AGGREGATING DATAFRAMES

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.

Load Pandas and data into workspace

In [4]:

```
#Load pandas and dataset
import pandas as pd
sales = pd.read_csv('sales_subset.csv')
```

Explore your new DataFrame first by printing the first few rows of the sales DataFrame.
Print information about the columns in sales.
Print the mean of the weekly_sales column.
Print the median of the weekly_sales column

In [5]:

```
# Print the head of the sales DataFrame
print(sales.head())

# Print the info about the sales DataFrame
print(sales.info())

# Print the mean of weekly_sales
print(sales['weekly_sales'].mean())

# Print the median of weekly_sales
print(sales['weekly_sales'].median())
```

```
   Unnamed: 0  store type  department        date  weekly_sales  is_holiday  \
0           0      1    A           1  2010-02-05      24924.50       False
1           1      1    A           1  2010-03-05      21827.90       False
2           2      1    A           1  2010-04-02      57258.43       False
3           3      1    A           1  2010-05-07      17413.94       False
4           4      1    A           1  2010-06-04      17558.09       False

   temperature_c  fuel_price_usd_per_l  unemployment
0       5.727778              0.679451         8.106
1       8.055556              0.693452         8.106
2      16.816667              0.718284         7.808
3      22.527778              0.748928         7.808
4      27.050000              0.714586         7.808
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10774 entries, 0 to 10773
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Unnamed: 0            10774 non-null  int64
 1   store                 10774 non-null  int64
 2   type                  10774 non-null  object
 3   department            10774 non-null  int64
 4   date                  10774 non-null  object
 5   weekly_sales          10774 non-null  float64
 6   is_holiday            10774 non-null  bool
 7   temperature_c         10774 non-null  float64
 8   fuel_price_usd_per_l  10774 non-null  float64
 9   unemployment          10774 non-null  float64
dtypes: bool(1), float64(4), int64(3), object(2)
memory usage: 768.2+ KB
```

```
None
23843.95014850566
12049.064999999999
```

Summarizing dates

Summary statistics can also be calculated on date columns that 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.

Print the maximum of the date column.
Print the minimum of the date column.

In [6]:

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

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

```
2012-10-26
2010-02-05
```

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. For example,

df['column'].agg(function)

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.

Use the custom iqr function defined for you along with .agg() to print the IQR of the temperature_c column of sales

In [7]:

```
# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Print IQR of the temperature_c column
print(sales['temperature_c'].agg(iqr))
```

```
16.583333333333336
```

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.

In [8]:

```
# A custom IQR function
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Update to print IQR of temperature_c, fuel_price_usd_per_l, & unemployment
print(sales[["temperature_c", 'fuel_price_usd_per_l','unemployment' ]].agg(iqr))
```

```
temperature_c           16.583333
fuel_price_usd_per_l     0.073176
unemployment             0.565000
```

```
unemployment            0.565000
dtype: float64
```

Update the aggregation functions called by .agg(): include iqr and np.median in that order

In [9]:

```
# Import NumPy and create custom IQR function
import numpy as np
def iqr(column):
    return column.quantile(0.75) - column.quantile(0.25)

# Update to print IQR and median of temperature_c, fuel_price_usd_per_l, & unemployment
print(sales[["temperature_c", "fuel_price_usd_per_l", "unemployment"]].agg([iqr, np.medi
an]))
```

```
        temperature_c  fuel_price_usd_per_l  unemployment
iqr         16.583333              0.073176         0.565
median      16.966667              0.743381         8.099
```

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.

In [42]:

```
#Creacting DATAFRAME
sales_s_1 = sales[sales['store'] ==1]
sales_1_1 = sales_s_1[sales_s_1['department']==1]

#Print first five elements
print(sales_1_1.head())
```

```
   Unnamed: 0  store type  department        date  weekly_sales  is_holiday  \
0           0      1    A           1  2010-02-05      24924.50       False
1           1      1    A           1  2010-03-05      21827.90       False
2           2      1    A           1  2010-04-02      57258.43       False
3           3      1    A           1  2010-05-07      17413.94       False
4           4      1    A           1  2010-06-04      17558.09       False

   temperature_c  fuel_price_usd_per_l  unemployment
0       5.727778              0.679451         8.106
1       8.055556              0.693452         8.106
2      16.816667              0.718284         7.808
3      22.527778              0.748928         7.808
4      27.050000              0.714586         7.808
```

Sort the rows of sales_1_1 by the date column in ascending order.
Get the cumulative sum of weekly_sales and add it as a new column of sales_1_1 called cum_weekly_sales.
Get the cumulative maximum of weekly_sales, and add it as a column called cum_max_sales.
Print the date, weekly_sales, cum_weekly_sales, and cum_max_sales columns.

In [43]:

```
# Sort sales_1_1 by date
sales_1_1 = sales_1_1.sort_values('date')

# Get the cumulative sum of weekly_sales, add as cum_weekly_sales col
sales_1_1['cum_weekly_sales'] = sales_1_1['weekly_sales'].cumsum()

# Get the cumulative max of weekly_sales, add as cum_max_sales col
sales_1_1['cum_max_sales'] = sales_1_1['weekly_sales'].cummax()
```

```
# See the columns you calculated
print(sales_1_1[["date", "weekly_sales", "cum_weekly_sales", "cum_max_sales"]])
```

```
          date  weekly_sales  cum_weekly_sales  cum_max_sales
0   2010-02-05      24924.50          24924.50       24924.50
1   2010-03-05      21827.90          46752.40       24924.50
2   2010-04-02      57258.43         104010.83       57258.43
3   2010-05-07      17413.94         121424.77       57258.43
4   2010-06-04      17558.09         138982.86       57258.43
5   2010-07-02      16333.14         155316.00       57258.43
6   2010-08-06      17508.41         172824.41       57258.43
7   2010-09-03      16241.78         189066.19       57258.43
8   2010-10-01      20094.19         209160.38       57258.43
9   2010-11-05      34238.88         243399.26       57258.43
10  2010-12-03      22517.56         265916.82       57258.43
11  2011-01-07      15984.24         281901.06       57258.43
```

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.

sales is available and pandas is imported as pd.

Remove rows of sales with duplicate pairs of store and type and save as store_types and print the head.
Remove rows of sales with duplicate pairs of store and department and save as store_depts and print the head.
Subset the rows that are holiday weeks using the is_holiday column, and drop the duplicate dates, saving as holiday_dates.
Select the date column of holiday_dates, and print.

In [44]:

```
# Drop duplicate store/type combinations
store_types = sales.drop_duplicates(subset=['store','type'])
print(store_types.head())

# Drop duplicate store/department combinations
store_depts = sales.drop_duplicates(subset=['store','department'])
print(store_depts.head())

# Subset the rows where is_holiday is True and drop duplicate dates
holiday_dates = sales[sales['is_holiday'] == True].drop_duplicates(subset='date')

# Print date col of holiday_dates
print(holiday_dates)
```

```
      Unnamed: 0  store type  department        date  weekly_sales  \
0              0      1    A           1  2010-02-05      24924.50   
901          901      2    A           1  2010-02-05      35034.06   
1798        1798      4    A           1  2010-02-05      38724.42   
2699        2699      6    A           1  2010-02-05      25619.00   
3593        3593     10    B           1  2010-02-05      40212.84   

      is_holiday  temperature_c  fuel_price_usd_per_l  unemployment  
0          False       5.727778              0.679451         8.106  
901        False       4.550000              0.679451         8.324  
1798       False       6.533333              0.686319         8.623  
2699       False       4.683333              0.679451         7.259  
3593       False      12.411111              0.782478         9.765  
    Unnamed: 0  store type  department        date  weekly_sales  is_holiday  \
0            0      1    A           1  2010-02-05      24924.50       False   
12          12      1    A           2  2010-02-05      50605.27       False   
24          24      1    A           3  2010-02-05      13740.12       False   
36          36      1    A           4  2010-02-05      39954.04       False   
48          48      1    A           5  2010-02-05      32229.38       False   

    temperature_c  fuel_price_usd_per_l  unemployment
```

```
    temperature_c  fuel_price_usd_per_l  unemployment
0        5.727778              0.679451         8.106
12       5.727778              0.679451         8.106
24       5.727778              0.679451         8.106
36       5.727778              0.679451         8.106
48       5.727778              0.679451         8.106
      Unnamed: 0  store type  department        date  weekly_sales  \
498          498      1    A          45  2010-09-10         11.47
691          691      1    A          77  2011-11-25       1431.00
2315        2315      4    A          47  2010-02-12        498.00
6735        6735     19    A          39  2012-09-07         13.41
6810        6810     19    A          47  2010-12-31       -449.00
6815        6815     19    A          47  2012-02-10         15.00
6820        6820     19    A          48  2011-09-09        197.00

      is_holiday  temperature_c  fuel_price_usd_per_l  unemployment
498         True      25.938889              0.677602         7.787
691         True      15.633333              0.854861         7.866
2315        True      -1.755556              0.679715         8.623
6735        True      22.333333              1.076766         8.193
6810        True      -1.861111              0.881278         8.067
6815        True       0.338889              1.010723         7.943
6820        True      20.155556              1.038197         7.806
```

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 using the DataFrames you created in the previous exercise:

Drop duplicate store/type combinations
store_types = sales.drop_duplicates(subset=["store", "type"])

Drop duplicate store/department combinations
store_depts = sales.drop_duplicates(subset=["store", "department"])

The store_types and store_depts DataFrames you created in the last exercise are available, and pandas is imported as pd.

Count the number of stores of each store type in store_types.
Count the proportion of stores of each store type in store_types.
Count the number of different departments in store_depts, sorting the counts in descending order.
Count the proportion of different departments in store_depts, sorting the proportions in descending order

In [45]:

```python
# Count the number of stores of each type
store_counts = store_types["type"].value_counts()
print(store_counts)

# Get the proportion of stores of each type
store_props = store_types["type"].value_counts(normalize=True)
print(store_props)

# Count the number of each department number and sort
dept_counts_sorted = store_depts["department"].value_counts(sort=True)
print(dept_counts_sorted)

# Get the proportion of departments of each number and sort
dept_props_sorted = store_depts["department"].value_counts(sort=True, normalize=True)
print(dept_props_sorted)
```

```
A    11
B     1
Name: type, dtype: int64
A    0.916667
B    0.083333
```

```
Name: type, dtype: float64
1      12
55     12
72     12
71     12
67     12
       ..
37     10
48      8
50      6
39      4
43      2
Name: department, Length: 80, dtype: int64
1      0.012917
55     0.012917
72     0.012917
71     0.012917
67     0.012917
         ...
37     0.010764
48     0.008611
50     0.006459
39     0.004306
43     0.002153
Name: department, Length: 80, dtype: float64
```

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 type.

sales is available and pandas is imported as pd.

Calculate the total weekly_sales over the whole dataset.
Subset for type "A" stores, and calculate their total weekly sales.
Do the same for type "B" and type "C" stores.
Combine the A/B/C results into a list, and divide by sales_all to get the proportion of sales by type.

In [46]:

```python
# Calc total weekly sales
sales_all = sales["weekly_sales"].sum()

# Subset for type A stores, calc total weekly sales
sales_A = sales[sales["type"] == "A"]["weekly_sales"].sum()

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

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

# Get proportion for each type
sales_propn_by_type = [sales_A,sales_B, sales_C] / sales_all
print(sales_propn_by_type)
```

```
[0.9097747 0.0902253 0.        ]
```

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 differ by store type depending on if it's a holiday week or not.

sales is available and pandas is loaded as pd

Group sales by "type", take the sum of "weekly_sales", and store as sales_by_type.
Calculate the proportion of sales at each store type by dividing by the sum of sales_by_type. Assign to sales_propn_by_type.

In [47]:

```
# Group by type; calc total weekly sales
sales_by_type = sales.groupby("type")["weekly_sales"].sum()

# Get proportion for each type
sales_propn_by_type = sales_by_type / sum(sales_by_type)
print(sales_propn_by_type)
```

```
type
A    0.909775
B    0.090225
Name: weekly_sales, dtype: float64
```

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 differ by store type depending on if it's a holiday week or not.

sales is available and pandas is loaded as pd

Group sales by "type" and "is_holiday", take the sum of weekly_sales, and store as sales_by_type_is_holiday

In [48]:

```
# From previous step
sales_by_type = sales.groupby("type")["weekly_sales"].sum()

# Group by type and is_holiday; calc total weekly sales
sales_by_type_is_holiday = sales.groupby(["type","is_holiday"])["weekly_sales"].sum()
print(sales_by_type_is_holiday)
```

```
type  is_holiday
A     False         2.336927e+08
      True          2.360181e+04
B     False         2.317678e+07
      True          1.621410e+03
Name: weekly_sales, dtype: float64
```

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, and np.median.

sales is available and pandas is imported as pd

Import numpy with the alias np.
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!

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.

In [49]:

```
# Import numpy with the alias np
import numpy as np

# For each store type, aggregate weekly_sales: get min, max, mean, and median
sales_stats = sales.groupby("type")["weekly_sales"].agg([np.min, np.max, np.mean, np.med
ian])

# Print sales_stats
print(sales_stats)

# For each store type, aggregate unemployment and fuel_price_usd_per_l: get min, max, mea
n, and median
unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([
np.min, np.max, np.mean, np.median])

# Print unemp_fuel_stats
print(unemp_fuel_stats)
```

```
        amin       amax          mean    median
type
A    -1098.0  293966.05  23674.667242  11943.92
B     -798.0  232558.51  25696.678370  13336.08
     unemployment                        fuel_price_usd_per_l            \
             amin   amax      mean median                 amin      amax
type
A           3.879  8.992  7.972611  8.067             0.664129  1.107410
B           7.170  9.765  9.279323  9.199             0.760023  1.107674


          mean    median
type
A     0.744619  0.735455
B     0.805858  0.803348
```

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

Get the mean weekly_sales by type using .pivot_table() and store as mean_sales_by_type

In [50]:

```
# Pivot for mean weekly_sales for each store type
mean_sales_by_type = sales.pivot_table(values="weekly_sales",index="type")

# Print mean_sales_by_type
print(mean_sales_by_type)
```

```
      weekly_sales
type
A     23674.667242
B     25696.678370
```

Get the mean and median (using NumPy functions) of weekly_sales by type using .pivot_table() and store as mean_med_sales_by_type.

In [51]:

```
# Import NumPy as np
import numpy as np

# Pivot for mean and median weekly_sales for each store type
mean_med_sales_by_type = sales.pivot_table(values="weekly_sales",index="type",aggfunc=[n
p.mean,np.median])
# Print mean_med_sales_by_type
print(mean_med_sales_by_type)
```

```
                mean       median
        weekly_sales weekly_sales
type
A       23674.667242     11943.92
B       25696.678370     13336.08
```

Get the mean of weekly_sales by type and is_holiday using .pivot_table() and store as mean_sales_by_type_holiday.

In [52]:

```
# Pivot for mean weekly_sales by store type and holiday
mean_sales_by_type_holiday = sales.pivot_table(values="weekly_sales",index="type",column
s="is_holiday")

# Print mean_sales_by_type_holiday
print(mean_sales_by_type_holiday)
```

```
is_holiday         False       True
type
A           23768.583523  590.04525
B           25751.980533  810.70500
```

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 missing values with is a topic big enough to have its own course (Dealing with Missing Data in Python), but the simplest thing to do is to substitute a dummy value.
margins is a shortcut for when you pivoted by two variables, but also wanted to pivot by each of those variables separately:
it gives the row and column totals of the pivot table contents.
In this exercise, you'll practice using these arguments to up your pivot table skills, which will help you crunch numbers more efficiently!

sales is available and pandas is imported as pd

Print the mean weekly_sales by department and type, filling in any missing values with 0

In [53]:

```
# Print mean weekly_sales by department and type; fill missing values with 0
print(sales.pivot_table(values="weekly_sales",index='department',columns='type',fill_val
ue=0))
```

```
type                    A              B
department
1            30961.725379   44050.626667
2            67600.158788  112958.526667
3            17160.002955   30580.655000
4            44285.399091   51219.654167
5            34821.011364   63236.875000
...                   ...            ...
95          123933.787121   77082.102500
96           21367.042857    9528.538333
```

```
97           28471.266970    5828.873333
98           12875.423182     217.428333
99             379.123659       0.000000

[80 rows x 2 columns]
```

Print the mean weekly_sales by department and type, filling in any missing values with 0 and summing all rows and columns.

In [54]:

```
# Print the mean weekly_sales by department and type; fill missing values with 0s; sum al
l rows and cols
print(sales.pivot_table(values="weekly_sales", index="department", columns="type", fill_
value=0,margins=True))
```

```
type                    A              B            All
department
1            30961.725379   44050.626667   32052.467153
2            67600.158788  112958.526667   71380.022778
3            17160.002955   30580.655000   18278.390625
4            44285.399091   51219.654167   44863.253681
5            34821.011364   63236.875000   37189.000000
...                   ...            ...            ...
96           21367.042857    9528.538333   20337.607681
97           28471.266970    5828.873333   26584.400833
98           12875.423182     217.428333   11820.590278
99             379.123659       0.000000     379.123659
All          23674.667242   25696.678370   23843.950149

[81 rows x 3 columns]
```

In [ ]: