```
In [1]:  import numpy as np
         import matplotlib as mpl
         import matplotlib.pyplot as plt
         import pandas as pd
```

```
In [6]:  url = 'https://raw.githubusercontent.com/justmarkham/DAT8/master/data/chipotle.ts
```

```
In [7]:  # read the above file
```

```
In [14]:  df = pd.read_csv(url, sep = '\t')
```

```
In [9]:  #print first 5 and last 7 records
```

```
In [15]:  df.head(5)
```

Out[15]:

	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

```
In [16]:  df.tail(7)
```

Out[16]:

	order_id	quantity	item_name	choice_description	item_price
4615	1832	1	Chicken Soft Tacos	[Fresh Tomato Salsa, [Rice, Cheese, Sour Cream]]	$8.75
4616	1832	1	Chips and Guacamole	NaN	$4.45
4617	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Sour ...	$11.75
4618	1833	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Sour Cream, Cheese...	$11.75
4619	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$11.25
4620	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	$8.75
4621	1834	1	Chicken Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	$8.75

```
In [12]:  # print total records and type of variables
```

```
In [17]: df.info()#

         # OR

         df.shape[0]
         # 4622 observations
```

```
         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 4622 entries, 0 to 4621
         Data columns (total 5 columns):
         order_id            4622 non-null int64
         quantity            4622 non-null int64
         item_name           4622 non-null object
         choice_description  3376 non-null object
         item_price          4622 non-null object
         dtypes: int64(2), object(3)
         memory usage: 180.6+ KB
```

Out[17]: 4622

```
In [18]: #Print the name of all the columns.
```

```
In [19]: df.columns
```

```
Out[19]: Index(['order_id', 'quantity', 'item_name', 'choice_description',
                'item_price'],
               dtype='object')
```

```
In [20]: #How is the dataset indexed?
```

```
In [21]: df.index
```

Out[21]: RangeIndex(start=0, stop=4622, step=1)

```
In [22]: #Which was the most ordered item? and How many items were ordered?
```

```
In [23]: c = df.groupby('item_name')
         c = c.sum()
         c = c.sort_values(['quantity'], ascending=False)
         c.head(1)
```

Out[23]:

item_name	order_id	quantity
Chicken Bowl	713926	761

```
In [24]: #What was the most ordered item in the choice_description column?
```

```
In [25]: c = df.groupby('choice_description').sum()
         c = c.sort_values(['quantity'], ascending=False)
         c.head(1)
```

Out[25]:

	order_id	quantity
choice_description		
[Diet Coke]	123455	159

```
In [26]: #Turn the item price into a float
```

```
In [27]: dollar = lambda x: float(x[1:-1])
         df.item_price = df.item_price.apply(dollar)
```

```
In [28]: #How much was the revenue for the period in the dataset?
```

```
In [30]: revenue = (df['quantity']* df['item_price']).sum()

         print('Revenue was: $' + str(np.round(revenue,2)))
```

```
Revenue was: $39237.02
```

```
In [31]: #print a data frame with only two columns item_name and item_price
```

```
In [32]:  # delete the duplicates in item_name and quantity
          filtered = df.drop_duplicates(['item_name','quantity'])

          # select only the products with quantity equals to 1
          one_prod = filtered[filtered.quantity == 1]

          # select only the item_name and item_price columns
          price_per_item = one_prod[['item_name', 'item_price']]

          # sort the values from the most to less expensive
          price_per_item.sort_values(by = "item_price", ascending = False)
```

Out[32]:

	item_name	item_price
606	Steak Salad Bowl	11.89
1229	Barbacoa Salad Bowl	11.89
1132	Carnitas Salad Bowl	11.89
7	Steak Burrito	11.75
168	Barbacoa Crispy Tacos	11.75
39	Barbacoa Bowl	11.75
738	Veggie Soft Tacos	11.25
186	Veggie Salad Bowl	11.25
62	Veggie Bowl	11.25
57	Veggie Burrito	11.25
250	Chicken Salad	10.98
5	Chicken Bowl	10.98
8	Steak Soft Tacos	9.25
554	Carnitas Crispy Tacos	9.25
237	Carnitas Soft Tacos	9.25
56	Barbacoa Soft Tacos	9.25
92	Steak Crispy Tacos	9.25
664	Steak Salad	8.99
54	Steak Bowl	8.99
3750	Carnitas Salad	8.99
21	Barbacoa Burrito	8.99
27	Carnitas Burrito	8.99
33	Carnitas Bowl	8.99
11	Chicken Crispy Tacos	8.75
12	Chicken Soft Tacos	8.75
44	Chicken Salad Bowl	8.75
1653	Veggie Crispy Tacos	8.49
16	Chicken Burrito	8.49

	item_name	item_price
1694	Veggie Salad	8.49
1414	Salad	7.40
510	Burrito	7.40
520	Crispy Tacos	7.40
673	Bowl	7.40
298	6 Pack Soft Drink	6.49
10	Chips and Guacamole	4.45
1	Izze	3.39
2	Nantucket Nectar	3.39
674	Chips and Mild Fresh Tomato Salsa	3.00
111	Chips and Tomatillo Red Chili Salsa	2.95
233	Chips and Roasted Chili Corn Salsa	2.95
38	Chips and Tomatillo Green Chili Salsa	2.95
3	Chips and Tomatillo-Green Chili Salsa	2.39
300	Chips and Tomatillo-Red Chili Salsa	2.39
191	Chips and Roasted Chili-Corn Salsa	2.39
0	Chips and Fresh Tomato Salsa	2.39
40	Chips	2.15
6	Side of Chips	1.69
263	Canned Soft Drink	1.25
28	Canned Soda	1.09
34	Bottled Water	1.09

In [33]: `#What was the quantity of the most expensive item ordered?`

In [34]: `df.sort_values(by = "item_price", ascending = False).head(1)`

Out[34]:

	order_id	quantity	item_name	choice_description	item_price
3598	1443	15	Chips and Fresh Tomato Salsa	NaN	44.25

In [35]: `#How many times were a Veggie Salad Bowl ordered?`

```
In [36]:  df[df.item_name == "Veggie Salad Bowl"]
```

Out[36]:

	order_id	quantity	item_name	choice_description	item_price
186	83	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
295	128	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	11.25
455	195	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
496	207	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Rice, Lettuce, Guacamole...	11.25
960	394	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Lettu...	8.75
1316	536	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	8.75
1884	760	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
2156	869	1	Veggie Salad Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25
2223	896	1	Veggie Salad Bowl	[Roasted Chili Corn Salsa, Fajita Vegetables]	8.75
2269	913	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	8.75
2683	1066	1	Veggie Salad Bowl	[Roasted Chili Corn Salsa, [Fajita Vegetables,...	8.75
3223	1289	1	Veggie Salad Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25
3293	1321	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Rice, Black Beans, Chees...	8.75
4109	1646	1	Veggie Salad Bowl	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	11.25
4201	1677	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Black...	11.25
4261	1700	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	11.25
4541	1805	1	Veggie Salad Bowl	[Tomatillo Green Chili Salsa, [Fajita Vegetabl...	8.75
4573	1818	1	Veggie Salad Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Pinto...	8.75

```
In [15]:  import numpy as np
          import matplotlib as mpl
          import matplotlib.pyplot as plt
          import pandas as pd
```

```
In [16]:  df = pd.read_csv('https://raw.githubusercontent.com/justmarkham/AT8/master/data/
          df.head()
```

Out[16]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	AS
1	Albania	89	132	54	4.9	EU
2	Algeria	25	0	14	0.7	AF
3	Andorra	245	138	312	12.4	EU
4	Angola	217	57	45	5.9	AF

```
In [17]:  #Which continent drinks more beer on average?
```

```
In [18]:  df.groupby('continent').beer_servings.mean()
```

```
Out[18]:  continent
          AF      61.471698
          AS      37.045455
          EU     193.777778
          OC      89.687500
          SA     175.083333
          Name: beer_servings, dtype: float64
```

```
In [19]:  #For each continent print the statistics for wine consumption.
```

```
In [20]:  df.groupby('continent').wine_servings.describe()
```

Out[20]:

	count	mean	std	min	25%	50%	75%	max
continent								
AF	53.0	16.264151	38.846419	0.0	1.0	2.0	13.00	233.0
AS	44.0	9.068182	21.667034	0.0	0.0	1.0	8.00	123.0
EU	45.0	142.222222	97.421738	0.0	59.0	128.0	195.00	370.0
OC	16.0	35.625000	64.555790	0.0	1.0	8.5	23.25	212.0
SA	12.0	62.416667	88.620189	1.0	3.0	12.0	98.50	221.0

```
In [21]:  url = "https://raw.githubusercontent.com/guipsamora/pandas_exercises/master/04_A|
          crime = pd.read_csv(url)
          crime.head()
```

Out[21]:

	Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
0	1960	179323175	3384200	288460	3095700	9110	17190	107840	1543
1	1961	182992000	3488000	289390	3198600	8740	17220	106670	1567
2	1962	185771000	3752200	301510	3450700	8530	17550	110860	1645
3	1963	188483000	4109500	316970	3792500	8640	17650	116470	1742
4	1964	191141000	4564600	364220	4200400	9360	21420	130390	2030

```
In [22]:  #Convert the type of the column Year to datetime64
```

```
In [23]:  crime.Year = pd.to_datetime(crime.Year, format='%Y')
          crime.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 55 entries, 0 to 54
          Data columns (total 12 columns):
          Year                55 non-null datetime64[ns]
          Population          55 non-null int64
          Total               55 non-null int64
          Violent             55 non-null int64
          Property            55 non-null int64
          Murder              55 non-null int64
          Forcible_Rape       55 non-null int64
          Robbery             55 non-null int64
          Aggravated_assault  55 non-null int64
          Burglary            55 non-null int64
          Larceny_Theft       55 non-null int64
          Vehicle_Theft       55 non-null int64
          dtypes: datetime64[ns](1), int64(11)
          memory usage: 5.2 KB
```

```
In [24]:  #Set the Year column as the index of the dataframe¶
```

```
In [25]:  crime = crime.set_index('Year', drop = True)
          crime.head()
```

Out[25]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault
1960-01-01	179323175	3384200	288460	3095700	9110	17190	107840	154320
1961-01-01	182992000	3488000	289390	3198600	8740	17220	106670	156760
1962-01-01	185771000	3752200	301510	3450700	8530	17550	110860	164570
1963-01-01	188483000	4109500	316970	3792500	8640	17650	116470	174210
1964-01-01	191141000	4564600	364220	4200400	9360	21420	130390	203050

```
In [26]:  #Group the year by decades and sum the values¶
```

```
In [27]:  # Uses resample to sum each decade
          crimes = crime.resample('10AS').sum()

          # Uses resample to get the max value only for the "Population" column
          population = crime['Population'].resample('10AS').max()

          # Updating the "Population" column
          crimes['Population'] = population

          crimes
```

Out[27]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Agg
1960-01-01	201385000.0	49295900.0	4134930.0	45160900.0	106180.0	236720.0	1633510.0	
1970-01-01	220099000.0	100991600.0	9607930.0	91383800.0	192230.0	554570.0	4159020.0	
1980-01-01	248239000.0	131123369.0	14074328.0	117048900.0	206439.0	865639.0	5383109.0	
1990-01-01	272690813.0	136582146.0	17527048.0	119053499.0	211664.0	998827.0	5748930.0	
2000-01-01	307006550.0	115012044.0	13968056.0	100944369.0	163068.0	922499.0	4230366.0	
2010-01-01	318857056.0	50167967.0	6072017.0	44095950.0	72867.0	421059.0	1749809.0	
2020-01-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

```
In [28]: crime.head()
```

Out[28]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault
1960-01-01	179323175	3384200	288460	3095700	9110	17190	107840	154320
1961-01-01	182992000	3488000	289390	3198600	8740	17220	106670	156760
1962-01-01	185771000	3752200	301510	3450700	8530	17550	110860	164570
1963-01-01	188483000	4109500	316970	3792500	8640	17650	116470	174210
1964-01-01	191141000	4564600	364220	4200400	9360	21420	130390	203050

```
In [31]: #Return the first 3 rows of the DataFrame df.

         df = crime
```

```
In [32]: df.iloc[:3]
```

Out[32]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assault
1960-01-01	179323175	3384200	288460	3095700	9110	17190	107840	154320
1961-01-01	182992000	3488000	289390	3198600	8740	17220	106670	156760
1962-01-01	185771000	3752200	301510	3450700	8530	17550	110860	164570

```
In [33]: #Select just the 'Murder' and 'Robbery' columns from the DataFrame df and print j
```

```
In [35]: df.loc[:, ['Murder', 'Robbery']].head()
```

Out[35]:

Year	Murder	Robbery
1960-01-01	9110	107840
1961-01-01	8740	106670
1962-01-01	8530	110860
1963-01-01	8640	116470
1964-01-01	9360	130390

```
In [37]: df[['Murder', 'Robbery']].head()
```

Out[37]:

Year	Murder	Robbery
1960-01-01	9110	107840
1961-01-01	8740	106670
1962-01-01	8530	110860
1963-01-01	8640	116470
1964-01-01	9360	130390

```
In [38]: #Select the data in rows [3, 4, 8] and in columns ['Murder', 'Robbery']
```

```
In [39]: df.loc[df.index[[3, 4, 8]], ['Murder', 'Robbery']]
```

Out[39]:

Year	Murder	Robbery
1963-01-01	8640	116470
1964-01-01	9360	130390
1968-01-01	13800	262840

```
In [45]: #Select only the rows where the number of murder is greater than 24,000
```

```
In [46]: df[df['Murder'] > 24000]
```

Out[46]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
1991-01-01	252177000	14872900	1911770	12961100	24700	106590	687730	1092?
1993-01-01	257908000	14144800	1926020	12218800	24530	106010	659870	1135(

```
In [47]: #Select the rows the murder is between 20k and 24k (inclusive)
```

In [51]: `df[df['Murder'].between(20000, 24000)]`

Out[51]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
1974-01-01	211392000	10253400	974720	9278700	20710	55400	442400	456
1975-01-01	213124000	11292400	1039710	10252700	20510	56090	470500	492
1979-01-01	220099000	12249500	1208030	11041500	21460	76390	480700	629
1980-01-01	225349264	13408300	1344520	12063700	23040	82990	565840	672
1981-01-01	229146000	13423800	1361820	12061900	22520	82500	592910	663
1982-01-01	231534000	12974400	1322390	11652000	21010	78770	553130	669
1986-01-01	240132887	13211869	1489169	11722700	20613	91459	542775	834
1987-01-01	242282918	13508700	1483999	12024700	20096	91110	517704	855
1988-01-01	245807000	13923100	1566220	12356900	20680	92490	542970	910
1989-01-01	248239000	14251400	1646040	12605400	21500	94500	578330	951
1990-01-01	248709873	14475600	1820130	12655500	23440	102560	639270	1054
1992-01-01	255082000	14438200	1932270	12505900	23760	109060	672480	1126
1994-01-01	260341000	13989500	1857670	12131900	23330	102220	618950	1113
1995-01-01	262755000	13862700	1798790	12063900	21610	97470	580510	1099

In [52]: *#Calculate the mean murder for each different year in df.*

```
In [53]: df.groupby('Year')['Murder'].mean()
```

Out[53]: Year
         1960-01-01   9110
         1961-01-01   8740
         1962-01-01   8530
         1963-01-01   8640
         1964-01-01   9360
         1965-01-01   9960
         1966-01-01   11040
         1967-01-01   12240
         1968-01-01   13800
         1969-01-01   14760
         1970-01-01   16000
         1971-01-01   17780
         1972-01-01   18670
         1973-01-01   19640
         1974-01-01   20710
         1975-01-01   20510
         1976-01-01   18780
         1977-01-01   19120
         1978-01-01   19560
         1979-01-01   21460
         1980-01-01   23040
         1981-01-01   22520
         1982-01-01   21010
         1983-01-01   19310
         1984-01-01   18690
         1985-01-01   18980
         1986-01-01   20613
         1987-01-01   20096
         1988-01-01   20680
         1989-01-01   21500
         1990-01-01   23440
         1991-01-01   24700
         1992-01-01   23760
         1993-01-01   24530
         1994-01-01   23330
         1995-01-01   21610
         1996-01-01   19650
         1997-01-01   18208
         1998-01-01   16914
         1999-01-01   15522
         2000-01-01   15586
         2001-01-01   16037
         2002-01-01   16229
         2003-01-01   16528
         2004-01-01   16148
         2005-01-01   16740
         2006-01-01   17030
         2007-01-01   16929
         2008-01-01   16442
         2009-01-01   15399
         2010-01-01   14772
         2011-01-01   14661
         2012-01-01   14866
         2013-01-01   14319

```
        2014-01-01    14249
        Name: Murder, dtype: int64
```

In [55]: #Sort df first by the values in the 'Murder' in decending order,
         #then by the value in the 'Violent' column in ascending order.

```
In [58]: df.sort_values(by=['Murder', 'Violent'], ascending=[False, True])
```

Out[58]:

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
1991-01-01	252177000	14872900	1911770	12961100	24700	106590	687730	1092
1993-01-01	257908000	14144800	1926020	12218800	24530	106010	659870	1135
1992-01-01	255082000	14438200	1932270	12505900	23760	109060	672480	1126
1990-01-01	248709873	14475600	1820130	12655500	23440	102560	639270	1054
1994-01-01	260341000	13989500	1857670	12131900	23330	102220	618950	1113
1980-01-01	225349264	13408300	1344520	12063700	23040	82990	565840	672
1981-01-01	229146000	13423800	1361820	12061900	22520	82500	592910	663
1995-01-01	262755000	13862700	1798790	12063900	21610	97470	580510	1099
1989-01-01	248239000	14251400	1646040	12605400	21500	94500	578330	951
1979-01-01	220099000	12249500	1208030	11041500	21460	76390	480700	629
1982-01-01	231534000	12974400	1322390	11652000	21010	78770	553130	669
1974-01-01	211392000	10253400	974720	9278700	20710	55400	442400	456
1988-01-01	245807000	13923100	1566220	12356900	20680	92490	542970	910
1986-01-01	240132887	13211869	1489169	11722700	20613	91459	542775	834
1975-01-01	213124000	11292400	1039710	10252700	20510	56090	470500	492
1987-01-01	242282918	13508700	1483999	12024700	20096	91110	517704	855
1996-01-01	265228572	13493863	1688540	11805300	19650	96250	535590	1037
1973-01-01	209851000	8718100	875910	7842200	19640	51400	384220	420
1978-01-01	218059000	11209000	1085550	10123400	19560	67610	426930	571
1983-01-01	233981000	12108600	1258090	10850500	19310	78920	506570	653
1977-01-01	216332000	10984500	1029580	9955000	19120	63500	412610	534
1985-01-01	238740000	12431400	1328800	11102600	18980	88670	497870	723

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
1976-01-01	214659000	11349700	1004210	10345500	18780	57080	427810	500!
1984-01-01	236158000	11881800	1273280	10608500	18690	84230	485010	685!
1972-01-01	208230000	8248800	834900	7413900	18670	46850	376290	3930
1997-01-01	267637000	13194571	1634770	11558175	18208	96153	498534	1023!
1971-01-01	206212000	8588200	816500	7771700	17780	42260	387700	368!
2006-01-01	299398484	11401511	1418043	9983568	17030	92757	447403	8608
2007-01-01	301621157	11251828	1408337	9843481	16929	90427	445125	855!
1998-01-01	270296000	12475634	1531044	10944590	16914	93103	446625	974
2005-01-01	296507061	11565499	1390745	10174754	16740	94347	417438	862!
2003-01-01	290690788	11826538	1383676	10442862	16528	93883	414235	859(
2008-01-01	304374846	11160543	1392628	9767915	16442	90479	443574	842
2002-01-01	287973924	11878954	1423677	10455277	16229	95235	420806	891
2004-01-01	293656842	11679474	1360088	10319386	16148	95089	401470	847!
2001-01-01	285317559	11876669	1439480	10437480	16037	90863	423557	909(
1970-01-01	203235298	8098000	738820	7359200	16000	37990	349860	334!
2000-01-01	281421906	11608072	1425486	10182586	15586	90178	408016	911
1999-01-01	272690813	11634378	1426044	10208334	15522	89411	409371	911
2009-01-01	307006550	10762956	1325896	9337060	15399	89241	408742	812!
2012-01-01	313873685	10219059	1217067	9001992	14866	85141	355051	762(
2010-01-01	309330219	10363873	1251248	9112625	14772	85593	369089	781!
1969-01-01	201385000	7410900	661870	6749000	14760	37170	298850	311(
2011-01-01	311587816	10258774	1206031	9052743	14661	84175	354772	752
2013-01-01	316497531	9850445	1199684	8650761	14319	82109	345095	726!

Year	Population	Total	Violent	Property	Murder	Forcible_Rape	Robbery	Aggravated_assa
2014-01-01	318857056	9475816	1197987	8277829	14249	84041	325802	7412
1968-01-01	199399000	6720200	595010	6125200	13800	31670	262840	2867
1967-01-01	197457000	5903400	499930	5403500	12240	27620	202910	257
1966-01-01	195576000	5223500	430180	4793300	11040	25820	157990	2353
1965-01-01	193526000	4739400	387390	4352000	9960	23410	138690	2153
1964-01-01	191141000	4564600	364220	4200400	9360	21420	130390	2030
1960-01-01	179323175	3384200	288460	3095700	9110	17190	107840	1543
1961-01-01	182992000	3488000	289390	3198600	8740	17220	106670	1561
1963-01-01	188483000	4109500	316970	3792500	8640	17650	116470	1742
1962-01-01	185771000	3752200	301510	3450700	8530	17550	110860	1645

In [59]:
```
#read the following data set
#https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datase
```

In [60]:
```
df = pd.read_csv('https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/
```

In [61]:
```
df.head()
```

Out[61]:

	Unnamed: 0	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [62]:
```
#For each cyl type and each number of gears, find the mean mileage.
```

In [65]:
```python
df.pivot_table(index='cyl', columns='gear', values='mpg', aggfunc='mean')
```

Out[65]:

gear	3	4	5
cyl			
4	21.50	26.925	28.2
6	19.75	19.750	19.7
8	15.05	NaN	15.4

In [ ]: