→ **Pandas** used for data analysis & manipulation

import pandas as pd

1D array-like object that can hold many datatypes

pd.Series(data = [,] , index = [' ', ' ']

controllable index

groceries . shape (,) groceries. index

groceries . ndim pandas dimension .values

groceries . size no. of elements in (check if index exists

not in

→ **Access & Delete**

- .loc( ) we use a labelled index
- .iloc( ) we use a numerical index
- drop( ) removes item from the series without altering original series
- drop(___ , inplace = True) alters the series

→ **Arithmetic Operations**

we can import numpy to use arithmetic operations on pandas

→ **DataFrame** a powerful spreadsheet

pd.DataFrame( )

row_labels → series index NaN → not a number

column_labels → dictionary keys

must study → SQL
→ Power BI
→ Tableau

if there was no index pandas we will use a numerical one

- values
- shape
- ndim
- size
- columns

→ Access DataFrames

items [[' _ ']] access whole column

items . loc [[' _ ']] access whole row

items [' _ '] [' _ '] acces element in specified row & column

column row

→ Adding

columns dataframe [' '] = [,]

new column data

rows 1) create a new dataframe with new row

2) append to previous dataframe

columns

with specific data dataframe [' _ '] = dataframe [' '] [1 :]

for special rows

insert columns anywhere database . insert (loc , ' _ ' , data)

→ Delete

columns . pop ( )

columns & rows . drop ( )

axis → 1 axis → 0

→ cut → query

represent very diverse data range with small parameters

→ Rename

database . rename (index = { previous : new }

. rename (columns = { previous : new }

. setIndex (' _ ')

→ Group by

pd . groupby ([' _ ' , ' _ '])

Dealing with Nan

- isnull() returns True when Nan is found
- count() returns non-Nan values
- dropna(axis =) remove columns axis=1 with Nan values in them
  inplace= rows axis=0
- fillna(__) replace Nans with a specific value
- fillna(method = 'ffill', axis =) fill Nan with value from previous column row
- fillna(method = 'backfill', axis=) fill Nan with value " next "
- interpolate(method = 'linear') estimate a value from Nan

Files

pd.read_csv(' __ ')

- head() 1st 5 rows
- tail() last 5 rows
- describe()

[ ].describe()

- corr() info about correlation

- groupby[' __ '][' __ ']

Data Analysis

Jupyter Notebooks

Markdown Cells → allows you to include links / style text

Headers → # Header 1

Header 2

emphasis → italic _____ or *_____*

→ bold __ or **__**

create a codeblock ```

```

Math Expressions $$

$$

Lists

unordered - * +

ordered .

Blockquotes >

>

>> ← nesting

Links

link text [](). inline link ( ) < >

reference style link

[1] : _____

Images

! [alt]( )

See shortcuts H → in command mode

Timing Code (Magic Words)

→ time it how long does it take for a function to run

% timeit single line %% timeit whole cell

Embedding visualization images with text & code % matplotlib inline

Debugging in Notebook %pdb
```