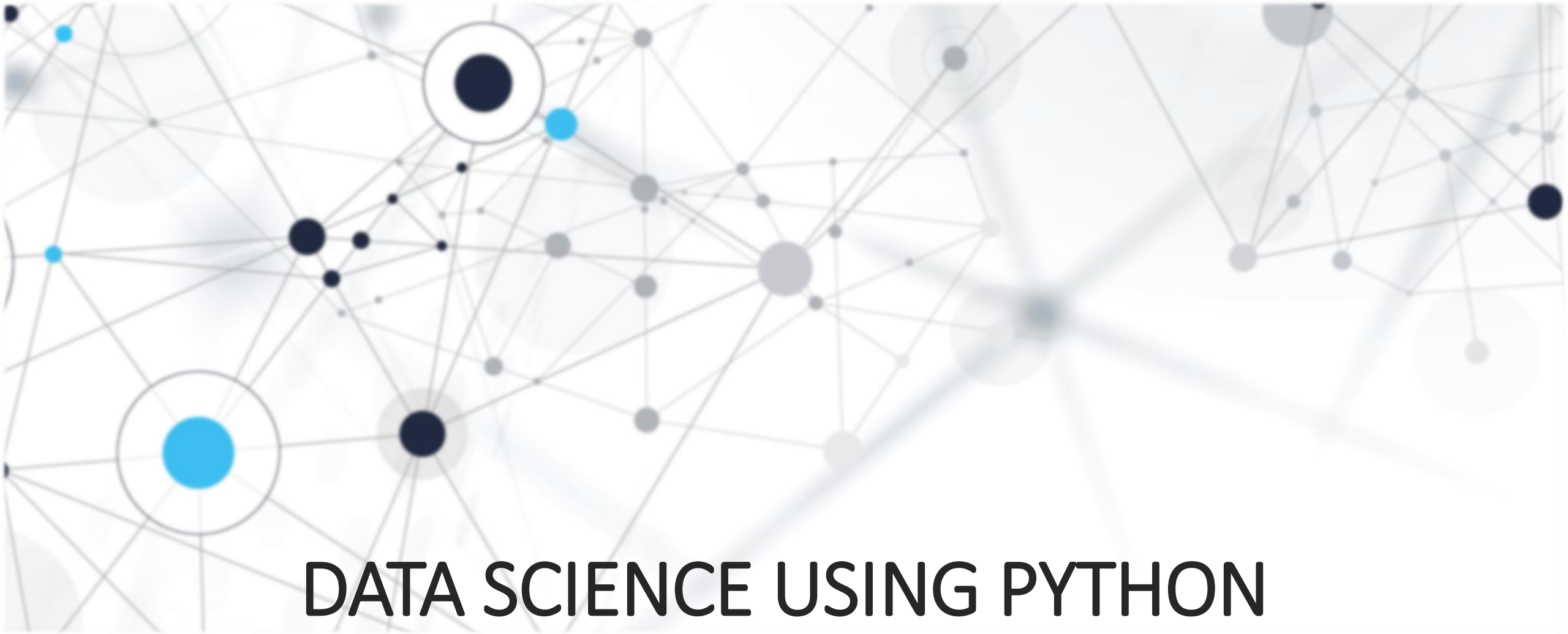


DATA SCIENCE USING PYTHON
AN INTRODUCTORY PERSPECTIVE

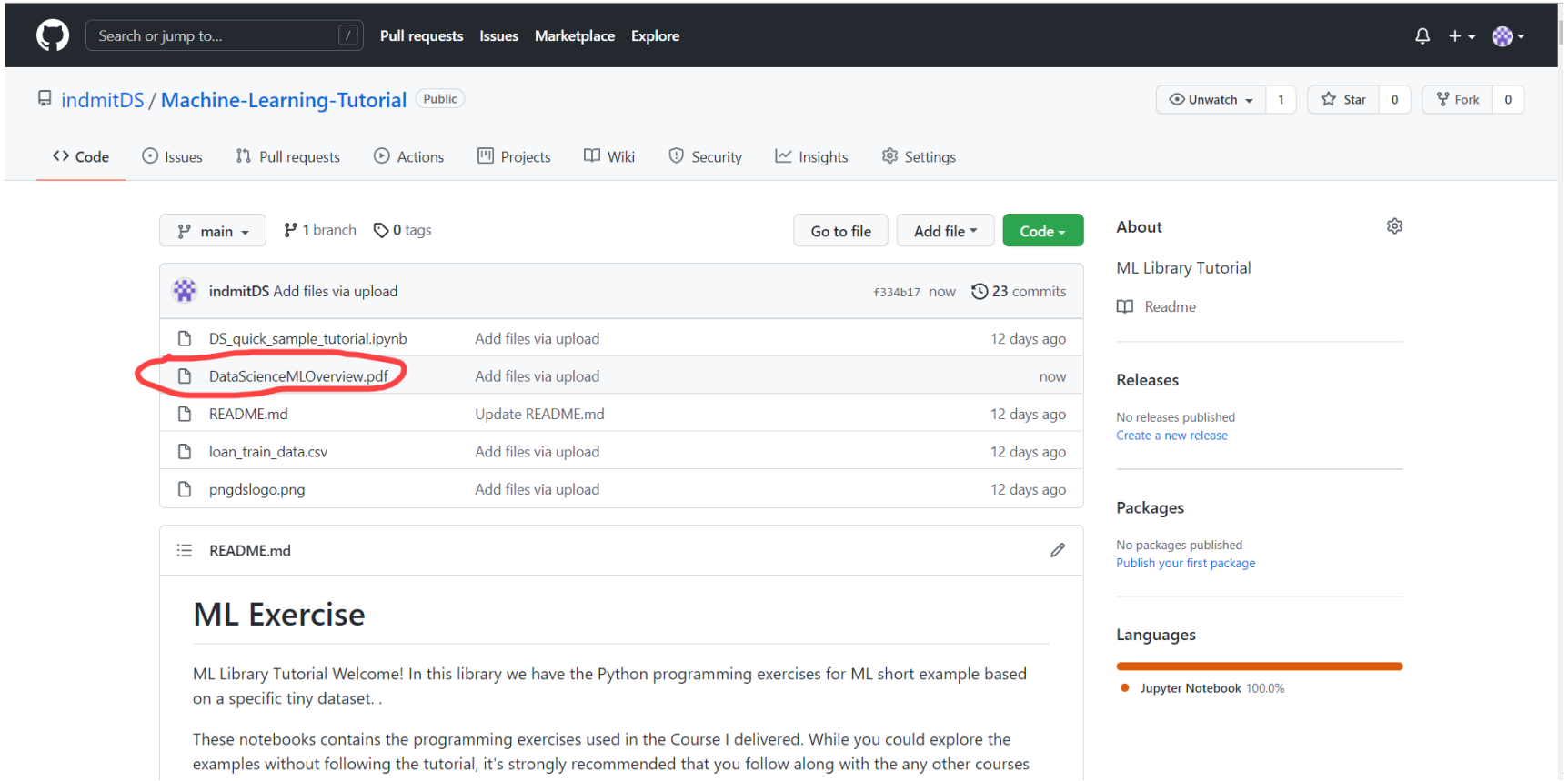
OBJECTIVE OF THE PRESENTATION

- Provide an in-Breadth review instead of in-depth on the current mainstream topics of Data Science (DS)
- Focus on the minimum essentials required to provide a glimpse on what DS is all about, so that based on interests one can redirect focus and brainstorming to choose the topics of interest
- Touch on the mechanics of Machine Learning (ML) as a vehicle for DS
- Provide a peek on the tools to build ML/DS frameworks[a]
- Walk through a possible curriculum suited for Grad/Undergrad Courses

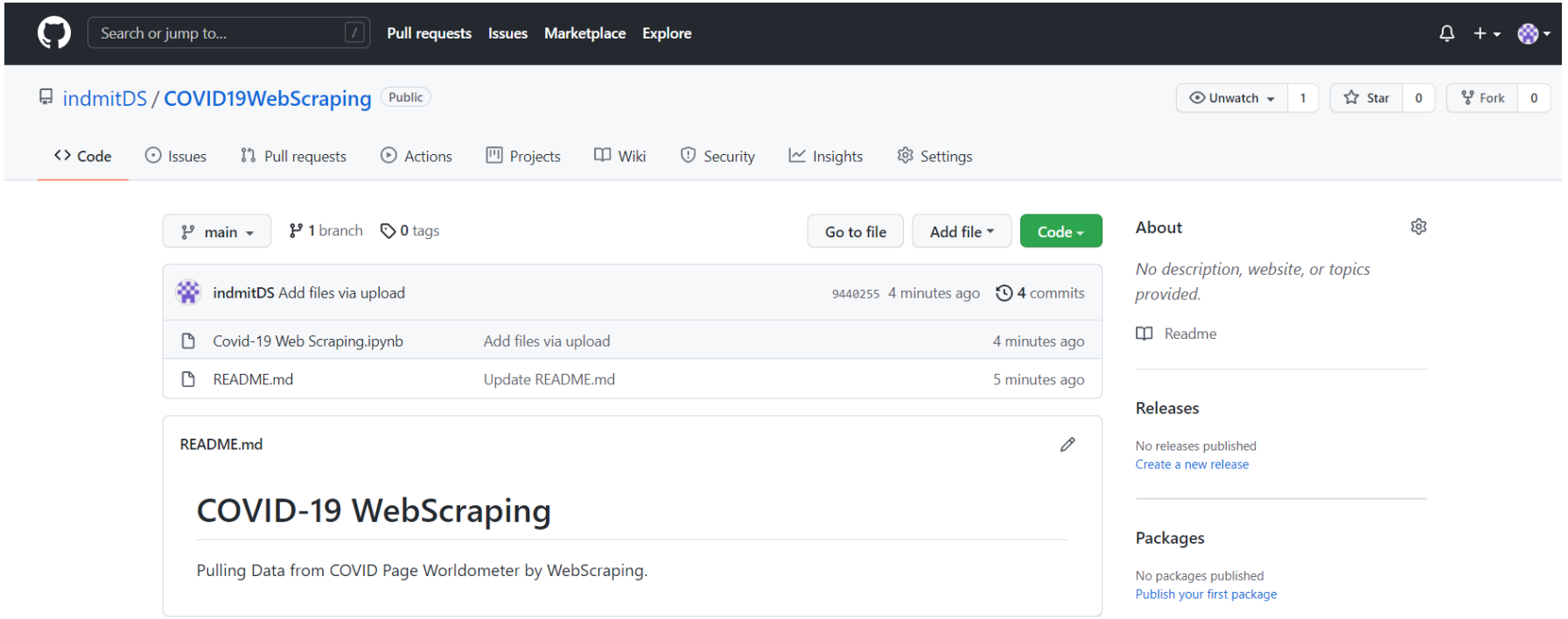
[a]Most material are public and the associated literature references are posted on github

TUTORIAL RESOURCES

(https://github.com/indmitDS/Machine-Learning-Tutorial)



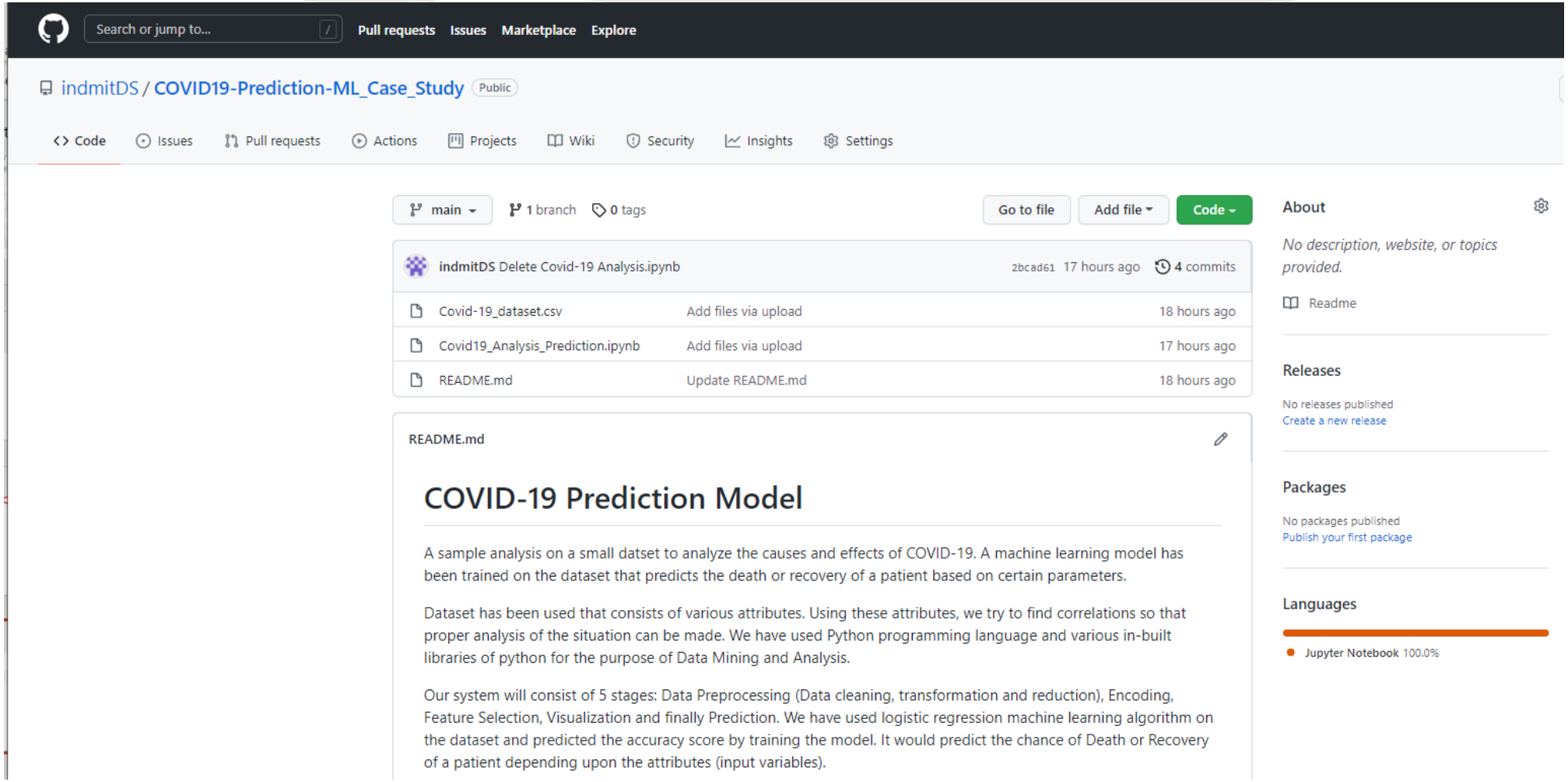
COVID WEB SCRAPING



COVID DATA SCRAPING FROM WEB

Github link

COVID CASE STUDY



COVID ANALYSIS USING LOGISTIC REGRESSION: Github link

OUTLINE OF THE TALK

- Data Science – A Brief Overview (10)
- ML and Deep learning, the main driver of Modern Data Science (30)
- Frameworks (Python & SQL) (Python Data Structures) (30)
- Environment Setup and Get Started
- COVID Data Scraping from WEB (10)
- COVID Case Study Using Python (10)
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

OUTLINE

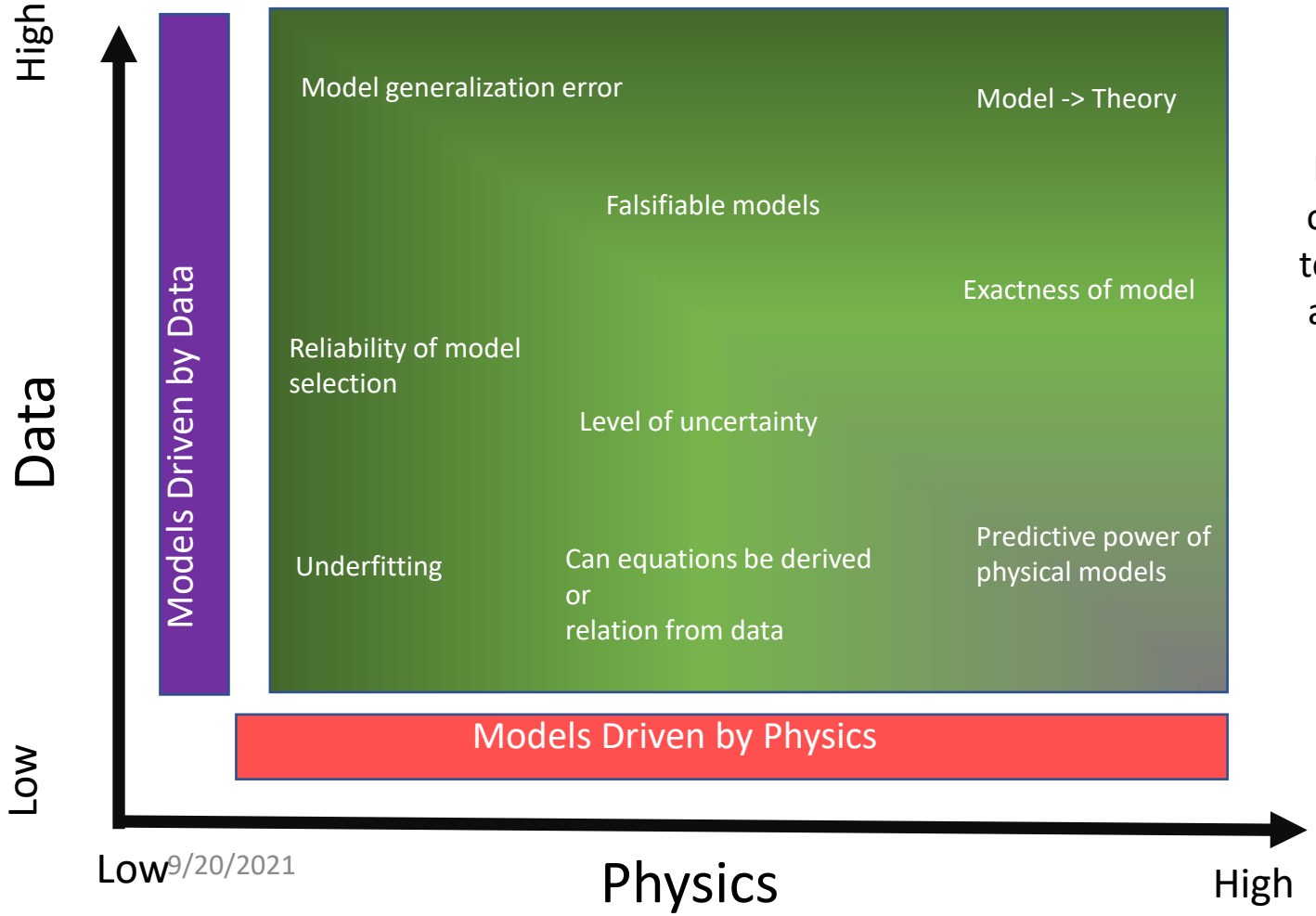
- Data Science – A Brief Overview
- ML and Deep learning, the main driver of Modern Data Science
- Frameworks (Python & SQL) (Python Data Structures)
- Environment Setup and Get Started
- COVID Data Scraping from WEB
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

PHYSICS AND DATA SCIENCE

PHYSICIST	DATA SCIENTIST
Primarily driven by problems of nature	Primarily driven by business problems (till date)
Collect data from experiments	Collect data from data sources/experiments
Primarily applied on quantitative data	Applied on quantitative as well as categorical data
Analyze the collected data	Analyze the collected data
Build models to explain observed data	Build models to explain observed data
Predictive power of models on new data	Predictive analytics & power of models on new data
Methodologies rely on creating exact models/theories	Methodologies are mostly based on developing algorithms and minimizing model errors.
	Train Machine to Learn -> AI **Let Data do the talk**

MODELS IN PERSPECTIVE

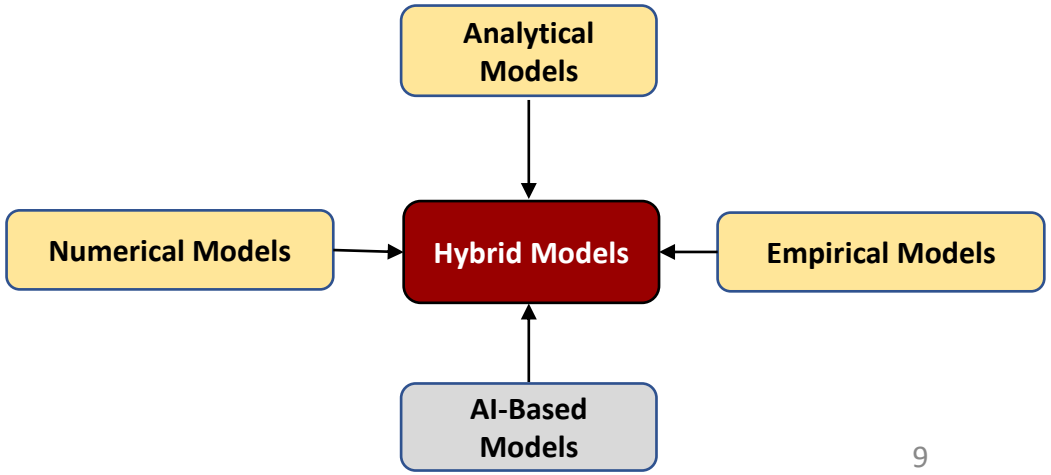


The Physics-Based View

Have a good understanding of the physical system. Able to describe it mathematically and can produce theoretical predictions of system behavior.

The Data Science View

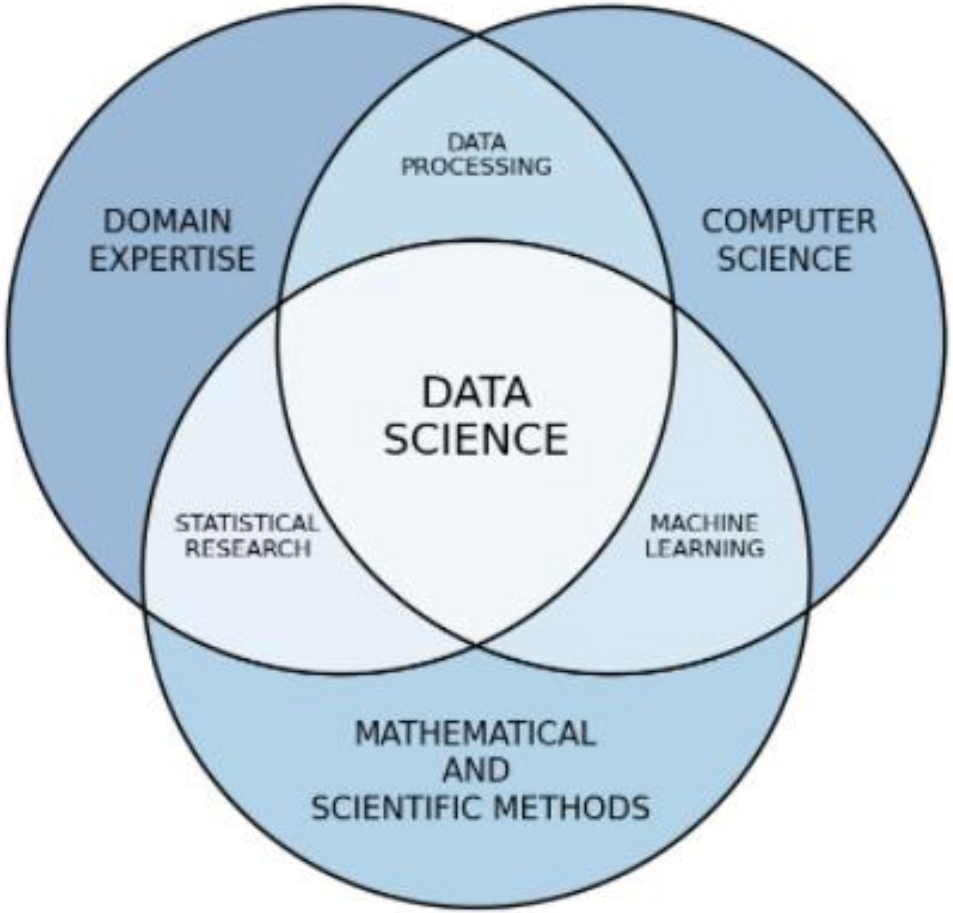
No direct theoretical knowledge about the system but have a lot of experimental data on how it behaves and can learn the physics and make predictions.



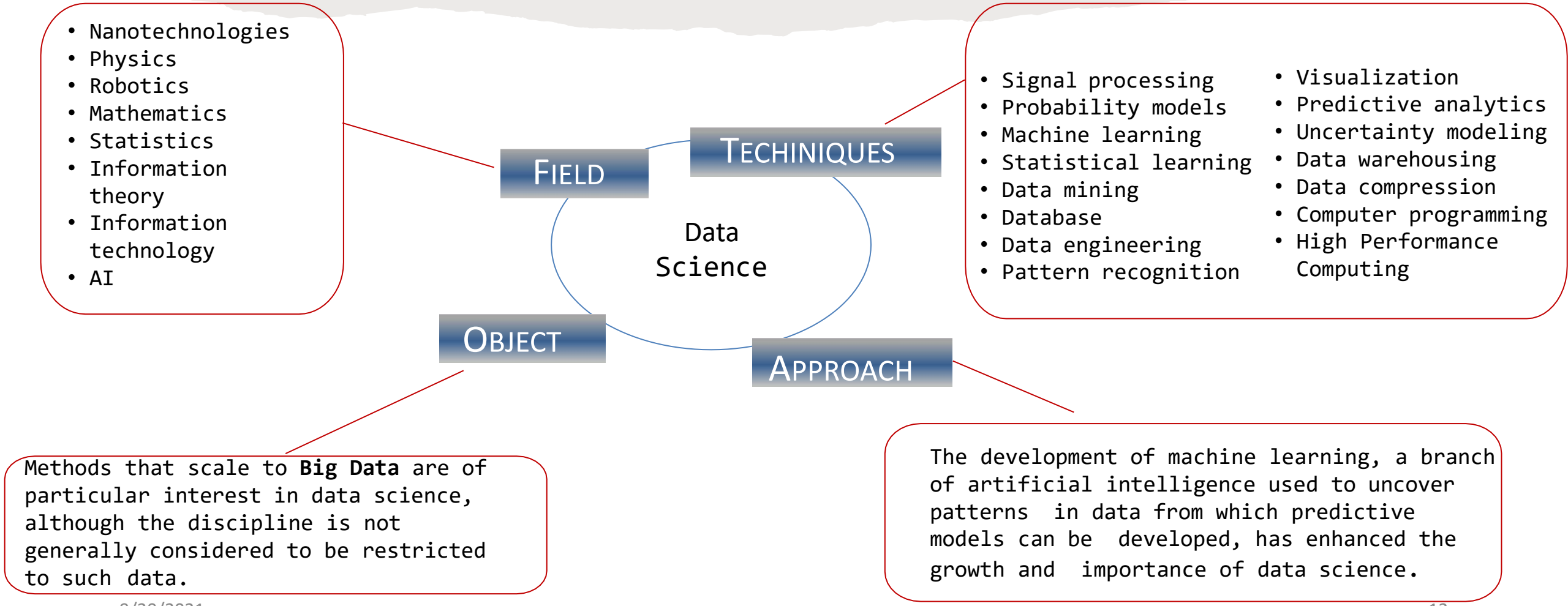
WHAT IS DATA SCIENCE?

- An area that manages, manipulates, extracts, and interprets knowledge from tremendous amount of data
- Data science (DS) is a multidisciplinary field of study with goal to address the challenges in big data
- Data science principles apply to all data – big and small
- Theories and techniques from diverse fields for decision making

DATA SCIENCE - NUTSHELL



DATA SCIENCE LANDSCAPE



5 Vs of BIG DATA

- **Big Data** is any data that is expensive to manage and hard to extract value from

  - Volume
    - The size of the data
  - Velocity
    - The latency of data processing relative to the growing demand for interactivity
  - Variety and Complexity
    - The diversity of sources, formats, quality, structures.
  - Veracity
    - Data quality
  - Value
    - Information for decision making

TYPES OF DATA

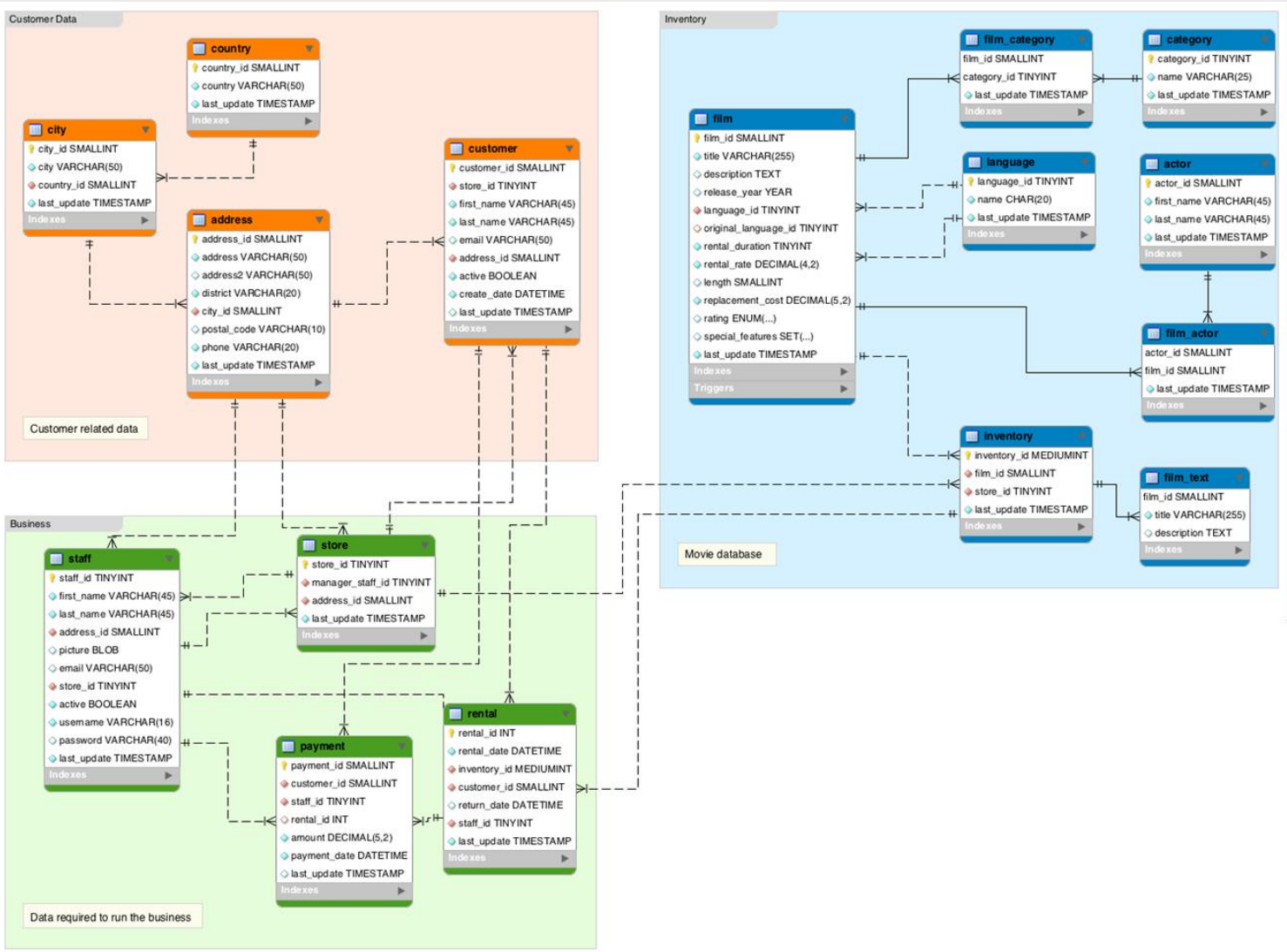
- Relational Data (Tables/Transaction/Legacy Data/Relational Databases (RDBMS))
- Text Data (Web)
- Semi-structured Data (XML)
- Graph Data
- Social Network, Semantic Web (RDF), …
- Streaming Data

RDF – Resource Description Framework/interconnected web data

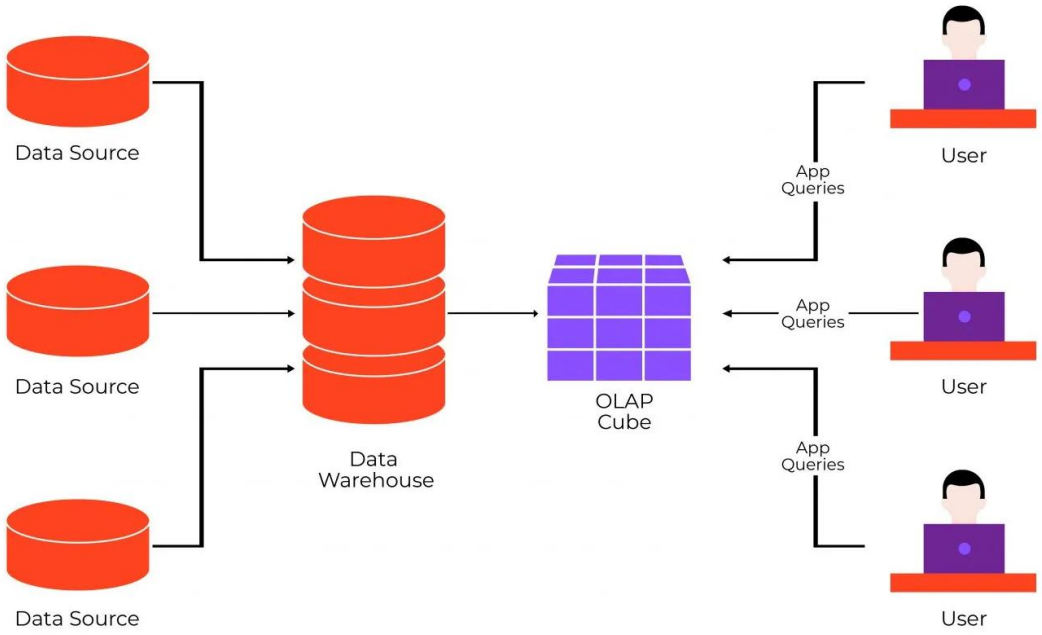
WHAT TO DO WITH THESE DATA?

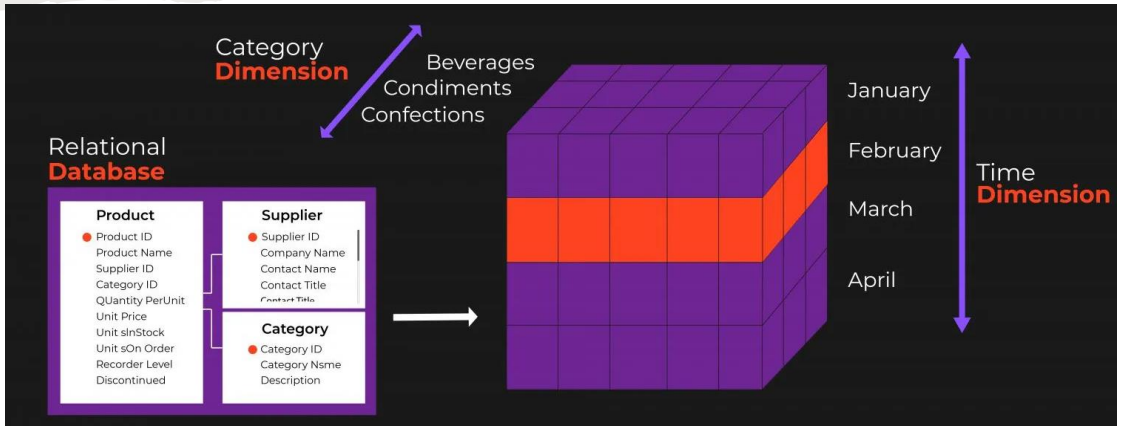
- Aggregation and Statistics
  - Data warehousing and OLAP
- Indexing, Searching, and Querying
  - Keyword based search
  - Pattern matching (XML/RDF)
- Knowledge discovery
  - Data Mining
  - Statistical Modeling

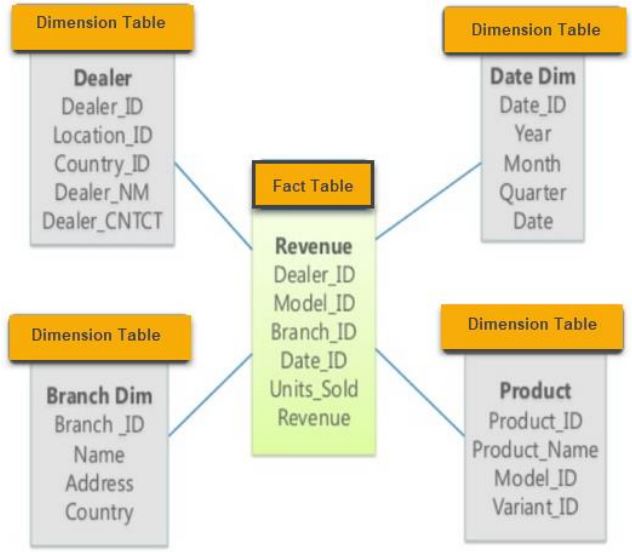
RDBMS


Customer Data
country
country_id SMALLINT
country VARCHAR(50)
last_update TIMESTAMP
Indexes
city
city_id SMALLINT
city VARCHAR(50)
country_id SMALLINT
last_update TIMESTAMP
Indexes
address
address_id SMALLINT
address VARCHAR(50)
address2 VARCHAR(50)
district VARCHAR(20)
city_id SMALLINT
postal_code VARCHAR(10)
phone VARCHAR(20)
last_update TIMESTAMP
Indexes
customer
customer_id SMALLINT
store_id TINYINT
first_name VARCHAR(45)
last_name VARCHAR(45)
email VARCHAR(50)
address_id SMALLINT
active BOOLEAN
create_date DATETIME
last_update TIMESTAMP
Indexes
Customer related data
Inventory
film_category
film_id SMALLINT
category_id TINYINT
last_update TIMESTAMP
Indexes
category
category_id TINYINT
name VARCHAR(25)
last_update TIMESTAMP
Indexes
film
film_id SMALLINT
title VARCHAR(255)
description TEXT
release_year YEAR
language_id TINYINT
original_language_id TINYINT
rental_duration TINYINT
rental_rate DECIMAL(4,2)
length SMALLINT
replacement_cost DECIMAL(5,2)
rating ENUM(...)
special_features SET(...)
last_update TIMESTAMP
Indexes
Triggers
language
language_id TINYINT
name CHAR(20)
last_update TIMESTAMP
Indexes
actor
actor_id SMALLINT
first_name VARCHAR(45)
last_name VARCHAR(45)
last_update TIMESTAMP
Indexes
film_actor
actor_id SMALLINT
film_id SMALLINT
last_update TIMESTAMP
Indexes
inventory
inventory_id MEDIUMINT
film_id SMALLINT
store_id TINYINT
last_update TIMESTAMP
Indexes
film_text
film_id SMALLINT
title VARCHAR(255)
description TEXT
Indexes
Movie database
Business
staff
staff_id TINYINT
first_name VARCHAR(45)
last_name VARCHAR(45)
address_id SMALLINT
picture BLOB
email VARCHAR(50)
store_id TINYINT
active BOOLEAN
username VARCHAR(16)
password VARCHAR(40)
last_update TIMESTAMP
Indexes
store
store_id TINYINT
manager_staff_id TINYINT
address_id SMALLINT
last_update TIMESTAMP
Indexes
rental
rental_id INT
rental_date DATETIME
inventory_id MEDIUMINT
customer_id SMALLINT
return_date DATETIME
staff_id TINYINT
last_update TIMESTAMP
Indexes
payment
payment_id SMALLINT
customer_id SMALLINT
staff_id TINYINT
rental_id INT
amount DECIMAL(5,2)
payment_date DATETIME
last_update TIMESTAMP
Indexes
Data required to run the business

OLAP


Data Source
Data Source
Data Source
Data Warehouse
OLAP Cube
App Queries
App Queries
App Queries
User
User
User


Category Dimension
Beverages
Condiments
Confections
Relational Database
Product
Product ID
Product Name
Supplier ID
Category ID
QUantity PerUnit
Unit Price
Unit sInStock
Unit sOn Order
Recorder Level
Discontinued
Supplier
Supplier ID
Company Name
Contact Name
Contact Title
Category
Category ID
Category Nsme
Description
January
February
March
April
Time Dimension


Dimension Table
Dealer
Dealer_ID
Location_ID
Country_ID
Dealer_NM
Dealer_CNTCT
Dimension Table
Date Dim
Date_ID
Year
Month
Quarter
Date
Fact Table
Revenue
Dealer_ID
Model_ID
Branch_ID
Date_ID
Units_Sold
Revenue
Dimension Table
Branch Dim
Branch _ID
Name
Address
Country
Dimension Table
Product
Product_ID
Product_Name
Model_ID
Variant_ID

DATA SCIENCE APPLICATIONS

- Finance: Predicted whether a consumer applying for a mortgage loan have a risk to default
  - Data: loan payment data
- Medical: Predicted whether a patient, hospitalized due to a heart attack, will have a second heart attack
  - Data: demographic, diet, clinical measurements
- Business/Economics: Predict the price of stock 6 months from now.
  - Data: company performance, economic data
- Recommenders: Identify films which you may like
  - Data: Netflix movie database
- Social Media: Product review and consumer satisfaction from ads
  - Data: Facebook/Twitter/LinkedIn consumer profiles
- Internet Of Things: Smart home/Real-time monitoring
  - Data: wireless sensor data
- Infrastructure: Automatic trouble shooting
  - Data: software log data
- Speech Recognition: Natural language processing
  - Data: word data & text mining

OUTLINE

- ~~Data Science—A Brief Overview~~
- ML and Deep learning, the main driver of Modern Data Science
- Frameworks (Python & SQL) (Python Data Structures)
- Environment Setup and Get Started
- COVID Data Scraping from WEB
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

WHAT IS LEARNING?

- Given
  - *a data set D,*
  - *a task T, and*
  - *a performance measure M,*

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as **compared to no learning.**

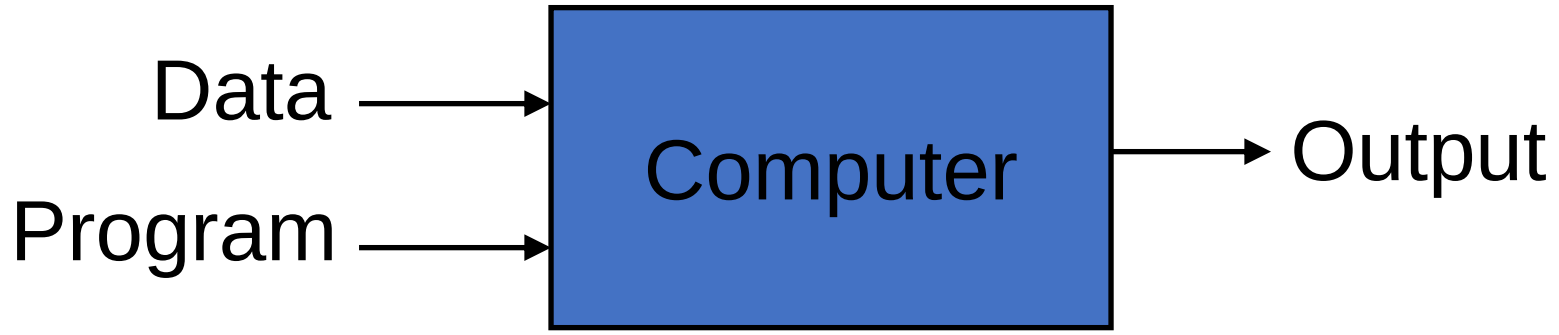
- Induction vs deduction
- Rote learning (memorization)
- Advice or instructional learning
- *Learning by example or practice*
  - *Most popular; many applications*
- Learning by analogy; transfer learning
- Discovery learning
- Others?

WHY MACHINE LEARNING?

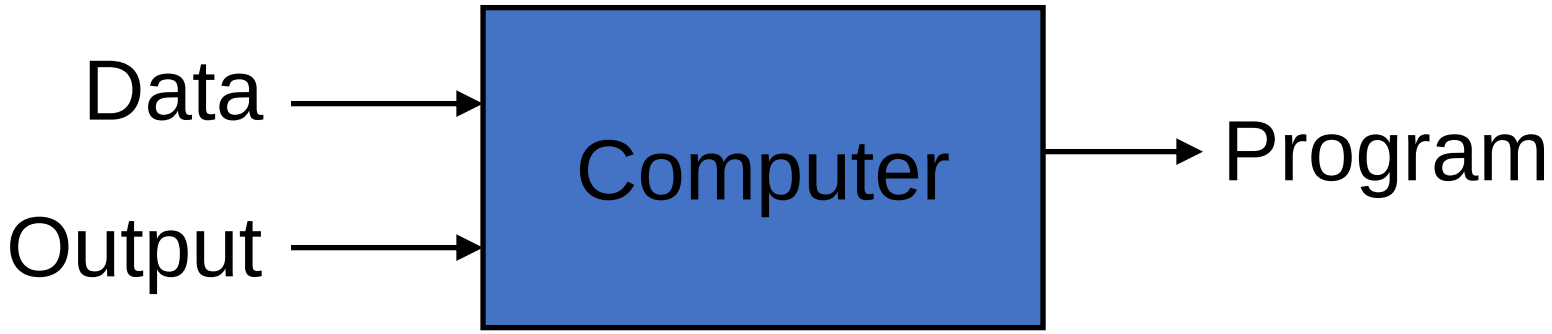
- Develop systems that are too difficult/expensive to construct manually because they require specific detailed skills or knowledge tuned to a specific task (***knowledge engineering bottleneck***).
- Develop systems that can automatically adapt and customize themselves to individual users
  - Personalized news or mail filter
  - Personalized tutoring
  - Personalized medicine
- Discover new knowledge from large databases (***data mining***)
  - Market basket analysis (e.g. diapers and cosmetics)
  - Medical text mining (e.g. migraines to calcium channel blockers to magnesium)
- Automate the automated systems new knowledge from large databases
  - Getting computers to program themselves The data should do the work

ROUTE TO LEARNING

Conventional Programming



Machine Learning

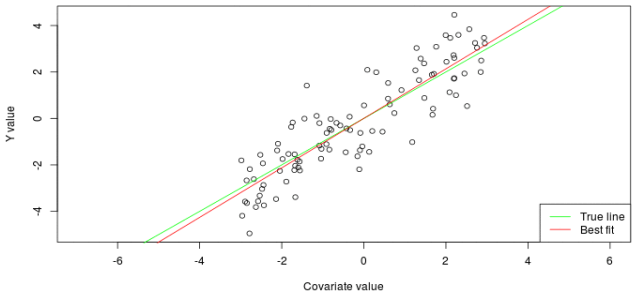


DATA TYPES & MODELS

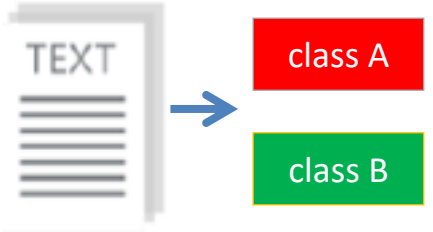
- Two basically different types of data
  - Quantitative (numerical): e.g., stock price
  - Categorical (discrete, often binary): diabetic/non-diabetic, spam/not-spam
- Data are predicted
  - on the basis of a set of features (e.g., diet or clinical measurements)
  - from a set of (observed) training data on these features
  - For a set of objects (e.g., people).
  - Inputs for the problems are also called predictors or independent variables
  - Outputs are also called responses or dependent variables
- The prediction model is called a learner or estimator.
  - Supervised learning: learn on outcomes for observed features
  - Unsupervised learning: no feature values available

LEARNING METHODOLOGIES

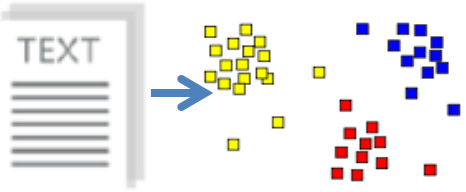
- Supervised learning
  - regression: predict numerical values
  - classification: predict categorical values, i.e., labels
- Unsupervised learning
  - clustering: group data according to "distance"
  - association: find frequent co-occurrences
  - link prediction: discover relationships in data
  - data reduction: project features to fewer features
- ~~Reinforcement learning (based on feedback/reward)~~



Regression

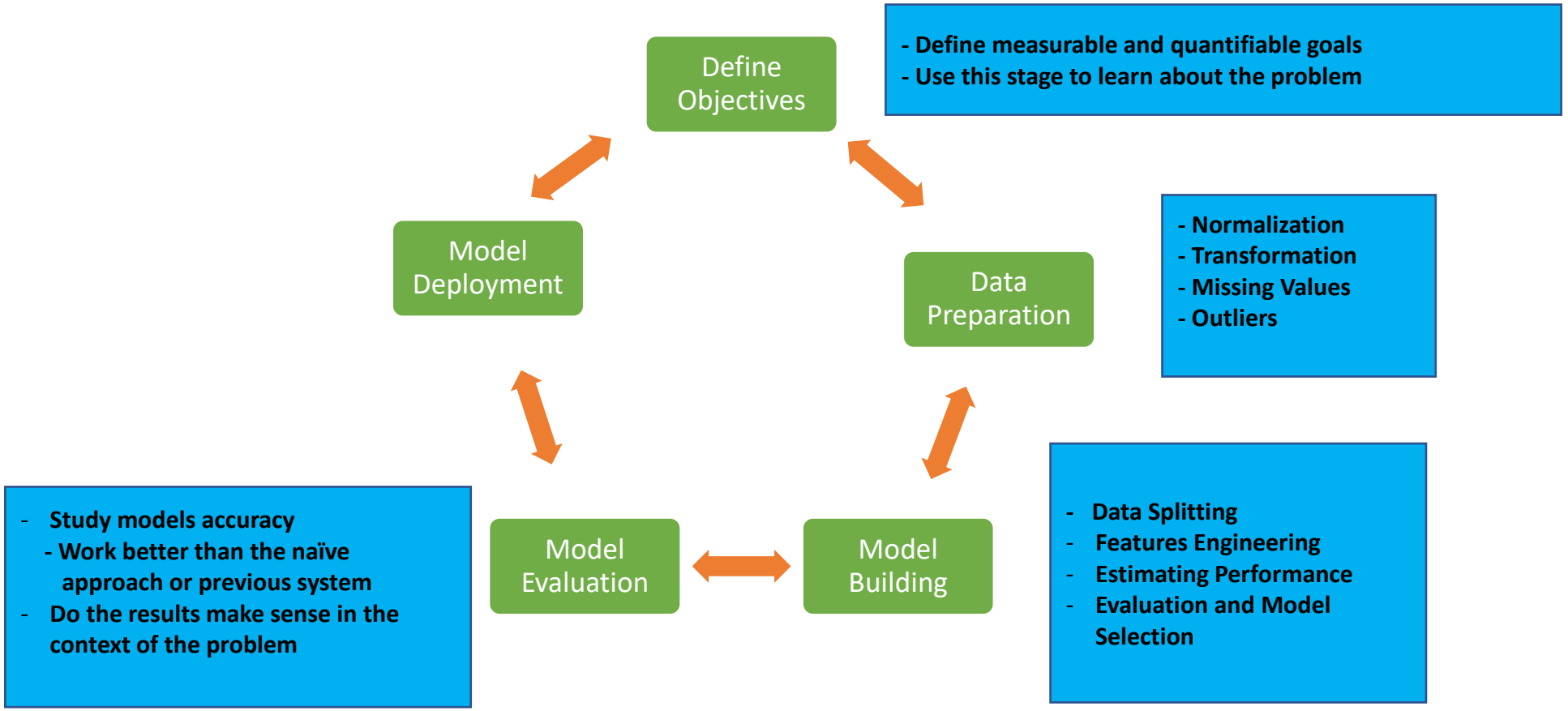


Classification

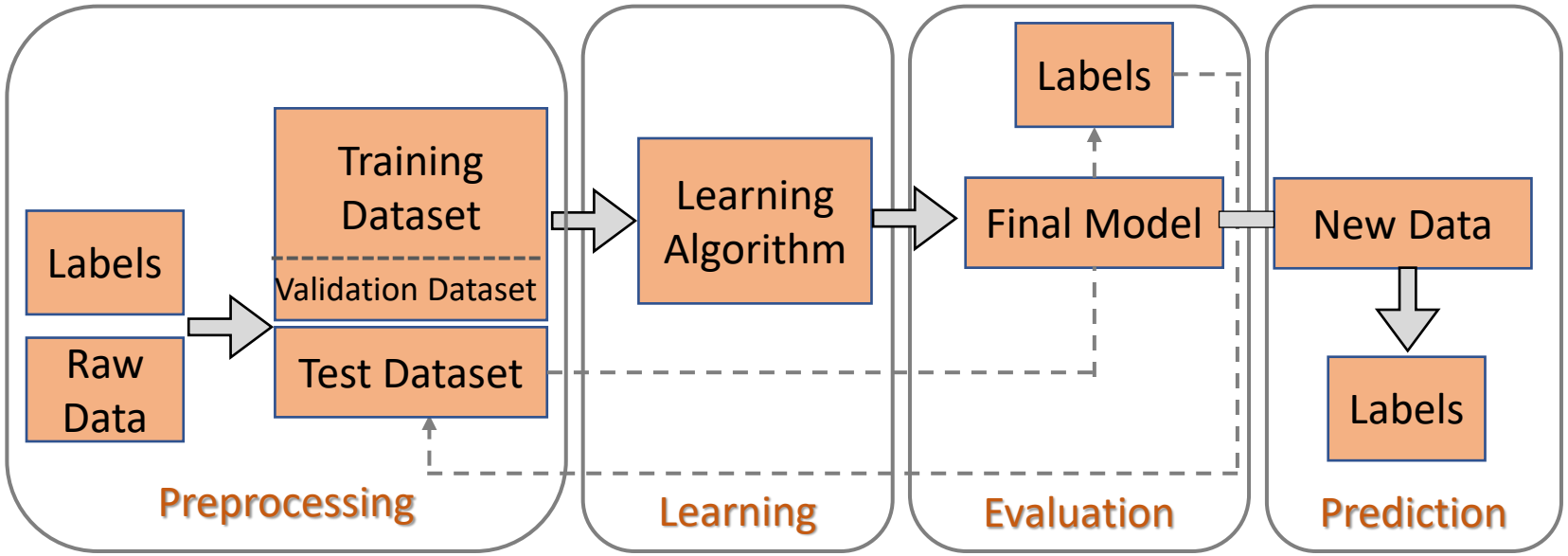


Clustering

MACHINE LEARNING PROCESS



MACHINE LEARNING FLOW



- Feature Extraction & Scaling
- Feature Selection
- Dimensionality Reduction
- Sampling

- Model Selection
- Cross-Validation
- Performance Metrics
- Hyperparameter Optimization

PERFORMANCE METRICS

Error Rate = (FP+FN)/ (TP+TN+FP+FN)

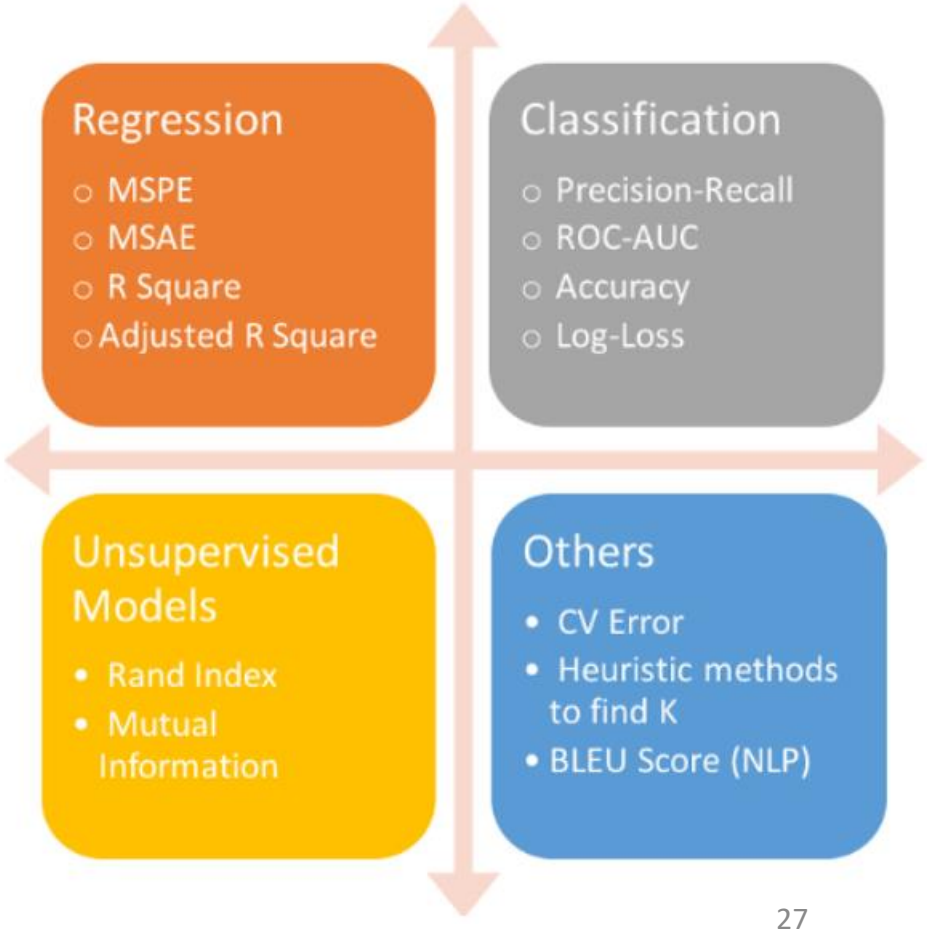
Predicted Class

Recall or True positive rate

Actual Class	**Positive**	**Negative**	
Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$ True negative rate
	Precision $\frac{TP}{(TP+FP)}$ Positive Predicted value	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

False positive rate = FP/ (FP+TN)

F-Score(Harmonic mean of precision and recall) = (1+b)(PREC.REC)/ (b^2PREC+REC) where b is commonly 0.5, 1, 2.



TRUE POSITIVE/FALSE POSITIVE

		Actual Value	
		Positive	Negative
Predicted Value	Positive	TP (True Positive)	FP (False Positive)
	Negative	FN (False Negative)	TN (True Negative)

- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

Accuracy:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall:

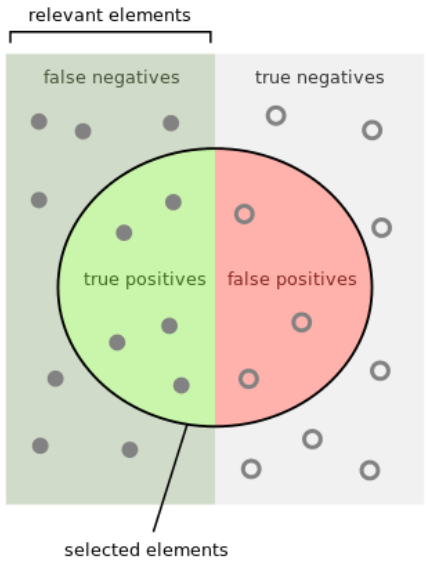
$$Recall = \frac{TP}{TP + FN}$$

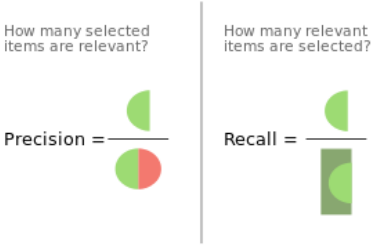
Precision:

$$Precision = \frac{TP}{TP + FP}$$

F_1 score:

$$F_1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$





PRECISION RECALL TRADE-OFF

CONFUSION MATRIX

Error Rate = (FP+FN)/ (TP+TN+FP+FN)

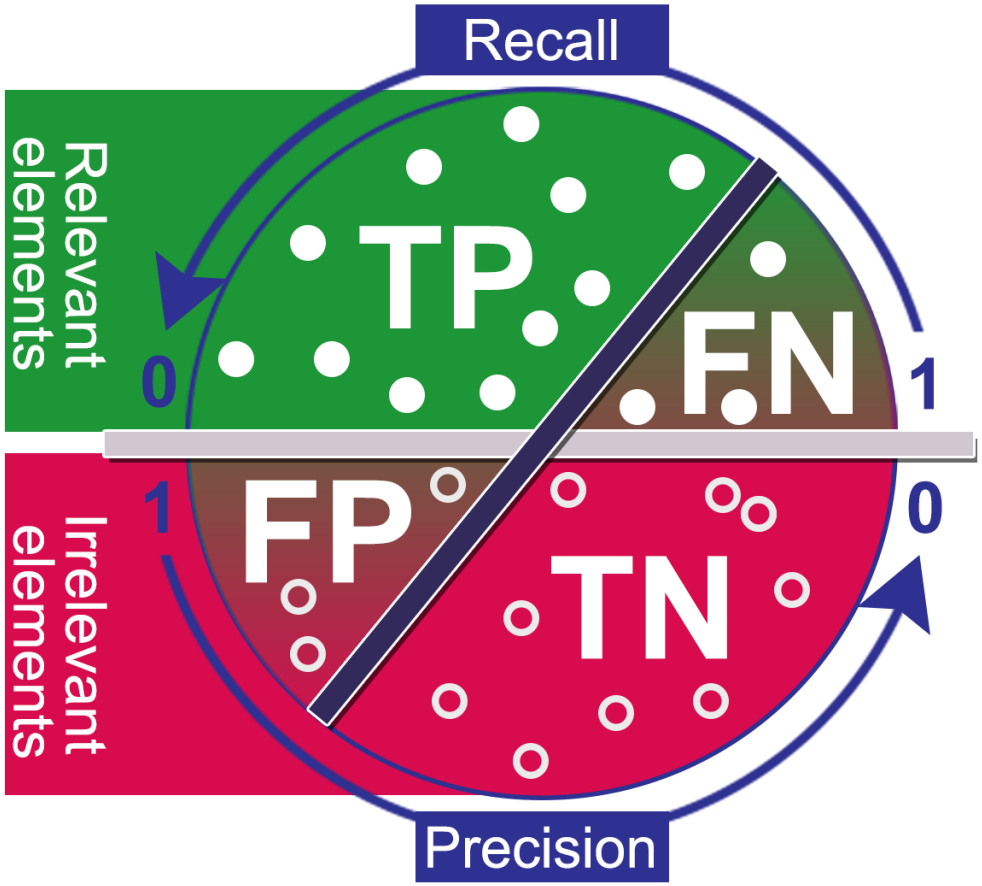
Predicted Class

Recall or True positive rate

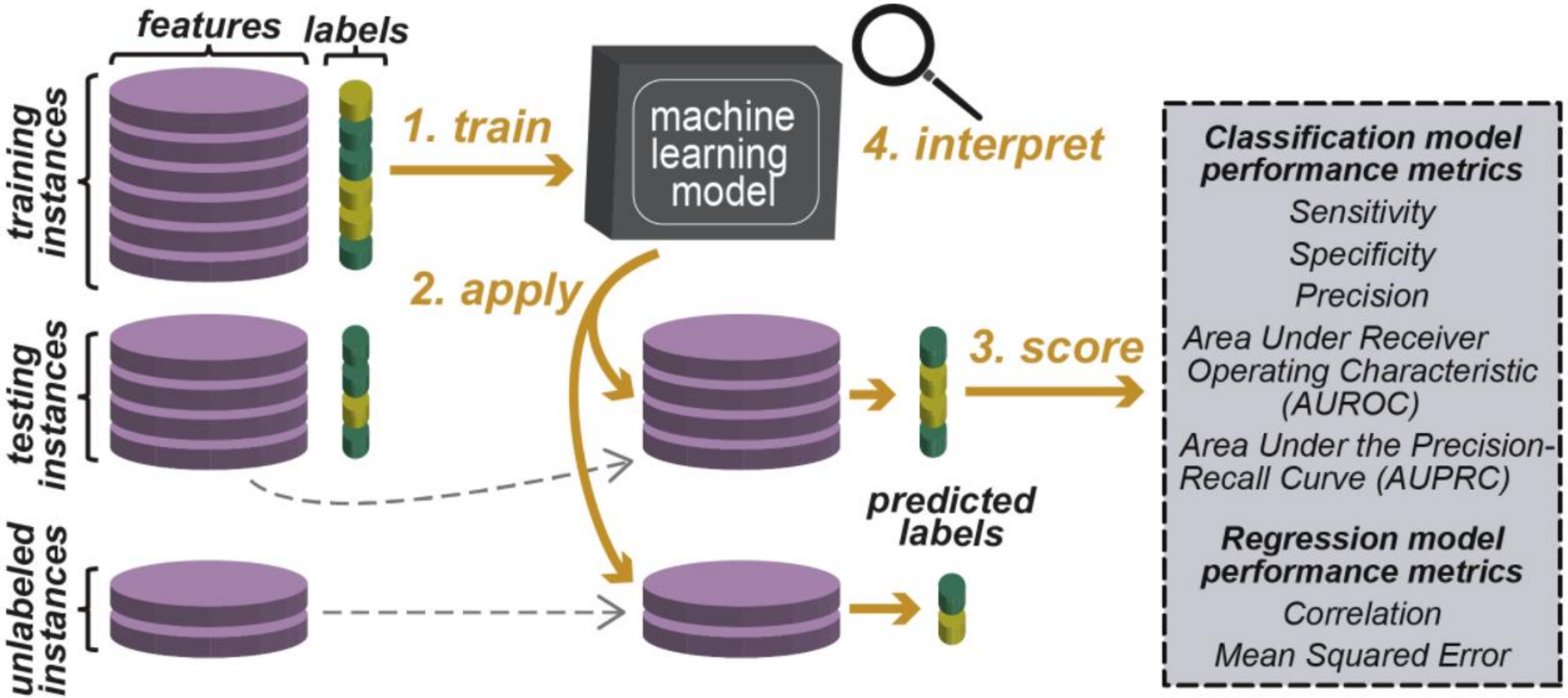
Actual Class	Positive	Negative	
Positive	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$ True negative rate
	Precision $\frac{TP}{(TP+FP)}$ Positive Predicted value	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

False positive rate = FP/ (FP+TN)

F-Score(Harmonic mean of precision and recall) = (1+b)(PREC.REC)/ (b²PREC+REC) where b is commonly 0.5, 1, 2.

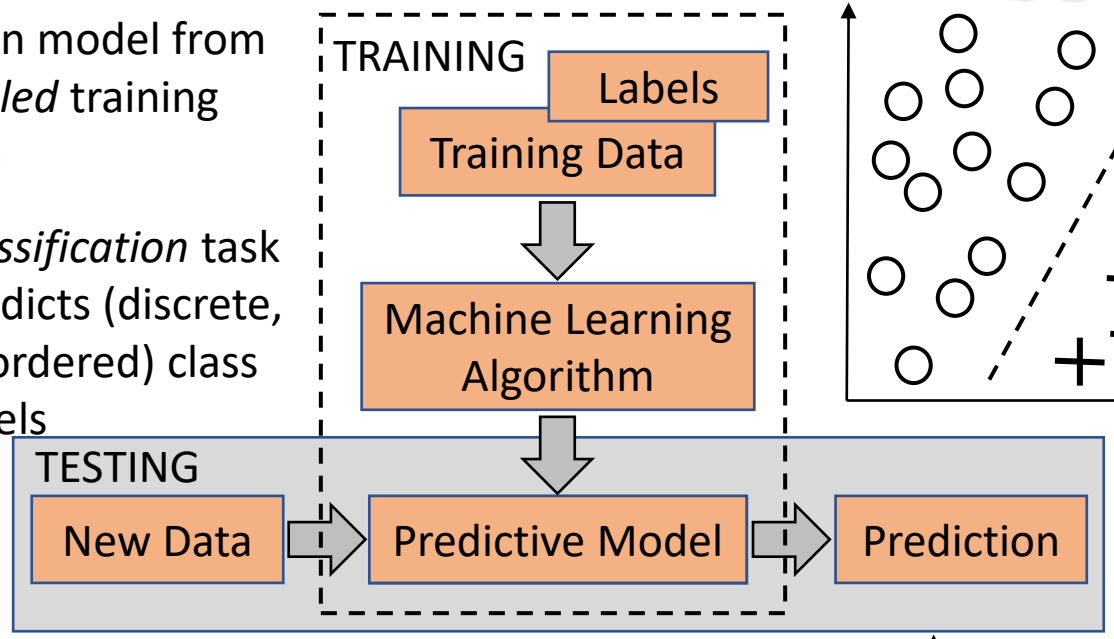


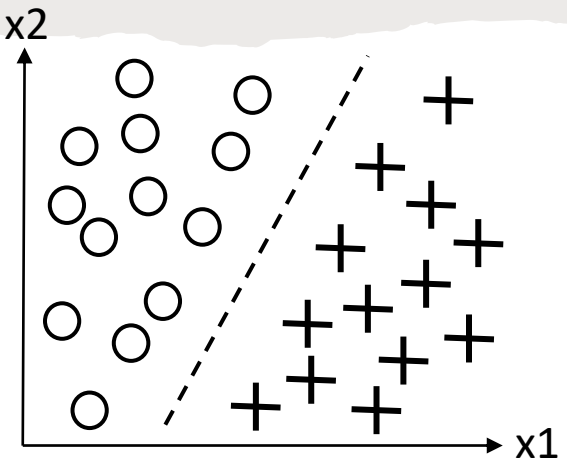
MACHINE LEARNING PIPELINE



SUPERVISED LEARNING

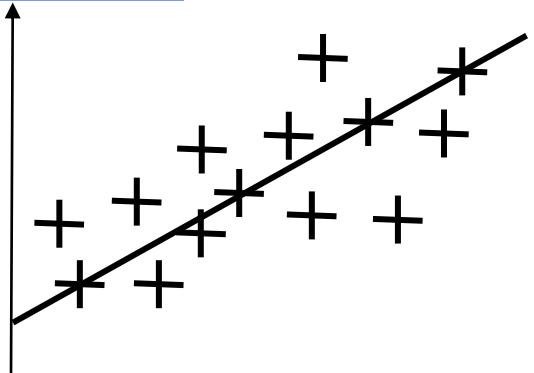
- Learn model from *labeled* training data
- *Classification* task predicts (discrete, unordered) class labels





decision boundary

- *Regression Analysis* predicts continuous outcome based on the relationship between *predictor* (explanatory) variables and a continuous response variable (outcome)



fit line that minimizes distance

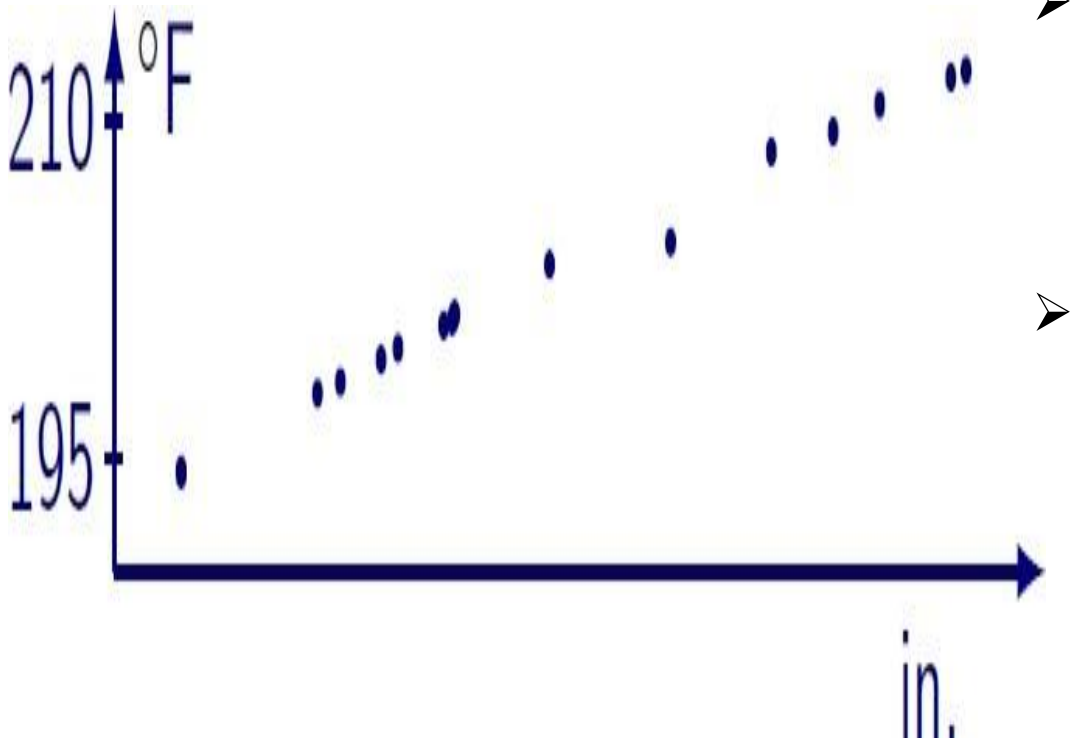
<u>Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).</u>

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

SUPERVISED MODEL FORMALISM

- Input data: Vector space X
- Output (label) : Vector space Y
- Training data (sub sample) with the correct class labels are provided
  - i.e. some correspondence between input (**X**) & output (**Y**) given
- Unknown function $f : X \rightarrow Y$
  - f may be probabilistic/deterministic
- Dataset $D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ with $\mathbf{x}_i \in X,\ y_i \in Y$
- Finite $Y \Rightarrow$ Classification Continuous $Y \Rightarrow$ Regression
  - Learning the model ≡ Fitting the parameters of model to minimise prediction error
  - Model can then be tested on test-data

REGRESSION – LINEAR MODEL



Measurements (possibly noisy) of liquid boiling point y as a function of barometric pressure x

- Set of N observations $(\mathbf{x}_i, y_i)$ with $i = 1, \ldots, N$ with $y_i \in \mathrm{R}$
  - **Does it make sense to use learning here?**
  - Choose a *model class* of functions
  - Design a criteria to guide the selection of one function from the selected class (*linear model here*)
- We want to fit a linear function to $(X, Y) = \{(\mathbf{x}_1, y_1) \ldots (\mathbf{x}_N, y_N)\}$
  - Fitting criteria: Least squares. Find the function that minimizes the sum of squared distances between actual y in the training set

General form: $f(\mathbf{x}; \theta) = \theta_0 + \theta_1 x_1 + \ldots \theta_d x_d$

1-D case: A line

$X \in \mathrm{R}^2$: a plane

Hyperplane in general d-D case

LOSS FUNCTION MINIMIZATION

- Targets are in Y
  - Binary Classification: $Y = \{-1, +1\}$
  - Univariate Regression: $Y \equiv \mathrm{R}$
- A **Loss Function** $L : Y \times Y \rightarrow \mathrm{R}$
- L maps decisions to costs. $L(\hat{y}, y)$ is the penalty for predicting $\hat{y}$ when the *correct* answer is y
- Standard choice for classification: 0/1 loss
- Standard choice for regression: $L(\hat{y}, y) = (\hat{y} - y)^2$
  - **Least Square Regression**

A parametric function $f(\mathbf{x}; \vartheta)$

Example: Linear function - $f(\mathbf{x}; \vartheta) = \vartheta + \vartheta_j x_{ij} = \vartheta^T \mathbf{x}$

The *empirical loss* of function $y = f(\mathbf{x}; \vartheta)$ on a set $\mathbf{X}$:

$$L(\theta, \mathbf{X}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} L(f(\mathbf{x}_i; \theta), y_i)$$

Least squares minimizes empirical loss for squared loss L

predicting labels for *new* examples

MAXIMUM LIKELIHOOD ESTIMATION

- Approximating f_θ by **maximizing the likelihood**
- Assume an independently drawn random sample y_i, $i = 1,\ldots,N$ from a probability density $\Pr_\theta(y)$. The log-probability of observing the sample is

$$L(\theta) = \sum_{i=1}^{N} \log \Pr_\theta(y_i)$$

- Set θ to maximize $L(\theta)$

- Least squares with the additive error model

$$Y = f_\theta(X) + \varepsilon$$

$$\varepsilon \sim N(0, \sigma^2)$$

- Equivalent to maximum likelihood with the likelihood function

$$\Pr(Y \mid X, \theta) \sim N(f_\theta(X), \sigma^2)$$

- This is, because in this case the log-likelihood function is

$$L(\theta) = -\frac{N}{2}\log(2\pi) - N\log\sigma$$

proportional to RSS

$$-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - f_\theta(x_i))^2$$

LEARNING VIA EMPIRICAL LOSS MINIMIZATION

- Learning is done in steps:
  - Select a restricted class H of *hypotheses* $f : X \rightarrow Y$

    Example: linear functions parameterized by ϑ: $f(\mathbf{x}, y) = \vartheta^T \mathbf{x}$

  - Select a hypothesis $f^* \in H$ based on the training set $D = (X, Y)$

    Example: minimize empirical squared loss. That is, select $f(\mathbf{x}, \vartheta^*)$ such that:

$$\vartheta = \arg\min \sum_{i=1}^{N} (y_i - \vartheta \mathbf{x})_i^2$$

Optimize (min. or max.) **objective/cost function** $f(\boldsymbol{\theta})$

Generate **error signal** that measures difference between predictions and target values

PROBABILISTIC MODEL

- N IID training samples: $\{x_n, c_n\}$
- Class label: $c_n \in \{0,1\}$
- Feature vector: $X \in R^d$

- Focus on modeling conditional probabilities $P(C|X)$
- Needs to be followed by a decision step

- Apply model for binary setting

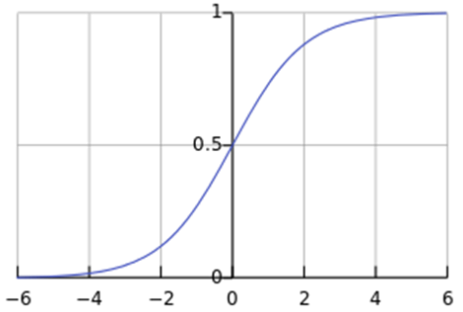
$$\mu(x) \equiv P(C = 1|x) = \frac{1}{1 + \exp\{-w^T x\}}$$

- Formulate likelihood with weights as parameters

$$L(w) = \prod_n \mu(x_n)^{c_n} \left(1 - \mu(x_n)\right)^{1-c_n}$$

$$l(w) = \sum_n c_n \log \mu + (1 - c_n) \log(1 - \mu)$$

where $\mu = \frac{1}{1+\exp\{-w^T x_n\}}$



LOGISTIC REGRESSION

Y = Binary response $= \beta_0 + \beta_1 X$ X = Quantitative predictor

Equivalent forms of the logistic regression model

Logit form

$$\log\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 X$$

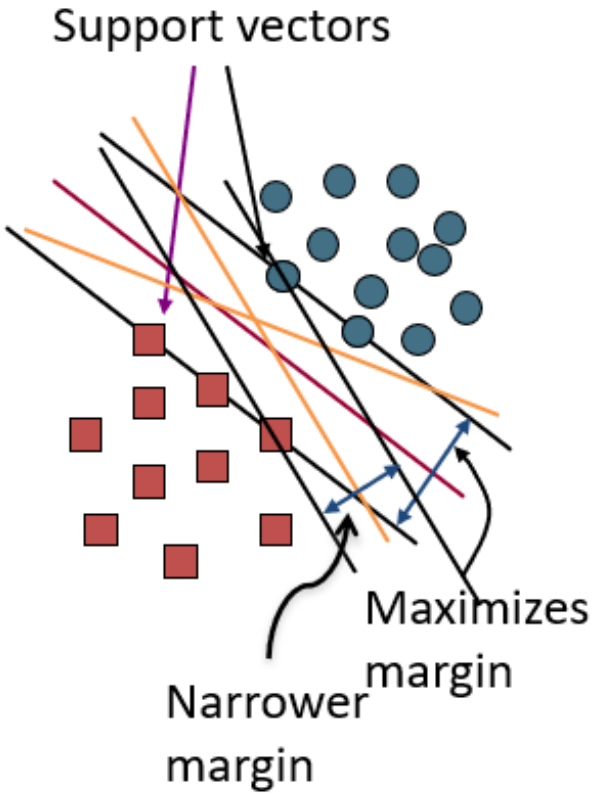
Probability form

$$p = \frac{e^{b_0 + b_1 X}}{1 + e^{b_0 + b_1 X}}$$

π = proportion of 1's (success) at any X

DECISION BOUNDARY HYPERPLANES

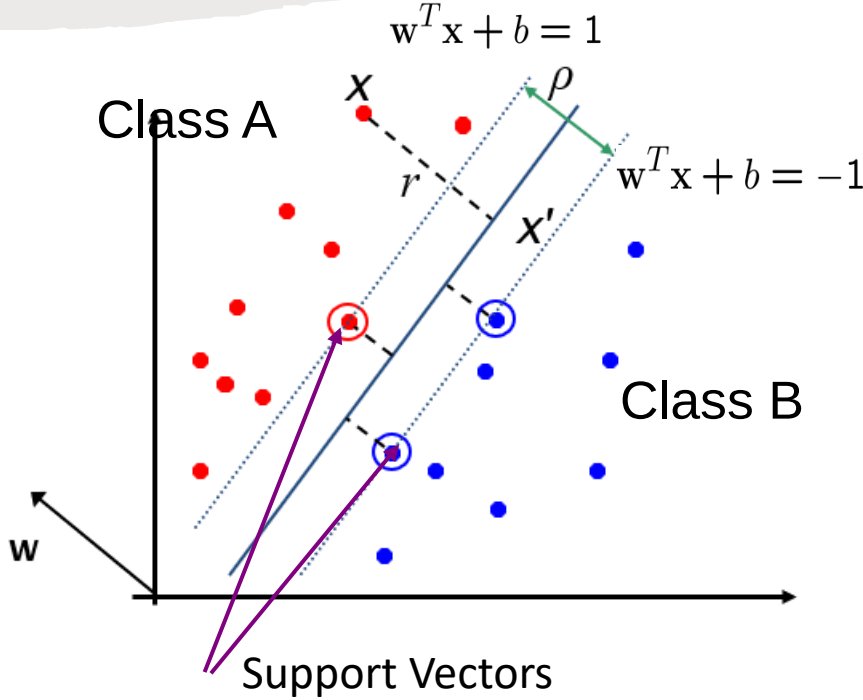
- SVMs maximize the *margin* around the separating hyperplane.
  - A.k.a. large margin classifiers
- The decision function is fully specified by a subset of training samples, *the support vectors*.
- Solving SVMs is a *quadratic programming* problem



SVM FORMALISM

- **w**: decision hyperplane normal vector
- $\mathbf{x}_i$: data point *i*
- y_i: class of data point *i* (+1 or -1)
- Classifier is: $f(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T\mathbf{x}_i + b)$
- Functional margin of $\mathbf{x}_i$ is: $y_i(\mathbf{w}^T\mathbf{x}_i + b)$
  - But note that we can increase this margin simply by scaling **w**, **b**….
- Functional margin of dataset is twice the minimum functional margin for any point $r = \frac{2}{\|\mathbf{w}\|}$

  - The factor of 2 comes from measuring the whole width of the margin



- Given a new point **x**, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^T\mathbf{x} + b = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$
    - Decide class based on whether < or > 0

SVM – MATHEMATICAL FORMULATION

Find $\mathbf{w}$ and b such that

$r = \frac{2}{\|\mathbf{w}\|}$ is maximized; and for all $\{(\mathbf{x_i}, y_i)\}$

$\mathbf{w^T x_i} + b \geq 1$ if $y_i = 1$; $\mathbf{w^T x_i} + b \leq -1$ if $y_i = -1$

⇩ min $||\mathbf{w}||$ = max $1/||\mathbf{w}||$

Find $\mathbf{w}$ and b such that

$\mathbf{\Phi(w)} = \frac{1}{2}\mathbf{w^T w}$ is minimized;

and for all $\{(\mathbf{x_i}, y_i)\}$: $y_i(\mathbf{w^T x_i} + b) \geq 1$

- This is now optimizing a *quadratic* function subject to *linear* constraints
- Quadratic optimization problems are a well-known class of mathematical programming problem, and many (intricate) algorithms exist for solving them
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

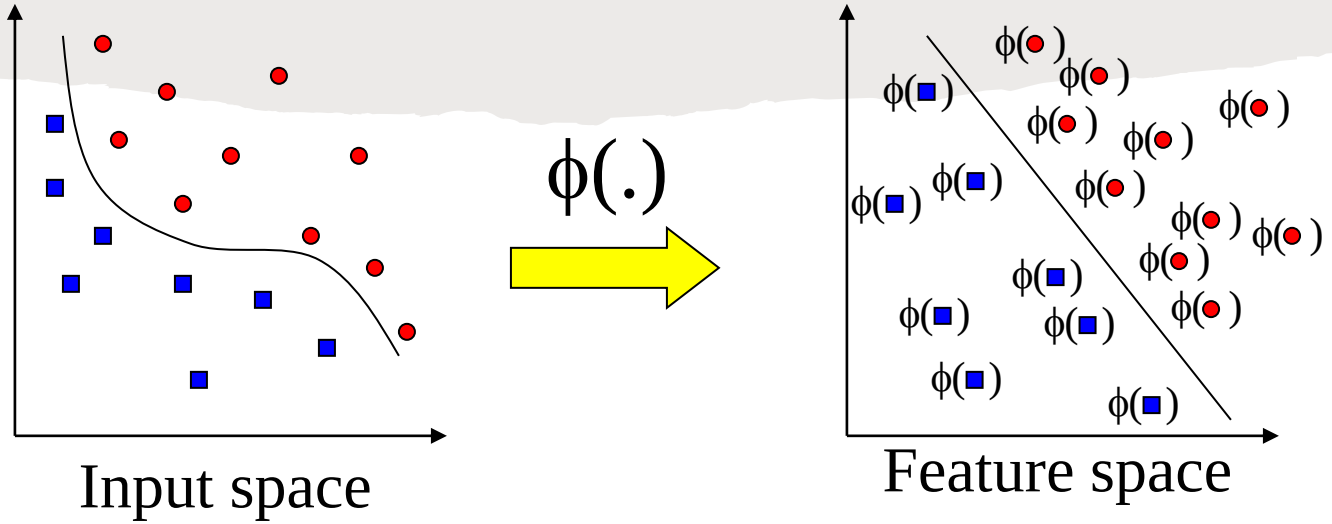
Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q(\alpha)} = \sum\alpha_i - \frac{1}{2}\sum\sum\alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j}$ is maximized and

(1) $\sum\alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

NON-LINEAR DECISION BOUNDARY

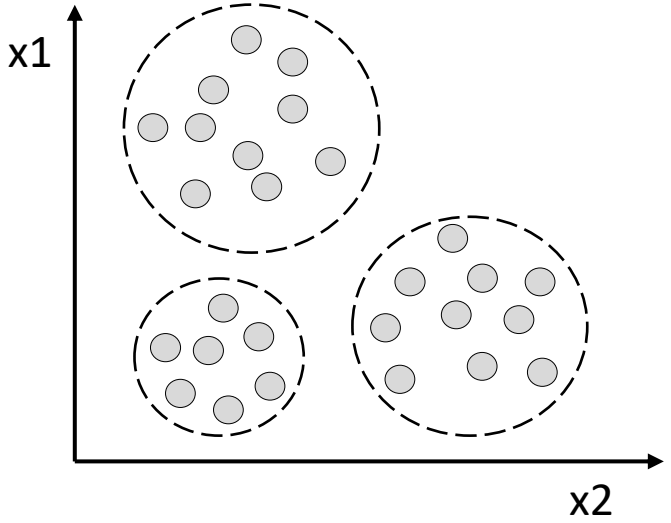


- Standard approach will be to transform the data to a higher dimensional space where the data can be separated by a linear surface.
- Define a kernel function:
- Examples of kernel functions include polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^t \mathbf{x}_j + 1)^d$

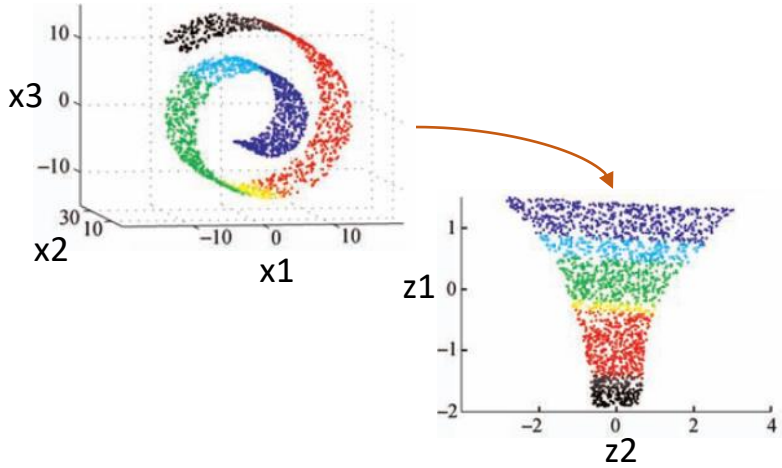
UNSUPERVISED MODEL

- Explore structure of data to extract meaningful information without the guidance of a known outcome variable or reward function.

- *Clustering* organizes data into meaningful subgroups (*clusters*) – "unsupervised classification"



- *Dimensionality Reduction* compresses data onto smaller dimensional subspace while retaining most of the relevant information



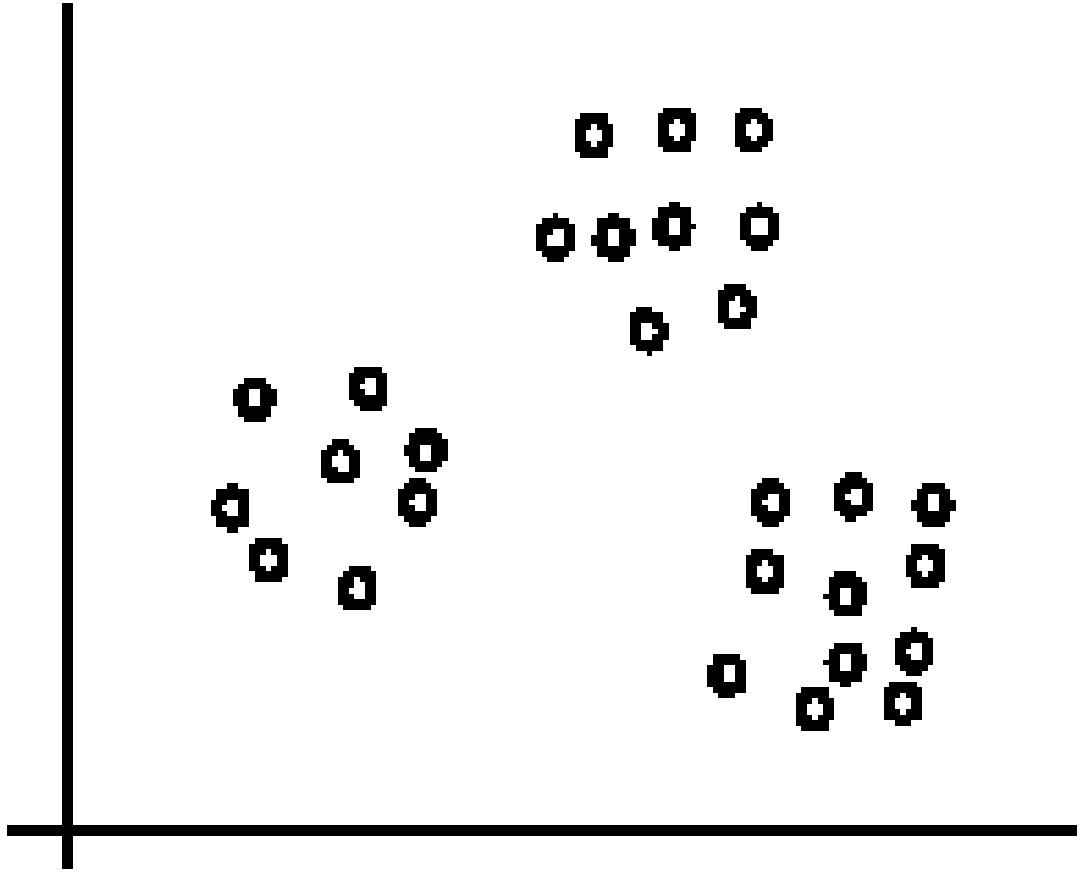
UNSUPERVISED MODEL - CLUSTERING

- Clustering is a technique for finding **similarity groups** in data, called **clusters**.
  - it groups data instances that are similar to (near) each other in one cluster and data instances that are very different (far away) from each other into different clusters.

- Clustering is often called an **unsupervised learning** task as no class values denoting an *a priori* grouping of the data instances are given, which is the case in supervised learning.

UNSUPERVISED MODEL FORMALISM

K-means algorithm
Representation of clusters
Hierarchical clustering
Distance functions



- Similarity or Distance Measure: Alternative Choices

  - Cosine similarity

$$cos(\vec{x}, \vec{y}) = \frac{\sum_i x_i y_i}{|\vec{x}||\vec{y}|}$$

  - Euclidean distance

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_i (x_i - y_i)}$$

  - Kernel functions, e.g.,

$$K(d(\vec{x}, \vec{y})) = e^{-d(\vec{x}, \vec{y})^2 / 2h^2}$$

K-MEANS CLUSTERING

- K-means is a partitional clustering algorithm
- Let the set of data points (or instances) *D* be

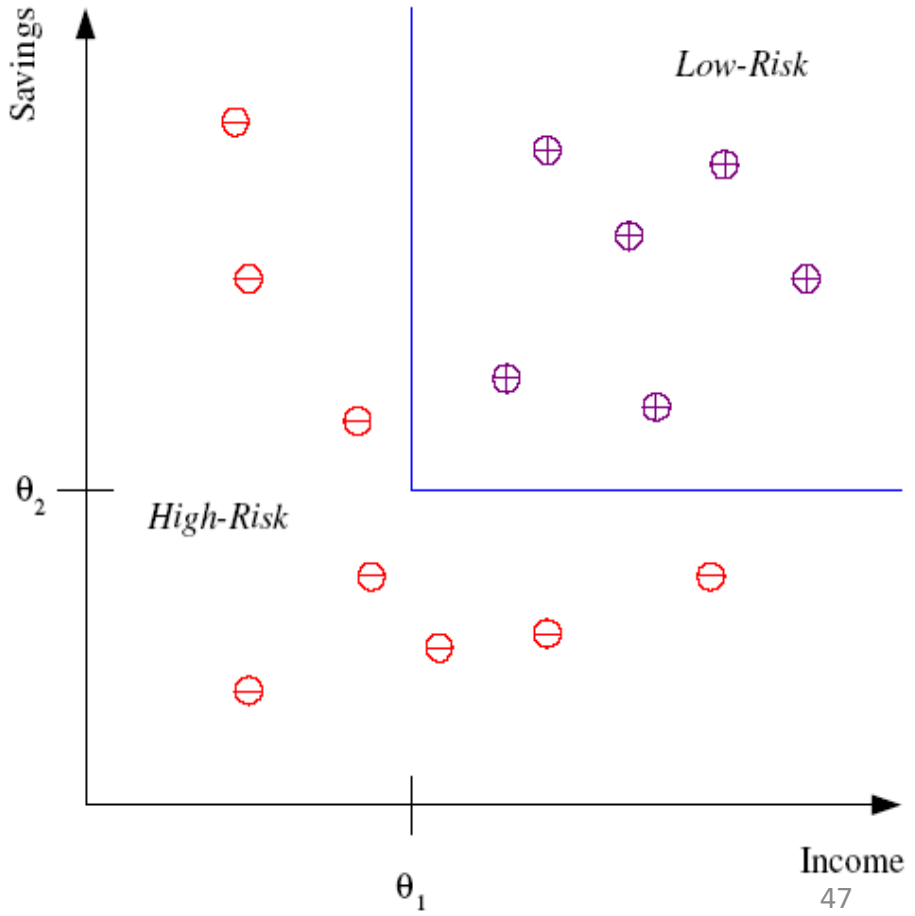
  $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$,

  where $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ir})$ is a vector in a real-valued space $X \subseteq R^r$, and *r* is the number of attributes (dimensions) in the data.
- The *k*-means algorithm partitions the given data into *k* clusters.
  - Each cluster has a cluster **center**, called **centroid**.
  - *k* is specified by the user

- **Given *k*, the *k-means* algorithm works as follows:**
  1) *Randomly choose k data points (seeds) to be the initial centroids, cluster centers*
  2) *Assign each data point to the closest centroid*
  3) *Re-compute the centroids using the current cluster memberships.*
  4) *If a convergence criterion is not met, go to 2).*

CLASSIFICATION – A COMPARITIVE CASE STUDY

- Example
  - Credit Scoring
- Goal
  - Differentiating between high-risk and low-risk customers based on their income and savings
- Discriminant
  - IF **income > θ_1** AND **savings > θ_2** THEN **low-risk** ELSE **high-risk**
- Discriminant is called '**hypothesis**'
- Input attribute space is called '**Feature Space**'
- Here Input data is 2-dimensional and the output is binary



COMPARING ALGORITHMS

- CASE

Data on two inputs X_1 and X_2

Output variable has values GREEN (coded 0) and RED (coded 1)

100 points per class

$$x^T \hat{\beta} = 0.5$$

- LINEAR CLASSIFIER

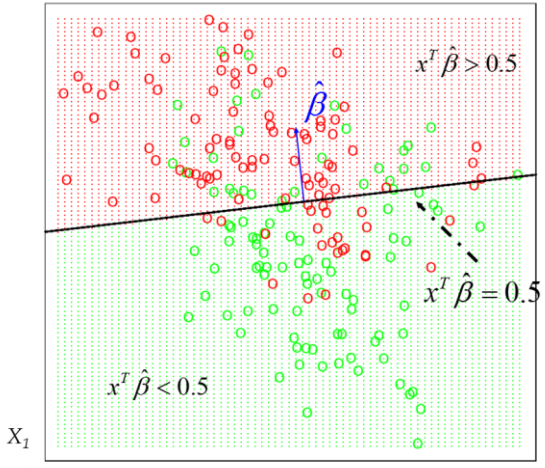
Regression Line defined by:

- NEAREST NEIGBOUR CLASSIFIER

Use the observation in training set closest to given input: $\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$

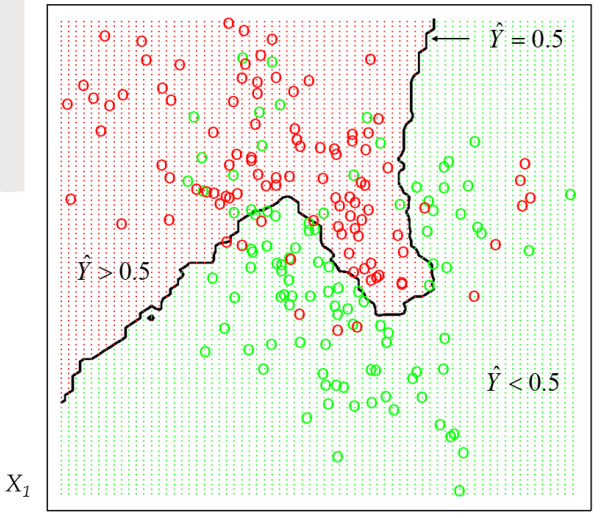
$N_k(x)$ is the set of the k closest points to x is the training sample

Average the outcome of the k closest training sample points



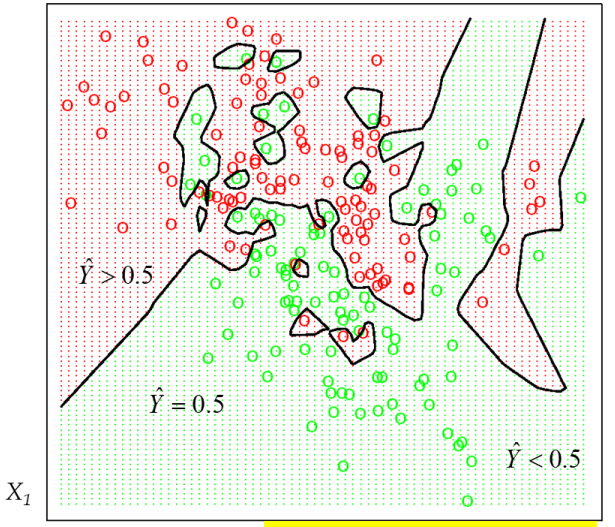
Multiple misclassifications

15-nearest neighbor averaging



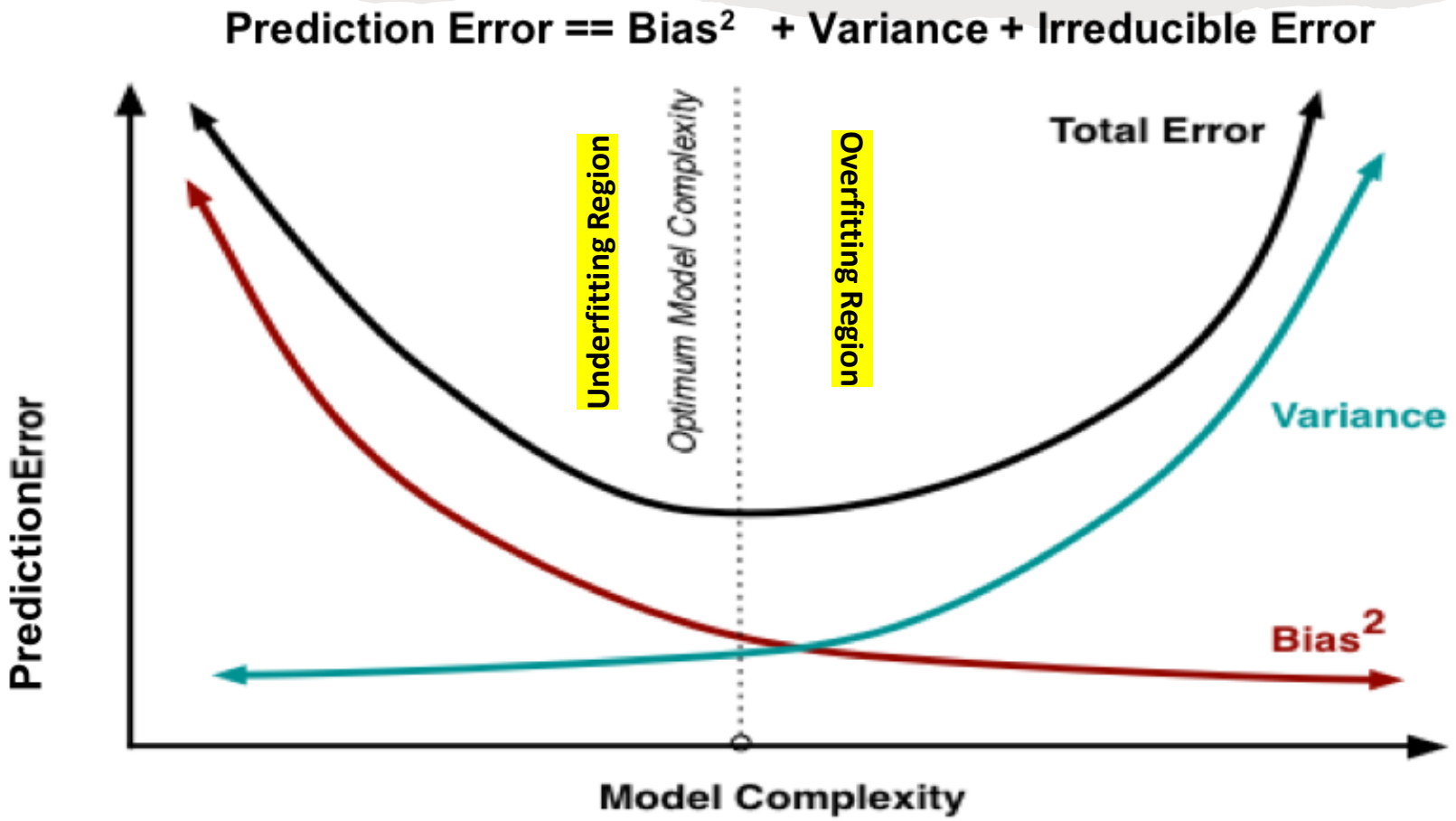
Fewer misclassifications

1-nearest neighbor averaging



No misclassifications: Overtraining

BIAS-VARIANCE TRADE OFF



- Model Complexity

  - **# of Features**
  - **Linear/Non-Linear**
  - Algorithmic
  - Computational

- Adjusting Methodologies

  - Regularization
  - Subset Selection
  - Dimensional reduction

MACHINE LEARNING ALGORITHMS

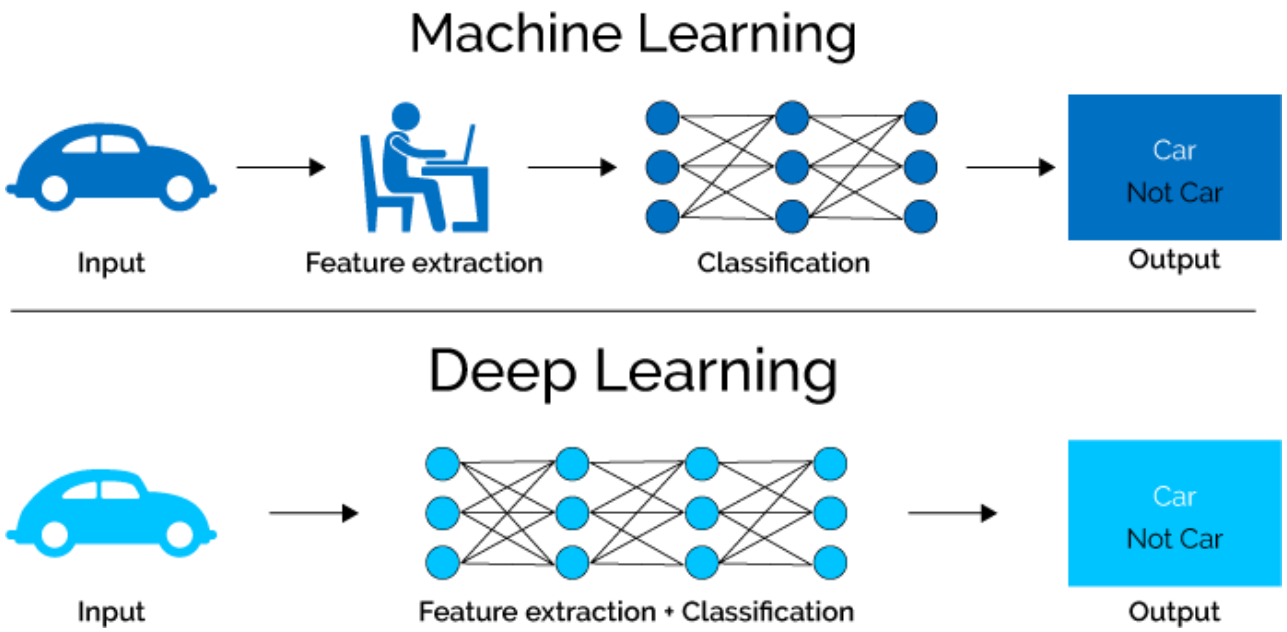
- Regression
  Ridge/Lasso regression, Support Vector Machines, Random Forest, Multilayer Neural Networks, Deep Neural Networks, ...

- Classification
  Naive Base,K-Nearest Neighbor, Support Ve Machines, Random Forest, Multilayer Neura Networks, Deep Neural Networks, ...

- Clustering
  k-Means, Hierarchical Clustering, ...

DEEP LEARNING

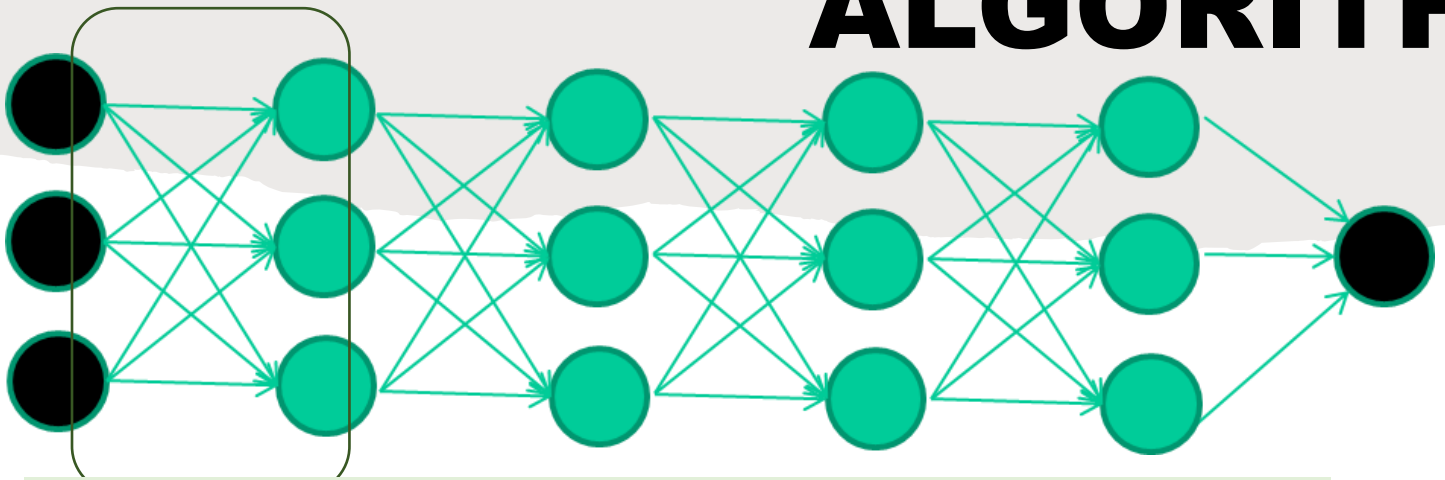
- Deep Learning is subfield of machine learning for learning **representations** of data. Exceptional effective at **learning patterns**.
- Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
- If you provide the system **tons of information**, it begins to understand it and respond in useful ways.

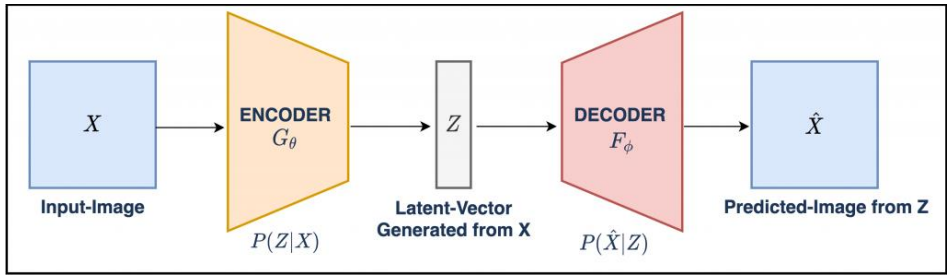


DEEP LEARNING – WHATS NEW?

- Deep Learning means using a ***neural network*** *(NN)* with several layers of nodes between input and output
- The series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.
- NN is an established field (e.g. Hopfield Model) good at learning the weights for networks with 1 hidden layer
- Issue with NN is multiple hidden layers -> ??? NEW ALGORITHMS

DEEP LEARNING – ADAPTIVE ALGORITHMS





Auto-encoder

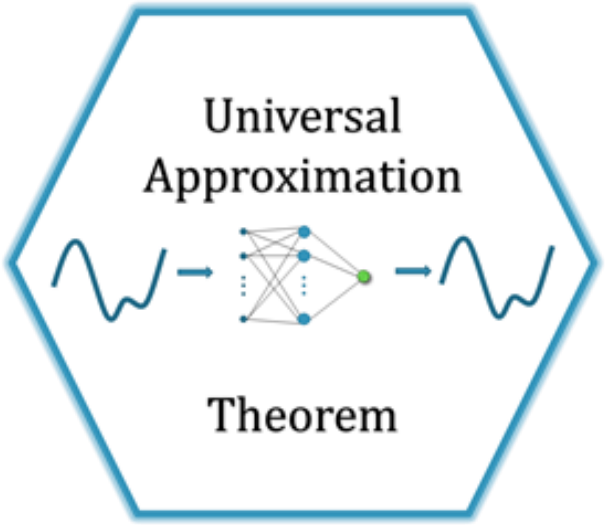
EACH of the (non-output) layers is trained to be an ***auto-encoder***

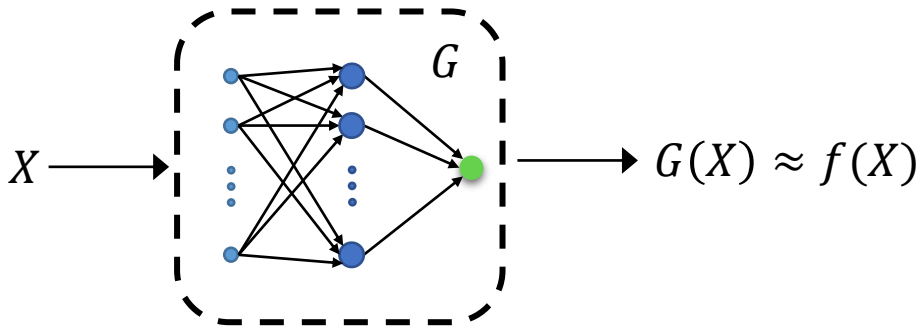
Mimicking brain in a way of learning features from previous experience (layers) and making new connections

DEEP LEARNING – MATHEMATICAL FOUNDATION?

A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units

Hornik, 1991



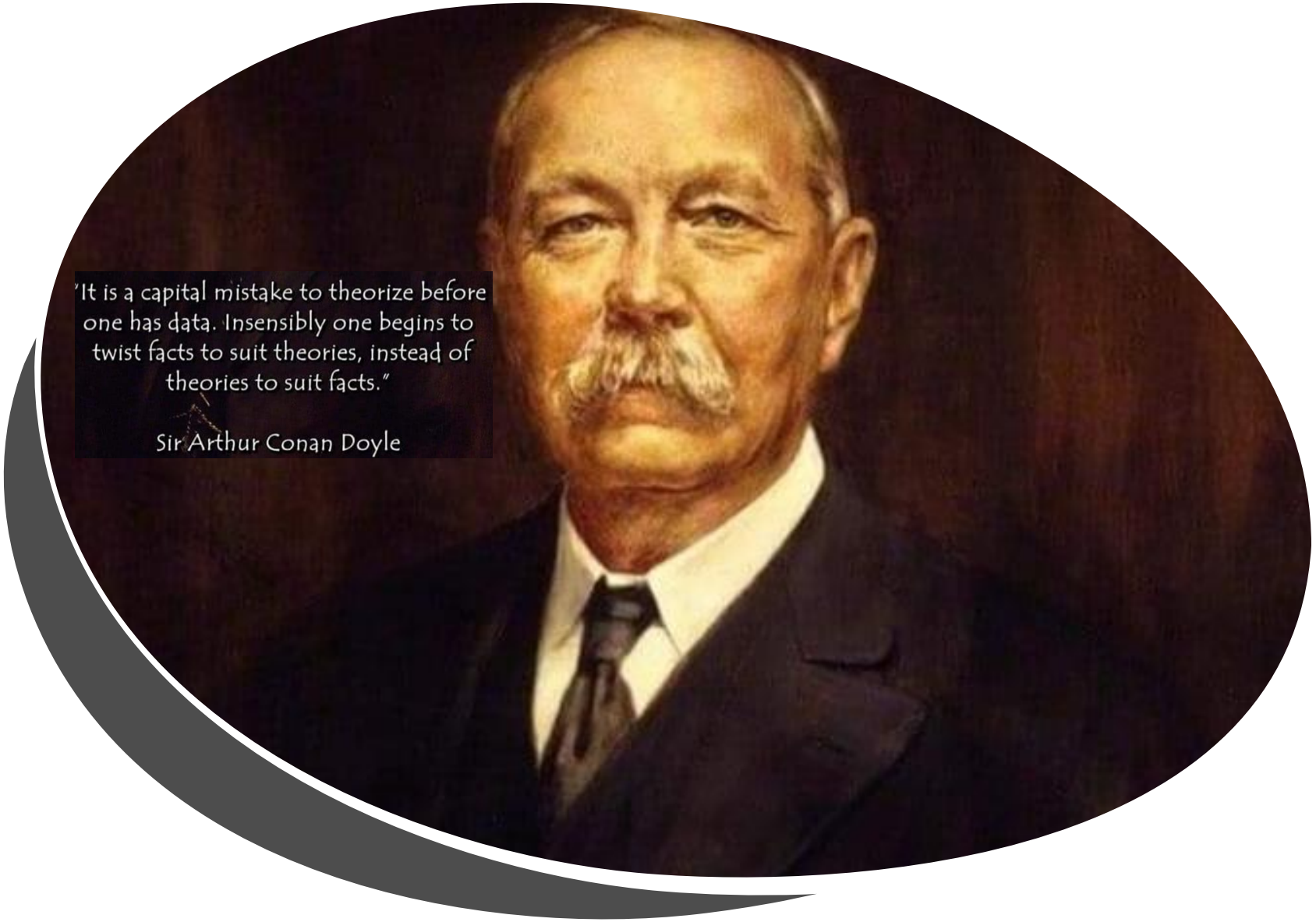


Theorem.

A feedforward single hidden layer network with finite width (multiple layers) can approximate continuous functions on compact subsets of $\mathbb{R}^n$ under mild assumptions on the activation function.

DEEP LEARNING – WHY USEFUL?

- Manually designed features are often over-specified, incomplete and take a long time to design and validate
- Learned Features are easy to adapt, fast to learn
- Deep learning provides a very flexible, (almost?) universal, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Utilize large amounts of training data
- DL outperforms other ML techniques first in speech, vision and NLP

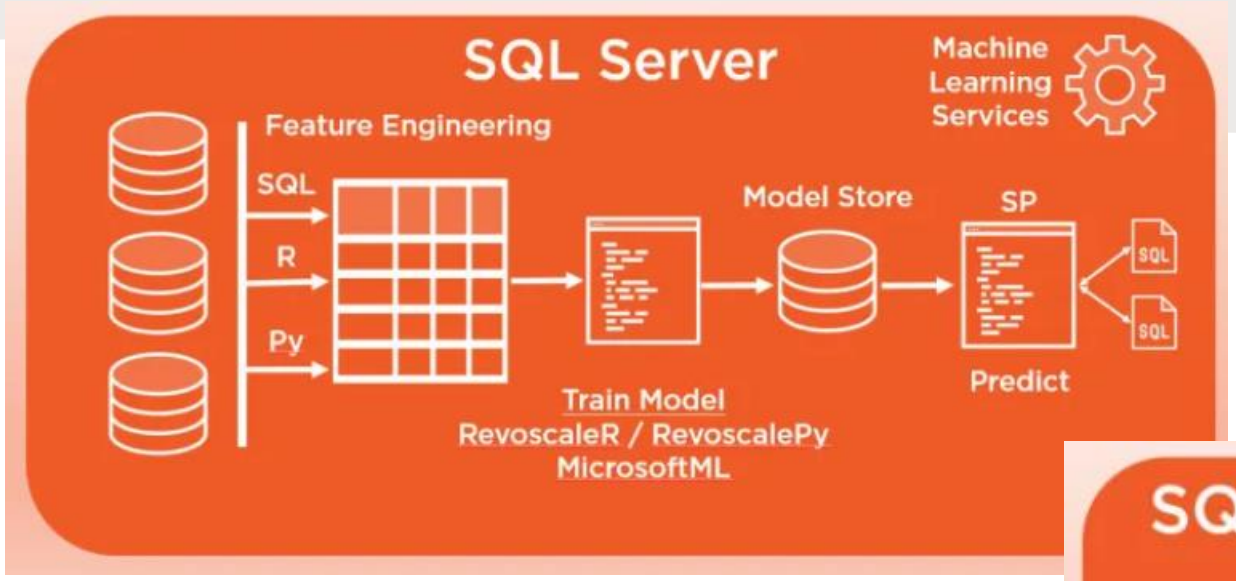

"It is a capital mistake to theorize before one has data. Insensibly one begins to twist facts to suit theories, instead of theories to suit facts."
Sir Arthur Conan Doyle

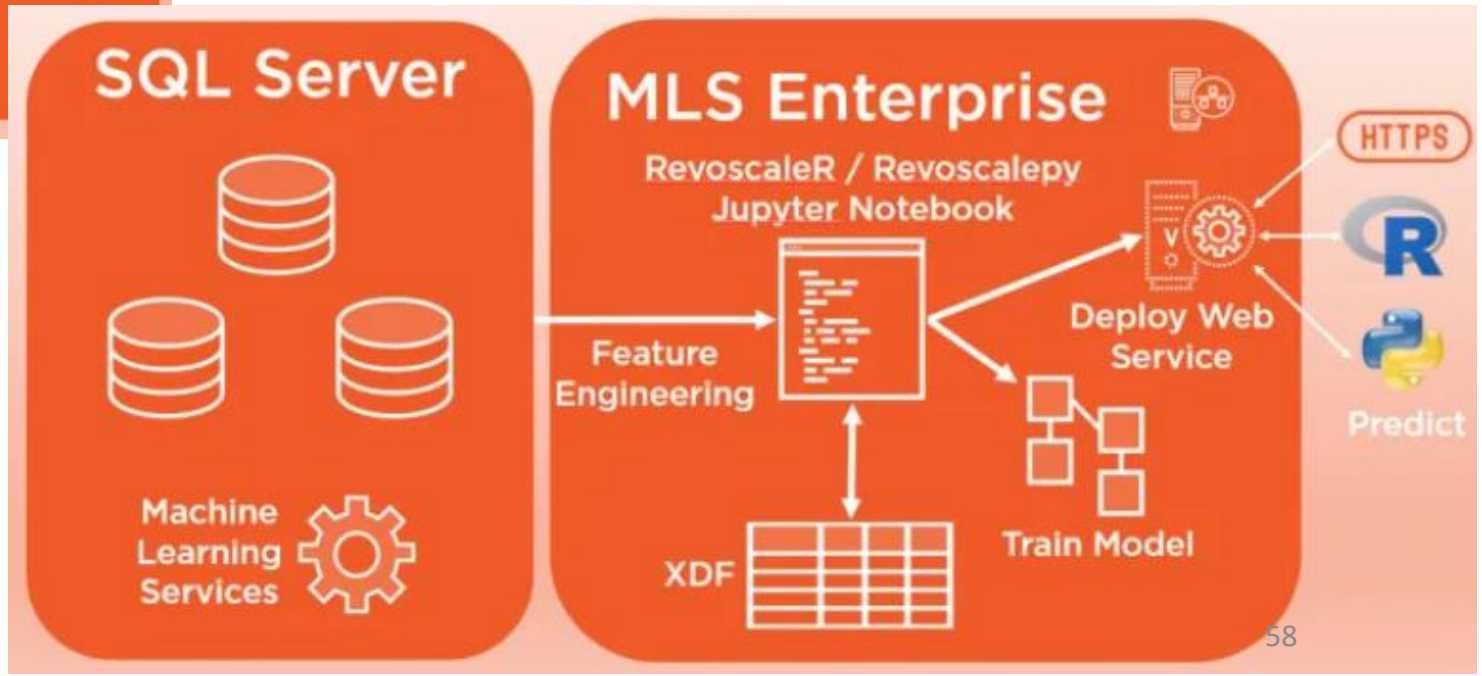
OUTLINE

- ~~Data Science—A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- Frameworks (Python & SQL) (Python Data Structures)
- Environment Setup and Get Started
- COVID Data Scraping from WEB
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

DATA PROCESSING PIPELINES

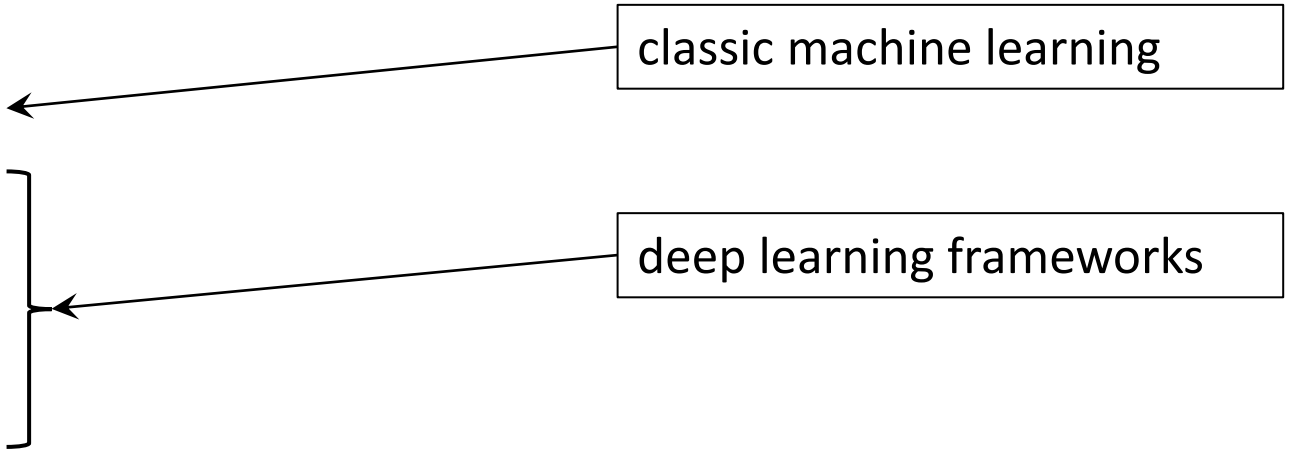




FRAMEWORKS

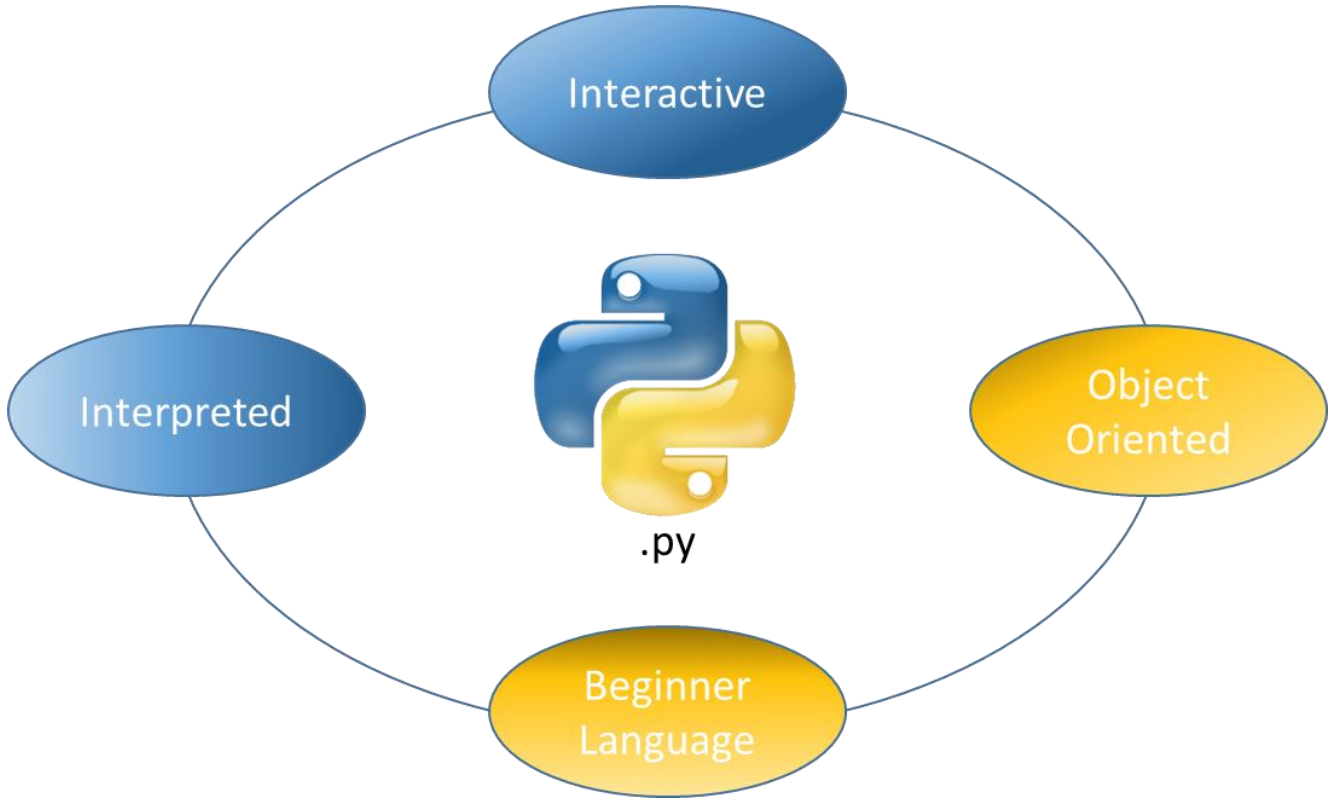
- Programming languages
  - Python
  - R
  - C++
  - Julia
  - ….
- Many libraries
  - scikit-learn
  - PyTorch
  - TensorFlow
  - Keras
  - Caffe
  - ……

Fast-evolving ecosystem!

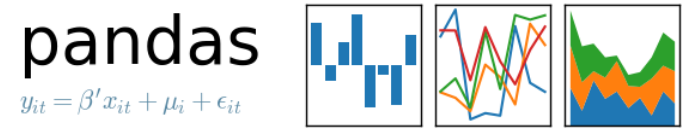


PYTHON

- An object-oriented interpreter-based programming language
- Basic essentials
  - Data types are numbers, strings, lists,tuples and dictionaries (hash-tables)
  - For loops and conditionals
  - Functions
  - Lists and hash-tables are references (like pointers in C)
  - All variables are passed by value

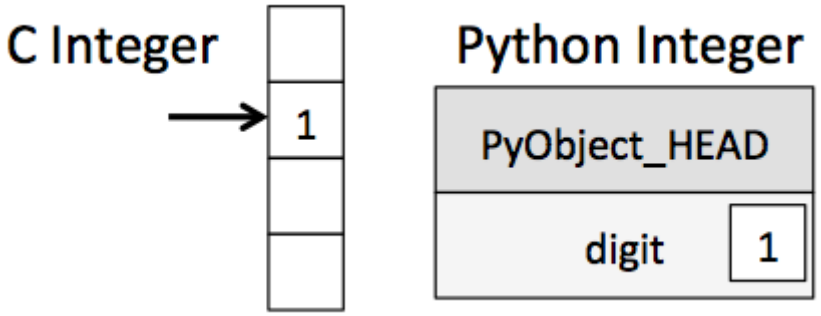


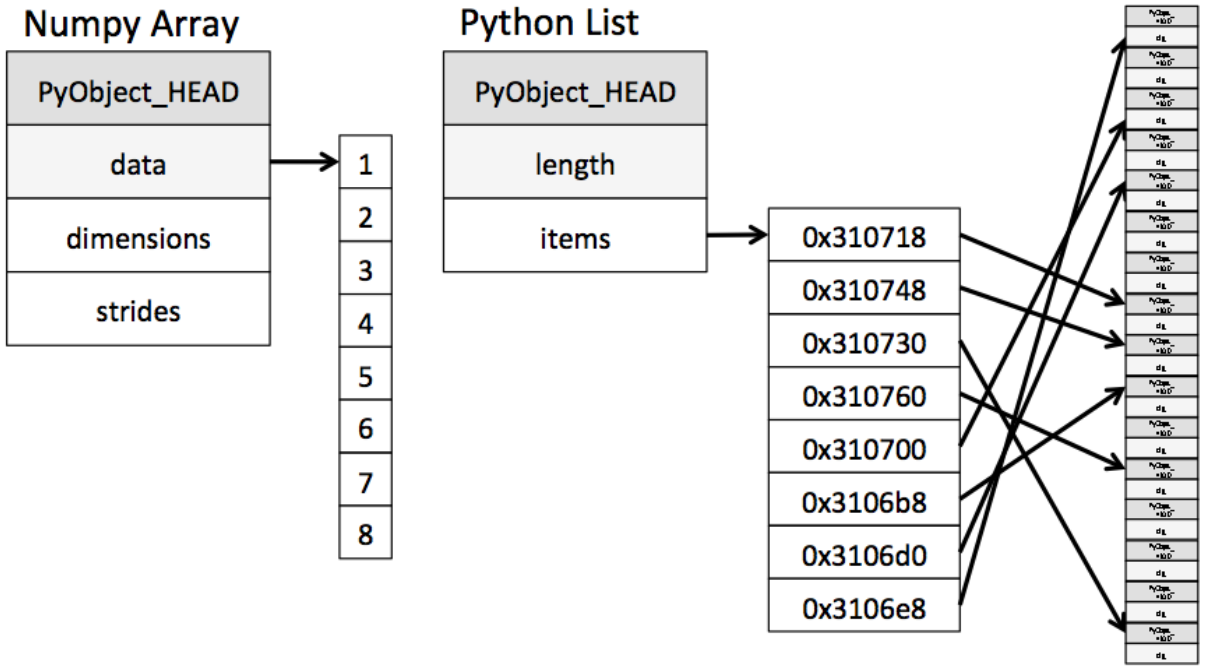
DATA PIPELINES



- Data ingestion
  - CSV/JSON/XML/H5 files, RDBMS, NoSQL, HTTP,...
- Data cleaning
  - outliers/invalid values? $\rightarrow$ filter
  - missing values? $\rightarrow$ impute
- Data transformation
  - scaling/normalization

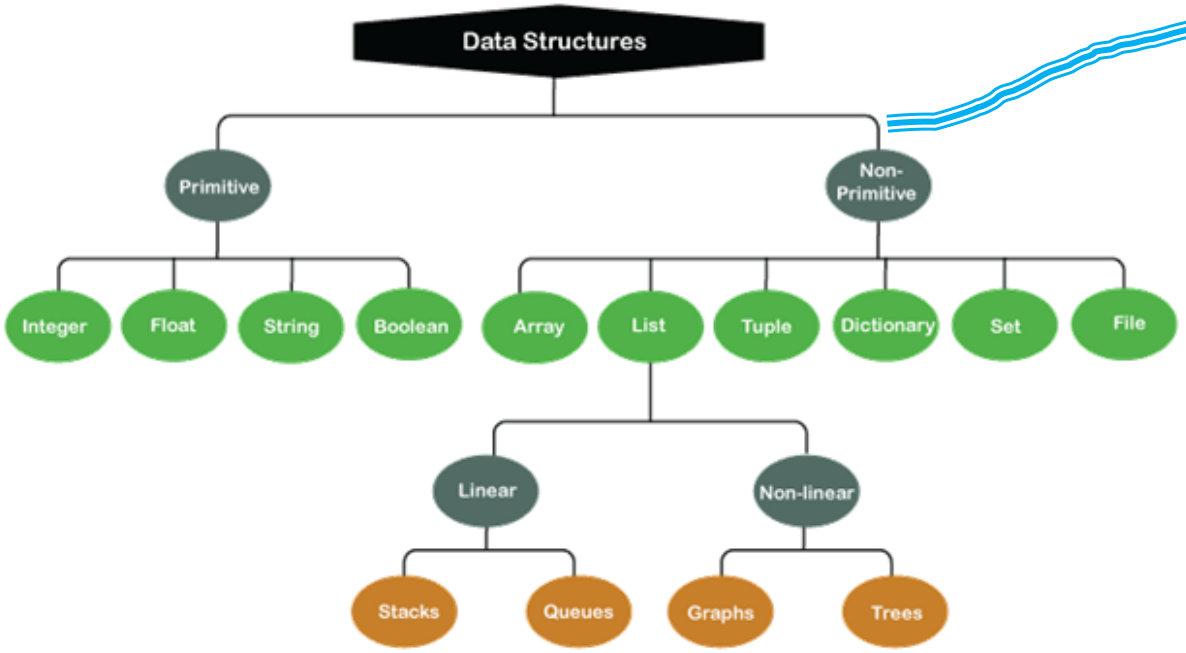
PYTHON DATA STRUCTURES

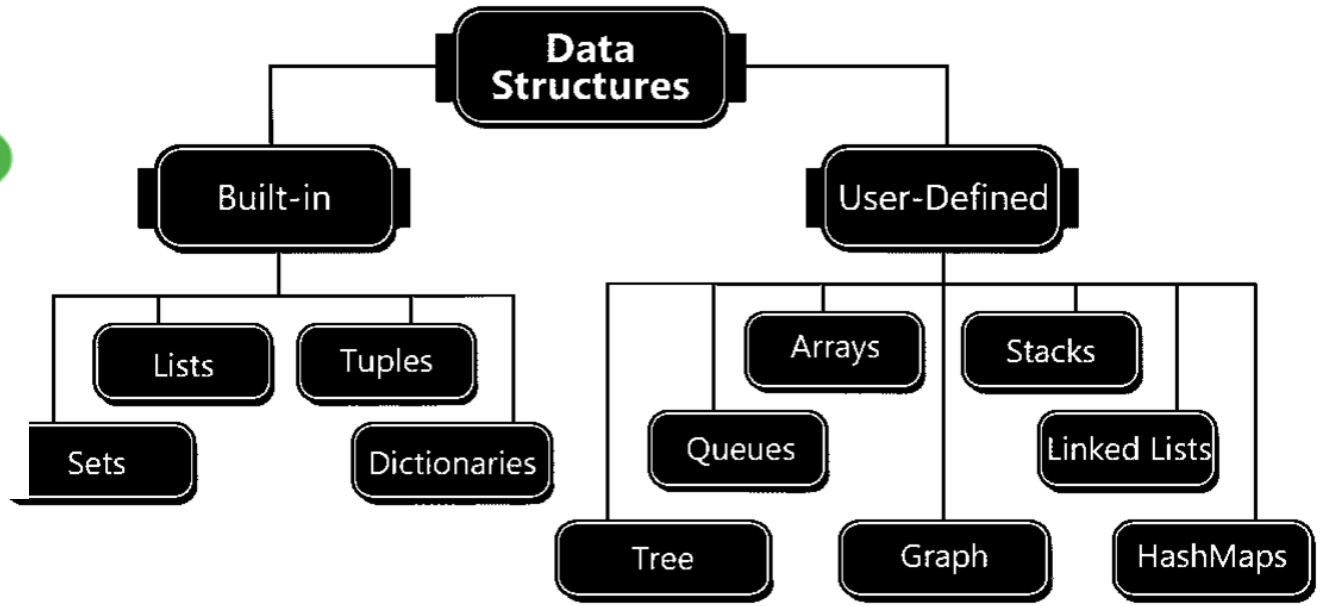




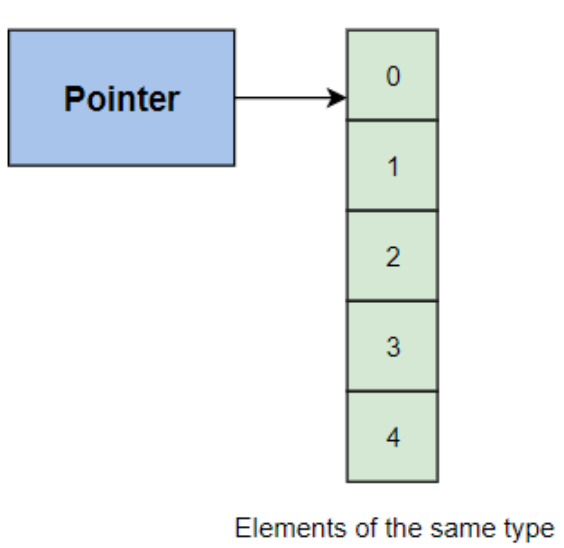

```
struct  int_object {
   long ob_refcnt;   /*a reference count  memory alloc/de-alloc*/
   PyTypeObject  *ob_type; /* type of variable*/
   size_t ob_size; */size of the data structure*/
   long ob_digit[1];  */ value of the integer*/
};
```

PYTHON DATA STRUCTURES CLASSIFICATION

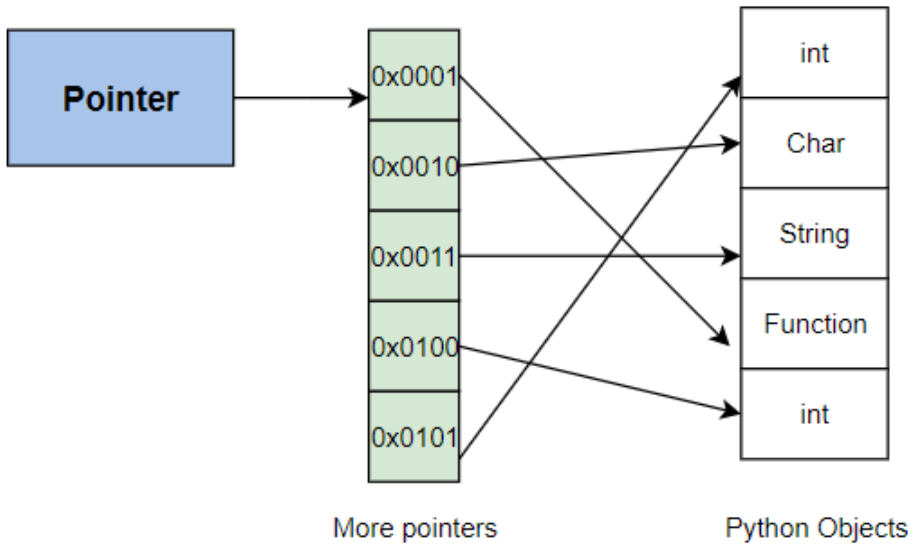




ARRAY VS LISTS

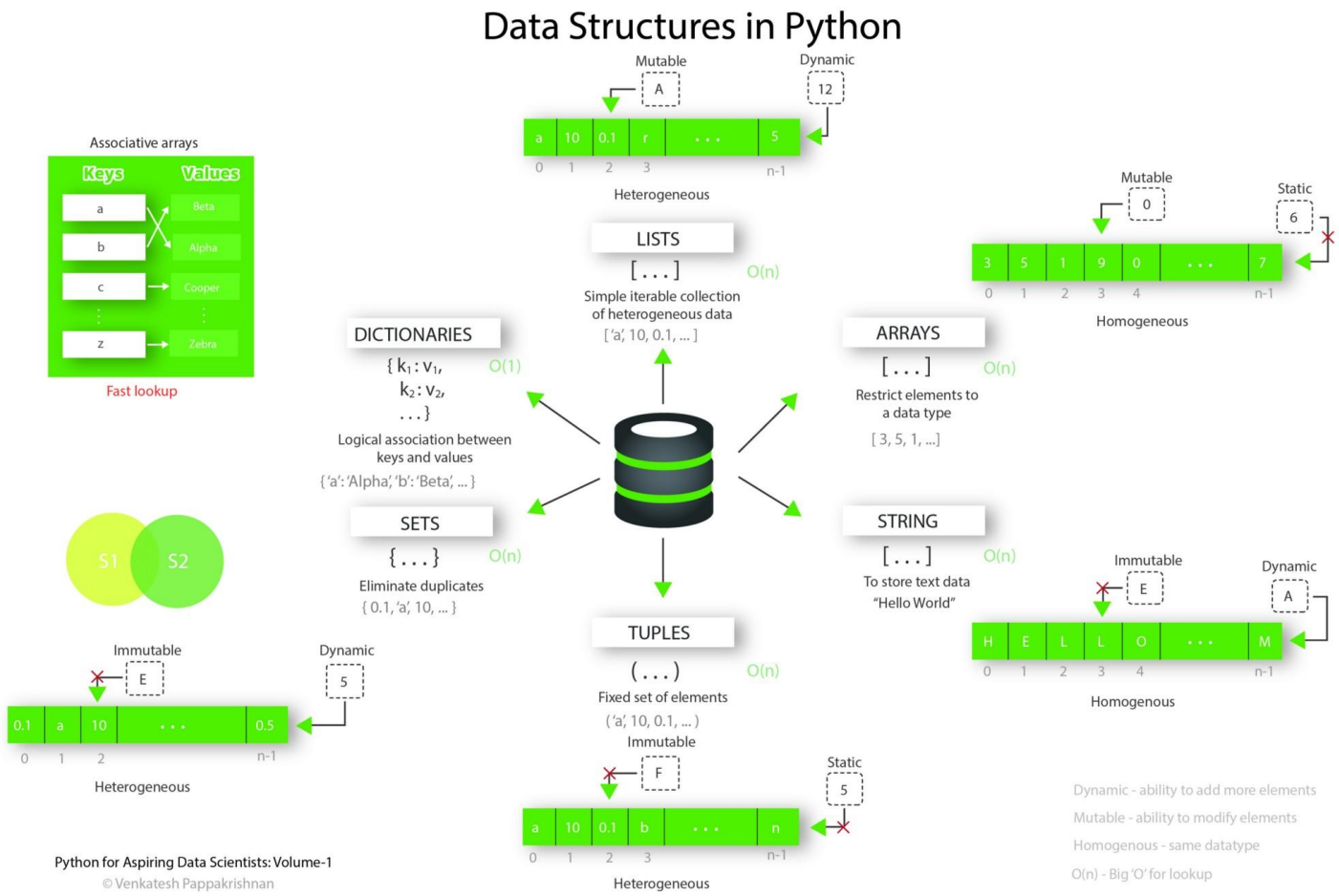


Python Arrays



Python Lists

NATIVE BUILTIN DATA STRUCTURES



ALGORITHM ANALYSIS EXAMPLE

Fibonacci:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

The sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Recursive – Version 1

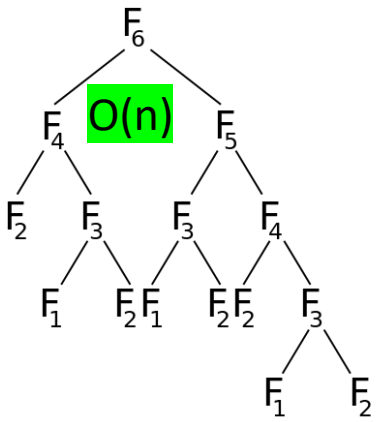
```
def fib(n):
    if n <= 1
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Space Complexity

O(n)

Time Complexity

2^n



T(n) = T(n-1) + T(n-2) + c
= 2T(n-1) + c (T(n-1) ~ T(n-2))
= 2*(2T(n-2) + c) + c
= 4T(n-2) + 3c
= 8T(n-3) + 7c
= 2^k * T(n - k) + (2^k - 1)*cLet's find the value of k for which: n - k = 0
k = nT(n) = 2^n * T(0) + (2^n - 1)*c
= 2^n * (1 + c) - i.e. T(n) ~ 2^n

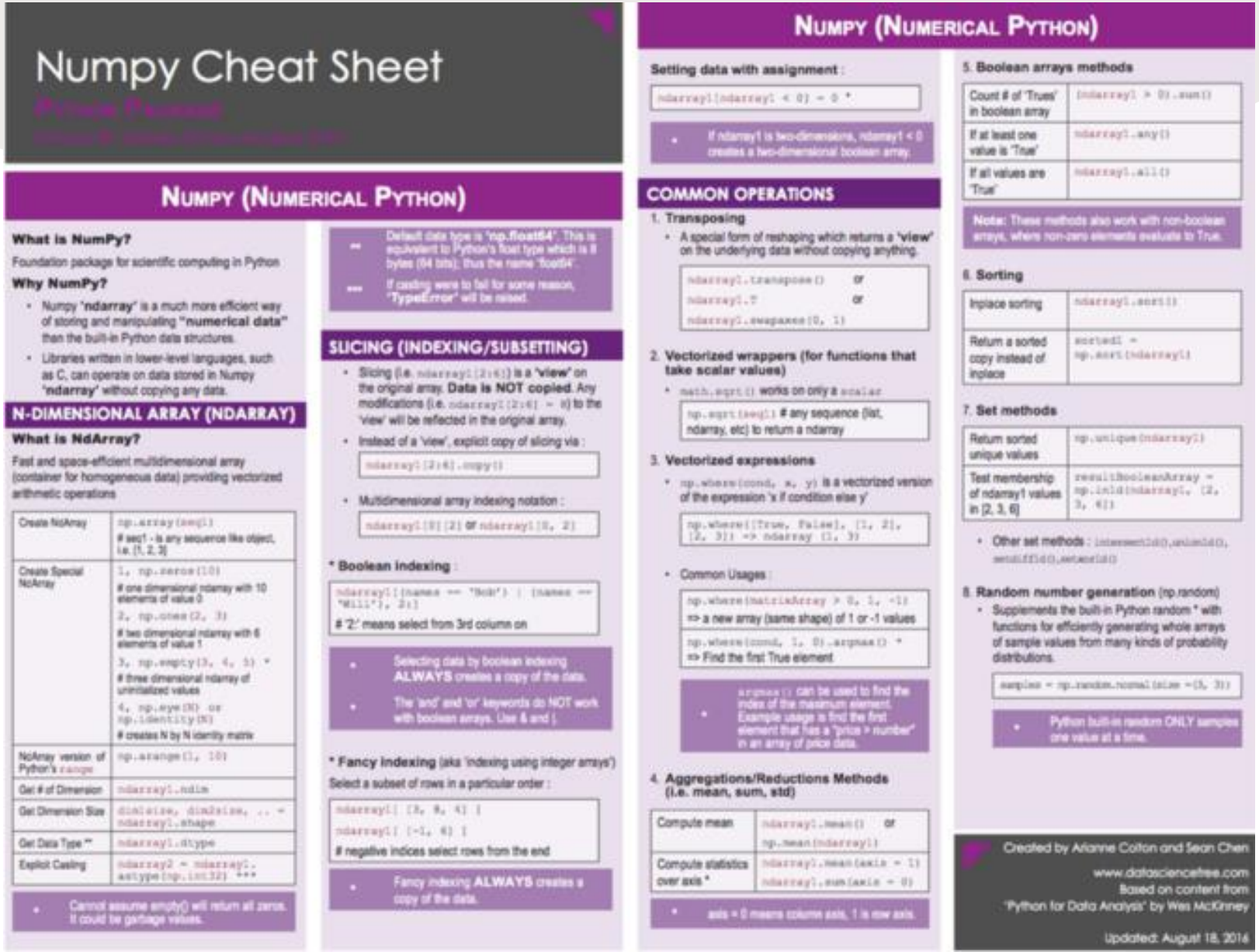
Recursive – Version 2

```
def fibonacci(n):
    if n <= 1
        return (n,0)
    else:
        (x,y) = fibonacci(n-1)
        return (x+y, y)
```

Time Complexity

O(n)

PYTHON – NUMPY DATA STRUCTURES



Numpy Cheat Sheet

NUMPY (NUMERICAL PYTHON)

What is NumPy?

Foundation package for scientific computing in Python

Why NumPy?

- Numpy '**ndarray**' is a much more efficient way of storing and manipulating "**numerical data**" than the built-in Python data structures.
- Libraries written in lower-level languages, such as C, can operate on data stored in Numpy '**ndarray**' without copying any data.

N-DIMENSIONAL ARRAY (NDARRAY)

What is NdArray?

Fast and space-efficient multidimensional array (container for homogeneous data) providing vectorized arithmetic operations

Create NdArray	`np.array(seq1)` # seq1 - is any sequence like object, i.e. [1, 2, 3]
Create Special NdArray	`1, np.zeros(10)` # one dimensional ndarray with 10 elements of value 0 `2, np.ones(2, 3)` # two dimensional ndarray with 6 elements of value 1 `3, np.empty(3, 4, 5)` * # three dimensional ndarray of uninitialized values `4, np.eye(N)` or `np.identity(N)` # creates N by N identity matrix
NdArray version of Python's `range`	`np.arange(1, 10)`
Get # of Dimension	`ndarray1.ndim`
Get Dimension Size	`dim1size, dim2size, .. = ndarray1.shape`
Get Data Type **	`ndarray1.dtype`
Explicit Casting	`ndarray2 = ndarray1.astype(np.int32)` ***

- Cannot assume empty() will return all zeros. It could be garbage values.

- ** Default data type is "**np.float64**". This is equivalent to Python's float type which is 8 bytes (64 bits); thus the name "float64".
- *** If casting were to fail for some reason, "**TypeError**" will be raised.

SLICING (INDEXING/SUBSETTING)

- Slicing (i.e. `ndarray1[2:6]`) is a "**view**" on the original array. **Data is NOT copied**. Any modifications (i.e. `ndarray1[2:6] = 0`) to the 'view' will be reflected in the original array.
- Instead of a 'view', explicit copy of slicing via :

```
ndarray1[2:6].copy()
```

- Multidimensional array indexing notation :

```
ndarray1[0][2] or ndarray1[0, 2]
```

*** Boolean Indexing** :

```
ndarray1[(names == "Bob") | (names == "Will"), 2:]
# '2:' means select from 3rd column on
```

- Selecting data by boolean indexing **ALWAYS** creates a copy of the data.
- The 'and' and 'or' keywords do NOT work with boolean arrays. Use & and |.

*** Fancy Indexing** (aka 'indexing using integer arrays')

Select a subset of rows in a particular order :

```
ndarray1[ [5, 0, 4] ]
ndarray1[ [-1, 6] ]
# negative indices select rows from the end
```

- Fancy indexing **ALWAYS** creates a copy of the data.

NUMPY (NUMERICAL PYTHON)

Setting data with assignment :

```
ndarray1[ndarray1 < 0] = 0 *
```

- * If ndarray1 is two-dimensional, ndarray1 < 0 creates a two-dimensional boolean array.

COMMON OPERATIONS

1. Transposing

- A special form of reshaping which returns a '**view**' on the underlying data without copying anything.

```
ndarray1.transpose()    or
ndarray1.T              or
ndarray1.swapaxes(0, 1)
```

2. Vectorized wrappers (for functions that take scalar values)

- `math.sqrt()` works on only a `scalar`

```
np.sqrt(seq1) # any sequence (list, ndarray, etc) to return a ndarray
```

3. Vectorized expressions

- `np.where(cond, x, y)` is a vectorized version of the expression 'x if condition else y'

```
np.where([True, False], [1, 2], [3, 3]) => ndarray ([1, 3])
```

- Common Usages :

```
np.where(matrixArray > 0, 1, -1)
=> a new array (same shape) of 1 or -1 values
np.where(cond, 1, 0).argmax() *
=> Find the first True element
```

- * `argmax()` can be used to find the index of the maximum element. Example usage is find the first element that has a "price > number" in an array of price data.

4. Aggregations/Reductions Methods (i.e. mean, sum, std)

Compute mean	`ndarray1.mean()` or `np.mean(ndarray1)`
Compute statistics over axis *	`ndarray1.mean(axis = 1)` `ndarray1.sum(axis = 0)`

- * axis = 0 means column axis, 1 is row axis.

5. Boolean arrays methods

Count # of 'Trues' in boolean array	`(ndarray1 > 0).sum()`
If at least one value is 'True'	`ndarray1.any()`
If all values are 'True'	`ndarray1.all()`

Note: These methods also work with non-boolean arrays, where non-zero elements evaluate to True.

6. Sorting

Inplace sorting	`ndarray1.sort()`
Return a sorted copy instead of inplace	`sorted1 = np.sort(ndarray1)`

7. Set methods

Return sorted unique values	`np.unique(ndarray1)`
Test membership of ndarray1 values in [2, 3, 6]	`result1booleanArray = np.in1d(ndarray1, [2, 3, 6])`

- Other set methods : `intersect1d()`,`union1d()`, `setdiff1d()`,`setxor1d()`

8. Random number generation (np.random)

- Supplements the built-in Python random * with functions for efficiently generating whole arrays of sample values from many kinds of probability distributions.

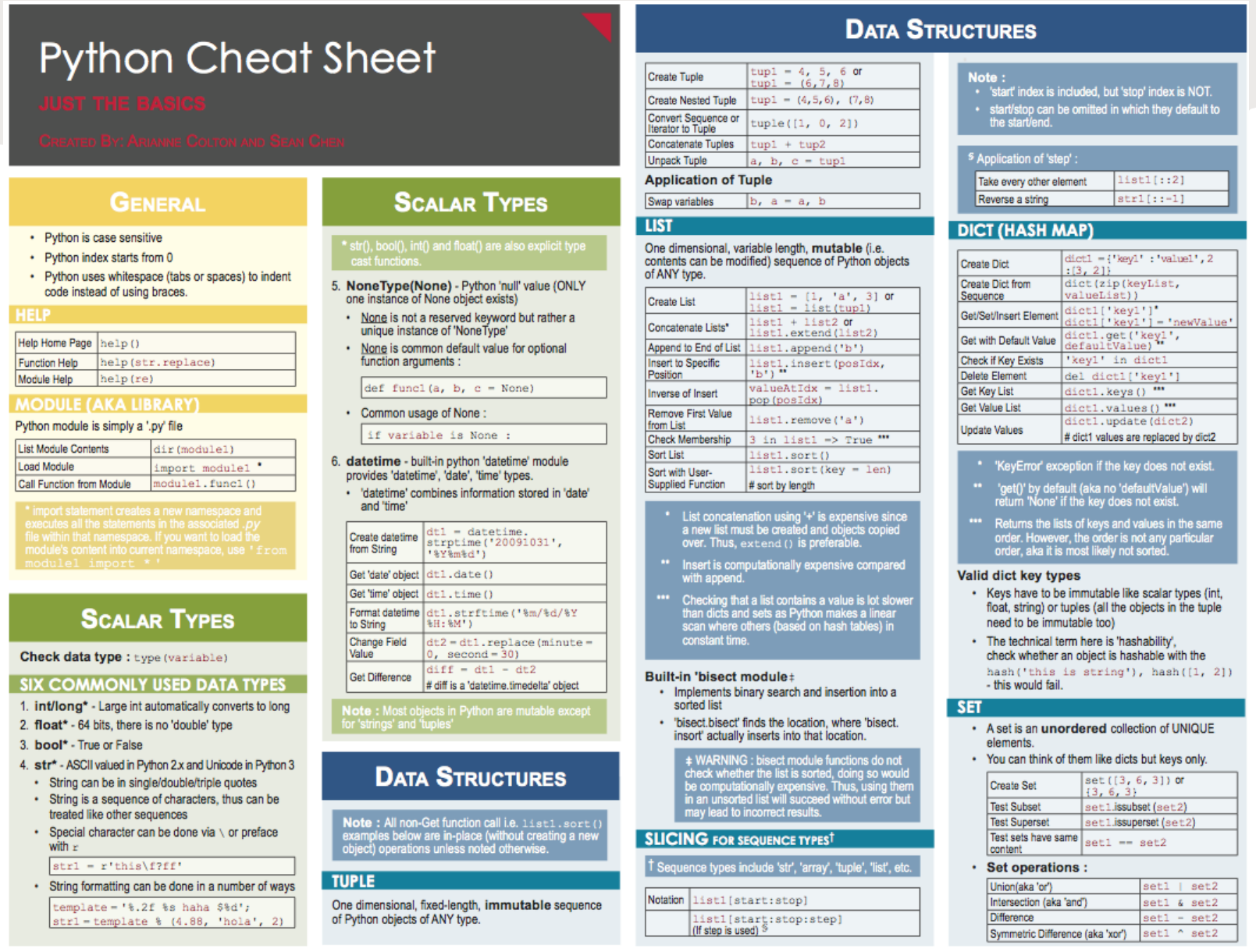
```
samples = np.random.normal(size =(5, 3))
```

- Python built-in random ONLY samples one value at a time.

Created by Arianne Colton and Sean Chen
www.datasciencefree.com
Based on content from
"Python for Data Analysis" by Wes McKinney

Updated: August 18, 2016

PYTHON – INNATE DATA STRUCTURES



Python Cheat Sheet

JUST THE BASICS

Created By: Arianne Colton and Sean Chen

General

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP

Help Home Page	`help()`
Function Help	`help(str.replace)`
Module Help	`help(re)`

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	`dir(module1)`
Load Module	`import module1` *
Call Function from Module	`module1.func1()`

* import statement creates a new namespace and executes all the statements in the associated *.py* file within that namespace. If you want to load the module's content into current namespace, use `'from module1 import *'`

Scalar Types

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via \ or preface with r

```
str1 = r'this\f?ff'
```

   - String formatting can be done in a number of ways

```
template = '%.2f %s haha $%d';
str1 = template % (4.88, 'hola', 2)
```

Scalar Types

* str(), bool(), int() and float() are also explicit type cast functions.

5. **NoneType(None)** - Python 'null' value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments :

```
def func1(a, b, c = None)
```

   - Common usage of None :

```
if variable is None :
```

6. **datetime** - built-in python 'datetime' module provides 'datetime', 'date', 'time' types.
   - 'datetime' combines information stored in 'date' and 'time'

Create datetime from String	`dt1 = datetime.strptime('20091031', '%Y%m%d')`
Get 'date' object	`dt1.date()`
Get 'time' object	`dt1.time()`
Format datetime to String	`dt1.strftime('%m/%d/%Y %H:%M')`
Change Field Value	`dt2 = dt1.replace(minute = 0, second = 30)`
Get Difference	`diff = dt1 - dt2` # diff is a 'datetime.timedelta' object

Note : Most objects in Python are mutable except for 'strings' and 'tuples'

Data Structures

Note : All non-Get function call i.e. `list1.sort()` examples below are in-place (without creating a new object) operations unless noted otherwise.

TUPLE

One dimensional, fixed-length, **immutable** sequence of Python objects of ANY type.

Create Tuple	`tup1 = 4, 5, 6` or `tup1 = (6,7,8)`
Create Nested Tuple	`tup1 = (4,5,6), (7,8)`
Convert Sequence or Iterator to Tuple	`tuple([1, 0, 2])`
Concatenate Tuples	`tup1 + tup2`
Unpack Tuple	`a, b, c = tup1`

Application of Tuple

Swap variables	`b, a = a, b`

Data Structures

LIST

One dimensional, variable length, **mutable** (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List	`list1 = [1, 'a', 3]` or `list1 = list(tup1)`
Concatenate Lists*	`list1 + list2` or `list1.extend(list2)`
Append to End of List	`list1.append('b')`
Insert to Specific Position	`list1.insert(posIdx, 'b')` **
Inverse of Insert	`valueAtIdx = list1.pop(posIdx)`
Remove First Value from List	`list1.remove('a')`
Check Membership	`3 in list1 => True` ***
Sort List	`list1.sort()`
Sort with User-Supplied Function	`list1.sort(key = len)` # sort by length

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, `extend()` is preferable.

** Insert is computationally expensive compared with append.

*** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in 'bisect module‡

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insort' actually inserts into that location.

‡ WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

SLICING FOR SEQUENCE TYPES†

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

Notation	`list1[start:stop]`
	`list1[start:stop:step]` (If step is used) $

Note :
- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

$ Application of 'step' :

Take every other element	`list1[::2]`
Reverse a string	`str1[::-1]`

DICT (HASH MAP)

Create Dict	`dict1 = {'key1' : 'value1', 2 :[3, 2]}`
Create Dict from Sequence	`dict(zip(keyList, valueList))`
Get/Set/Insert Element	`dict1['key1']` `dict1['key1'] = 'newValue'`
Get with Default Value	`dict1.get('key1', defaultValue)` **
Check if Key Exists	`'key1' in dict1`
Delete Element	`del dict1['key1']`
Get Key List	`dict1.keys()` ***
Get Value List	`dict1.values()` ***
Update Values	`dict1.update(dict2)` # dict1 values are replaced by dict2

* 'KeyError' exception if the key does not exist.

** 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

Valid dict key types

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string')`, `hash([1, 2])` - this would fail.

SET

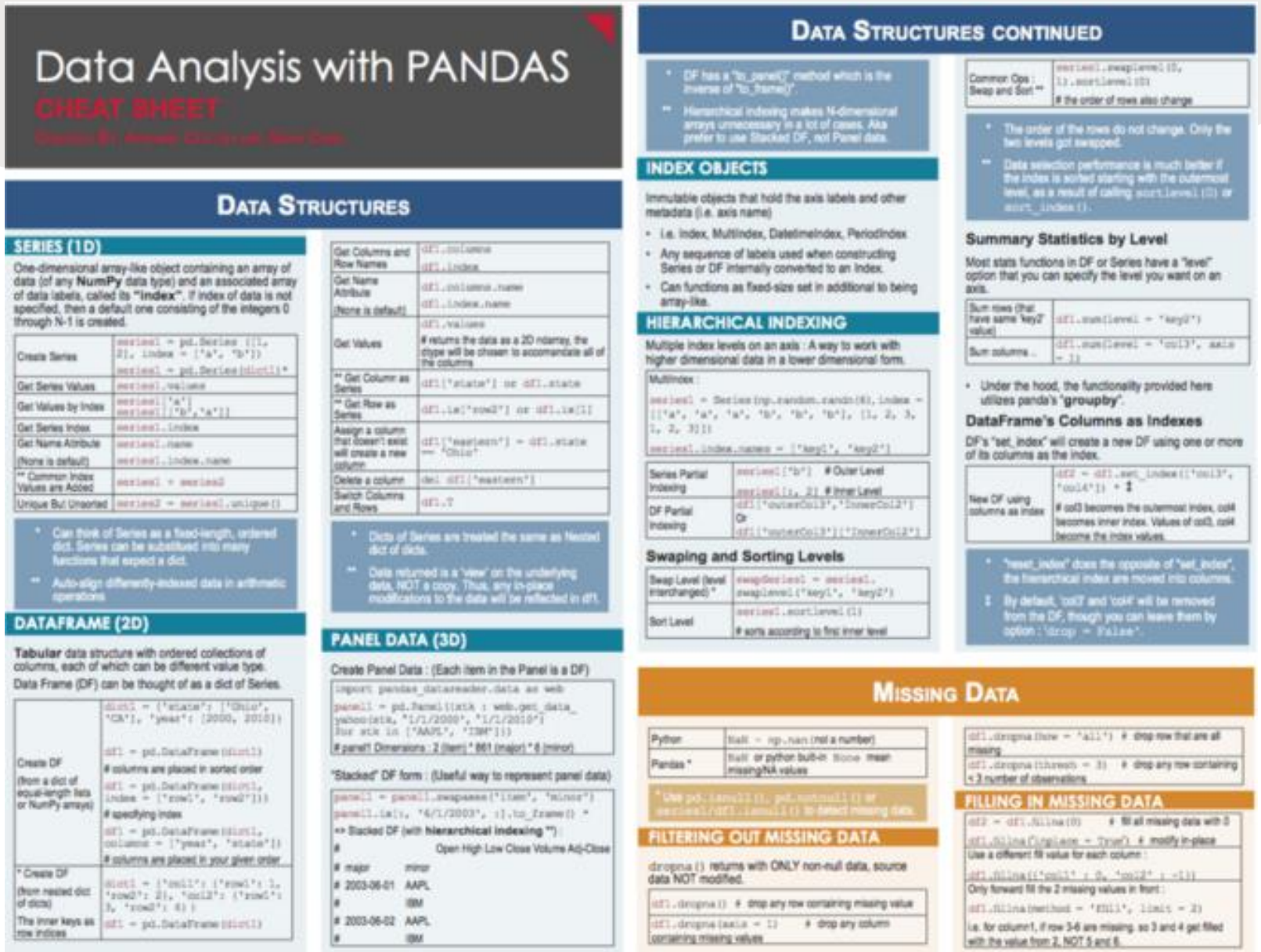
- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	`set([3, 6, 3])` or `{3, 6, 3}`
Test Subset	`set1.issubset(set2)`
Test Superset	`set1.issuperset(set2)`
Test sets have same content	`set1 == set2`

- **Set operations :**

Union(aka 'or')	`set1 | set2`
Intersection (aka 'and')	`set1 & set2`
Difference	`set1 - set2`
Symmetric Difference (aka 'xor')	`set1 ^ set2`

PYTHON – PANDAS



Data Analysis with PANDAS

CHEAT SHEET

DATA STRUCTURES

SERIES (1D)

One-dimensional array-like object containing an array of data (of any **NumPy** data type) and an associated array of data labels, called its **"index"**. If index of data is not specified, then a default one consisting of the integers 0 through N-1 is created.

Create Series	`series1 = pd.Series ([1, 2], index = ['a', 'b'])` `series2 = pd.Series(dict1)*`
Get Series Values	`series1.values`
Get Values by Index	`series1['a']` `series1[['b','a']]`
Get Series Index	`series1.index`
Get Name Attribute (None is default)	`series1.name` `series1.index.name`
** Common Index Values are Acced	`series1 + series2`
Unique But Unsorted	`series2 = series1.unique()`

- * Can think of Series as a fixed-length, ordered dict. Series can be substituted into many functions that expect a dict.
- ** Auto-align differently-indexed data in arithmetic operations

Get Columns and Row Names	`df1.columns` `df1.index`
Get Name Attribute (None is default)	`df1.columns.name` `df1.index.name`
Get Values	`df1.values` # returns the data as a 2D ndarray, the dtype will be chosen to accomandate all of the columns
** Get Column as Series	`df1['state']` or `df1.state`
** Get Row as Series	`df1.ix['row2']` or `df1.ix[1]`
Assign a column that doesn't exist will create a new column.	`df1['eastern'] = df1.state == 'Ohio'`
Delete a column	`del df1['eastern']`
Switch Columns and Rows	`df1.T`

- * Dicts of Series are treated the same as Nested dict of dicts.
- ** Data returned is a 'view' on the underlying data, NOT a copy. Thus, any in-place modifications to the data will be reflected in df1.

DATAFRAME (2D)

Tabular data structure with ordered collections of columns, each of which can be different value type.

Data Frame (DF) can be thought of as a dict of Series.

Create DF
(from a dict of equal-length lists or NumPy arrays)
	`dict1 = {'state': ['Ohio', 'CA'], 'year': [2000, 2010]}`
	`df1 = pd.DataFrame(dict1)`
	# columns are placed in sorted order
	`df1 = pd.DataFrame(dict1, index = ['row1', 'row2'])`
	# specifying index
	`df1 = pd.DataFrame(dict1, columns = ['year', 'state'])`
	# columns are placed in your given order

* Create DF (from nested dict of dicts) The inner keys as row indices	`dict1 = {'col1': {'row1': 1, 'row2': 2}, 'col2': {'row1': 3, 'row2': 4}}` `df1 = pd.DataFrame(dict1)`

PANEL DATA (3D)

Create Panel Data : (Each item in the Panel is a DF)

```
import pandas.datareader.data as web
panel1 = pd.Panel(dict : web.get_data_
yahoo(stk, '1/1/2000', '1/1/2010')
for stk in ['AAPL', 'IBM']})
# panel1 Dimensions : 2 (item) * 861 (major) * 6 (minor)
```

"Stacked" DF form : (Useful way to represent panel data)

```
panel1 = panel1.swapaxes('item', 'minor')
panel1.ix[:, '6/1/2003', :].to_frame() *
```

=> Stacked DF (with **hierarchical indexing** **) :

		Open	High	Low	Close	Volume	Adj-Close
# major	minor						
# 2003-06-01	AAPL						
#	IBM						
# 2003-06-02	AAPL						
#	IBM						

DATA STRUCTURES CONTINUED

- * DF has a "to_panel()" method which is the inverse of "to_frame()".
- ** Hierarchical indexing makes N-dimensional arrays unnecessary in a lot of cases. Aka prefer to use Stacked DF, not Panel data.

INDEX OBJECTS

Immutable objects that hold the axis labels and other metadata (i.e. axis name)

- i.e. Index, MultiIndex, DatetimeIndex, PeriodIndex
- Any sequence of labels used when constructing Series or DF internally converted to an Index.
- Can functions as fixed-size set in additional to being array-like.

HIERARCHICAL INDEXING

Multiple index levels on an axis : A way to work with higher dimensional data in a lower dimensional form.

Multiindex :

```
series1 = Series(np.random.randn(6), index =
[['a', 'a', 'a', 'b', 'b', 'b'], [1, 2, 3,
1, 2, 3]])
series1.index.names = ['key1', 'key2']
```

Series Partial Indexing	`series1['b']` # Outer Level `series1[:, 2]` # Inner Level

DF Partial Indexing	`df1['outerCol3','innerCol2']`
	Or
	`df1['outerCol3']['innerCol2']`

Swaping and Sorting Levels

Swap Level (level interchanged) *	`swapDerlevel = series1.swaplevel('key1', 'key2')`
Sort Level	`series1.sortlevel(1)` # sorts according to first inner level

Common Ops : Swap and Sort **	`series1.swaplevel(0, 1).sortlevel(0)` # the order of rows also change

- * The order of the rows do not change. Only the two levels got swapped.
- ** Data selection performance is much better if the index is sorted starting with the outermost level, as a result of calling `sortlevel(0)` or `sort_index()`.

Summary Statistics by Level

Most stats functions in DF or Series have a "level" option that you can specify the level you want on an axis.

Sum rows (that have same 'key2' value)	`df1.sum(level = 'key2')`
Sum columns ..	`df1.sum(level = 'col3', axis = 1)`

- Under the hood, the functionality provided here utilizes panda's "**groupby**".

DataFrame's Columns as Indexes

DF's "set_index" will create a new DF using one or more of its columns as the index.

New DF using columns as index	`df2 = df1.set_index(['col3', 'col4']) ‡` # col3 becomes the outermost index, col4 becomes inner index. Values of col3, col4 become the index values.

- "reset_index" does the opposite of "set_index", the hierarchical index are moved into columns.
- ‡ By default, 'col3' and 'col4' will be removed from the DF, though you can leave them by option : `drop = False`.

MISSING DATA

Python	`NaN = np.nan` (not a number)
Pandas *	`NaN` or python built-in `None` mean missing/NA values

* Use `pd.isnull()`, `pd.notnull()` or `series1/df1.isnull()` to detect missing data.

FILTERING OUT MISSING DATA

`dropna()` returns with ONLY non-null data, source data NOT modified.

`df1.dropna()`	# drop any row containing missing value
`df1.dropna(axis = 1)`	# drop any column containing missing values

`df1.dropna(how = 'all')`	# drop row that are all missing
`df1.dropna(thresh = 3)`	# drop any row containing < 3 number of observations

FILLING IN MISSING DATA

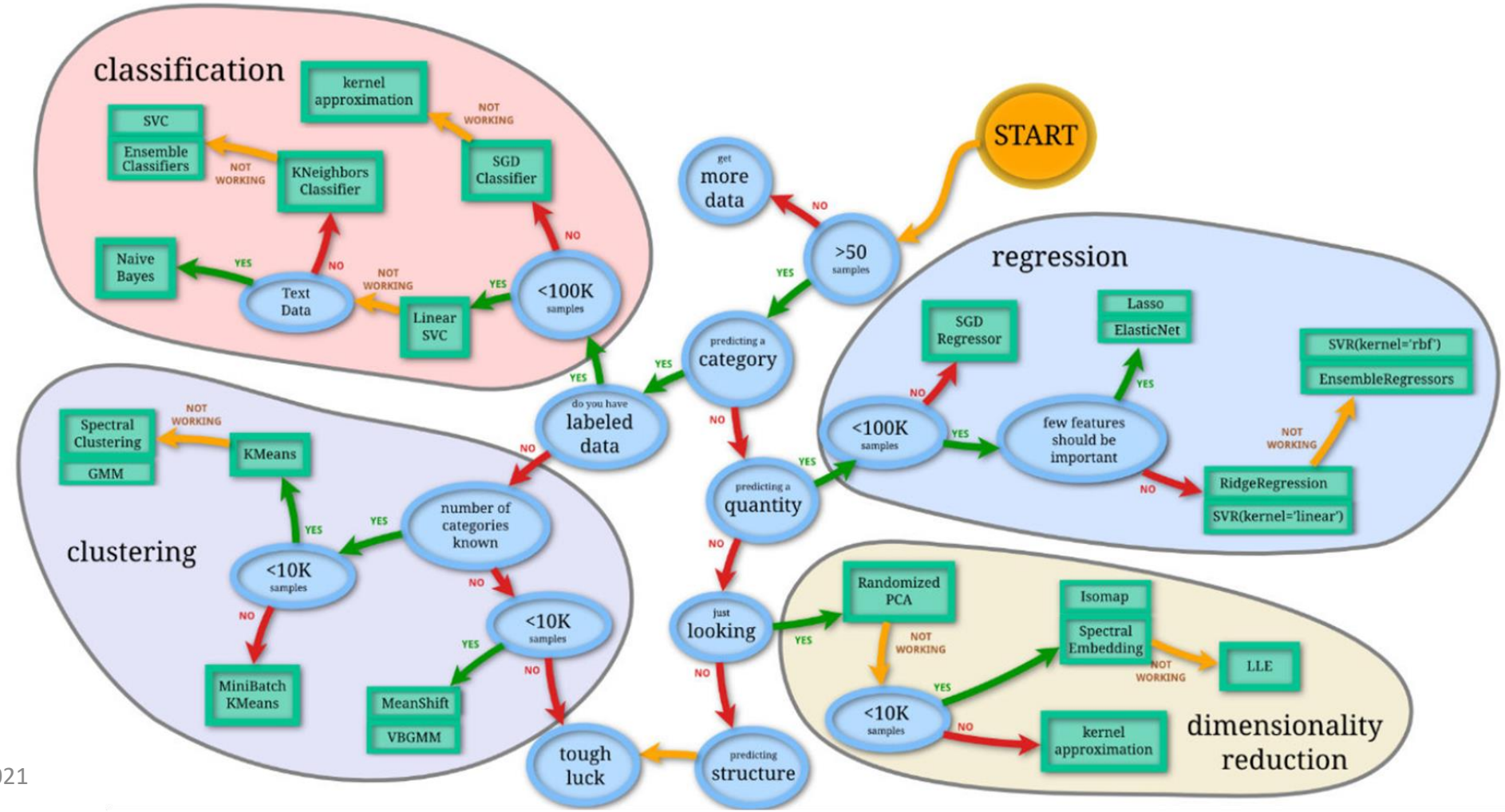
`df2 = df1.fillna(0)`	# fill all missing data with 0
`df1.fillna(0,inplace = True)`	# modify in-place
Use a different fill value for each column :	
`df1.fillna({'col1' : 0, 'col2' : -1})`	
Only forward fill the 2 missing values in front :	
`df1.fillna(method = 'ffill', limit = 2)`	
i.e. for column1, if row 3-6 are missing, so 3 and 4 get filled with the value from 2, NOT 5 and 6.	

PYTHON MACHINE LEARNING LIBRARIES

- End-to-end framework
  - data exploration (+ pandas + holoviews)
  - data preprocessing (+ pandas)
    - cleaning/missing values
    - normalization
  - training
  - testing
  - application
- "Classic" machine learning only
- https://scikit-learn.org/stable/

- High-level framework for deep learning
- TensorFlow backend
- Layer types
  - dense
  - convolutional
  - pooling
  - embedding
  - recurrent
  - activation
  - …
- https://keras.io/

SCIKIT-LEARN ALGORITHM FLOW



OUTLINE

- ~~Data Science—A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- ~~Frameworks (Python & SQL) (Python Data Structures)~~
- Environment Setup and Get Started
- COVID Data Scraping from WEB
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

SETTING UP ANACONDA

Windows:

- Open anaconda prompt
- Type `conda -V`
- **If you get an error**, install Anaconda: https://docs.anaconda.com/anaconda/install/windows/
  - #8 is important: **DO NOT** add to your path
- **If no error**, consider upgrading conda:

```
conda update conda
```

Linux:

- Open a terminal
- Type `conda -V`
- **If you get an error**, install Anaconda: https://docs.anaconda.com/anaconda/install/linux/
- **If no error**, consider upgrading conda:

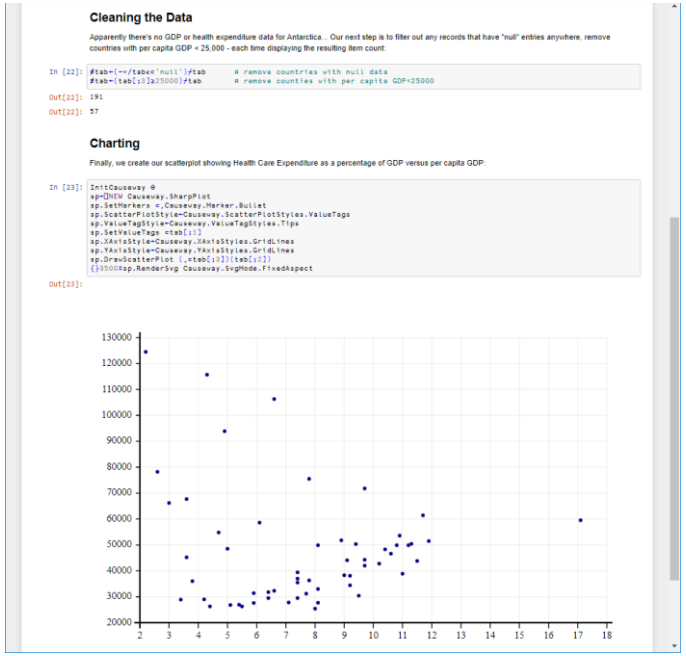
```
conda update conda
```

SETTING UP ENVIRONMENT

- Create (once): `conda env create -f [path]`
  - Turn on an environment
    - Windows: `conda activate [envname]`
    - Linux: `source activate [envname]`
  - Use the environment (write/save code, upgrade/install packages)
  - Switch back to the global environment, named (base):
    - Windows: `conda deactivate`
    - Linux: `source deactivate`
- Destroy (once): `conda remove --name [envname] --all`

JUPYTER NOTEBOOK

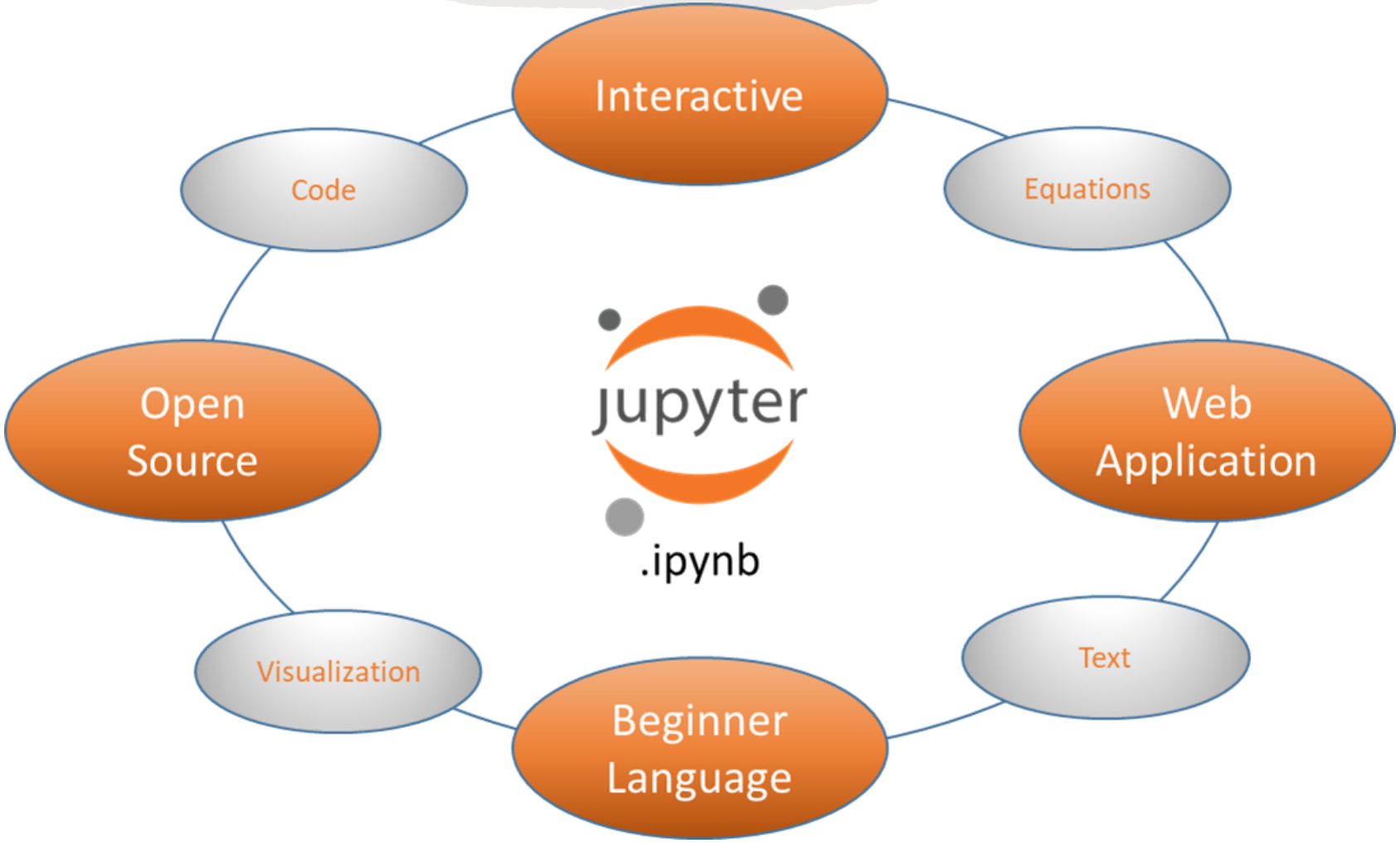
- A *notebook* combines the functionality of
  - a word processor — handles formatted text
  - a "shell" or "kernel" — executes statements in a programming language and includes output inline
  - a rendering engine — renders HTML in addition to plain text



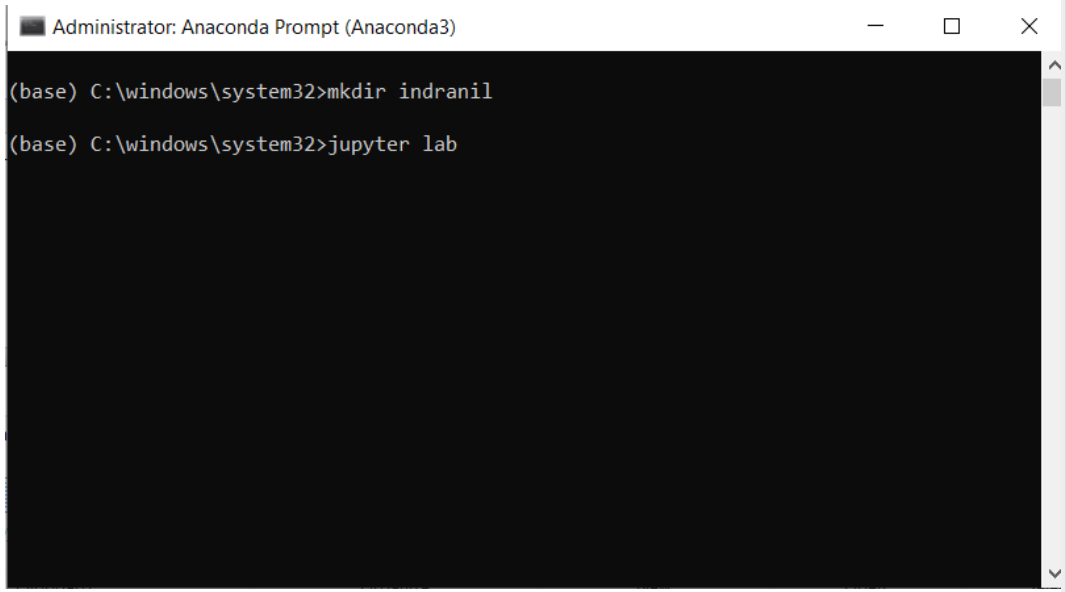
- A single document that combines explanations with executable code and its output — an ideal way to provide:
  - reproducible research results
  - documentation of processes
  - instructions
  - tutorials and training materials of all shapes and sizes
- A digital learning environment for computational thinking
- *Dozens of programming languages*
- beginning with **Pyt**hon, **Ju**lia, **R**

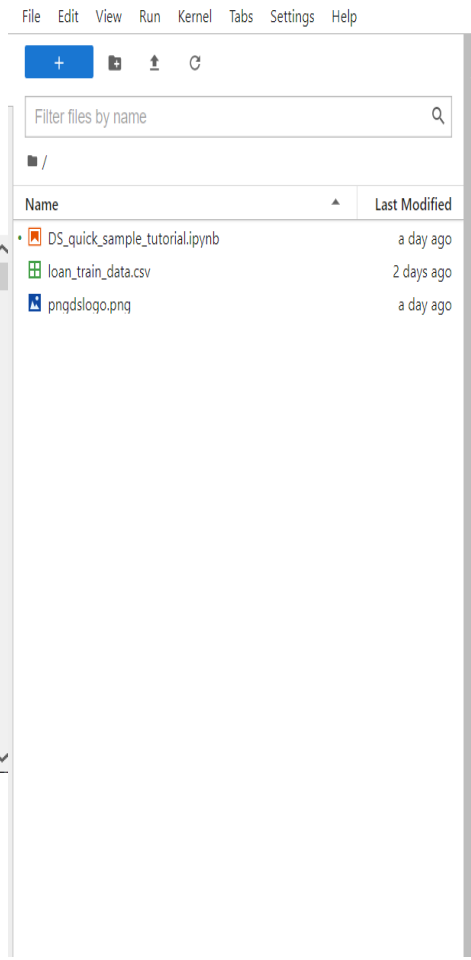
Spyder/VS Studio Code/PyCharm

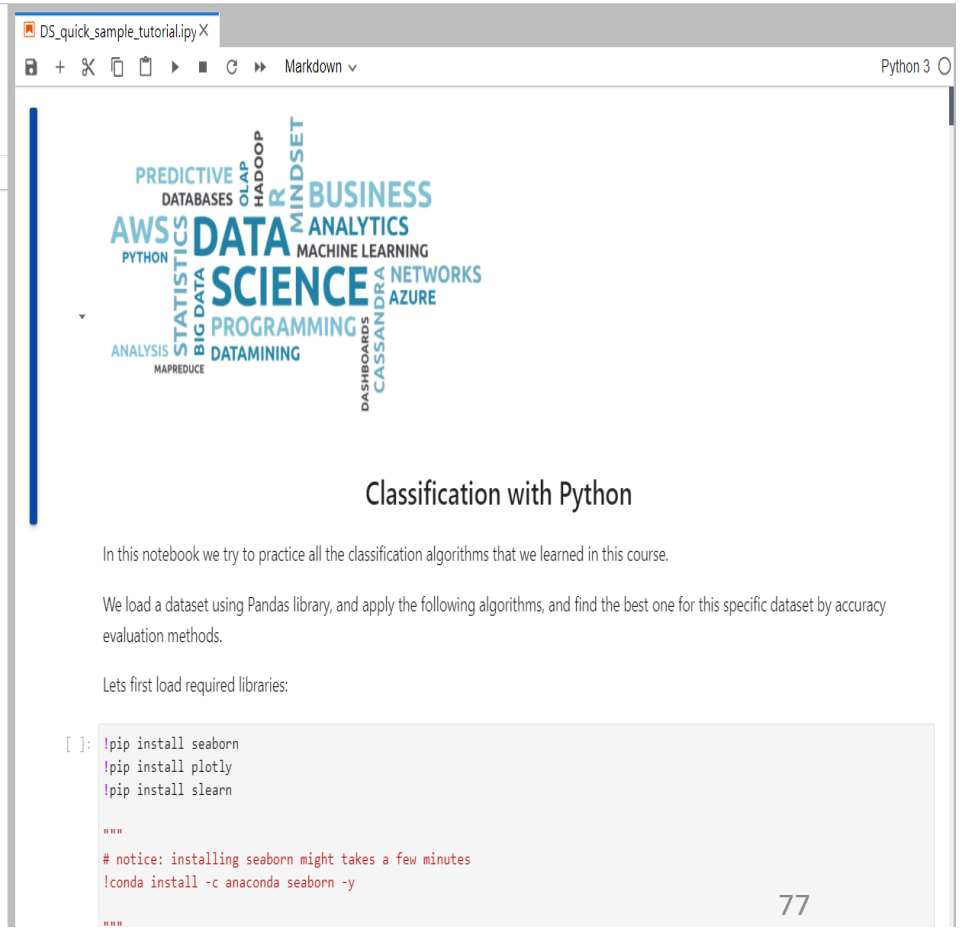
NOTEBOOK FLOW



SIMPLE TUTORIAL

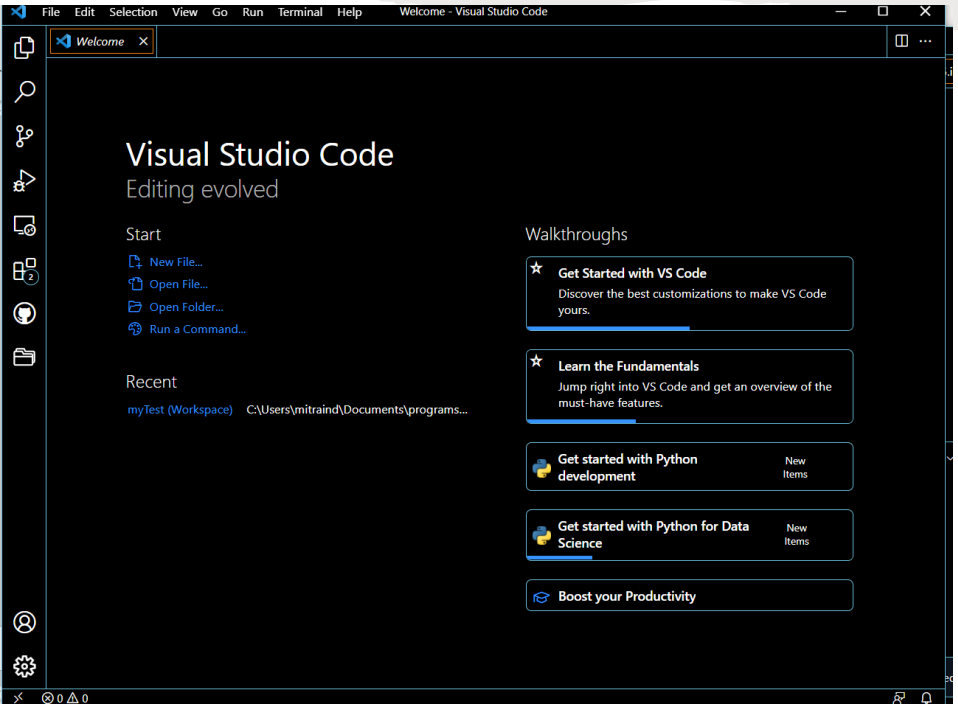






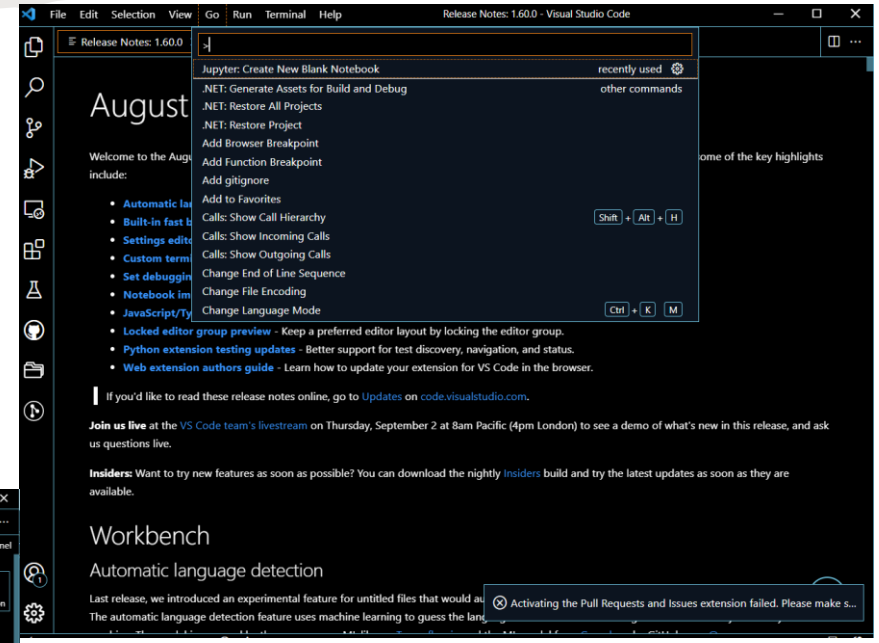
JUPYTER LAB - VS STUDIO CODE

1 [VS Code Download Link]

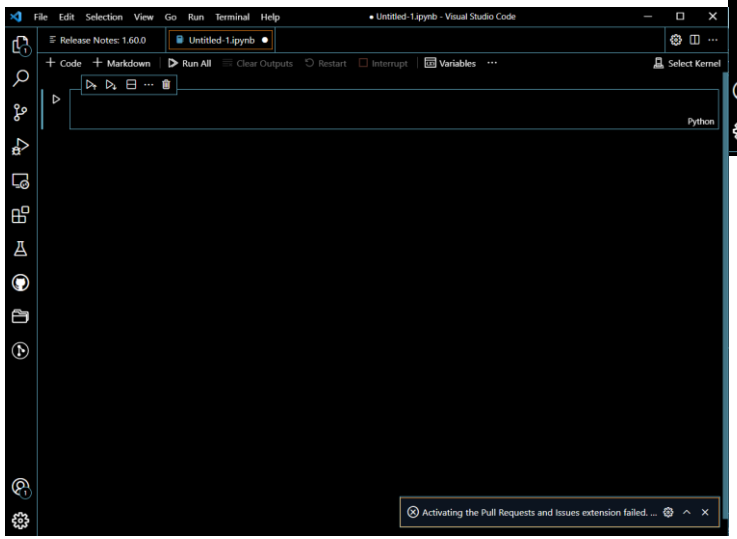


2

Ctr+Shift+P



3



RESOURCES

Data Ingestion

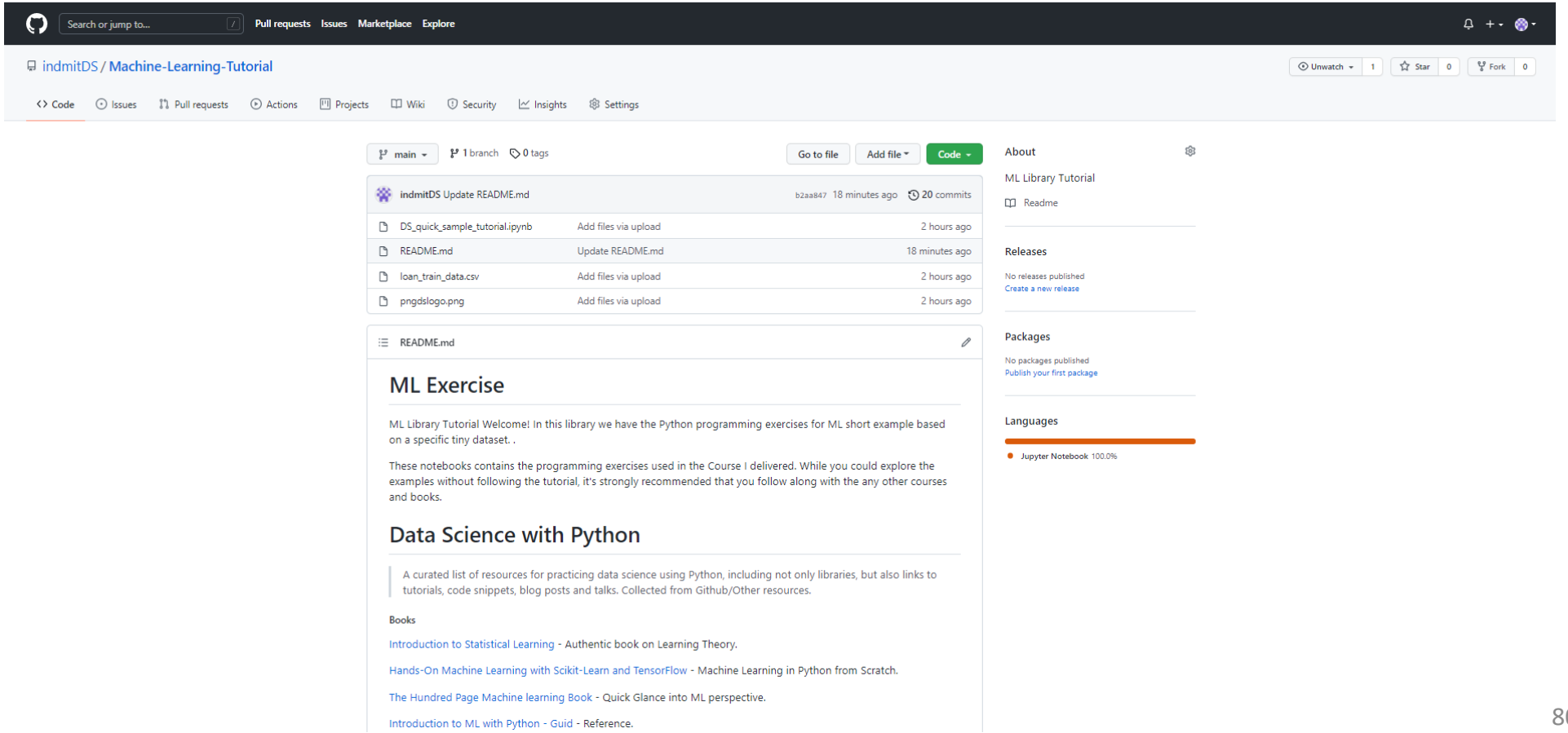
- Data Formats
- Sample Database
- SQL Server w/ Python

Learning Resources

- Google Colab
- Github
- Kaggle
- UCI ML repository
- Links to resources
- Machine Learning Resources

TUTORIAL RESOURCES

github.com/indmitDS/Machine-Learning-Tutorial

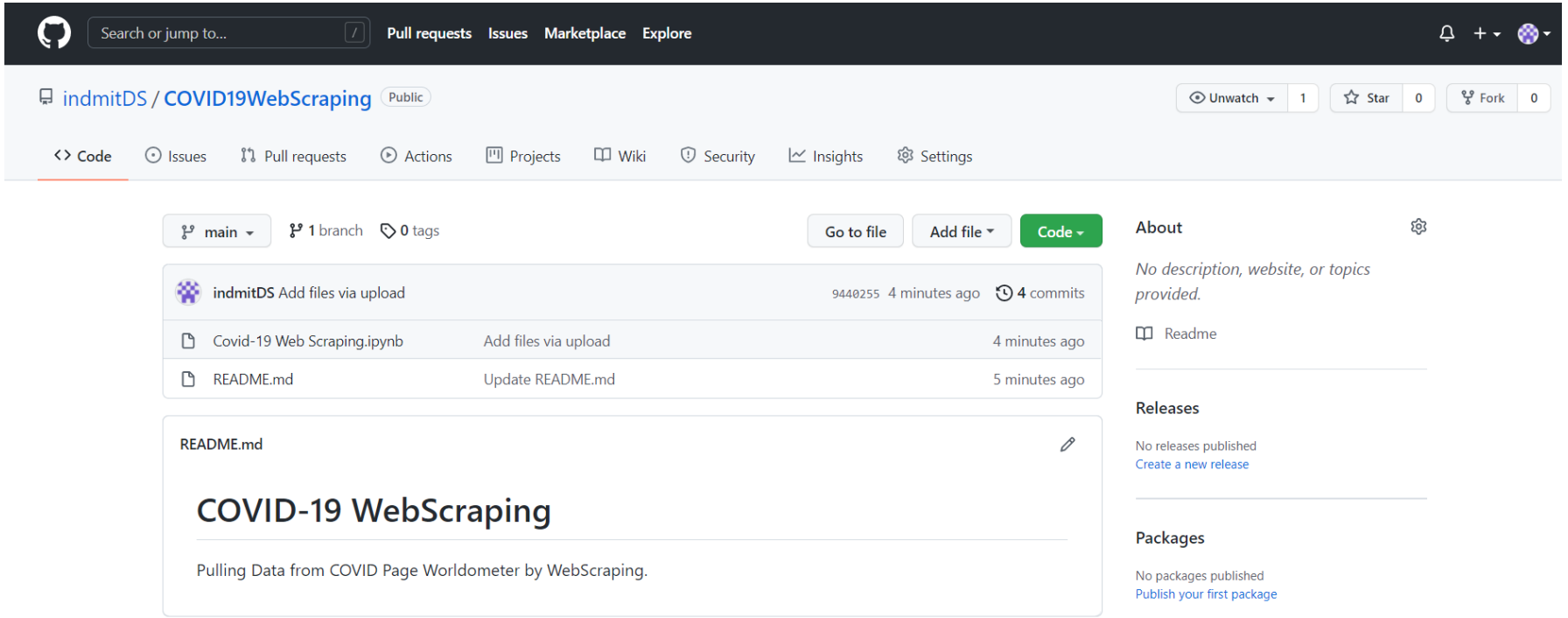


OUTLINE

- ~~Data Science—A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- ~~Frameworks (Python & SQL) (Python Data Structures)~~
- ~~Environment Setup and Get Started~~
- COVID Data Scraping from WEB
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

COVID WEB SCRAPING



COVID DATA SCRAPING FROM WEB

Github link

COVID WORLDOMETER DATA

worldometer Coronavirus Population

COVID-19 CORONAVIRUS PANDEMIC

Last updated: September 17, 2021, 21:18 GMT

Weekly Trends - Graphs - Countries - News

Coronavirus Cases:

228,270,957

view by country

#	Country, Other	Total Cases	New Cases	Total Deaths	New Deaths	Total Recovered	New Recovered	Active Cases	Serious, Critical	Tot Cases/ 1M pop	Deaths/ 1M pop	Total Tests	Tests/ 1M pop	Population
	World	228,270,957	+460,602	4,690,855	+7,064	204,878,950	+407,077	18,701,152	100,647	29,285	601.8			
1	USA	42,740,483	+101,481	690,101	+1,345	32,392,179	+44,453	9,658,203	25,255	128,216	2,070	617,526,522	1,852,502	333,347,223
2	India	33,415,889	+35,367	444,563	+285	32,624,718	+33,850	346,608	8,944	23,930	318	547,701,729	392,229	1,396,383,360
3	Brazil	21,080,219	+11,202	589,573	+296	20,173,064		317,582	8,318	98,329	2,750	57,095,219	266,322	214,384,223
4	UK	7,371,301	+32,651	134,983	+178	5,934,018	+26,989	1,302,300	1,020	107,899	1,976	289,860,893	4,242,893	68,316,802
5	Russia	7,234,425	+19,905	196,626	+791	6,469,017	+16,619	568,782	2,300	49,547	1,347	185,600,000	1,271,145	146,010,102
6	France	6,942,105	+7,373	115,960	+66	*6,612,964*	+17,525	213,181	2,000	106,070	1,772	134,787,318	2,059,455	65,448,064
7	Turkey	6,794,700	+27,692	61,140	+237	6,285,887	+23,197	447,673	633	79,531	716	81,764,689	957,046	85,434,482
8	Iran	5,396,013	+17,605	116,436	+364	4,708,195	+25,491	571,382	6,902	63,267	1,365	30,123,729	353,191	85,290,152
9	Argentina	5,234,851		114,101		5,087,120		33,630	1,614	114,557	2,497	24,252,818	530,737	45,696,487
10	Colombia	4,936,052		125,782		4,774,661		35,609	542	95,776	2,441	24,920,135	483,534	51,537,524
11	Spain	4,929,546	+3,222	85,783	+44	4,633,527	+8,475	210,236	1,028	105,385	1,834	63,040,355	1,347,688	46,776,669
12	Italy	4,627,699	+4,552	130,233	+50	4,383,195	+6,549	114,271	525	76,676	2,158	88,800,986	1,471,327	60,354,334
13	Indonesia	4,185,144	+3,835	140,138	+219	3,976,064	+7,912	68,942		15,108	506	35,945,697	129,761	277,014,855
14	Germany	4,139,009	+9,871	93,517	+63	3,873,700	+10,700	171,792	1,217	49,211	1,112	70,379,237	836,776	84,107,665
15	Mexico	3,549,229	+7,040	270,346	+434	2,897,667	+9,051	381,216	4,798	27,183	2,071	10,302,152	78,903	130,567,887
16	Poland	2,896,599	+652	75,473	+8	2,658,812	+176	162,314	101	76,637	1,997	20,373,472	539,031	37,796,497
17	South Africa	2,877,063	+3,648	85,952	+173	2,714,565	+8,272	76,546	546	47,783	1,428	17,280,650	287,002	60,210,991

COVID WORLDOMETER DATA SCRAPING

```
import requests
import pandas as pd
from bs4 import BeautifulSoup
```

```
url = 'https://www.worldometers.info/coronavirus/'

resp = requests.get(url)

soup = BeautifulSoup(resp.content,'html')
```

✓ 1.9s

```
df = pd.DataFrame(row,columns=col)
df
```

✓ 0.1s

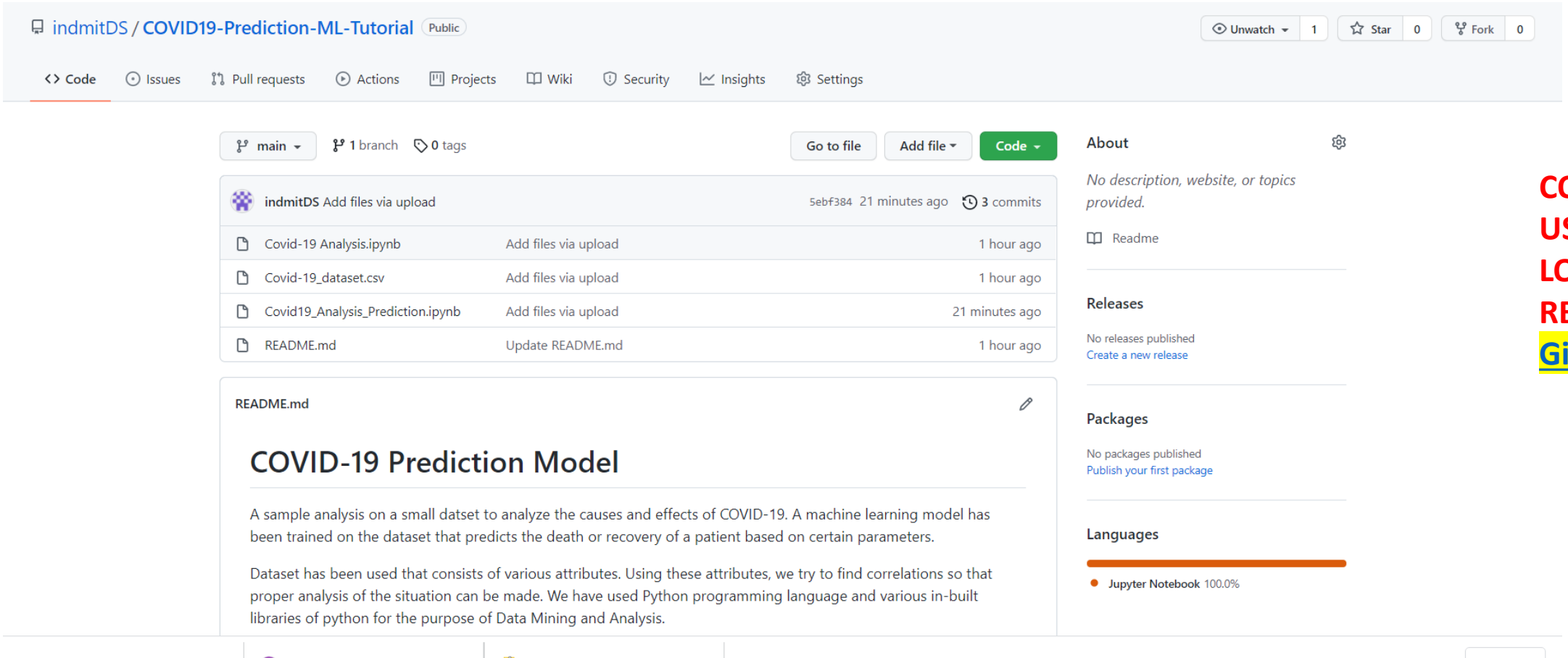
	Country,Other	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases
0	USA	42,736,585	+97,583	690,059	+1,303	32,391,956	+44,230	9,654,570
1	India	33,415,889	+35,367	444,563	+285	32,624,718	+33,850	346,608
2	Brazil	21,069,017	None	589,277	None	20,173,064	None	306,676
3	UK	7,371,301	+32,651	134,983	+178	5,934,018	+26,989	1,302,300
4	Russia	7,234,425	+19,905	196,626	+791	6,469,017	+16,619	568,782
...	...	...	...	...	...	...	...	...
218	Marshall Islands	4	None	None	None	4	None	0
219	Samoa	3	None	None	None	3	None	0
220	Saint Helena	2	None	None	None	2	None	0
221	Micronesia	1				1		0
222	China	95,577	+84	4,636		90,025	+45	916

OUTLINE

- ~~Data Science—A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- ~~Frameworks (Python & SQL) (Python Data Structures)~~
- ~~Environment Setup and Get Started~~
- ~~COVID Data Scraping from WEB~~
- COVID Case Study Using Python
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

COVID CASE STUDY



COVID ANALYSIS USING LOGISTIC REGRESSION:
Github link

TINY DATA SAMPLE

```
data.shape
✓ 0.3s                                                    Python
(1085, 20)
```

```
data.columns
✓ 0.7s
Index(['reporting_date', 'location', 'country', 'gender', 'age',
       'visiting_wuhan', 'from_wuhan', 'death', 'recovered'],
      dtype='object')
```

```
data.dtypes
✓ 0.5s
reporting_date    datetime64[ns]
location                  object
country                   object
gender                    object
age                      float64
visiting_wuhan             int64
from_wuhan                 int64
death                      int64
recovered                  int64
```

```
data.info()
✓ 0.5s
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1085 entries, 0 to 1084
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   reporting_date  1084 non-null   datetime64[ns]
 1   location        1085 non-null   object
 2   country         1085 non-null   object
 3   gender          902 non-null    object
 4   age             843 non-null    float64
 5   visiting_wuhan  1085 non-null   int64
 6   from_wuhan      1085 non-null   int64
 7   death           1085 non-null   int64
 8   recovered       1085 non-null   int64
```

```
int('Number of Null values in Columns')
ata.isnull().sum()
0.6s
ber of Null values in Columns

orting_date      1
ation            0
ntry             0
gender         183
age            242
visiting_wuhan   0
from_wuhan       0
death            0
recovered        0
```

ENCODING CATEGORICAL DATA

```
data.head()
```

✓ 0.1s Python

	reporting_date	location	country	gender	age	visiting_wuhan	from_wuhan	death	recovered
0	2020-02-10	Vinh Phuc	Vietnam	NaN	0.25	0	0	0	1
1	2020-02-05	Singapore	Singapore	male	0.50	0	0	0	1
2	2020-02-17	Singapore	Singapore	male	1.00	0	0	0	1
3	2020-01-25	Hechi, Guangxi	China	female	2.00	1	0	0	0
4	2020-01-25	Johor	Malaysia	male	2.00	0	0	0	1

```
labelencoder = LabelEncoder()
refined_data[refined_data.columns[3]] = labelencoder.fit_transform(refined_data[refined_data.columns[3]])
```

✓ 0.4s Python

```
refined_data.sample(5)
```

✓ 0.3s Python

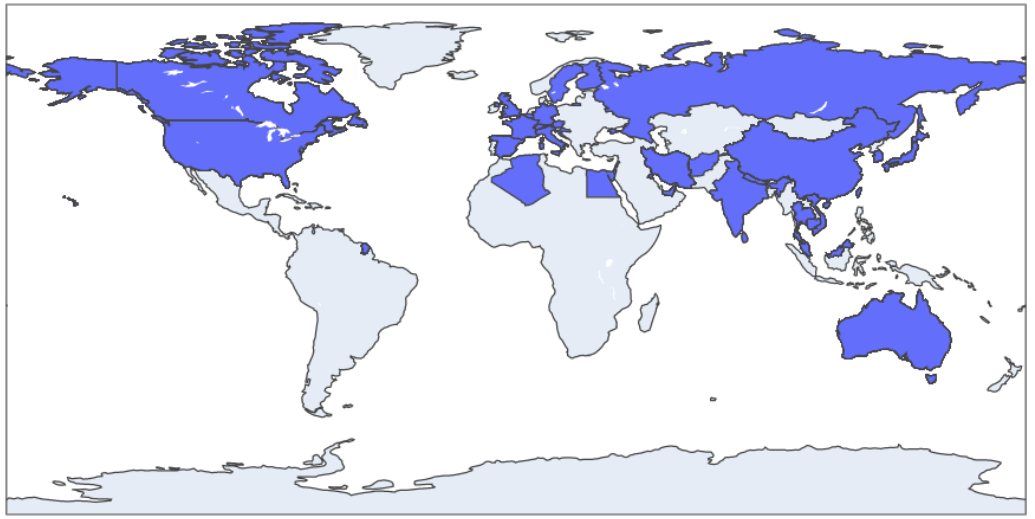
	reporting_date	location	country	gender	age	visiting_wuhan	from_wuhan	death	recovered
259	2020-02-02	59	13	0	38.0	0	1	0	1
136	2020-02-13	88	14	1	30.0	0	0	0	1
1027	2020-01-22	109	3	0	48.0	0	1	1	0
634	2020-02-26	65	9	1	65.0	0	0	0	0
733	2020-02-25	65	9	0	75.0	0	0	0	0

PATIENT GEO SPATIAL DISTRIBUTION

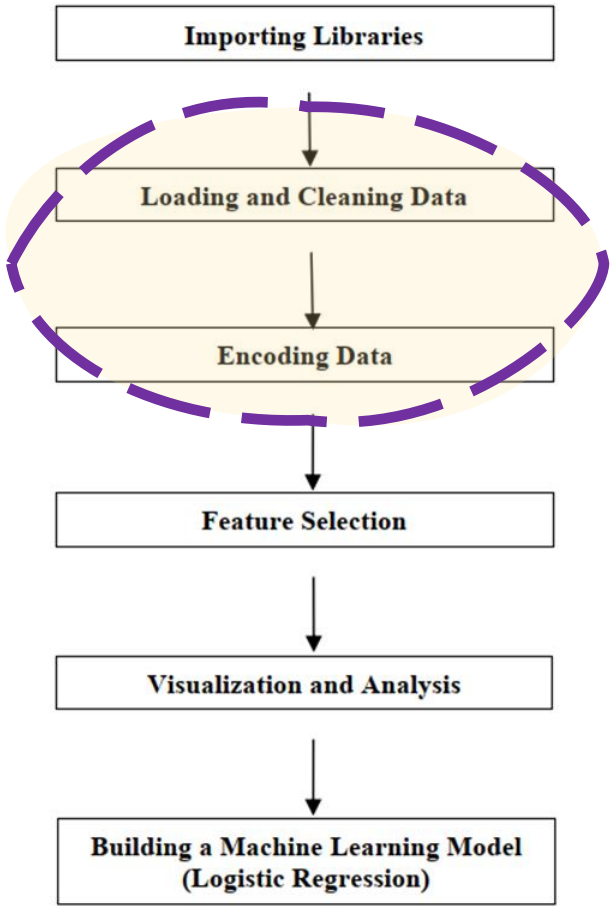
```
# WORLD MAP SHOWING LOCATIONS WITH COVID-19 PATIENTS
import plotly.io as pio
pio.renderers.default = "vscode"
import plotly.express as px
fig = px.choropleth(data, locations="country", locationmode='country names',
                    hover_name="country", title='PATIENTS IDENTIFIED AT DIFFERENT LOCATIONS',
                    color_continuous_scale=px.colors.sequential.Magenta)
fig.update(layout_coloraxis_showscale=True)
#####fig.update_geos(fitbounds='locations',visible=True)
fig.show(renderer='notebook')
```

✓ 0.3s

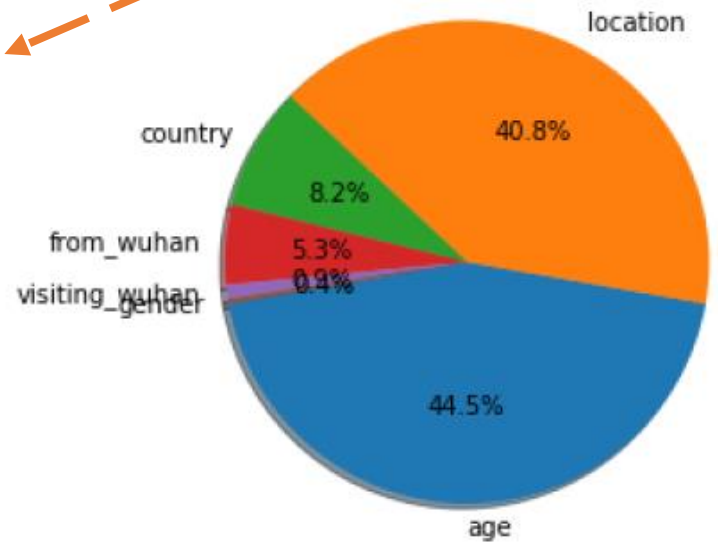
PATIENTS IDENTIFIED AT DIFFERENT LOCATIONS



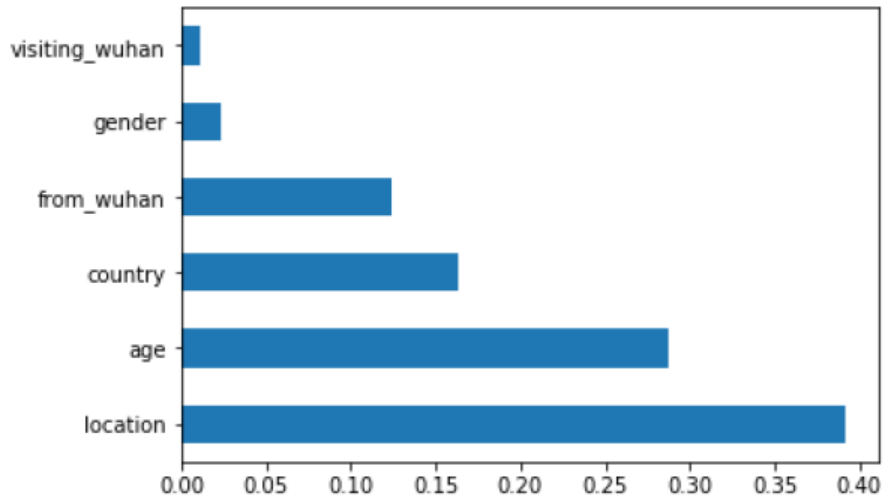
PIPELINE



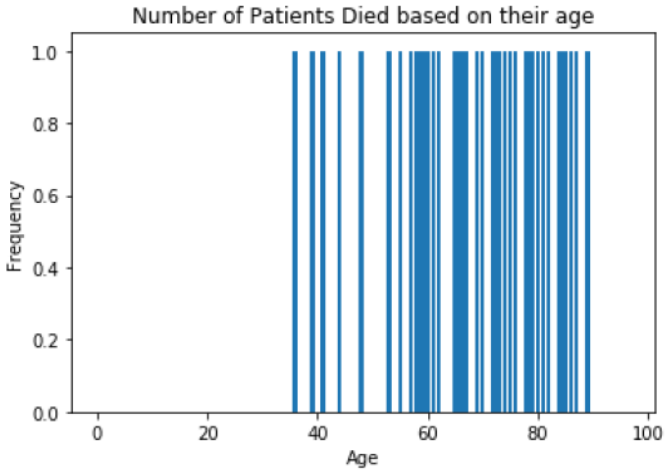
DATA CLEANING

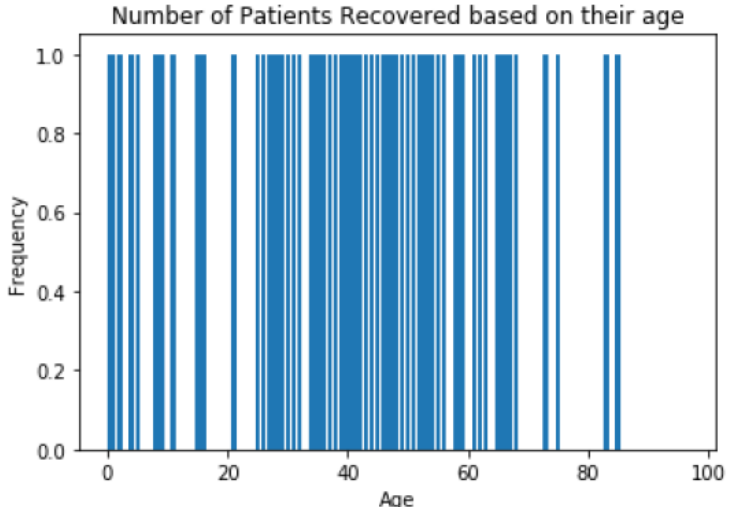


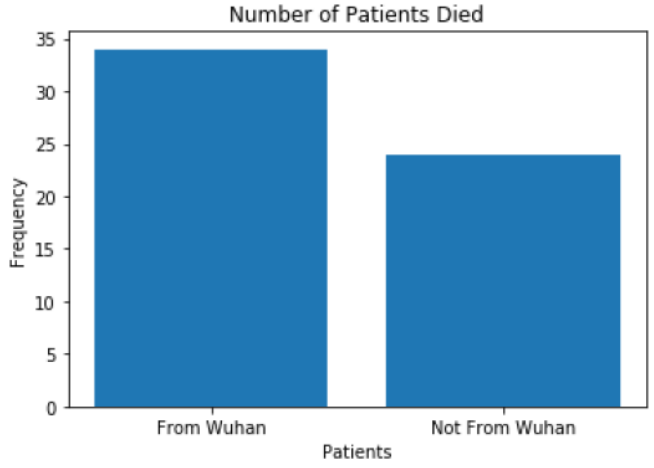
FEATURE SELECTION

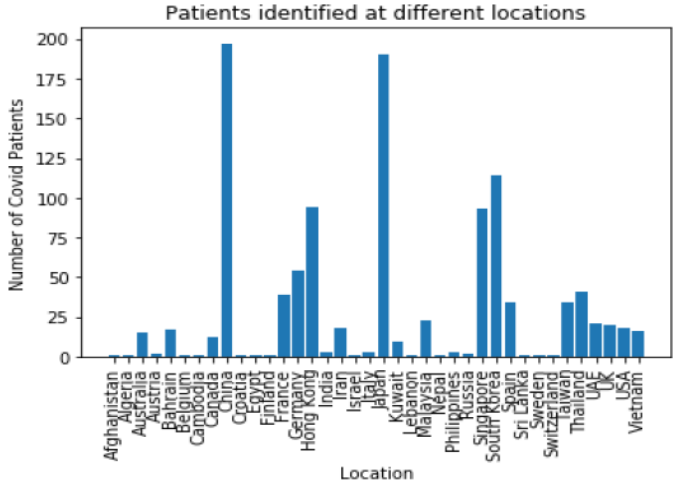


VISUALIZATION









Logistic Regression:

- Accuracy of the model predicting death is – 94.7%
- Accuracy of the model predicting recovery is – 81.1%

PREDICTION USING LOGISTIC REGRESSION

Predicting DEATH of a Patient

```
X = refined_data[refined_data.columns[1:7]] #(location, country, gender, age, visiting wuhan, from wuhan)
y = refined_data[refined_data.columns[[7]]] #death
```
✓ 0.1s

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

reg=LogisticRegression()
reg.fit(X_train,y_train)
```
✓ 0.7s

```
LogisticRegression()
```

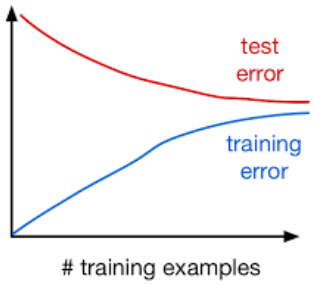
```
reg.score(X_train,y_train)
```
✓ 0.5s

```
0.953030303030303
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pdt))
```
✓ 0.7s

	precision	recall	f1-score	support
0	0.97	0.99	0.98	154
1	0.75	0.55	0.63	11
accuracy			0.96	165
macro avg	0.86	0.77	0.80	165
weighted avg	0.95	0.96	0.95	165

- Accuracy of the model predicting death is : 95%



Predicting RECOVERY of a Patient

```
X = refined_data[refined_data.columns[1:7]] #(location, country, gender, age, visiting wuhan, from wuhan)
y = refined_data[refined_data.columns[[8]]] #recovered
```
✓ 0.5s

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)

reg=LogisticRegression()
reg.fit(X_train,y_train)
```
✓ 0.1s

```
LogisticRegression()
```

```
reg.score(X_train,y_train)
```
✓ 0.5s

```
0.8166666666666667
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pdt))
```
✓ 0.3s

	precision	recall	f1-score	support
0	0.86	0.97	0.91	139
1	0.50	0.15	0.24	26
accuracy			0.84	165
macro avg	0.68	0.56	0.57	165
weighted avg	0.80	0.84	0.81	165

- Accuracy of the TRAINING model predicting recovery is: 81.1%

OUTLINE

- ~~Data Science – A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- ~~Frameworks (Python & SQL) (Python Data Structures)~~
- ~~Environment Setup and Get Started~~
- ~~COVID Data Scraping from WEB~~
- ~~COVID Case Study Using Python~~
- A Possible Outline of Topics to be Introduced Into the Course Structure
- New Directions*

*BACKUP POSTED AT THE END FOR DATA STRUCTURES

POSSIBLE ROUTES

- THEORETICAL ML & ALGORITHMS
- APPLIED ML
- DATA SCIENCE ANALYTICS
- STATISTICS
- DATA ENGINEERING
- BUSINESS ANALYTICS

A REFLECTIVE CURRICULUM

CORE:

- **LINEAR ALGEBRA/STATISTICS/OPTIMIZATION**
- **PYTHON FOUNDATIONS**
- **DATABASES**
- **DATA VISUALIZATION**
- **SUPERVISED LEARNING**
- **SUPERVISED LEARNING CLASSIFICATION**
- **ENSEMBLE TECHNIQUES**
- **MODEL TUNING**
- **UNSUPERVISED LEARNING**

ELECTIVES:

- TIME SERIES FORECASTING
- RECOMMENDER SYSTEMS
- REINFORCEMENT LEARNING
- DEEP LEARNING WITH TF/KERAS
- NATURAL LANGUAGE PROCESSING
- ADVANCED DEEP LEARNING & COMPUTER VISION
- CLOUD ANALYTICS
- TIME SERIES FORECASTING

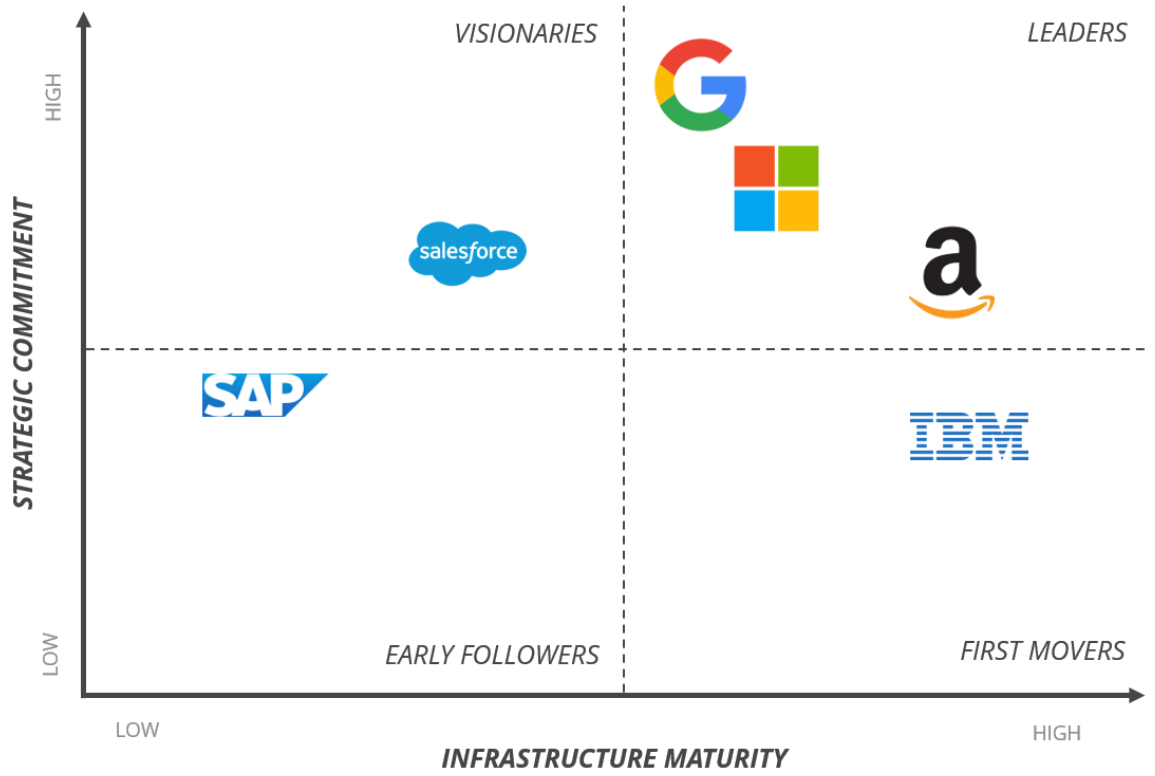
OUTLINE

- ~~Data Science — A Brief Overview~~
- ~~ML and Deep learning, the main driver of Modern Data Science~~
- ~~Frameworks (Python & SQL) (Python Data Structures)~~
- ~~Environment Setup and Get Started~~
- ~~COVID Data Scraping from WEB~~
- ~~COVID Case Study Using Python~~
- ~~A Possible Outline of Topics to be Introduced Into the Course Structure~~
- New Directions*

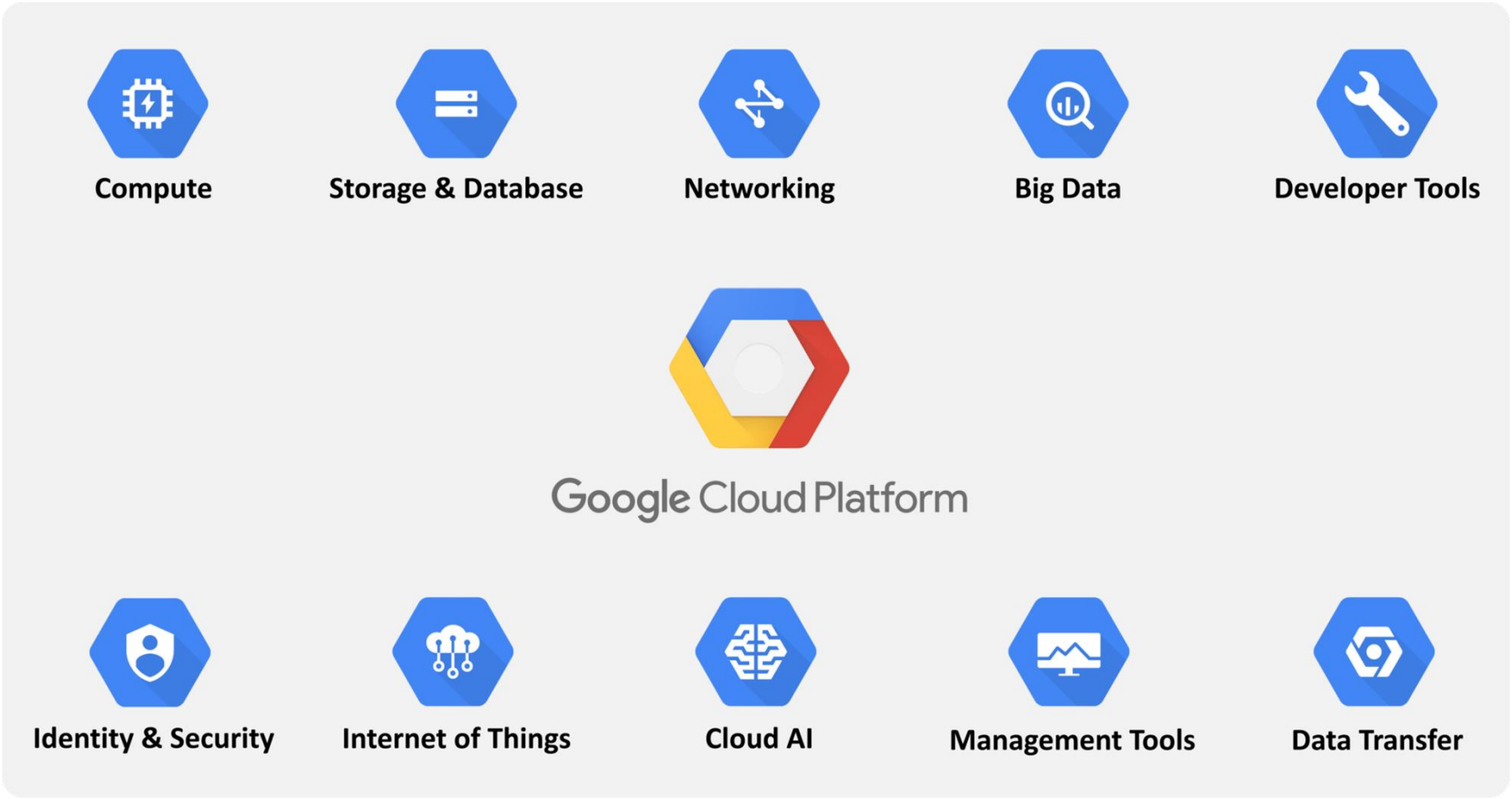
*BACKUP POSTED AT THE END FOR DATA STRUCTURES

MOVING TO CLOUD

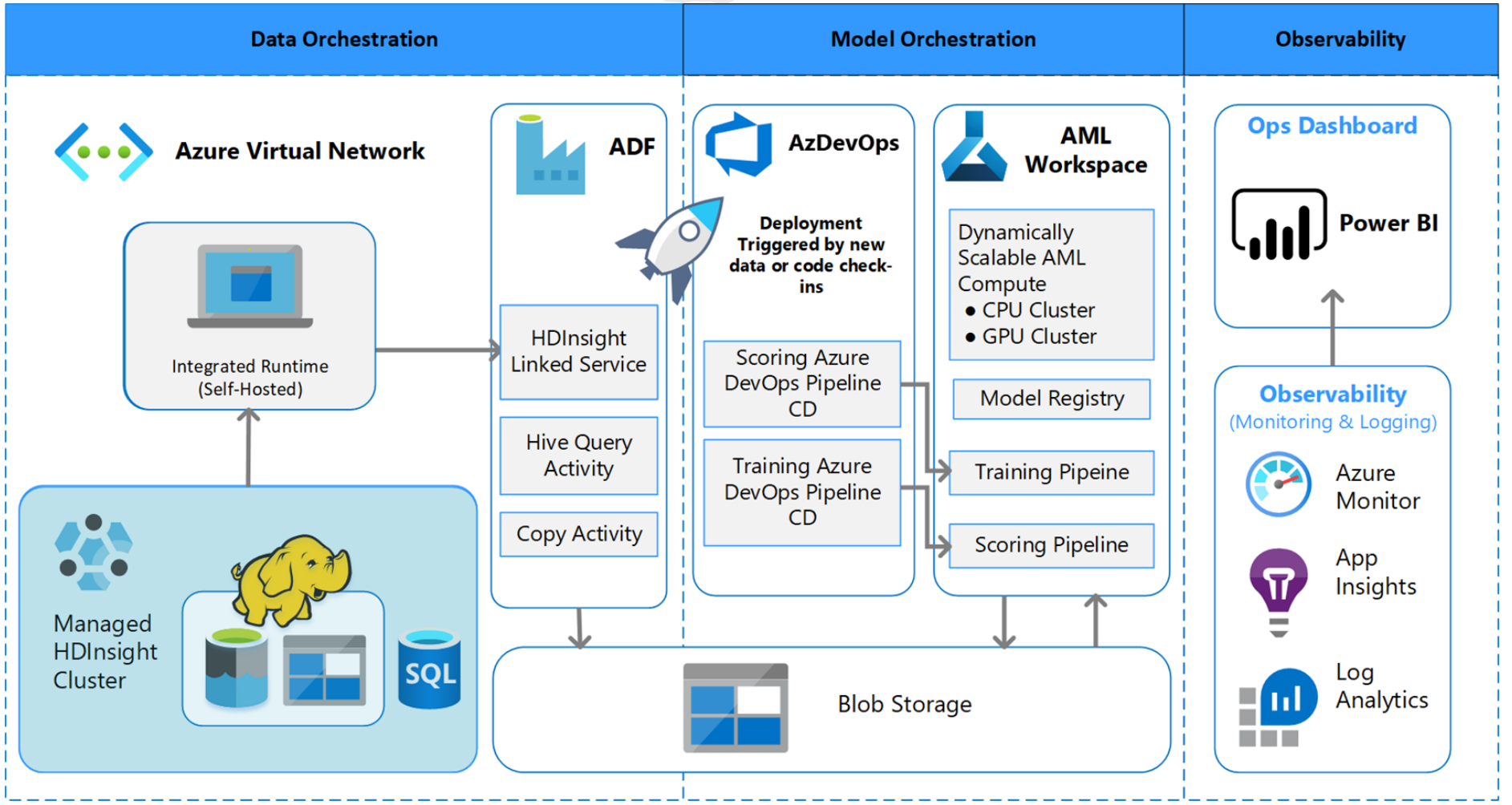
AI LANDSCAPE: CLOUD PLATFORM VENDORS



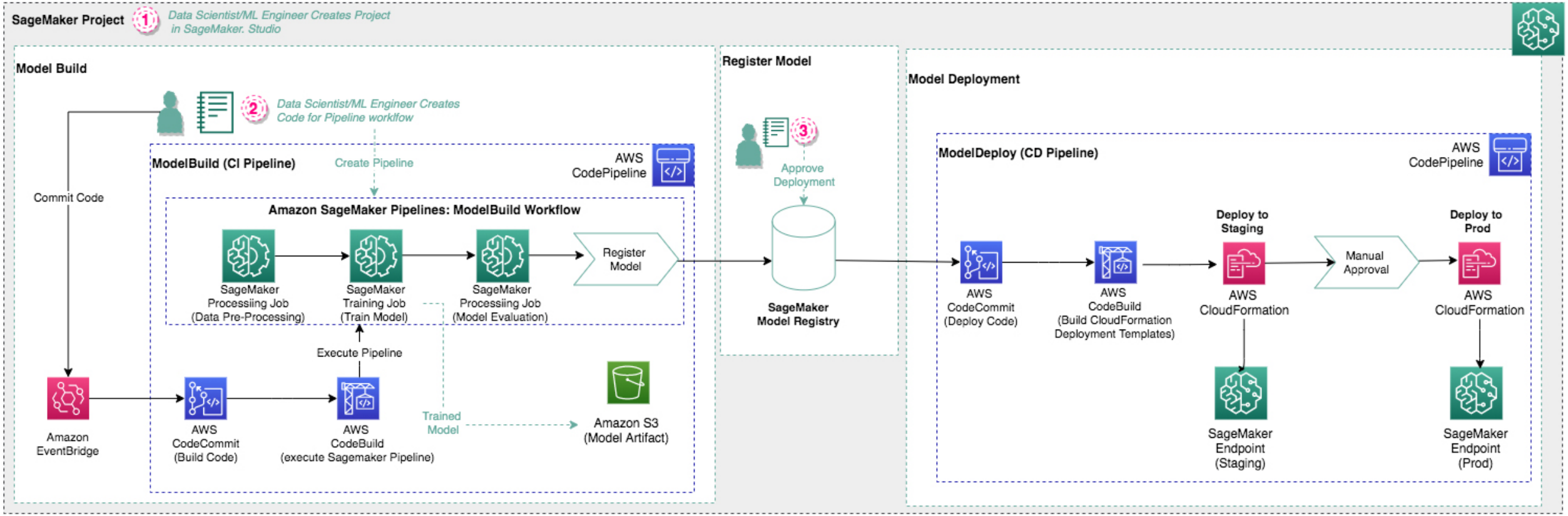
GOOGLE CLOUD



AZURE CLOUD


Data Orchestration
Model Orchestration
Observability
Azure Virtual Network
ADF
AzDevOps
AML Workspace
Ops Dashboard
Power BI
Deployment Triggered by new data or code check-ins
Integrated Runtime (Self-Hosted)
HDInsight Linked Service
Hive Query Activity
Copy Activity
Scoring Azure DevOps Pipeline CD
Training Azure DevOps Pipeline CD
Dynamically Scalable AML Compute
• CPU Cluster
• GPU Cluster
Model Registry
Training Pipeine
Scoring Pipeline
Observability
(Monitoring & Logging)
Azure Monitor
App Insights
Log Analytics
Managed HDInsight Cluster
SQL
Blob Storage

AWS CLOUD


SageMaker Project
1
Data Scientist/ML Engineer Creates Project in SageMaker. Studio
Model Build
2
Data Scientist/ML Engineer Creates Code for Pipeline workflow
ModelBuild (CI Pipeline)
Create Pipeline
AWS CodePipeline
Commit Code
Amazon SageMaker Pipelines: ModelBuild Workflow
SageMaker Processing Job (Data Pre-Processing)
SageMaker Training Job (Train Model)
SageMaker Processing Job (Model Evaluation)
Register Model
Execute Pipeline
Amazon EventBridge
AWS CodeCommit (Build Code)
AWS CodeBuild (execute Sagemaker Pipeline)
Trained Model
Amazon S3 (Model Artifact)
Register Model
3
Approve Deployment
SageMaker Model Registry
Model Deployment
ModelDeploy (CD Pipeline)
AWS CodePipeline
AWS CodeCommit (Deploy Code)
AWS CodeBuild (Build CloudFormation Deployment Templates)
Deploy to Staging
AWS CloudFormation
Manual Approval
Deploy to Prod
AWS CloudFormation
SageMaker Endpoint (Staging)
SageMaker Endpoint (Prod)

DEEP DIVES

- DETAIL LOOK INTO CS ALGORITHMS
- DATA STRUCTURE FOUNDATIONS
- ML AGORITHMS (SVM/NN/DECISION TREE ETC)

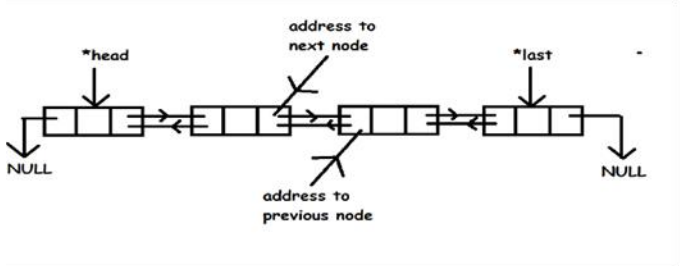

thank you

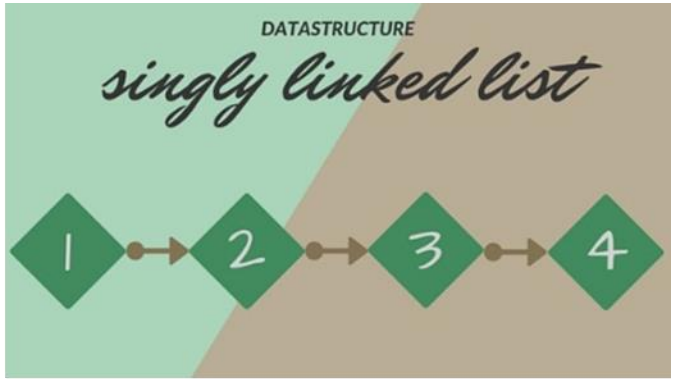
BACKUP

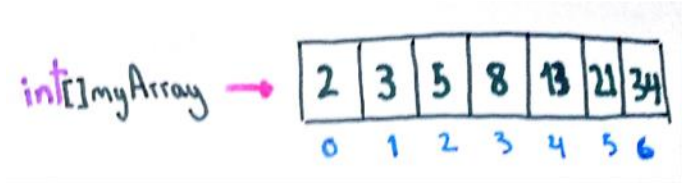
ABSTRACT DATA STRUCTURES

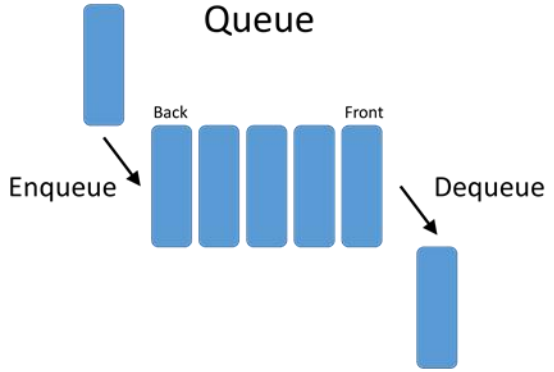
Data Structures

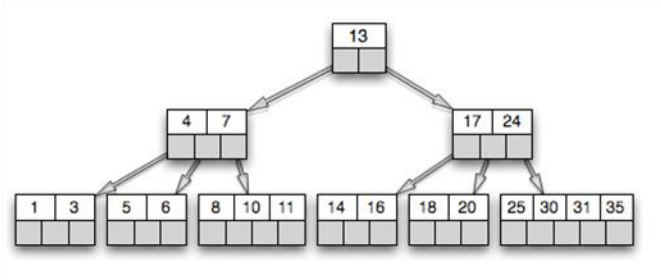
- A *Data Structure* is:
  - "An organization of information, usually in computer memory", for better algorithm efficiency."











Core Operations

- Data Structures have three core operations
  - a way to add things
  - a way to remove things
  - a way to access things (without modifying the data)
- Details depend on the data structure
  - For instance, a *List* may need to:
    - add at the end
    - access by location (index)
    - remove by location (index)
- More operations added depending on what data structure is designed to do

Implementation-Dependent Data Structures

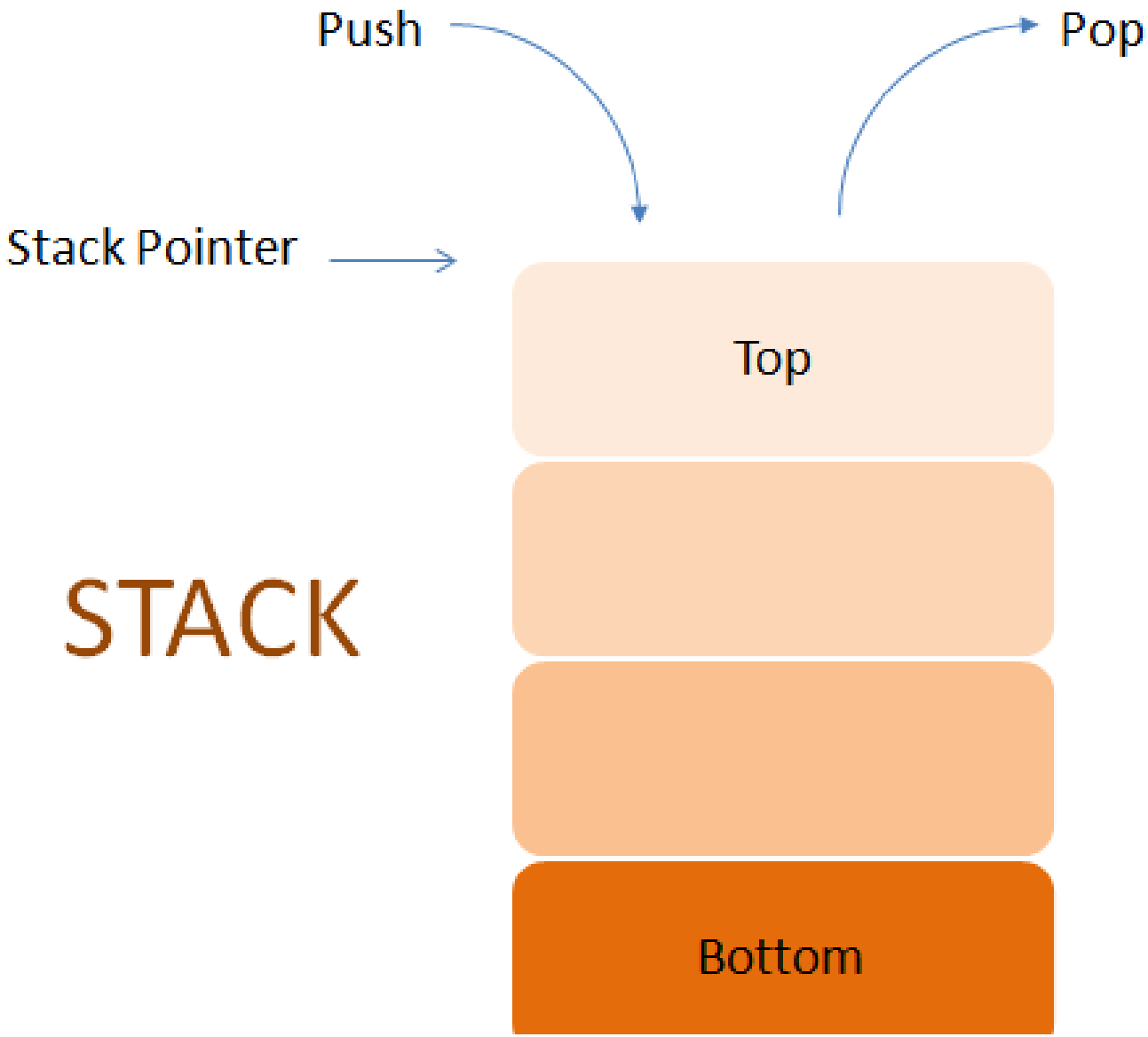
- Arrays
  - Collection of objects stored contiguously in memory
  - Accessed through an index
- Linked Structures
  - Collection of node objects
    - Store data and reference to one or more other nodes
  - Linked Lists
    - Linear collection of nodes
    - Single-linked List – nodes contain references to next node in list
    - Double-Linked List – nodes contain reference to next and previous nodes in list
  - Trees
    - Hierarchical structure
    - Nodes reference two or more “children”

Implementation-Independent Data Structures

- Abstract Data Types (ADTs)
  - Descriptions of how a data type will work without implementation details
  - Description can be a formal, mathematical description
- Examples:
  - Bag, Set, Stack, Queue, List

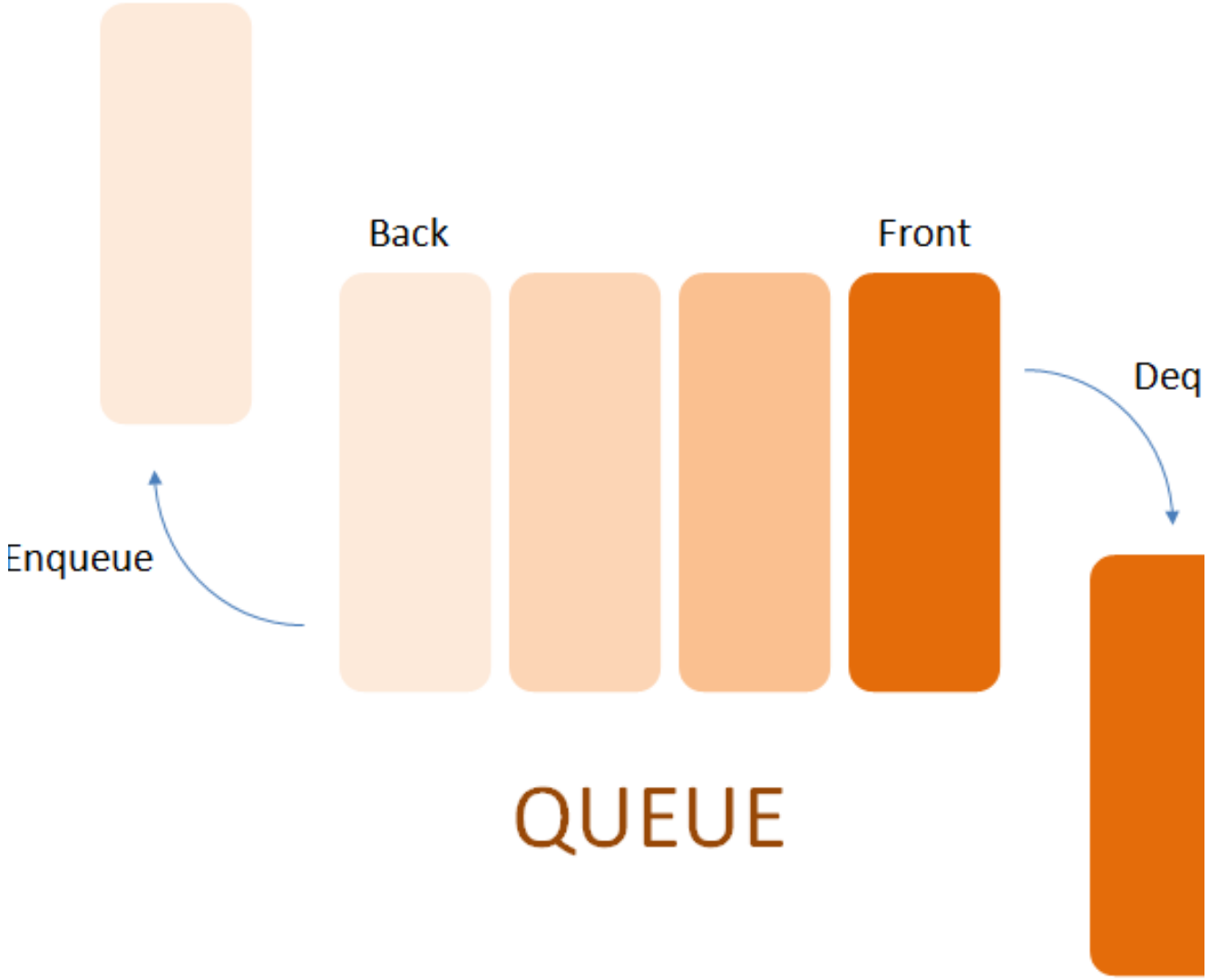
Stacks

- Only access last item inserted
  - Expected behavior: Last in, First out (LIFO)
    - push (put object on top)
    - pop (remove object from top)
    - peek / top (look at object on top)
  - Other useful operations
    - make empty
    - size
    - is empty?



Queues

- Only access item that has been there the longest
  - Expected behavior: First in, First out (FIFO)
    - enqueue (insert at the back)
    - dequeue (remove from the front)
    - front (look at the object at the front)
  - Other useful operations
    - make empty
    - size
    - is empty?

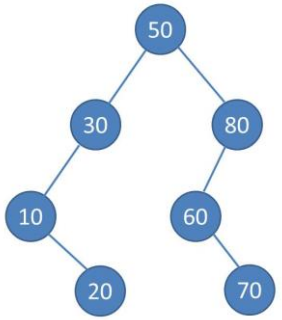


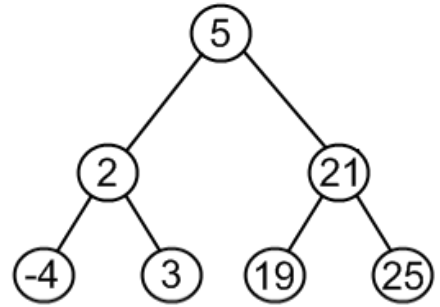
Lists

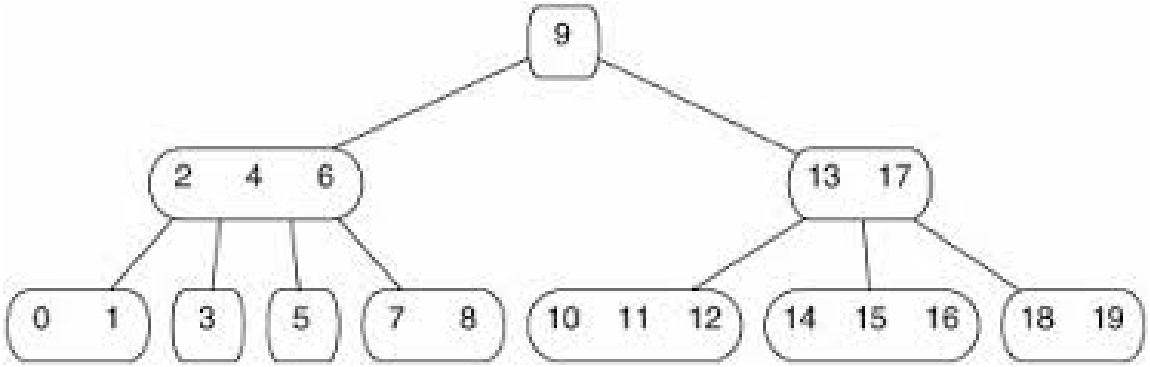
- Linear collection of items
  - Sorted List
    - Items in list are arranged in a pre-determined ordering
      - For instance, alphabetically or by ascending numerical value
  - Indexed List
    - Items are accessed by position in list
    - Additions / deletions can also be done by position
  - Unsorted List
    - Items are stored without an implicit ordering

Types of Trees

- Binary Search Trees (BSTs)
  - Items stored in sorted order
- Heaps
  - Items stored according to the “Heap Property”
- AVL and Red-Black Trees
  - BSTs that stay balanced
- Splay Trees
  - BST with most recently items at top
- B-Trees
  - Another variation of BST
  - Nodes can have more than two children







Other ADTs

- Hash Tables
  - Hash function:
    - computes an *index* into an array of *buckets* or *slots*
    - look in the bucket for the desired value
- Maps
  - Collection of items with a key and associated values
  - Similar to hash tables
- Graphs
  - Nodes with unlimited connections between other nodes
- Sparse vectors and sparse matrices

PYTHON DATA STRUCTURES

Lists

- An ordered group of items
- Does not need to be the same type
  - Could put numbers, strings or donkeys in the same list
- List notation
  - A = [1,”This is a list”, c, Donkey(“kong”)]

Methods of Lists

- List.append(x)
  - adds an item to the end of the list
- List.extend(L)
  - Extend the list by appending all in the given list L
- List.insert(I,x)
  - Inserts an item at index I
- List.remove(x)
  - Removes the first item from the list whose value is x

Examples of other methods

- a = [66.25, 333, 333, 1, 1234.5] //Defines List
  - **print** a.count(333), a.count(66.25), a.count('x') //calls method
  - 2 1 0 //output
- a.index(333)
  - //Returns the first index where the given value appears
  - 1 //ouput
- a.reverse() //Reverses order of list
  - a //Prints list a
  - [333, 1234.5, 1, 333, -1, 66.25] //Ouput
- a.sort()
  - a //Prints list a
  - [-1, 1, 66.25, 333, 333, 1234.5] //Output

Using Lists as Stacks

- The last element added is the first element retrieved
- To add an item to the stack, append() must be used
  - stack = [3, 4, 5]
  - stack.append(6)
  - Stack is now [3, 4, 5, 6]

- To retrieve an item from the top of the stack, pop must be used
  - Stack.pop()
  - 6 is output
  - Stack is now [3, 4, 5] again

Using Lists as Queues

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

- First element added is the first element retrieved
- To do this collections.deque must be implemented

List Programming Tools FILTER

- Filter(function, sequence)
  - Returns a sequence consisting of the items from the sequence for which function(item) is true

```
>>> def f(x): return x % 2 != 0 and x % 3 != 0
...
>>> filter(f, range(2, 25))
[5, 7, 11, 13, 17, 19, 23]
```

  - Computes primes up to 25

List Programming Tools
MAP

- Map(function, sequence)
  - Calls function(item) for each of the sequence's items

```
>>> def cube(x): return x*x*x
...
>>> map(cube, range(1, 11))
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

  - Computes the cube for the range of 1 to 11

List Programming Tools REDUCE

- Reduce(function, sequence)
  - Returns a single value constructed by calling the binary function (function)

```
>>> def add(x,y): return x+y
...
>>> reduce(add, range(1, 11))
55
```

  - Computes the sum of the numbers 1 to 10

del statement

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

A specific index or range can be deleted

Tuples

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

Tuple

- A number of values separated by commas
- Immutable
  - Cannot assign values to individual items of a tuple
  - However, tuples can contain mutable objects such as lists
- Single items must be defined using a comma
  - Singleton = ‘hello’,

Sets

An unordered collection with no duplicate elements

Basket = [‘apple’, ‘orange’, ‘apple’, ‘pear’]

Fruit = set(basket)

Fruit

- Set([‘orange’, ‘apple’, ‘pear’])

Dictionaries

- Indexed by keys
  - This can be any immutable type (strings, numbers…)
  - Tuples can be used if they contain only immutable objects

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Looping Techniques

Iteritems():

- for retrieving key and values through a dictionary

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.iteritems():
...     print k, v
...
gallahad the pure
robin the brave
```

Looping Techniques

Enumerate():

- for the position index and values in a sequence

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print i, v
...
0 tic
1 tac
2 toe
```

Looping Techniques

Zip():

- for looping over two or more sequences

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print 'What is your {0}?  It is {1}.'.format(q, a)
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

Comparisons

- Operators “in” and “not in” can be used to see if an item exists in a sequence
- Comparisons can be chained
  - a < b == c
    - This tests whether a is less than b and that b equals c