# Step 5. What is the number of observations in the dataset?

len(chipo)
```

Out[226…

```
4622
```

In [227…

```
# Step 6. What is the number of columns in the dataset?

len(chipo.columns)
```

Out[227…

```
5
```

In [228…

```
# Step 7. Print the name of all the columns.

chipo.columns
```

Out[228…

```
Index(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'],
      dtype='object')
```

In [229…

```
# Step 8. How is the dataset indexed?

chipo.index
```

Out[229…

```
RangeIndex(start=0, stop=4622, step=1)
```

In [230…

```
# Step 9. Which was the most-ordered item?

most_ordered_item = chipo['item_name'].value_counts()
most_ordered_item

# as we can see, "Chicken Bowl" is the most ordered item with 726 orders.
```

Out[230…

```
Chicken Bowl                             726
Chicken Burrito                          553
Chips and Guacamole                      479
Steak Burrito                            368
Canned Soft Drink                        301
Chips                                    211
Steak Bowl                               211
Bottled Water                            162
Chicken Soft Tacos                       115
Chips and Fresh Tomato Salsa             110
Chicken Salad Bowl                       110
Canned Soda                              104
Side of Chips                            101
Veggie Burrito                            95
Barbacoa Burrito                          91
Veggie Bowl                               85
Carnitas Bowl                             68
Barbacoa Bowl                             66
Carnitas Burrito                          59
Steak Soft Tacos                          55
6 Pack Soft Drink                         54
Chips and Tomatillo Red Chili Salsa       48
Chicken Crispy Tacos                      47
Chips and Tomatillo Green Chili Salsa     43
Carnitas Soft Tacos                       40
Steak Crispy Tacos                        35
Chips and Tomatillo-Green Chili Salsa     31
Steak Salad Bowl                          29
Nantucket Nectar                          27
Barbacoa Soft Tacos                       25
Chips and Roasted Chili Corn Salsa        22
Izze                                      20
Chips and Tomatillo-Red Chili Salsa       20
Chips and Roasted Chili-Corn Salsa        18
Veggie Salad Bowl                         18
Barbacoa Crispy Tacos                     11
Barbacoa Salad Bowl                       10
Chicken Salad                              9
Veggie Soft Tacos                          7
Carnitas Crispy Tacos                      7
Carnitas Salad Bowl                        6
Veggie Salad                               6
Burrito                                    6
Steak Salad                                4
Salad                                      2
Bowl                                       2
Crispy Tacos                               2
Veggie Crispy Tacos                        1
Chips and Mild Fresh Tomato Salsa          1
Carnitas Salad                             1
Name: item_name, dtype: int64
```

In [231…

```
# Step 10. For the most-ordered item, how many items were ordered?

most_ordered_item[:1]
```

Out[231…

```
Chicken Bowl    726
Name: item_name, dtype: int64
```

In [232…

```
# Step 11. What was the most ordered item in the choice_description column?

chipo.choice_description.value_counts()

# as we can see, Diet Code is the most ordered item with 134 orders.
```

Out[232…

```
[Diet Coke]
134
[Coke]
123
[Sprite]
77
[Fresh Tomato Salsa, [Rice, Black Beans, Cheese, Sour Cream, Lettuce]]
42
[Fresh Tomato Salsa, [Rice, Black Beans, Cheese, Sour Cream, Guacamole, Lettuce]]
```

```
40

...
[Fresh Tomato (Mild), [Rice, Sour Cream, Cheese]]
1
[Tomatillo Red Chili Salsa, [Rice, Cheese, Guacamole]]
1
[Tomatillo Red Chili Salsa, [Rice, Fajita Vegetables, Black Beans, Cheese, Lettuce]]
1
[Fresh Tomato Salsa (Mild), [Fajita Veggies, Cheese, Sour Cream, Guacamole, Lettuce]]
1
[[Roasted Chili Corn Salsa (Medium), Tomatillo-Red Chili Salsa (Hot)], [Black Beans, Rice, Fajita
Veggies, Cheese, Sour Cream, Guacamole, Lettuce]]       1
Name: choice_description, Length: 1043, dtype: int64
```

In [233…

```
# Step 12. How many items were orderd in total?

chipo['quantity'].sum()
```

Out[233…

```
4972
```

In [234…

```
# Step 13.
# • Turn the item price into a float
# • Check the item price type
# • Create a lambda function and change the type of item price
# • Check the item price type

# turn the item into a float
chipo['item_price'] = chipo['item_price'].apply(lambda x: float(x[1:]))
chipo['item_price'].dtypes
```

Out[234…

```
dtype('float64')
```

In [235…

```
# Step 14. How much was the revenue for the period in the dataset?

chipo['revenue'] = chipo['quantity'] * chipo['item_price']
chipo['revenue'].sum()
```

Out[235…

```
39237.02
```

In [236…

```
# Step 15. How many orders were made in the period?

orders = chipo['order_id'].nunique()
orders
```

Out[236…

```
1834
```

In [237…

```
# Step 16. What is the average revenue amount per order?

chipo.groupby('order_id')['revenue'].mean()
```

Out[237…

```
order_id
1        2.890000
2       33.960000
3        6.335000
4       10.500000
5        6.850000
          ...
1830    11.500000
1831     4.300000
1832     6.600000
1833    11.750000
1834     9.583333
Name: revenue, Length: 1834, dtype: float64
```

In [238…

```
# Step 17. How many different items are sold?

chipo['item_name'].nunique()
```

Out[238…

```
50
```

Question 6

Create a line plot showing the number of marriages and divorces per capita in the U.S. between 1867 and 2014. Label both lines and show the legend.

Don't forget to label your axes!

In [1]:

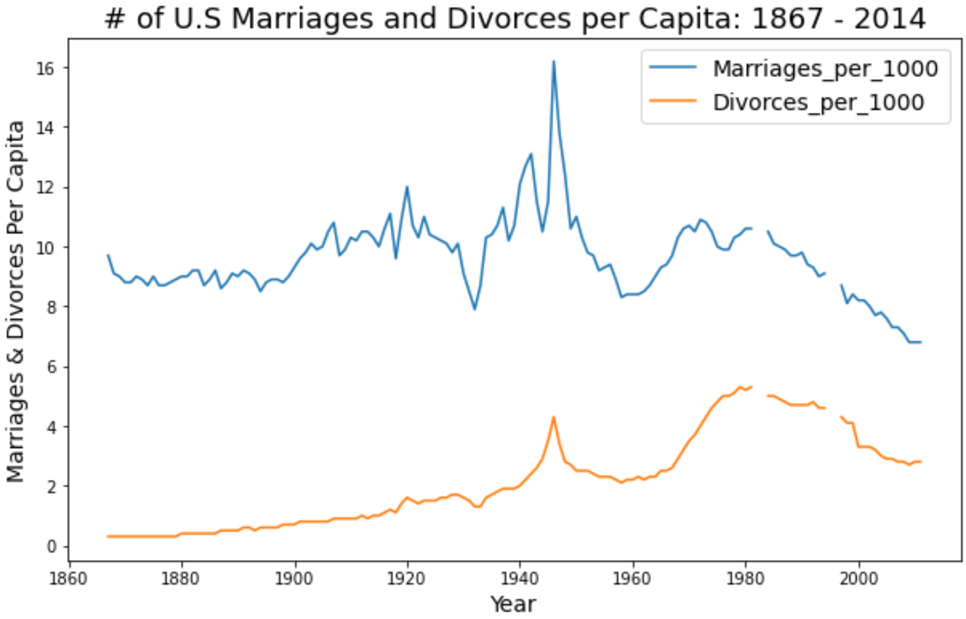
```
# import pandas to first inspect the data
import pandas as pd

marriages_divorces = pd.read_csv('us-marriages-divorces-1867-2014.csv')
marriages_divorces.head()
```

Out[1]:

	Year	Marriages	Divorces	Population	Marriages_per_1000	Divorces_per_1000
0	1867	357000.0	10000.0	36970000	9.7	0.3
1	1868	345000.0	10000.0	37885000	9.1	0.3
2	1869	348000.0	11000.0	38870000	9.0	0.3
3	1870	352000.0	11000.0	39905000	8.8	0.3
4	1871	359000.0	12000.0	41010000	8.8	0.3

In [2]:

```
# import other libraries
import matplotlib.pyplot as plt
%matplotlib inline

marriages_divorces.plot.line(x = 'Year', y = ['Marriages_per_1000', 'Divorces_per_1000'], figsize

# label the plot and axes

# x-axis
plt.xlabel('Year', fontsize = 14)

# y-axis
plt.ylabel('Marriages & Divorces Per Capita', fontsize = 14)

# add the title
plt.title('# of U.S Marriages and Divorces per Capita: 1867 - 2014', fontsize = 18)

# show the legend
plt.legend(prop = dict(size = 14))

plt.show()
```



Question 7

Create a vertical bar chart comparing the number of marriages and divorces per capita in the U.S. between 1900, 1950, and 2000.

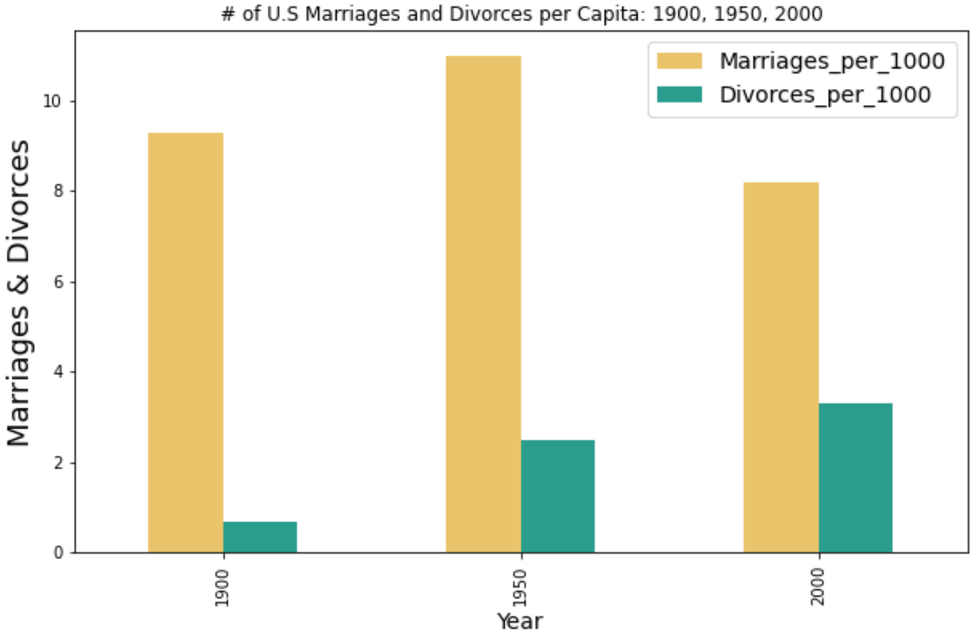
Don't forget to label your axes!

In [3]:

```
# only the years we need
marriages_divorces = marriages_divorces[(marriages_divorces.Year == 1900) | (marriages_divorces.Y
```

In [4]:

```
# import other libraries
import matplotlib.pyplot as plt
%matplotlib inline

marriages_divorces = marriages_divorces.drop(columns = ['Marriages', 'Divorces', 'Population'])
marriages_divorces
marriages_divorces = marriages_divorces.set_index('Year')

marriages_divorces.plot.bar(figsize = (10,6), color = {"#2a9d8f", "#e9c46a"})

# label the chart and axes

# x-axis
plt.xlabel('Year', fontsize = 14)

# y-axis
plt.ylabel('Marriages & Divorces', fontsize = 18)

# add the title
plt.title('# of U.S Marriages and Divorces per Capita: 1900, 1950, 2000')

# show the legend
plt.legend(prop = dict(size = 14))

plt.show()
```



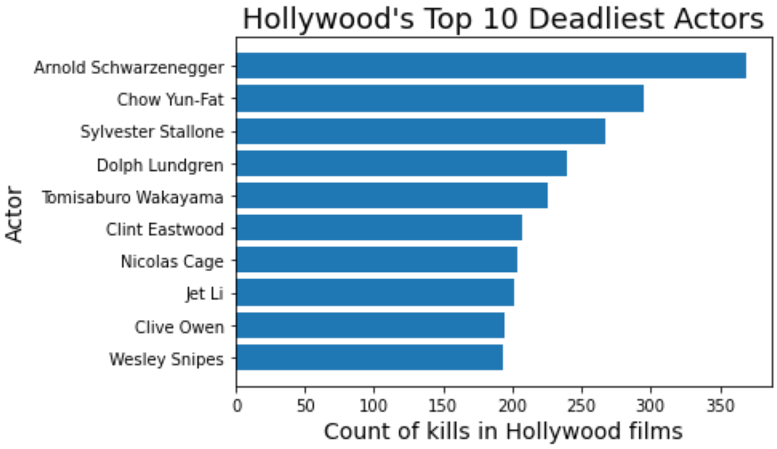
Question 8

Create a horizontal bar chart that compares the deadliest actors in Hollywood. Sort the actors by their kill count and label each bar with the corresponding actor's name.

Don't forget to label your axes!

In [5]:

```
# import to create the visual
import matplotlib.pyplot as plt
%matplotlib inline

# import pandas to use the Data Frame
import pandas as pd

# import the data into the Data Frame
hollywood_kills = pd.read_csv('actor_kill_counts.csv')
hollywood_kills = hollywood_kills.sort_values('Count')

# add the title
plt.title("Hollywood's Top 10 Deadliest Actors", fontsize = 18)

# label the chart and axes

# x-axis
plt.ylabel('Actor', fontsize = 14)

# y-axis
plt.xlabel('Count of kills in Hollywood films', fontsize = 14)

plt.barh(hollywood_kills['Actor'], hollywood_kills.Count)

plt.show()
```



Question 9

Create a pie chart showing the fraction of all Roman Emperors that were assassinated.

Make sure that the pie chart is an even circle, labels the categories, and shows the percentage breakdown of the categories.

In [6]:

```
# import pandas to use the Data Frame
import pandas as pd

# add data to the Data Frame
roman_emp = pd.read_csv('roman-emperor-reigns.csv')
roman_emp.head()
```

Out[6]:

	Emperor	Length_of_Reign	Cause_of_Death
0	Augustus	40.58	Possibly assassinated
1	Tiberius	22.50	Possibly assassinated
2	Caligula	4.83	Assassinated
3	Claudius	13.75	Possibly assassinated
4	Nero	13.67	Suicide

In [7]:

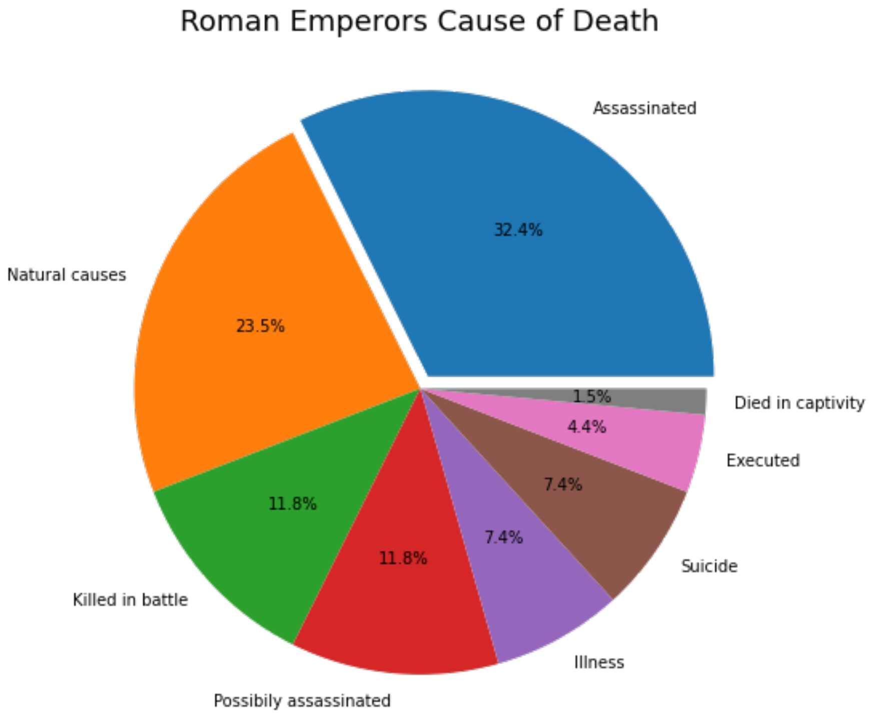
```
# group by cause of death and sum totals of each death type
roman_emp_death = roman_emp.groupby('Cause_of_Death').count().drop(columns = 'Length_of_Reign')

# sort them in the decreasing order
roman_emp_death.sort_values(by = ['Emperor'], ascending = False)
```

Out[7]:

Cause_of_Death	Emperor
Assassinated	22
Natural causes	16
Killed in battle	8
Possibly assassinated	8
Illness	5
Suicide	5
Executed	3
Died in captivity	1

In [8]:

```
# import other libraries
import matplotlib.pyplot as plt
import numpy as np

# specify the data
y = np.array([22, 16, 8, 8, 5, 5, 3, 1])
label = ["Assassinated", 'Natural causes', 'Killed in battle', 'Possibily assassinated',
'Illness', 'Suicide','Executed', 'Died in captivity']

# explode on "Assassinated"
explode = (0.05, 0, 0, 0, 0, 0, 0, 0)

# create the plot
fig1, ax1 = plt.subplots(figsize = (8,8))

# set the labels, axes, and title
ax1.axis('equal')
ax1.pie(y, labels = label, autopct = '%1.1f%%', explode = explode)
ax1.set_title('Roman Emperors Cause of Death', fontsize = 18)

plt.show()
```



Question 10

Create a scatter plot showing the relationship between the total revenue earned by arcades and the number of Computer Science PhDs awarded in the U.S. between 2000 and 2009.

Don't forget to label your axes!

Color each dot according to its year.

In [9]:

```python
# import pandas to use the Data Frame
import pandas as pd

# import the csv file and add it to the Data Frame
arcade_science_phd = pd.read_csv('arcade-revenue-vs-cs-doctorates.csv')
arcade_science_phd
```

Out[9]:

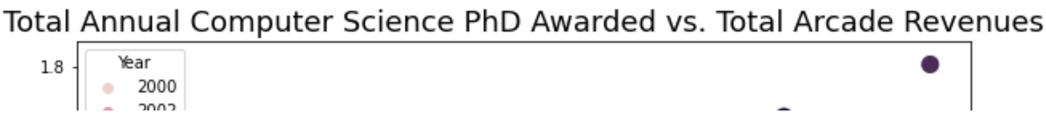
	Year	Total Arcade Revenue (billions)	Computer Science Doctorates Awarded (US)
0	2000	1.196	861
1	2001	1.176	830
2	2002	1.269	809
3	2003	1.240	867
4	2004	1.307	948
5	2005	1.435	1129
6	2006	1.601	1453
7	2007	1.654	1656
8	2008	1.803	1787
9	2009	1.734	1611

In [10]:

```python
# import other libraries
import seaborn as sns
import matplotlib.pyplot as plt

# increase the size of the figure
plt.figure(figsize = (10,6))

# create the scatter plot, set the title, and increase the size of the data points
sns.scatterplot(x = 'Computer Science Doctorates Awarded (US)',
                y = 'Total Arcade Revenue (billions)',
                data = arcade_science_phd, hue = 'Year', s = 150,).set_title('Total Annual Comput

# label the axes
plt.xlabel('Year', fontsize = 14)
plt.ylabel('Total Arcade Revenue ($ billions)', fontsize = 14)
```

Out[10]:

```
Text(0, 0.5, 'Total Arcade Revenue ($ billions)')
```


Total Annual Computer Science PhD Awarded vs. Total Arcade Revenues
1.8
Year
2000
2002

In [ ]: