jupyter Problem Set-3_200515736_Meet Patel Last Checkpoint: 5 hours ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Name: Meet Patel

Student id: 200515736

Question 1

Introduction: Special thanks to: https://github.com/justmarkham for sharing the dataset and materials.

Step 1. Import the necessary libraries

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

Step 2. Import the dataset from this address. Step 3. Assign it to a variable called users

In [2]:
```
url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/u.user'
users = pd.read_csv(url,delimiter="|",header=0)
```

Step 4. Discover what is the mean age per occupation

In [3]:
```
users.groupby('occupation')['age'].mean()
```

Out[3]:
```
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
student          22.081633
technician       33.148148
writer           36.311111
Name: age, dtype: float64
```

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

In [4]:
```
male_ratio = users.pivot_table(index='occupation', columns='gender', aggfunc='size', fill_value=0)

sums = male_ratio[['F', 'M']].sum(axis=1)
male_ratio['MaleRatio'] = round(100 * male_ratio['M'] / sums , 1)

male_ratio['MaleRatio'].sort_values(ascending=False)
```

Out[4]:
```
occupation
doctor           100.0
engineer          97.0
technician        96.3
retired           92.9
programmer        90.9
executive         90.6
scientist         90.3
entertainment     88.9
lawyer            83.3
salesman          75.0
educator          72.6
student           69.4
other             65.7
marketing         61.5
writer            57.8
none              55.6
administrator     54.4
artist            53.6
librarian         43.1
healthcare        31.2
homemaker         14.3
Name: MaleRatio, dtype: float64
```

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

In [5]:
```
result = users.groupby('occupation').agg({'age': ['min', 'max']})
result
```

Out[5]:

	age	
	min	max
occupation		
administrator	21	70
artist	19	48
doctor	28	64
educator	23	63
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
student	7	42
technician	21	55
writer	18	60

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

In [6]:
```
users.groupby(['occupation','gender'])['age'].max().reset_index()
```

Out[6]:

	occupation	gender	age
0	administrator	F	62
1	administrator	M	70
2	artist	F	48
3	artist	M	45
4	doctor	M	64
5	educator	F	51
6	educator	M	63
7	engineer	F	36
8	engineer	M	70
9	entertainment	F	38
10	entertainment	M	50
11	executive	F	49
12	executive	M	69
13	healthcare	F	53
14	healthcare	M	62
15	homemaker	F	50
16	homemaker	M	23
17	lawyer	F	51
18	lawyer	M	53
19	librarian	F	59
20	librarian	M	69
21	marketing	F	50
22	marketing	M	55
23	none	F	55
24	none	M	33
25	other	F	55
26	other	M	64
27	programmer	F	38
28	programmer	M	63
29	retired	F	70
30	retired	M	73
31	salesman	F	33
32	salesman	M	66
33	scientist	F	31
34	scientist	M	55
35	student	F	38
36	student	M	42
37	technician	F	38
38	technician	M	55
39	writer	F	56
40	writer	M	60

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

In [7]:
```
male_ratio['FemaleRatio'] = round(100 * male_ratio['F'] / sums , 1)

male_ratio
```

Out[7]:

gender	F	M	MaleRatio	FemaleRatio
occupation				
administrator	36	43	54.4	45.6
artist	13	15	53.6	46.4
doctor	0	7	100.0	0.0
educator	26	69	72.6	27.4
engineer	2	65	97.0	3.0
entertainment	2	16	88.9	11.1
executive	3	29	90.6	9.4
healthcare	11	5	31.2	68.8
homemaker	6	1	14.3	85.7
lawyer	2	10	83.3	16.7
librarian	29	22	43.1	56.9
marketing	10	16	61.5	38.5
none	4	5	55.6	44.4
other	36	69	65.7	34.3
programmer	6	60	90.9	9.1
retired	1	13	92.9	7.1
salesman	3	9	75.0	25.0
scientist	3	28	90.3	9.7
student	60	136	69.4	30.6
technician	1	26	96.3	3.7
writer	19	26	57.8	42.2

Question 2

Euro Teams

Step 1. Import the necessary libraries

In [8]:
```
import pandas as pd
```

Step 2. Import the dataset from this address
Step 3. Assign it to a variable called euro12

In [9]:
```
url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/02_Filtering_%26_Sorting/Euro12/Euro_2012_stats_TEAM.
euro12 = pd.read_csv(url,delimiter=",",header=0)
```

Step 4. Select only the Goal column

In [10]:
```
euro12["Goals"]
```

Out[10]:
```
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
```

Step 5. How many team participated in the Euro2012?

In [11]:
```
euro12
```

Out[11]:

	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	Fouls Conceded	Offsides	Yellow Cards	Red Cards
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	13	81.3%	41	62	2	9	0
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	9	60.1%	53	73	8	7	0
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	10	66.7%	25	38	8	4	0
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	22	88.1%	43	45	6	5	0
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	6	54.6%	36	51	5	6	0
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	13	65.1%	67	48	12	9	1
7	Italy	6	34	45	43.0%	7.5%	110	2	0	0	...	20	74.1%	101	89	16	16	0
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	0	...	12	70.6%	35	30	3	5	0
9	Poland	2	15	23	39.4%	5.2%	48	0	0	0	...	6	66.7%	48	56	3	7	1
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	0	...	10	71.5%	73	90	10	12	0
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	0	...	17	65.4%	43	51	11	6	1
12	Russia	5	9	31	22.5%	12.5%	59	2	0	0	...	10	77.0%	34	43	4	6	0
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	15	93.8%	102	83	19	11	0
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	0	...	8	61.6%	35	51	7	7	0
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	0	...	13	76.5%	48	31	4	5	0

16 rows × 35 columns

In [12]:
```
print("Team participated in the Euro2012:" + str(euro12["Team"].nunique()))
```

```
Team participated in the Euro2012:16
```

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

In [13]:
```
print("Number of Columns: "+ str(len(euro12.axes[1])))
```

```
Number of Columns: 35
```

Step 7. View only the columns Team, Yellow Cards and Red Cards and assign them to a dataframe called discipline

In [14]:
```
discipline = euro12[["Team", "Yellow Cards", "Red Cards"]]
discipline
```

Out[14]:

	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

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

In [15]:
```
discipline.sort_values(by=['Red Cards', 'Yellow Cards'], inplace = True)
```

```
C:\Users\Meet Patel\AppData\Local\Temp\ipykernel_35188\3564691409.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  discipline.sort_values(by=['Red Cards', 'Yellow Cards'], inplace = True)
```

In [16]:
```
discipline
```

Out[16]:

	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

Step 9. Calculate the mean Yellow Cards given per Team

In [17]:
```
discipline['Yellow Cards'].mean()
```

Out[17]: 7.4375

Step 10. Filter teams that scored more than 6 goalsStep

In [18]:
```
euro12[euro12['Goals'] > 6]
```

Out[18]:

	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	Fouls Conceded	Offsides	Yellow Cards	Red Cards
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	15	93.8%	102	83	19	11	0

2 rows × 35 columns

Step 11. Select the teams that start with G

In [19]:
```
euro12[euro12.Team.str.startswith('G')]
```

Out[19]:

	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	Fouls Conceded	Offsides	Yellow Cards	Red Cards	S
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	10	62.6%	63	49	12	4	0	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	13	65.1%	67	48	12	9	1	

2 rows × 35 columns

Step 12. Select the first 7 columns

In [20]:
```
euro12.iloc[: , 0:7]
```

Out[20]:

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

Step 13. Select all columns except the last 3

In [21]:
```
euro12.iloc[:, :-3]
```

Out[21]:

	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	Goals conceded	Saves made	Saves-to-shots ratio	Fouls Won	Fo Concec
0	Croatia	4	13	12	51.9%	16.0%	32	0	0	0	...	0	10	3	13	81.3%	41	
1	Czech Republic	4	13	18	41.9%	12.9%	39	0	0	0	...	1	10	6	9	60.1%	53	
2	Denmark	4	10	10	50.0%	20.0%	27	1	0	0	...	1	10	5	10	66.7%	25	
3	England	5	11	18	50.0%	17.2%	40	0	0	0	...	2	29	3	22	88.1%	43	
4	France	3	22	24	37.9%	6.5%	65	1	0	0	...	1	7	5	6	54.6%	36	
5	Germany	10	32	32	47.8%	15.6%	80	2	1	0	...	1	11	6	10	62.6%	63	
6	Greece	5	8	18	30.7%	19.2%	32	1	1	1	...	1	23	7	13	65.1%	67	
7	Italy	6	34	45	43.0%	7.5%	110	2	0	0	...	2	18	7	20	74.1%	101	
8	Netherlands	2	12	36	25.0%	4.1%	60	2	0	0	...	0	9	5	12	70.6%	35	
9	Poland	2	15	23	39.4%	5.2%	48	0	0	0	...	0	8	3	6	66.7%	48	
10	Portugal	6	22	42	34.3%	9.3%	82	6	0	0	...	2	11	4	10	71.5%	73	
11	Republic of Ireland	1	7	12	36.8%	5.2%	28	0	0	0	...	0	23	9	17	65.4%	43	
12	Russia	5	9	31	22.5%	12.5%	59	2	0	0	...	0	8	3	10	77.0%	34	
13	Spain	12	42	33	55.9%	16.0%	100	0	1	0	...	5	8	1	15	93.8%	102	
14	Sweden	5	17	19	47.2%	13.8%	39	3	0	0	...	1	12	5	8	61.6%	35	
15	Ukraine	2	7	26	21.2%	6.0%	38	0	0	0	...	0	4	4	13	76.5%	48	

16 rows × 32 columns

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

In [22]:
```
euro12.loc[euro12.Team.isin(['England', 'Italy', 'Russia']), ['Team','Shooting Accuracy']]
```

Out[22]:

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

Question 3

Housing Step 1. Import the necessary libraries Step 2. Create 3 differents Series, each of length 100, as follows: • The first a random number from 1 to 4 • The second a random number from 1 to 3 • The third a random number from 10,000 to 30,000 Step 3. Create a DataFrame by joinning the Series by column Step 4. Change the name of the columns to bedrs, bathrs, price_sqr_meter Step 5. Create a one column DataFrame with the values of the 3 Series and assign it to 'bigcolumn' Step 6. Ops it seems it is going only until index 99. Is it true? Step 7. Reindex the DataFrame so it goes from 0 to 299

In [23]:
```
import pandas as pd
import numpy as np
```

The first a random number from 1 to 4

In [24]:
```
import random
d_first = [[random.randint(1, 4)] for _ in range(100)]
df_1 = pd.DataFrame(d_first)
```

The second a random number from 1 to 3

In [25]:
```
import random
d_second = [[random.randint(1, 3)] for _ in range(100)]
df_2 = pd.DataFrame(d_second)
```

• The third a random number from 10,000 to 30,000

In [26]:
```
import random
d_third = [[random.randint(10000, 30000)] for _ in range(100)]
df_3 = pd.DataFrame(d_third)
```

Create a DataFrame by joinning the Series by column

In [27]:
```
df = pd.concat([df_1, df_2 , df_3], axis=1)
```

Change the name of the columns to bedrs, bathrs, price_sqr_meter

In [28]:
```
df.columns = ['bedrs', 'bathrs', 'price_sqr_meter']
```

In [29]:
```
df_new = df.bedrs.astype(str).str.cat(df.bathrs.astype(str)).str.cat(df.price_sqr_meter.astype(str))
df_new.columns = ['bigcolumn']
```

Create a one column DataFrame with the values of the 3 Series and assign it to 'bigcolumn'

In [30]:
```
print(df_new)
```

```
0     4322566
1     2124160
2     1117961
3     4220041
4     4122724
       ...
95    3229597
96    3111619
97    2117439
98    4315724
99    2113558
Name: bedrs, Length: 100, dtype: object
```

In [31]:
```
df_new = df_new.reset_index(drop=True)
```

In [32]:
```
df_new.reset_index()
df_new.reindex(index=range(0,299))
```

Out[32]:
```
0      4322566
1      2124160
2      1117961
3      4220041
4      4122724
        ...
294        NaN
295        NaN
296        NaN
297        NaN
298        NaN
Name: bedrs, Length: 299, dtype: object
```

Question 4

Wind Statistics The data have been modified to contain some missing values, identified by NaN. Using pandas should make this exercise easier, in particular for the bonus question. You should be able to perform all of these operations without using a for loop or other looping construct. The data in 'wind.data' has the following format: Yr Mo Dy RPT VAL ROS KIL SHA BIR DUB CLA MUL CLO BEL MAL 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 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 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

The first three columns are year, month, and day. The remaining 12 columns are average windspeeds in knots at 12 locations in Ireland on that day.

Step 1. Import the necessary libraries

In [33]:
```
import pandas as pd
```

```
import datetime
```

Step 2. Import the dataset from the attached file wind.txt

In [ ]:

```

```

Step 3. Assign it to a variable called data and replace the first 3 columns by a proper datetime index.

In [34]:

```
data_url = 'https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/06_Stats/Wind_Stats/wind.data'
data = pd.read_csv(data_url, sep = "\s+", parse_dates = [[0,1,2]])
data.head()
```

Out[34]:

	Yr_Mo_Dy	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

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

In [35]:

```
def fix_century(x):
  year = x.year - 100 if x.year > 1989 else x.year
  return datetime.date(year, x.month, x.day)

data['Yr_Mo_Dy'] = data['Yr_Mo_Dy'].apply(fix_century)

data.head()
```

Out[35]:

	Yr_Mo_Dy	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

Step 5. Set the right dates as the index. Pay attention at the data type, it should be datetime64[ns].

In [36]:

```
data["Yr_Mo_Dy"] = pd.to_datetime(data["Yr_Mo_Dy"])

data = data.set_index('Yr_Mo_Dy')

data.head()
```

Out[36]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
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
1961-01-02	14.71	NaN	10.83	6.50	12.62	7.67	11.50	10.04	9.79	9.67	17.54	13.83
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
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
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

Step 6. Compute how many values are missing for each location over the entire record.They should be ignored in all calculations below.

In [37]:

```
data.isnull().sum()
```

Out[37]:

```
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
```

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

In [38]:

```
data.shape[0] - data.isnull().sum()

#or

data.notnull().sum()
```

Out[38]:

```
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
```

Step 8. Calculate the mean windspeeds of the windspeeds over all the locations and all the times.

In [39]:

```
data.sum().sum() / data.notna().sum().sum()
```

Out[39]:

```
10.227883764282181
```

Step 9. Create a DataFrame called loc_stats and calculate the min, max and mean windspeeds and standard deviations of the windspeeds at each location over all the days A different set of numbers for each location.

In [40]:

```
data.describe(percentiles=[])
```

Out[40]:

	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	
count	6568.000000	6571.000000	6572.000000	6569.000000	6572.000000	6574.000000	6571.000000	6572.000000	6571.000000	6573.000000	6574.000000	6570.0
mean	12.362987	10.644314	11.660526	6.306468	10.455834	7.092254	9.797343	8.495053	8.493590	8.707332	13.121007	15.5
std	5.618413	5.267356	5.008450	3.605811	4.936125	3.968683	4.977555	4.499449	4.166872	4.503954	5.835037	6.6
min	0.670000	0.210000	1.500000	0.000000	0.130000	0.000000	0.000000	0.000000	0.000000	0.040000	0.130000	0.6
50%	11.710000	10.170000	10.920000	5.750000	9.960000	6.830000	9.210000	8.080000	8.170000	8.290000	12.500000	15.0

max	35.800000	33.370000	33.840000	28.460000	37.540000	26.160000	30.370000	31.080000	25.880000	28.210000	42.380000	42.5

Step 10. Create a DataFrame called day_stats and calculate the min, max and mean windspeed and standard deviations of the windspeeds across all the locations at each day. A different set of numbers for each day.

In [41]:
```
day_stats = pd.DataFrame()

# this time we determine axis equals to one so it gets each row.
day_stats['min'] = data.min(axis = 1) # min
day_stats['max'] = data.max(axis = 1) # max
day_stats['mean'] = data.mean(axis = 1) # mean
day_stats['std'] = data.std(axis = 1) # standard deviations

day_stats.head()
```

Out[41]:

Yr_Mo_Dy	min	max	mean	std
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

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

In [42]:
```
data.loc[data.index.month == 1].mean()
```

Out[42]:
```
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
```

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

In [43]:
```
data.groupby(data.index.to_period('A')).mean()
```

Out[43]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
1961	12.299583	10.351796	11.362369	6.958227	10.881763	7.729726	9.733923	8.858788	8.647652	9.835577	13.502795	13.680773
1962	12.246923	10.110438	11.732712	6.960440	10.657918	7.393068	11.020712	8.793753	8.316822	9.676247	12.930685	14.323956
1963	12.813452	10.836986	12.541151	7.330055	11.724110	8.434712	11.075699	10.336548	8.903589	10.224438	13.638877	14.999014
1964	12.363661	10.920164	12.104372	6.787787	11.454481	7.570874	10.259153	9.467350	7.789016	10.207951	13.740546	14.910301
1965	12.451370	11.075534	11.848767	6.858466	11.024795	7.478110	10.618712	8.879918	7.907425	9.918082	12.964247	15.591644
1966	13.461973	11.557205	12.020630	7.345726	11.805041	7.793671	10.579808	8.835096	8.514438	9.768959	14.265836	16.307260
1967	12.737151	10.990986	11.739397	7.143425	11.630740	7.368164	10.652027	9.325616	8.645014	9.547425	14.774548	17.135945
1968	11.835628	10.468197	11.409754	6.477678	10.760765	6.067322	8.859180	8.255519	7.224945	7.832978	12.808634	15.017486
1969	11.166356	9.723699	10.902000	5.767973	9.873918	6.189973	8.564493	7.711397	7.924521	7.754384	12.621233	15.762904
1970	12.600329	10.726932	11.730247	6.217178	10.567370	7.609452	9.609890	8.334630	9.297616	8.289808	13.183644	16.456027
1971	11.273123	9.095178	11.088329	5.241507	9.440329	6.097151	8.385890	6.757315	7.915370	7.229753	12.208932	15.025233
1972	12.463962	10.561311	12.058333	5.929699	9.430410	6.358825	9.704508	7.680792	8.357295	7.515273	12.727377	15.028716
1973	11.828466	10.680493	10.680493	5.547863	9.640877	6.548740	8.482110	7.614274	8.245534	7.812411	12.169699	15.441096
1974	13.643096	11.811781	12.336356	6.427041	11.110986	6.809781	10.084603	9.896986	9.331753	8.736356	13.252959	16.947671
1975	12.008575	10.293836	11.564712	5.269096	9.190082	5.668521	8.562603	7.843836	8.797945	7.382822	12.631671	15.307863
1976	11.737842	10.203115	10.761230	5.109426	8.846339	6.311038	9.149126	7.146202	8.883716	7.883087	12.332377	15.471448
1977	13.099616	11.144493	12.627836	6.073945	10.003836	8.586438	11.523205	8.378384	9.098192	8.821616	13.459068	16.590849
1978	12.504356	11.044274	11.380000	6.082356	10.167233	7.650658	9.489342	8.800466	9.089753	8.301699	12.967397	16.771370

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

In [44]:
```
data.groupby(data.index.to_period('M')).mean()
```

Out[44]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
1961-01	14.841333	11.988333	13.431613	7.736774	11.072759	8.588065	11.184839	9.245333	9.085806	10.107419	13.880968	14.703226
1961-02	16.269286	14.975357	14.441481	9.230741	13.852143	10.937500	11.890714	11.846071	11.821429	12.714286	18.583214	15.411786
1961-03	10.890000	11.296452	10.752903	7.284000	10.509355	8.866774	9.644194	9.829677	10.294138	11.251935	16.410968	15.720000
1961-04	10.722667	9.427667	9.998000	5.830667	8.435000	6.495000	6.925333	7.094667	7.342333	7.237000	11.147333	10.278333
1961-05	9.860968	8.850000	10.818065	5.905333	9.490323	6.574839	7.604000	8.177097	8.039355	8.499355	11.900323	12.011613
...	...	...	...	...	...	...	...	...	...	...	...	...
1978-08	9.645161	8.259355	9.032258	4.502903	7.368065	5.935161	5.650323	5.417742	7.241290	5.536774	10.466774	12.054194
1978-09	10.913667	10.895000	10.635000	5.725000	10.372000	9.278333	10.790333	9.583000	10.069333	8.939000	15.680333	19.391333
1978-10	9.897742	8.670968	9.295806	4.721290	8.525161	6.774194	8.115484	7.337742	8.297742	8.243871	13.776774	17.150000
1978-11	16.151667	14.802667	13.508000	7.317333	11.475000	8.743000	11.492333	9.657333	10.701333	10.676000	17.404667	20.723000
1978-12	16.175484	13.748065	15.635161	7.094839	11.398710	9.241613	12.077419	10.194839	10.616774	11.028710	13.859677	21.371613

216 rows × 12 columns

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

In [45]:
```
data.groupby(data.index.to_period('W')).mean()
```

Out[45]:

Yr_Mo_Dy	RPT	VAL	ROS	KIL	SHA	BIR	DUB	CLA	MUL	CLO	BEL	MAL
1960-12-26/1961-01-01	15.040000	14.960000	13.170000	9.290000	NaN	9.870000	13.670000	10.250000	10.830000	12.580000	18.500000	15.040000
1961-01-02/1961-01-08	13.541429	11.486667	10.487143	6.417143	9.474286	6.435714	11.061429	6.616667	8.434286	8.497143	12.481429	13.238571
1961-01-09/1961-01-15	12.468571	8.967143	11.958571	4.630000	7.351429	5.072857	7.535714	6.820000	5.712857	7.571429	11.125714	11.024286
1961-01-16/1961-01-22	13.204286	9.862857	12.982857	6.328571	8.966667	7.417143	9.257143	7.875714	7.145714	8.124286	9.821429	11.434286
1961-01-23/1961-01-29	19.880000	16.141429	18.225714	12.720000	17.432857	14.828571	15.528571	15.160000	14.480000	15.640000	20.930000	22.530000
...	...	...	...	...	...	...	...	...	...	...	...	...
1978-11-27/1978-12-03	14.934286	11.232857	13.941429	5.565714	10.215714	8.618571	9.642857	7.685714	9.011429	9.547143	11.835714	18.728571

1978-12-04/1978-12-10	20.740000	19.190000	17.034286	9.777143	15.287143	12.774286	14.437143	12.468571	13.870000	14.082857	16.517143	23.061429
1978-12-11/1978-12-17	16.758571	14.692857	14.987143	6.917143	11.397143	7.272857	10.208571	7.967143	9.168571	8.565714	11.102857	15.562857
1978-12-18/1978-12-24	11.155714	8.008571	13.172857	4.004286	7.825714	6.290000	7.798571	8.667143	7.151429	8.072857	11.845714	18.977143
1978-12-25/1978-12-31	14.951429	11.801429	16.035714	6.507143	9.660000	8.620000	13.708571	10.477143	10.868571	11.471429	12.947143	26.844286

940 rows × 12 columns

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

In [46]:
```
weekly = data.resample('W').agg(['min','max','mean','std'])

# slice it for the first 52 weeks and locations
weekly.loc[weekly.index[1:53], "RPT":"MAL"] .head(10)
```

Out[46]:

	RPT				VAL				ROS		...	CLO						BEL	
	min	max	mean	std	min	max	mean	std	min	max	...	mean	std	min	max	mean	std	min	
Yr_Mo_Dy																			
1961-01-08	10.58	18.50	13.541429	2.631321	6.63	16.88	11.486667	3.949525	7.62	12.33	...	8.497143	1.704941	5.46	17.54	12.481429	4.349139	10.88	
1961-01-15	9.04	19.75	12.468571	3.555392	3.54	12.08	8.967143	3.148945	7.08	19.50	...	7.571429	4.084293	5.25	20.71	11.125714	5.552215	5.17	
1961-01-22	4.92	19.83	13.204286	5.337402	3.42	14.37	9.862857	3.837785	7.29	20.79	...	8.124286	4.783952	6.50	15.92	9.821429	3.626584	6.79	
1961-01-29	13.62	25.04	19.880000	4.619061	9.96	23.91	16.141429	5.170224	12.67	25.84	...	15.640000	3.713368	14.04	27.71	20.930000	5.210726	17.50	
1961-02-05	10.58	24.21	16.827143	5.251408	9.46	24.21	15.460000	5.187395	9.04	19.70	...	9.460000	2.839501	9.17	19.33	14.012857	4.210858	7.17	
1961-02-12	16.00	24.54	19.684286	3.587677	11.54	21.42	16.417143	3.608373	13.67	21.34	...	14.440000	1.746749	15.21	26.38	21.832857	4.063753	17.04	
1961-02-19	6.04	22.50	15.130000	5.064609	11.63	20.17	15.091429	3.575012	6.13	19.41	...	13.542857	2.531361	14.09	29.63	21.167143	5.910938	10.96	
1961-02-26	7.79	25.80	15.221429	7.020716	7.08	21.50	13.625714	5.147348	6.08	22.42	...	12.730000	4.920064	9.59	23.21	16.304286	5.091162	6.67	
1961-03-05	10.96	13.33	12.101429	0.997721	8.83	17.00	12.951429	2.851955	8.17	13.67	...	12.370000	1.593685	11.58	23.45	17.842857	4.332331	8.83	
1961-03-12	4.88	14.79	9.376667	3.732263	8.08	16.96	11.578571	3.230167	7.54	16.38	...	10.458571	3.655113	10.21	22.71	16.701429	4.358759	5.54	

10 rows × 48 columns

Question 5

Step 1. Import the necessary libraries

In [47]:
```
import pandas as pd
```

Step 2. Import the dataset from this address.

Step 3. Assign it to a variable called chipo.

In [48]:
```
chipo = pd.read_csv('https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.tsv', sep='\t')
```

Step 4. See the first 10 entries

In [49]:
```
chipo.head(10)
```

Out[49]:

	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

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

In [50]:
```
chipo.shape[0]
```

Out[50]:
```
4622
```

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

In [51]:
```
chipo.shape[1]
```

Out[51]:
```
5
```

Step 7. Print the name of all the columns.

In [52]:
```
chipo.columns.values
```

Out[52]:
```
array(['order_id', 'quantity', 'item_name', 'choice_description',
       'item_price'], dtype=object)
```

Step 8. How is the dataset indexed?

In [53]:
```
chipo.index
```

Out[53]:
```
RangeIndex(start=0, stop=4622, step=1)
```

Step 9. Which was the most-ordered item?

In [54]:
```
chipo.groupby(['item_name']).quantity.sum().sort_values(ascending = False).index[0]
```

Out[54]:
```
'Chicken Bowl'
```

In [55]:
```
chipo.groupby('item_name').sum().sort_values(['quantity'], ascending=False).head(1)
```

Out[55]:

	order_id	quantity
item_name		
Chicken Bowl	713926	761

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

In [56]:
```
chipo.groupby(['item_name']).quantity.sum().sort_values(ascending = False).values[0]
```

Out[56]: 761

In [57]:
```
chipo.groupby('item_name').sum().sort_values(['quantity'], ascending=False).head(1)
```

Out[57]:

	order_id	quantity
item_name		
Chicken Bowl	713926	761

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

In [58]:
```
chipo.groupby(['choice_description']).quantity.sum().sort_values(ascending = False).index[0]
```

Out[58]: '[Diet Coke]'

In [59]:
```
chipo.groupby('choice_description').sum().sort_values(['quantity'], ascending=False).head(1)
```

Out[59]:

	order_id	quantity
choice_description		
[Diet Coke]	123455	159

Step 12. How many items were orderd in total?

In [60]:
```
chipo.quantity.sum()
```

Out[60]: 4972

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

In [61]:
```
chipo.dtypes.item_price
chipo.item_price = chipo.item_price.apply(lambda x: float(x[1:]))
chipo.item_price.dtypes
```

Out[61]: dtype('float64')

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

In [62]:
```
chipo['revenue'] = chipo['quantity']*chipo.item_price
total_revenue = chipo.revenue.sum()
total_revenue
```

Out[62]: 39237.02

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

In [63]:
```
total_order = chipo.order_id.nunique()
total_order
```

Out[63]: 1834

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

In [64]:
```
total_revenue / total_order
```

Out[64]: 21.39423118865867

In [65]:
```
order_grouped = chipo.groupby(by=['order_id']).sum()
order_grouped.mean()['revenue']
```

Out[65]: 21.394231188658654

Step 17. How many different items are sold?

chipo.item_name.nunique()

In [66]:
```
chipo.item_name.value_counts().count()
```

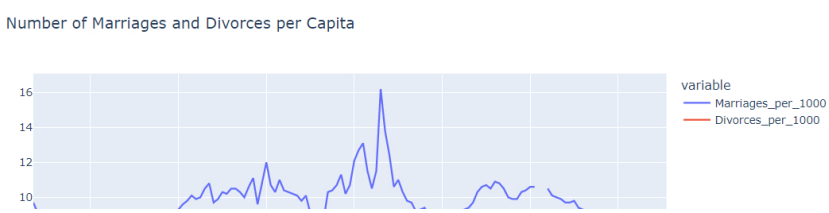
Out[66]: 50

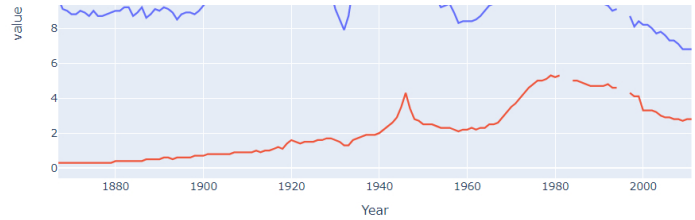
Question 6

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 [4]:
```
import pandas as pd
import plotly.express as px
import numpy as np
```

In [5]:
```
df = pd.read_csv("us-marriages-divorces-1867-2014.csv")
fig = px.line(df, x='Year', y=['Marriages_per_1000','Divorces_per_1000'], title='Number of Marriages and Divorces per Capita')
fig.show()
```

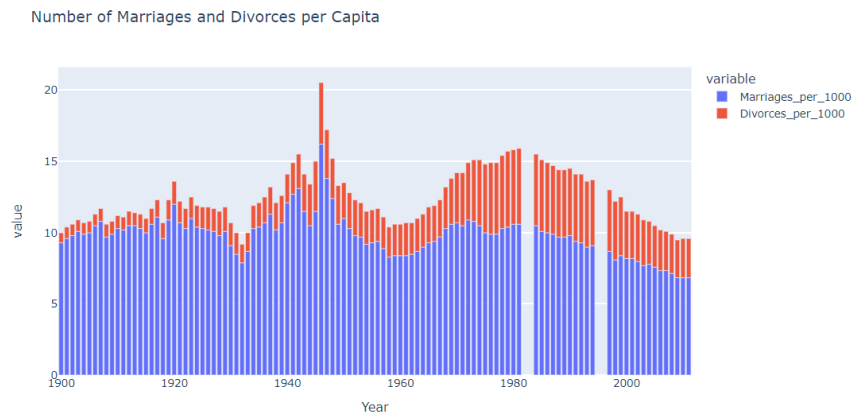




Question 7

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 [6]:
```
df = pd.read_csv("us-marriages--divorces-1867-2014.csv")
fig = px.bar(df, x='Year', y=['Marriages_per_1000','Divorces_per_1000'], title='Number of Marriages and Divorces per Capita')
fig.show()
```



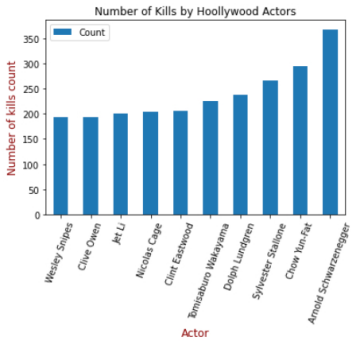
Question 8

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 [7]:
```
import pandas as pd
import matplotlib.pyplot as plot

actor_kill_counts = pd.read_csv("actor_kill_counts.csv")
```

In [8]:
```
actor_kill_counts.sort_values('Count').plot.bar(x="Actor", y="Count", rot=70, title="Number of Kills by Hoollywood Actors");
plot.xlabel("Actor", fontdict={'color':'darkred','size':12})
plot.ylabel("Number of kills count",fontdict={'color':'darkred','size':12})
plot.show(block=True);
```



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 [9]:
```
import pandas as pd
import plotly.express as px
roman = pd.read_csv("roman-emperor-reigns.csv")
```

In [10]:
```
roman
```

Out[10]:

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
...	...	...	...
63	Valentinian I	11.00	Natural causes
64	Valens	14.00	Killed in battle
65	Gratian	16.00	Assassinated
66	Valentinian II	17.00	Possibly assassinated
67	Theodosius I	16.00	Natural causes

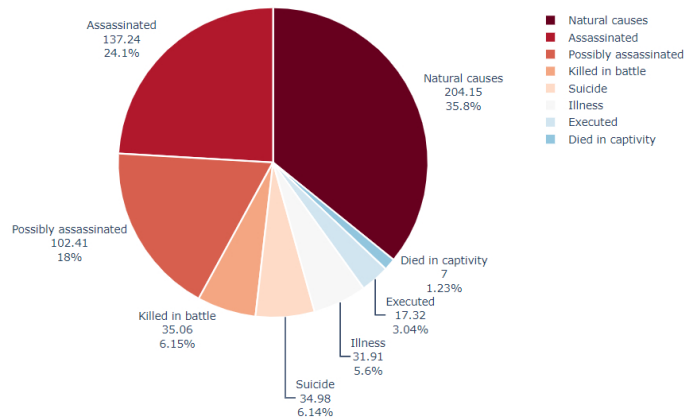
68 rows × 3 columns

In [11]:

```
fig = px.pie(values=roman["Length_of_Reign"], names=roman["Cause_of_Death"],
             color_discrete_sequence=px.colors.sequential.RdBu)

fig.update_traces(textposition='outside',
                  textinfo='percent+label+value',
                  marker=dict(line=dict(color='#FFFFFF', width=2)),
                  textfont_size=12)

fig.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 [12]:

```
import pandas as pd
import plotly.express as px

df = pd.read_csv("arcade-revenue-vs-cs-doctorates.csv")
```

In [13]:

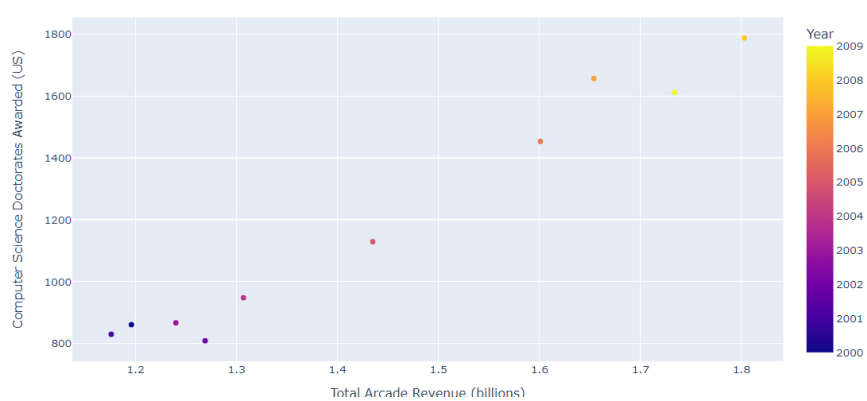
```
df
```

Out[13]:

	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 [14]:

```
fig = px.scatter(df, x='Total Arcade Revenue (billions)', y='Computer Science Doctorates Awarded (US)', color='Year')
fig.show()
```



```
In [ ]:
```