The Pandas Library

* Python library for data manipulation and analysis
* used with Jupyter Notebook
  + read raw data
  + clean and inspect data
  + merge data sets
  + perform calculations
  + create tables
* Data Wrangling
  + change data layout and structure
  + organize data by type
  + format rows or columns
  + sort or group data

The Pandas Library Advantages

* very flexible
* optimized for performance
* functions exceed Excel

A **development environment,** or **virtual environment,** is an isolated, working copy of the coding environment that programmers use to install different versions of software packages for specific projects.

Create a development environment: (In terminal)

conda create -n PythonData python=3.7

(PythonData is the name of the environment)

Activate a development environment:

conda activate PythonData

Delete a development environment:

conda deactivate

conda env remove –name <env\_name>

The command python -m ipykernel install --user --name PythonData tells Python to use the IPython kernel to install the PythonData environment in the Jupyter kernels.

A Pandas **Series** is a one-dimensional, labeled array capable of holding any data type.

df = pd.DataFrame({"column name": column values}), to add a list or Series to a DataFrame.

* Import direct

# Files to load

school\_data\_to\_load = "Resources/schools\_complete.csv"

* Import indirect

# Add the dependencies.

import pandas as pd

import os

# Files to load

school\_data\_to\_load = os.path.join("Resources", "schools\_complete.csv"

read\_csv() function, easier by converting the CSV file to a DataFrame.

school\_data\_df = pd.read\_csv(school\_data\_to\_load)

# Create a Pandas Series from a list.

school\_series = pd.Series(high\_schools)

pandas.DataFrame(), convert the data in this Excel file to a Pandas

Graphical user interface, table

Description automatically generated with medium confidence

df.columns, to get the column names.

df.values, to get the values.

Determine whether there are missing values in large datasets:

* .count(), get a count of the rows for each column.
* .isnull() , "True" for empty rows, "False" for not empty.
* .notnull() , "False" for empty rows, "True" for not empty.

# Determine if there are any missing values in the student data.

student\_data\_df.isnull().sum()

.dropna() , to drop a row with NaNs. .dropna(how='any')

.fillna() , to fill in a row.

df.fillna(0), fill all empty rows with zero.

df.dtypes / df[“column”].dtypes, check the data types of each column.

df.head(), to view the first five rows.

df.tail(), to view the last five rows.

df.tolist()/df[“column”].tolist(), will add items from the column to a list

.split(), split a Python string object on the whitespace

sort\_values() function, sort a DataFrame or Series.

* ascending=False to sort from highest to lowest,
* ascending=True to sort from lowest to highest (default)

set(list), get the unique items in each list.

.strip(), removes any combination of letters and words inside the parentheses.

.replace("Dr.", "")

We can chain the str method with the replace() method:

student\_data\_df["student\_name"].str.replace()

Example: Combine similar offenses

no\_null\_crime\_df = no\_null\_crime\_df.replace(

{"Commercial": "Prostitution", "Assisting or Promoting": "Prostitution"})

We'll merge school\_data\_df and student\_data\_df on a shared column using the merge() method:

# Combine the data into a single dataset.

school\_data\_complete\_df = pd.merge(student\_data\_df, school\_data\_df, on=["school\_name", "school\_name"])

unique() get the unique items in the ["school\_name"] column, will return a n-dimensional array of all the unique values of that column.

To Filter:

# Get all the students who are passing math in a new DataFrame.

passing\_math = school\_data\_complete\_df

[school\_data\_complete\_df ["math\_score"] >= 70]

map(), **function** to substitute each value in a Series with another value:

Column.map("value\_1" : "new\_value\_1", "value\_2" : "new\_value\_2")

**format(), function**  to format a value to a specific format:

# Format by adding a dollar sign, a thousands separator, and one decimal place.

my\_grades.map("${:,.1f}".format)

Reorder the column in the order you want them to appear:

new\_column\_order = ["column2", "column4", "column1"]

(We filtered “column3” out by not including it.)

# Assign a new or the same DataFrame the new column order.

df = df[new\_column\_order]

set\_index, set the index to the school\_name column, and a column with the types.

per\_school\_types = school\_data\_df.set\_index(["school\_name"])["type"]

value\_counts(), to get the number of times appears.

per\_school\_counts = school\_data\_complete\_df["school\_name"].value\_counts()

Example: Check to see if there are any values with mispelled or similar values in "Offense Type".

no\_null\_crime\_df["Offense Type"].value\_counts()

groupby(), function will split an object (like a DataFrame), apply a mathematical operation, and combine the results.

Example: Get the average weight and length membership of the members for each trainer.

average\_weight = new\_df.groupby(["Trainer"]).mean()

average\_weight.sort\_values(by="Membership (Days)", ascending=False)

index.name = None, to remove the index.

describe(), it will return the summary descriptive statistics of rows:

* number as count
* average as mean
* standard deviation as std
* minimum value as min
* 25th percentile as 25%
* 50th percentile as 50%
* 75th percentile as 75%
* maximum value as max

\*\*\*The min is 578 and the max is 655.

We don't want the lowest bin to be $578 because there is only one school at $578.

There are four schools that spend less than $585, so $585 will be our lowest bin.

Also, because the standard deviation is about 30, we will increase the bins by $30, up to $675.

The four bins will be: $585, $615, $645, and $675.

We can write the ranges for the bins as: spending\_bins = [0, 585, 615, 645, 675].

cut(), function segments and sorts data values into bins.

pd.cut(df, ranges) creates a new DataFrame on the given ranges.

df.index.name = None, Remove the index

Example: We only like good movies, so find those that scored over 7.

good\_movie = movie\_df.loc[movie\_df["IMDB"] < 7, ['FILM', 'IMDB', 'IMDB\_user\_vote\_count']]

(前面是filter，后面是需要add的columns)

Example: Convert the "duration" column's values to numeric

converted\_ufo["duration"] = converted\_ufo.loc[:, "duration"].astype(float)

Example: Figure out the minimum and maximum views for a TED Talk

ted\_df["views"].min()

ted\_df["views"].max()