neueda TRAINING

Python for Financial Data Analysis

AGENDA

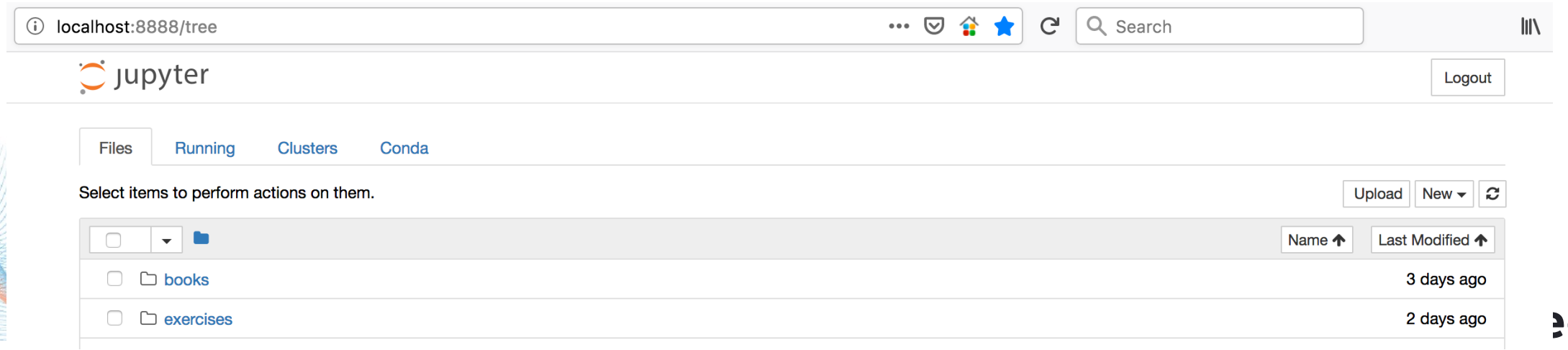
- Jupyter Notebooks and Python Fundamentals
- Overview of Data Analysis & Pandas Fundamentals
- Data Preparation & Cleaning
- Time Series Data
- Visualization
- Time Series Data
- Use-Cases and Data Challenges

PYTHON ESSENTIALS

Jupyter Notebooks

- Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages

- How to run Jupyter:
  - In a terminal (*cmd* on Windows) type: **jupyter notebook** or **jupyter lab**
  - It will show a Notebook Dashboard with the folders you have (from where you run the jupyter command)

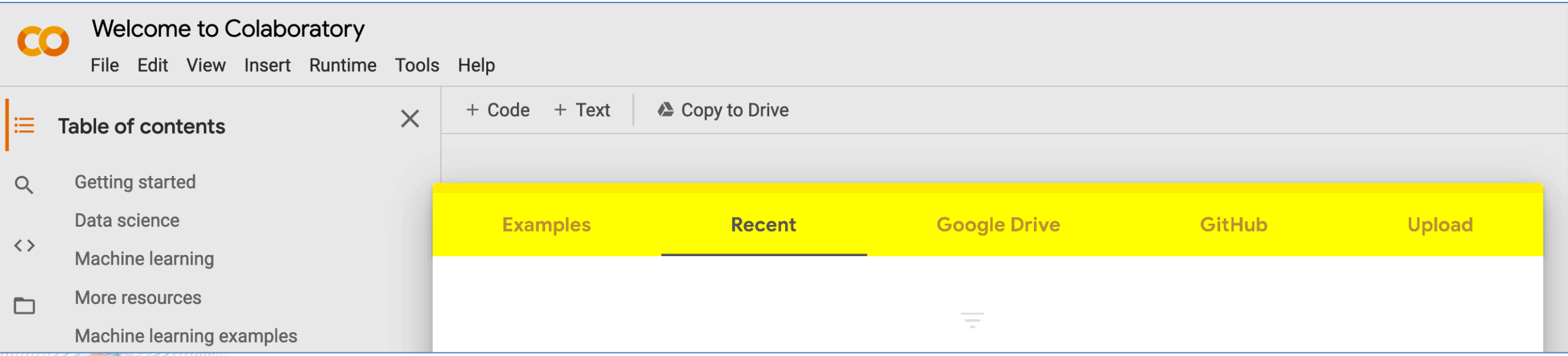


https://jupyter.org/
https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/execute.html

eda TRAINING

Google Colab

Go to https://colab.research.google.com/



Interactive help – notebooks

At any cell you can "ask" interactive help about a particular object, function, etc.
Type **?** followed by the object, method or attribute you want to know more about

airports is a dataframe

cumsum one method and you asked help to know more about this method

```
?airports.cumsum
```

```
Signature: airports.cumsum(axis=None, skipna=True, *args, **kwargs)
Docstring:
Return cumulative sum over a DataFrame or Series axis.

Returns a DataFrame or Series of the same size containing the cumulative
sum.
```

neueda TRAINING

Python built-in types preview

Object type	Example literals/creation
Numbers	1234, 3.1415, 3+4j, Ob111, Decimal(), Fraction()
Strings	'spam', “Bob's”, b'a\x01c', u'sp\xc4m'
Lists	[1, [2, 'three'], 4.5], list(range(10))
Dictionaries	{'food': 'spam', 'taste': 'yum'}, dict(hours=10) }
Tuples	(1, 'spam', 4, 'U'), tuple('spam'), namedtuple
Files	open('eggs.txt'), open(r'C:\ham.bin', 'wb')
Sets	set('abc'), {'a', 'b', 'c'}
Other core types	Booleans, None
Program unit types	Functions, modules, classes
Implementation-related types	Compiled code, stack tracebacks

Learning Python, 5th Edition. Mark Lutz, O'Reilly
http://www.java2s.com/example/python-book/unicode-strings.html

Python operators

Operator	Meaning
+	Add two operands or unary plus
-	Subtract right operand from the left or unary minus
*	Multiply two operands
/	Divide left operand by the right one (always results into float)
//	Floor division - division that results into whole number adjusted to the left in the number line
%	Modulus - remainder of the division of left operand by the right
**	Exponent - left operand raised to the power of right

Logical Operators	It either returns True or False according to the condition.
not	True if operand is false (complements the operand)
and	True if both the operands are true
or	True if either of the operands is true

https://www.programiz.com/python-programming/operators

Python operators

Comparison Operators	Comparison operators are used to compare values. It either returns True or False according to the condition.
>	Greater that - True if left operand is greater than the right
<	Less that - True if left operand is less than the right
>=	Greater than or equal to - True if left operand is greater than or equal to the right
<=	Less than or equal to - True if left operand is less than or equal to the right
!=	Not equal to - True if operands are not equal
==	Equal to - True if both operands are equal
Identity Operators	
is	True if the operands are identical (refer to the same object)
is not	True if the operands are not identical (do not refer to the same object)

https://www.programiz.com/python-programming/operators

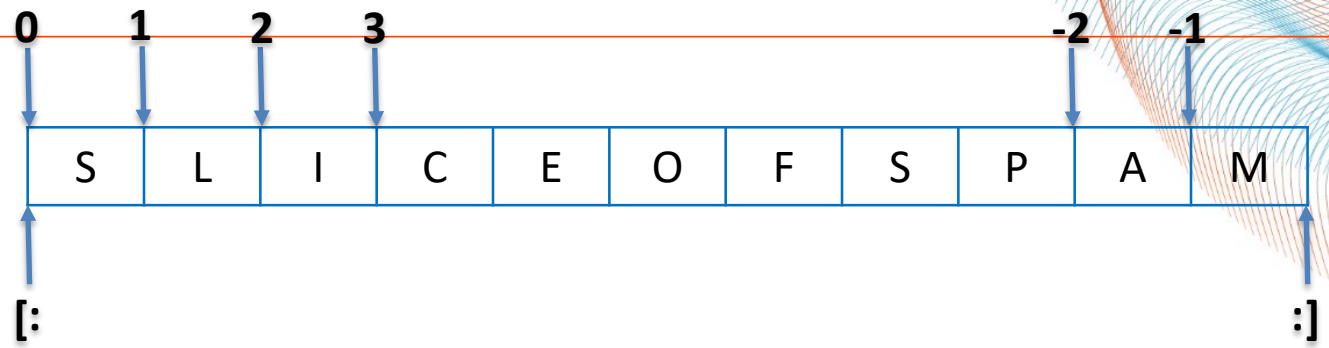
Python Operators

Membership Operators	
in	True if value/variable is found in the sequence
not in	True if value/variable is not found in the sequence

Bitwise Operators	Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name. For example, 2 is 10 in binary and 7 is 111.
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
~	Bitwise NOT
>> or <<	Bitwise right or left shift

https://www.programiz.com/python-programming/operators

Offsets and Slices



- As sequences strings support operations that assume a positional ordering among items
- In Python indexes are coded as offsets from the front and start from 0

Python

```
>>> S = 'shrubbery'
>>> S[0]                # The first item in S, indexing is by zero-based position
's'

>>> S[1]                # The second item from the left
'h'
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]        # Everything but the last
'Spa'
>>> S[:3]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]        # Everything but the last
'Spa'
>>> S[:3]         # Same as S[0:3]
'Spa'
>>> S[:-1]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]        # Everything but the last
'Spa'
>>> S[:3]         # Same as S[0:3]
'Spa'
>>> S[:-1]        # Everything but the last
'Spa'
>>> S[:]
?
```

Python

```
>>> S = 'Spam'    # Make a 4-character string, and assign it to a variable S
>>> S[1:3]        # Slice of S from offsets 1 through 2(not 3 – started in 0)
'pa'
>>> S[1:]         # Everything past the first (1:len(S))
'pam'
>>> S[0:3]        # Everything but the last
'Spa'
>>> S[:3]         # Same as S[0:3]
'Spa'
>>> S[:-1]        # Everything but the last
'Spa'
>>> S[:] # All of S as a top-level copy (0:len(S))
'Spam'
```

Strings concatenation

- Concatenation with the + sign (join two strings into a new string)
- Notice how the plus sign (+) means different things for different objects[1]
  - Addition for numbers
  - Concatenation for strings

Python

```
>>> S = 'Spam'
>>> S + 'xyz'                  # Concatenation
'Spamxyz'

>>> name = 'Monty'
>>> last_name = 'Python'
>>> name + last_name
'MontyPython'
```

[1] This is a general property of Python that is known as ***polymorphism***

Lists

Python

```
>>> L = [123, 'spam', 4.56]    # A list of three different-type objects
>>> L
[123, 'spam', 4.56]
>>> len(L) # Number of items in the list
3
```

Python

```
>>> L[0]     # Indexing by position
123
>>> L[:-1]    # Slicing a list returns a new list
[123, 'spam']
```

Type-specific operations

write a dot after the object
. and press **Tab**

```
>>> L = [123, 'spam', 1.23]
>>> L.
```

append clear copy count extend index insert pop remove reverse sort

Python

```
>>> M = ['bob', 'aa', 'cc']
>>> M.sort()    # Order list – by default orders the list by ascending fashion
>>> M           # Changed the original list
['aa', 'bob', 'cc']
```

Python

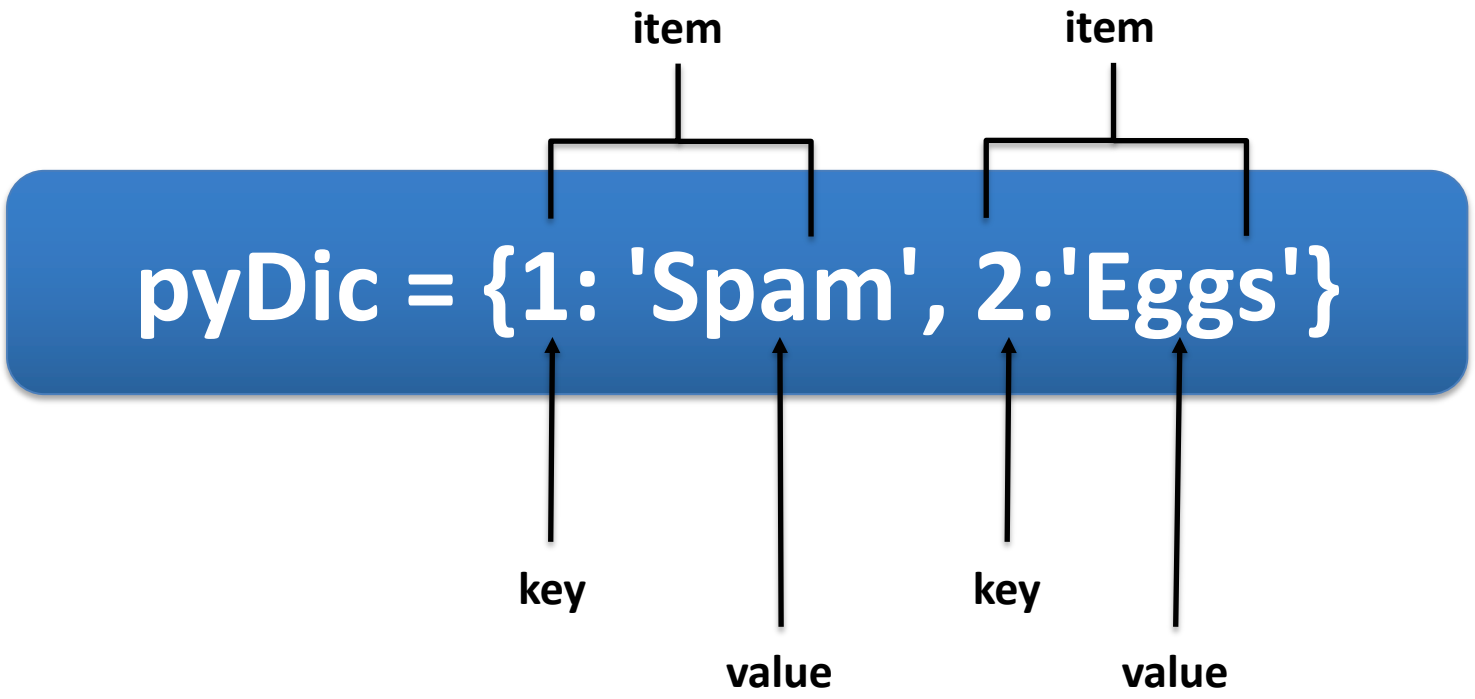
```
>>> M.sort()
>>> M.reverse()   # If we want the list in descending fashion
>>> M             # Changed the original list
['cc', 'bob', 'aa']
```

Reverse does not order the list. Only *reverses* it.

neueda TRAINING

Dictionaries

- Dictionaries (alike Lists) are one of the most flexible built-in data types in Python
- In Dictionaries items are stored and fetched by *key* instead of by positional *offset* (Lists)



Dictionaries: Create, fetch and change values

Python

```
>>> D = {'food': 'Spam', 'quantity': 4, 'color': 'pink'}
>>> D  # Dictionary used to describe the properties of something
{'quantity': 4, 'food': 'Spam', 'color': 'pink'}
```

Python

```
>>> D['food']  # Fetch value of key 'food'
'Spam'
```

Python

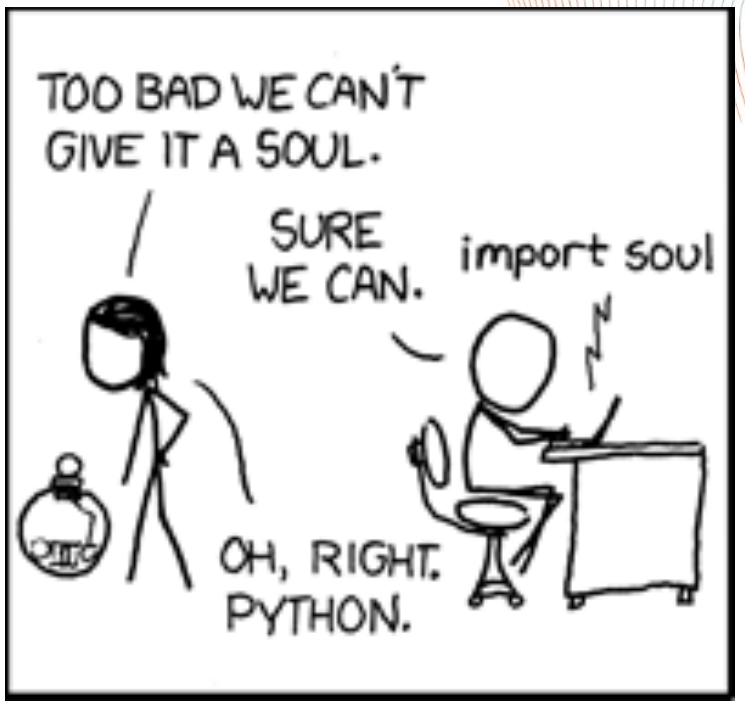
```
>>> D['quantity']  += 1    # Add 1 to 'quantity' value
>>> D
{'quantity': 5, 'food': 'Spam', 'color': 'pink'}
```

Go to Jupyter Lab

Create your first notebook and explore Python ☺

imports

- Python source code that is saved with *.py* extension is a module
- No special code or syntax is required to make a file a module
- Other files can access the items by *importing* that module
- **import operations:** load another file and grant access to that file's content



https://xkcd.com/413/

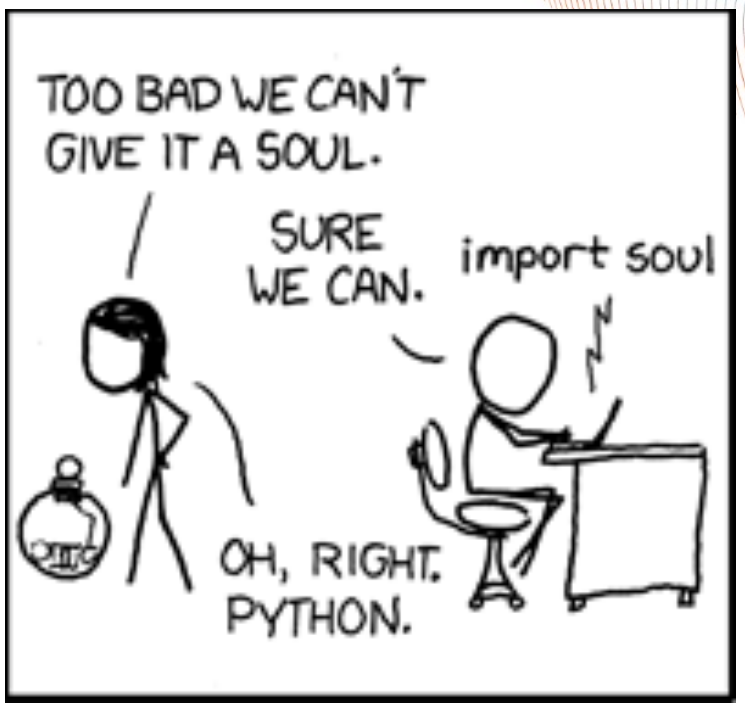
neueda TRAINING

Python – import external libraries

Numpy: is the fundamental package for scientific computing. Provides a high-performance multidimensional array object, and tools for working with these arrays.

Pandas: fast, flexible, and expressive data structures designed to make working with structured and time series data both easy and intuitive.

Matplotlib: produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.



https://xkcd.com/413/

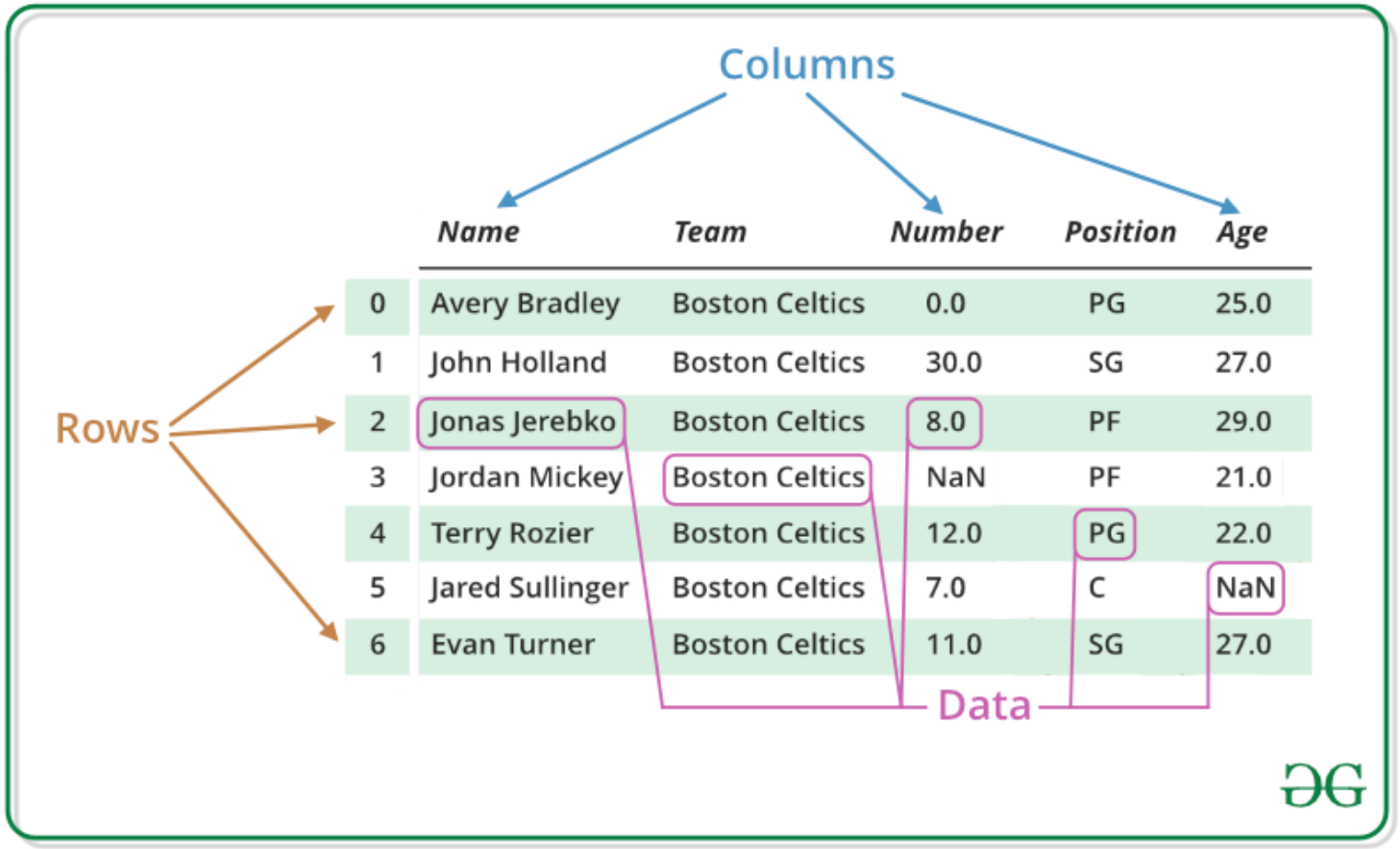
https://numpy.org/
https://pandas.pydata.org/
https://matplotlib.org/

neueda TRAINING

Pandas - DataFrames

2-dimensional labelled data structure with columns of potentially different types
You can think of it like a spreadsheet or SQL table, or a dict of Series objects

- Each **series** is a column in the table
- The **series** must be of the same length
- The **series** may have different data types



https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html#dataframe
https://www.geeksforgeeks.org/python-pandas-dataframe/

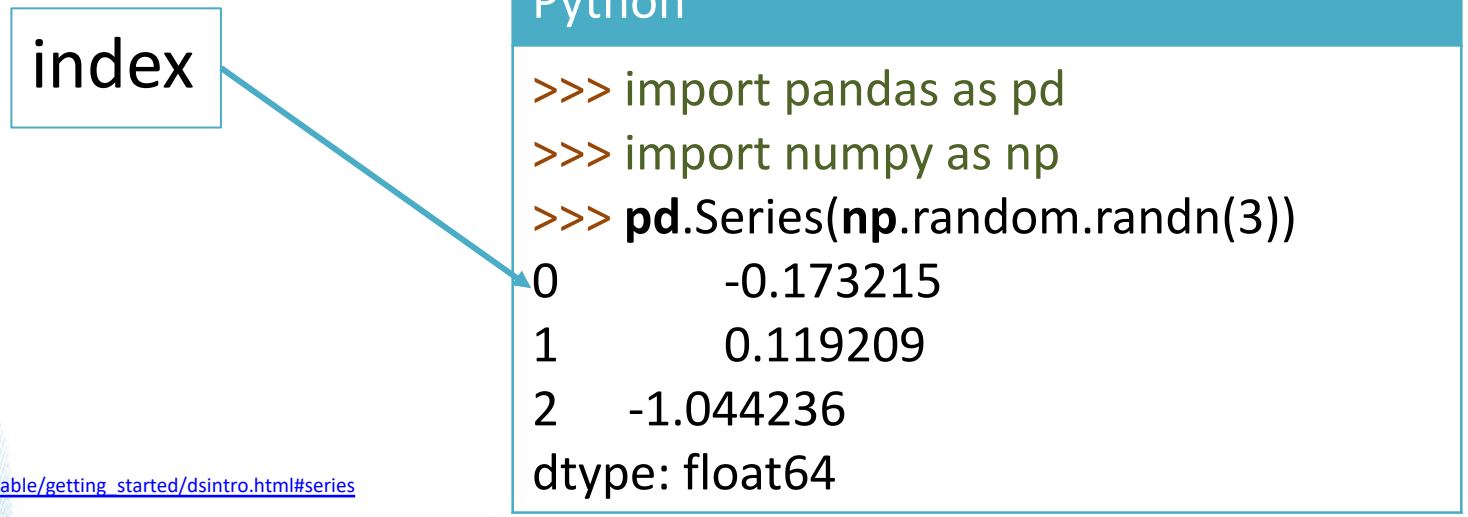
Pandas - Series

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.)

The axis labels are collectively referred to as the index

Series are implemented as Numpy arrays

All values in Series must be of the same type



https://pandas.pydata.org/pandas-docs/stable/getting_started/dsintro.html#series

Load Data into a Pandas Dataframe

Pandas can read various data formats like CSV, JSON, Excel, etc.

```
In [13]: pd.read_
read_clipboard() read_excel()    read_fwf()    read_hdf()    read_json()    read_parquet()  read_sas()    read_sql_query() read_stata()
read_csv         read_feather()  read_gbq()    read_html()   read_msgpack() read_pickle()   read_sql()    read_sql_table() read_table
```

```
import pandas as pd
```

```
airports = pd.read_csv('airports.csv')
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183

https://www.kaggle.com/usdot/flight-delays

Go to notebook

Pandas Getting Started

neueda TRAINING

RECTANGULAR DATA

PANDAS DATAFRAMES

Pandas methods – head and tail

Methods are called with parenthesis

head() and **tail()** will print the first or last rows

In the round brackets you can pass a number of rows or get the *default* of 5 rows

```
airports.head()
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

```
airports.head(1)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404

```
airports.tail()
```

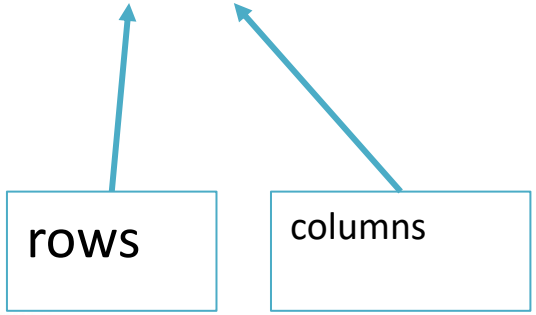
	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
317	WRG	Wrangell Airport	Wrangell	AK	USA	56.48433	-132.36982
318	WYS	Westerly State Airport	West Yellowstone	MT	USA	44.68840	-111.11764
319	XNA	Northwest Arkansas Regional Airport	Fayetteville/Springdale/Rogers	AR	USA	36.28187	-94.30681
320	YAK	Yakutat Airport	Yakutat	AK	USA	59.50336	-139.66023
321	YUM	Yuma International Airport	Yuma	AZ	USA	32.65658	-114.60597

Pandas attributes – shape, dtypes and columns

Attributes are called without parenthesis
Attributes are properties of the object

```
airports.shape
```

```
(322, 7)
```




```
airports.columns
```

```
Index(['IATA_CODE', 'AIRPORT', 'CITY', 'STATE', 'COUNTRY', 'LATITUDE',
       'LONGITUDE'],
      dtype='object')
```

```
airports.dtypes
```

```
IATA_CODE      object
AIRPORT        object
CITY           object
STATE          object
COUNTRY        object
LATITUDE      float64
LONGITUDE     float64
dtype: object
```

neueda TRAINING

Pandas – datatypes

When we read data into a Pandas dataframe the data types are automatically inferred

Pandas dtype	Python type	NumPy type	Usage
object	str	string_, unicode_	Text
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

https://pbpython.com/pandas_dtypes.html

Pandas methods – describe

Generate descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN valuesl

```
airports.dtypes
```

```
IATA_CODE      object
AIRPORT        object
CITY           object
STATE          object
COUNTRY        object
LATITUDE      float64
LONGITUDE     float64
dtype: object
```

Numeric Data

```
airports.describe()
```

	LATITUDE	LONGITUDE
count	319.000000	319.000000
mean	38.981244	-98.378964
std	8.616736	21.523492
min	13.483450	-176.646030
25%	33.652040	-110.839385
50%	39.297610	-93.403070
75%	43.154675	-82.722995
max	71.285450	-64.798560

Object Data(e.g. strings or timestamps)

```
airports[['IATA_CODE', 'AIRPORT', 'CITY', 'STATE', 'COUNTRY']].describe()
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY
count	322	322	322	322	322
unique	322	322	308	54	1
top	MLI	Melbourne International Airport	Columbia	TX	USA
freq	1	1	2	24	322

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html

neueda TRAINING

Pandas methods transpose

Flip Pandas Dataframe

```
airports.head()
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

```
airports.transpose().head(5)
```

	0	1	2	3	4	5	6	7	8	9	...	312	313	314	
IATA_CODE	ABE	ABI	ABQ	ABR	ABY	ACK	ACT	ACV	ACY	ADK	...	TYS	UST	VEL	
AIRPORT	Lehigh Valley International Airport	Abilene Regional Airport	Albuquerque International Sunport	Aberdeen Regional Airport	Southwest Georgia Regional Airport	Nantucket Memorial Airport	Waco Regional Airport	Arcata Airport	Atlantic City International Airport	Adak Airport	...	McGhee Tyson Airport	Northeast Florida Regional Airport (St. August...	Valdez Airport	Val Reg A
CITY	Allentown	Abilene	Albuquerque	Aberdeen	Albany	Nantucket	Waco	Arcata/Eureka	Atlantic City	Adak	...	Knoxville	St. Augustine	Vernal	Val
STATE	PA	TX	NM	SD	GA	MA	TX	CA	NJ	AK	...	TN	FL	UT	
COUNTRY	USA	USA	USA	USA	USA	USA	USA	USA	USA	USA	...	USA	USA	USA	

5 rows × 322 columns

Dataframe index column

```
import pandas as pd
```

```
airports = pd.read_csv('airports.csv')
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183

```
airports_index = pd.read_csv('airports.csv', index_col=0)
```

```
airports_index = pd.read_csv('airports.csv', index_col="IATA_CODE")
```

```
airports_index.head(5)
```

IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

column position or name

neueda TRAINING

Select columns in a dataframe

Using square brackets is the general way we select columns in a DataFrame

Single square brackets returns Series

```
airports['CITY'].head(5)
```

```
0       Allentown
1         Abilene
2     Albuquerque
3        Aberdeen
4          Albany
Name: CITY, dtype: object
```

Double square brackets returns Pandas Dataframe

```
airports[['CITY']].head(5)
```

	CITY
0	Allentown
1	Abilene
2	Albuquerque
3	Aberdeen
4	Albany

Double square brackets returns more than one column in a Pandas Dataframe

```
airports[['CITY', 'COUNTRY']].head(5)
```

	CITY	COUNTRY
0	Allentown	USA
1	Abilene	USA
2	Albuquerque	USA
3	Aberdeen	USA
4	Albany	USA

Pandas method – rename

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

```
airports.rename(columns={'AIRPORT':'AIRPORT_NAME'}).head(2)
```

	IATA_CODE	AIRPORT_NAME	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

Old column name

New column name

neueda TRAINING

Pandas methods – round

It’s often useful to express floating-point-decimal results with a fixed number of digits to the right of the decimal

Python built-in round function

We cannot use DataFrame methods on Series

```
round(flights['ARRIVAL_TIME'].mean(), 2)
```

```
1476.49
```

Pandas round method

```
flights[['ARRIVAL_TIME']].mean().round(2)
```

```
ARRIVAL_TIME    1476.49
dtype: float64
```

```
flights[['ARRIVAL_TIME', 'ARRIVAL_DELAY']].mean().round(2)
```

```
ARRIVAL_TIME     1476.49
ARRIVAL_DELAY       4.41
dtype: float64
```

Filtering Data

Boolean expressions to filter DataFrames based on columns value

```
flights[['ARRIVAL_TIME', 'ARRIVAL_DELAY']].head(2)
```

	ARRIVAL_TIME	ARRIVAL_DELAY
0	408.0	-22.0
1	741.0	-9.0

```
flights[flights.ARRIVAL_TIME<100][['ARRIVAL_TIME', 'ARRIVAL_DELAY']].head(2)
```

	ARRIVAL_TIME	ARRIVAL_DELAY
4376	12.0	264.0
6063	32.0	512.0

Boolean expression

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

```
airports[airports.STATE == 'PA'].head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
24	AVP	Wilkes-Barre/Scranton International Airport	Wilkes-Barre/Scranton	PA	USA	41.33815	-75.72427

Pandas methods – drop

Remove (drop) column(s) from a DataFrame

Notice that **axis=1 is for columns** and **axis=0 is for rows**

drop column name equal to 'IATA_CODE'

drop row with index equal to 1

```
airports.drop(1).head(2)
```

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

```
airports.drop('IATA_CODE', axis=1).head(2)
```

	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

```
airports.drop(1, axis=0).head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919

Add columns to a DataFrame

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819

new column named SOURCE

The column will automatically have the same number of rows of the Dataframe

```
airports['SOURCE'] = 'Kaggle'
```

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE	SOURCE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404	Kaggle
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819	Kaggle

new column named CITY_STATE

Value based on the value of two other columns – concatenation of two String columns

```
airports['CITY_STATE'] = airports['CITY']+'-'+airports['STATE']
```

```
airports.head(2)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE	SOURCE	CITY_STATE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.4404	Kaggle	Allentown-PA
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.6819	Kaggle	Abilene-TX

Pandas method – sort_values

```
airports.head(5)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

```
airports.sort_values('AIRPORT').head(5)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
290	SPI	Abraham Lincoln Capital Airport	Springfield	IL	USA	39.84395	-89.67762
9	ADK	Adak Airport	Adak	AK	USA	51.87796	-176.64603
54	CAK	Akron-Canton Regional Airport	Akron	OH	USA	40.91631	-8

```
airports.sort_values('STATE').head(5)
```

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
108	FAI	Fairbanks International Airport	Fairbanks	AK	USA	64.81368	-147.85967
17	ANC	Ted Stevens Anchorage International Airport	Anchorage	AK	USA	61.17432	-149.99619
89	DLG	Dillingham Airport	Dillingham	AK	USA	59.04541	-158.50334
320	YAK	Yakutat Airport	Yakutat	AK	USA	59.50336	-139.66023
13	AKN	King Salmon Airport	King Salmon	AK	USA	58.67680	-156.64922

Aggregate Functions

You can take aggregates of the values in a Series or all the columns in a DataFrame

```
flights[['ARRIVAL_TIME', 'ARRIVAL_DELAY']].head(2)
```

	ARRIVAL_TIME	ARRIVAL_DELAY
0	408.0	-22.0
1	741.0	-9.0

Series

```
flights['ARRIVAL_TIME'].mean()
```

```
1476.4911879126164
```

DataFrame
One Column

```
flights[['ARRIVAL_TIME']].mean()
```

```
ARRIVAL_TIME    1476.491188
dtype: float64
```

DataFrame
Two columns

```
flights[['ARRIVAL_TIME', 'ARRIVAL_DELAY']].mean()
```

```
ARRIVAL_TIME     1476.491188
ARRIVAL_DELAY       4.407057
dtype: float64
```

There are many more aggregate functions:

- **count()**
- **sum()**
- **median()**
- **mode()**
- **min()**
- **max()**
- **var()**
- **std()**

neueda TRAINING

Pandas method – isnull and notnull

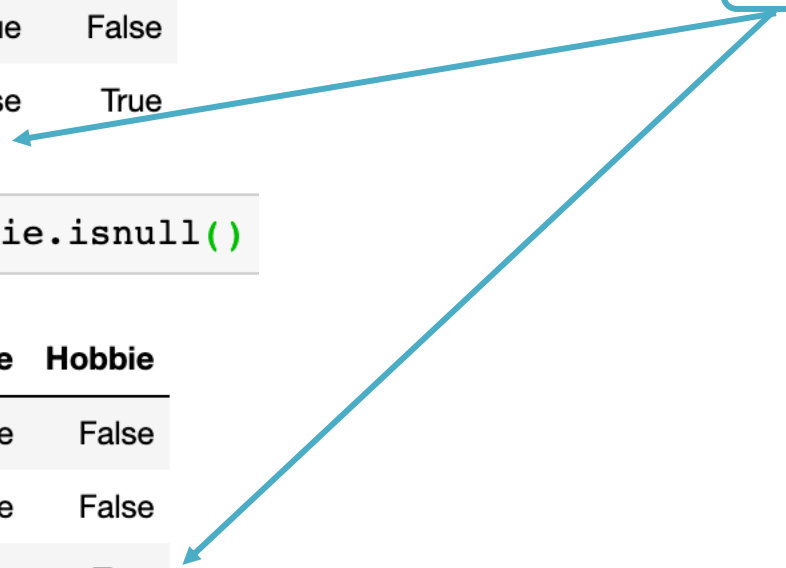
```
nameAgeHobbie.notnull()
```

	Name	Age	Hobbie
0	True	True	True
1	True	True	True
2	True	True	False
3	True	False	True

```
nameAgeHobbie.isnull()
```

	Name	Age	Hobbie
0	False	False	False
1	False	False	False
2	False	False	True
3	False	True	False

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NaN
3	Mark	NaN	BoardGames



neueda TRAINING

Pandas method – dropna

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NaN
3	Mark	NaN	BoardGames

drops all rows with missing values

```
nameAgeHobbie.dropna()
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel

neueda TRAINING

Pandas method – fillna

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NaN
3	Mark	NaN	BoardGames

- **fillna()** - Will detect and empty values and fill them in
- You can give it a value to fill with
- Alternatively, it can fill with values from cells before or after the missing value (backfill or forwardfill)

fills null with a constant value

```
nameAgeHobbie.fillna('NotAvailable')
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NotAvailable
3	Mark	NotAvailable	BoardGames

neueda TRAINING

Pandas method – drop_duplicates and duplicated

duplicated() : indicates whether each row is a duplicate

drop_duplicates() : returns a copy of the DataFrame with the duplicates removed (or inplace=True)

Pandas method - replace

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NotAvailable
3	Mark	NotAvailable	BoardGames

```
nameAgeHobbie.replace('Frank', 'Frankie')
```

	Name	Age	Hobbie
0	Frankie	34	Read
1	Helen	23	Travel
2	Li	34	NaN
3	Mark	NaN	BoardGames

df.replace(to_replace, value) : find and replace specific values

The parameters **to_replace** and **value** can both be either single values or lists of values

Returns a copy so again either use **inplace=True** or catch the returned DataFrame in a new variable

Go to notebook

Clean Your Data
Filters
Data Exploration and Filters

neueda TRAINING

LAB – CREDIT RISK

DATA EXPLORATION

Concatenation
pd.concat

Glues together DataFrames
Dimensions should match along the axis
Default concatenates by row
To concatenate by column use **axis=1** parameter

Initial Pandas DataFrames

1

```
nameAge_df.head()
```

	Name	Age
0	Audrey	40
1	John	12
2	Tango	32
3	Sharon	23

```
nameAge_df2.head()
```

	Name	Age
4	Frank	34
5	Helen	23

```
nameHobbie_df2.head()
```

	Name	Hobbie
4	Frank	Read
5	Helen	Travel

2

```
pd.concat([nameAge_df, nameAge_df2])
```

	Name	Age
0	Audrey	40
1	John	12
2	Tango	32
3	Sharon	23
4	Frank	34
5	Helen	23

3

```
pd.concat([nameAge_df2, nameHobbie_df2[['Hobbie']]], axis=1)
```

	Name	Age	Hobbie
4	Frank	34	Read
5	Helen	23	Travel

neueda TRAINING

Merge

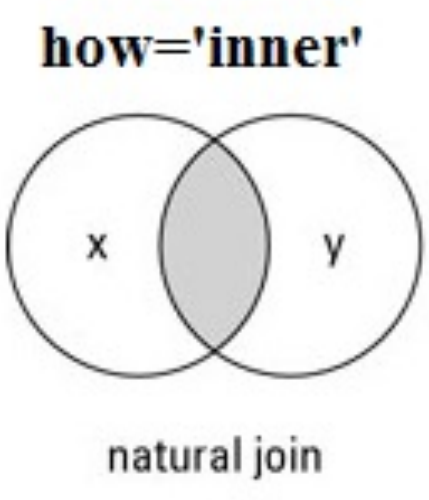
Inner Merge – The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right DataFrames

Left Merge – (or left join) Keep every row in the left DataFrame. Where there are missing values of the “on” variable in the right DataFrame, add empty / NaN values in the result

Right Merge – (right join) Keep every row in the right DataFrame. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result

Outer Merge – A full outer join returns all the rows from the left DataFrame, all the rows from the right DataFrame, and matches up rows where possible, with NaNs elsewhere

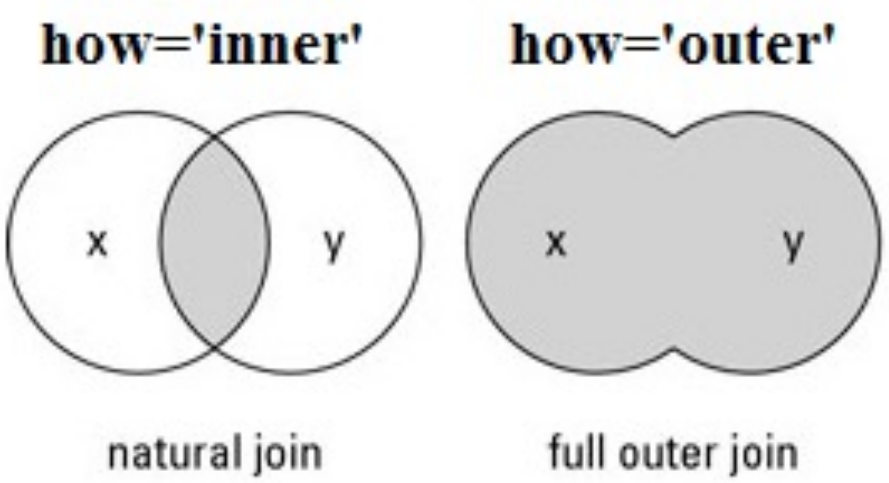
inner, outer, left and right



http://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/

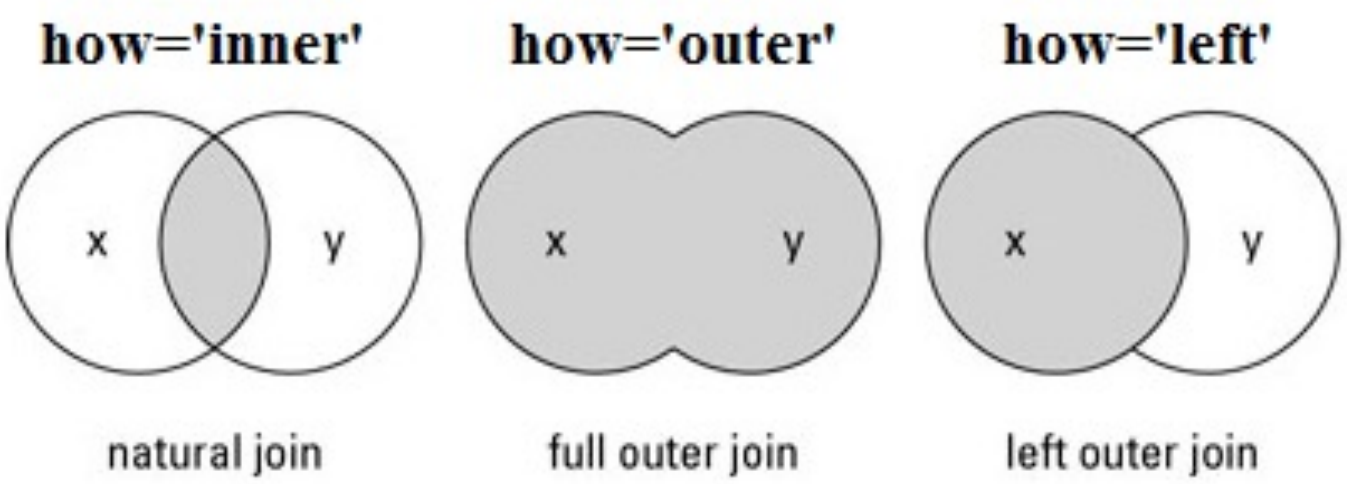
neueda TRAINING

inner, outer, left and right



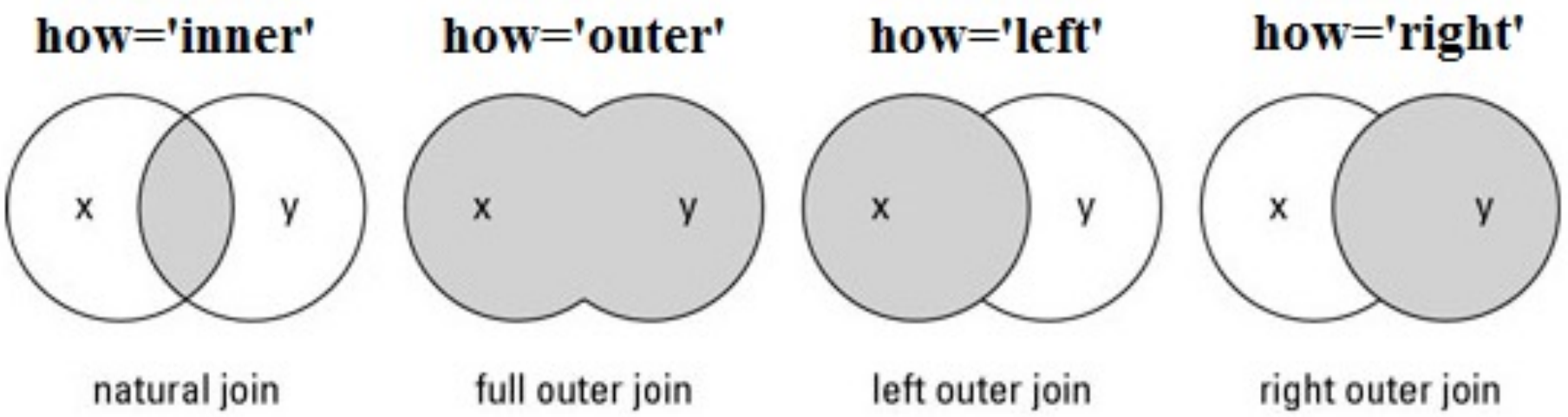
http://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/

inner, outer, left and right



http://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/

inner, outer, left and right



http://www.datasciencemadesimple.com/join-merge-data-frames-pandas-python/

neueda TRAINING

inner Merge (default) – pd.merge

- **Inner Merge** – The default Pandas behaviour, only keep rows where the merge “on” value exists in both the left and right DataFrames

```
nameAge_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Li'], 'Age': ['34', '23', '34']}, index=[4, 5, 7])
```

```
nameHobbie_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Mark'], 'Hobbie': ['Read', 'Travel', 'BoardGames']},
                              index=[4, 5, 6])
```

```
pd.merge(nameAge_df2, nameHobbie_df2, how='inner', on='Name').head()
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel

neueda TRAINING

Left Merge

Left Merge – (or left join) Keep every row in the left DataFrame. Where there are missing values of the "on" variable in the right DataFrame, add empty / NaN values in the result

```
nameAge_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Li'], 'Age': ['34', '23', '34']}, index=[4, 5, 7])
```

```
nameHobbie_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Mark'], 'Hobbie': ['Read', 'Travel', 'BoardGames']},
                              index=[4, 5, 6])
```

```
pd.merge(nameAge_df2, nameHobbie_df2, how='left', on='Name').head()
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NaN

neueda TRAINING

Right Merge

Right Merge – (right join) Keep every row in the right DataFrame. Where there are missing values of the “on” variable in the left column, add empty / NaN values in the result

```
nameAge_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Li'], 'Age': ['34', '23', '34']}, index=[4, 5, 7])
```

```
nameHobbie_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Mark'], 'Hobbie': ['Read', 'Travel', 'BoardGames']},
                              index=[4, 5, 6])
```

```
pd.merge(nameAge_df2, nameHobbie_df2, how='right', on='Name').head()
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Mark	NaN	BoardGames

Outer Merge

Outer Merge – A full outer join returns all the rows from the left DataFrame, all the rows from the right DataFrame, and matches up rows where possible, with NaNs elsewhere

```
nameAge_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Li'], 'Age': ['34', '23', '34']}, index=[4, 5, 7])
```

```
nameHobbie_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Mark'], 'Hobbie': ['Read', 'Travel', 'BoardGames']},
                              index=[4, 5, 6])
```

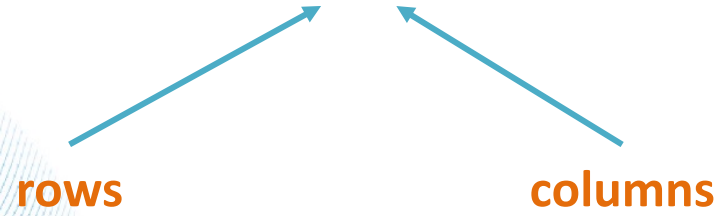
```
pd.merge(nameAge_df2, nameHobbie_df2, how='outer', on='Name').head()
```

	Name	Age	Hobbie
0	Frank	34	Read
1	Helen	23	Travel
2	Li	34	NaN
3	Mark	NaN	BoardGames

Notice the differences in the DataFrames columns and rows

```
print(nameAge_df2.shape)
print(nameHobbie_df2.shape)
print(pd.merge(nameAge_df2, nameHobbie_df2, how='inner', on='Name').shape)
print(pd.merge(nameAge_df2, nameHobbie_df2, how='left', on='Name').shape)
print(pd.merge(nameAge_df2, nameHobbie_df2, how='right', on='Name').shape)
print(pd.merge(nameAge_df2, nameHobbie_df2, how='outer', on='Name').shape)
```

```
(3, 2)
(3, 2)
(2, 3)
(3, 3)
(3, 3)
(4, 3)
```



Join

by default, uses index to join the dataframes
does not allow columns with the same name to be joined
left-, right-, outer-join will apply the same logic as we've seen with merge

```
nameAge_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Li'], 'Age': ['34', '23', '34']}, index=[4, 5, 7])
```

```
nameHobbie_df2 = pd.DataFrame({'Name': ['Frank', 'Helen', 'Mark'], 'Hobbie': ['Read', 'Travel', 'BoardGames']},
                              index=[4, 5, 6])
```

```
nameAge_df2.join(nameHobbie_df2[['Hobbie']], how="inner").head(2)
```

	Name	Age	Hobbie
4	Frank	34	Read
5	Helen	23	Travel

or use in_place=True to change index and save the changes in the DataFrame object

```
nameAge_df3 = nameAge_df2.set_index('Name')
nameHobbie_df3 = nameHobbie_df2.set_index('Name')
nameAge_df3.join(nameHobbie_df3, how="inner").head(2)
```

Name	Age	Hobbie
Frank	34	Read
Helen	23	Travel

Groupby

```
df.groupby('species').sum()
```

	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	5.1	3.5	1.4	0.2
1	setosa	4.9	3.0	1.4	0.2
2	setosa	4.7	3.2	1.3	0.2
3	setosa	4.6	3.1	1.5	0.2
4	setosa	5.0	3.6	1.4	0.2
50	versicolor	7.0	3.2	4.7	1.4
51	versicolor	6.4	3.2	4.5	1.5
52	versicolor	6.9	3.1	4.9	1.5
53	versicolor	5.5	2.3	4.0	1.3
54	versicolor	6.5	2.8	4.6	1.5
100	virginica	6.3	3.3	6.0	2.5
101	virginica	5.8	2.7	5.1	1.9
102	virginica	7.1	3.0	5.9	2.1
103	virginica	6.3	2.9	5.6	1.8
104	virginica	6.5	3.0	5.8	2.2

SUM

SUM

SUM

species	sepal_length	sepal_width	petal_length	petal_width
setosa	24.3	16.4	7.0	1.0
versicolor	32.3	14.6	22.7	7.2
virginica	32.0	14.9	28.4	10.5

https://towardsdatascience.com/how-to-use-the-split-apply-combine-strategy-in-pandas-groupby-29e0eb44b62e

neueda TRAINING

Grouping Data

```
flights.columns
```

```
Index(['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'AIRLINE', 'FLIGHT_NUMBER',
       'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT',
       'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'DEPARTURE_DELAY', 'TAXI_OUT',
       'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE',
       'WHEELS_ON', 'TAXI_IN', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME',
       'ARRIVAL_DELAY', 'DIVERTED', 'CANCELLED', 'CANCELLATION_REASON',
       'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY',
       'Llate_AIRCRAFT_DELAY', 'WEATHER_DELAY'],
      dtype='object')
```

group by one column

group by 2 columns

```
flights.groupby('YEAR')[['ORIGIN_AIRPORT']].count()
```

YEAR	ORIGIN_AIRPORT
2015	5819079

```
flights.groupby(['YEAR', 'DAY'])[['ORIGIN_AIRPORT']].count()
```

YEAR	DAY	ORIGIN_AIRPORT
2015	1	189477
	2	195986
	3	190007
	4	190893
	5	189766
	6	191232

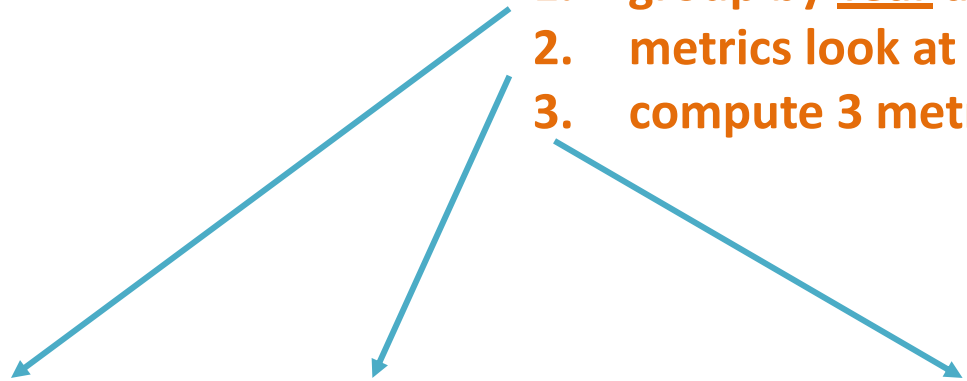
See aggregation metric (count) for a specific column

if we do not specify computes for all columns

neueda TRAINING

Aggregate (agg)

1. group by Year and Day
2. metrics look at value in column AIRLINE_DELAY
3. compute 3 metrics (count, sum and mean)



```
flights.groupby(['YEAR', 'DAY'])[['AIRLINE_DELAY']].agg(['count', 'sum', 'mean']).head(2)
```

		AIRLINE_DELAY		
		count	sum	mean
YEAR	DAY			
2015	1	36198	698542.0	19.297807
	2	40966	744830.0	18.181663

Pandas method – pivot_table

```
flights.groupby(['YEAR', 'DAY'])[['AIRLINE_DELAY']].agg(['count', 'sum', 'mean']).head(2)
```

		AIRLINE_DELAY		
		count	sum	mean
YEAR	DAY			
2015	1	36198	698542.0	19.297807
	2	40966	744830.0	18.181663

by default computes the mean

```
flights.pivot_table(index=['YEAR', 'DAY']).head(2)
```

		AIRLINE_DELAY	AIR_SYSTEM_DELAY	AIR_TIME	ARRIVAL_DELAY	ARRIVAL_TIME	CANCELLED	DAY_OF_WEEK	DEPARTURE_DELAY	DEPARTURI
YEAR	DAY									
2015	1	19.297807	14.353196	113.778051	5.685146	1481.054696	0.028077	4.161233	10.538542	1344.
	2	18.181663	12.666113	113.395514	6.255026	1481.098571	0.025257	3.478233	10.623491	1339.

2 rows × 24 columns

neueda TRAINING

Pandas method – pivot_table

```
flights.groupby(['YEAR', 'DAY'])[['AIRLINE_DELAY']].agg(['count', 'sum', 'mean']).head(2)
```

		AIRLINE_DELAY		
		count	sum	mean
YEAR	DAY			
2015	1	36198	698542.0	19.297807
	2	40966	744830.0	18.181663

We can specify other metrics using the parameter *aggfunc*

```
flights.pivot_table(index=['YEAR', 'DAY'], aggfunc='sum').head(2)
```

		AIRLINE_DELAY	AIR_SYSTEM_DELAY	AIR_TIME	ARRIVAL_DELAY	ARRIVAL_TIME	CANCELLED	DAY_OF_WEEK	DEPARTURE_DELAY	DEPARTURI
YEAR	DAY									
2015	1	698542.0	519557.0	20886692.0	1043645.0	272591079.0	5320	788458	1942738.0	24778
	2	744830.0	518880.0	21603433.0	1191670.0	282812810.0	4950	681685	2030999.0	25606

2 rows × 24 columns

neueda TRAINING

Pandas method – pivot_table

Similar to *groupby* method we can aggregate and compute different metrics

```
flights_subset = flights[['YEAR', 'DAY', 'AIRLINE_DELAY', 'AIR_SYSTEM_DELAY']]
flights_subset.pivot_table(index=['YEAR', 'DAY'], aggfunc=['sum', 'mean']).head(2)
```

		sum		mean	
		AIRLINE_DELAY	AIR_SYSTEM_DELAY	AIRLINE_DELAY	AIR_SYSTEM_DELAY
YEAR	DAY				
2015	1	698542.0	519557.0	19.297807	14.353196
	2	744830.0	518880.0	18.181663	12.666113

Go to notebook

Merging and Grouping
Pivot, Cross-Sections and Multi-Index

neueda
TRAINING

Library datetime

```
from datetime import datetime
```

The lower level libraries can be imported individually.
This can be confusing at first, but it is quite logical.
The confusion arises from the fact that name **datetime** is used twice, once as the name of the root library and once as the name of a lower level library.
To import a lower level library directly, use the syntax:

from **base-library** import **sub-library**

Convert Datetime to String


```
dt_object = datetime(2017,8,20)
dt_object
```

```
datetime.datetime(2017, 8, 20, 0, 0)
```

```
dt_object.strftime('%Y-%m-%d')
```

```
'2017-08-20'
```

```
dt_object.strftime('%A %d %B %Y')
```

```
'Sunday 20 August 2017'
```

Common formatting strings

- %a --- Weekday as locale's abbreviated name. eg. **Mon**
- %a --- Weekday as locale's abbreviated name. eg. **Mon**
- %A --- Weekday as locale's full name. eg. **Monday**
- %w --- Weekday as a decimal number, where 0 is Sunday and 6 is Saturday. eg. **1**
- %d --- Day of the month as a zero-padded decimal number. eg. **30**
- %b --- Month as locale's abbreviated name. eg. **Sep**
- %B --- Month as locale's full name. eg. **September**
- %m --- Month as a zero-padded decimal number. eg. **09**
- %y --- Year without century as a zero-padded decimal number. eg. **13**
- %Y --- Year with century as a decimal number. eg.**2013**

http://strftime.org/

Convert Datetime to Datetime

```
df = pd.read_csv(filepath_or_buffer='../Data/Equities/GOOGL.csv')
```

convert column to datetime object

object

```
df.dtypes
```

```
Date              object
Open             float64
High             float64
Low              float64
Close            float64
Volume           float64
Ex-Dividend      float64
Split Ratio      float64
Adj. Open        float64
Adj. High        float64
Adj. Low         float64
Adj. Close       float64
Adj. Volume      float64
dtype: object
```

convert to datetime object

```
df['Date'] = pd.to_datetime(df_notindex['Date'])
```

```
df.dtypes
```

```
Date              datetime64[ns]
Open                     float64
High                     float64
Low                      float64
Close                    float64
Volume                   float64
Ex-Dividend              float64
Split Ratio              float64
Adj. Open                float64
Adj. High                float64
Adj. Low                 float64
Adj. Close               float64
Adj. Volume              float64
dtype: object
```

datetime

Datetime properties

Property	Description
year	The year of the datetime
month	The month of the datetime
day	The days of the datetime
hour	The hour of the datetime
minute	The minutes of the datetime
second	The seconds of the datetime
microsecond	The microseconds of the datetime
nanosecond	The nanoseconds of the datetime
date	Returns datetime.date (does not contain timezone information)
time	Returns datetime.time (does not contain timezone information)
timetz	Returns datetime.time as local time with timezone information
dayofyear	The ordinal day of year
weekofyear	The week ordinal of the year
week	The week ordinal of the year
dayofweek	The number of the day of the week with Monday=0, Sunday=6
weekday	The number of the day of the week with Monday=0, Sunday=6
weekday_name	The name of the day in a week (ex: Friday)
quarter	Quarter of the date: Jan-Mar = 1, Apr-Jun = 2, etc.
days_in_month	The number of days in the month of the datetime
is_month_start	Logical indicating if first day of month (defined by frequency)
is_month_end	Logical indicating if last day of month (defined by frequency)
is_quarter_start	Logical indicating if first day of quarter (defined by frequency)
is_quarter_end	Logical indicating if last day of quarter (defined by frequency)
is_year_start	Logical indicating if first day of year (defined by frequency)
is_year_end	Logical indicating if last day of year (defined by frequency)
is_leap_year	Logical indicating if the date belongs to a leap year

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

```
df['Date'].dt.date
```

```
0    2004-08-19
1    2004-08-20
2    2004-08-23
3    2004-08-24
4    2004-08-25
5    2004-08-26
```

```
df['Date'].dt.weekday
```

```
0    3
1    4
2    0
3    1
4    2
```

```
df['Date'].dt.daysinmonth
```

```
0    31
1    31
2    31
3    31
4    31
5    31
6    31
7    31
8    31
```

Timezones

use the pytz package (python timezone)
use **all_timezones** to see the entire list

```
import pytz
```

```
tz = pytz.timezone('America/New_York')
datetime.now(tz)
```

```
datetime.datetime(2019, 9, 3, 17, 39, 26, 226748, tzinfo=<DstTzInfo 'America/New_York' EDT-1 day, 20:00:00 DST>)
```

```
tz = pytz.timezone('Europe/London')
datetime.now(tz)
```

```
datetime.datetime(2019, 9, 3, 22, 40, 3, 748326, tzinfo=<DstTzInfo 'Europe/London' BST+1:00:00 DST>)
```

```
tz = pytz.timezone('Europe/Lisbon')
datetime.now(tz)
```

```
datetime.datetime(2019, 9, 3, 22, 40, 36, 447830, tzinfo=<DstTzInfo 'Europe/Lisbon' WEST+1:00:00 DST>)
```

Date Ranges

if the index is sorted ascending (earliest date first) then the slice will be: **df['early_date' : 'late_date']**
if the index is sorted descending (earliest date last) then the slice will be: **df['late_date' : 'early_date']**
if your index and slice order aren't the same then an empty DataFrame will be returned

```
# slice between specific dates
df_GOOGL['2010-12-02':'2010-12-25']
```

```
# In steps of 30 calendar days
df_GOOGL['2010-12':'2012-1':30]
```

```
# between months in steps of 45 days
df_GOOGL['2010-Nov':'2011-MAY':45]
```

index needs to be sorted in ascending order

Moves every 30 days – from entries in DataFrame index e.g. if 2 days of the month are missing and we start on December 1 then we will include at least one day from January

neueda TRAINING

Date Ranges and Frequencies

Extremely useful in the field of finance
Convenient syntax
version 1 - start, stop, frequency
version 2 - start, frequency, periods

More
http://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

Resampling

Resampling is a conversion between frequencies

Downsampling – going from a finer grained frequency to a lower grained frequency. e.g. Days to Months, Months to Years

Upsampling – the reverse, e.g. months to days, days to minutes

Upsampling will require some interpolation

Offset aliases

Alias	Description
B	business day frequency
C	custom business day frequency
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency
BQ	business quarter end frequency
QS	quarter start frequency
BQS	business quarter start frequency
A, Y	year end frequency
BA, BY	business year end frequency
AS, YS	year start frequency
BAS, BYS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

A number of string aliases are given to useful common time series frequencies.

https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html

Moving Windows

rolling() - create a window and slide along, returning a Series as you go

expanding() - gradually increase the size of your window

```
# A rolling 250 day mean for business days
df_Open = df_GOOGL['Adj. Open'].resample('D').ffill()
df_Open.plot()
df_Open.rolling(250).mean().plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11803b320>
```

neueda TRAINING

Go to notebook

Time Series

neueda
TRAINING

Create a plot for a Series

```
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('1-1-2000', periods=1000))
ts.head(4)
```

```
2000-01-01    0.491216
2000-01-02    2.117036
2000-01-03    0.195288
2000-01-04    0.476865
Freq: D, dtype: float64
```

Cumulative sum sums the column value with its value plus all previous columns

0.491216+2.117036 = 2.608252

0.491216+2.117036+0.195288 = 2.803540

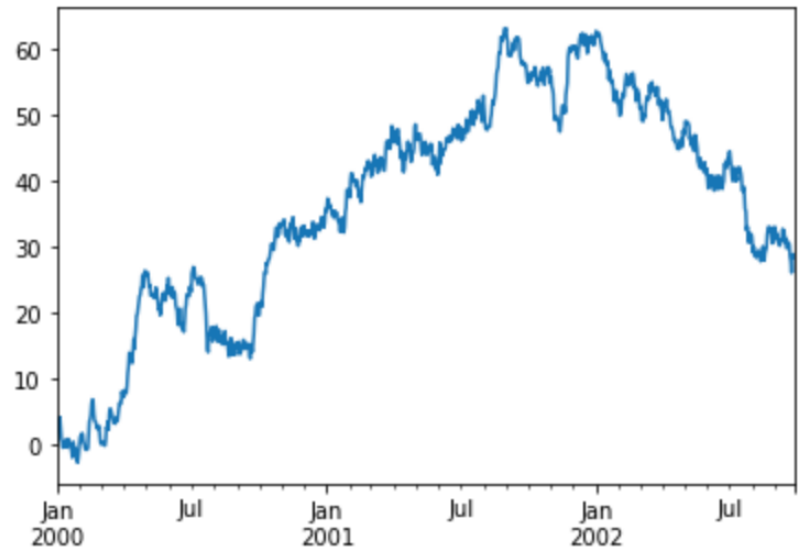
....

```
ts = ts.cumsum()
ts.head(4)
```

```
2000-01-01    0.491216
2000-01-02    2.608252
2000-01-03    2.803540
2000-01-04    3.280405
Freq: D, dtype: float64
```

```
ts.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x12261f160>
```



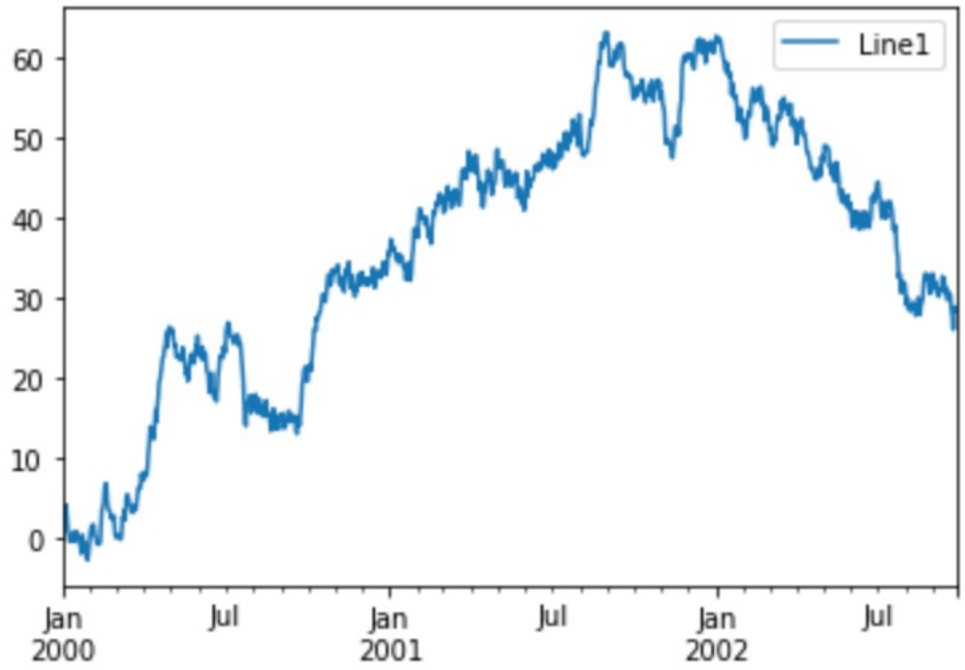
https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html

Create a plot from a Pandas DataFrame

```
cumsum_df = pd.DataFrame(ts, columns=['Line1'])
```

```
cumsum_df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1227810b8>
```



- Notice that automatically creates a legend
- How can I add more lines to this plot?
  - Add more columns to the DataFrame and plot them

neueda TRAINING

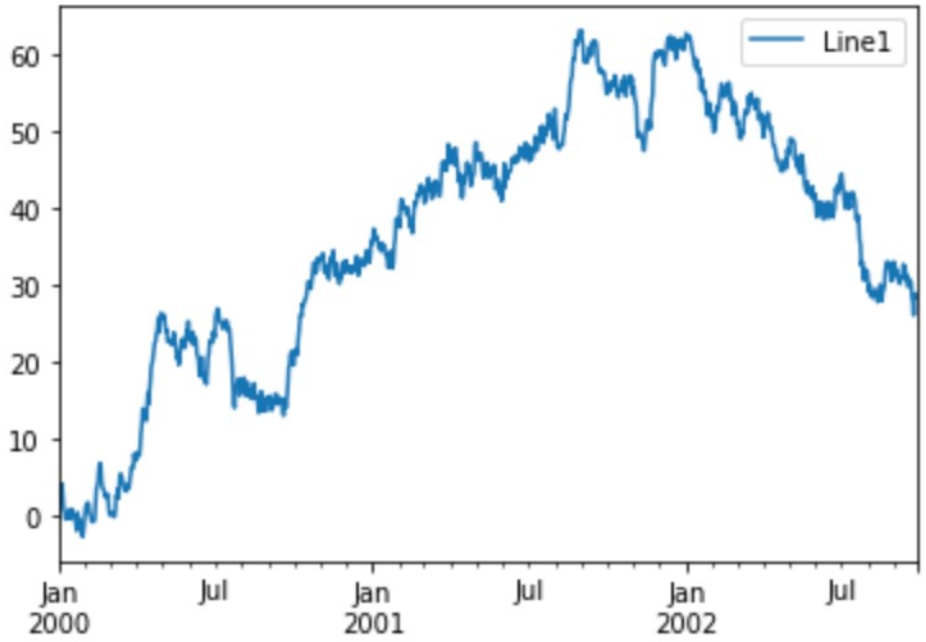
Create a plot from a Pandas DataFrame

1

```
cumsum_df = pd.DataFrame(ts, columns=['Line1'])
```

```
cumsum_df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1227810b8>
```



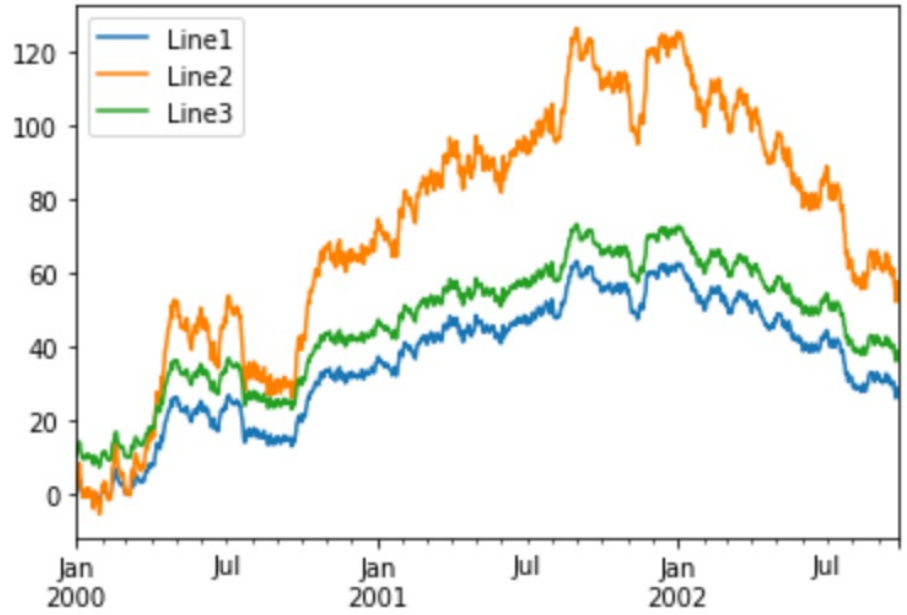
2

```
cumsum_df['Line2'] = cumsum_df['Line1']*2
```

```
cumsum_df['Line3'] = cumsum_df['Line1']+10
```

```
cumsum_df.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11ebdda58>
```



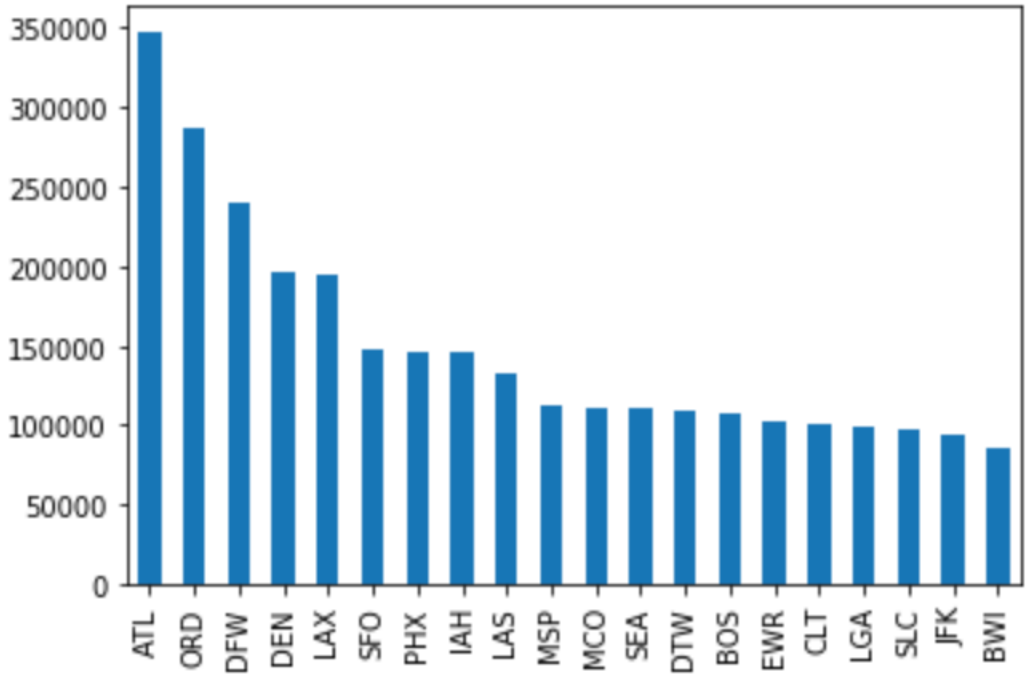
Bar plot

```
: flights['ORIGIN_AIRPORT'].value_counts()[:10]
```

```
: ATL    346836
  ORD    285884
  DFW    239551
  DEN    196055
  LAX    194673
  SFO    148008
  PHX    146815
  IAH    146622
  LAS    133181
  MSP    112117
  Name: ORIGIN_AIRPORT, dtype: int64
```

```
flights['ORIGIN_AIRPORT'].value_counts()[:20].plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x124e96940>
```



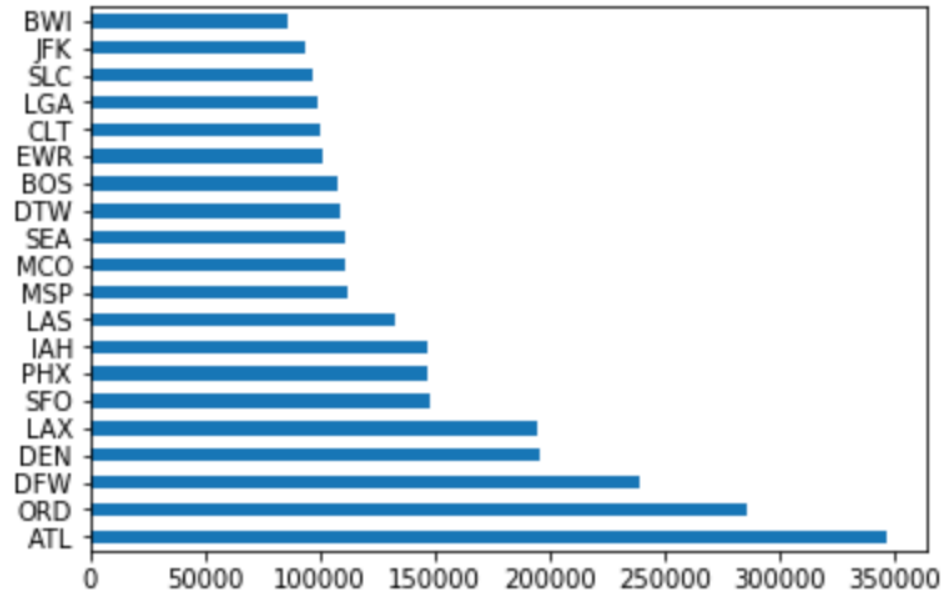
neueda TRAINING

Barh to get horizontal bars

```
: flights['ORIGIN_AIRPORT'].value_counts()[:10]
```

```
: ATL    346836
  ORD    285884
  DFW    239551
  DEN    196055
  LAX    194673
  SFO    148008
  PHX    146815
  IAH    146622
  LAS    133181
  MSP    112117
  Name: ORIGIN_AIRPORT, dtype: int64
```

```
plt_bar = flights['ORIGIN_AIRPORT'].value_counts()[:20].plot.barh()
```



neueda TRAINING

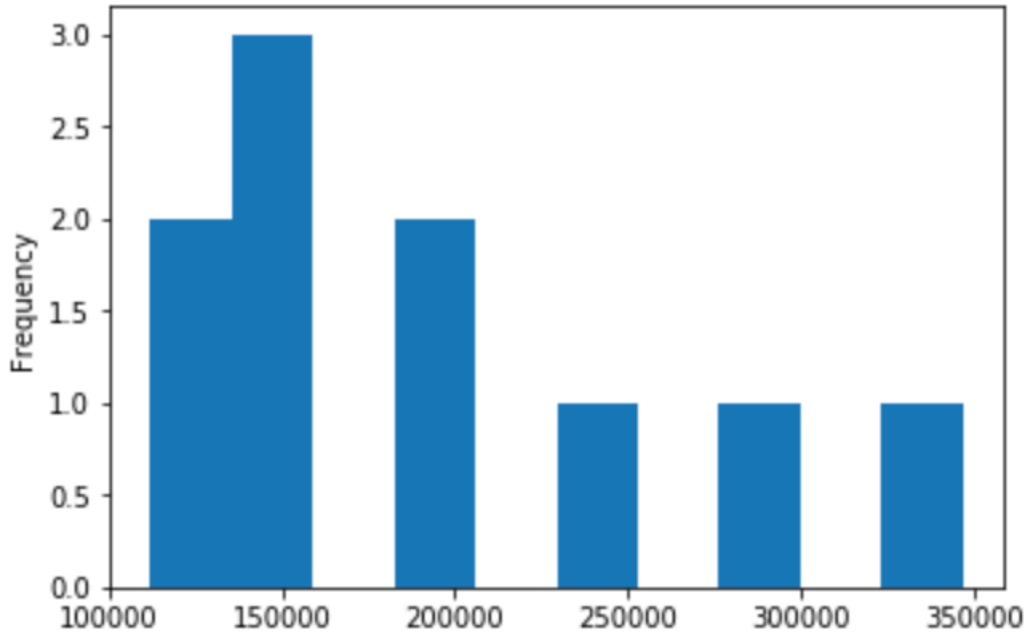
Histogram plot

```
: flights['ORIGIN_AIRPORT'].value_counts()[:10]
```

```
: ATL    346836
  ORD    285884
  DFW    239551
  DEN    196055
  LAX    194673
  SFO    148008
  PHX    146815
  IAH    146622
  LAS    133181
  MSP    112117
  Name: ORIGIN_AIRPORT, dtype: int64
```

```
flights['ORIGIN_AIRPORT'].value_counts()[:10].plot.hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x124f46d30>
```



neueda TRAINING

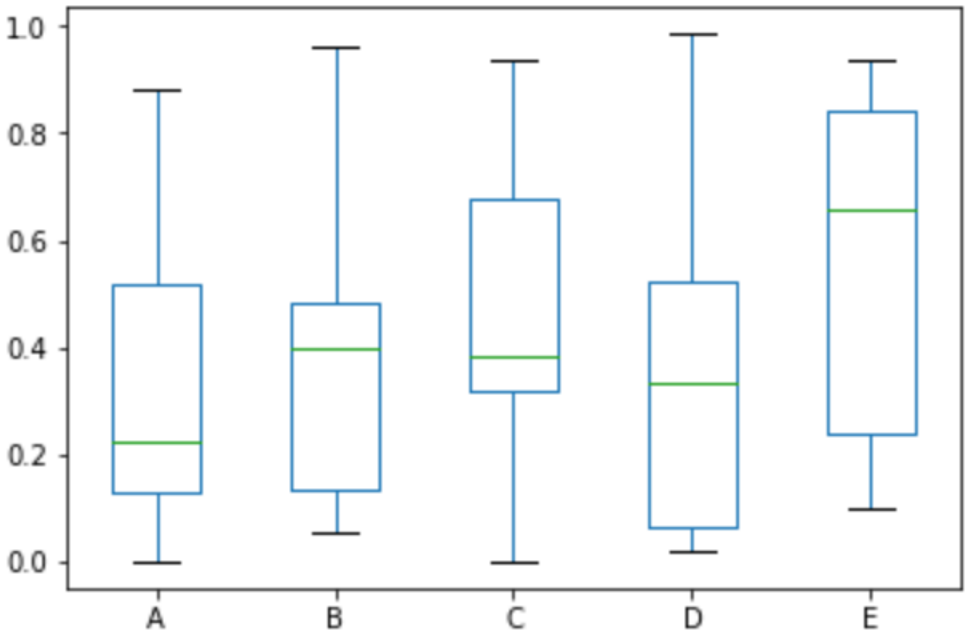
Whiskers plot

```
random_df = pd.DataFrame(np.random.rand(10, 5),
                         columns=['A', 'B', 'C', 'D', 'E'])
random_df.head(10)
```

	A	B	C	D	E
0	0.613069	0.543585	0.912002	0.850830	0.767939
1	0.874579	0.223861	0.140228	0.115048	0.426954
2	0.105654	0.888425	0.148061	0.544411	0.106718
3	0.499051	0.899433	0.636343	0.136189	0.065387
4	0.889739	0.350896	0.406432	0.232018	0.111568
5	0.304291	0.597283	0.505421	0.668846	0.313556
6	0.090246	0.979429	0.453856	0.450882	0.385523
7	0.135110	0.538971	0.569672	0.330625	0.502135
8	0.965026	0.998419	0.049006	0.442440	0.541584
9	0.201406	0.615795	0.404819	0.975760	0.543544

```
random_df.plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x125320cc0>
```

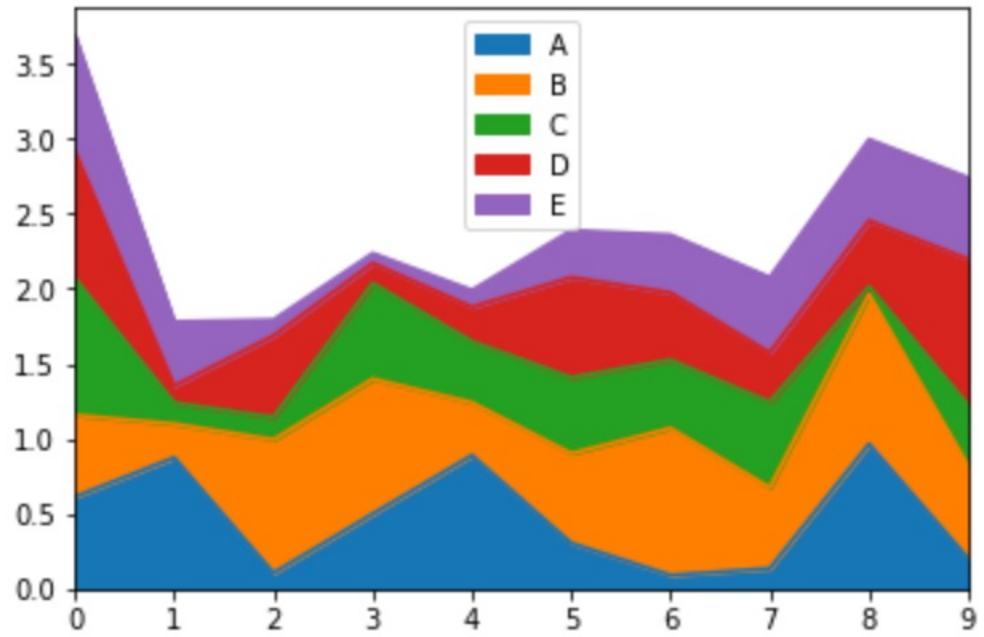


Plot Area

```
random_df = pd.DataFrame(np.random.rand(10, 5),
                         columns=['A', 'B', 'C', 'D', 'E'])
random_df.head(10)
```

	A	B	C	D	E
0	0.613069	0.543585	0.912002	0.850830	0.767939
1	0.874579	0.223861	0.140228	0.115048	0.426954
2	0.105654	0.888425	0.148061	0.544411	0.106718
3	0.499051	0.899433	0.636343	0.136189	0.065387
4	0.889739	0.350896	0.406432	0.232018	0.111568
5	0.304291	0.597283	0.505421	0.668846	0.313556
6	0.090246	0.979429	0.453856	0.450882	0.385523
7	0.135110	0.538971	0.569672	0.330625	0.502135
8	0.965026	0.998419	0.049006	0.442440	0.541584
9	0.201406	0.615795	0.404819	0.975760	0.543544

```
cplot = random_df.plot.area()
```



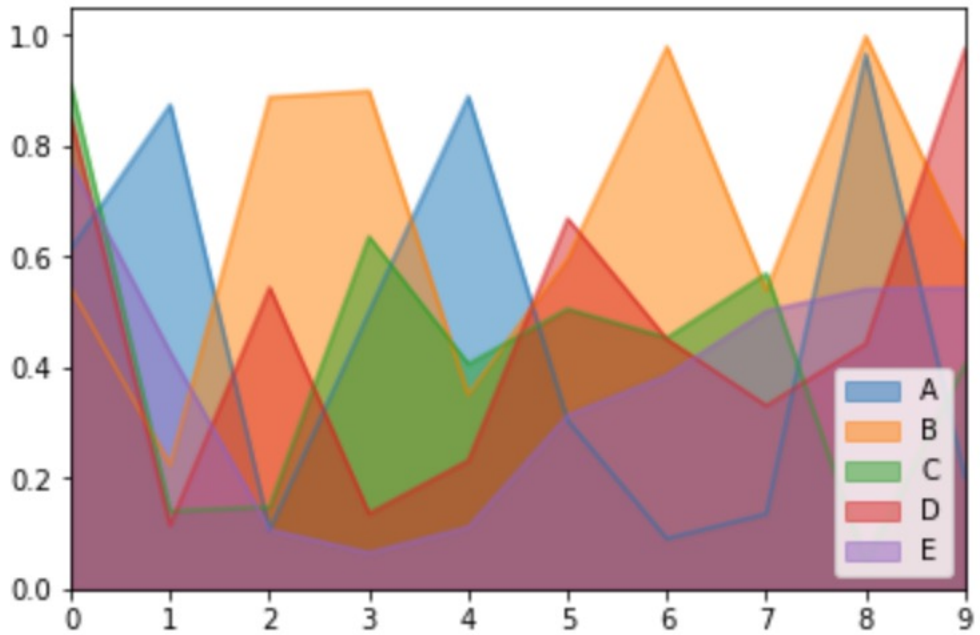
eueda TRAINING

Plot Area (stacked=False)

```
random_df = pd.DataFrame(np.random.rand(10, 5),
                         columns=['A', 'B', 'C', 'D', 'E'])
random_df.head(10)
```

	A	B	C	D	E
0	0.613069	0.543585	0.912002	0.850830	0.767939
1	0.874579	0.223861	0.140228	0.115048	0.426954
2	0.105654	0.888425	0.148061	0.544411	0.106718
3	0.499051	0.899433	0.636343	0.136189	0.065387
4	0.889739	0.350896	0.406432	0.232018	0.111568
5	0.304291	0.597283	0.505421	0.668846	0.313556
6	0.090246	0.979429	0.453856	0.450882	0.385523
7	0.135110	0.538971	0.569672	0.330625	0.502135
8	0.965026	0.998419	0.049006	0.442440	0.541584
9	0.201406	0.615795	0.404819	0.975760	0.543544

```
cplot = random_df.plot.area(stacked=False)
```

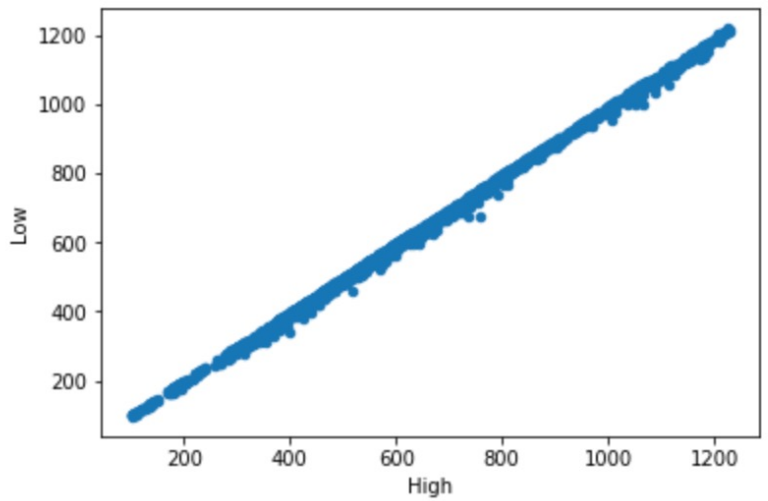


Scatter plot

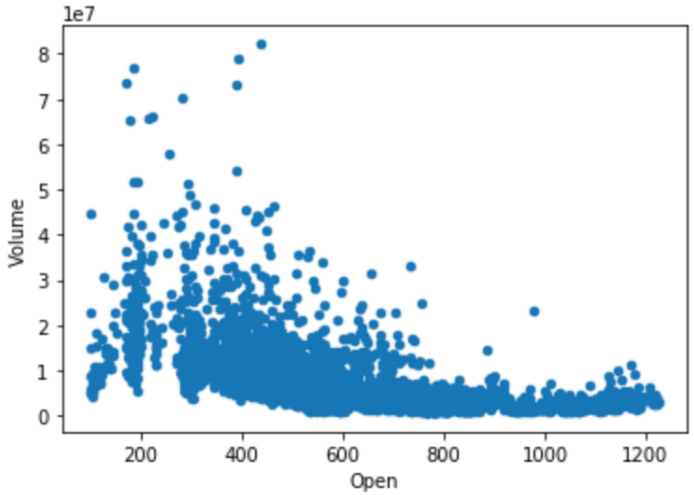
```
df_GOOGL.head()
```

Date	Open	High	Low	Close	High-Low Spread	Volume	Ex-Dividend	Split Ratio	Adj. Open	Adj. High	Adj. Low	Adj. Close	Adj. Volume
2004-08-19	100.01	104.06	95.96	100.335	8.10	44659000.0	0.0	1.0	50.159839	52.191109	48.128568	50.322842	44659000.0
2004-08-20	101.01	109.08	100.50	108.310	8.58	22834300.0	0.0	1.0	50.661387	54.708881	50.405597	54.322689	22834300.0
2004-08-23	110.76	113.48	109.05	109.400	4.43	18256100.0	0.0	1.0	55.551482	56.915693	54.693835	54.869377	18256100.0
2004-08-24	111.24	111.60	103.57	104.870	8.03	15247300.0	0.0	1.0	55.792225	55.972783	51.945350	52.597363	15247300.0
2004-08-25	104.76	108.00	103.88	106.000	4.12	9188600.0	0.0	1.0	52.542193	54.167209	52.100830	53.164113	9188600.0

```
scater_plt = df_GOOGL.plot.scatter(x='High', y='Low')
```

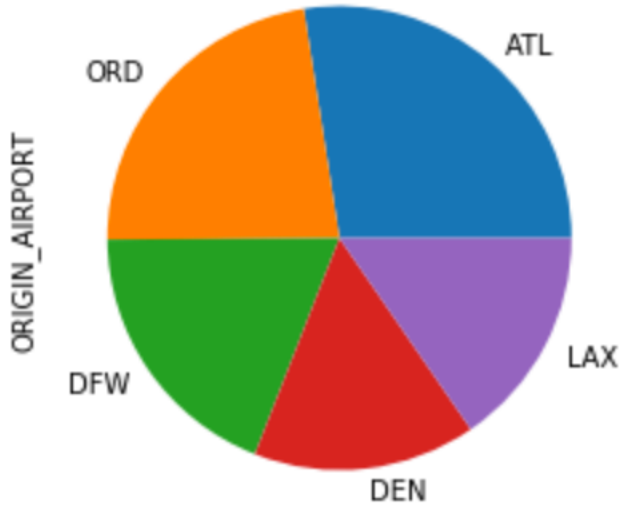



```
scater_plt = df_GOOGL.plot.scatter(x='Open', y='Volume')
```



Pie plot

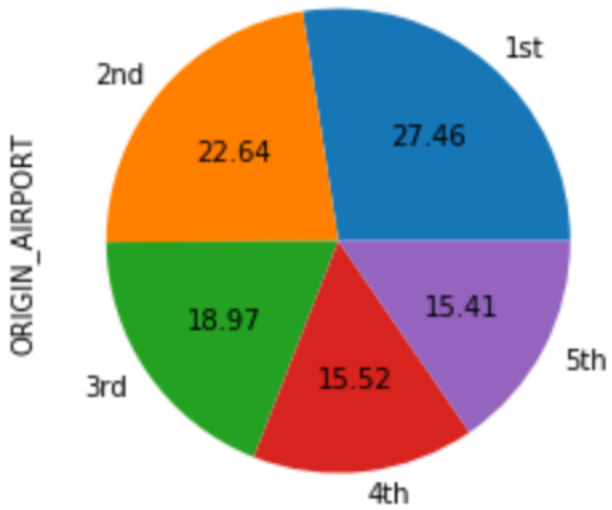
```
origin_plt = flights['ORIGIN_AIRPORT'].value_counts()[:5].plot.pie()
```



neueda TRAINING

Pie plot – work with some *pie* arguments

```
origin_plt = flights['ORIGIN_AIRPORT'].value_counts()[:5].plot.pie(labels=['1st', '2nd', '3rd', '4th', '5th'],
                                                                   autopct='%.2f')
```



neueda TRAINING

Go to notebook

VIsualisation Plotly Seaborn
BollongerBand

neueda
TRAINING

Useful Data Science Libraries

Scikit-learn

One of the most important ML libraries out there
All sorts of useful models and text preprocessing tricks

Scipy

Used for scientific computing and technical computing
Contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering

Useful Libraries for working with text – Gensim, Spacy NLTK

Topic modelling for humans
NLP library with lots of useful (mostly unsupervised) models with a nice API
Game-changing NLP library
Linguistic features and preprocessing done for you for many languages
Natural Language ToolKit
Useful preprocessing tools (mainly stemming)

DATA SCIENCE - RESOURCES

Tableau Website

- Tableau instruction videos
- Data Viz tips: https://www.tableau.com/en-gb/learn/articles/data-visualization-tips

Code Academy

- Python & R for Data Science

More data visualization principles - Github

- https://rafalab.github.io/dsbook/data-visualization-principles.html

Other Data Visualisation tools

- Data Wrapper
- Flourish Studio – for slick animations

neueda TRAINING