Data Manipulation with pandas

Transforming Data

Let’s master the pandas basics. Learn how to inspect DataFrames and perform fundamental manipulations, including sorting rows, subsetting, and adding new columns.

Link for reference

Inspecting a DataFrame

In [ ]:
```python
# Import pandas using the alias pd
import pandas as pd

#pathway of the file
homelessness = pd.read_csv('C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python
```

In [ ]:
```python
# Print the head of the homelessness data
print(homelessness.head())
```

```
   Unnamed: 0              region       state  individuals  family_members  \
0           0  East South Central     Alabama       2570.0           864.0   
1           1             Pacific      Alaska       1434.0           582.0   
2           2            Mountain     Arizona       7259.0          2606.0   
3           3  West South Central    Arkansas       2280.0           432.0   
4           4             Pacific  California     109008.0         20964.0   

   state_pop  
0    4887681  
1     735139  
2    7158024  
3    3009733  
4   39461588  
```

In [ ]:
```python
# Print information about homelessness
print(homelessness.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Unnamed: 0      51 non-null     int64  
 1   region          51 non-null     object 
 2   state           51 non-null     object 
 3   individuals     51 non-null     float64
 4   family_members  51 non-null     float64
 5   state_pop       51 non-null     int64  
dtypes: float64(2), int64(2), object(2)
memory usage: 2.5+ KB
None
```

In [ ]:

```
# Print the shape of homelessness
print(homelessness.shape)
```

```
(51, 6)
```

In [ ]:

```
# Print a description of homelessness
print(homelessness.describe())
```

```
       Unnamed: 0    individuals  family_members     state_pop
count   51.000000      51.000000       51.000000  5.100000e+01
mean    25.000000    7225.784314     3504.882353  6.405637e+06
std     14.866069   15991.025083     7805.411811  7.327258e+06
min      0.000000     434.000000       75.000000  5.776010e+05
25%     12.500000    1446.500000      592.000000  1.777414e+06
50%     25.000000    3082.000000     1482.000000  4.461153e+06
75%     37.500000    6781.500000     3196.000000  7.340946e+06
max     50.000000  109008.000000    52070.000000  3.946159e+07
```

Parts of a DataFrame

In [ ]:

```
# Print the values of homelessness
print(homelessness.values)
```

```
[[0 'East South Central' 'Alabama' 2570.0 864.0 4887681]
 [1 'Pacific' 'Alaska' 1434.0 582.0 735139]
 [2 'Mountain' 'Arizona' 7259.0 2606.0 7158024]
 [3 'West South Central' 'Arkansas' 2280.0 432.0 3009733]
 [4 'Pacific' 'California' 109008.0 20964.0 39461588]
 [5 'Mountain' 'Colorado' 7607.0 3250.0 5691287]
 [6 'New England' 'Connecticut' 2280.0 1696.0 3571520]
 [7 'South Atlantic' 'Delaware' 708.0 374.0 965479]
 [8 'South Atlantic' 'District of Columbia' 3770.0 3134.0 701547]
 [9 'South Atlantic' 'Florida' 21443.0 9587.0 21244317]
 [10 'South Atlantic' 'Georgia' 6943.0 2556.0 10511131]
 [11 'Pacific' 'Hawaii' 4131.0 2399.0 1420593]
 [12 'Mountain' 'Idaho' 1297.0 715.0 1750536]
 [13 'East North Central' 'Illinois' 6752.0 3891.0 12723071]
 [14 'East North Central' 'Indiana' 3776.0 1482.0 6695497]
 [15 'West North Central' 'Iowa' 1711.0 1038.0 3148618]
 [16 'West North Central' 'Kansas' 1443.0 773.0 2911359]
 [17 'East South Central' 'Kentucky' 2735.0 953.0 4461153]
 [18 'West South Central' 'Louisiana' 2540.0 519.0 4659690]
 [19 'New England' 'Maine' 1450.0 1066.0 1339057]
 [20 'South Atlantic' 'Maryland' 4914.0 2230.0 6035802]
 [21 'New England' 'Massachusetts' 6811.0 13257.0 6882635]
 [22 'East North Central' 'Michigan' 5209.0 3142.0 9984072]
 [23 'West North Central' 'Minnesota' 3993.0 3250.0 5606249]
 [24 'East South Central' 'Mississippi' 1024.0 328.0 2981020]
 [25 'West North Central' 'Missouri' 3776.0 2107.0 6121623]
 [26 'Mountain' 'Montana' 983.0 422.0 1060665]
 [27 'West North Central' 'Nebraska' 1745.0 676.0 1925614]
 [28 'Mountain' 'Nevada' 7058.0 486.0 3027341]
 [29 'New England' 'New Hampshire' 835.0 615.0 1353465]
 [30 'Mid-Atlantic' 'New Jersey' 6048.0 3350.0 8886025]
 [31 'Mountain' 'New Mexico' 1949.0 602.0 2092741]
 [32 'Mid-Atlantic' 'New York' 39827.0 52070.0 19530351]
 [33 'South Atlantic' 'North Carolina' 6451.0 2817.0 10381615]
 [34 'West North Central' 'North Dakota' 467.0 75.0 758080]
 [35 'East North Central' 'Ohio' 6929.0 3320.0 11676341]
 [36 'West South Central' 'Oklahoma' 2823.0 1048.0 3940235]
 [37 'Pacific' 'Oregon' 11139.0 3337.0 4181886]
 [38 'Mid-Atlantic' 'Pennsylvania' 8163.0 5349.0 12800922]
 [39 'New England' 'Rhode Island' 747.0 354.0 1058287]
 [40 'South Atlantic' 'South Carolina' 3082.0 851.0 5084156]
 [41 'West North Central' 'South Dakota' 836.0 323.0 878698]
 [42 'East South Central' 'Tennessee' 6139.0 1744.0 6771631]
 [43 'West South Central' 'Texas' 19199.0 6111.0 28628666]
 [44 'Mountain' 'Utah' 1904.0 972.0 3153550]
 [45 'New England' 'Vermont' 780.0 511.0 624358]
 [46 'South Atlantic' 'Virginia' 3928.0 2047.0 8501286]
 [47 'Pacific' 'Washington' 16424.0 5880.0 7523869]
 [48 'South Atlantic' 'West Virginia' 1021.0 222.0 1804291]
 [49 'East North Central' 'Wisconsin' 2740.0 2167.0 5807406]
 [50 'Mountain' 'Wyoming' 434.0 205.0 577601]]
```

In [ ]:

```python
# Print the column index of homelessness
print(homelessness.columns)
```

```
Index(['Unnamed: 0', 'region', 'state', 'individuals', 'family_members',
       'state_pop'],
      dtype='object')
```

In [ ]:

```python
# Print the row index of homelessness
print(homelessness.index)
```

```
RangeIndex(start=0, stop=51, step=1)
```

Sorting rows

In [ ]:

```python
# Sort homelessness by individuals
homelessness_ind = homelessness.sort_values(["individuals"])

# Print the top few rows
print(homelessness_ind.head())
```

```
    Unnamed: 0              region         state  individuals  family_members  \
50          50            Mountain       Wyoming        434.0           205.0
34          34  West North Central  North Dakota        467.0            75.0
7            7      South Atlantic      Delaware        708.0           374.0
39          39         New England  Rhode Island        747.0           354.0
45          45         New England       Vermont        780.0           511.0

    state_pop
50     577601
34     758080
7      965479
39    1058287
45     624358
```

In [ ]:

```python
# Sort homelessness by descending family members
homelessness_fam = homelessness.sort_values(["family_members"], ascending=[False])

# Print the top few rows
print(homelessness_fam.head())
```

```
    Unnamed: 0              region          state  individuals  \
32          32        Mid-Atlantic       New York      39827.0   
4            4             Pacific     California     109008.0   
21          21         New England  Massachusetts       6811.0   
9            9      South Atlantic        Florida      21443.0   
43          43  West South Central          Texas      19199.0   

    family_members  state_pop  
32         52070.0   19530351  
4          20964.0   39461588  
21         13257.0    6882635  
9           9587.0   21244317  
43          6111.0   28628666  
```

In [ ]:

```python
# Sort homelessness by region, then descending family members
homelessness_reg_fam = homelessness.sort_values(["region", "family_members"], ascending=[True, False])

# Print the top few rows
print(homelessness_reg_fam.head())
```

```
    Unnamed: 0              region      state  individuals  family_members  \
13          13  East North Central   Illinois       6752.0          3891.0   
35          35  East North Central       Ohio       6929.0          3320.0   
22          22  East North Central   Michigan       5209.0          3142.0   
49          49  East North Central  Wisconsin       2740.0          2167.0   
14          14  East North Central    Indiana       3776.0          1482.0   

    state_pop  
13   12723071  
35   11676341  
22    9984072  
49    5807406  
14    6695497  
```

Subsetting columns

In [ ]:

```python
# Select the individuals column
individuals = homelessness["individuals"]

# Print the head of the result
print(individuals.head())
```

```
0      2570.0
1      1434.0
2      7259.0
3      2280.0
4    109008.0
Name: individuals, dtype: float64
```

In [ ]:

```
# Select the state and family_members columns
state_fam = homelessness[["state", "family_members"]]

# Print the head of the result
print(state_fam.head())
```

```
        state  family_members
0     Alabama           864.0
1      Alaska           582.0
2     Arizona          2606.0
3    Arkansas           432.0
4  California         20964.0
```

In [ ]:

```
# Select only the individuals and state columns, in that order
ind_state = homelessness[["individuals", "state"]]

# Print the head of the result
print(ind_state.head())
```

```
   individuals       state
0       2570.0     Alabama
1       1434.0      Alaska
2       7259.0     Arizona
3       2280.0    Arkansas
4     109008.0  California
```

Subsetting rows

In [ ]:

```
# Filter for rows where individuals is greater than 10000
ind_gt_10k = homelessness[homelessness["individuals"] > 10000]

# See the result
print(ind_gt_10k)
```

```
    Unnamed: 0              region       state  individuals  family_members  \
4            4             Pacific  California     109008.0         20964.0   
9            9      South Atlantic     Florida      21443.0          9587.0   
32          32        Mid-Atlantic    New York      39827.0         52070.0   
37          37             Pacific      Oregon      11139.0          3337.0   
43          43  West South Central       Texas      19199.0          6111.0   
47          47             Pacific  Washington      16424.0          5880.0   

    state_pop  
4    39461588  
9    21244317  
32   19530351  
37    4181886  
43   28628666  
47    7523869  
```

In [ ]:

```
# Filter for rows where region is Mountain
mountain_reg = homelessness[homelessness["region"] == "Mountain"]

# See the result
print(mountain_reg)
```

```
    Unnamed: 0    region       state  individuals  family_members  state_pop
2            2  Mountain     Arizona       7259.0          2606.0    7158024
5            5  Mountain    Colorado       7607.0          3250.0    5691287
12          12  Mountain       Idaho       1297.0           715.0    1750536
26          26  Mountain     Montana        983.0           422.0    1060665
28          28  Mountain      Nevada       7058.0           486.0    3027341
31          31  Mountain  New Mexico       1949.0           602.0    2092741
44          44  Mountain        Utah       1904.0           972.0    3153550
50          50  Mountain     Wyoming        434.0           205.0     577601
```

In [ ]:

```
# Filter for rows where family_members is less than 1000
# and region is Pacific
fam_lt_1k_pac = homelessness[(homelessness["family_members"] < 1000) & (homelessness["region"]=="Pacific")]

# See the result
print(fam_lt_1k_pac)
```

```
   Unnamed: 0   region   state  individuals  family_members  state_pop
1           1  Pacific  Alaska       1434.0           582.0     735139
```

Subsetting rows by categorical variables

In [ ]:

```
# Subset for rows in South Atlantic or Mid-Atlantic regions
south_mid_atlantic = homelessness[homelessness["region"].isin(["South Atlantic", "Mid-Atlantic"])]

# See the result
print(south_mid_atlantic)
```

```
    Unnamed: 0          region                 state  individuals  \
7            7  South Atlantic              Delaware        708.0   
8            8  South Atlantic  District of Columbia       3770.0   
9            9  South Atlantic               Florida      21443.0   
10          10  South Atlantic               Georgia       6943.0   
20          20  South Atlantic              Maryland       4914.0   
30          30    Mid-Atlantic            New Jersey       6048.0   
32          32    Mid-Atlantic              New York      39827.0   
33          33  South Atlantic        North Carolina       6451.0   
38          38    Mid-Atlantic          Pennsylvania       8163.0   
40          40  South Atlantic        South Carolina       3082.0   
46          46  South Atlantic              Virginia       3928.0   
48          48  South Atlantic         West Virginia       1021.0   

    family_members  state_pop  
7            374.0     965479  
8           3134.0     701547  
9           9587.0   21244317  
10          2556.0   10511131  
20          2230.0    6035802  
30          3350.0    8886025  
32         52070.0   19530351  
33          2817.0   10381615  
38          5349.0   12800922  
40           851.0    5084156  
46          2047.0    8501286  
48           222.0    1804291  
```

In [ ]:

```python
# The Mojave Desert states
canu = ["California", "Arizona", "Nevada", "Utah"]

# Filter for rows in the Mojave Desert states
mojave_homelessness = homelessness[homelessness["state"].isin(canu)]

# See the result
print(mojave_homelessness.head())
```

```
    Unnamed: 0    region       state  individuals  family_members  state_pop
2            2  Mountain     Arizona       7259.0          2606.0    7158024
4            4   Pacific  California     109008.0         20964.0   39461588
28          28  Mountain      Nevada       7058.0           486.0    3027341
44          44  Mountain        Utah       1904.0           972.0    3153550
```

Adding new columns

In [ ]:

```python
# Add total col as sum of individuals and family_members
homelessness["total"] = homelessness["individuals"] + homelessness["family_members"]
# Add p_individuals col as proportion of individuals
homelessness["p_individuals"] = homelessness["individuals"] / homelessness["total"]
```

```
# See the result
print(homelessness.head())
```

```
   Unnamed: 0              region       state  individuals  family_members  \
0           0  East South Central     Alabama       2570.0           864.0
1           1             Pacific      Alaska       1434.0           582.0
2           2            Mountain     Arizona       7259.0          2606.0
3           3  West South Central    Arkansas       2280.0           432.0
4           4             Pacific  California     109008.0         20964.0

   state_pop     total  p_individuals
0    4887681    3434.0       0.748398
1     735139    2016.0       0.711310
2    7158024    9865.0       0.735834
3    3009733    2712.0       0.840708
4   39461588  129972.0       0.838704
```

Combo-attack!

In [ ]:

```
# Create indiv_per_10k col as homeless individuals per 10k state pop
homelessness["indiv_per_10k"] = 10000 * homelessness["individuals"] / homelessness["state_pop"]

# Subset rows for indiv_per_10k greater than 20
high_homelessness = homelessness[homelessness["indiv_per_10k"] > 20]

# Sort high_homelessness by descending indiv_per_10k
high_homelessness_srt = high_homelessness.sort_values("indiv_per_10k", ascending=False)

# From high_homelessness_srt, select the state and indiv_per_10k cols
result = high_homelessness_srt[["state","indiv_per_10k"]]

# See the result
print(result)
```

```
                   state  indiv_per_10k
8   District of Columbia      53.738381
11                Hawaii      29.079406
4             California      27.623825
37                Oregon      26.636307
28                Nevada      23.314189
47            Washington      21.829195
32              New York      20.392363
```

otros codigos

In [ ]:

```
#example using group by
homelessness.groupby("family_members")["individuals"].mean()
print(homelessness.head())
```

```
   Unnamed: 0              region       state  individuals  family_members  \
0           0  East South Central     Alabama       2570.0           864.0
1           1             Pacific      Alaska       1434.0           582.0
2           2            Mountain     Arizona       7259.0          2606.0
3           3  West South Central    Arkansas       2280.0           432.0
4           4             Pacific  California     109008.0         20964.0

   state_pop     total  p_individuals  indiv_per_10k
0    4887681    3434.0       0.748398       5.258117
1     735139    2016.0       0.711310      19.506515
2    7158024    9865.0       0.735834      10.141067
3    3009733    2712.0       0.840708       7.575423
4   39461588  129972.0       0.838704      27.623825
```

Ch2 Aggregating DataFrames

Aggregating Data

In this chapter, you'll calculate summary statistics on DataFrame columns, and master grouped summary statistics and pivot tables.

In [ ]:

```
# Import pandas using the alias pd
import pandas as pd

sales = pd.read_csv('C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python Course
```

Mean and median

In [ ]:

```
# 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

In [ ]:

```
# 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

In [ ]:

```
# 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
```

Cumulative statistics

In [ ]:

```
import pandas as pd

# Sample data
data = {
    'store': [5, 1, 4, 9, 8, 7, 10, 3, 1, 6, 11, 2],
    'type': ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A'],
    'department': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    'date': ['2010-07-02', '2010-02-05', '2010-06-04', '2010-11-05', '2010-10-01', '2010-09-03', '2010-12-03', '2010-05-07', '2010
    'weekly_sales': [16333.14, 24924.50, 17558.09, 34238.88, 20094.19, 16241.78, 22517.56, 17413.94, 21827.90, 17508.41, 15984.24,
    'is_holiday': [False, False, False, False, False, False, False, False, False, False, False, False],
    'temperature_c': [27.172, 5.728, 27.050, 14.856, 22.161, 27.339, 9.594, 22.528, 8.056, 30.644, 9.039, 16.817],
    'fuel_price_usd_per_l': [0.705, 0.679, 0.715, 0.710, 0.688, 0.681, 0.715, 0.749, 0.693, 0.694, 0.786, 0.718],
    'unemployment': [7.787, 8.106, 7.808, 7.838, 7.838, 7.787, 7.838, 7.808, 8.106, 7.787, 7.742, 7.808]
}

# Create a Pandas DataFrame
sales_1_1 = pd.DataFrame(data)

# Display the DataFrame as a table
sales_1_1
```

Out[ ]:

	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	5	A	1	2010-07-02	16333.14	False	27.172	0.705	7.787
1	1	A	1	2010-02-05	24924.50	False	5.728	0.679	8.106
2	4	A	1	2010-06-04	17558.09	False	27.050	0.715	7.808
3	9	A	1	2010-11-05	34238.88	False	14.856	0.710	7.838
4	8	A	1	2010-10-01	20094.19	False	22.161	0.688	7.838
5	7	A	1	2010-09-03	16241.78	False	27.339	0.681	7.787
6	10	A	1	2010-12-03	22517.56	False	9.594	0.715	7.838
7	3	A	1	2010-05-07	17413.94	False	22.528	0.749	7.808
8	1	A	1	2010-03-05	21827.90	False	8.056	0.693	8.106
9	6	A	1	2010-08-06	17508.41	False	30.644	0.694	7.787
10	11	A	1	2011-01-07	15984.24	False	9.039	0.786	7.742
11	2	A	1	2010-04-02	57258.43	False	16.817	0.718	7.808

In [ ]:

```python
# Sort sales_1_1 by date
sales_1_1 = sales_1_1.sort_values(by='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
1   2010-02-05      24924.50          24924.50       24924.50
8   2010-03-05      21827.90          46752.40       24924.50
11  2010-04-02      57258.43         104010.83       57258.43
7   2010-05-07      17413.94         121424.77       57258.43
2   2010-06-04      17558.09         138982.86       57258.43
0   2010-07-02      16333.14         155316.00       57258.43
9   2010-08-06      17508.41         172824.41       57258.43
5   2010-09-03      16241.78         189066.19       57258.43
4   2010-10-01      20094.19         209160.38       57258.43
3   2010-11-05      34238.88         243399.26       57258.43
6   2010-12-03      22517.56         265916.82       57258.43
10  2011-01-07      15984.24         281901.06       57258.43
```

Dropping duplicates

In [ ]:

```
# 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"]].drop_duplicates("date")

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

```
      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  
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   

      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  
```

Counting categorical variables

In [ ]:

```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="department", ascending=False)
print(dept_counts_sorted)

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

```
type
A    11
B     1
Name: count, dtype: int64
type
A    0.916667
B    0.083333
Name: proportion, dtype: float64
department
1     12
55    12
72    12
71    12
67    12
      ..
37    10
48     8
50     6
39     4
43     2
Name: count, Length: 80, dtype: int64
department
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: proportion, Length: 80, dtype: float64
```

What percent of sales occurred at each store type?

In [ ]:

```
# 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()

In [ ]:

```
# 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)

# 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
A    0.909775
B    0.090225
Name: weekly_sales, dtype: float64
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

In [ ]:

```
# 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([min, max, np.mean, np.median])

# Print sales_stats
print(sales_stats)
```

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

# Print unemp_fuel_stats
print(unemp_fuel_stats)
```

```
         min        max          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            \
              min    max      mean median                  min       max
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
```

```
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:5: FutureWarning: The provided callable <built-in function min> i
s currently using SeriesGroupBy.min. In a future version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "min" instead.
  sales_stats = sales.groupby("type")["weekly_sales"].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:5: FutureWarning: The provided callable <built-in function max> i
s currently using SeriesGroupBy.max. In a future version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "max" instead.
  sales_stats = sales.groupby("type")["weekly_sales"].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:5: FutureWarning: The provided callable <function mean at 0x00000
1EC3C73B100> is currently using SeriesGroupBy.mean. In a future version of pandas, the provided callable will be used directly. T
o keep current behavior pass the string "mean" instead.
  sales_stats = sales.groupby("type")["weekly_sales"].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:5: FutureWarning: The provided callable <function median at 0x000
001EC3C862340> is currently using SeriesGroupBy.median. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "median" instead.
  sales_stats = sales.groupby("type")["weekly_sales"].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:11: FutureWarning: The provided callable <built-in function min>
is currently using SeriesGroupBy.min. In a future version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "min" instead.
  unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:11: FutureWarning: The provided callable <built-in function max>
is currently using SeriesGroupBy.max. In a future version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "max" instead.
  unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:11: FutureWarning: The provided callable <function mean at 0x0000
01EC3C73B100> is currently using SeriesGroupBy.mean. In a future version of pandas, the provided callable will be used directly.
To keep current behavior pass the string "mean" instead.
  unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:11: FutureWarning: The provided callable <function median at 0x00
0001EC3C862340> is currently using SeriesGroupBy.median. In a future version of pandas, the provided callable will be used direct
ly. To keep current behavior pass the string "median" instead.
  unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\1976951964.py:11: FutureWarning: The provided callable <built-in function min>
is currently using SeriesGroupBy.min. In a future version of pandas, the provided callable will be used directly. To keep current
behavior pass the string "min" instead.
  unemp_fuel_stats = sales.groupby("type")[["unemployment", "fuel_price_usd_per_l"]].agg([min, max, np.mean, np.median])
```

Pivoting on one variable

```
In [ ]:
# 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
```

Fill in missing values and sum values with pivot tables

In [ ]:

```
# Print mean weekly_sales by department and type; fill missing values with 0
import numpy as np
print(sales.pivot_table(values="weekly_sales", index="department", columns="type", aggfunc=np.mean, fill_value=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]
```

```
C:\Users\yeiso\AppData\Local\Temp\ipykernel_33652\321967142.py:3: FutureWarning: The provided callable <function mean at 0x000001
EC3C73B100> is currently using DataFrameGroupBy.mean. In a future version of pandas, the provided callable will be used directly.
To keep current behavior pass the string "mean" instead.
  print(sales.pivot_table(values="weekly_sales", index="department", columns="type", aggfunc=np.mean, fill_value=0))
```

Chapter 3 - Slicing and Indexing DataFrames

Indexes are supercharged row and column names. Learn how they can be combined with slicing for powerful DataFrame subsetting.

In [ ]:

```
# Import pandas using the alias pd
import pandas as pd

temperatures = pd.read_csv('C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python
```

Setting and removing indexes

In [ ]:

```
# Look at temperatures
print(temperatures.head())

# Index temperatures by city
temperatures_ind = temperatures.set_index("city")

# Look at temperatures_ind
print(temperatures_ind.head())
```

```python
# Reset the index, keeping its contents
print(temperatures_ind.reset_index())

# Reset the index, dropping its contents
print(temperatures_ind.reset_index(drop=True))
```

```
   Unnamed: 0        date     city        country  avg_temp_c
0           0  2000-01-01  Abidjan  Côte D'Ivoire      27.293
1           1  2000-02-01  Abidjan  Côte D'Ivoire      27.685
2           2  2000-03-01  Abidjan  Côte D'Ivoire      29.061
3           3  2000-04-01  Abidjan  Côte D'Ivoire      28.162
4           4  2000-05-01  Abidjan  Côte D'Ivoire      27.547
         Unnamed: 0        date        country  avg_temp_c
city
Abidjan           0  2000-01-01  Côte D'Ivoire      27.293
Abidjan           1  2000-02-01  Côte D'Ivoire      27.685
Abidjan           2  2000-03-01  Côte D'Ivoire      29.061
Abidjan           3  2000-04-01  Côte D'Ivoire      28.162
Abidjan           4  2000-05-01  Côte D'Ivoire      27.547
          city  Unnamed: 0        date        country  avg_temp_c
0      Abidjan           0  2000-01-01  Côte D'Ivoire      27.293
1      Abidjan           1  2000-02-01  Côte D'Ivoire      27.685
2      Abidjan           2  2000-03-01  Côte D'Ivoire      29.061
3      Abidjan           3  2000-04-01  Côte D'Ivoire      28.162
4      Abidjan           4  2000-05-01  Côte D'Ivoire      27.547
...        ...         ...         ...            ...         ...
16495     Xian       16495  2013-05-01          China      18.979
16496     Xian       16496  2013-06-01          China      23.522
16497     Xian       16497  2013-07-01          China      25.251
16498     Xian       16498  2013-08-01          China      24.528
16499     Xian       16499  2013-09-01          China         NaN

[16500 rows x 5 columns]
       Unnamed: 0        date        country  avg_temp_c
0               0  2000-01-01  Côte D'Ivoire      27.293
1               1  2000-02-01  Côte D'Ivoire      27.685
2               2  2000-03-01  Côte D'Ivoire      29.061
3               3  2000-04-01  Côte D'Ivoire      28.162
4               4  2000-05-01  Côte D'Ivoire      27.547
...           ...         ...            ...         ...
16495       16495  2013-05-01          China      18.979
16496       16496  2013-06-01          China      23.522
16497       16497  2013-07-01          China      25.251
16498       16498  2013-08-01          China      24.528
16499       16499  2013-09-01          China         NaN

[16500 rows x 4 columns]
```

Subsetting with .loc[]

In [ ]:

```
# Make a list of cities to subset on
cities = ["Moscow", "Saint Petersburg"]

# Subset temperatures using square brackets
print(temperatures[temperatures["city"].isin(cities)])

# Subset temperatures_ind using .loc[]
print(temperatures_ind.loc[cities])
```

```
       Unnamed: 0        date              city country  avg_temp_c
10725       10725  2000-01-01            Moscow  Russia      -7.313
10726       10726  2000-02-01            Moscow  Russia      -3.551
10727       10727  2000-03-01            Moscow  Russia      -1.661
10728       10728  2000-04-01            Moscow  Russia      10.096
10729       10729  2000-05-01            Moscow  Russia      10.357
...           ...         ...               ...     ...         ...
13360       13360  2013-05-01  Saint Petersburg  Russia      12.355
13361       13361  2013-06-01  Saint Petersburg  Russia      17.185
13362       13362  2013-07-01  Saint Petersburg  Russia      17.234
13363       13363  2013-08-01  Saint Petersburg  Russia      17.153
13364       13364  2013-09-01  Saint Petersburg  Russia         NaN

[330 rows x 5 columns]
                  Unnamed: 0        date country  avg_temp_c
city
Moscow                 10725  2000-01-01  Russia      -7.313
Moscow                 10726  2000-02-01  Russia      -3.551
Moscow                 10727  2000-03-01  Russia      -1.661
Moscow                 10728  2000-04-01  Russia      10.096
Moscow                 10729  2000-05-01  Russia      10.357
...                      ...         ...     ...         ...
Saint Petersburg       13360  2013-05-01  Russia      12.355
Saint Petersburg       13361  2013-06-01  Russia      17.185
Saint Petersburg       13362  2013-07-01  Russia      17.234
Saint Petersburg       13363  2013-08-01  Russia      17.153
Saint Petersburg       13364  2013-09-01  Russia         NaN

[330 rows x 4 columns]
```

Setting multi-level indexes

In [ ]:

```
# Index temperatures by country & city
temperatures_ind = temperatures.set_index(["country", "city"])

# List of tuples: Brazil, Rio De Janeiro & Pakistan, Lahore
rows_to_keep = [("Brazil", "Rio De Janeiro"), ("Pakistan", "Lahore")]
```

```
# Subset for rows to keep
print(temperatures_ind.loc[rows_to_keep])
```

```
                         Unnamed: 0        date  avg_temp_c
country  city                                              
Brazil   Rio De Janeiro       12540  2000-01-01      25.974
         Rio De Janeiro       12541  2000-02-01      26.699
         Rio De Janeiro       12542  2000-03-01      26.270
         Rio De Janeiro       12543  2000-04-01      25.750
         Rio De Janeiro       12544  2000-05-01      24.356
...                             ...         ...         ...
Pakistan Lahore                8575  2013-05-01      33.457
         Lahore                8576  2013-06-01      34.456
         Lahore                8577  2013-07-01      33.279
         Lahore                8578  2013-08-01      31.511
         Lahore                8579  2013-09-01         NaN

[330 rows x 3 columns]
```

Sorting by index values

In [ ]:

```
# Sort temperatures_ind by index values
print(temperatures_ind.sort_index())

# Sort temperatures_ind by index values at the city level
print(temperatures_ind.sort_index(level=["city","country"]))

# Sort temperatures_ind by country then descending city
print(temperatures_ind.sort_index(level=["country", "city"], ascending=[True, False]))
```

```
                    Unnamed: 0        date  avg_temp_c
country     city                                      
Afghanistan Kabul         7260  2000-01-01       3.326
            Kabul         7261  2000-02-01       3.454
            Kabul         7262  2000-03-01       9.612
            Kabul         7263  2000-04-01      17.925
            Kabul         7264  2000-05-01      24.658
...                        ...         ...         ...
Zimbabwe    Harare        5605  2013-05-01      18.298
            Harare        5606  2013-06-01      17.020
            Harare        5607  2013-07-01      16.299
            Harare        5608  2013-08-01      19.232
            Harare        5609  2013-09-01         NaN

[16500 rows x 3 columns]
                       Unnamed: 0        date  avg_temp_c
country       city                                       
Côte D'Ivoire Abidjan           0  2000-01-01      27.293
              Abidjan           1  2000-02-01      27.685
              Abidjan           2  2000-03-01      29.061
              Abidjan           3  2000-04-01      28.162
              Abidjan           4  2000-05-01      27.547
...                           ...         ...         ...
China         Xian          16495  2013-05-01      18.979
              Xian          16496  2013-06-01      23.522
              Xian          16497  2013-07-01      25.251
              Xian          16498  2013-08-01      24.528
              Xian          16499  2013-09-01         NaN

[16500 rows x 3 columns]
                    Unnamed: 0        date  avg_temp_c
country     city                                      
Afghanistan Kabul         7260  2000-01-01       3.326
            Kabul         7261  2000-02-01       3.454
            Kabul         7262  2000-03-01       9.612
            Kabul         7263  2000-04-01      17.925
            Kabul         7264  2000-05-01      24.658
...                        ...         ...         ...
Zimbabwe    Harare        5605  2013-05-01      18.298
            Harare        5606  2013-06-01      17.020
            Harare        5607  2013-07-01      16.299
            Harare        5608  2013-08-01      19.232
            Harare        5609  2013-09-01         NaN

[16500 rows x 3 columns]
```

Slicing index values

In [ ]:

```
# Sort the index of temperatures_ind
temperatures_srt = temperatures_ind.sort_index()

# Subset rows from Pakistan to Russia
print(temperatures_srt.loc["Pakistan":"Russia"])

# Try to subset rows from Lahore to Moscow
print(temperatures_srt.loc["Lahore":"Moscow"])

# Subset rows from Pakistan, Lahore to Russia, Moscow
print(temperatures_srt.loc[("Pakistan","Lahore"):("Russia","Moscow")])
```

```
                           Unnamed: 0        date  avg_temp_c
country  city
Pakistan Faisalabad              4785  2000-01-01      12.792
         Faisalabad              4786  2000-02-01      14.339
         Faisalabad              4787  2000-03-01      20.309
         Faisalabad              4788  2000-04-01      29.072
         Faisalabad              4789  2000-05-01      34.845
...                               ...         ...         ...
Russia   Saint Petersburg       13360  2013-05-01      12.355
         Saint Petersburg       13361  2013-06-01      17.185
         Saint Petersburg       13362  2013-07-01      17.234
         Saint Petersburg       13363  2013-08-01      17.153
         Saint Petersburg       13364  2013-09-01         NaN

[1155 rows x 3 columns]
                    Unnamed: 0        date  avg_temp_c
country city
Mexico  Mexico           10230  2000-01-01      12.694
        Mexico           10231  2000-02-01      14.677
        Mexico           10232  2000-03-01      17.376
        Mexico           10233  2000-04-01      18.294
        Mexico           10234  2000-05-01      18.562
...                        ...         ...         ...
Morocco Casablanca        3130  2013-05-01      19.217
        Casablanca        3131  2013-06-01      23.649
        Casablanca        3132  2013-07-01      27.488
        Casablanca        3133  2013-08-01      27.952
        Casablanca        3134  2013-09-01         NaN

[330 rows x 3 columns]
                 Unnamed: 0        date  avg_temp_c
country  city
Pakistan Lahore        8415  2000-01-01      12.792
         Lahore        8416  2000-02-01      14.339
         Lahore        8417  2000-03-01      20.309
         Lahore        8418  2000-04-01      29.072
         Lahore        8419  2000-05-01      34.845
...                     ...         ...         ...
Russia   Moscow       10885  2013-05-01      16.152
         Moscow       10886  2013-06-01      18.718
         Moscow       10887  2013-07-01      18.136
         Moscow       10888  2013-08-01      17.485
         Moscow       10889  2013-09-01         NaN

[660 rows x 3 columns]
```

Slicing in both directions

In [ ]:

```
# Subset rows from India, Hyderabad to Iraq, Baghdad
print(temperatures_srt.loc[("India","Hyderabad"):("Iraq","Baghdad")])

# Subset columns from date to avg_temp_c
print(temperatures_srt.loc[:,"date":"avg_temp_c"])

# Subset in both directions at once
print(temperatures_srt.loc[("India","Hyderabad"):("Iraq","Baghdad"),"date":"avg_temp_c"])
```

```
                   Unnamed: 0        date  avg_temp_c
country city
India   Hyderabad        5940  2000-01-01      23.779
        Hyderabad        5941  2000-02-01      25.826
        Hyderabad        5942  2000-03-01      28.821
        Hyderabad        5943  2000-04-01      32.698
        Hyderabad        5944  2000-05-01      32.438
...                       ...         ...         ...
Iraq    Baghdad          1150  2013-05-01      28.673
        Baghdad          1151  2013-06-01      33.803
        Baghdad          1152  2013-07-01      36.392
        Baghdad          1153  2013-08-01      35.463
        Baghdad          1154  2013-09-01         NaN

[2145 rows x 3 columns]
                          date  avg_temp_c
country     city
Afghanistan Kabul   2000-01-01       3.326
            Kabul   2000-02-01       3.454
            Kabul   2000-03-01       9.612
            Kabul   2000-04-01      17.925
            Kabul   2000-05-01      24.658
...                        ...         ...
Zimbabwe    Harare  2013-05-01      18.298
            Harare  2013-06-01      17.020
            Harare  2013-07-01      16.299
            Harare  2013-08-01      19.232
            Harare  2013-09-01         NaN

[16500 rows x 2 columns]
                         date  avg_temp_c
country city
India   Hyderabad  2000-01-01      23.779
        Hyderabad  2000-02-01      25.826
        Hyderabad  2000-03-01      28.821
        Hyderabad  2000-04-01      32.698
        Hyderabad  2000-05-01      32.438
...                       ...         ...
Iraq    Baghdad    2013-05-01      28.673
        Baghdad    2013-06-01      33.803
        Baghdad    2013-07-01      36.392
        Baghdad    2013-08-01      35.463
        Baghdad    2013-09-01         NaN

[2145 rows x 2 columns]
```

Slicing time series

In [ ]:

```
# Use Boolean conditions to subset temperatures for rows in 2010 and 2011
temperatures_bool = temperatures[(temperatures["date"] >= "2010-01-01") & (temperatures["date"] <= "2011-12-31")]
print(temperatures_bool)

# Set date as an index and sort the index
temperatures_ind = temperatures.set_index("date").sort_index()

# Use .loc[] to subset temperatures_ind for rows in 2010 and 2011
print(temperatures_ind.loc["2010":"2011"])

# Use .loc[] to subset temperatures_ind for rows from Aug 2010 to Feb 2011
print(temperatures_ind.loc["2010-08":"2011-02"])
```

```
       Unnamed: 0        date     city        country  avg_temp_c
120           120  2010-01-01  Abidjan  Côte D'Ivoire      28.270
121           121  2010-02-01  Abidjan  Côte D'Ivoire      29.262
122           122  2010-03-01  Abidjan  Côte D'Ivoire      29.596
123           123  2010-04-01  Abidjan  Côte D'Ivoire      29.068
124           124  2010-05-01  Abidjan  Côte D'Ivoire      28.258
...           ...         ...      ...            ...         ...
16474       16474  2011-08-01     Xian          China      23.069
16475       16475  2011-09-01     Xian          China      16.775
16476       16476  2011-10-01     Xian          China      12.587
16477       16477  2011-11-01     Xian          China       7.543
16478       16478  2011-12-01     Xian          China      -0.490

[2400 rows x 5 columns]
            Unnamed: 0       city    country  avg_temp_c
date
2010-01-01        4905  Faisalabad   Pakistan      11.810
2010-01-01       10185  Melbourne  Australia      20.016
2010-01-01        3750  Chongqing      China       7.921
2010-01-01       13155  São Paulo     Brazil      23.738
2010-01-01        5400  Guangzhou      China      14.136
...                ...        ...        ...         ...
2010-12-01        6896    Jakarta  Indonesia      26.602
2010-12-01        5246      Gizeh      Egypt      16.530
2010-12-01       11186     Nagpur      India      19.120
2010-12-01       14981     Sydney  Australia      19.559
2010-12-01       13496   Salvador     Brazil      26.265

[1200 rows x 4 columns]
            Unnamed: 0           city        country  avg_temp_c
date
2010-08-01        2602       Calcutta          India      30.226
2010-08-01       12337           Pune          India      24.941
2010-08-01        6562          Izmir         Turkey      28.352
2010-08-01       15637        Tianjin          China      25.543
2010-08-01        9862         Manila    Philippines      27.101
...                ...            ...            ...         ...
2011-01-01        4257  Dar Es Salaam       Tanzania      28.541
2011-01-01       11352        Nairobi          Kenya      17.768
2011-01-01         297    Addis Abeba       Ethiopia      17.708
2011-01-01       11517        Nanjing          China       0.144
2011-01-01       11847       New York  United States      -4.463

[600 rows x 4 columns]
```

Subsetting by row/column number

In [ ]:

```
# Get 23rd row, 2nd column (index 22, 1)
print(temperatures.iloc[22,1])
```

```
# Use slicing to get the first 5 rows
print(temperatures.iloc[:5])

# Use slicing to get columns 3 to 4
print(temperatures.iloc[:,2:4])

# Use slicing in both directions at once
print(temperatures.iloc[:5,2:4])
```

```
2001-11-01
   Unnamed: 0        date     city        country  avg_temp_c
0           0  2000-01-01  Abidjan  Côte D'Ivoire      27.293
1           1  2000-02-01  Abidjan  Côte D'Ivoire      27.685
2           2  2000-03-01  Abidjan  Côte D'Ivoire      29.061
3           3  2000-04-01  Abidjan  Côte D'Ivoire      28.162
4           4  2000-05-01  Abidjan  Côte D'Ivoire      27.547
          city        country
0      Abidjan  Côte D'Ivoire
1      Abidjan  Côte D'Ivoire
2      Abidjan  Côte D'Ivoire
3      Abidjan  Côte D'Ivoire
4      Abidjan  Côte D'Ivoire
...        ...            ...
16495     Xian          China
16496     Xian          China
16497     Xian          China
16498     Xian          China
16499     Xian          China

[16500 rows x 2 columns]
      city        country
0  Abidjan  Côte D'Ivoire
1  Abidjan  Côte D'Ivoire
2  Abidjan  Côte D'Ivoire
3  Abidjan  Côte D'Ivoire
4  Abidjan  Côte D'Ivoire
```

Pivot temperature by city and year

In [ ]:

```
import pandas as pd

# Assuming you have a DataFrame named temperatures

# Convert "date" column to datetime format
temperatures["date"] = pd.to_datetime(temperatures["date"])

# Add a year column to temperatures
temperatures["year"] = temperatures["date"].dt.year
```

```python
# Pivot avg_temp_c by country and city vs year
temp_by_country_city_vs_year = temperatures.pivot_table(values="avg_temp_c", index=["country", "city"], columns="year")

# See the result
print(temp_by_country_city_vs_year)
```

```
year                                2000       2001       2002       2003  \
country       city
Afghanistan   Kabul            15.822667  15.847917  15.714583  15.132583
Angola        Luanda           24.410333  24.427083  24.790917  24.867167
Australia     Melbourne        14.320083  14.180000  14.075833  13.985583
              Sydney           17.567417  17.854500  17.733833  17.592333
Bangladesh    Dhaka            25.905250  25.931250  26.095000  25.927417
...                                  ...        ...        ...        ...
United States Chicago          11.089667  11.703083  11.532083  10.481583
              Los Angeles      16.643333  16.466250  16.430250  16.944667
              New York          9.969083  10.931000  11.252167   9.836000
Vietnam       Ho Chi Minh City 27.588917  27.831750  28.064750  27.827667
Zimbabwe      Harare           20.283667  20.861000  21.079333  20.889167

year                                2004       2005       2006       2007  \
country       city
Afghanistan   Kabul            16.128417  14.847500  15.798500  15.518000
Angola        Luanda           24.216167  24.414583  24.138417  24.241583
Australia     Melbourne        13.742083  14.378500  13.991083  14.991833
              Sydney           17.869667  18.028083  17.749500  18.020833
Bangladesh    Dhaka            26.136083  26.193333  26.440417  25.951333
...                                  ...        ...        ...        ...
United States Chicago          10.943417  11.583833  11.870500  11.448333
              Los Angeles      16.552833  16.431417  16.623083  16.699917
              New York         10.389500  10.681417  11.519250  10.627333
Vietnam       Ho Chi Minh City 27.686583  27.884000  28.044000  27.866667
Zimbabwe      Harare           20.307667  21.487417  20.699750  20.746250

year                                2008       2009       2010       2011  \
country       city
Afghanistan   Kabul            15.479250  15.093333  15.676000  15.812167
Angola        Luanda           24.266333  24.325083  24.440250  24.150750
Australia     Melbourne        14.110583  14.647417  14.231667  14.190917
              Sydney           17.321083  18.175833  17.999000  17.713333
Bangladesh    Dhaka            26.004500  26.535583  26.648167  25.803250
...                                  ...        ...        ...        ...
United States Chicago          10.242417  10.298333  11.815917  11.214250
              Los Angeles      17.014750  16.677000  15.887000  15.874833
              New York         10.641667  10.141833  11.357583  11.272250
Vietnam       Ho Chi Minh City 27.611417  27.853333  28.281750  27.675417
Zimbabwe      Harare           20.680500  20.523833  21.165833  20.781750

year                                2012       2013
country       city
Afghanistan   Kabul            14.510333  16.206125
Angola        Luanda           24.240083  24.553875
Australia     Melbourne        14.268667  14.741500
              Sydney           17.474333  18.089750
Bangladesh    Dhaka            26.283583  26.587000
...                                  ...        ...
```

```
United States Chicago         12.821250  11.586889
              Los Angeles     17.089583  18.120667
              New York        11.971500  12.163889
Vietnam       Ho Chi Minh City 28.248750  28.455000
Zimbabwe      Harare          20.523333  19.756500

[100 rows x 14 columns]
```

Subsetting pivot tables

In [ ]:

```python
# Subset for Egypt to India
temp_by_country_city_vs_year.loc["Egypt":"India"]

# Subset for Egypt, Cairo to India, Delhi
temp_by_country_city_vs_year.loc[("Egypt","Cairo"):("India","Delhi")]

# Subset in both directions at once
temp_by_country_city_vs_year.loc[("Egypt","Cairo"):("India","Delhi"),"2005":"2010"]
```

Out[ ]:

	year	2005	2006	2007	2008	2009	2010
country	**city**						
Egypt	**Cairo**	22.006500	22.050000	22.361000	22.644500	22.625000	23.718250
	Gizeh	22.006500	22.050000	22.361000	22.644500	22.625000	23.718250
Ethiopia	**Addis Abeba**	18.312833	18.427083	18.142583	18.165000	18.765333	18.298250
France	**Paris**	11.552917	11.788500	11.750833	11.278250	11.464083	10.409833
Germany	**Berlin**	9.919083	10.545333	10.883167	10.657750	10.062500	8.606833
India	**Ahmadabad**	26.828083	27.282833	27.511167	27.048500	28.095833	28.017833
	Bangalore	25.476500	25.418250	25.464333	25.352583	25.725750	25.705250
	Bombay	27.035750	27.381500	27.634667	27.177750	27.844500	27.765417
	Calcutta	26.729167	26.986250	26.584583	26.522333	27.153250	27.288833
	Delhi	25.716083	26.365917	26.145667	25.675000	26.554250	26.520250

Calculating on a pivot table

In [ ]:

```python
# Get the worldwide mean temp by year
mean_temp_by_year = temp_by_country_city_vs_year.mean(axis="index")

# Filter for the year that had the highest mean temp
print(mean_temp_by_year[mean_temp_by_year == max(mean_temp_by_year)])
```

```
# Get the mean temp by city
mean_temp_by_city = temp_by_country_city_vs_year.mean(axis="columns")

# Filter for the city that had the lowest mean temp
print(mean_temp_by_city[mean_temp_by_city == min(mean_temp_by_city)])
```

```
year
2013    20.312285
dtype: float64
country  city
China    Harbin    4.876551
dtype: float64
```

hcapter 4 - Creating and Visualizing DataFrames

Learn to visualize the contents of your DataFrames, handle missing data values, and import data from and export data to CSV files.

In [ ]:

```
# Import pandas using the alias pd
import pandas as pd

avocados = pd.read_pickle('C:\\Users\\yeiso\\OneDrive - Douglas College\\0. DOUGLAS COLLEGE\\3. Fund Machine Learning\\0. Python C
print(avocados.head(3))
```

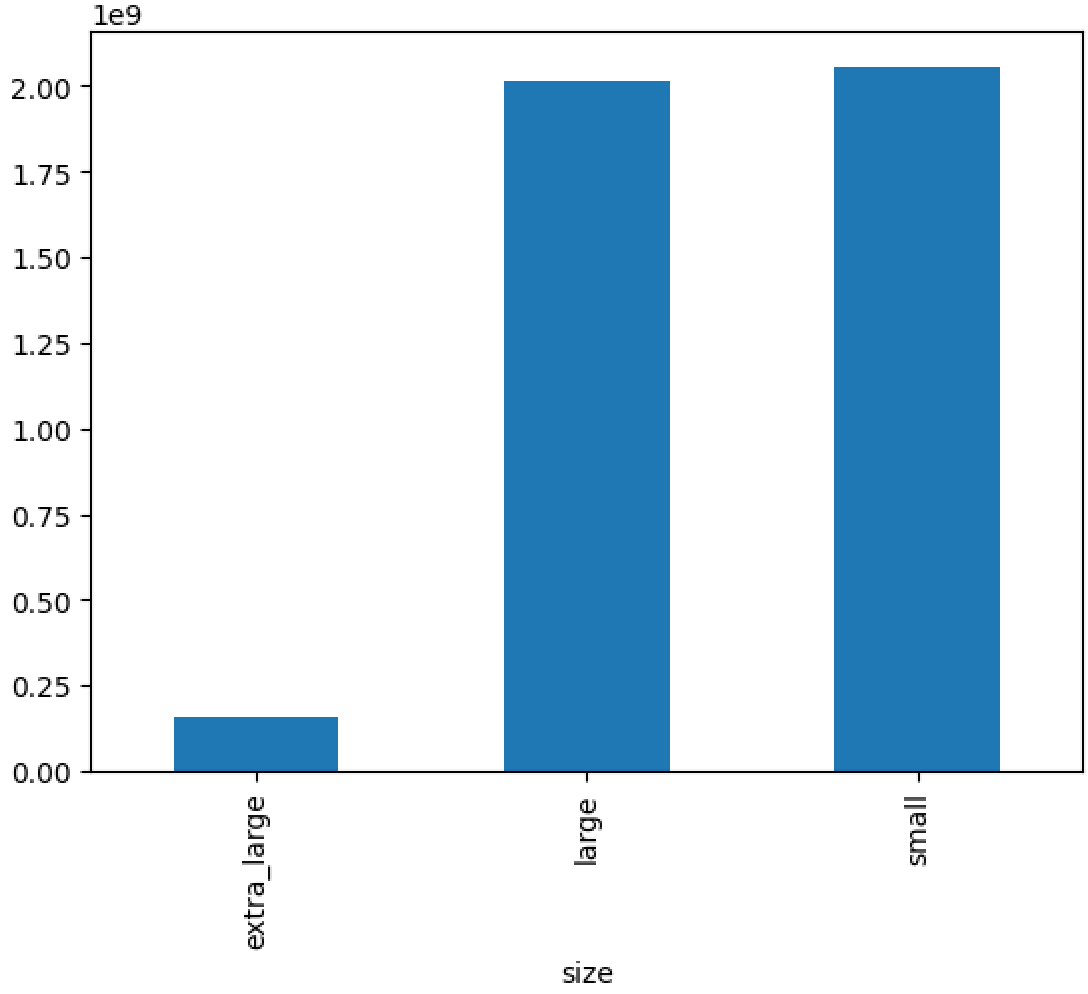
```
         date          type  year  avg_price   size     nb_sold
0  2015-12-27  conventional  2015       0.95  small  9626901.09
1  2015-12-20  conventional  2015       0.98  small  8710021.76
2  2015-12-13  conventional  2015       0.93  small  9855053.66
```

Which avocado size is most popular?

In [ ]:

```
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Look at the first few rows of data
print(avocados.head())

# Get the total number of avocados sold of each size
nb_sold_by_size = avocados.groupby("size")["nb_sold"].sum()

# Create a bar plot of the number of avocados sold by size
nb_sold_by_size.plot(kind="bar")

# Show the plot
plt.show()
```

```
         date          type  year  avg_price   size     nb_sold
0  2015-12-27  conventional  2015       0.95  small  9626901.09
1  2015-12-20  conventional  2015       0.98  small  8710021.76
2  2015-12-13  conventional  2015       0.93  small  9855053.66
3  2015-12-06  conventional  2015       0.89  small  9405464.36
4  2015-11-29  conventional  2015       0.99  small  8094803.56
```

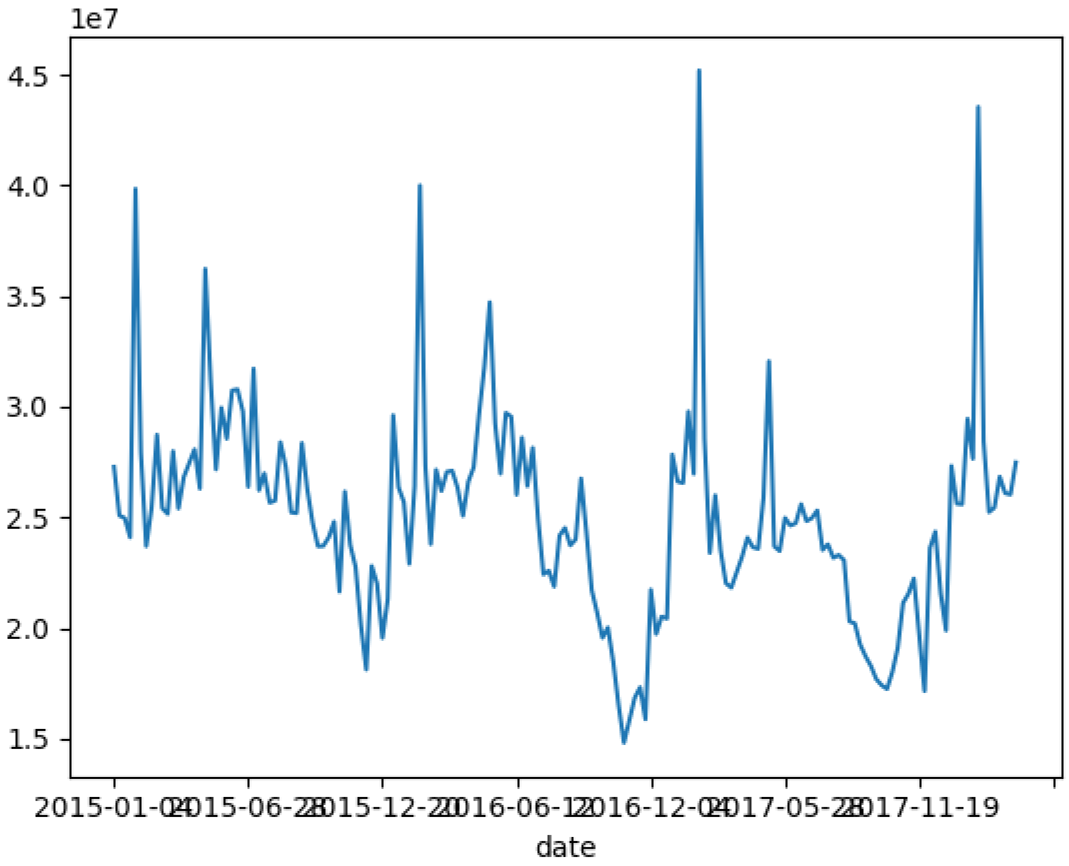


Changes in sales over time

In [ ]:

```python
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Get the total number of avocados sold on each date
nb_sold_by_date = avocados.groupby("date")["nb_sold"].sum()

# Create a line plot of the number of avocados sold by date
nb_sold_by_date.plot(kind="line")
```

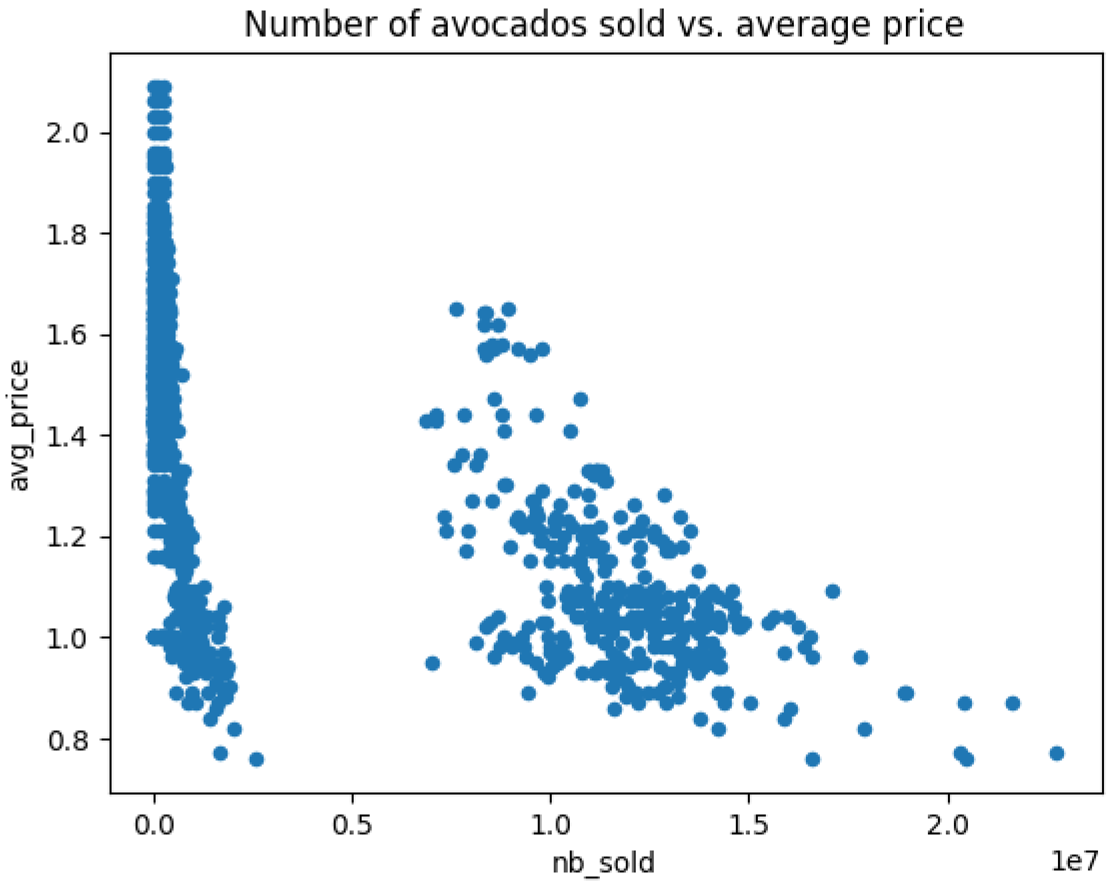
```
# Show the plot
plt.show()
```



Avocado supply and demand

In [ ]:

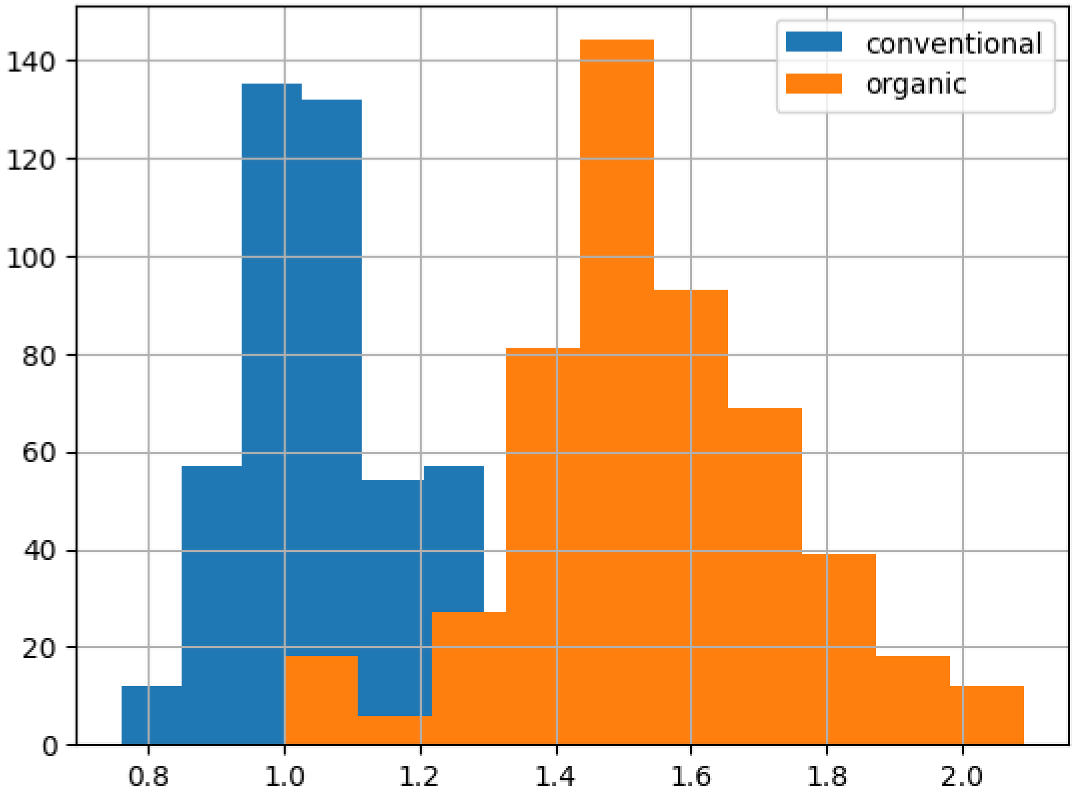
```
# Scatter plot of nb_sold vs avg_price with title
avocados.plot(x="nb_sold", y="avg_price", kind="scatter", title="Number of avocados sold vs. average price")

# Show the plot
plt.show()
```



Price of conventional vs. organic avocados

In [ ]:

```
# Histogram of conventional avg_price
avocados[avocados["type"] == "conventional"]["avg_price"].hist()

# Histogram of organic avg_price
avocados[avocados["type"] == "organic"]["avg_price"].hist()

# Add a legend
plt.legend(["conventional", "organic"])

# Show the plot
plt.show()
```



Finding missing values

In [ ]:

```python
# Import matplotlib.pyplot with alias plt
import matplotlib.pyplot as plt

# Check individual values for missing values
print(avocados_2016.isna())

# Check each column for missing values
print(avocados_2016.isna().any())

# Bar plot of missing values by variable
avocados_2016.isna().sum().plot(kind="bar")

# Show plot
plt.show()
```

Removing missing values

In [ ]:

```python
# Remove rows with missing values
avocados_complete = avocados_2016.dropna()
```

```
# Check if any columns contain missing values
print(avocados_complete.isna().any())
```

Replacing missing values

In [ ]:

```
# List the columns with missing values
cols_with_missing = ["small_sold", "large_sold", "xl_sold"]

# Create histograms showing the distributions cols_with_missing
avocados_2016[cols_with_missing].plot(kind="hist")

# Fill in missing values with 0
avocados_filled = avocados_2016.fillna(0)

# Create histograms of the filled columns
avocados_filled[cols_with_missing].hist()

# Show the plot
plt.show()
```

List of dictionaries

In [ ]:

```
# Create a list of dictionaries with new data
avocados_list = [
    {"date": "2019-11-03", "small_sold": 10376832, "large_sold": 7835071},
    {"date": "2019-11-10", "small_sold": 10717154, "large_sold": 8561348},
]

# Convert list into DataFrame
avocados_2019 = pd.DataFrame(avocados_list)

# Print the new DataFrame
print(avocados_2019)
```

```
         date  small_sold  large_sold
0  2019-11-03    10376832     7835071
1  2019-11-10    10717154     8561348
```

Dictionary of lists

In [ ]:

```
# Create a dictionary of lists with new data
avocados_dict = {
  "date": ["2019-11-17","2019-12-01"],
  "small_sold": [10859987,9291631],
  "large_sold": [7674135,6238096]
}

# Convert dictionary into DataFrame
```

```
avocados_2019 = pd.DataFrame(avocados_dict)

# Print the new DataFrame
print(avocados_2019)
```

```
         date  small_sold  large_sold
0  2019-11-17    10859987     7674135
1  2019-12-01     9291631     6238096
```

CSV to DataFrame

In [ ]:

```
# Read CSV as DataFrame called airline_bumping
airline_bumping = pd.read_csv("airline_bumping.csv")

# Take a look at the DataFrame
print(airline_bumping.head())

# For each airline, select nb_bumped and total_passengers and sum
airline_totals = airline_bumping.groupby("airline")[["nb_bumped","total_passengers"]].sum()

# Create new col, bumps_per_10k: no. of bumps per 10k passengers for each airline
airline_totals["bumps_per_10k"] = airline_totals["nb_bumped"] / airline_totals["total_passengers"] * 10000

# Print airline_totals
print(airline_totals)
```

DataFrame to CSV

In [ ]:

```
# Create airline_totals_sorted
airline_totals_sorted = airline_totals.sort_values("bumps_per_10k", ascending=False)

# Print airline_totals_sorted
print(airline_totals_sorted)

# Save as airline_totals_sorted.csv
airline_totals_sorted.to_csv("airline_totals_sorted.csv")
```