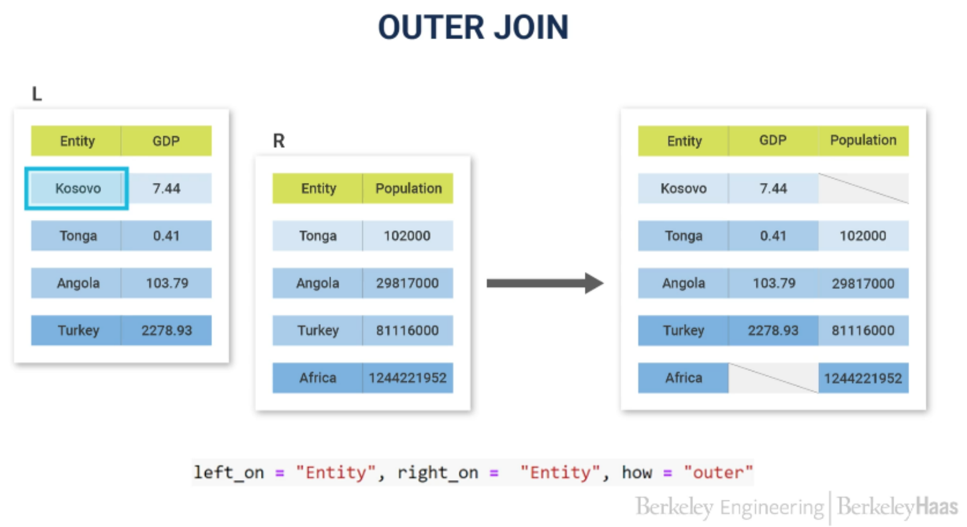
1 Module 4: Fundamentals of Data Analysis

Contents

1.1 Transforming data using pandas merge

- He has a GDP df and a separate Population df
- **Renaming Columns**: [DataFrame].rename(columns = "[old col. name]" : "[new col. name]")
- **Joining tables**: pd.merge(left = "[DataFrame 1]", right = "[DataFrame 2]", left_on = "[Column name 1]" ,right_on = "[Column Name 1]", how = "[left, right, inner, outer]")
  e.g. pd.merge(left='GDP',right='pop',right_on='Entity',left_on='Entity',how='outer')
  Pandas looks for all values which match in both the left & right dfs' "Entity" column
- There are four join types: **left, right, inner and outer**
  - Outer: if value does not exist in one or the other df, then the value is filled in as NaN



OUTER JOIN

L

Entity	GDP
Kosovo	7.44
Tonga	0.41
Angola	103.79
Turkey	2278.93

R

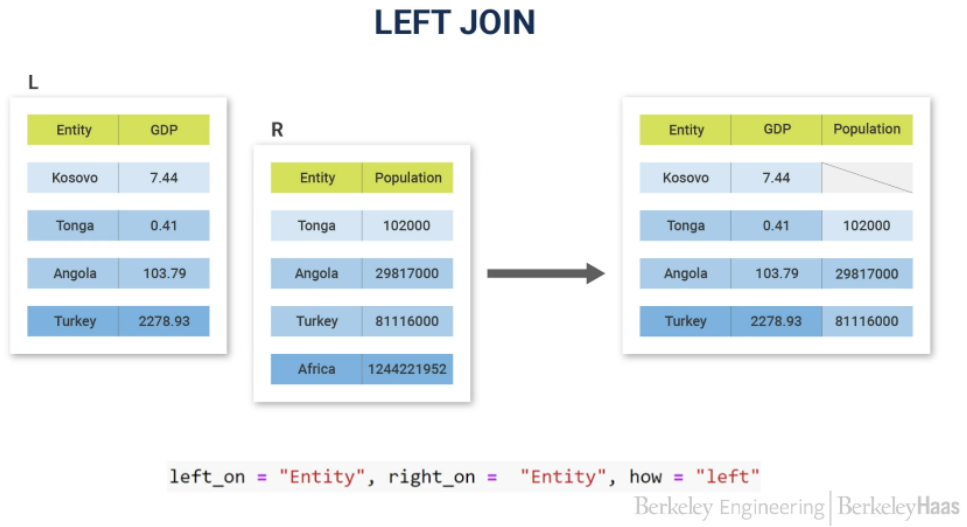
Entity	Population
Tonga	102000
Angola	29817000
Turkey	81116000
Africa	1244221952

Entity	GDP	Population
Kosovo	7.44	
Tonga	0.41	102000
Angola	103.79	29817000
Turkey	2278.93	81116000
Africa		1244221952

```
left_on = "Entity", right_on =  "Entity", how = "outer"
```

Berkeley Engineering | BerkeleyHaas

- Left/right join: Prioritizes the L/R table to always have its column value in the table; will drop rows in the unprioritized table if it does not exist in the main one



LEFT JOIN

L

Entity	GDP
Kosovo	7.44
Tonga	0.41
Angola	103.79
Turkey	2278.93

R

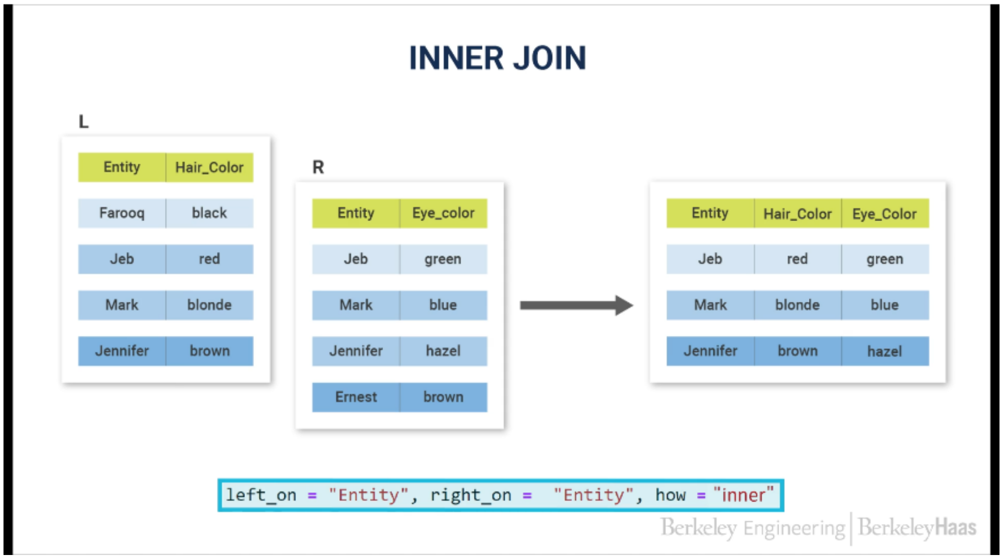
Entity	Population
Tonga	102000
Angola	29817000
Turkey	81116000
Africa	1244221952

Entity	GDP	Population
Kosovo	7.44	
Tonga	0.41	102000
Angola	103.79	29817000
Turkey	2278.93	81116000

```
left_on = "Entity", right_on =  "Entity", how = "left"
```

Berkeley Engineering | BerkeleyHaas

- Inner: Keeps rows only if the column values exist in BOTH tables



INNER JOIN

L

Entity	Hair_Color
Farooq	black
Jeb	red
Mark	blonde
Jennifer	brown

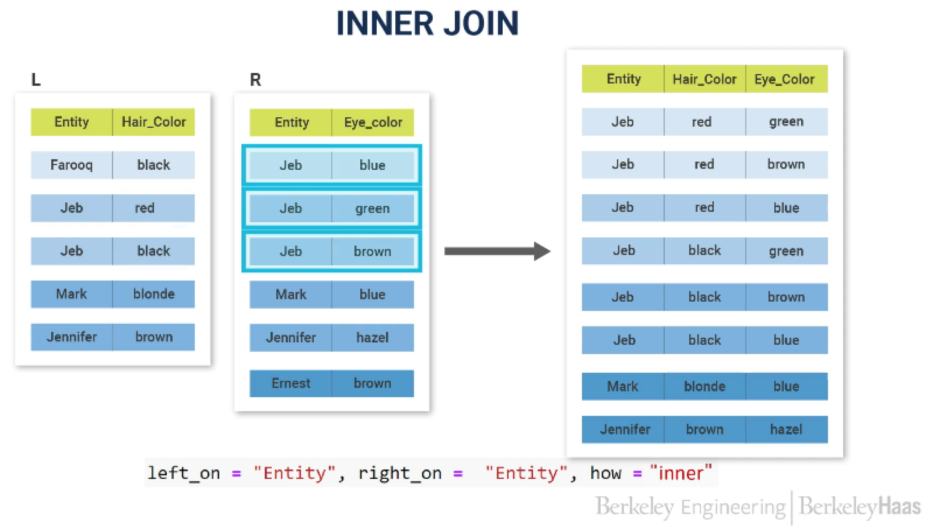
R

Entity	Eye_color
Jeb	green
Mark	blue
Jennifer	hazel
Ernest	brown

Entity	Hair_Color	Eye_Color
Jeb	red	green
Mark	blonde	blue
Jennifer	brown	hazel

```
left_on = "Entity", right_on =  "Entity", how = "inner"
```

Note that inner will permutate for each possible combination for nonunique keys. Resolve with "merging on multiple cols" below.



INNER JOIN

L

Entity	Hair_Color
Farooq	black
Jeb	red
Jeb	black
Mark	blonde
Jennifer	brown

R

Entity	Eye_color
Jeb	blue
Jeb	green
Jeb	brown
Mark	blue
Jennifer	hazel
Ernest	brown

Entity	Hair_Color	Eye_Color
Jeb	red	green
Jeb	red	brown
Jeb	red	blue
Jeb	black	green
Jeb	black	brown
Jeb	black	blue
Mark	blonde	blue
Jennifer	brown	hazel

```
left_on = "Entity", right_on =  "Entity", how = "inner"
```

1.2 Joining by multiple fields

- **Merging on multiple columns from each df**: pd.merge(left=, right=, how=, left_on = ['Col1', 'Col2'], right_on = ['Col1', 'Col2'])

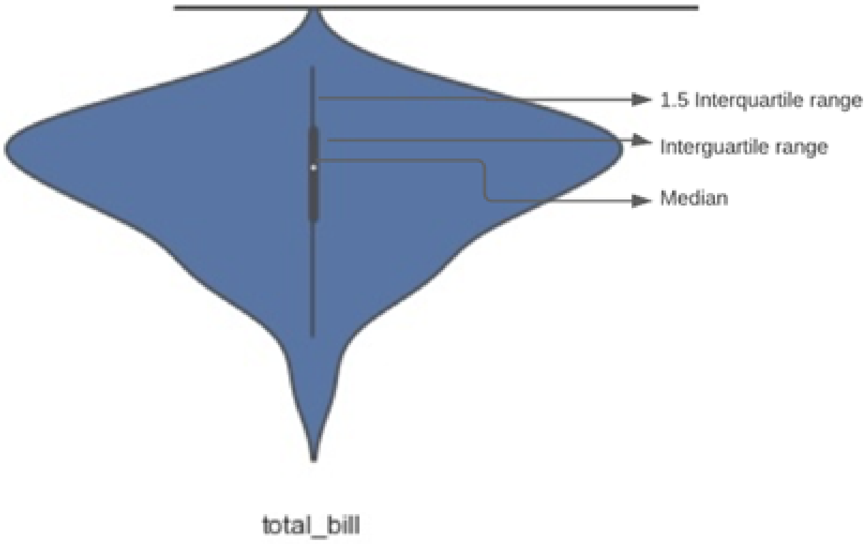
1.3 Plotting (Joint & Violin Plots)

- **Histogram**:
  px.histogram(df[col])
  sns.distplot(df[col], kde = Bool) for kernel density estimate

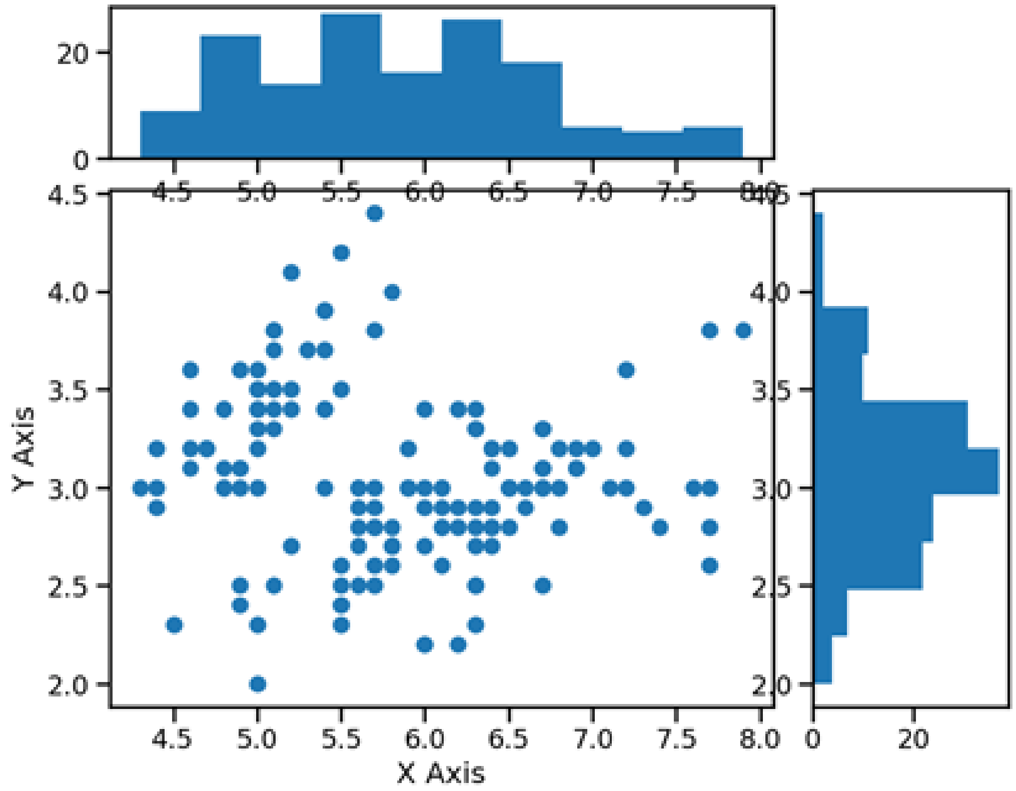
- **Scatter**: px.scatter(df, x=, y=, color=, size=) with size as 3rd param
  px Log axes: as a constructor, log_x or log_y = True

- Box and whisker plot: Median marked with green line; box borders describe first and third quartiles
  px.box(df, y=, color=)

- **Violin Plot**: combines a box+whisker with a kernel density plot – the width is its probability density
  px.violin(df, points=all) where points=all shows distribution of points (x-axis has no value)



- **Joint (or "Marginal") Plot**: contrasts two distributions to form a scatter plot
  px.scatter(df, x=, y=, marginal_x = , marginal_y =) where marginal constructor can be histogram, box, etc.



- Heat Map: the color indicates magnitude for locations the x-y plane
  px.density_heatmap(df, x=, y=, marginal_x=, marginal_y=)
  sns.jointplot(df, x=, y=, kind='hex')

1.4 Data Cleaning

- Steps to Data Cleaning
  - Eliminate duplicates
  - Resolve structural errors (capitalization, naming conventions, typos, inconsistencies, mislabeling, etc.)
  - Filter outliers
  - Handle missing NaN or null data
  - Validate that data is useful and correct
- **String patterns**: df[col].str.contains(" ")
  .startswith(" ")

.replace(" "," ") – make sure to backslash BRACKETS
.upper() or .lower()
NOTE that you can use .str. commands on df.columns too

- **Convert vals to int64**: pd.to_numeric(df[col])
- **Show unique value counts**: df[col].value_counts()