LECTURE 5

Pandas, Part III

Advanced Pandas (More on Grouping, Aggregation, Pivot Tables, and Merging)

Data Science, Fall 2023 @ Knowledge Stream

Sana Jabbar

1

• Pandas, Part III

• Groupby Review

• More on Groupby

• Pivot Tables

• Joining Tables

• EDA, Part I

• Structure: Tabular Data

• Granularity Today’s Roadmap • Structure: Variable Types Lecture 5

2



• **Pandas, Part III**

• **Groupby Review**

• More on Groupby

• Pivot Tables

• Joining Tables

• EDA, Part I

• Structure: Tabular Data

• Granularity Groupby Review • Structure: Variable Types Lecture 5

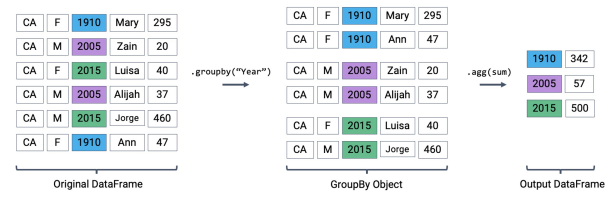
3



Revisiting groupby.agg

dataframe.groupby(column\_name).agg(aggregation\_function)

babynames.groupby("Year")[["Count"]].agg(sum) computes the total number of babies born in each year.

4

Revisiting groupby.agg

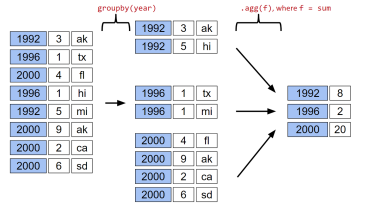
A groupby operation involves some combination of **splitting the object, applying a function**, and **combining the results**.

● So far, we've seen that df.groupby("year").agg(sum):

○ **Split** df into sub-DataFrames based on year.

○ **Apply** the sum function to each column of each sub-DataFrame.

○ **Combine** the results of sum into a single DataFrame, indexed by year.

5

Groupby Review Question

groupby .agg(f), where f = **max**

A 3 B 1 C 4 A 1 B 5 C 9 A 2 C 5

ak tx

fl

hi

mi ak ca sd

A 3 A 1 A 2

B 1 B 5 B 6

C 4 C 9

ak hi

ca

tx

mi nc

fl

ak

A 3 B 6 C 9

?? ?? ??

B 6

nc

C 5

sd

What will go in the ?? 6

Answer

groupby .agg(f), where f = **max**

A3 B1 C4 A1 B5 C9 A2 C5

ak tx

fl

hi

mi ak ca sd

A3 A1 A2

B1 B5 B6

C4 C9

ak hi

ca

tx

mi nc

fl

ak

A3 B6 C9

hi

tx sd

B6

nc

C5

sd7

Aggregation Functions

What goes inside of .agg( )?

● Any function that aggregates several values into one summary value ● Common examples:

In-Built Python Functions

NumPy

Functions

In-Built pandas functions

.agg(sum) .agg(np.sum) .agg("sum")

.agg(max) .agg(np.max) .agg("max")

.agg(min) .agg(np.min) .agg("min")

.agg(np.mean) .agg("mean")

.agg("first")

.agg("last")

Some commonly-used aggregation functions can even be called directly, without the explicit use of .agg( )

babynames.groupby("Year").mean()

8

Putting Things Into Practice

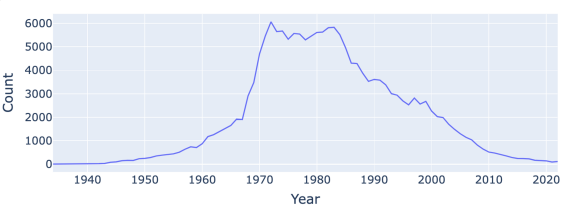
Goal: Find the baby name with sex "F" that has fallen in popularity the most in California.

f\_babynames = babynames[babynames["Sex"] == "F"]

f\_babynames = f\_babynames.sort\_values(["Year"])

jenn\_counts\_series = f\_babynames[f\_babynames["Name"] == "Jennifer"]["Count"]

Number of Jennifers Born in California Per Year.

9

What Is "Popularity"?

Goal: Find the baby name with sex "F" that has fallen in popularity the most in California.

How do we define "fallen in popularity?"

● Let’s create a metric: "Ratio to Peak" (RTP).

● The RTP is the ratio of babies born with a given name in 2022 to the *maximum* number of babies born with that name in *any* year.

Example for "Jennifer":

● In 1972, we hit peak Jennifer. 6,065 Jennifers were born.

● In 2022, there were only 114 Jennifers.

● RTP is 114 / 6065 = 0.018796372629843364.

10

Calculating RTP

max\_jenn = max(f\_babynames[f\_babynames["Name"] == "Jennifer"]["Count"]) 6065

curr\_jenn = f\_babynames[f\_babynames["Name"] == "Jennifer"]["Count"].iloc[-1]

114

Remember: f\_babynames is sorted by year.

rtp = curr\_jenn / max\_jenn 0.018796372629843364

def ratio\_to\_peak(series):

.iloc[-1] means “grab the latest year”

return series.iloc[-1] / max(series)

jenn\_counts\_ser = f\_babynames[f\_babynames["Name"] == "Jennifer"]["Count"] ratio\_to\_peak(jenn\_counts\_ser)

0.018796372629843364

11

Renaming Columns After Grouping

By default, .groupby will not rename any aggregated columns (the column is still named "Count", even though it now represents the RTP.

For better readability, we may wish to rename "Count" to "Count RTP"

rtp\_table = f\_babynames.groupby("Name")[["Count"]].agg(ratio\_to\_peak) rtp\_table = rtp\_table.rename(columns = {"Count": "Count RTP"})

12

Some Data Science Payoff

By sorting rtp\_table we can see the names whose popularity has decreased the most.

rtp\_table.sort\_values("Count RTP")

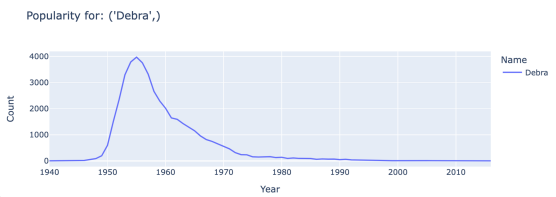
13

Some Data Science Payoff

By sorting rtp\_table we can see the names whose popularity has decreased the most. rtp\_table.sort\_values("Count RTP") 

px.line(f\_babynames[f\_babynames["Name"] == "Debra"],

x = "Year", y = "Count")

We’ll learn about plotting in week 4. 14

Some Data Science Payoff

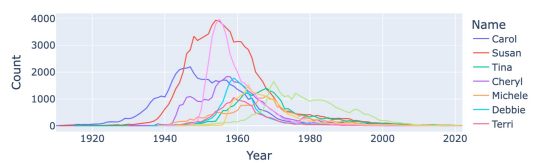
We can get the list of the top 10 names and then plot popularity with::

top10 = rtp\_table.sort\_values("Count RTP").head(10).index



px.line(f\_babynames[f\_babynames["Name"].isin(top10)],

x = "Year", y = "Count", color = "Name")

15

Answer

Before, we saw that the code below generates the Count RTP for all female names. babynames.groupby("Name")[["Count"]].agg(ratio\_to\_peak)

We use similar logic to compute the summed counts of all baby names.

babynames.groupby("Name")[["Count"]].agg(sum) 

or

babynames.groupby("Name")[["Count"]].sum()

16

Answer

Now, we create groups for each *year*.

babynames.groupby("Year")[["Count"]].agg(sum) 

or

babynames.groupby("Year")[["Count"]].sum()

or

babynames.groupby("Year").sum(numeric\_only=True)

17

Plotting Birth Counts

Plotting the DataFrame we just generated tells an interesting story.

puzzle2 = babynames.groupby("Year")[["Count"]].agg(sum)

px.line(puzzle2, y = "Count")

18

• **Pandas, Part III**

• Groupby Review

• **More on Groupby**

• Pivot Tables

• Joining Tables

• EDA, Part I

• Structure: Tabular Data

• Granularity More on Groupby • Structure: Variable Types Lecture 5

19



Raw GroupBy Objects and Other Methods

The result of a groupby operation applied to a DataFrame is a DataFrameGroupBy object. ● It is not a DataFrame!

grouped\_by\_year = elections.groupby("Year")

type(grouped\_by\_year)

pandas.core.groupby.generic.DataFrameGroupBy

Given a DataFrameGroupBy object, can use various functions to generate DataFrames (or Series). agg is only one choice:

df.groupby(col).mean() df.groupby(col).sum() df.groupby(col).min() df.groupby(col).max()

df.groupby(col).first() df.groupby(col).last() df.groupby(col).size() df.groupby(col).count()

df.groupby(col).filter() �What’s the difference?

See https://pandas.pydata.org/docs/reference/groupby.html for a list of DataFrameGroupBy methods. 20

groupby.size() and groupby.count()

groupby("year") .size()

1992 3 1996 1 2000 4 1996 1 1992 NaN

2000 9 2000 2 2000 6

ak tx

fl

hi

mi NaN ca sd

1992 3 1992 NaN

1996 1 1996 1

2000 4 2000 9

ak mi

tx

hi

fl

NaN

Returns a Series object counting the number of rows in each group.

1992 2

1996 2

2000 4

Similar to value\_counts() except that size() does not

2000 2 ca

sort the index based on the

2000 6

sd

frequency of entries.

21

groupby.size() and groupby.count()

groupby("year") .count()

1992 3 1996 1 2000 4

ak tx

fl

1992 3 1992 NaN

ak mi

Returns a DataFrame with the counts of non-missing values in each column.

1996 1 1992 NaN 2000 9 2000 2 2000 6

hi

mi NaN ca sd

1996 1 1996 1

2000 4 2000 9

tx

hi

fl

NaN

1992 1 1996 2 2000 4

2 2 3

2000 2 ca

2000 6

sd

22

Filtering by Group

Another common use for groups is to filter data.

● groupby.filter takes an argument func.

● func is a function that:

○ Takes a DataFrame as input.

○ Returns either True or False.

● filter applies func to each group/sub-DataFrame:

○ If func returns **True** for a group, then all rows belonging to the group are **preserved**.

○ If func returns **False** for a group, then all rows belonging to that group are **filtered out**.

● Notes:

○ Filtering is done per group, not per row. Different from boolean filtering.

○ Unlike agg(), the column we grouped on does NOT become the index!

23

groupby.filter()

groupby .filter(f), where

num

num

A 3 B 1

C 4 A 1

A 3 A 1 A 2

B 1

f = lambda sf: sf["num"].sum() > 10

6

B 1

C 4

B 5 C 9 A 2 D 5 B 6

B 5 B 6

C 4 C 9

D 5

12 13 5

B 5 C 9 B 6

24

Filtering Elections Dataset

Going back to the elections dataset.

Let's keep only election year results where the max '%' is less than 45%.

elections.groupby("Year").filter(lambda sf: sf["%"].max() < 45)

25

groupby Puzzle

**Puzzle**: We want to know the **best election by each party**.

● Best election: The election with the highest % of votes.

● For example, Democrat’s best election was in 1964, with candidate Lyndon Johnson

winning 61.3% of votes.

26



Attempt #1

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020?

elections.groupby("Party").max().head(10)

27

Problem with Attempt #1

Why does the table seem to claim that Woodrow Wilson won the presidency in 2020? elections.groupby("Party").max().head(10) 

Every column is calculated

independently! Among

Democrats:

● Last year they ran: 2020.

● Alphabetically the latest

candidate name: Woodrow

Wilson.

● Highest % of vote: 61.34%.

28

Attempt #2: Motivation

● We want to preserve entire rows, so we need an aggregate function that does that. 

29

Attempt #2: Solution

.sort\_values("%",

ascending = False) .groupby("Party") Order is preserved

in sub-DataFrames!

.first()

DR 1824 57%

Dem 1964

61%

Dem 1964 61%

DR 1824 43% Dem 1828 56% Nat 1828 44%

Dem 1832 54% …

Dem 2020 51% Rep 2020 47% Green 2020 0.2%

Dem 1936 60% Rep 1972 60% Rep 1920 60%

Rep 1984 59% …

Cons 2004 0.1% Pop 1992 0.1% Green 2004 0.01%

Dem 1936 60%

Rep 1972 60% Rep 1920 60% Rep 1984 59%

Green 2020 0.2% Green 2004 0.01%

Dem 1964 61% Rep 1972 60% Green 2000 2.7%

30

Attempt #2: Solution

● First sort the DataFrame so that rows are in descending order of %.

● Then group by Party and take the first item of each sub-DataFrame.

● Note: Lab will give you a chance to try this out if you didn't quite follow during lecture.

elections\_sorted\_by\_percent = elections.sort\_values("%", ascending=False) elections\_sorted\_by\_percent.groupby("Party").first()



elections\_sorted\_by\_percent 31

groupby Puzzle - Alternate Approaches

Using a lambda function

elections\_sorted\_by\_percent = elections.sort\_values("%", ascending=False) elections\_sorted\_by\_percent.groupby("Party").agg(lambda x : x.iloc[0])

Using idxmax function

best\_per\_party = elections.loc[elections.groupby("Party")["%"].**idxmax**()] Using drop\_duplicates function

best\_per\_party2 = elections.sort\_values("%").**drop\_duplicates**(["Party"], keep="last") 32

There's More Than One Way to Find the Best Result by Party

In Pandas, there’s more than one way to get to the same answer.

● Each approach has different tradeoffs in terms of readability, performance, memory consumption, complexity, etc.

● Takes a very long time to understand these tradeoffs!

● If you find your current solution to be particularly convoluted or hard to read, maybe try finding another way!

33

More on DataFrameGroupby Object

We can look into DataFrameGroupby objects in following ways:

grouped\_by\_party = elections.groupby("Party")

grouped\_by\_party.groups

grouped\_by\_party.get\_group("Socialist")

34

• **Pandas, Part III**

• Groupby Review

• More on Groupby

• **Pivot Tables**

• Joining Tables

• EDA, Part I

• Structure: Tabular Data

• Granularity Pivot Tables • Structure: Variable Types Lecture 5

35



Grouping by Multiple Columns

Suppose we want to build a table showing the total number of babies born of each Gender in each year. One way is to groupby *using both columns* of interest:

babynames.groupby(["Year", "Sex"])[["Count"]].agg(sum).head(6)

Note: Resulting DataFrame is 

multi-indexed. That is, its index

has multiple dimensions. Will

explore in a later lecture.

36

Pivot Tables

A more natural approach is to create a pivot table.

babynames\_pivot = babynames.pivot\_table( 

index = "Year", # rows (turned into index)

columns = "Sex", # column values

values = ["Count"], # field(s) to process in each group

aggfunc = np.sum, # group operation

)

babynames\_pivot.head(6)

37

groupby(["Year", "Sex"]) vs. pivot\_table

The pivot table more naturally represents our data.

groupby output pivot\_table output



38

Pivot Table Mechanics

RC

A 3

F

B 1

M

C 4

F

A3

F

A2 F

AM1

f = sum f

f

AF5 AM1

FM

A 1 M

B 5 F

C 9 M

A 2 F

D 5 F

B 6 M

group

f

BF5

B1

M

f

B6

M

f

CF4

f

CM9

f

DF5

BF5 BM7

CF4 CM9 DF5

...

ABCD

5 545

1

7

9

NaN

39

Pivot Tables with Multiple Values

We can include multiple values in our pivot tables.

babynames\_pivot = babynames.pivot\_table( 

index = "Year", # rows (turned into index)

columns = "Sex", # column values

values = ["Count", "Name"],

aggfunc = np.max, # group operation

)

babynames\_pivot.head(6)

40

• **Pandas, Part III**

• Groupby Review

• More on Groupby

• Pivot Tables

• **Joining Tables**

• EDA, Part I

• Structure: Tabular Data

• Granularity Join Tables • Structure: Variable Types Lecture 5

41



Joining Tables

Suppose want to know the popularity of presidential candidate's names in 2022. ● Example: Dwight Eisenhower's name Dwight is not popular today, with only 5 babies born with this name in California in 2022.

To solve this problem, we’ll have to join tables.

42

Creating Table 1: Babynames in 2022

Let's set aside names in California from 2022 first:

babynames\_2022 = babynames[babynames["Year"] == 2022]

babynames\_2022 

43

Creating Table 2: Presidents with First Names

To join our table, we’ll also need to set aside the first names of each candidate. elections["First Name"] = elections["Candidate"].str.split().str[0]

44

Joining Our Tables

merged = pd.merge(left = elections, right = babynames\_2022,

left\_on = "First Name", right\_on = "Name")

merged = elections.merge(right = babynames\_2022,

left\_on = "First Name", right\_on = "Name")

45

Joining Our Tables: Four Options

1. inner\_merged\_df = pd.merge(left\_df, right\_df, on='key’) 2. outer\_merged\_df = pd.merge(left\_df, right\_df, on='key', how='outer') 3. left\_merged\_df = pd.merge(left\_df, right\_df, on='key', how='left’) 4. right\_merged\_df = pd.merge(left\_df, right\_df, on='key', how='right')

**DataFrame 1 DataFrame 2**

**ID Age**

0 2 25 1 3 30 2 4 28

**ID Name**

0 0 Bob 1 1 Alice 2 2 John

46

Joining Our Tables: Inner Merge

1. inner\_merged\_df = pd.merge(left\_df, right\_df, on=‘ID’)

**DataFrame 1 DataFrame 2**

**ID Name**

0 1 Bob 1 2 Alice 2 3 John

**Inner merge**

**ID Age**

0 2 25

1 3 30

2 4 28

**ID Name Age**

0 2 Alice 25 1 3 John 30

47

Joining Our Tables: Outer Merge

1. inner\_merged\_df = pd.merge(left\_df, right\_df=, on=‘ID’, how=‘outer’) **DataFrame 1 DataFrame 2**

**ID Name**

0 1 Bob 1 2 Alice 2 3 John

**ID Age**

0 2 25

1 3 30

2 4 28

**ID Name Age**

**outer merge**

0 1 Bob NaN

1 2 Alice 25

2 3 John 30

4 4 NaN 28

48

Joining Our Tables: right Merge

1. inner\_merged\_df = pd.merge(left\_df, right\_df=, on=‘ID’, how=‘right’) **DataFrame 1 DataFrame 2**

**ID Name**

0 1 Bob 1 2 Alice 2 3 John

**ID Age**

0 2 25

1 3 30

2 4 28

**ID Name Age**

**right merge**

0 2 Alice 25 1 3 John 30 2 4 NaN 28

49

Joining Our Tables: left Merge

1. inner\_merged\_df = pd.merge(left\_df, right\_df=, on=‘ID’, how=‘left’) **DataFrame 1 DataFrame 2**

**ID Name**

0 1 Bob 1 2 Alice 2 3 John

**ID Age**

0 2 25

1 3 30

2 4 28

**ID Name Age**

**left merge**

0 1 Bob NaN

1 2 Alice 25

2 3 John 30

50

Data Wrangling and EDA, Part I Exploratory Data Analysis and its role in the data science lifecycle

51

Box of Data The Next Step



**EDA Guiding Principles** 52

Plan for First Few Weeks

? 

Question & Problem

Formulation

Prediction and Inference

Data 

Acquisition

**Exploratory** 

**Data Analysis**

Reports, Decisions,

and Solutions

**(Weeks 1 and 2) (Weeks 2 and 3)**

Exploring and Cleaning Tabular Data From datascience to pandas

Data Science in Practice

**EDA, Data Cleaning**, Text processing (regular expressions)

53

Structure: Tabular Data

Lecture 05



• Pandas, Part III

• Groupby Review

• More on Groupby

• Pivot Tables

• Joining Tables

• **EDA, Part I**

• **Structure: Tabular Data**

• Granularity

• Structure: Variable Types

54

Rectangular and Non-rectangular Data

Data come in many different shapes.

Rectangular data



Non-rectangular data

1723786 55

Rectangular Data

We often prefer rectangular data for data analysis (why?)

• Regular structures are easy manipulate and analyze **Records**/Rows

• A big part of data cleaning is about

transforming data to be more rectangular

Two kinds of rectangular data: **Tables** and **Matrices**.

**Fields**/Attributes/ Features/Columns

**Tables** (a.k.a. DataFrames in R/Python and relations in SQL)

● Named columns with different types

● Manipulated using data

transformation languages (map, filter, group by, join, …)

**Matrices**

● Numeric data of the same type

(float, int, etc.)

● Manipulated using linear algebra

What are the differences?

Why would you use one over the

other?

56

Tuberculosis – United States, 2021

CDC Morbidity and Mortality Weekly Report (MMWR) 03/25/2022. 

57



What is **incidence**?

Why use it here?

How was “9.4%

increase” computed?

**Question**: Can we **reproduce** these rates using government data?

CSV: Comma-Separated Values

Tuberculosis in the US [CDC **source**].

CSV is a very common **tabular file format**.

● **Records** (rows) are delimited by a newline: '\n', "\r\n"

● **Fields** (columns) are delimited by commas: ','

Pandas: **pd.read\_csv**(header=...) **Fields**/Attributes/Features/Columns

**Demo Slides **

**Records**/Rows

U.S. jurisdiction TB cases 2019 …

0 Total 8,900 … 1 Alabama 87 …58

Granularity Lecture 04, Data 100 Fall 2023



• Pandas, Part III

• Groupby Review

• More on Groupby

• Pivot Tables

• Joining Tables

• **EDA, Part I**

• Structure: Tabular Data

• **Granularity**

• Structure: Variable Types

59

(we’ll come back to this)

Key Data

**Structure** -- the “shape” of a data file

**Granularity** -- how fine/coarse is each datum

**Scope** -- how (in)complete is the data

Properties to Consider in EDA



**Temporality** -- how is the data situated in time

**Faithfulness** -- how well does the data capture “reality”

60

Granularity: How Fine/Coarse Is Each Datum?

What does each **record** represent?

● Examples: a purchase, a person, a group of users Do all records capture granularity at the same level? ● Some data will include summaries (aka **rollups**) as records. If the data are **coarse**, how were the records aggregated? ● Sampling, averaging, maybe some of both…

Rec. 1 Rec. 2 Rec. 3

Rec. 1 Rec. 2 Rec. 3

Fine 

Grained

Rec. 1

Coarse

Grained

To the demo!!

61

Structure: Variable Types

Lecture 04, Data 100 Fall 2023



• Pandas, Part III

• Groupby Review

• More on Groupby

• Pivot Tables

• Joining Tables

• **EDA, Part I**

• Structure: Tabular Data

• Granularity

• **Structure: Variable Types**

62

(we’re back to this)

**Variable Type**

****

**Structure** -- the “shape” of a data file

**Granularity** -- how fine/coarse is each datum

**Scope** -- how (in)complete is the data

**Temporality** -- how is the data situated in time

**Faithfulness** -- how well does the data capture “reality”

63

Variables Are Columns

Let’s look at records with the same granularity. What does each **column** represent?

A **variable** is a **measurement** of a particular concept.

It has two common properties:

● **Datatype/Storage type**:

The U.S. Jurisdiction **variable**

U.S. jurisdiction TB cases 2019 … 1 Alabama 87 … 2 Alaska 58 … … … … …

How each variable value is stored in memory. df[colname].dtype

○ integer, floating point, boolean, object (string-like), etc.

Affects which pandas functions you use.

● **Variable type/Feature type**:

Conceptualized measurement of information (and therefore what values it can take on). ○ Use expert knowledge

○ Explore data itself

○ Consult data codebook (if it exists). Affects how you visualize and interpret the data.

⚠In this class, “variable types” are conceptual!!

64

Variable Feature Types

**Many variables do not sit**

Variable

**Intervals have meaning.**

**neatly in one of these**

**categories!!**

**Quantitative Qualitative (categorical)**

Continuous Discrete Ordinal Nominal

Could be measured to arbitrary precision.

**Examples:**

• Price

• Temperature

Finite possible values

**Examples:**

• Number of siblings • Yrs of education

Categories w/ordered levels; no consistent meaning to difference **Examples:**

• Preferences

• Level of education

Categories w/ no specific ordering. **Examples:**

• Political Affiliation • Cal lD number

Note that **qualitative variables** could have numeric levels; conversely, **quantitative variables** could be stored as strings!

65

Variable Feature Types

**Many variables do not sit**

Variable

**Intervals have meaning.**

**neatly in one of these**

**categories!!**

**Quantitative Qualitative (categorical)**

Continuous Discrete Ordinal Nominal

Could be measured to arbitrary precision.

**Examples:**

• Price

• Temperature

Finite possible values

**Examples:**

• Number of siblings • Yrs of education

Categories w/ordered levels; no consistent meaning to difference **Examples:**

• Preferences

• Level of education

Categories w/ no specific ordering. **Examples:**

• Political Affiliation • Cal lD number

Note that **qualitative variables** could have numeric levels; conversely, **quantitative variables** could be stored as strings!

66

Variable Types

What is the feature type (i.e., variable � type) of each variable?

**Q Variable Feature Type** 1 CO2 level (ppm)

Variable

**Quantitative Qualitative**

2 Number of siblings

Cont inuous

Dis

crete

Ord inal

Nom inal

3 GPA

4 Income bracket (low, med, high)

5 Race/Ethnicity

6 Number of years of education

7 Yelp Rating

A B C D

67

Variable Types

� What is the feature type of each variable? **Q Variable Feature Type**

Variable

**Quantitative Qualitative**

1 CO2 level (ppm) 2 Number of siblings

**A. Quantitative Cont. B. Quantitative Discrete**

Cont inuous

Dis

crete

Ord inal

Nom inal

3 GPA

4 Income bracket (low, med, high)

5 Race/Ethnicity

6 Number of years of education

7 Yelp Rating

**A. Quantitative Cont. C. Qualitative Ordinal**

**D. Qualitative Nominal B. Quantitative Discrete**

**C. Qualitative Ordinal**

A B C D

Many of these examples show how “shaggy” these categories are!! We will revisit variable types when we learn how to visualize variables.

68