**Subsetting rows by categorical variables**

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.

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

south\_mid\_atlantic = homelessnesshomelessness["region"].isin(["South Atlantic"]) | homelessness["region"].isin(["Mid-Atlantic"])

# See the result

print(homelessness[south\_mid\_atlantic]) # homelessness[condition]

Mutating, transforming, feature engineering ==== names for adding columns to a dataframe

Sorting

pd.sort\_values(“col\_name”)

or pd.sort\_values([“col\_name”])

* 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.*

# 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)

Notes from above -> homelessness[condition] gives the rows

Condition gives true or false

Type(homelessness[“indiv\_per\_10k”] > 20) ---🡪 pandas.core.series.Series

Type(high\_homelessness) --🡪 pandas.core.frame.DataFrame

Also note -> sort\_values by descending --🡪 use ascending = False

* Aggregating .agg()
* Update the aggregation functions called by .agg(): include iqr and np.median in that order.
* # Import NumPy and create custom IQR function
* import numpy as np
* def iqr(column):
* return column.quantile(0.75) - column.quantile(0.25)
* # this works def npmedian(column):
* ## return np.median(column.to\_numpy())
* # Update to print IQR and median of temperature\_c, fuel\_price\_usd\_per\_l, & unemployment
* print(sales[["temperature\_c", "fuel\_price\_usd\_per\_l", "unemployment"]].agg([iqr, np.median])) # used npmedian
* # my code worked but the tester requires to pass np.median as a function into .agg

Sort\_values -> to sort the entire dataframe values by a column

Use df.sort\_values(“column\_name”)

Drop\_duplicates(subset = “column\_name”)

df.drop\_duplicates(subset=”column\_name\_you\_want\_to\_drop\_duplicates\_from”)

Dropping Duplicates

* Remove rows of sales with duplicate pairs of store and type and save as store\_types and print the head.
* Remove rows of sales with duplicate pairs of store and department and save as store\_depts and print the head.
* Subset the rows that are holiday weeks using the is\_holiday column, and drop the duplicate dates, saving as holiday\_dates.
* Select the date column of holiday\_dates, and print.
* # Drop duplicate store/type combinations
* store\_types = sales.drop\_duplicates(["store", "type"])
* print(store\_types.head())
* # Drop duplicate store/department combinations
* store\_depts = sales.drop\_duplicates(["store", "department"])
* print(store\_depts.head())
* # Subset the rows where is\_holiday is True and drop duplicate dates
* holiday\_dates = sales[sales["is\_holiday"] == True].drop\_duplicates(["date"])
* # Print date col of holiday\_dates
* print(holiday\_dates["date"])
* Count the number of stores of each store type in store\_types.
* Count the proportion of stores of each store type in store\_types.
* Count the number of different departments in store\_depts, sorting the counts in descending order.
* Count the proportion of different departments in store\_depts, sorting the proportions in descending order.

# 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()

print(dept\_counts\_sorted)

# Get the proportion of departments of each number and sort

dept\_props\_sorted = store\_depts["department"].value\_counts(sort=True, normalize=True)

print(dept\_props\_sorted)

* Import numpy with the alias np.
* Get the min, max, mean, and median of weekly\_sales for each store type using .groupby() and .agg(). Store this as sales\_stats. Make sure to use numpy functions!
* Get the min, max, mean, and median of unemployment and fuel\_price\_usd\_per\_l for each store type. Store this as unemp\_fuel\_stats.
* # Import numpy with the alias np
* import numpy as np
* # For each store type, aggregate weekly\_sales: get min, max, mean, and median
* sales\_stats = sales.groupby("type")["weekly\_sales"].agg([np.min, np.max, np.mean, np.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([np.min, np.max, np.mean, np.median])
* # Print unemp\_fuel\_stats
* print(unemp\_fuel\_stats)

PIVOT TABLE

* Print the mean weekly\_sales by department and type, filling in any missing values with 0.
* # Print mean weekly\_sales by department and type; fill missing values with 0
* print(sales.pivot\_table(values = "weekly\_sales", index = "type", columns = "department", fill\_value = 0))
* Print the mean weekly\_sales by department and type, filling in any missing values with 0 and summing all rows and columns.
* # Print the mean weekly\_sales by department and type; fill missing values with 0s; sum all rows and cols
* print(sales.pivot\_table(values="weekly\_sales", index="department", columns="type", fill\_value = 0, margins = True))

Setting and Removing indexes

* *Look at temperatures*.
* Set the index of temperatures to "city", assigning to temperatures\_ind.
* *Look at temperatures\_ind. How is it different from temperatures?*
* Reset the index of temperatures\_ind, keeping its contents.
* Reset the index of temperatures\_ind, dropping its contents.
* # Look at temperatures
* print(temperatures)
* # Index temperatures by city
* temperatures\_ind = temperatures.set\_index(["city"])
* # Look at temperatures\_ind
* print(temperatures\_ind)
* # Reset the index, keeping its contents
* print(temperatures\_ind.reset\_index())
* # Reset the index, dropping its contents
* print(temperatures\_ind.reset\_index(drop = True))
* Create a list called cities that contains "Moscow" and "Saint Petersburg".
* Use [] subsetting to filter temperatures for rows where the city column takes a value in the cities list.
* Use .loc[] subsetting to filter temperatures\_ind for rows where the city is in the cities list.
* # 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[["Moscow", "Saint Petersburg"]])
* 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[]
* # 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])
* Sort temperatures\_ind by the index values.
* Sort temperatures\_ind by the index values at the "city" level.
* Sort temperatures\_ind by ascending country then descending city.
* # 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"))
* # Sort temperatures\_ind by country then descending city
* print(temperatures\_ind.sort\_index(level = ["country", "city"], ascending = [True, False]))
* Sort the index of temperatures\_ind.
* Use slicing with .loc[] to get these subsets:
  + from Pakistan to Russia.
  + from Lahore to Moscow. (*This will return nonsense.*)
  + from Pakistan, Lahore to Russia, Moscow.
* # 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")])
* Use .loc[] slicing to subset rows from India, Hyderabad to Iraq, Baghdad.
* Use .loc[] slicing to subset columns from date to avg\_temp\_c.
* Slice in both directions at once from Hyderabad to Baghdad, and date to avg\_temp\_c.
* # 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"])
* Use Boolean conditions, not .isin() or .loc[], and the full date "yyyy-mm-dd", to subset temperatures for rows in 2010 and 2011 and print the results.
* Set the index to the date column and sort it.
* Use .loc[] to subset temperatures\_ind for rows in 2010 and 2011.
* Use .loc[] to subset temperatures\_ind for rows from Aug 2010 to Feb 2011.

Note when using Boolean and date time -> the full date needs to be used eg, “2010-01-01’ , just using the year will not work

# 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 the 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["2010-08-01":"2011-02-01"])

Note above, also a thing about loc, it is end inclusive, so you’ll need the exact label for start and end.

Use .iloc[] on temperatures to take subsets.

* Get the 23rd row, 2nd column (index positions 22 and 1).
* Get the first 5 rows (index positions 0 to 5).
* Get all rows, columns 3 and 4 (index positions 2 to 4).
* Get the first 5 rows, columns 3 and 4.
* ####################
* # 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])