NumPy, Pandas, MySQL & EDA

DR. ZVI BEN AMI

About me

- BA, Business Management – Insurance
- MBA, Business Management – Marketing
- PhD, Business Management – Data Science (HUJI)
  - Specializing in NLP and Sentiment Analysis
- Post-Doc, Business Management – Data Science (TAU)
- Data Data Scientist, LVision – focused on CV
- Senior Data Scientist, Lsport
- Co-founder and CTO of Vela Health AI
- Lecturer @ HUJI, TAU, Ono Academic Collage

Agenda

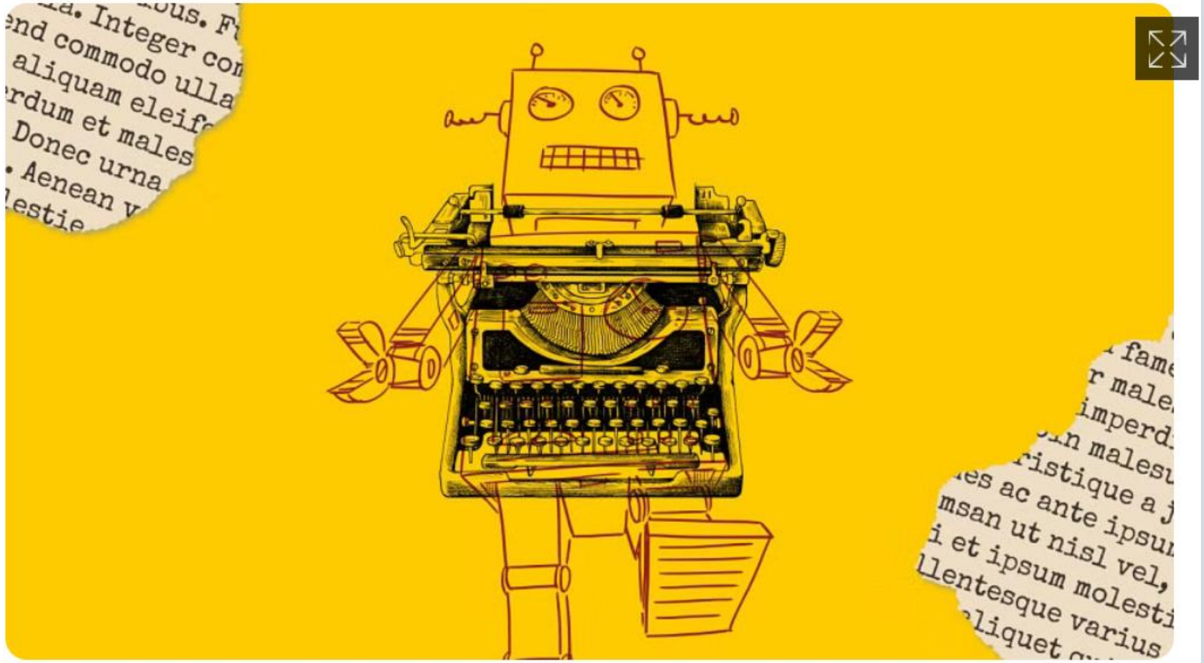
- Foundations of AI & ML
  - Evolution of AI
  - CRISP-DM & Data Mining process
  - Machine Learning tasks
- Python Foundations
  - Setting up environment (IDE, packages)
  - Python core concepts & collections
- NumPy & Pandas
  - Arrays, broadcasting, efficiency
  - Pandas data structures & operations
- Databases for Data Science
  - MySQL basics & Workbench
  - SQL queries & Python connectors
- Exploratory Data Analysis (EDA)
  - Visualization methods
  - Manual vs automatic EDA tools
- Summary & Q&A

Our Future with AI

AI will not replace you, but people who use it may

Toggle Desk

Fri Feb 10, 2023 09:30 AM Last update on: Fri Feb 10, 2023 09:30 AM



AI is not designed to replace humans, but to enhance and augment their capabilities. Illustration: Zarif Faiaz

Source: https://www.thedailystar.net/tech-startup/news/ai-will-not-replace-you-people-who-use-it-may-3244006

Evolution of AI

Data Analysis

~2005–2010	Descriptive stats, dashboards	Analyzing past data to uncover patterns.

Machine Learning

~2010–2015	Predictive modeling	Train models to forecast future outcomes.

Deep Learning

~2015–2021	Image/audio/text understanding	Use neural networks for high-dimensional data.

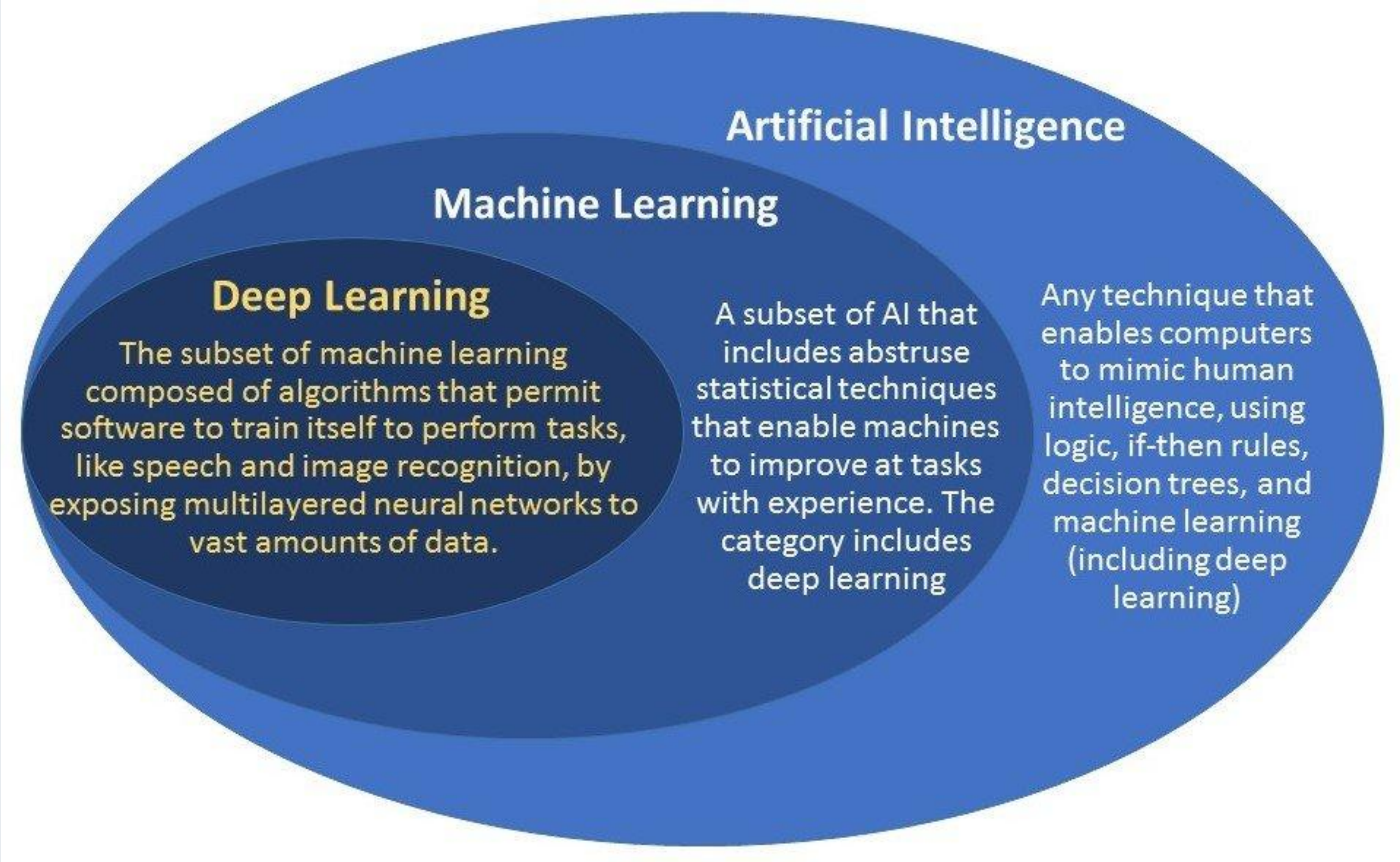
Generative AI

~2022–2024	Content creation	Produce text, images, and videos from prompts.

AI Agents / MCP

~2024–Now	Autonomous/collaborative task execution	Agents that complete complex tasks together.

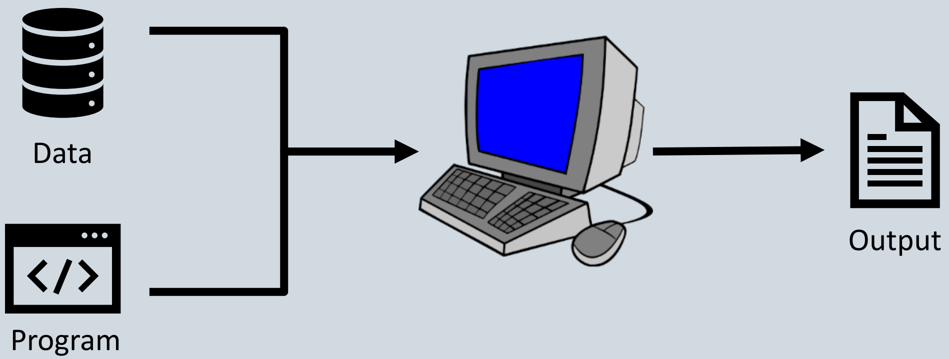
Machine Learning in Data Science



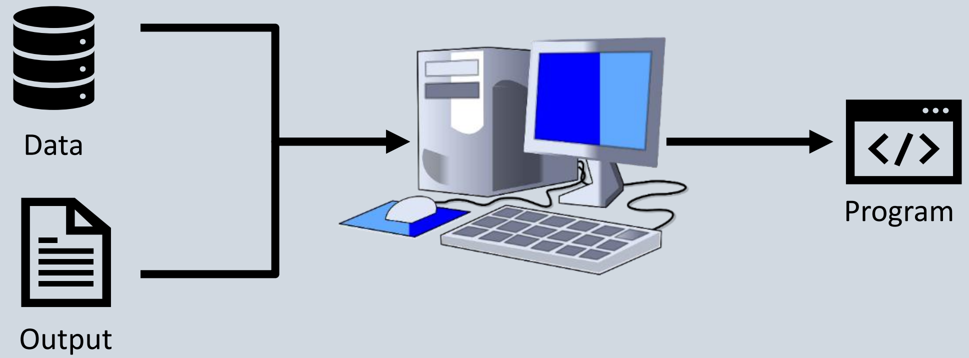
Source: https://www.geospatialworld.net/blogs/difference-between-ai%EF%BB%BF-machine-learning-and-deep-learning/

Programing vs ML

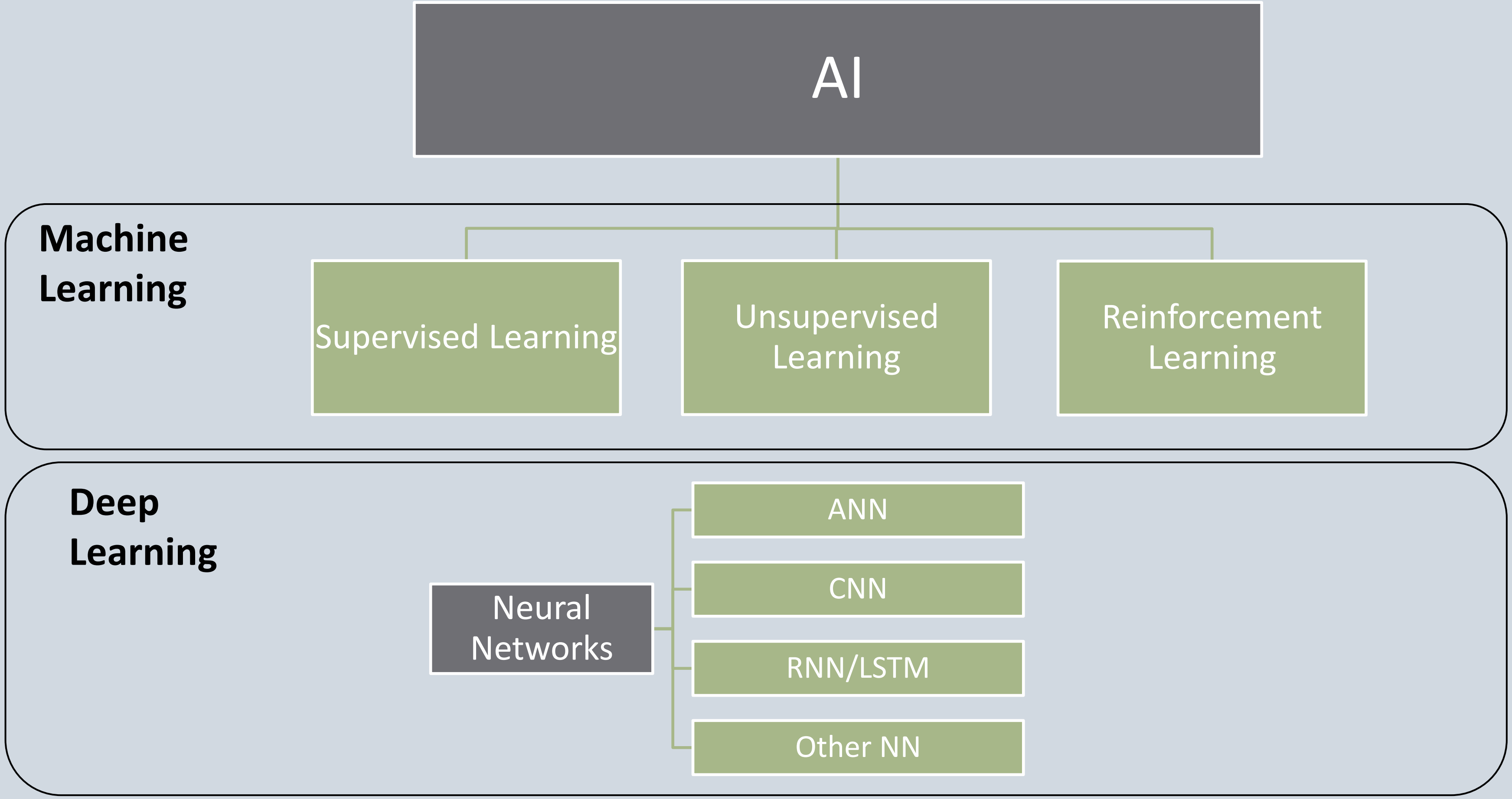
Traditional programing



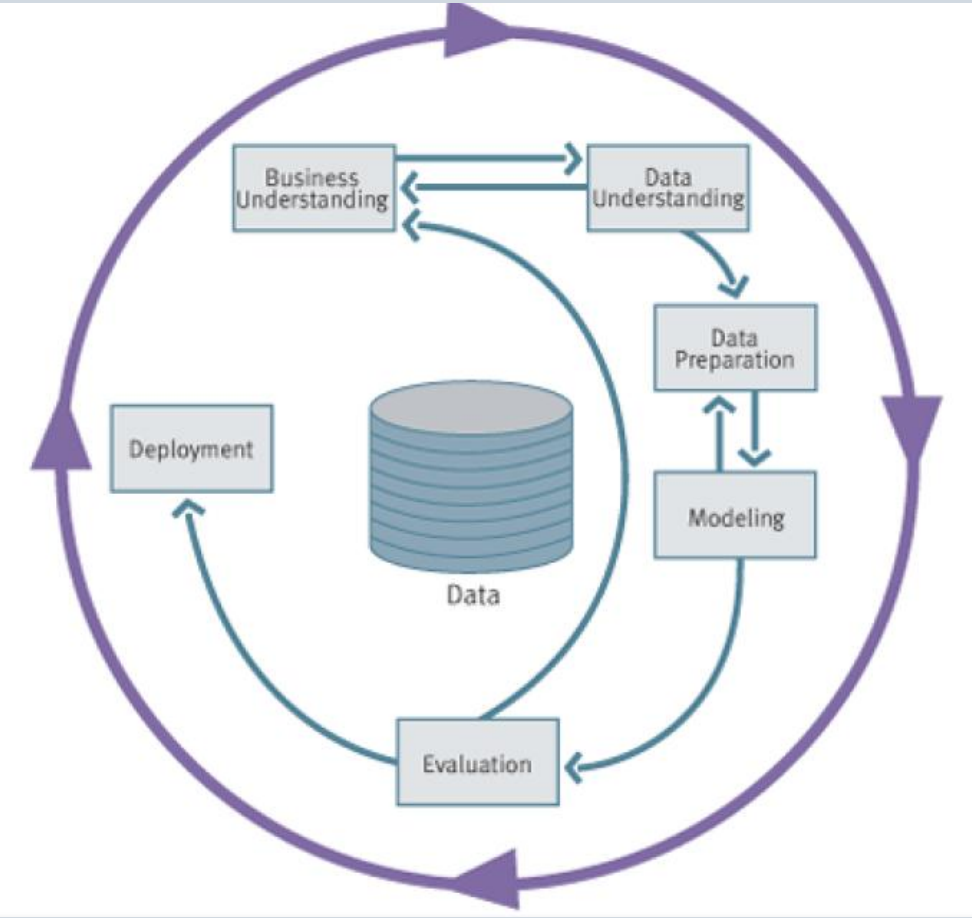
Supervised Learning



AI & Machine Learning



Data Mining Process



- **Not a software development project**
- Outcomes are uncertain
- Iterative process – prepare sufficient resources
- Intensive data preparation stage
- Different skill set than of programmers

CRISP-DM in details:
https://www.ibm.com/docs/it/SS3RA7_18.3.0/pdf/ModelerCRISPDM.pdf

Data sources → Storage → Analysis → EDA → ML.

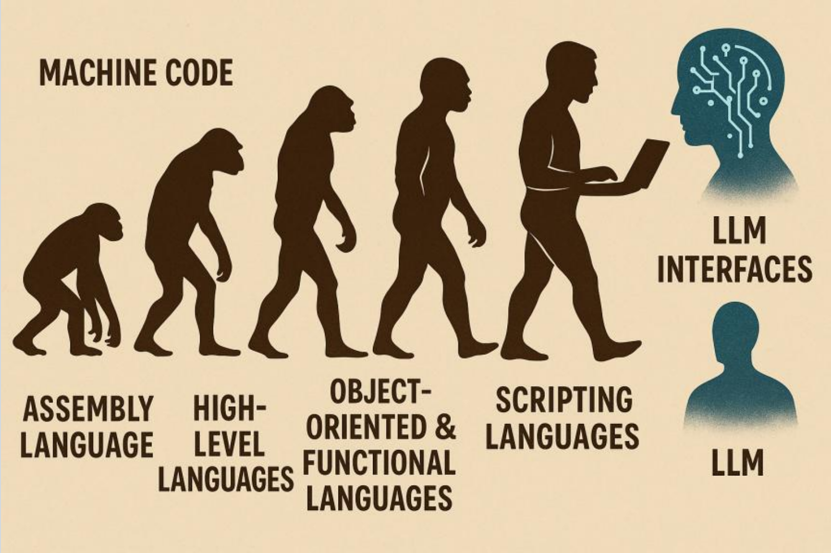
Cross Industry Standard Process for Data Mining (Shearer, 2000)

From Foundations to MCP
Smart City as a Metaphor for AI Learning

AI	Smart House
🔨 **Pandas + NumPy**	The tools (hammer, saw)
📦 **MySQL**	The Warehouse for Raw Materials
📊 **EDA**	The Architect's Blueprint
🏠 **Supervised Learning**	Foundations and walls
🏠 **Unsupervised Learning**	Building methods without blueprints
🏡 **Deep Learning**	The hidden systems in the house (electricity, plumbing)
🏠 **LLM**	Connect the house to the internet
🔗 **RAG**	See what is in the house, know which groceries are needed
🤖 **Agents**	Personal assistants in the house
🏘️ **Multi-Agents**	A whole neighborhood of smart houses
🌆 **MCP**	The City Planner and Utility Coordinator

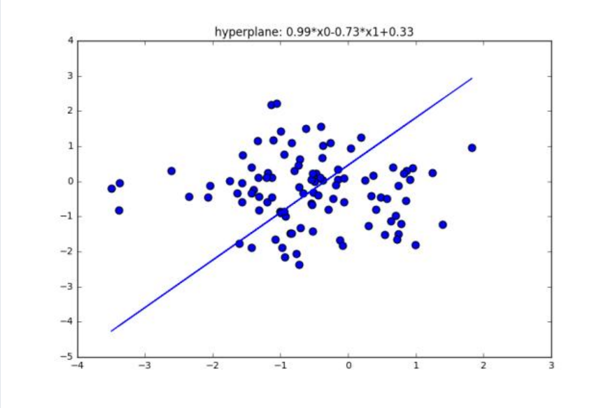
Evolution of Programming Languages

Programming Languages	Evolution
Machine Code **(🟦 0/1 binary)**	✓ Direct instructions to the processor. ➢ Very low-level, unreadable for humans.
Assembly Language **(⚙️ mnemonics)**	✓ Human-readable abbreviations (MOV, ADD). ➢ Assembler translates it to machine code.
High-Level Languages **(🧮 abstraction)**	• Easier to read/write than assembly. ➢ Examples: Fortran, C, COBOL.
Object-Oriented & Functional **(🧩 modularity)**	✓ Supports classes, objects, and functional paradigms. ➢ Examples: Java, C++, Haskell.
Scripting Languages **(📝 dynamic & interpreted)**	✓ Fast prototyping, automation, lightweight execution. ➢ Examples: Python, JavaScript, Bash.
LLM Interfaces **(🤖 natural language layer)**	✓ Human speaks in natural language → LLM generates scripting code → executed by interpreter. ➢ Examples: GPT, Claude.



ML Tasks

- **Classification**: For each individual in a population, which of a (small) set of classes that individual belongs to.
  - Class probability estimation (or scoring) – what is the probability/score for the individual to belong to each category

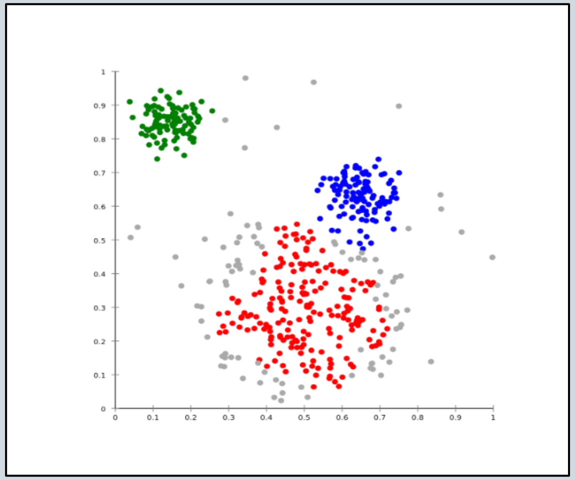


- **Regression** ("value estimation") attempts to estimate or predict, for each individual, the numerical value of some variable for that individual

ML Tasks

- **Similarity matching** attempts to identify similar individuals based on data known about them.
  - Similarity matching can be used directly to find similar entities.

- **Clustering** attempts to *group* individuals in a population together by their similarity, but *not driven by any specific purpose (example or variable)*



ML Tasks

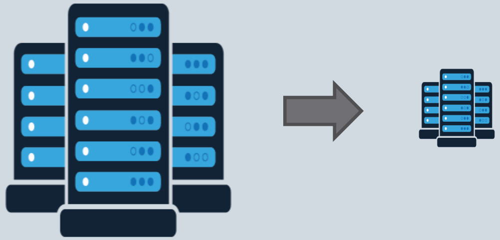
- **Co-occurrence grouping** - attempts to find associations between entities based on transactions involving them.
  - Aka:
    - frequent itemset mining,
    - association rule discovery, and
    - market-basket analysis

- **Profiling** (also known as behavior description) attempts to characterize the typical behavior of an individual, group, or population.

ML Tasks

- **Link prediction** attempts to predict connections between data items, usually by suggesting that a link should exist, and possibly also estimating the strength of the link.
  - *Suggest possible social media links.*
  - *Predict future connections.*

- **Data reduction** attempts to take a large set of data and replace it with a smaller set of data that contains much of the important information in the larger set.



Supervised vs. Unsupervised Learning

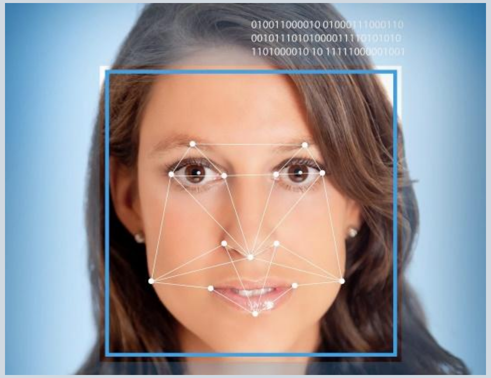
Supervised		Unsupervised
Classification	Data reduction	Clustering
Regression	Similarity matching	Co-occurrence grouping

Key Questions:

- Is there a specific target variable?
- (Is there data on this target variable?)

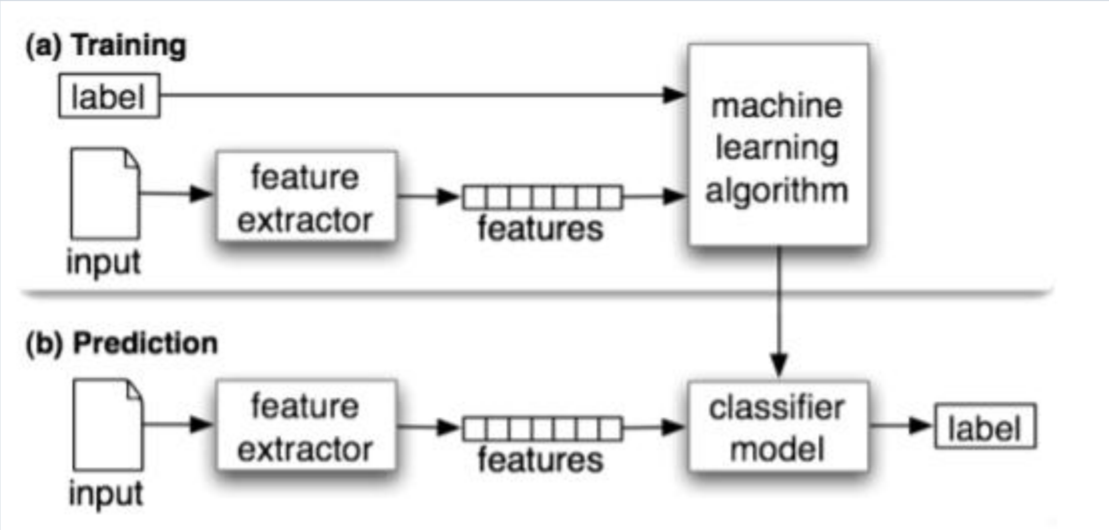
Terminology

- **Structured data** means data that have a pre-defined structure.
  - Tables and DBs (SQL, CSV,etc.)
- **Unstructured data** means free formatted data that does not have a predefined structured.
  - Text, Pictures, Audio Video, etc.



- **Semi-structured data** the structure can be inferred, to an extant, directly from the data.

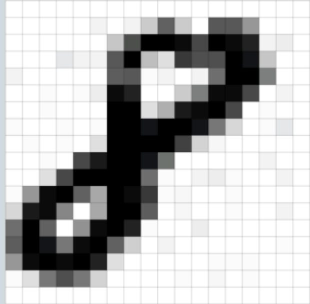
Feature Engineering of Text Documents



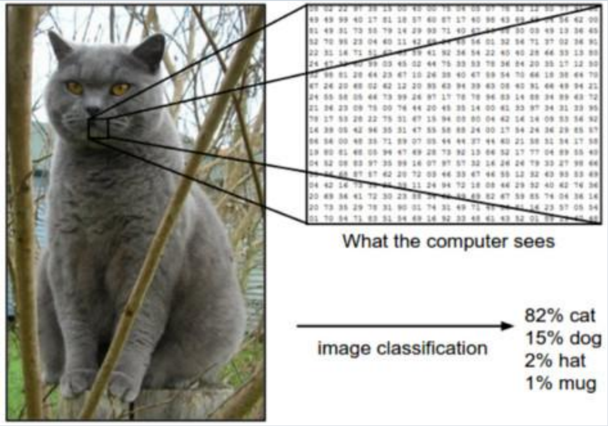
Source: https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

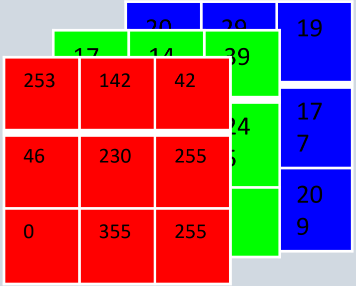
Words	• Data Science is fan-> [“Data”, ”Science”, “is”, ”fan”]
Linguistic Phrases	• “Data Science”, “is fan”
Character Trigrams	• Data Science -> [dat,ata,tas,asc,….ite,teh,eho,hou,ous,use]
Non-consecutive phrases	• research institution -> [“Data Science”, “machine learning”]
Parse trees	• Rooted tree that represents the syntactic structure of a sentence according to some formal grammar
Scripts	• Scenario/Sequence of events or actions
Annotation	• XML tagging <Person>Zvi Ben Ami</Person>
Vector Representation	• Represent the text as embedded vector

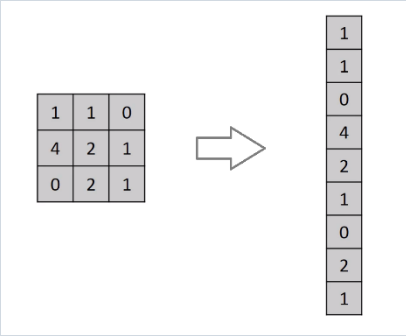
Feature Engineering of Images



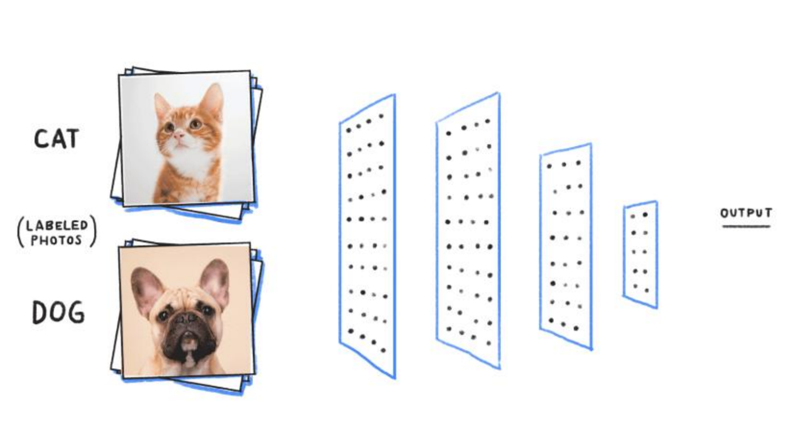
Source: https://cdn-images-1.medium.com/max/1200/1*zY1qFB9aFfZz66YxxoI2aw.gif







Source: https://www.kdnuggets.com/2017/08/convolutional-neural-networks-image-recognition.html



Source: https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8

Data Terminology

Variables (columns)

Dataset
“The data table”
(Flat file)

Attributes, Features, Explanatory, independent Variables

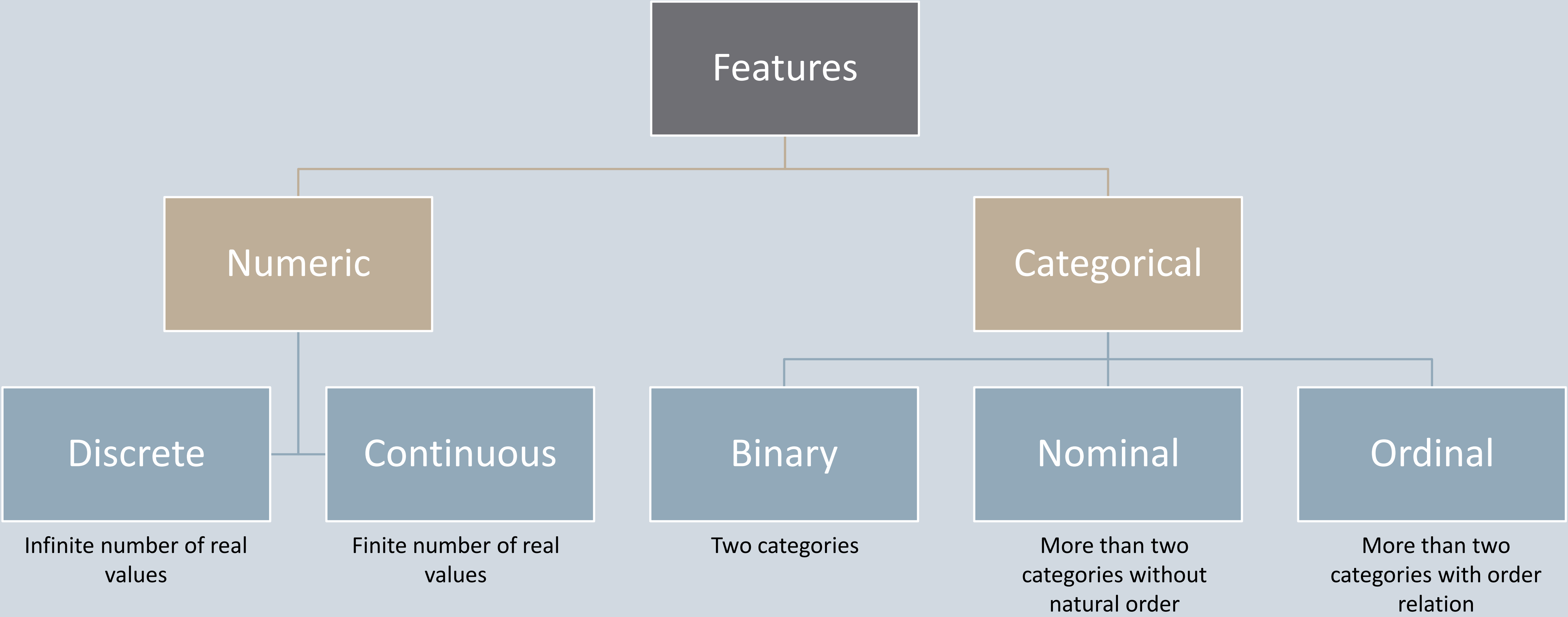
Target Variable, Data class, Dependent variable

feature vector

id	Age	Has a job	Own a house	Credit Rating	Loan
1	Young	False	False	Fair	Disapproved
2	Young	False	False	Good	Disapproved
3	Young	True	False	Good	Approved
4	Young	True	True	Fair	Approved
5	Young	False	False	Fair	Disapproved
6	Middle	False	False	Fair	Disapproved
7	Middle	False	False	Good	Disapproved
8	Middle	False	True	Excellent	Approved
9	Old	False	True	Excellent	Approved
10	Old	False	True	Good	Approved
11	Young	False	False	Good	?

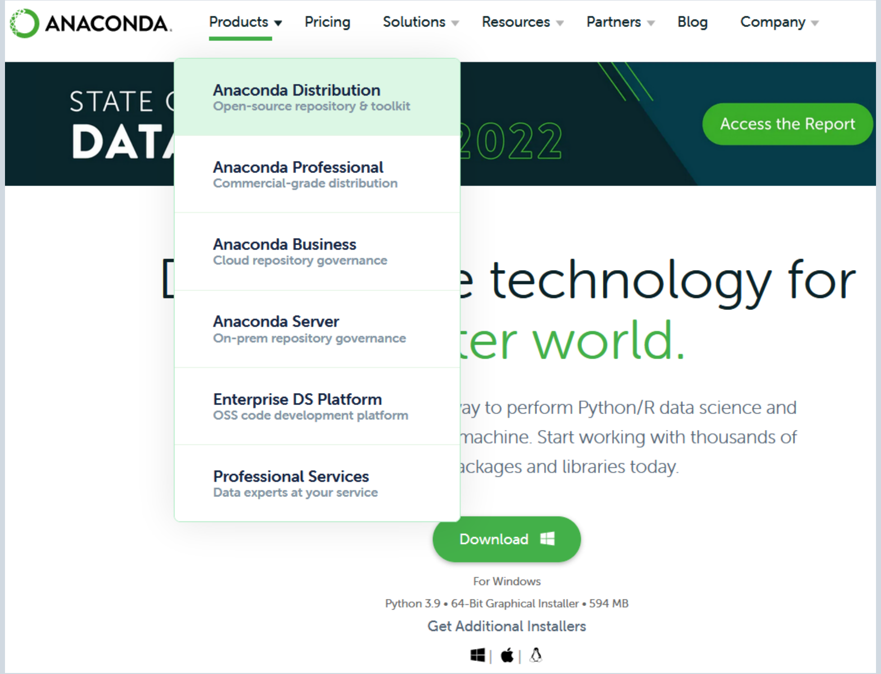
Observations
Records
(Data) Instances
Rows

Features Types



Python Intro
Setting the Environment

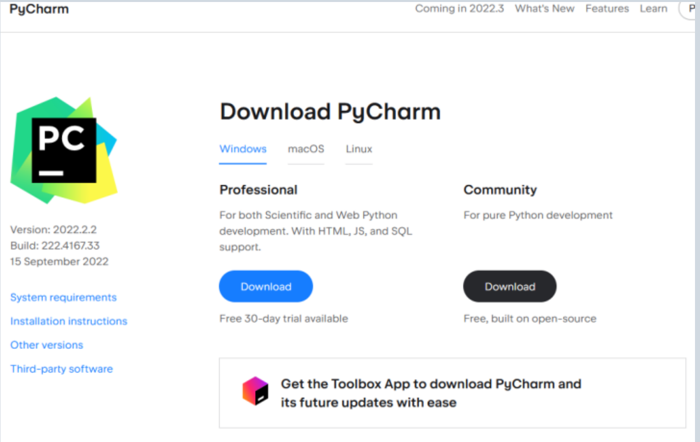
- Why Do We Set the Environment?
  - Reproducibility: Same versions of Python and packages across all machines.
  - Isolation: Each project has its own environment → avoids version conflicts.
  - Portability: Easy to share with classmates/colleagues (environment file).
  - Reliability: Prevents “works on my machine” problems.
- Environment Options:
  - Anaconda / Miniconda (graphical & CLI, very popular in data science)
  - venv (built-in Python tool for lightweight virtual environments)
  - pip + requirements.txt (simple dependency management)
  - Docker (containerized environments, more advanced, for deployment)
  - uv / poetry (modern package & environment managers)
- Installing Anaconda Python
  - https://www.anaconda.com/



IDEs: PyCharm/VSCode/ Cursor

- Integrated Development Environment
- Creating a new project
- Setting python environment
- Installing packages
- Creating Python file
- Writing Code
  - Refactor
  - Jump to definition
  - Autocomplete
- Using the Debugger

https://www.jetbrains.com/pycharm/download



https://code.visualstudio.com/

https://cursor.com/downloads

Python – Collections

lecture_numpy_pandas_mysql_eda.ipynb

- List

```
basket_list = ['apple', 'orange', 'pear', 'pear', 'orange', 'banana']
print(basket_list)
```

  - ['apple', 'orange', 'apple', 'pear', 'orange', 'banana’]

- In python list can be of mixed types

```
basket_list = ['apple', 2, 'apple', 4.5,'orange', 'banana']
print(basket_list)
```

  - ['apple', 2, 'apple', 4.5,'orange', 'banana']

https://docs.python.org/3/

Python – Collections

- Set

```
basket_set = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
# set(basket_list)
print(basket_set)
```

➢{'pear', 'apple', 'orange', 'banana'}

https://docs.python.org/3/

Python – Collections

- Tuples

```
t = (12345, 54321, 'hello!')
print(t[0])
```

  - t[0] = 12
  - TypeError: 'tuple' object does not support item assignment

- Dictionary

```
tel = {'jack': 4098, 'sape': 4139}
tel['guido'] = 4127
print(tel)
```

  - {'jack': 4098, 'sape': 4139, 'guido': 4127}

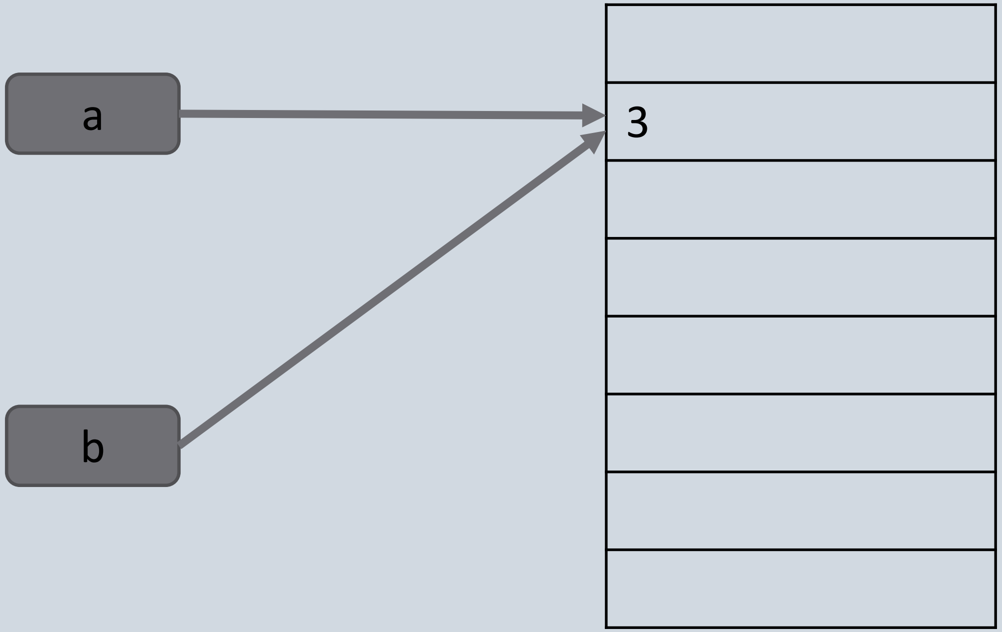
https://docs.python.org/3/

Python Concepts – Memory Allocation

Reference:

```
a = 3
b = 3
```

print("a= ", a)
print(“b= ", b)



a = 3
b = 3

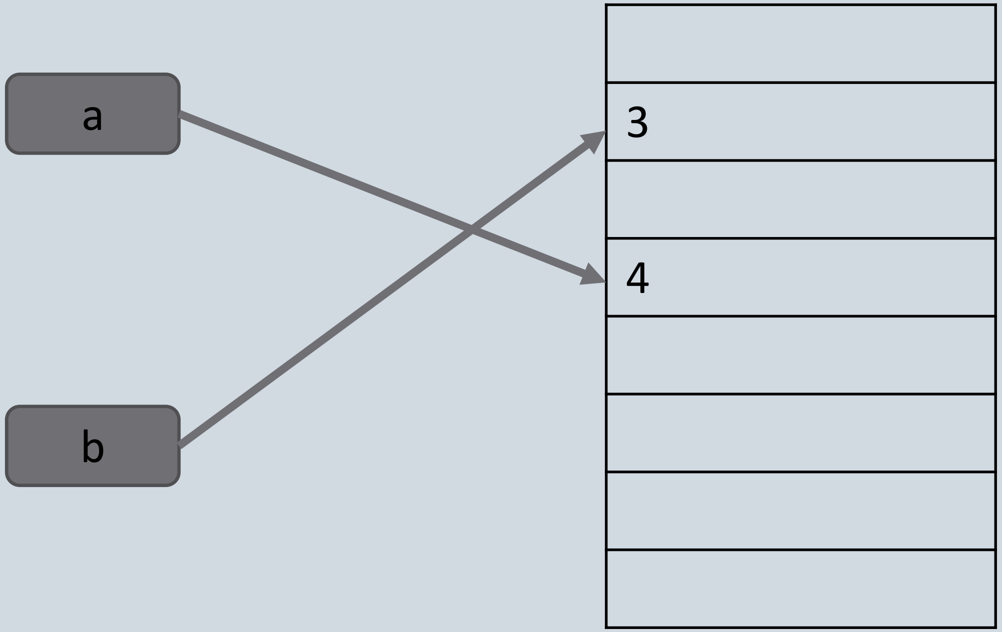
Python Concepts - Memory

Reference:

```
a = 4
print("a= ", a)
print("b= ", b)
```

a = 4

b = 3



Numbers are ***immutable***

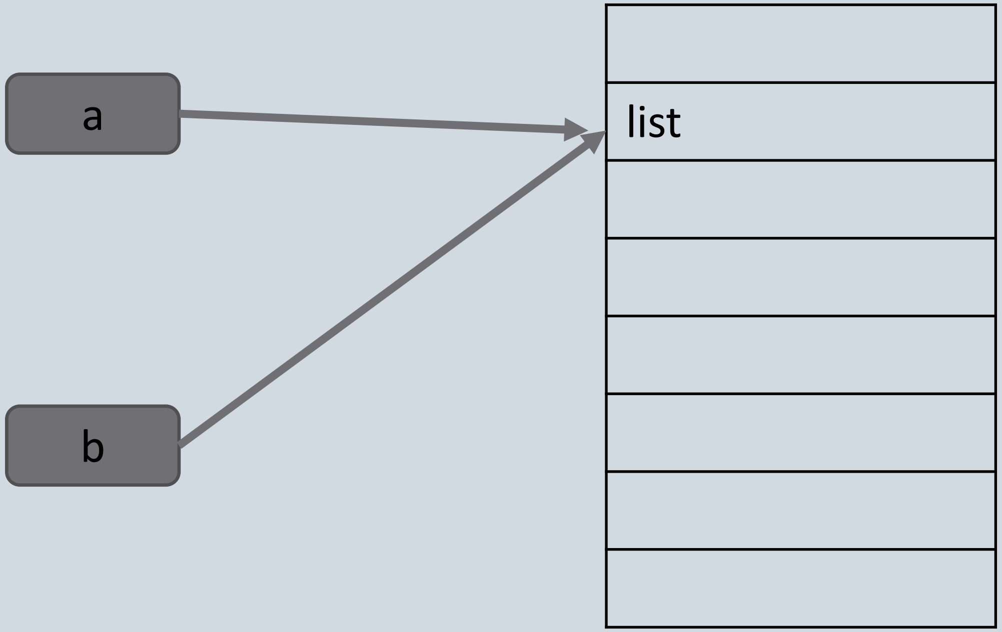
Python Concepts – Mutable Objects

Reference:

```
a = [1,2,3]
b = a
print('a = ', a)
print('b = ', b)
b[0] = 'NewValue'
print('a = ', a)
print('b = ', b)
```

```
a = [1, 2, 3]
b = [1, 2, 3]

a = ['NewValue', 2, 3]
b = ['NewValue', 2, 3]
```

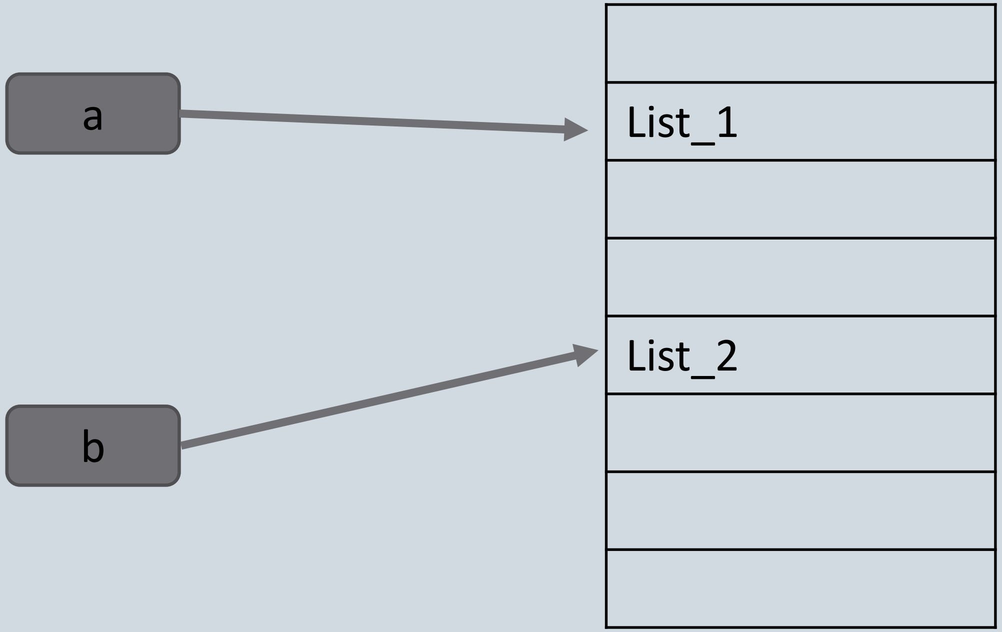


Python Concepts – Mutable Objects 1

Reference:

```
a = [1,2,3]
b = a
print('a = ', a)
print('b = ', b)
```

```
a = [1, 2, 3]
b = [1, 2, 3]
```

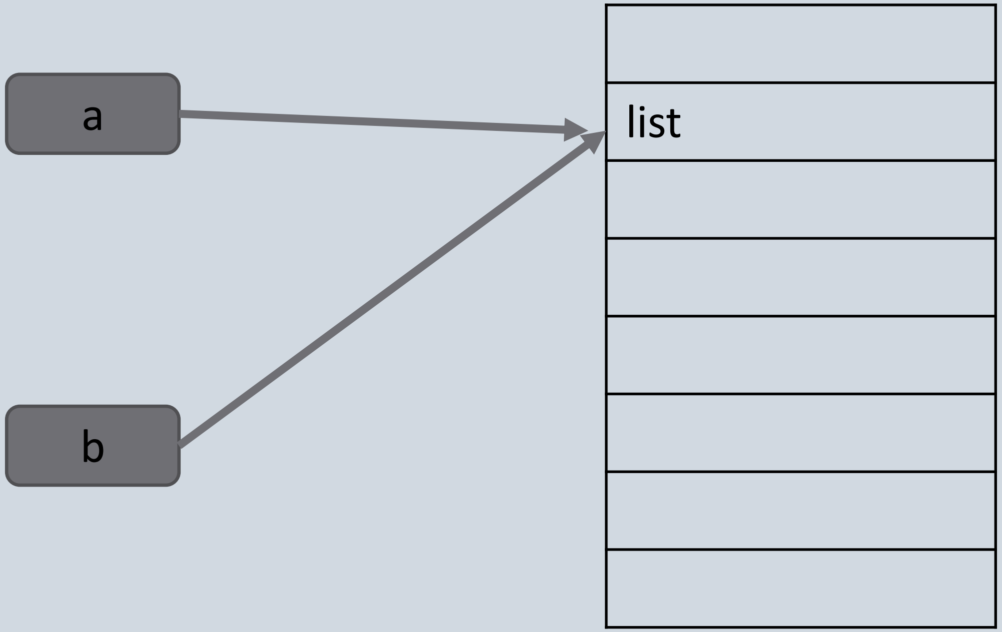


Python Concepts – Mutable Objects 2

Reference:

```
b = ['NewValue', 2, 3]
print('a = ', a)
print('b = ', b)
```

```
a = ['NewValue', 2, 3]
b = ['NewValue', 2, 3]
```



List Collections are ***mutable***

Python – List Comprehension

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
```

- [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

List comprehension

```
squares = [x**2 for x in range(10)]
print(squares)
```

- [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Python Function Definitions And Use

```
def sqr(x):
    s = x**2
    return s
```

print(sqr(2))

- 4

Strings

Strings are “lists” of characters

```
s = 'Hello World'
print(s[1])
```

- e

```
print(s*2)
```

- Hello WorldHello World

```
print(s+s)
```

- Hello WorldHello World

Single quotes: 'allows embedded "double" quotes'

Double quotes: "allows embedded 'single' quotes"

Triple quoted: '''Three single quotes''', """Three double quotes"""“

String with special characters:

- print("tab\tsign, \nnewline")
  - tab	sign,
  - newline

Strings

Raw string:

- print(r"tab\tsign, \nnewline")
  - ➢ tab   sign,
  - ➢ Newline

Formated string:

- print(r"C:\Users\zvi.b\Documents\Zvi\HUJI\Data Science 2024\Code\1. python_recap")
  - ➢ C:\Users\zvi.b\Documents\Zvi\HUJI\Data Science 2024\Code\1. python_recap

String format with input params:

- user = ”zvi"
  print(f"user name {user}")
  print("user name {}".format(user))

➢ user name zvi

➢ user name zvi

Concatenate a string from iterable

- print(",".join(['a', 'b', 'c’]))
- 'a,b,c'

Strings – Other Methods

str.lower()

str.upper()

Include substring “th” in “Python”

str.replace(*old*, *new*[, *count*])

str.split(*sep=None*, *maxsplit=- 1)*

Many other:

https://docs.python.org/3/library/stdtypes.html#textseq

Numpy Intro

- Arrays vs lists: memory efficiency & speed.
- Array creation: np.array, arange, linspace, zeros, random.
- Indexing & slicing: 1D, 2D, boolean masks.
- Broadcasting: operations across rows/columns without loops.
- Vectorized operations: sum, mean, std, dot, exp.

Numpy Arrays vs Lists

- Quick speed comparison (conceptual)

```python
# Arrays vs Lists: quick speed comparison (conceptual)
import time

lst = list(range(1_000_000))  # smaller to keep runtime quick in class
arr = np.array(lst)

t0 = time.time()
_ = sum(lst)
t_list = time.time() - t0

t0 = time.time()
_ = arr.sum()
t_np = time.time() - t0

print(f"Sum Python list: {t_list:.6f}s  |  Sum NumPy array: {t_np:.6f}s")
```

Sum Python list: 0.003378s | Sum NumPy array: 0.000699s

Numpy Array Creation

```
print(np.array([1,2,3]))
print(np.arange(0, 10, 2))
print(np.linspace(0, 1, 5))
print(np.zeros((2,3)))
print(np.random.randint(1, 10, (3,3)))
```

```
[1 2 3]
[0 2 4 6 8]
[0.   0.25 0.5  0.75 1.  ]
[[0. 0. 0.]
 [0. 0. 0.]]
[[6 8 7]
 [5 9 8]
 [8 6 7]]
```

Broadcasting

- **Broadcasting** in NumPy is a set of rules that allows **arithmetic operations** (like addition, subtraction, multiplication, division, etc.) between arrays of **different shapes**, without explicitly replicating data.

```
sales = np.array([[10,20,30],
                  [5,15,25]])
discount = np.array([0.9, 0.8, 0.95])   # per column
print("Sales * discount ->\n", sales * discount)
```

```
Sales * discount ->
[[ 9. 16. 28.5 ]
[ 4.5 12. 23.75]]
```

Array Shape and Reshape

```
# array shape
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

print(arr.shape)
```

```
(2, 4)
```

```
# Reshape
# Convert the following 1-D array with 15 elements into a 2-D array.
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])

newarr = arr.reshape(5, 3)

print(newarr)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]
 [13 14 15]]
```

Simple Arithmetic with Numpy

```python
arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

add_arr = np.add(arr1, arr2)
print('Add arrays', add_arr)

subtract_arr = np.subtract(arr1, arr2)
print('Subtract arrays', subtract_arr)

multiply_arr = np.multiply(arr1, arr2)
print('Multiply arrays', multiply_arr)

divide_arr = np.divide(arr1, arr2)
print('Divide arrays', divide_arr)

power_arr = np.power(arr1, arr2)
print('Power arrays', power_arr)

prod_arr = np.prod([arr1, arr2])
print('Product of arrays', prod_arr)

# product of single array
arr = np.array([1, 2, 3, 4])
x_prod_array = np.prod(arr)
print('Product of single array', x_prod_array)
```

```
Add arrays
[30 32 34 36 38 40]
Subtract arrays
[-10 -10 -10 -10 -10 -10]
Multiply arrays
[200 231 264 299 336 375]
Divide arrays [0.5 0.52380952 0.54545455 0.56521739
0.58333333 0.6 ]
Power arrays [ 1661992960 602408795 0 1487897765
1090519040 -1144744561]
Product of arrays 872070144
Product of single array 24
```

Pandas

Pandas is a data analysis library

- Import/export data from/to multiple file formats
- Data Transformations
- Data Analysis
- Vectors and tables

Pandas installation

- Standard package coming with anaconda installations

Activate an environment and install pandas

- conda activate “env_name”
- conda install pandas
  - Or
- pip install pandas

Pandas Data Structure

```
import pandas as pd
```

```
pd.Series(…)
```

- One dimensional vector of values

```
pd.DataFrame(…)
```

- Two-dimensional table
- Collection of equal length Series

Pandas Series

class pandas.Series(data=None, index=None, dtype=None, name=None, copy=False)

```
import numpy as np
import pandas as pd
pd.Series([1,2,3])
```

Out[14]:

a 1

b 2

c 3

dtype: int64

```
pd.Series([1,2,3],  index=['a', 'b', 'c'])
```

Out[15]:

a 1

b 2

c 3

dtype: int64

```
ser = pd.Series({'a': 1, 'b': 2, 'c': 3})
```

```
ser.shape
```

```
(3,)
```

Row vs Columnar Databases

Row oriented databases

- Designed for adding/deleting rows to a database
- OLTP (online transactional processing)
- not efficient doing calculation on a small number of columns
- Not efficient in adding/removing columns

Columnar oriented dataset

- Works better for data analysis
- Very efficient in adding new column
- Inefficient in adding or deleting rows

Pandas - Dataframe

pandas.DataFrame(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=None*)

Columnar Data Structure.

- Columns are pandas.Series

Examples:

```
d = {'col1': [1, 2, 3], 'col2': [3, 4, 4]}
df = pd.DataFrame(data=d)
print(df)
```

```
   col1  col2
0     1     3
1     2     4
```

```
df.shape
```

```
(3, 2)
```

Pandas - Dataframe

```
df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8,
9]]),columns=['a', 'b', 'c'])
print(df2)
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

Pandas Object Attributes

```
index = pd.date_range("1/1/2003", periods=8)
df = pd.DataFrame(np.random.randn(8, 3), index=index, columns=["A", "B", "C"])
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
#long_series = pd.Series(np.random.randn(1000))
#print(long_series)
#print(long_series.head(2))
print(s.tail(3))
```

```
c 0.580021
d -0.022711
e 0.395441
dtype: float64
```

Pandas Basic Functions

df_s=df.shape : returns the dimensions of a dataframe

df_s[0] : number of rows

df_s[1] : number of columns

len(s): length of series

sss.isna(): returns Boolean Series indicating missing values

```
print("df shape:", df.shape)
print("len(s):", len(s))
s1 = pd.Series([1 ,np.nan ,None ,3 ,5], index=["a", "b", "c", "d", "e"])
print("first element in s1:", s1[0])
print("na in s1:", s1.isna())
print(df.head())
print(df['A'])
print(type(df['B']))
```

```
df shape: (8, 3)
len(s): 5
first element in s1: 1.0
na in s1:
a False
b True
c True
d False
e False
dtype: bool
```

Pandas Indexing And Selection [X]

Returns the element having index X

- Series: `s[‘a’]` returns a scalar with the element stored in index ‘a’
- DataFrame: `df[‘a’]` - returns the column name ‘a’ as pandas.Series
- DataFrame: `df[[‘a’, ‘b’]]`– returns a dataframe with columns [‘a’, ‘b’]

Pandas Indexing And Selection .Loc[]

Returns the element with the index X

- Series: `s.loc[‘a’]` returns a scalar with the element stored in index ‘a’
- DataFrame: `df.loc[‘a’]` - returns row name ‘a’ as pandas.Series
- DataFrame: `df.loc[‘a’,’b’]` returns a scalar

Pandas Indexing And Selection .Iloc[]

Returns the numeric position of

- Series: `s.iloc[n]` returns a scalar with the element position *n*
- DataFrame: `df.iloc[n]` - returns row position *n* as pandas.Series
- DataFrame: `df.iloc[n,k]` returns a scalar

Pandas Indexing And Selection By Logical Exp.

b is Boolean vector with length Returns the numeric position of

- Series: `s.loc[~ s.isna()]` returns all elements that are not ‘na’
- DataFrame: `df.loc[~ df.col1.isna()]` – returns a dataframe with rows that values in col1 are not missing
- DataFrame: `df.loc[df.col1.isna(), [‘c1’,’c2’,’c3’]]`

Pandas Object Attributes

```
print('object size')
print("DataFrame")
print(df.shape)
print("Series")
print(len(s))
```

```
object size
DataFrame
(8, 3)
Series
5
```

Pandas Apply

DataFrame.apply(*func*, *axis=0*, *raw=False*, *result_type=None*, *args=()*, ***kwargs*)

```
df = pd.DataFrame([[4, 9]] * 3, columns=['A', 'B'])
print(df)
```

	A	B
0	4	9
1	4	9
2	4	9

Pandas Apply

DataFrame.apply(*func*, *axis=0*, *raw=False*, *result_type=None*, *args=()*, ***kwargs*)

```
print(df.apply(np.sqrt))
```

```
     A    B
0  2.0  3.0
1  2.0  3.0
2  2.0  3.0
```

Pandas Apply

DataFrame.apply(*func*, *axis=0*, *raw=False*, *result_type=None*, *args=()*, ****kwargs*)

```
df.apply(np.sum, axis=0)
```

```
A     12
B     27
dtype: int64
```

```
df.apply(np.sum, axis=1)
```

```
0   13
1   13
2   13
dtype: int64
```

The lambda Function

Syntax: Arguments: expression

```
f=lambda a,b: a**b
f(2,3)
```

Out[13]: 5

```
# df.apply with axis=1 applies the lambda function to each
column.
df2= df.apply(lambda x: x.A + x.B, axis=1)
print(df2)
```

```
2003-01-01 -1.968880
2003-01-02 -2.487895
2003-01-03 1.415460
2003-01-04 -0.679664
2003-01-05 -0.290989
2003-01-06 -1.290294
2003-01-07 0.836008
2003-01-08 2.673087
Freq: D, dtype: float64
```

```
# df.apply with axis=0 applies the lambda function to each
row.
df2= df.apply(lambda x: x[0] + x[1], axis=0)
```

```
A -1.749245
B -2.707530
C 3.115878
dtype: float64
```


59

lambda with other Function

Combining lambda with other function:

```
def add_one(num):
    return num+1
df.apply(lambda x: add_one(x['2003-01-05']), axis=0)
```

```
A -1.749245
B -2.707530
C 3.115878
dtype: float64
```

Categorical Encoding

Machine learning models require a numeric input and generate a numeric output.

How can we handle categorical (string) variable?

Binary features :

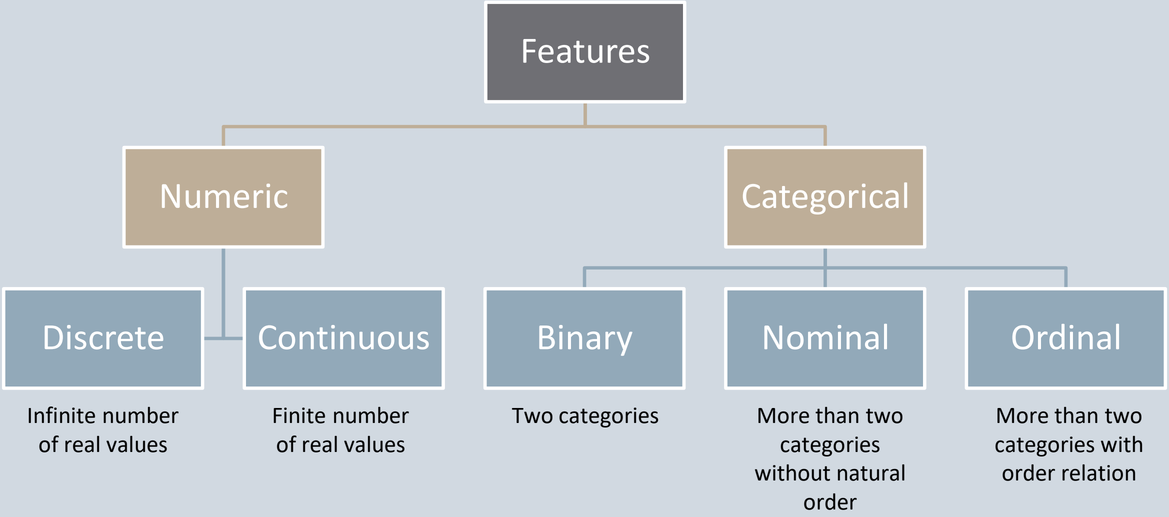
- 1/0

Ordinal features:

- encodes the values as integer.

Nominal features:

- One-Hot Encoding: encodes the values as a binary vector array.
  - pd.get_dummies(**df.cut**)
  - pd.get_dummies(df.cut, **prefix=cut'**)
- Dummy Variable Encoding: same as One-Hot Encoding, but one less column.
  - There is some redundancy in One-Hot encoding
  - pd.get_dummies(df.cut, **drop_first=True**)
- pd.**concat**(**[df, embarked_dummies]**, axis=1)



Word Representations (one-hot encoding)

- ML Algorithms Work with Numbers
- One-hot encoding
  - Creates a **binary column for each category**.
  - Each word is represented by a vector, W of the size of the vocabulary with a single non-zero value.

$$[0 \quad 0 \quad 0 \quad 0 \quad \ldots 1 \quad \ldots \quad 0 \quad 0 \quad 0]$$

$$w_{i,j} = \begin{cases} 1 & for\ i = j \\ 0 & for\ i \neq j \end{cases}$$

$$|w_i| = number\ of\ terms\ in\ the\ vocabulary$$

	Cat	Dog	Horse	Owl
Cat	1	0	0	0
Dog	0	1	0	0
Horse	0	0	1	0
Owl	0	0	0	1

- Dummy Variables
  - A simplified version of one-hot encoding.
  - To avoid multicollinearity (perfect correlation between features), one category column is dropped.

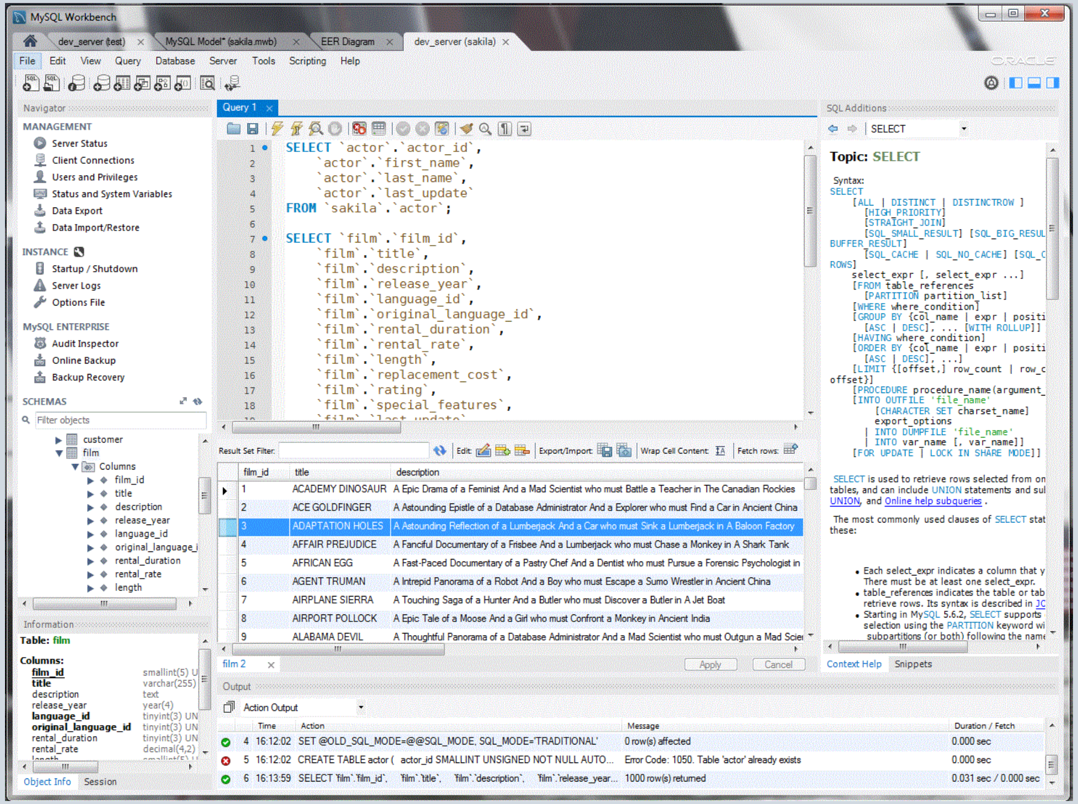
MySQL for Data Science

- Why SQL in Data Science?
  - Most data lives in relational databases (MySQL, PostgreSQL, etc.)
  - Enables efficient data extraction & transformation
  - SQL queries are optimized for performance
  - Works seamlessly with Python, R, and BI tools
  - Ensures data consistency, reliability, and scalability
- Relational Database Concepts:
  - Tables → store structured data (like spreadsheets)
  - Rows (records) → represent individual data entries
  - Columns (fields) → attributes of the data
  - Primary Key (PK) → uniquely identifies each row
  - Foreign Key (FK) → connects rows across tables

https://www.mysql.com/

MySQL Workbench

- GUI tool to manage MySQL databases
- Features:
  - Visual schema design
  - Query editor
  - Data import/export
  - Server monitoring
- Download:
  - https://dev.mysql.com/downloads/workbench/

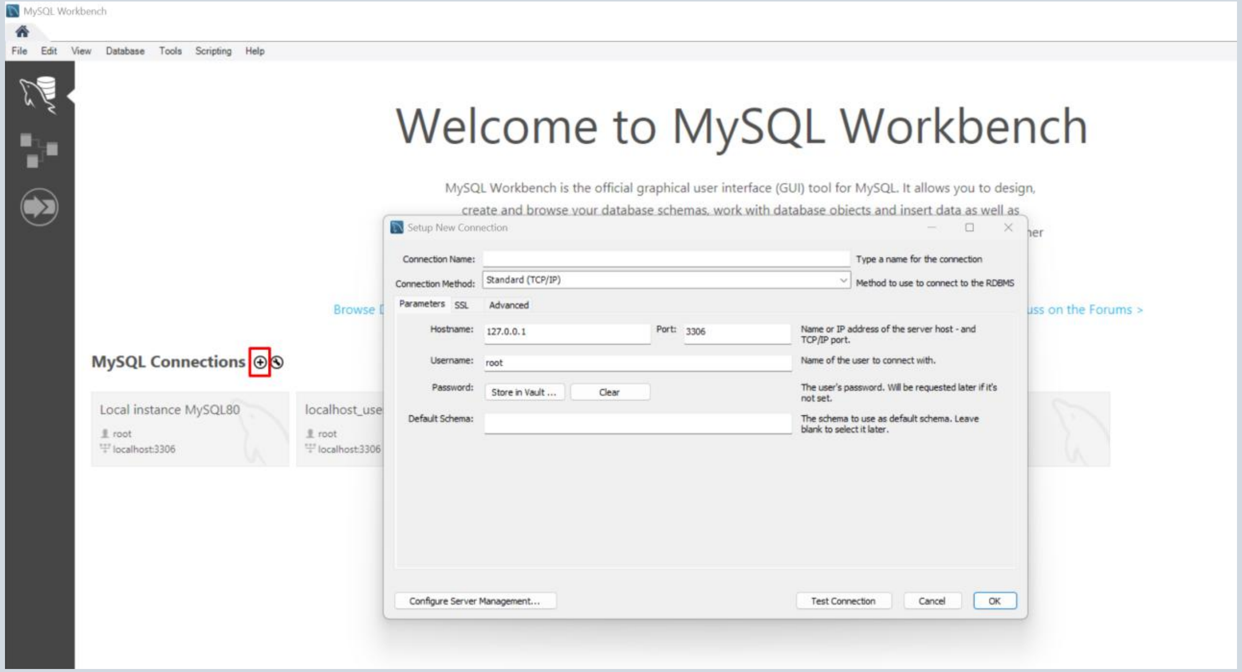


Create New Database in MySQL Workbench

1. Open MySQL Workbench
   - Launch MySQL Workbench and connect to your MySQL server.

2. Open a New SQL Tab
   - Click on SQL + (new query tab) in the toolbar.
   - This opens a workspace where you can run SQL commands.

3. Create a Database (Schema)In the query editor, type:
   - CREATE DATABASE my_database;
   - Press Execute (lightning icon).
   - my_database will now appear under Schemas in the Navigator



SQL Basics

Create a Table:

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT);
```

Insert Data:

```
INSERT INTO users (user_id, name, age)
VALUES
(1, 'Alice', 28),
(2, 'Bob', 35),
(3, 'Charlie', 22);
```

SELECT all rows:

```
SELECT * FROM users;
```

Filter with WHERE

```
SELECT * FROM users
WHERE age > 25;
```

Sort with ORDER BY

```
SELECT * FROM users
ORDER BY age DESC;
```

Tutorials:

- https://www.w3schools.com/MySQL/default.asp
- https://www.geeksforgeeks.org/mysql/mysql-tutorial/

Python Connector

- Use MySQL connection (requires running MySQL server & Python driver)
- Or SQLite fallback that create an in-memory DB and a transactions table (does not requires running MySQL server)

```
# --- MySQL connection (requires running MySQL server &
Python driver) ---
import mysql.connector

conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="AIdev2025",
    database="demo_llm"
)
# --- SQLite fallback: create an in-memory DB and a
transactions table ---
# import sqlite3

# conn = sqlite3.connect(":memory:")
```

Create and Read Table

```
cur = conn.cursor()

# Drop table if it already exists (optional, for clean re-runs)
cur.execute("DROP TABLE IF EXISTS transactions;")

# Create table
cur.execute("""
CREATE TABLE transactions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    product VARCHAR(50),
    quantity INT,
    price DECIMAL(10,2),
    discount DECIMAL(5,2),
    date DATE
);
""")
```

```
# Seed data
rows = [
    ("A", 1, 10.0, 0.00, "2024-01-01"),
    ("B", 2, 25.0, 0.10, "2024-01-01"),
    ("A", 3, 10.0, 0.05, "2024-01-02"),
    ("C", 1, 40.0, 0.00, "2024-01-03"),
    ("B", 2, 25.0, 0.00, "2024-01-04"),
    ("C", 1, 40.0, 0.15, "2024-01-05"),
    ("A", 2, 10.0, 0.00, "2024-01-06"),
]

# Use %s placeholders for MySQL
cur.executemany(
    "INSERT INTO transactions (product, quantity, price, discount,
date) VALUES (%s, %s, %s, %s, %s)",
    rows
)
conn.commit()

# Read into pandas DataFrame
df_sql = pd.read_sql("SELECT * FROM transactions", conn)
```

Top Products by Revenue

```python
df_top_sql = pd.read_sql_query(
    """
    SELECT product, SUM(quantity*price*(1-discount)) AS revenue
    FROM transactions
    GROUP BY product
    ORDER BY revenue DESC
    LIMIT 5
    """, conn
)
print("SQL result:\n", df_top_sql)

# Pandas equivalent
df_sql["Revenue"] = df_sql["quantity"] * df_sql["price"] * (1 -
df_sql["discount"])
df_top_pd =
df_sql.groupby("product")["Revenue"].sum().sort_values(ascending=False).
head(5).reset_index()
print("\nPandas result:\n", df_top_pd)
```

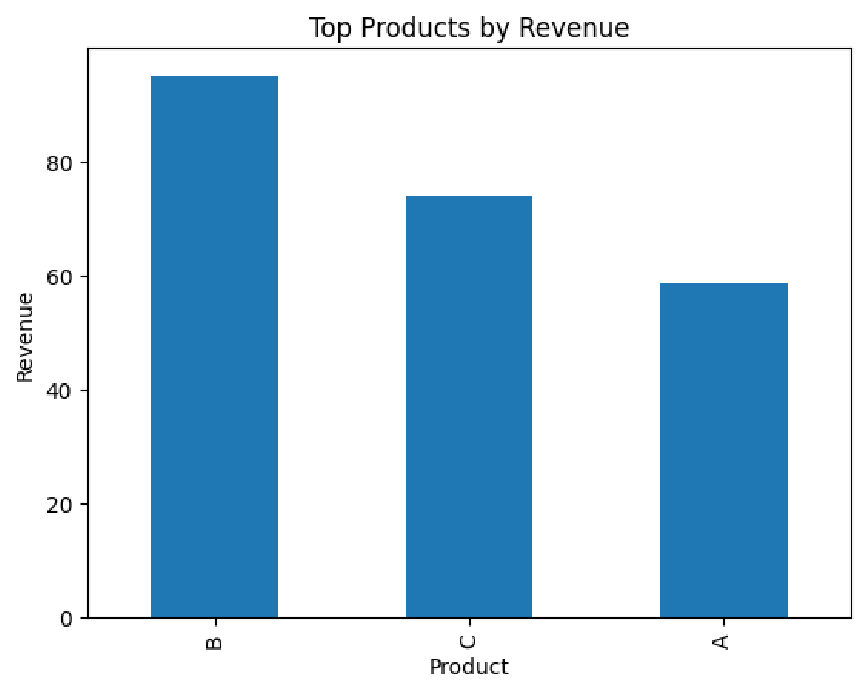
```
SQL result:
   product  revenue
0       B     95.0
1       C     74.0
2       A     58.5

Pandas result:
   product  Revenue
0       B     95.0
1       C     74.0
2       A     58.5
```

Top Products by Revenue

- Simple plot: top products by revenue (from Pandas result)

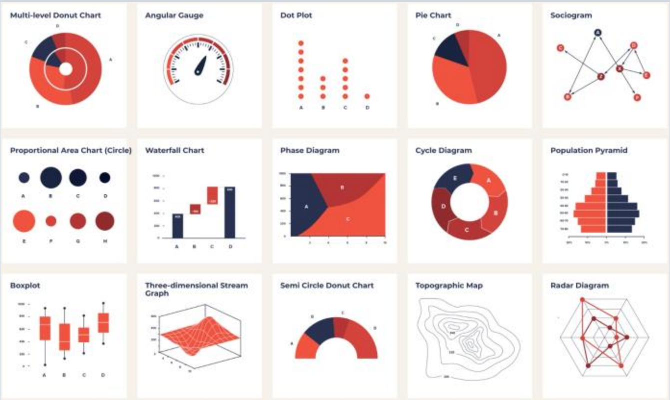
```
import matplotlib.pyplot as plt
plt.figure()
df_top_pd.plot(x="product", y="Revenue", kind="bar", legend=False)
plt.title("Top Products by Revenue")
plt.xlabel("Product")
plt.ylabel("Revenue")
plt.show()
```



Exploratory Data Analysis (EDA)

- Exploratory Data Analysis (EDA) helps in :
  - Better understanding of data.
  - Identifying obvious patterns in the data.
  - Better understanding of the business problem.

- The objective of Exploratory Data Analysis (EDA) is to gain an initial understanding of the data by examining distributions, relationships, missingness, correlations, outliers, and trends to guide further analysis and modeling.

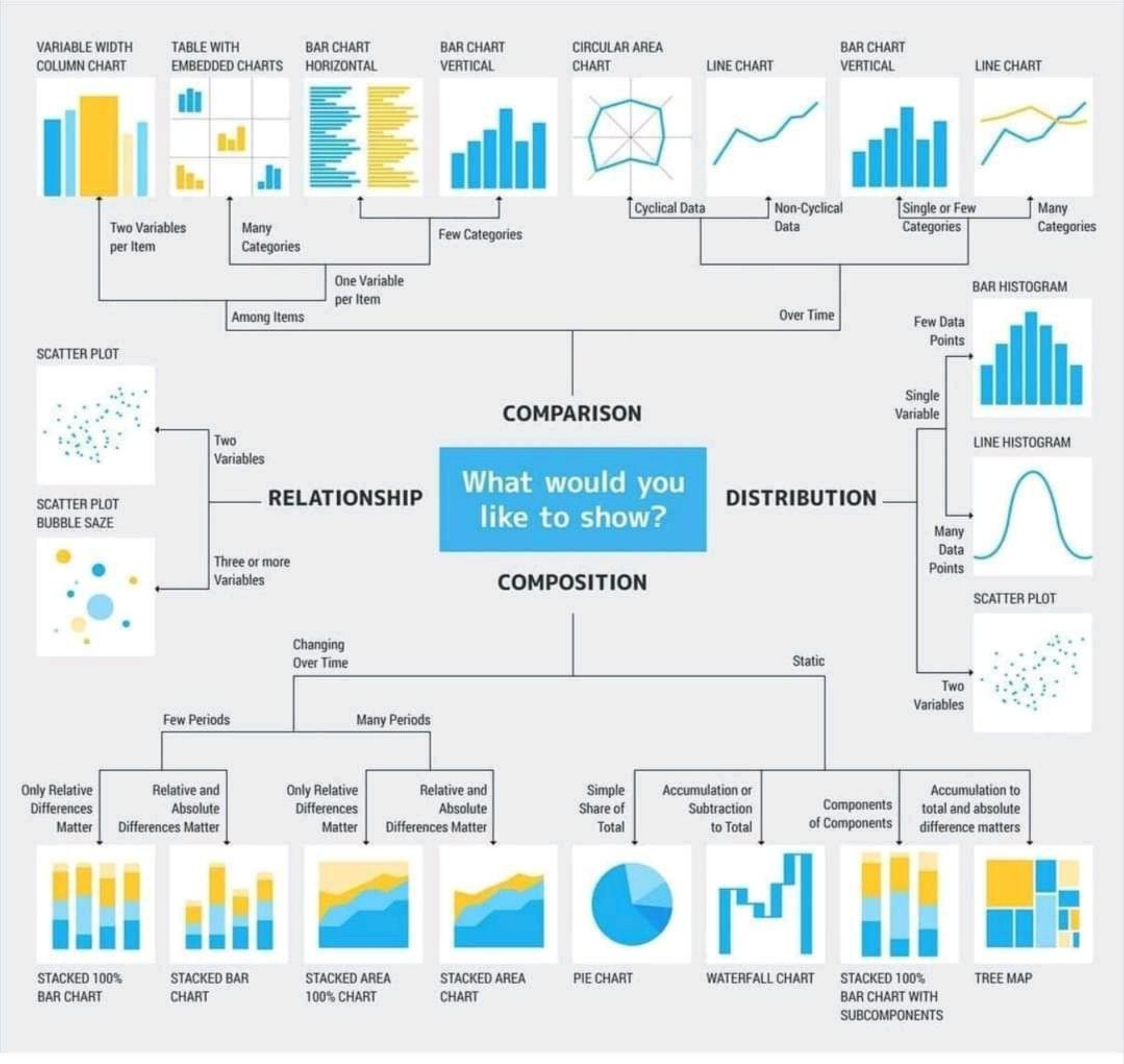


Source: https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/

Plots

Common plots types:

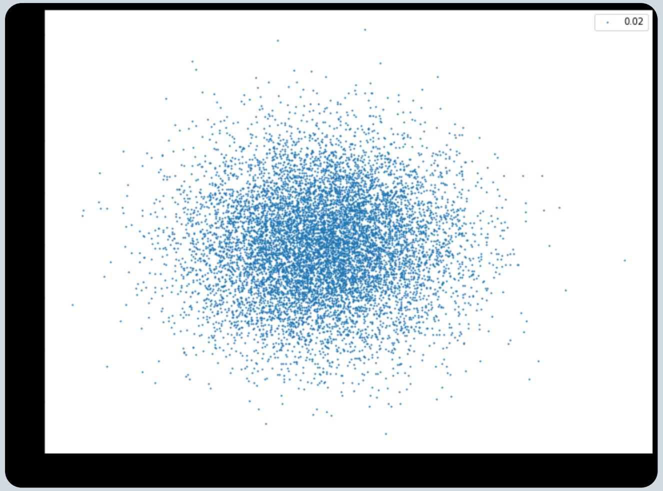
- Pie
- Bars
- Scatters
- Histograms
- Lines
- Heatmaps



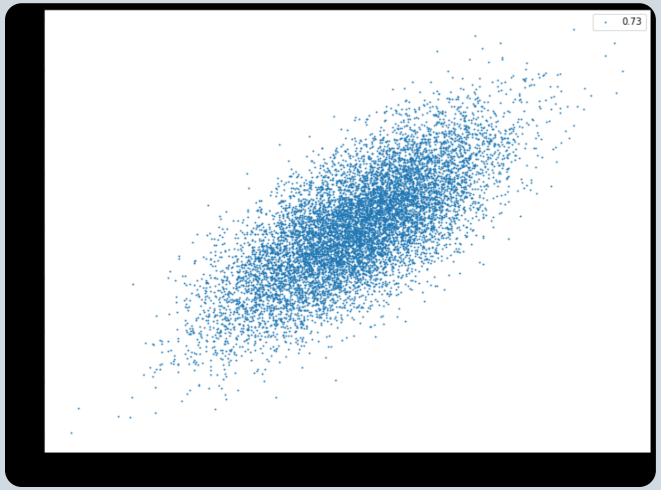
Packages:

- Matplotlib
  - https://matplotlib.org/stable/plot_types/index.html
- Pandas Plots:
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html
- Plotly
  - https://plotly.com/python/
- Seaborn
  - https://seaborn.pydata.org/examples/index.html

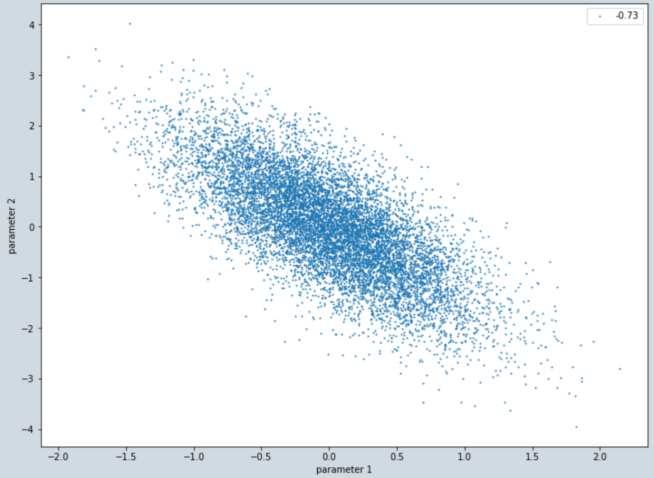
EDA: Scatter Plot



Not correlated



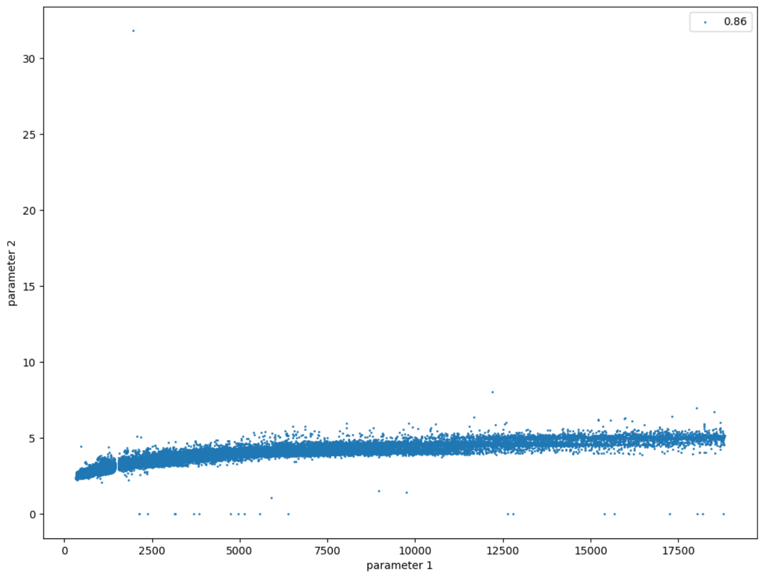
Positively correlated



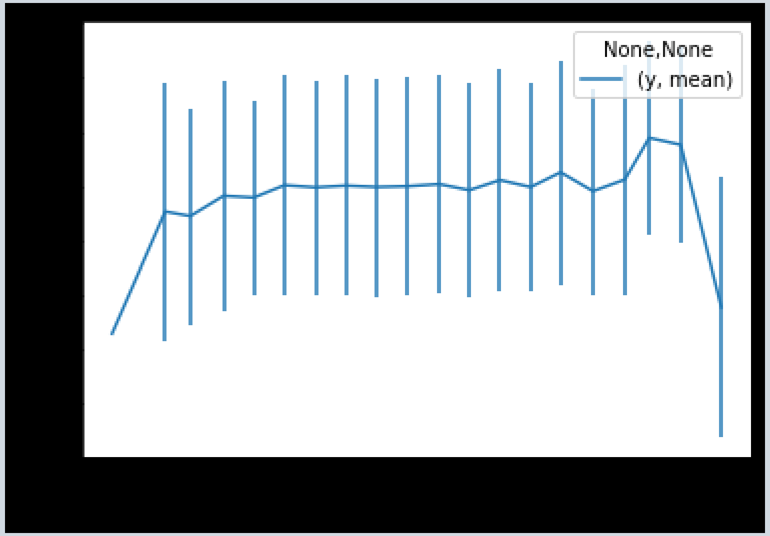
Negatively correlated

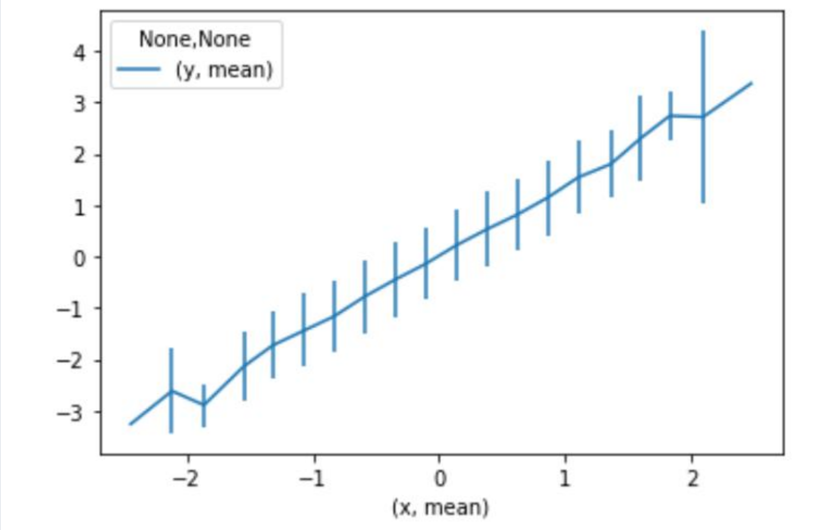
EDA: Scatter Plot

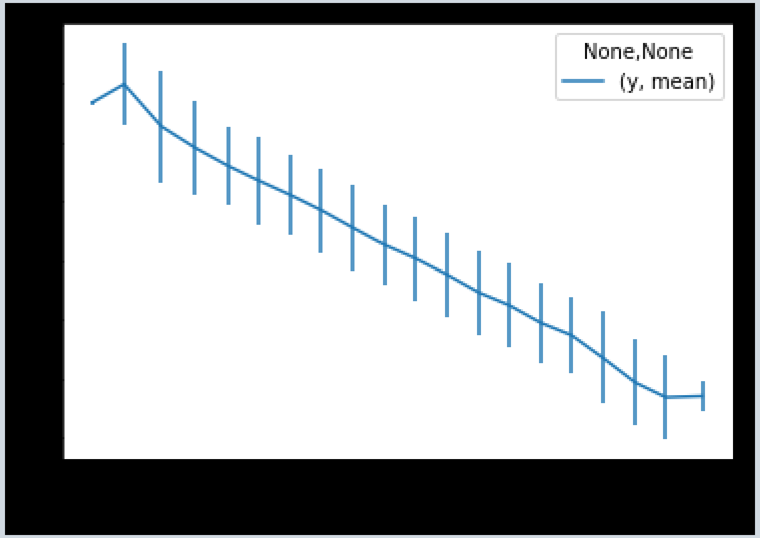
```
def plot_2vars(x,y):
    import pandas as pd
    data = pd.DataFrame({'x':x,'y':y})
    ax=data.plot.scatter(x='x',y='y', s=1, label = f"{data['x'].corr(data['y']):2.2f}")
    ax.legend(); ax.set_xlabel('parameter 1'); ax.set_ylabel('parameter 2');
    return ax
```



EDA: Reveal Functional Dependence

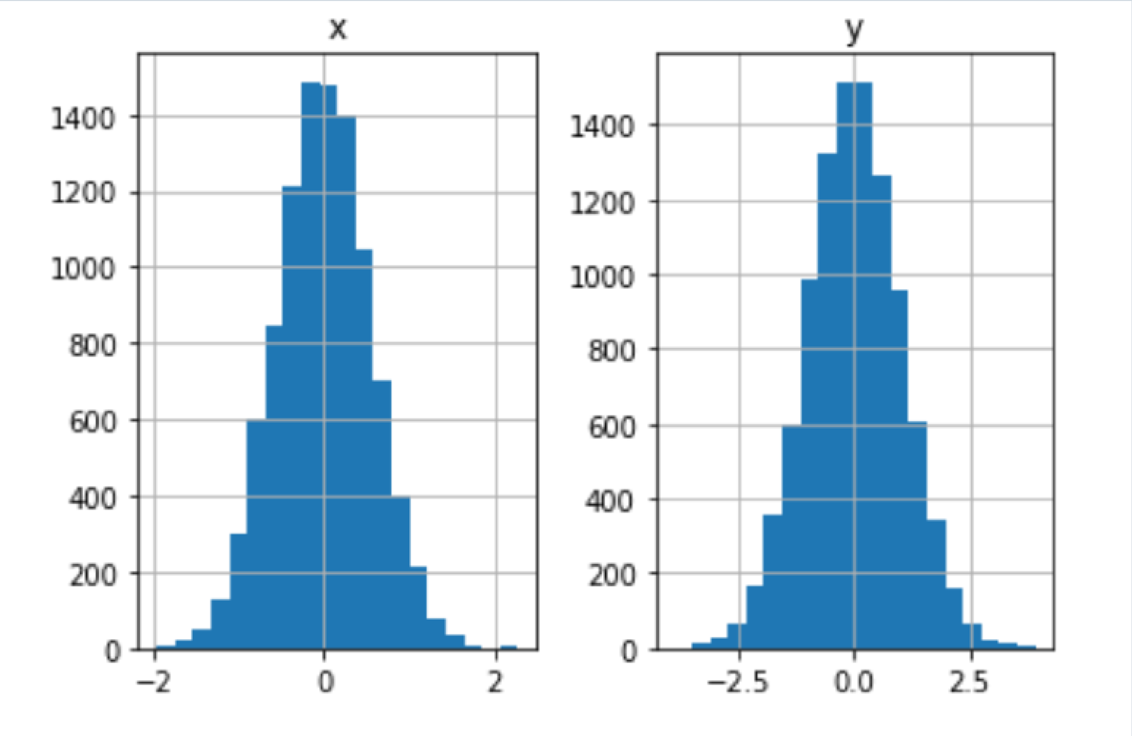







```python
def plot_averaged_line(x,y):
    import pandas as pd
    data = pd.DataFrame({'x':x,'y':y})
    categories  = pd.cut(data['x'],bins=20)
    averaged_data = data.groupby(categories).agg({'x':['mean'], 'y':['mean','std']})
    averaged_data.plot(x=('x','mean'), y=('y','mean'), yerr=averaged_data[('y','std')])
```

EDA: Histogram




```
data = pd.DataFrame({'x':x,'y':y})
data.hist(bins=20);
```

SOME Automatic EDA packages

pandas-profiling

- https://github.com/ydataai/pandas-profiling
- https://www.analyticsvidhya.com/blog/2021/06/generate-reports-using-pandas-profiling-deploy-using-streamlit/

D-Tale

- D-Tale is the combination of a Flask back-end and a React front-end to bring you an easy way to view & analyze Pandas data structures
- https://github.com/man-group/dtale

Sweetviz and AutoViz

- https://pypi.org/project/sweetviz/
- https://github.com/AutoViML/AutoViz
- https://analyticsindiamag.com/tips-for-automating-eda-using-pandas-profiling-sweetviz-and-autoviz-in-python

Manual vs Automatic EDA

Manual EDA	Automatic EDA
Custom, deep insights	Quick overview & summary
Flexible & domain-driven	Prebuilt, standardized analysis
Slower but more control	Faster, less flexible
Best for teaching & fine-tuning	Best for large data & first look

THANK YOU FOR LISTENING

ZVI.BENAMI@MAIL.HUJI.AC.IL