Seatwork 7.2 Programming Exercise: Data Wrangling with Pandas - Part 2

Exercise Part 4:

1. Using the meteorite data from the Meteorite_Landings.csv file, create a pivot table that shows both the number of meteorites and the 95th percentile of meteorite mass for those that were found versus observed falling per year from 2005 through 2009 (inclusive). Hint: Be sure to convert the year column to a number as we did in the previous exercise.

2. Using the meteorite data from the Meteorite_Landings.csv file, compare summary statistics of the mass column for the meteorites that were found versus observed falling.

In [18]:
```
#1.
import pandas as pd

meteorite = pd.read_csv('Meteorite_Landings.csv') # loading the data

meteorite['year'] = meteorite['year'].str.slice().str[6:10] # getting the year from
meteorite['year'] = meteorite['year'].fillna(0) # fill missing values
meteorite.year = meteorite.year.astype('int64') # converting first the year column

meteorite2 = meteorite.copy() # create a copy of the dataframe

meteorite2.head() # checking of the dataframe
```

Out[18]:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoL
0	Aachen	1	Valid	L5	21.0	Fell	1880	50.77500	6.08333	6
1	Aarhus	2	Valid	H6	720.0	Fell	1951	56.18333	10.23333	(56 10
2	Abee	6	Valid	EH4	107000.0	Fell	1952	54.21667	-113.00000	(54
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976	16.88333	-99.90000	(16
4	Achiras	370	Valid	L6	780.0	Fell	1902	-33.16667	-64.95000	(-33

In [26]:
```
# get the years from 2005 to 2009
years = meteorite2[(meteorite2["year"] >= 2005) & (meteorite2["year"] <= 2009)]
```

In [31]:
```
newPivotTable = years.pivot_table(index = 'year', # use the year as index
                                  columns = 'fall', # use the fall as column
                                  values = 'mass (g)', # use mass as values
```

```
                                    aggfunc = ["count", lambda x: x.quantile(0.95)])
newPivotTable # calling the new pivot table
```

Out[31]:

	count		<lambda>	
fall	**Fell**	**Found**	**Fell**	**Found**
year				
2005	NaN	874.0	NaN	4500.00
2006	5.0	2450.0	25008.0	1600.50
2007	8.0	1181.0	89675.0	1126.90
2008	9.0	948.0	106000.0	2274.80
2009	5.0	1492.0	8333.4	1397.25

In [39]:

```
#2
meteorite3 = meteorite.copy() # create another copy for this number

meteorite3.groupby(['fall'])['mass (g)'].describe()
'''Using the group by function it will set the index as the two unique instance in
then the the summary statistics for the mass column is called afterwards to serve a
for this table.'''
```

Out[39]:

	count	mean	std	min	25%	50%	75%	max
fall								
Fell	1075.0	47070.715023	717067.125826	0.1	686.00	2800.0	10450.0	23000000.0
Found	44510.0	12461.922983	571105.752311	0.0	6.94	30.5	178.0	60000000.0

In [ ]:

Exercise Part 5:

1. Using the taxi trip data in the 2019_Yellow_Taxi_Trip_Data.csv file, resample the data to an hourly frequency based on the dropoff time. Calculate the total trip_distance, fare_amount, tolls_amount, and tip_amount, then find the 5 hours with the most tips.

In [137…

```
import pandas as pd

taxi = pd.read_csv('2019_Yellow_Taxi_Trip_Data.csv') # loading the dataframe

taxi2 = taxi.copy() # creating a copy of dataframe
taxi2 = taxi2.rename(columns = {"tpep_dropoff_datetime":"dropoff"}) # changing colu

taxi2['dropoff'] = pd.to_datetime(taxi2['dropoff']) # changing the datatype to date
```

In [139…

```
taxi2 = taxi2.set_index('dropoff').sort_index() # setting the dropoff as the index
```

```
data = taxi2.resample('h').count()
# using the resample and count method I can resample the data to an hourly frequenc

data.head() # checking the modification of the dataset
```

Out[139…

dropoff	vendorid	tpep_pickup_datetime	passenger_count	trip_distance	ratecodeid	store
2019-10-23 07:00:00	1	1	1	1	1	
2019-10-23 08:00:00	2	2	2	2	2	
2019-10-23 09:00:00	2	2	2	2	2	
2019-10-23 10:00:00	0	0	0	0	0	
2019-10-23 11:00:00	0	0	0	0	0	

In [141…

```
data[['trip_distance', 'fare_amount', 'tolls_amount', 'tip_amount']]

'''I called the data variable and then showed the needed columns for output.
This is possible since the resample method allows us to view the data form a
aggregated time series data to a new frequency'''
```

Out[141…

	trip_distance	fare_amount	tolls_amount	tip_amount
dropoff				
2019-10-23 07:00:00	1	1	1	1
2019-10-23 08:00:00	2	2	2	2
2019-10-23 09:00:00	2	2	2	2
2019-10-23 10:00:00	0	0	0	0
2019-10-23 11:00:00	0	0	0	0
2019-10-23 12:00:00	1	1	1	1
2019-10-23 13:00:00	3	3	3	3
2019-10-23 14:00:00	5	5	5	5
2019-10-23 15:00:00	16	16	16	16
2019-10-23 16:00:00	6291	6291	6291	6291
2019-10-23 17:00:00	3254	3254	3254	3254
2019-10-23 18:00:00	384	384	384	384
2019-10-23 19:00:00	5	5	5	5
2019-10-23 20:00:00	0	0	0	0
2019-10-23 21:00:00	0	0	0	0
2019-10-23 22:00:00	0	0	0	0
2019-10-23 23:00:00	0	0	0	0
2019-10-24 00:00:00	2	2	2	2
2019-10-24 01:00:00	0	0	0	0
2019-10-24 02:00:00	0	0	0	0
2019-10-24 03:00:00	0	0	0	0
2019-10-24 04:00:00	0	0	0	0
2019-10-24 05:00:00	0	0	0	0
2019-10-24 06:00:00	0	0	0	0
2019-10-24 07:00:00	4	4	4	4
2019-10-24 08:00:00	3	3	3	3
2019-10-24 09:00:00	1	1	1	1
2019-10-24 10:00:00	0	0	0	0
2019-10-24 11:00:00	0	0	0	0

	trip_distance	fare_amount	tolls_amount	tip_amount
dropoff				
2019-10-24 12:00:00	2	2	2	2
2019-10-24 13:00:00	0	0	0	0
2019-10-24 14:00:00	3	3	3	3
2019-10-24 15:00:00	5	5	5	5
2019-10-24 16:00:00	15	15	15	15
2019-10-24 17:00:00	1	1	1	1

In [133…

```
data.nlargest(5, 'tip_amount')['tip_amount']
'''Since the data is now resampled we can find the 5 hours with the most tips.
Using the nlargest and finding the top 5 from the tip_amount column.'''
```

Out[133…

```
dropoff
2019-10-23 16:00:00    6291
2019-10-23 17:00:00    3254
2019-10-23 18:00:00     384
2019-10-23 15:00:00      16
2019-10-24 16:00:00      15
Name: tip_amount, dtype: int64
```

In [ ]: