Inspecting a DataFrame

Exercise

When you get a new DataFrame to work with, the first thing you need to do is explore it and see what it contains. There are several useful methods and attributes for this.

.head() returns the first few rows (the “head” of the DataFrame).

.info() shows information on each of the columns, such as the data type and number of missing values.

.shape returns the number of rows and columns of the DataFrame.

.describe() calculates a few summary statistics for each column.

homelessness is a DataFrame containing estimates of homelessness in each U.S. state in 2018. The individual column is the number of homeless individuals not part of a family with children. The family_members column is the number of homeless individuals part of a family with children. The state_pop column is the state's total population.

pandas is imported for you.

Instructions

Print the head of the homelessness DataFrame.

Print information about the column types and missing values in homelessness.

Print the number of rows and columns in homelessness.

Print some summary statistics that describe the homelessness DataFrame.

In [ ]:

```
# Print the head of the homelessness data

# Print information about homelessness

# Print the shape of homelessness

# Print a description of homelessness

#____________________________________________________#
#Solutions

# Print the head of the homelessness data
import pandas as pd
print(homelessness.head())

# Print information about homelessness
print(homelessness.info())

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

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

Parts of a DataFrame

Exercise

To better understand DataFrame objects, it's useful to know that they consist of three components, stored as attributes:

.values: A two-dimensional NumPy array of values.

.columns: An index of columns: the column names.

.index: An index for the rows: either row numbers or row names.

You can usually think of indexes as a list of strings or numbers, though the pandas Index data type allows for more sophisticated options. (These will be covered later in the course.)

homelessness is available.

Instructions

Import pandas using the alias pd.

Print a 2D NumPy array of the values in homelessness.

Print the column names of homelessness.

Print the index of homelessness.

In [ ]:

```
# Import pandas using the alias pd

# Print the values of homelessness

# Print the column index of homelessness

# Print the row index of homelessness

#__________________________________________________________#
#Solutions

# Import pandas using the alias pd
import pandas as pd

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

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

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

Sorting rows

Exercise

Finding interesting bits of data in a DataFrame is often easier if you change the order of the rows. You can sort the rows by passing a column name to .sort_values().

In cases where rows have the same value (this is common if you sort on a categorical variable), you may wish to break the ties by sorting on another column. You can sort on multiple columns in this way by passing a list of column names.

Sort on ..>..> Syntax

one column ..>..> df.sort_values("breed")

multiple columns ..>..> df.sort_values(["breed", "weight_kg"])

By combining .sort_values() with .head(), you can answer questions in the form, "What are the top cases where…?".

homelessness is available and pandas is loaded as pd.

Instructions

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.

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.

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.

In [ ]:

```
# Sort homelessness by descending family members

# Print the top few rows

# Sort homelessness by region, then descending family members

# Print the top few rows

#____________________________________________________#
#Solutions


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

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

# 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())
```

Subsetting columns

Exercise

When working with data, you may not need all of the variables in your dataset. Square brackets ([]) can be used to select only the columns that matter to you in an order that makes sense to you. To select only "col_a" of the DataFrame df, use

df["col_a"]

To select "col_a" and "col_b" of df, use

df[["col_a", "col_b"]]

homelessness is available and pandas is loaded as pd.

Instructions

Create a DataFrame called individuals that contains only the individuals column of homelessness.

Print the head of the result.

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.

Create a DataFrame called ind_state that contains the individuals and state columns of homelessness, in that order.

Print the head of the result.

In [ ]:

```
# Select the individuals column

# Print the head of the result

# Select the state and family_members columns

# Print the head of the result

# Select only the individuals and state columns, in that order

# Print the head of the result

#_____________________________________________________#
#Solutions

# Select the individuals column
individuals = homelessness['individuals']

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

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

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

# 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())
```

Subsetting rows

Exercise

A large part of data science is about finding which bits of your dataset are interesting. One of the simplest techniques for this is to find a subset of rows that match some criteria. This is sometimes known as filtering rows or selecting rows.

There are many ways to subset a DataFrame, perhaps the most common is to use relational operators to return True or False for each row, then pass that inside square brackets.

dogs[dogs["height_cm"] > 60]

dogs[dogs["color"] == "tan"]

You can filter for multiple conditions at once by using the "bitwise and" operator, &.

dogs[(dogs["height_cm"] > 60) & (dogs["color"] == "tan")]

homelessness is available and pandas is loaded as pd.

Instructions

Filter homelessness for cases where the number of individuals is greater than ten thousand, assigning to ind_gt_10k. View the printed result.

Filter homelessness for cases where the USA Census region is "Mountain", assigning to mountain_reg. View the printed result.

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.

In [ ]:

```
# Filter for rows where individuals is greater than 10000

# See the result

# Filter for rows where region is Mountain

# See the result

# Filter for rows where family_members is less than 1000

# and region is Pacific

# See the result

#____________________________________________________#
#Solutions

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

# See the result
print(ind_gt_10k)

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

# See the result
print(mountain_reg)

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


# See the result
print(fam_lt_1k_pac)
```

Subsetting rows by categorical variables

Exercise

Subsetting data based on a categorical variable often involves using the "or" operator (|) to select rows from multiple categories. This can get tedious when you want all states in one of three different regions, for example. Instead, use the .isin() method, which will allow you to tackle this problem by writing one condition instead of three separate ones.

colors = ["brown", "black", "tan"]

condition = dogs["color"].isin(colors)

dogs[condition]

homelessness is available and pandas is loaded as pd.

Instructions

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.

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.

In [ ]:

```
# Subset for rows in South Atlantic or Mid-Atlantic regions

# See the result

# The Mojave Desert states

# Filter for rows in the Mojave Desert states

# See the result

#____________________________________________________#
#Solutions

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

# See the result
print(south_mid_atlantic)

# 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)
```

Adding new columns

Exercise

You aren't stuck with just the data you are given. Instead, you can add new columns to a DataFrame. This has many names, such as transforming, mutating, and feature engineering.

You can create new columns from scratch, but it is also common to derive them from other columns, for example, by adding columns together or by changing their units.

homelessness is available and pandas is loaded as pd.

Instructions

Add a new column to homelessness, named total, containing the sum of the individuals and family_members columns.

Add another column to homelessness, named p_individuals, containing the proportion of homeless people in each state who are individuals.

In [ ]:

```
# Add total col as sum of individuals and family_members

# Add p_individuals col as proportion of individuals

# See the result

#____________________________________________________#
#Solutions

# 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)
```

Combo-attack!

Exercise

You've seen the four most common types of data manipulation: sorting rows, subsetting columns, subsetting rows, and adding new columns. In a real-life data analysis, you can mix and match these four manipulations to answer a multitude of questions.

In this exercise, you'll answer the question, "Which state has the highest number of homeless individuals per 10,000 people in the state?" Combine your new pandas skills to find out.

Instructions

Add a column to homelessness, indiv_per_10k, containing the number of homeless individuals per ten thousand people in each state.

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.

In [ ]:

```
# Create indiv_per_10k col as homeless individuals per 10k state pop

# Subset rows for indiv_per_10k greater than 20

# Sort high_homelessness by descending indiv_per_10k

# From high_homelessness_srt, select the state and indiv_per_10k cols

# See the result

#____________________________________________________#
#Solutions

# 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)
```