Sales Walmart June 1

June 1, 2020

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

[2]:
```
sales = pd.read_pickle('walmart_sales.pkl')
sales.head()
```

[2]:

	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	1	A	1	2010-02-05	24924.50	False	5.727778	0.679451	8.106
1	1	A	2	2010-02-05	50605.27	False	5.727778	0.679451	8.106
2	1	A	3	2010-02-05	13740.12	False	5.727778	0.679451	8.106
3	1	A	4	2010-02-05	39954.04	False	5.727778	0.679451	8.106
4	1	A	5	2010-02-05	32229.38	False	5.727778	0.679451	8.106

0.0.1 Mean and median

Summary statistics are exactly what they sound like - they summarize many numbers in one statistic. For example, mean, median, minimum, maximum, and standard deviation are summary statistics. Calculating summary statistics allows you to get a better sense of your data, even if there's a lot of it.

sales is available and pandas is loaded as pd.

Explore your new DataFrame first by printing the first few rows of the sales DataFrame.

[4]:
```
# Print the head of the sales DataFrame
sales.head()
```

[4]:

	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	1	A	1	2010-02-05	24924.50	False	5.727778	0.679451	8.106
1	1	A	2	2010-02-05	50605.27	False	5.727778	0.679451	8.106

```
2      1    A           3 2010-02-05      13740.12       False       5.727778
3      1    A           4 2010-02-05      39954.04       False       5.727778
4      1    A           5 2010-02-05      32229.38       False       5.727778

   fuel_price_usd_per_l  unemployment
0              0.679451         8.106
1              0.679451         8.106
2              0.679451         8.106
3              0.679451         8.106
4              0.679451         8.106
```

[5]:
```
# Print information about the columns in sales.
# Print the info about the sales DataFrame
sales.info()
#So, it contains 9 columns and 413119 rows with all non-null values
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 413119 entries, 0 to 413118
Data columns (total 9 columns):
store                  413119 non-null int64
type                   413119 non-null object
department             413119 non-null int32
date                   413119 non-null datetime64[ns]
weekly_sales           413119 non-null float64
is_holiday             413119 non-null bool
temperature_c          413119 non-null float64
fuel_price_usd_per_l   413119 non-null float64
unemployment           413119 non-null float64
dtypes: bool(1), datetime64[ns](1), float64(4), int32(1), int64(1), object(1)
memory usage: 27.2+ MB
```

[6]:
```
# Print the mean of the weekly_sales column.
sales["weekly_sales"].mean()
```

[6]:
```
16094.726811185497
```

[10]:
```
# Print the median of the weekly_sales column.
sales['weekly_sales'].median()
```

[10]:
```
7682.47
```

0.0.2 Summarizing dates

Summary statistics can also be calculated on date columns which have values with the data type datetime64. Some summary statistics — like mean — don't make a ton of sense on dates, but others are super helpful, for example minimum and maximum, which allow you to see what time range your data covers.

sales is available and pandas is loaded as pd.

[12]:
```
#Print the maximum of the date column.
print(sales['date'].max())
```

```
2012-10-26 00:00:00
```

[13]:
```
# Print the minimum of the date column.
print(sales['date'].min())
```

```
2010-02-05 00:00:00
```

0.0.3 Efficient summaries

While pandas and NumPy have tons of functions, sometimes you may need a different function to summarize your data.

The .agg() method allows you to apply your own custom functions to a DataFrame, as well as apply functions to more than one column of a DataFrame at once, making your aggregations super efficient.

In the custom function for this exercise, "IQR" is short for inter-quartile range, which is the 75th percentile minus the 25th percentile. It's an alternative to standard deviation that is helpful if your data contains outliers.

[14]:
```
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)
```

[17]:
```
# Use the custom iqr function defined for you along with .agg()
# to print the IQR of the temperature_c column of sales.
# iqr(sales['weekly_sales'])
sales['temperature_c'].agg(iqr)
```

[17]:
```
15.299999999999994
```

[20]:
```
'''
Update the column selection to use the custom iqr function with .agg()
to print the IQR of temperature_c, fuel_price_usd_per_l, and unemployment, in␣
 ↪that order.
'''
sales[['temperature_c', 'fuel_price_usd_per_l', 'unemployment']].agg(iqr)
```

[20]:
```
temperature_c           15.300000
fuel_price_usd_per_l     0.211866
unemployment             1.672000
dtype: float64
```

[21]:
```
# Update the aggregation functions called by .agg(): include iqr and np.median␣
↪in that order.
sales[['temperature_c', 'fuel_price_usd_per_l', 'unemployment']].agg([iqr, np.
↪median])
```

[21]:

	temperature_c	fuel_price_usd_per_l	unemployment
iqr	15.30	0.211866	1.672
median	16.75	0.911922	7.852

0.0.4 Cumulative statistics

Cumulative statistics can also be helpful in tracking summary statistics over time. In this exercise, you'll calculate the cumulative sum and cumulative max of a department's weekly sales, which will allow you to identify what the total sales were so far as well as what the highest weekly sales were so far.

A DataFrame called sales_1_1 has been created for you, which contains the sales data for department 1 of store 1. pandas is loaded as pd.

[35]:
```
sales_1_1 = sales[(sales['department'] == 1) & (sales['store'] == 1)]
```

[36]:
```
sales_1_1.head()
```

[36]:

	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	1	A	1	2010-02-05	24924.50	False	5.727778	0.679451	8.106
73	1	A	1	2010-02-12	46039.49	True	3.616667	0.673111	8.106
145	1	A	1	2010-02-19	41595.55	False	4.405556	0.664129	8.106
218	1	A	1	2010-02-26	19403.54	False	8.127778	0.676545	8.106
290	1	A	1	2010-03-05	21827.90	False	8.055556	0.693452	8.106

[37]:
```
# Sort the rows of sales_1_1 by the date column in ascending order.
sales_1_1.sort_values("date")
```

[37]:

	store	type	department	date	weekly_sales	is_holiday	…
0	1	A	1	2010-02-05	24924.50	False	…
73	1	A	1	2010-02-12	46039.49	True	…
145	1	A	1	2010-02-19	41595.55	False	…
218	1	A	1	2010-02-26	19403.54	False	…
290	1	A	1	2010-03-05	21827.90	False	…
…	…	…	…	…	…	…	…

```
9883       1    A           1 2012-09-28      18947.81       False
9956       1    A           1 2012-10-05      21904.47       False
10028      1    A           1 2012-10-12      22764.01       False
10101      1    A           1 2012-10-19      24185.27       False
10172      1    A           1 2012-10-26      27390.81       False

       temperature_c  fuel_price_usd_per_l  unemployment
0           5.727778              0.679451         8.106
73          3.616667              0.673111         8.106
145         4.405556              0.664129         8.106
218         8.127778              0.676545         8.106
290         8.055556              0.693452         8.106
…                …                     …             …
9883       24.488889              0.968455         6.908
9956       20.305556              0.955511         6.573
10028      17.216667              0.951284         6.573
10101      19.983333              0.949435         6.573
10172      20.644444              0.926188         6.573

[143 rows x 9 columns]
```

[40]:
```
# Get the cumulative sum of weekly_sales and add it as a new column of␣
↪sales_1_1 called cum_weekly_sales.
sales_1_1['cum_weekly_sales'] = sales_1_1['weekly_sales'].cumsum()
sales_1_1
```

```
/home/roshan/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[40]:
```
       store type  department       date  weekly_sales  is_holiday  \
0          1    A           1 2010-02-05      24924.50       False
73         1    A           1 2010-02-12      46039.49        True
145        1    A           1 2010-02-19      41595.55       False
218        1    A           1 2010-02-26      19403.54       False
290        1    A           1 2010-03-05      21827.90       False
…      …  …           …          …             …              …
9883       1    A           1 2012-09-28      18947.81       False
9956       1    A           1 2012-10-05      21904.47       False
10028      1    A           1 2012-10-12      22764.01       False
10101      1    A           1 2012-10-19      24185.27       False
10172      1    A           1 2012-10-26      27390.81       False
```

```
       temperature_c  fuel_price_usd_per_l  unemployment  cum_weekly_sales
0           5.727778              0.679451         8.106          24924.50
73          3.616667              0.673111         8.106          70963.99
145         4.405556              0.664129         8.106         112559.54
218         8.127778              0.676545         8.106         131963.08
290         8.055556              0.693452         8.106         153790.98
…                …                     …             …               …
9883       24.488889              0.968455         6.908        3123160.62
9956       20.305556              0.955511         6.573        3145065.09
10028      17.216667              0.951284         6.573        3167829.10
10101      19.983333              0.949435         6.573        3192014.37
10172      20.644444              0.926188         6.573        3219405.18

[143 rows x 10 columns]
```

[42]:
```
# Get the cumulative maximum of weekly_sales, and add it as a column called␣
↪cum_max_sales.
sales_1_1['cum_max_sales'] = sales_1_1['weekly_sales'].cummax()
sales_1_1
```

```
/home/roshan/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[42]:
```
       store type  department       date  weekly_sales  is_holiday  \
0          1    A           1 2010-02-05      24924.50       False
73         1    A           1 2010-02-12      46039.49        True
145        1    A           1 2010-02-19      41595.55       False
218        1    A           1 2010-02-26      19403.54       False
290        1    A           1 2010-03-05      21827.90       False
…      …  …          …         …             …           …
9883       1    A           1 2012-09-28      18947.81       False
9956       1    A           1 2012-10-05      21904.47       False
10028      1    A           1 2012-10-12      22764.01       False
10101      1    A           1 2012-10-19      24185.27       False
10172      1    A           1 2012-10-26      27390.81       False

       temperature_c  fuel_price_usd_per_l  unemployment  cum_weekly_sales  \
0           5.727778              0.679451         8.106          24924.50
73          3.616667              0.673111         8.106          70963.99
145         4.405556              0.664129         8.106         112559.54
```

```
218         8.127778              0.676545         8.106          131963.08
290         8.055556              0.693452         8.106          153790.98
…                …                     …             …                …
9883       24.488889              0.968455         6.908         3123160.62
9956       20.305556              0.955511         6.573         3145065.09
10028      17.216667              0.951284         6.573         3167829.10
10101      19.983333              0.949435         6.573         3192014.37
10172      20.644444              0.926188         6.573         3219405.18

       cum_max_sales
0           24924.50
73          46039.49
145         46039.49
218         46039.49
290         46039.49
…                …
9883        57592.12
9956        57592.12
10028       57592.12
10101       57592.12
10172       57592.12

[143 rows x 11 columns]
```

[44]:
```
# Print the date, weekly_sales, cum_weekly_sales, and cum_max_sales columns.
sales_1_1[['date', 'weekly_sales', 'cum_weekly_sales', 'cum_max_sales']]
```

[44]:
```
            date  weekly_sales  cum_weekly_sales  cum_max_sales
0     2010-02-05      24924.50          24924.50       24924.50
73    2010-02-12      46039.49          70963.99       46039.49
145   2010-02-19      41595.55         112559.54       46039.49
218   2010-02-26      19403.54         131963.08       46039.49
290   2010-03-05      21827.90         153790.98       46039.49
…           …            …                 …              …
9883  2012-09-28      18947.81        3123160.62       57592.12
9956  2012-10-05      21904.47        3145065.09       57592.12
10028 2012-10-12      22764.01        3167829.10       57592.12
10101 2012-10-19      24185.27        3192014.37       57592.12
10172 2012-10-26      27390.81        3219405.18       57592.12

[143 rows x 4 columns]
```

[45]:
```
sales.head()
```

[45]:
```
   store type  department       date  weekly_sales  is_holiday  temperature_c  \
0      1    A           1 2010-02-05      24924.50       False       5.727778
1      1    A           2 2010-02-05      50605.27       False       5.727778
```

```
2      1    A           3 2010-02-05      13740.12       False       5.727778
3      1    A           4 2010-02-05      39954.04       False       5.727778
4      1    A           5 2010-02-05      32229.38       False       5.727778

   fuel_price_usd_per_l  unemployment
0              0.679451         8.106
1              0.679451         8.106
2              0.679451         8.106
3              0.679451         8.106
4              0.679451         8.106
```

0.0.5 Dropping duplicates

Removing duplicates is an essential skill to get accurate counts, because often you don't want to count the same thing multiple times. In this exercise, you'll create some new DataFrames using unique values from sales.

[48]:
```
'''
Remove rows of sales with duplicate pairs of store and type and save as␣
 ↪store_types and print the head.
'''
store_types = sales.drop_duplicates(subset = ['store', 'type'])
store_types.head()
```

[48]:
```
       store type  department       date  weekly_sales  is_holiday  \
0          1    A           1 2010-02-05      24924.50       False
10244      2    A           1 2010-02-05      35034.06       False
20482      3    B           1 2010-02-05       6453.58       False
29518      4    A           1 2010-02-05      38724.42       False
39790      5    B           1 2010-02-05       9323.89       False

       temperature_c  fuel_price_usd_per_l  unemployment
0           5.727778              0.679451         8.106
10244       4.550000              0.679451         8.324
20482       7.616667              0.679451         7.368
29518       6.533333              0.686319         8.623
39790       4.277778              0.679451         6.566
```

[52]:
```
# Remove rows of sales with duplicate pairs of store and department and save as␣
 ↪store_depts and print the head.
store_depts = sales.drop_duplicates(subset = ['store', 'department'])
store_depts.head()
```

[52]:
```
   store type  department       date  weekly_sales  is_holiday  temperature_c  \
0      1    A           1 2010-02-05      24924.50       False       5.727778
1      1    A           2 2010-02-05      50605.27       False       5.727778
```

```
2      1    A           3 2010-02-05      13740.12       False       5.727778
3      1    A           4 2010-02-05      39954.04       False       5.727778
4      1    A           5 2010-02-05      32229.38       False       5.727778

   fuel_price_usd_per_l  unemployment
0              0.679451         8.106
1              0.679451         8.106
2              0.679451         8.106
3              0.679451         8.106
4              0.679451         8.106
```

[53]:
```
# Subset the rows that are holiday weeks, and drop the duplicate dates, saving␣
↪as holiday_dates.
holiday_dates = sales[sales['is_holiday'] == True].drop_duplicates(subset =␣
↪"date")
holiday_dates.head()
```

[53]:
```
      store type  department       date  weekly_sales  is_holiday  \
73        1    A           1 2010-02-12      46039.49        True
2218      1    A           1 2010-09-10      18194.74        True
3014      1    A           1 2010-11-26      18820.29        True
3372      1    A           1 2010-12-31      19124.58        True
3800      1    A           1 2011-02-11      37887.17        True

      temperature_c  fuel_price_usd_per_l  unemployment
73         3.616667              0.673111         8.106
2218      25.938889              0.677602         7.787
3014      18.066667              0.722511         7.838
3372       9.127778              0.777459         7.838
3800       2.438889              0.798328         7.742
```

[54]:
```
# Select the date column of holiday_dates, and print.
holiday_dates['date']
# Dazzling duplicate dropping! The holiday weeks correspond to the Superbowl in␣
↪February, Labor Day
# in September, Thanksgiving in November, and Christmas
# in December. Now that the duplicates are removed, it's time to do some␣
↪counting.
```

[54]:
```
73     2010-02-12
2218   2010-09-10
3014   2010-11-26
3372   2010-12-31
3800   2011-02-11
5940   2011-09-09
6731   2011-11-25
7096   2011-12-30
```

```
7527   2012-02-10
9667   2012-09-07
Name: date, dtype: datetime64[ns]
```

0.0.6 Counting categorical variables

Counting is a great way to get an overview of your data and to spot curiosities that you might not notice otherwise. In this exercise, you'll count the number of each type of store and the number of each department number.

The stores and departments DataFrames you created in the last exercise are available and pandas is imported as pd.

[76]:
```
store_type = sales.drop_duplicates(subset = ['store', 'type'])
stores = store_type[['store', 'type']].reset_index(drop = True)
stores.columns = ['store_num', 'store_type']
stores.head()
```

[76]:

	store_num	store_type
0	1	A
1	2	A
2	3	B
3	4	A
4	5	B

[73]:
```
store_department = sales.drop_duplicates(subset = ['store', 'department'])
departments = store_department[['store', 'department']].reset_index(drop = True)
departments.columns = ['store_num', 'department_num'] # Renaming column names
departments.head()
```

[73]:

	store_num	department_num
0	1	1
1	1	2
2	1	3
3	1	4
4	1	5

[77]:
```
stores.head()
```

[77]:

	store_num	store_type
0	1	A
1	2	A
2	3	B
3	4	A
4	5	B

[75]:
```
departments.head()
```

[75]:
```
   store_num  department_num
0          1               1
1          1               2
2          1               3
3          1               4
4          1               5
```

[81]:
```
# Count the number of stores of each store type.
store_counts = stores['store_type'].value_counts()
store_counts
```

[81]:
```
A    22
B    17
C     6
Name: store_type, dtype: int64
```

[84]:
```
# Count the proportion of stores of each store type
store_props = stores['store_type'].value_counts(normalize = True)*100
store_props  # In percentage
```

[84]:
```
A    48.888889
B    37.777778
C    13.333333
Name: store_type, dtype: float64
```

[85]:
```
# Count the number of different department numbers, sorting the counts in␣
↪descending order.
dept_counts_sorted = departments['department_num'].value_counts(sort = True)
dept_counts_sorted
```

[85]:
```
1     45
9     45
4     45
6     45
8     45
      ..
37    20
50    14
43     5
39     5
65     1
Name: department_num, Length: 81, dtype: int64
```

[88]:
```
# Count the proportion of different department numbers, sorting the proportions␣
↪in descending order.
dept_props_sorted = departments['department_num'].value_counts(normalize =␣
↪True, sort= True)*100
```

```
dept_props_sorted #In percentage
```

[88]:
```
1     1.377832
9     1.377832
4     1.377832
6     1.377832
8     1.377832
        …
37    0.612370
50    0.428659
43    0.153092
39    0.153092
65    0.030618
Name: department_num, Length: 81, dtype: float64
```

0.0.7 What percent of sales occurred at each store type?

While .groupby() is useful, you can calculate grouped summary statistics without it.

Walmart distinguishes three types of stores: "supercenters", "discount stores", and "neighborhood markets", encoded in this dataset as type "A", "B", and "C". In this exercise, you'll calculate the total sales made at each store type, without using .groupby(). You can then use these numbers to see what proportion of Walmart's total sales were made at each.

sales is available and pandas is imported as pd.

[90]:
```
# Calculate the total weekly sales over the whole dataset.
sales_all = sales['weekly_sales'].sum()
sales_all
```

[90]:
```
6649037445.509999
```

[92]:
```
# Subset for type "A" stores, and calculate their total weekly sales.
sales_A = sales[sales['type'] == 'A']['weekly_sales'].sum()
sales_A
```

[92]:
```
4331014722.749999
```

[93]:
```
# Subset for type B stores, calc total weekly sales
sales_B = sales[sales['type'] == 'B']['weekly_sales'].sum()
sales_B
```

[93]:
```
1912519195.2199998
```

[94]:
```
# Subset for type C stores, calc total weekly sales
sales_C = sales[sales['type'] == 'C']['weekly_sales'].sum()
sales_C
```

[94]:
```
405503527.53999996
```

[99]:
```
# Combine the A/B/C results into a list, and divide by overall sales to get the␣
↪proportion of sales by type.
sales_propn_by_type = ([sales_A, sales_B, sales_C]/sales_all)*100
sales_propn_by_type
```

[99]:
```
array([65.13746927, 28.76385057,  6.09868016])
```

0.0.8 Calculations with .groupby()

The .groupby() method makes life much easier. In this exercise, you'll perform the same calculations as last time, except you'll use the .groupby() method. You'll also perform calculations on data grouped by two variables to see if sales differs by store type depending on if it's a holiday week or not.

sales is available and pandas is loaded as pd.

[107]:
```
# Group sales by "type", take the sum of "weekly_sales", and store as␣
↪sales_by_type.
sales_by_type = sales.groupby('type')["weekly_sales"].sum()
sales_by_type
```

[107]:
```
type
A    4.331015e+09
B    1.912519e+09
C    4.055035e+08
Name: weekly_sales, dtype: float64
```

[113]:
```
sales_propn_by_type= sales_by_type/sum(sales_by_type)
sales_propn_by_type*100
```

[113]:
```
type
A    65.137469
B    28.763851
C     6.098680
Name: weekly_sales, dtype: float64
```

0.0.9 Multiple grouped summaries

Earlier in this chapter you saw that the .agg() method is useful to compute multiple statistics on multiple variables. It also works with grouped data. NumPy, which is imported as np, has many different summary statistics functions, including:

```
np.min()
np.max()
np.mean()
```

np.median()

sales is available and pandas is imported as pd.

[119]:
```
# Get the min, max, mean, and median of weekly_sales for each store type using
#.groupby() and .agg(). Store this as sales_stats. Make sure to use numpy␣
 ↪functions!
import numpy as np
sales_stats = sales.groupby('type')['weekly_sales'].agg([np.min, np.max, np.
 ↪mean, np.median])
sales_stats
```

[119]:

type	amin	amax	mean	median
A	-4988.94	474330.10	20099.568043	10105.17
B	-3924.00	693099.36	12335.331875	6269.02
C	-379.00	112152.35	9519.532538	1149.67

[120]:
```
# Get the min, max, mean, and median of unemployment and fuel_price_usd_per_l
# for each store type. Store this as unemp_fuel_stats
unemp_fuel_stats = sales.groupby('type')['unemployment',␣
 ↪'fuel_price_usd_per_l'].agg([np.min, np.max, np.mean, np.median])
```

[121]:
```
unemp_fuel_stats
```

[121]:

	unemployment				fuel_price_usd_per_l			
	amin	amax	mean	median	amin	amax	mean	median
type								
A	3.879	14.313	7.791595	7.818	0.653034	1.180321	0.883391	0.902676
B	4.125	14.313	7.889666	7.806	0.664129	1.180321	0.892997	0.922225
C	5.217	14.313	8.934350	8.300	0.664129	1.180321	0.888848	0.902676

0.0.10 Pivoting on one variable

Pivot tables are the standard way of aggregating data in spreadsheets. In pandas, pivot tables are essentially just another way of performing grouped calculations. That is, the .pivot_table() method is just an alternative to .groupby().

In this exercise, you'll perform calculations using .pivot_table() to replicate the calculations you performed in the last lesson using .groupby().

sales is available and pandas is imported as pd

[122]:
```
# Get the mean weekly_sales by type using .pivot_table() and store as␣
↪mean_sales_by_type
mean_sales_by_type = sales.pivot_table(values = 'weekly_sales', index = 'type')␣
↪# group by type and find mean weekly sales for each type
mean_sales_by_type
```

[122]:
```
      weekly_sales
type
A     20099.568043
B     12335.331875
C      9519.532538
```

[124]:
```
# Get the mean and median (using NumPy functions) of weekly_sales by type using␣
↪.pivot_table() and store as mean_med_sales_by_type.
mean_med_sales_by_type = sales.pivot_table(values = 'weekly_sales', index =␣
↪'type', aggfunc=[np.mean, np.median])
mean_med_sales_by_type
```

[124]:
```
              mean       median
      weekly_sales weekly_sales
type
A     20099.568043     10105.17
B     12335.331875      6269.02
C      9519.532538      1149.67
```

[126]:
```
# Get the mean of weekly_sales by type and is_holiday using .pivot_table() and␣
↪store as mean_sales_by_type_holiday.
mean_sales_by_type_holiday = sales.pivot_table(values = 'weekly_sales', index =␣
↪"type", columns = 'is_holiday')
mean_sales_by_type_holiday
```

[126]:
```
is_holiday         False         True
type
A           20008.746759  21297.517824
B           12248.741339  13478.844240
C            9518.528116   9532.963131
```

0.0.11 Fill in missing values and sum values with pivot tables

The .pivot_table() method has several useful arguments, including fill_value and margins.

`fill_value replaces missing values with a real value (known as imputation). What to replace mis`
`margins is a shortcut for when you pivoted by two variables, but also wanted to pivot by each `

In this exercise, you'll practice using these arguments to up your pivot table skills, which will help you crunch numbers more efficiently!

[127]:
```
# Print the mean weekly_sales by department and type, filling in any missing␣
↪values with 0.
# Check for null values if any
sales['weekly_sales'].isnull().values.any()
```

[127]:
```
False
```

[128]:
```
sales.pivot_table(values = 'weekly_sales', index = 'department', columns =␣
↪'type', fill_value = 0)
```

[128]:
```
type                   A             B             C
department
1           22956.887886  17990.876158   8951.733462
2           51994.674873  43051.996919  14424.851713
3           13881.033137  12965.414311    820.276818
4           32973.814075  21259.895804  13669.370396
5           26803.448045  21184.602916    767.600774
…                    …             …             …
95          97094.026043  40580.306862  50641.564872
96          19900.943552   4752.674874  15766.025431
97          22093.807101   3543.243304  13419.542809
98          10979.816195    299.951644   5479.758054
99            431.443064     25.716667      8.330952

[81 rows x 3 columns]
```

[130]:
```
# Print the mean weekly_sales by department and type,
# filling in any missing values with 0 and summing all rows and columns.
sales.pivot_table(values = 'weekly_sales', index = 'department', columns =␣
↪'type', fill_value=0, margins=True)
# All section will contain the mean of the row, through all column values
```

[130]:
```
type                   A             B             C           All
department
1           22956.887886  17990.876158   8951.733462  19213.485088
2           51994.674873  43051.996919  14424.851713  43607.020113
3           13881.033137  12965.414311    820.276818  11793.698516
4           32973.814075  21259.895804  13669.370396  25974.630238
5           26803.448045  21184.602916    767.600774  21365.583515
…                    …             …             …           …
96          19900.943552   4752.674874  15766.025431  15217.211505
97          22093.807101   3543.243304  13419.542809  14437.120839
98          10979.816195    299.951644   5479.758054   6973.013875
99            431.443064     25.716667      8.330952    415.487065
```

```
All          20099.568043  12335.331875   9519.532538  16094.726811

[82 rows x 4 columns]
```

[ ]: