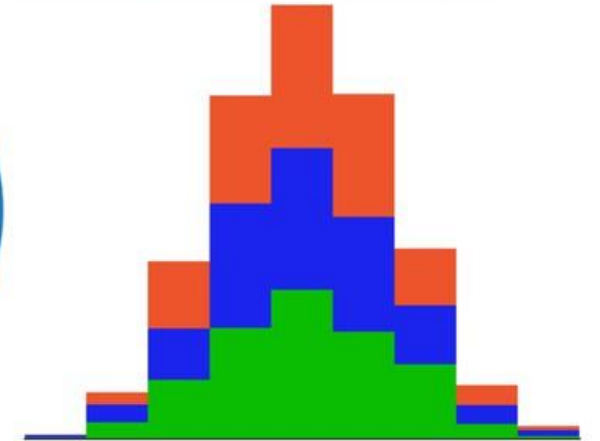
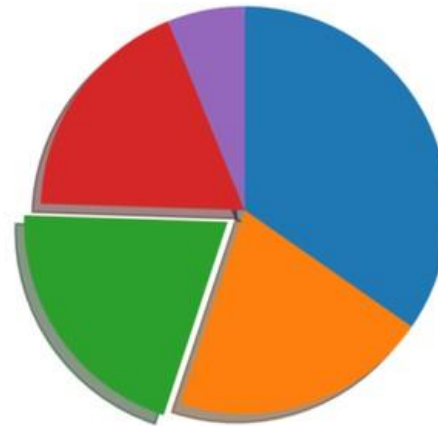


DATA SCIENCE TOOLS, PART 2: PANDAS, MATPLOTLIB

Pandas



matplotlib



PRE-QUIZ: HOW MUCH DO YOU ALREADY KNOW ABOUT PANDAS?

- What is python pandas?
- How is a pandas DataFrame different than a NumPy array?
- Name three unique operations (i.e., methods) you can do with pandas?
- How do you read from or write to a csv file using pandas?
- Keep this Quiz and see if you can fill-in any missing questions during the discussion today



WHAT IS PANDAS?

- Definition
- Series
- DataFrame
- Conditional Selection
- Iris Flower Data Set

PANDAS DEFINITION

- Open-Source software library written for Python
- Pandas derived from the term “**panel data**” from econometrics
- Data structures and operations for manipulating numerical tables and time series
- DataFrame patterned after R DataFrame
- DataFrame is a 2-Dimensional structure built as a combination of Series arrays with a shared index
- Built on NumPy
- Originally released 11 Jan 2008.

[https://en.wikipedia.org/wiki/Pandas_\(software\)](https://en.wikipedia.org/wiki/Pandas_(software))



PANDAS SERIES

- A **pandas Series** is a one-dimensional labeled array in Python, capable of holding data of any type (integers, floats, strings, Python objects, etc.).

Key features of a pandas Series:

- **Indexing:** Each element in the Series has a corresponding index, which allows for easy access and manipulation of data.
- **Homogeneous:** The Series can hold elements of the same type (though mixed types are possible, but uncommon).
- **Data alignment:** The index labels allow for automatic alignment of data when performing operations, which can be useful when dealing with time series or other labeled data.

python

```
import pandas as pd

# Create a pandas Series
data = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])

print(data)
```

This will output:

```
a    10
b    20
c    30
d    40
dtype: int64
```

PANDAS DATAFRAME

- A **Pandas DataFrame** is a two-dimensional, labeled data structure in Python, similar to a table or spreadsheet, that stores data in rows and columns. Each column in a DataFrame can have a different data type (e.g., integers, floats, strings, etc.).

Key features of a DataFrame:

- **Rows and Columns:** Like a table, with rows representing individual records and columns representing variables or features.
- **Labels (Index):** Rows and columns can have labels (names), making it easy to access, slice, or manipulate data.
- **Data Handling:** It can handle missing data and supports arithmetic operations, data filtering, aggregation, and transformation.
- **Data Input/Output:** Can read and write data from various file formats (e.g., CSV, Excel, SQL, etc.).
- Multiple Series Objects that share an index

Example:

```
python
import pandas as pd

# Create a DataFrame
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'San Francisco', 'Los Angeles']
}

df = pd.DataFrame(data)

print(df)
```

This code would output:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles

IRIS FLOWER DATA SET

- The ***Iris* flower data set** was made famous by the British statistician and biologist Ronald Fisher in 1936.
- It is sometimes called Anderson's Iris data set because Edgar Anderson collected the data to quantify the morphologic variation of Iris flowers of three related species.
- The dataset contains a set of 150 records with five attributes: sepal length, sepal width, petal length, petal width and species.
- The iris data set is widely used for machine learning purposes. The dataset is included in Python in the machine learning library [scikit-learn](https://scikit-learn.org/).

https://en.wikipedia.org/wiki/Iris_flower_data_set#External_links

iris setosa



petal sepal

iris versicolor



petal sepal

iris virginica



petal sepal

```
from sklearn.datasets import load_iris

iris = load_iris()
iris
```

This code gives:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
               [4.9, 3., 1.4, 0.2],
               [4.7, 3.2, 1.3, 0.2],
               [4.6, 3.1, 1.5, 0.2],...
               ], dtype=float64),
 'target': array([0, 0, 0, ..., 1, 1, 1, ..., 2, 2, 2, ...
                  ], dtype=int64),
 'target_names': array(['setosa', 'versicolor', 'virginica'],
                        dtype='<U10'),
 ...}
```

MOST COMMONLY USED PANDAS OPERATIONS

Data Wrangling with pandas Cheat Sheet <http://pandas.pydata.org>

Pandas API Reference Pandas User Guide

Creating DataFrames

	a	b	c
1	4	5	6
2	5	8	11
3	6	9	12

```
df = pd.DataFrame({
    "a": [4, 5, 6],
    "b": [7, 8, 9],
    "c": [10, 11, 12]},
    index=[1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame([
    [4, 7, 10],
    [5, 8, 11],
    [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

	a	b	c
1	4	5	6
2	5	8	11
3	6	9	12

```
df = pd.DataFrame({
    "a": [4, 5, 6],
    "b": [7, 8, 9],
    "c": [10, 11, 12]},
    index = pd.MultiIndex.from_tuples(
        [(1, 'd'), (2, 'e'), (3, 'f')],
        names=['id', 'v']))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

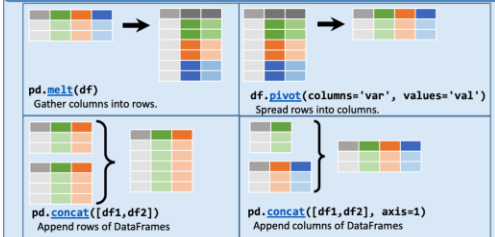
```
df = (pd.melt(df)
      .rename(columns={
          'variable': 'var',
          'value': 'val'})
      .query('val >= 200'))
```

Tidy Data – A foundation for wrangling in pandas



Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Reshaping Data – Change layout, sorting, reindexing, renaming



```
df.sort_values('mpg')
# Order rows by values of a column (low to high).

df.sort_values('mpg', ascending=False)
# Order rows by values of a column (high to low).

df.rename(columns={'y': 'year'})
# Rename the columns of a DataFrame

df.sort_index()
# Sort the index of a DataFrame

df.reset_index()
# Reset index of DataFrame to row numbers, moving index to columns.

df.drop(columns=['Length', 'Height'])
# Drop columns from DataFrame
```

Subset Observations - rows

```
df[df.Length > 7]
# Extract rows that meet logical criteria.

df.drop_duplicates()
# Remove duplicate rows (only considers columns).

df.sample(frac=0.5)
# Randomly select fraction of rows.

df.sample(n=10)
# Randomly select n rows.

df.nlargest(n, 'value')
# Select and order top n entries.

df.nsmallest(n, 'value')
# Select and order bottom n entries.

df.head(n)
# Select first n rows.

df.tail(n)
# Select last n rows.
```

Subset Variables - columns

```
df[['width', 'length', 'species']]
# Select multiple columns with specific names.

df['width'] or df.width
# Select single column with specific name.

df.filter(regex='regex')
# Select columns whose name matches regular expression regex.
```

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
# Select and order top n entries.

df.query('Length > 7 and Width < 8')
# Select and order top n entries.

df.query('Name.str.startswith("abc")')
# Select rows meeting logical condition, and only the specific columns.
```

	Logic in Python (and pandas)	Logic in SQL
<	Less than	df.column < value
>	Greater than	df.column > value
==	Equals	df.column == value
<=	Less than or equals	df.column <= value
>=	Greater than or equals	df.column >= value

	regex (Regular Expressions) Examples
^	Matches strings containing a period '.'
Lengths	Matches strings ending with word 'Length'
^Sepal	Matches strings beginning with the word 'Sepal'
^x[1-5]\$	Matches strings beginning with 'x' and ending with 1,2,3,4,5
^(?!Species)\$	Matches strings except the string 'Species'

Cheatsheet for pandas (<http://pandas.pydata.org>) originally written by Ivo Luning, www.ivo.nl, inspired by www.ivo.nl Data Wrangling Cheatsheet

Summarize Data

```
df['w'].value_counts()
# Count number of rows with each unique value of variable

len(df)
# # of rows in DataFrame.

df.shape
# Tuple of # of rows, # of columns in DataFrame.

df['w'].nunique()
# # of distinct values in a column.

df.describe()
# Basic descriptive and statistics for each column (or GroupBy).
```

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

```
sum()
# Sum values of each object.

count()
# Count non-NA/null values of each object.

median()
# Median value of each object.

quantile([0.25, 0.75])
# Quantiles of each object.

apply(function)
# Apply function to each object.
```

Group Data



```
df.groupby(by="col")
# Return a GroupBy object, grouped by values in column named "col".

df.groupby(level="ind")
# Return a GroupBy object, grouped by values in index level named "ind".
```

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

```
size()
# Size of each group.

agg(function)
# Aggregate group using function.
```

Windows

```
df.expanding()
# Return an Expanding object allowing summary functions to be applied cumulatively.

df.rolling(n)
# Return a Rolling object allowing summary functions to be applied cumulatively.
```

Handling Missing Data

```
df.dropna()
# Drop rows with any column having NA/null data.

df.fillna(value)
# Replace all NA/null data with value.
```

Make New Columns

```
df.assign(Area=lambda df: df.Length*df.Height)
# Compute and append one or more new columns.

df['Volume'] = df.Length*df.Height*df.Depth
# Add single column.

pd.cut(df.col, n, labels=False)
# Bin column into n buckets.
```

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

```
max(axis=1)
# Element-wise max.

min(axis=1)
# Element-wise min.

clip(lower=-10, upper=10)
# Trim values at input thresholds.

abs()
# Absolute value.
```

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

```
shift(1)
# Copy with values shifted by 1.

rank(method='dense')
# Ranks with no gaps.

rank(method='min')
# Ranks. Ties get min rank.

rank(pct=True)
# Ranks rescaled to interval [0, 1].

rank(method='first')
# Ranks. Ties get to first value.
```

Plotting

```
df.plot.hist()
# Histogram for each column.

df.plot.scatter(x='w', y='h')
# Scatter chart using pairs of points.
```

Combine Data Sets

```
df + bdf
# Join matching rows from bdf to df.
```

Standard Joins

```
pd.merge(df, bdf, how='left', on='x1')
# Join matching rows from bdf to df.
```

```
pd.merge(df, bdf, how='right', on='x1')
# Join matching rows from df to bdf.
```

```
pd.merge(df, bdf, how='inner', on='x1')
# Join data. Retain only rows in both sets.
```

```
pd.merge(df, bdf, how='outer', on='x1')
# Join data. Retain all values, all rows.
```

Filtering Joins

```
adf[adf.x1.isin(bdf.x1)]
# All rows in adf that have a match in bdf.

adf[~adf.x1.isin(bdf.x1)]
# All rows in adf that do not have a match in bdf.
```

```
ydf + zdf
# Join matching rows from ydf to zdf.
```

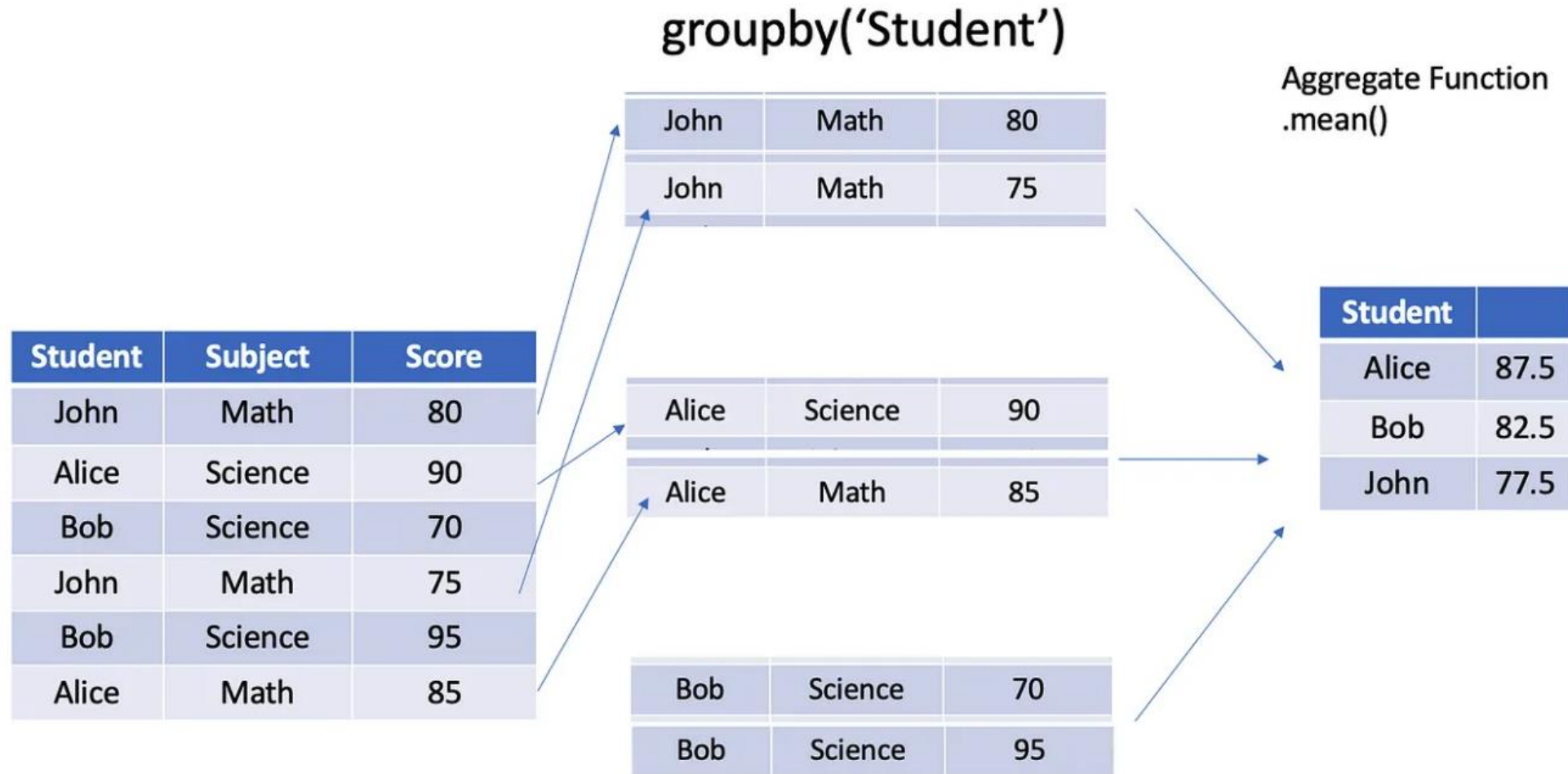
Set-like Operations

```
pd.merge(ydf, zdf)
# Rows that appear in both ydf and zdf (Intersection).
```

```
pd.merge(ydf, zdf, how='outer', indicator=True)
# Rows that appear in either or both ydf and zdf (Union).
```

```
pd.merge(ydf, zdf, how='left', indicator=True)
# Rows that appear in ydf but not zdf (Setdiff).
```


GROUPBY GENERAL CONCEPT



QUIZ: HOW MUCH DO YOU NOW KNOW ABOUT PANDAS?

- What is Python Pandas?
- How is a Pandas DataFrame different than a NumPy array?
- Name three unique operations (i.e., methods) you can do with Pandas?
- How do you read from or write to a csv file using Pandas?



PYTHON CODE UPDATE W/ CURRENT PANDAS NOTEBOOKS!!!

- Notebook for practice on Canvas