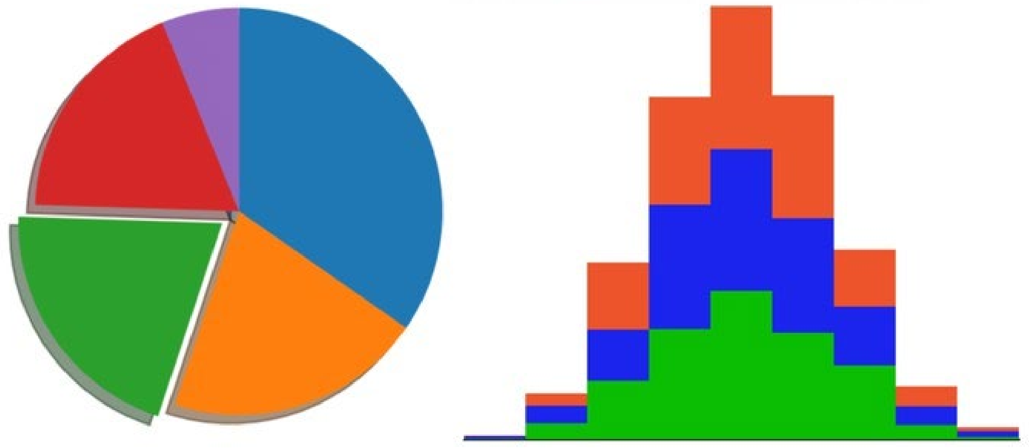
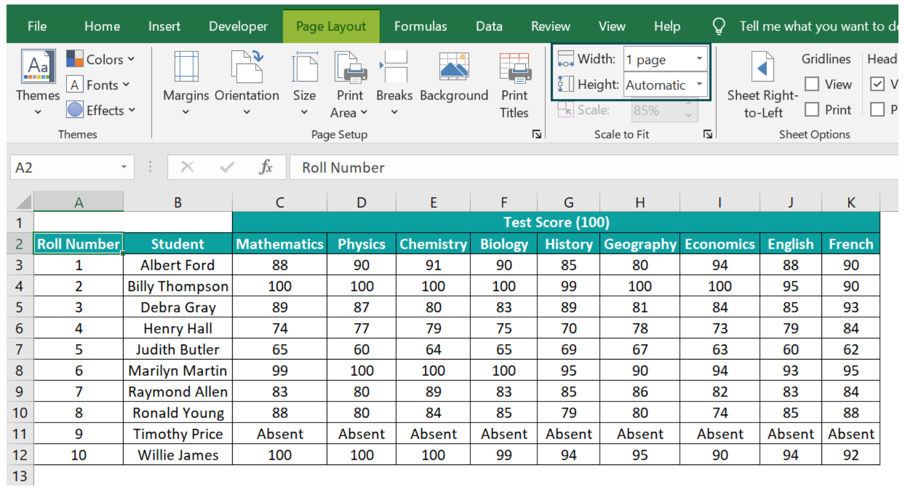
DATA SCIENCE TOOLS, PART 2: PANDAS, MATPLOTLIB



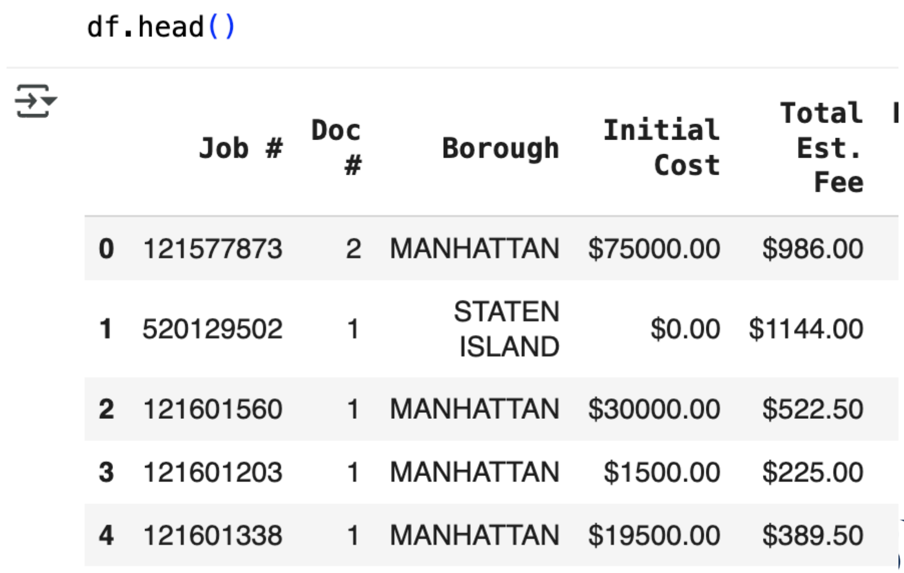
WHAT IS PANDAS?

- Think of **pandas** as **Excel for Python**.
- If you've ever used **Excel** or **Google Sheets** — adding columns, filtering rows, making summaries — pandas does the same, but with **code**.



A2 Roll Number

		Test Score (100)								
Roll Number	Student	Mathematics	Physics	Chemistry	Biology	History	Geography	Economics	English	French
1	Albert Ford	88	90	91	90	85	80	94	88	90
2	Billy Thompson	100	100	100	100	99	100	100	95	90
3	Debra Gray	89	87	80	83	89	81	84	85	93
4	Henry Hall	74	77	79	75	70	78	73	79	84
5	Judith Butler	65	60	64	65	69	67	63	60	62
6	Marilyn Martin	99	100	100	100	95	90	94	93	95
7	Raymond Allen	83	80	89	83	85	86	82	83	84
8	Ronald Young	88	80	84	85	79	80	74	85	88
9	Timothy Price	Absent	Absent	Absent	Absent	Absent	Absent	Absent	Absent	Absent
10	Willie James	100	100	100	99	94	95	90	94	92



```
df.head()
```

	Job #	Doc #	Borough	Initial Cost	Total Est. Fee
0	121577873	2	MANHATTAN	$75000.00	$986.00
1	520129502	1	STATEN ISLAND	$0.00	$1144.00
2	121601560	1	MANHATTAN	$30000.00	$522.50
3	121601203	1	MANHATTAN	$1500.00	$225.00
4	121601338	1	MANHATTAN	$19500.00	$389.50

PANDAS DEFINITION

- Open-Source software library written for Python
- Pandas derived from the term "**pan**el **da**ta" from econometrics
- Data structures and operations for manipulating numerical tables and time series
- 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. The current stable release is **version 2.3.2**, released August 21, 2025.

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

BYU

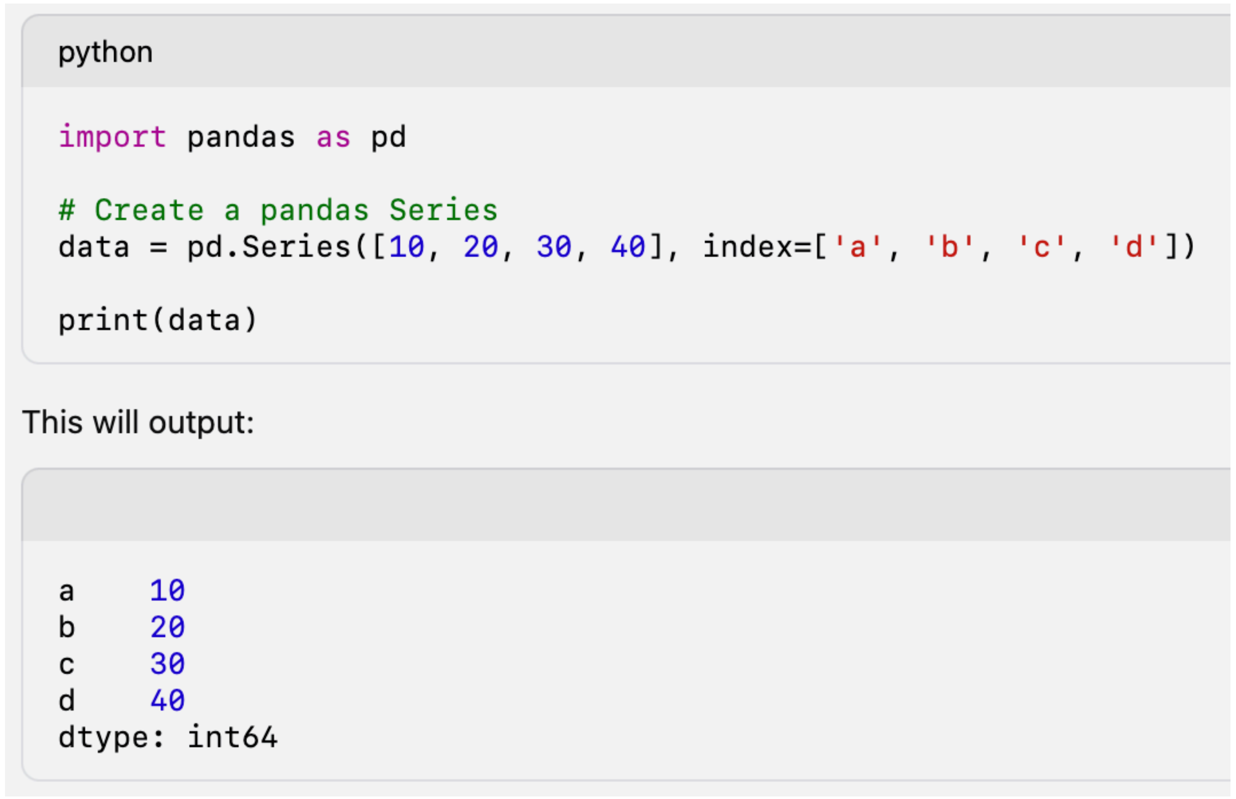
PANDAS SERIES

A **pandas Series** is a one-dimensional labeled array, 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. Default is numeric indexing.

• **Homogeneous**: The Series can hold elements of the same type (though mixed types are possible, but uncommon and not recommended practice).



python

```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:

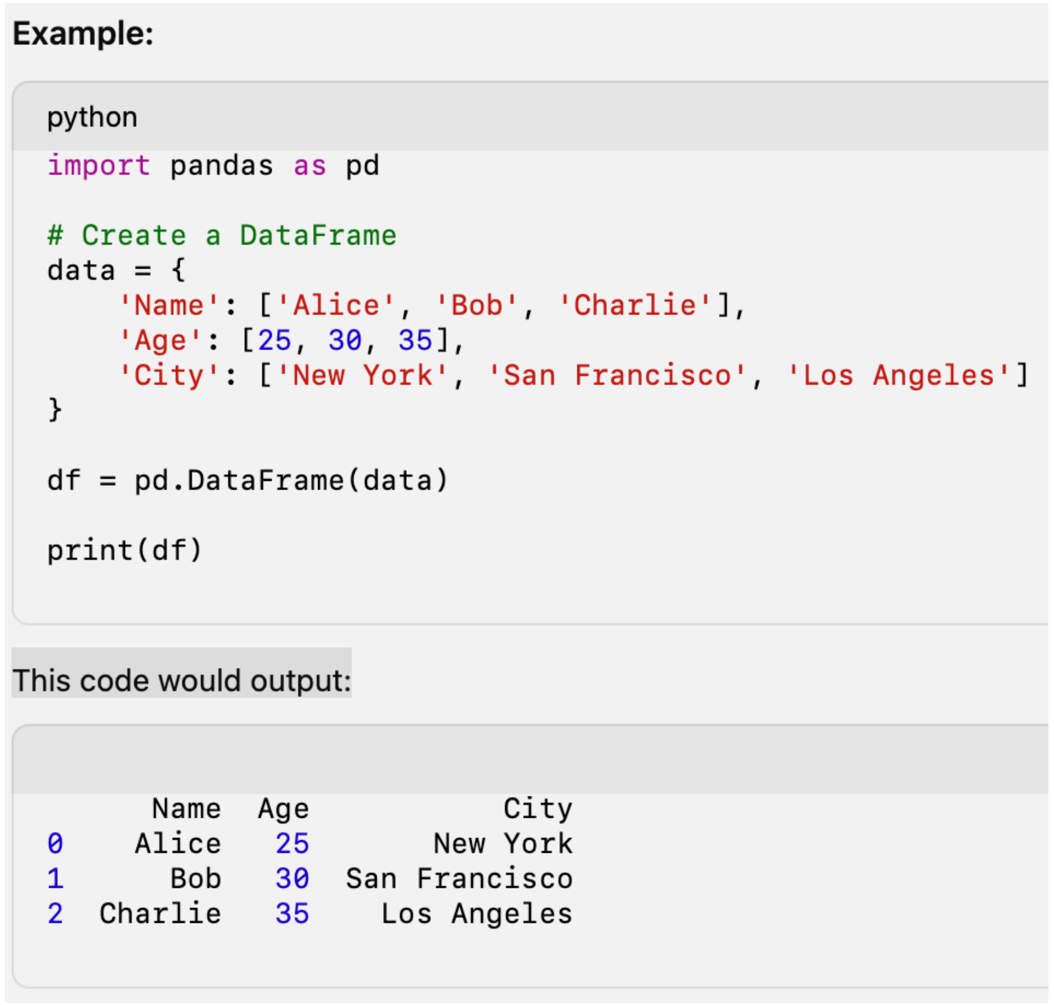
```
a    10
b    20
c    30
d    40
dtype: int64
```

PANDAS DATAFRAME

- Defined as multiple **Series** objects that share an index
- 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:** Rows and columns can have labels (names), making it easy to access, slice, or manipulate data. Index and Column for row/column labels
- **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.).



Example:

python
```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
```

BYU

SUBSETTING DATA

- *Subsetting data* involves choosing specific rows and columns from a dataframe according to labels, indices, and slices.
- A single column can be selected by using the label of the desired column. Ex: Using the country dataset assigned to the variable country, the Population column can be selected using the country['Population'] or country.Population. Multiple columns can also be selected by using an array of strings. Ex: country[['Name', 'Population']]
- The iloc(x,y) method for a dataframe is used to select an individual element using an index location, where x is the row and y is the column. Ex: country.iloc[0,1] returns the element in row 0 and column 1. The colon character : is used in slice notation to select multiple rows or columns. Ex: country.iloc[:5,1:3] returns rows before row 5 and columns 1 thru 2.
- The loc(x,y) method can also be used to subset data, but y, in this case, is an array of column labels, instead of an integer or a range of integers. Ex:
  Both country.iloc[:7,1:3] and country.loc[:6,['Continent','Population']] give the same results.

CONDITIONAL FILTERING

- Comparison and logical operators can be used to subset data. When these operators are used, only rows for which the expression is true will be returned. Ex: country[country['Population'] > 100000] will display rows whose 'Population' column values are greater than 100,000.

MISSING VALUES

There are several methods to deal with missing values in the data frame:

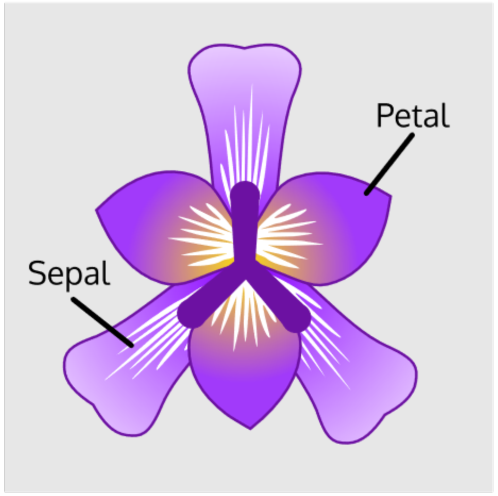
df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

BYU

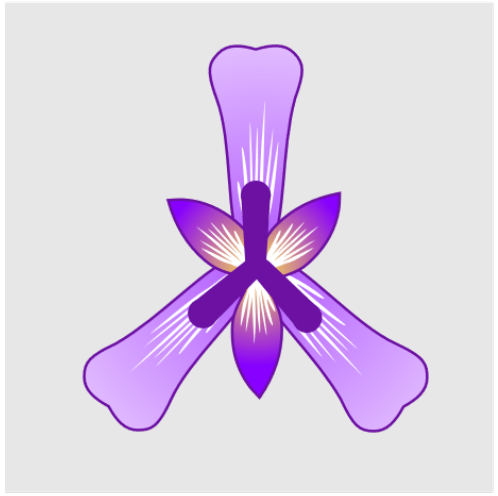
IRIS FLOWER DATA SET

- 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 teaching machine learning. The dataset is included in Python in the machine learning library scikit-learn.

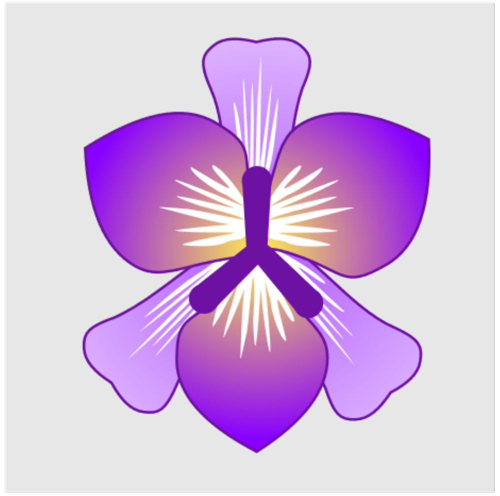
https://en.wikipedia.org/wiki/Iris_flower_data_set#External_links



Iris Versicolor



Iris Setosa



Iris Virginica

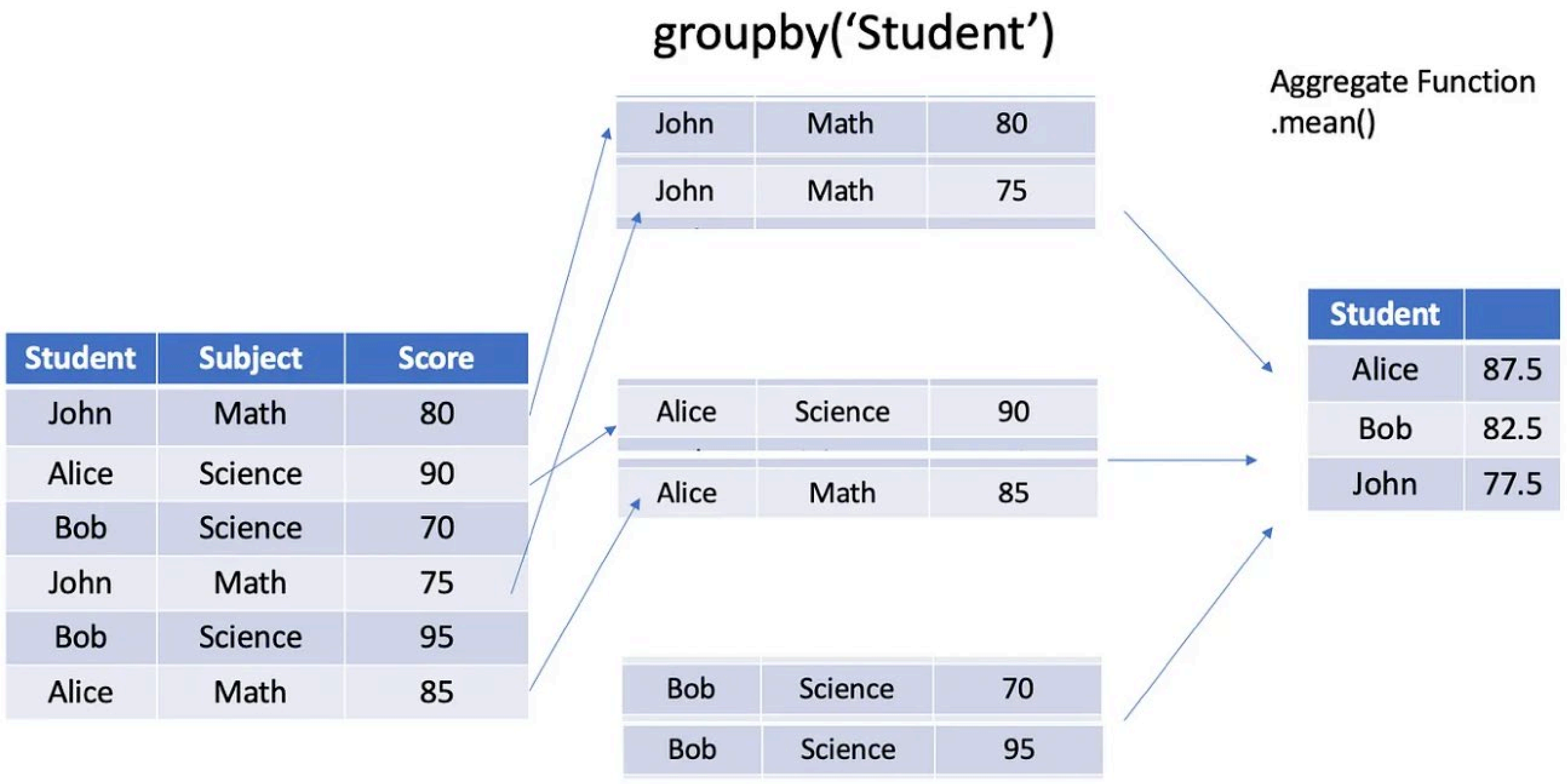
```
from sklearn.datasets import load_iris

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],...
'target': array([0, 0, 0, ... 1, 1, 1, ... 2, 2, 2, ...
'target_names': array(['setosa', 'versicolor', 'virginica'],
dtype='<U10'),
...}
```

BYU

GROUPBY GENERAL CONCEPT



groupby('Student')

Aggregate Function
.mean()

Student	Subject	Score
John	Math	80
Alice	Science	90
Bob	Science	70
John	Math	75
Bob	Science	95
Alice	Math	85

John	Math	80
John	Math	75

Alice	Science	90
Alice	Math	85

Bob	Science	70
Bob	Science	95

Student	
Alice	87.5
Bob	82.5
John	77.5

QUIZ: HOW MUCH DO YOU NOW KNOW ABOUT PANDAS?

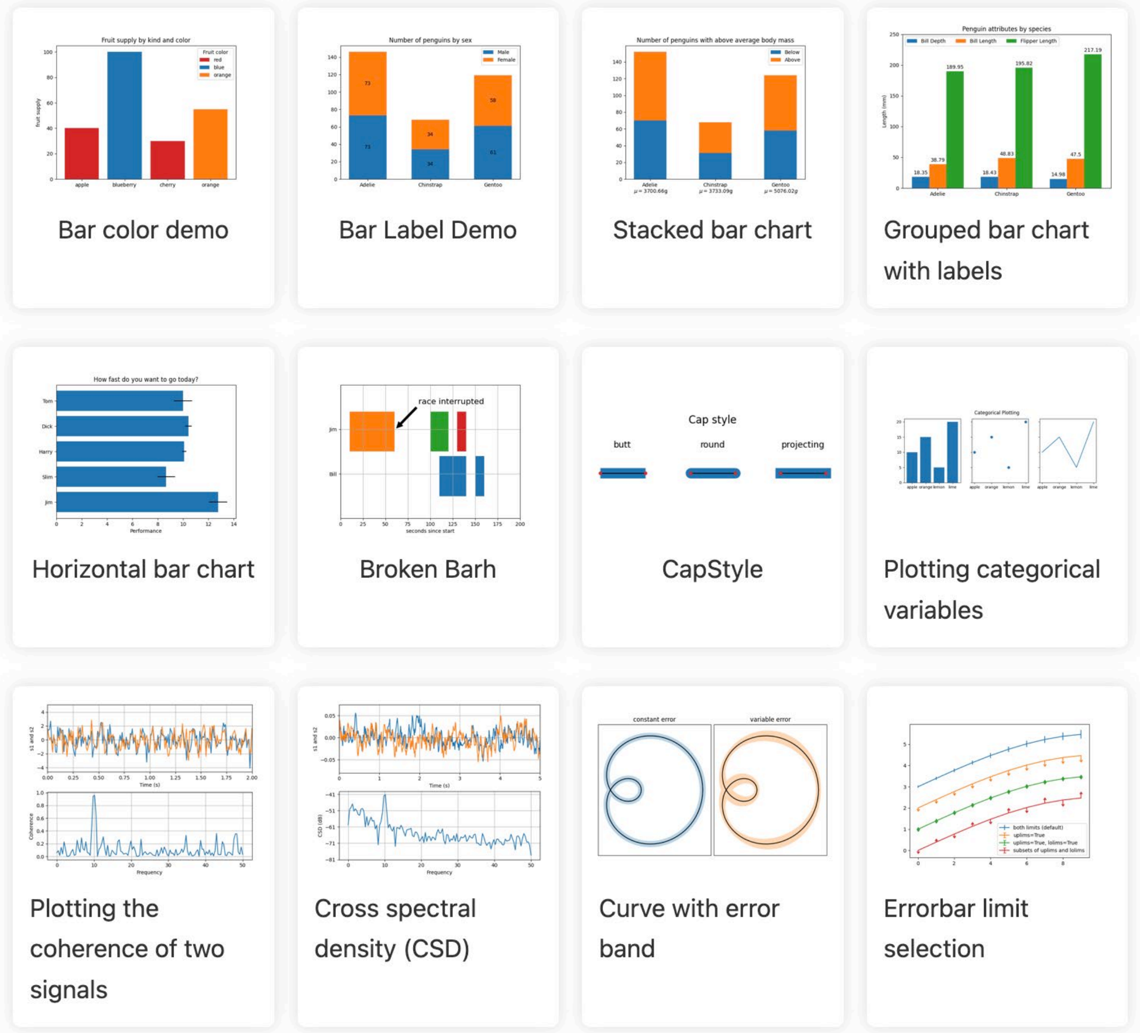
- What is Python Pandas?
- How is a Pandas DataFrame different than a NumPy array?
- Name three unique operations (i.e., methods) you can do with Pandas?
- How do you read from or write to a csv file using Pandas?

MATPLOTLIB

Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations. Matplotlib can produce a variety of plots, such as:

- Line plots
- Bar charts
- Scatter plots
- Histograms
- Pie charts
- 3D plots

It works closely with other libraries like NumPy for numerical computations and Pandas for handling data structures.

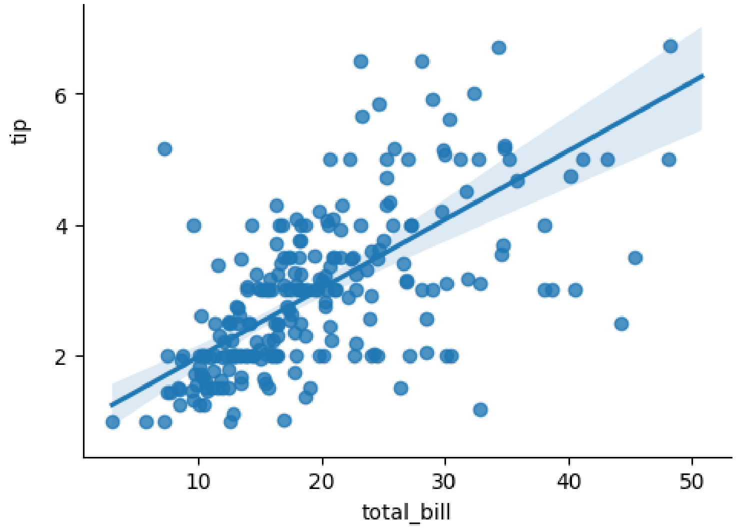


BYU

SEABORN

- Seaborn is a Python data visualization library built on top of Matplotlib, designed to make it easier to create informative and aesthetically pleasing statistical graphics.
- Seaborn integrates closely with Pandas data structures, which makes it especially powerful for working with data frames and structured data.
- **Plot Types**: Seaborn supports many types of plots, such as:
  - Line plots
  - Bar plots
  - Scatter plots
  - Heatmaps
  - **Pair plots (My favorite!)**
  - Box plots
  - Violin plots

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Load an example dataset from Seaborn
tips = sns.load_dataset("tips")

# Create a scatter plot with a linear fit
sns.lmplot(x="total_bill", y="tip", data=tips)

# Display the plot
plt.show()
```



BYU

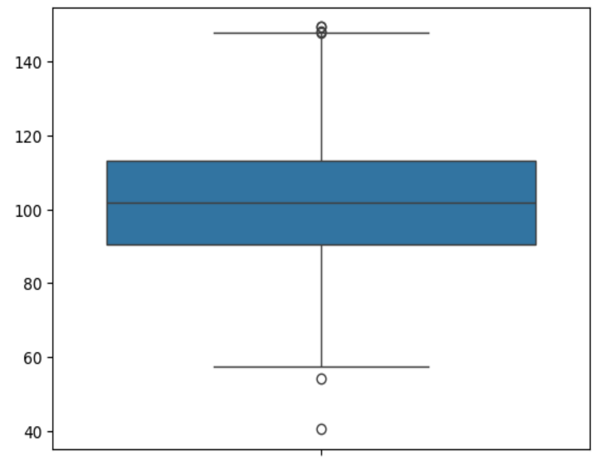
BOX AND WHISKER PLOT

A **boxplot** is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum. It helps to show the spread and skewness of the data, highlighting potential outliers.

- **Key Elements :**
- 1. **Box**: Spans from Q1 to Q3 (the interquartile range, IQR).
- 2. **Median**: A line inside the box showing the middle of the dataset.
- 3. **Whiskers**: Lines extending from Q1 to the minimum and Q3 to the maximum values within 1.5 times the IQR.
- 4. **Outliers**: Data points that fall outside the whiskers.

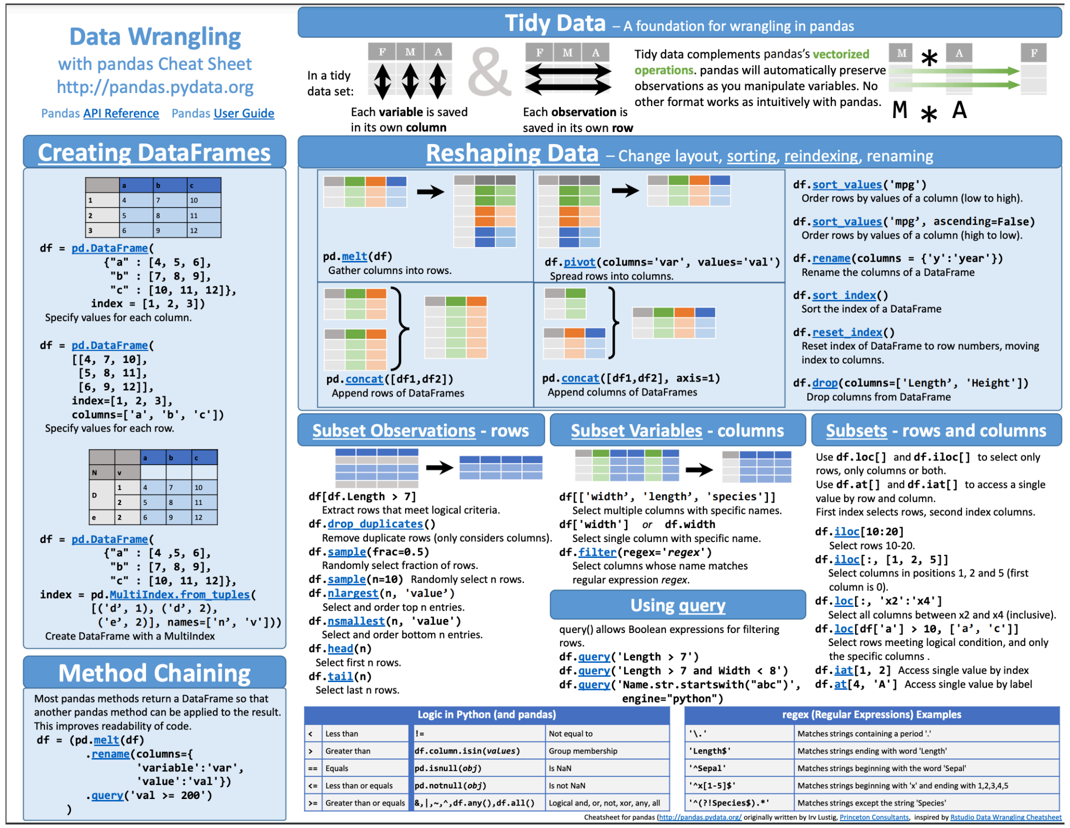
By combining these elements, a box plot quickly provides insights into the data's **distribution, variability, and any unusual observations** (like outliers).

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Generate random data
np.random.seed(10)
data = np.random.normal(100, 20, 200)

# Create a boxplot using seaborn
sns.boxplot(data=data)

# Display the plot
plt.show()
```



APPENDIX

MOST COMMONLY USED PANDAS OPERATIONS



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

Pandas API Reference   Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

In a tidy data set: Each **variable** is saved in its own **column** & Each **observation** is saved in its own **row**

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

M * A

Creating DataFrames

	a	b	c
1	4	7	10
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
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

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

Reshaping Data – Change layout, sorting, reindexing, renaming

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`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')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
              engine="python")
```

Subsets - rows and columns

Use **df.loc[]** and **df.iloc[]** to select only rows, only columns or both.
Use **df.at[]** and **df.iat[]** to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]`
Select rows 10-20.
`df.iloc[:, [1, 2, 5]]`
Select columns in positions 1, 2 and 5 (first column is 0).
`df.loc[:, 'x2':'x4']`
Select all columns between x2 and x4 (inclusive).
`df.loc[df['a'] > 10, ['a', 'c']]`
Select rows meeting logical condition, and only the specific columns .
`df.iat[1, 2]` Access single value by index
`df.at[4, 'A']` Access single value by label

Logic in Python (and pandas)

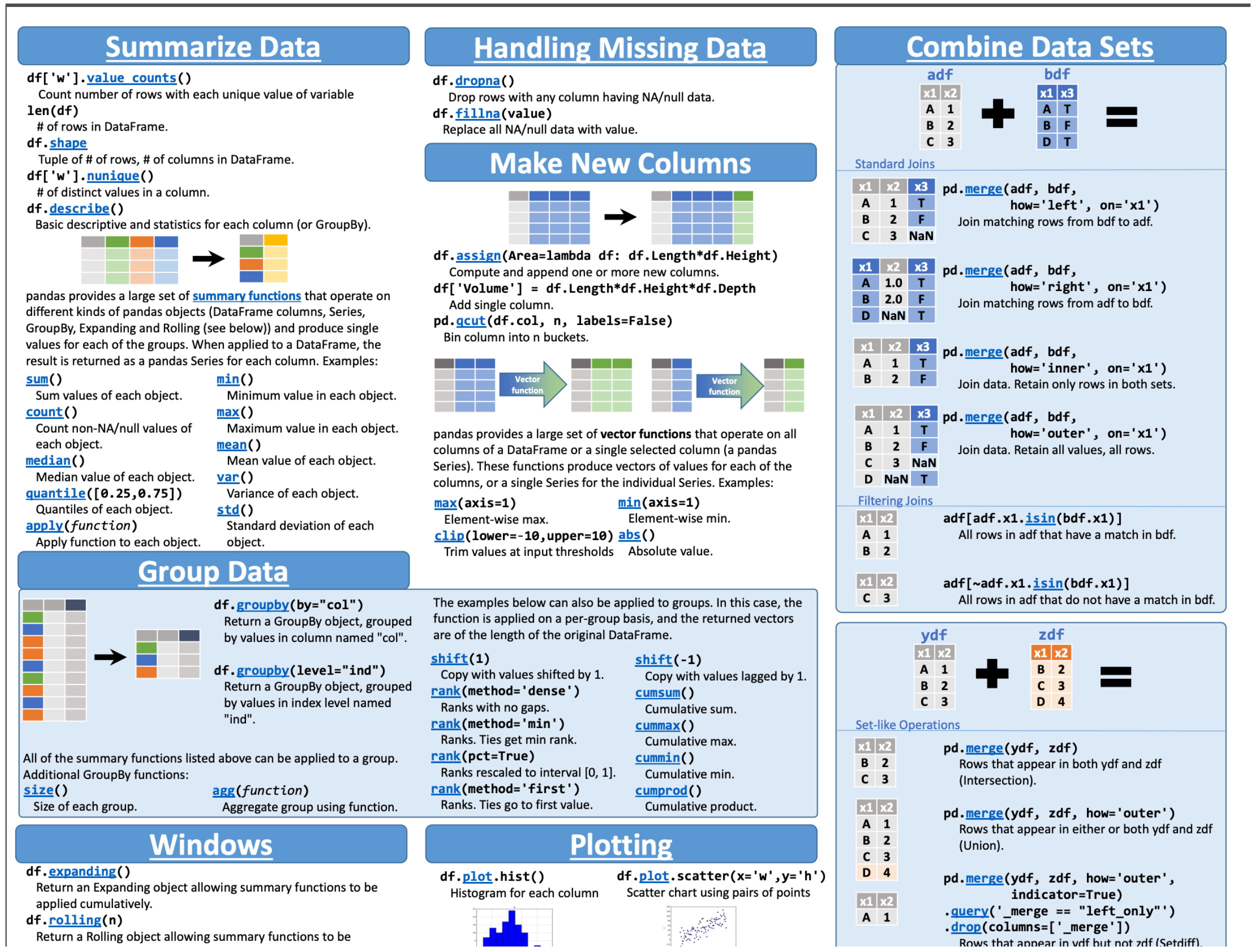
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	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 Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf



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.
`min()`
Minimum value in each object.
`max()`
Maximum value in each object.
`mean()`
Mean value of each object.
`var()`
Variance of each object.
`std()`
Standard deviation of each object.

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.qcut(df.col, n, labels=False)`
Bin column into n buckets.

Vector function   Vector function

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.

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.

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 go to first value.
`shift(-1)`
Copy with values lagged by 1.
`cumsum()`
Cumulative sum.
`cummax()`
Cumulative max.
`cummin()`
Cumulative min.
`cumprod()`
Cumulative product.

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

Plotting

`df.plot.hist()`
Histogram for each column
`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points

Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

bdf

x1	x3
A	T
B	F
D	T

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

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

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

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

x1	x2	x3
A	1	T
B	2	F

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

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

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

Filtering Joins

x1	x2
A	1
B	2

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

x1	x2
C	3

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

ydf

x1	x2
A	1
B	2
C	3

zdf

x1	x2
B	2
C	3
D	4

Set-like Operations

x1	x2
B	2
C	3

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

x1	x2
A	1
B	2
C	3
D	4

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

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
            indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).