|  |
| --- |
| **W3 – Data Manipulation** |

Save your W3 notebook with the following naming conventions.

ID\_Name\_SecNo\_W3.ipynb,

for example

**6113333\_JohnWick\_541\_W3.ipynb**

**What is Pickle? (.p or .pkl)**

Pickle is a serialized way of storing a Pandas dataframe. Basically, you are writing down the exact representation of the dataframe to disk. This means the types of the columns are and the indices are the same. Download taxi\_owners.p and load the data using pd.read\_pickle()

taxi\_owner = pd.read\_pickle('taxi\_owners.p')

taxi\_owner.head()

A screenshot of a computer

Description automatically generated with low confidence

Today we will be working with Pickle files as well. Back to that soon.

There are several ways to store data for analysis, but rectangular data, sometimes called "tabular data" is the most common form. In this example, with dogs, each observation, or each dog, is a row, and each variable, or each dog property, is a column.

A picture containing text, screenshot, font, number

Description automatically generated

Exploring a DataFrame can be done by many methods such as .head(), .info(). The .info() method displays the names of columns, the data types they contain, and whether they have any missing values.

|  |  |
| --- | --- |
| df.describe() | The describe method computes some summary statistics for numerical columns, like mean and median. "count" is the number of non-missing values in each column. describe is good for a quick overview of numeric variables |
| df.shape | The shape attribute contains a tuple that holds the number of rows followed by the number of columns. |
| df.values | The values attribute contains the data values in a 2-dimensional NumPy array. |
| df.columns | The columns attribute contains column names. |
| df.index | The index attribute contains row numbers or row names. |

Let’s try them all. Load homelessness.csv to homelessness DataFrame and explore the data.

**Sorting and Subnetting (Filtering) – We did some in the previous worksheet. Let’s dig deeper.**

**A screenshot of a computer

Description automatically generated**

Setting the ascending argument to False will sort the data the other way around, from heaviest dog to lightest dog.

**A screenshot of a computer

Description automatically generated with medium confidence**

**We can sort with multiple variables. You need to pass a list of variables as an argument.**

**A screenshot of a computer

Description automatically generated with medium confidence**

To change the direction values are sorted in, pass a list to the ascending argument to specify which direction sorting should be done for each variable.

**A screenshot of a computer

Description automatically generated with medium confidence**

Selecting column(s) or subsetting column(s) can be done as follows.

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**A screenshot of a computer code

Description automatically generated with low confidence**

**Let’s practice with homelessness DataFrame. Load homelessness.csv into homelessness**

1. **Sort homelessness by the number of homeless individuals, from smallest to largest, and save this as homelessness\_ind. Print the head of the sorted DataFrame.**

**A picture containing text, receipt, font, white

Description automatically generated**

1. **Sort**homelessness**by the number of homeless**family\_members**in descending order, and save this as**homelessness\_fam**. Print the head of the sorted DataFrame.**

**A picture containing text, font, receipt, white

Description automatically generated**

1. **Sort**homelessness**first by region (ascending), and then by number of family members (descending). Save this as**homelessness\_reg\_fam**. Print the head of the sorted DataFrame.**

**A picture containing text, font, white, screenshot

Description automatically generated**

1. **Create a DataFrame called state\_fam that contains only the state and family\_members columns of homelessness, in that order. Print the head of the result.**

**A picture containing text, font, receipt, white

Description automatically generated**

**Let’s subset rows.**

1. **Filter homelessness for cases where the number of individuals is greater than ten thousand, assigning to ind\_gt\_10k. *View the printed result.***
2. **Filter homelessness for cases where the USA Census region is "Mountain", assigning to mountain\_reg. *View the printed result.***
3. **Filter homelessness for cases where the number of family\_members is less than one thousand and the region is "Pacific", assigning to fam\_lt\_1k\_pac. *View the printed result.***

**A screenshot of a computer screen

Description automatically generated with low confidence**

1. **Filter homelessness for cases where the USA census region is "South Atlantic" or it is "Mid-Atlantic", assigning to south\_mid\_atlantic. *View the printed result.***

**A picture containing text, screenshot, font, number

Description automatically generated**

1. **Filter homelessness for cases where the USA census state is in the list of Mojave states, canu, assigning to mojave\_homelessness. *View the printed result.***

**A close-up of numbers

Description automatically generated with low confidence**

****

**A screenshot of a computer

Description automatically generated with low confidence**

**Let’s add new columns.**

1. **Add a new column to homelessness, named total, containing the sum of the individuals and family\_members columns. (Not all rows are shown. There should be 50 rows in total.)**

**A screenshot of a computer

Description automatically generated with low confidence**

1. **Add another column to homelessness, named p\_individuals, containing the proportion of homeless people in each state who are individuals.**

A screenshot of a computer

Description automatically generated with low confidence

1. **Combo Attack…..**
   * **Add a column to homelessness, indiv\_per\_10k, containing the number of homeless individuals per ten thousand people in each state.**

Hint: 10000 \* homelessness['individuals'] / homelessness['state\_pop']

* + **Subset rows where indiv\_per\_10k is higher than 20, assigning to high\_homelessness.**
  + **Sort high\_homelessness by descending indiv\_per\_10k, assigning to high\_homelessness\_srt.**
  + **Select only the state and indiv\_per\_10k columns of high\_homelessness\_srt and save as result. Look at the result.**

**A picture containing text, font, screenshot, number

Description automatically generated**

**Summary Statistics with Pandas**

**A picture containing text, screenshot, font

Description automatically generated**

**A screenshot of a computer

Description automatically generated with low confidence**

**Writing your own function is also possible.**

**A picture containing text, screenshot, font

Description automatically generated**

**A screenshot of a computer

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Let’s Practice.**

**Load sales\_subset.csv to sales and try the following codes.**

# 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())

# Print the maximum of the date column

print(sales['date'].max())

# Print the minimum of the date column

print(sales['date'].min())

**We are going to see more about .cumsum() and .cummax() with the following codes.**

**You need to create a DataFrame, sales\_1\_1, which** **contains the sales data for department 1 of store 1.**

**sales\_1\_1 = sales[(sales[‘deparment’] == ……) & (…………………) ]**

# Sort sales\_1\_1 by date

sales\_1\_1 = sales\_1\_1.sort\_values('date', ascending = True)

# Get the cumulative sum of weekly\_sales, add as cum\_weekly\_sales col

sales\_1\_1['cum\_weekly\_sales'] = sales['weekly\_sales'].cumsum()

# Get the cumulative max of weekly\_sales, add as cum\_max\_sales col

sales\_1\_1['cum\_max\_sales'] = sales['weekly\_sales'].cummax()

# See the columns you calculated

print(sales\_1\_1[["date", "weekly\_sales", "cum\_weekly\_sales", "cum\_max\_sales"]])

**A picture containing text, font, screenshot, menu

Description automatically generated**

**Dropping Duplicates – It is possible that the data set contains duplicated values and sometimes we do not want to have them in the analysis.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated**

**Since Max and Max are different breeds, we can drop the rows with pairs of name and breed listed earlier in the dataset.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**After dropping duplicates, you may want to count the values.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Normalization helps us to understand data better. The normalize argument can be used to turn the counts into proportions of the total. 25% of the dogs that go to this vet are Labradors.**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Let’s practice.**

1. **Dropping Duplicates**

* Remove rows of sales with duplicate pairs of store and type and save as store\_types and print the head.

A screenshot of a computer

Description automatically generated with low confidence

* Remove rows of sales with duplicate pairs of store and department and save as store\_depts and print the head.

A screenshot of a computer

Description automatically generated with low confidence

* 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.

A picture containing text, font, screenshot, white

Description automatically generated

1. Counting and Normalization

* Count the number of stores of each store type in store\_types.
* Count the proportion of stores of each store type in store\_types.

A black text on a white background

Description automatically generated with low confidence

* 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.

A screenshot of a computer

Description automatically generated with medium confidence

**Summaries by Group (.groupby() method)**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer program

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Let’s see how multiple grouped summaries work. Try the following codes.**

# 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)

1. What do you see in the result? Analyze the data and answer the question in your opinion.

**Now let’s try the following codes.**

# 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)

# 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)

**A screenshot of a computer

Description automatically generated with low confidence**

**Pivot Table**

Pivot tables are another way of calculating grouped summary statistics. If you've ever used a spreadsheet, chances are you've used a pivot table. Let's see how to create pivot tables in pandas.

**A screenshot of a computer program

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

**A screenshot of a computer

Description automatically generated with medium confidence**

You also previously computed the mean weight grouped by two variables: color and breed. We can also do this using the pivot\_table method. To group by two variables, we can pass a second variable name into the columns argument.

**A screenshot of a computer

Description automatically generated with medium confidence**

If we set the margins argument to True, the last row and last column of the pivot table contain the mean of all the values in the column or row, not including the missing values that were filled in with 0s. For example, in the last row of the Labrador column, we can see that the mean weight of the Labradors is 26 kilograms. In the last column of the Brown row, the mean weight of the Brown dogs is 24 kilograms. The value in the bottom right, in the last row and last column, is the mean weight of all the dogs in the dataset. Using margins equals True allows us to see a summary statistic for multiple levels of the dataset: the entire dataset, grouped by one variable, by another variable, and by two variables.

**A screenshot of a computer

Description automatically generated with medium confidence**

**Try**

# 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\_value = 0))

**Multi-level Indexes (Hierarchical Indexes)**

**A screenshot of a computer code

Description automatically generated with medium confidence**

**A picture containing text, screenshot, font, number

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Let’s see some examples.**

**Load temperatures.csv to temperatures dataframe and try the following codes.**

# Look at temperatures

print(temperatures)

# Set the index of temperatures to city

temperatures\_ind = temperatures.set\_index('city')

# Look at temperatures\_ind

print(temperatures\_ind)

# Reset the temperatures\_ind index, keeping its contents

print(temperatures\_ind.reset\_index())

# Reset the temperatures\_ind index, dropping its contents

print(temperatures\_ind.reset\_index(drop = True))

# 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])

1. Complete the following tasks.

* Set the index of temperatures to the "country" and "city" columns, and assign this to temperatures\_ind.
* Specify two country/city pairs to keep: "Brazil"/"Rio De Janeiro" and "Pakistan"/"Lahore", assigning to rows\_to\_keep.
* Print and subset temperatures\_ind for rows\_to\_keep using .loc[].

**A screenshot of a computer

Description automatically generated with medium confidence**

**Layering Plots**

**Let’s see what we can do if we have overlapping plots.**

**Load avoplotto.pkl (pickle file) to avocados dataframe and try the following.**

# 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()

# 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', rot = 45)

# Show the plot

plt.show()

# Scatter plot of avg\_price vs. nb\_sold 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()

# Histogram of conventional avg\_price

avocados[avocados['type'] == 'conventional']['avg\_price'].plot(kind = 'hist')

# Histogram of organic avg\_price

avocados[avocados['type'] == 'organic']['avg\_price'].plot(kind = 'hist')

# Add a legend

plt.legend(['conventional','organic'])

# Show the plot

plt.show()

**A picture containing screenshot, text, diagram, pixel

Description automatically generated Add alpha = 0.6 in plot() and run again.**

**Missing Values in DataFrame**

In a pandas DataFrame, missing values are indicated with N-a-N, which stands for "not a number."

**A picture containing text, screenshot, font, number

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

If we chain dot-isna with dot-any, we get one value for each variable that tells us if there are any missing values in that column.

**A blue rectangle with white text

Description automatically generated with low confidence**

**A blue rectangle with white text

Description automatically generated with low confidence**

**A picture containing text, screenshot, line, rectangle

Description automatically generated**

One option is to remove the rows in the DataFrame that contain missing values. This can be done using the dropna method. However, this may not be ideal if you have a lot of missing data, since that means losing a lot of observations.

**A screenshot of a computer

Description automatically generated with low confidence**

**A screenshot of a computer

Description automatically generated with low confidence**

**Import avocados\_2016.csv to avocados dataframe and try**

# 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()

# Remove rows with missing values

avocados\_complete = avocados\_2016.dropna()

# Check if any columns contain missing values

print(avocados\_complete.isna().any())

**From WorldBank\_GDP.csv, answer the following questions.**

1. **Which country’s GDP is growing during the Year 2010 and Year 2018? Support your answer with visualizations.**

**From temperatures.csv, answer the following questions.**

1. **Find out which country has the highest average temperature.**
2. **Find out how many countries where the average temperature is in the range of 20 and 30 Celsius. Show all the countries and their average temperature.**
3. **Show the average temperature of Thailand during 2005-01-01 and 2010-01-01 and also find the average temperature during that period.**

**A screenshot of a computer

Description automatically generated with low confidence**

****