# DATA SCIENCE

Khirod Chandra Maharana

*Founder & Business Consultant, ERIXA.Lab*

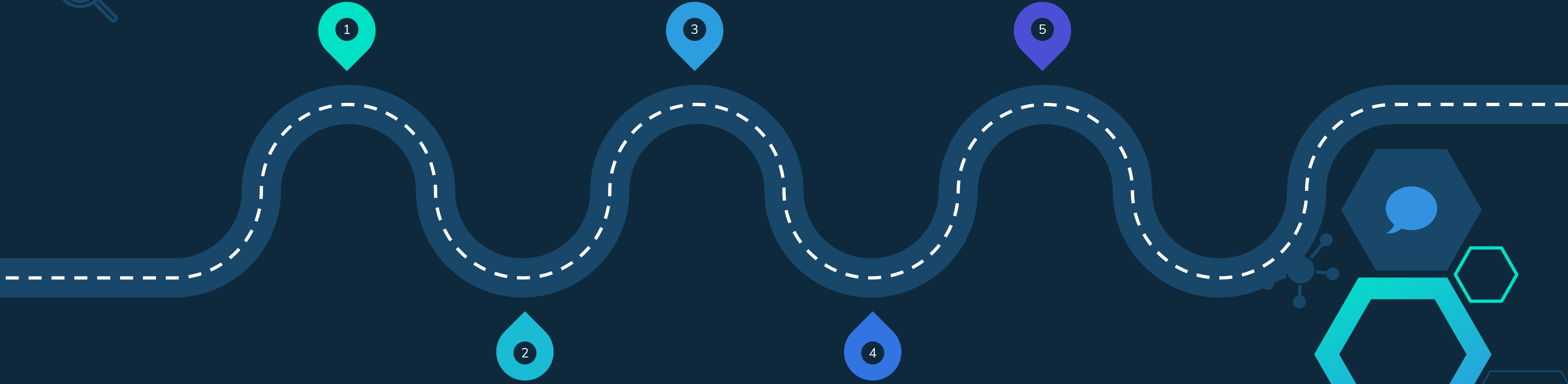
# Introduction

- Data = Collection of Facts
- Forms of Data = Text, Numbers, Images, Videos
- Data Back Then = Small and Structured
- Data Now : Huge and Unstructured
- Data Science Purpose = Solutions to Business Problems

# Types of Data

| | | | |
|---|---|---|---|
| **Categorical** | Ordered (Rank,Priorities) | True Binary (0,1 / Yes, No) | Nominal (Unordered) |
| **Numerical** | Discrete (Calendar Days) | Continuous (Height of Persons) | |

# Data Science Life-Cycle



Data Acquisition

Data Preprocessing

ML Algorithm

Pattern Evaluation

Knowledge Representation

- Data Acquisition :
  - Data from Multiple Sources
  - Data Storage
  - Target Data
- Data Preprocessing : (EDA)
  - Data Manipulation
  - Data Visualisation
- Machine Learning :
  - **Algorithm :** Regression, Classification, Clustering
- Pattern Evaluation :
  - Once the data mining technique have been applied the results have to be evaluated.
- Knowledge Representation :
  - The identified pattern must be represented using simple and aesthetic graphs

# Installing Python

1. **Step 1:** Select Version of Python to Install.
2. **Step 2:** Download Python Executable Installer.
3. **Step 3:** Run Executable Installer.
4. **Step 4:** Verify Python Was Installed On Windows.
5. **Step 5:** Verify Pip Was Installed.
6. **Step 6:** Add Python Path to Environment Variables (Optional)
7. **Step 7:** Install virtual (Optional)

# Python Variables and Data Types

- Data / Values can be stored in temporary storage spaces called variable.
- Storing Variable  Student = “Ram”
- Every Variable is associated with a data type
  - Integer : 50,100
  - Float : 3.14, 5.67
  - Boolean : True / False (Binary Value)
  - String : “ Sam”, “ Ram”
  - Complex : 3 + 4i, 5-7i

# Python Codes

```
a = 10
print(a)
Output = 10
type(a)
Output = int
```

```
a = 3.5
print(a)
Output = 3.5
type(a)
Output = float
```

```
a = True
print(a)
Output = True
type(a)
Output = bool
```

```
a = “ Hello World ”
print(a)
Output = Hello World
type(a)
Output = str
```

```
a = 3 + 4j
print(a)
Output = 3+4j
type(a)
Output = complex
```

# Operators in Python

## RELATIONAL

Greater Than >
Less Than <
Equal to ==
Not Equal to !=

## ARITHMETIC

Add +
Substract, -
Multiply *
Divide

## LOGICAL

And = &
Or = I

# Python Tokens (KILO)

- Keywords (33) : Special Reserved Words
- Identifiers : Names used for variables
  - No special characters except underscore
  - Identifiers are Case Sensitive
  - First Letter cannot be a digit
- Literals : Constant in Python (Can’t Change)
- Operators : Relational, Arithmetic, Logical

| KEYWORDS | | |
|---|---|---|
| .and | .except | .nonlocal |
| .as | .False | .not |
| .assert | .finally | .or |
| .async | .for | .pass |
| .await | .from | .raise |
| .break | .global | .return |
| .class | .if | .True |
| .continue | .import | .try |
| .def | .in | .while |
| .del | .is | .with |
| .elif | .lambda | .yield |
| .else | .None | |

# Python Strings

```
#Extracting Individual Characters
my_strings = “ My Name is John ”
my_strings (0)
Output = ‘M’
```

```
#Extracting Last Character
my_strings = “ My Name is John ”
my_strings (-1)
Output = ‘n’
```

```
#Finding the length of a string
my_strings = “ My Name is John ”
len(my_strings)
Output = 15
```

```
#Converting String to Lowercase
my_strings = “ My Name is John ”
my_strings.lower ()
Output = ‘my name is john’
```

```
#Converting String to Uppercase
my_strings = “ My Name is John ”
my_strings.upper ()
Output = ‘MY NAME IS JOHN’
```

```
#Replace a Substring
my_strings = “ My Name is John ”
my_strings.replace (‘y’, ‘a’)
Output = ‘Ma Name is John’
```

```
#No of occurrence in Substring
my_strings = “ My Name is is John ”
my_strings.count (‘is’)
Output = 2
```

```
#Finding the index of substring
my_strings = “ My Name is John ”
my_strings.find (‘is’)
Output = 8
```

```
#Spliting the string
fruit = “ I like Apples, Mangoes, Lichi ”
fruit.split (‘ , ’)
Output = 8
```

# Data Structures in Python

**TUPLE**
Immutable Arrays

**LIST**
Mutable Arrays

**DICTIONARY**
Hash Tables

**SET**

# Tuple

DICTIONARY

- Unordered collection of (key + values) pairs enclosed with{ }
- Fruit={ “Apple”: 10, “Orange” : 20}
- Mutable

# List

- Lists are ordered collection of elements enclosed with [ ].
- Lists are mutable (can be modified)
- Can store heterogeneous values

# Dictionary

**Unordered collection of key value pairs enclosed with { }.**
**Mutable**
**Fruit = { “Apple”, 10, “Orange”, 20}**

### Extracting keys and Values

```
fruit= {apple:10, orange:20, banana:30,guava:40}
fruit.keys()
{apple, orange, banana,guava}
fruit.values()
{10, 20, 30,40}
```

### Adding a new element

```
fruit= {apple:10, orange:20, banana:30,guava:40}
fruit.[“mango”] = 50
{apple:10, orange:20,
banana:30,guava:40,mango:50}
```

### Popping an element

```
fruit= {apple:10, orange:20, banana:30,guava:40}
fruit.pop(“orange”)
{apple:10, banana:30,guava:40}
```

### Changing an existing element

```
fruit= {apple:10, orange:20, banana:30,guava:40}
fruit.[“apple”] = 100
{apple:100, orange:20, banana:30, guava:40,
mango:50}
```

### Updating dictionary’s elements with another

```
fruit 1= {apple:10, orange:20}
fruit 2={banana:30,guava:40}
fruit 1.update(“fruit2”)
{apple:100, orange:20, banana:30, guava:40,
mango:50}
```

# Set

- Set is an unordered and unindexed collection of elements enclosed with { }
- Duplicate are not allowed in set
- s1 = {1, 0, ”True”}

### Updating one dictionary’s elements.

```
s1= {1,a,True,2,b,False}
s1.add(“Hello”)
{1,2,False,Hello,’a’,’b’,True}
```

### Updating Multiple Elements

```
s1= {1, ‘a’ ,True, 2, ‘b’, False}
s1.update([ 10,20,30])
s1
{{1, ‘a’ ,10, True, 2, 20, ‘b’, False, 30}
{1,2,False,Hello,’a’,’b’,True}
```

### Removing an Element

```
s1= {1,a,True,2,b,False}
s1.remove (“b”)
{1,2,False,’a’,}
```

### Union of Two Sets (Append)

```
s1= {1,2,3}
s2={‘a’ ,’b’, ‘c’}
s1.union (s2)
{1,2,3,a,b,c}
```

### Intersection of two sets

```
s1= {1,2,3,4,5,6}
s2={5,6,7,8,9}
s1.intersection(s2)
{5,6}
```

# Flow Control Statements

**If it is raining, sit inside. Else go to play football.**

```
a=10
b=20
if b>a
        print(‘b is greater than a’)
if a>b
        print(‘b is greater than a’)
```

```
If (condition) {
        statement should be executed
        }
else {
        statement should be executed
```

```
a=10
b=20
c=30
if (a>b) and (a>c):
        print(‘a is greatest’)
elif (b>a) and (b>c):
        print (‘b is the greatest’)
else:
        print(‘c is the greatest’)
```

# Looping Statements

It is used to repeat a task multiple times. In a water bucket with a mug of  water while it is not full.

### While Loop

```
i=1
while i<=10
print (i)
1
2
3
4
5
6
7
8
9
10
```

### Nested For Loop

```
l1=[‘orange’, ‘black’, ‘white’,]
l2=[‘chair’, ‘book’, ‘laptop’]
for i in l1
for i in l2
print (i, j)
orange chair
orange book
orange laptop
back chair
black book
black laptop
white chair
white book
white laptop
```

### For Loop

```
l1=[‘mango’, ‘apple’, ‘grape’, ‘orange’]
for i = l1
print (i)
mango
apple
grapes
orange
```

# Functions

Functions in real life : Eating, Running, Cycling

Python Functions : Functions is a block of code which performs a specific task.

```
# To Check a Number Even or Odd
def odd_even (x) :
        If x / 2 = = 0
        print (x, is even)
Else
        Print (x, is odd)
```

```
# Lambda with Map
l1 = [1,2,3,4,5,6,7,8]
k = list(map[lambda x : x *2, l1])
print(k)
```

```
#def = define a function
def hello () :
        print (‘ Hello World”)
hello ()
Out : ‘Hello World’
```

```
# Lambda Function
g = lambda x: x*x
g(7)
343
```

```
# Lambda with Reduce
from functools import reduce
sum=reduce(lambda,x,y: x+y,l1)
sum = 36
```

```
def add_10(x):
        return x +10
IN : add_10(9)
Out : 19
```

```
# Lambda with Filter
l1 = [87,56,90,34,2,5,7,1,7,98]
K = list(filter[lambda x : x /2 !=0, l1])
```

# Object Oriented Programming

# Inheritance in Python

# Libraries in Python

Pandas
PyTorch
NumPy
plotly
matplotlib
theano
TensorFlow
scikit learn
K Keras
python™
Natural Language Analyses
with NLTK

# Numpy (Numerical Python)

- It is the core library for numerical and scientific computing.
- It consists of multidimensional array objects and a collection of routines for processing those arrays.

Single Dimensional Array

```
import numpy as np
n1=np.array([10,20,30,40])
print(n1)
array([10,20,30,40])
```

Multidimensional Array

```
import numpy as np
n1=np.array([10,20,30,40],[40,30,20,10])
print(n1)
array ([10,20,30,40],
[40,30,20,10] )
```

Initialising numpy array with random

```
import numpy as np
n1=np.random.randint(1,100,5)
print(n1)
array (95,88,26,22,76)
```

Initialising numpy array with zeros

```
import numpy as np
n1=np.zeros((1,2))
print(n1)
array([0,0,])
#1 row , 2 columns
```

Initialising numpy array with a range

```
import numpy as np
n1=np.arange(10,20)
print(n1)
array (10,11,12,13,14,15,16,17,18,19)
```

# Numpy (Numerical Python)

### Numpy Intersection and Difference

```
import numpy as np
n1=np.array([10,20,30,40,50,60])
n2=np.array)[50,60,70,80,90])
np.intersect1d((n1,n2))
array([50,60])

np.set difference 1d((n1,n2))
array([10,20,30,40])

np.set difference 1d((n2,n1))
array([70,80,90])
```

### Joining Numpy Array

```
import numpy as np
n1=np.array([10,20,30])
n2=np.array)[40,50,60])
np.vstack((n1,n2)) #vertical
np.hstack((n1,n2)) #horizontal
np.coloumn_stack((n1,n2)) #coloumnwise
```

### Addition of Numpy Arrays

```
import numpy as np
n1=np.array([10,20])
n2=np.array)[30,40])
np.sum(n1,n2)
100
np.sum([n1,n2],axis=0)
array([40,60])
np.sum([n1,n2],axis=1)
array([30,70])
```

### Basic Statistics

```
import numpy as np
n1=np.array([10,20,30,40,50,60])
np.mean(n1)
35.0
np.median(n1)
55.5
np.std(n1)
36.59
```

### Basic Mathematics

```
import numpy as np
n1=np.array([10,20,30])
n1=n1+1
array([11,21,31])
n1=n1-1
array([9,19,29])
n1=n1*2
array([20,40,60])
n1=n1/2
array([5,10,15])
```

# Matplotlib

## Line Chart

```
import numpy as np
From matplotlib import pyplot as plt
x=np.arange(1,11)
array([1,2,3,4,5,6,7,8,9,10])
y=2*x
array([2,4,6,8,10,12,14,16,18,20])
plt.plot(x,y)
plt.title(“Line Plot”)
plt.xlabel(“x-label”)
plt.ylabel(“y-label”)
pl.show()
```

## Changing Line Aesthetics

```
plt.plot(x,y,color=”g”, linestyle=”:”, linewidth =2)
plt.show()
```

## Bar Plot

```
import numpy as np
From matplotlib import pyplot as plt
student = {“Bob”: 87, “Matt” :56, “Sam” : 27
names = list (student.keys () )
names = list (student.values () )
plt.bar(names,values)
pl.show()
```

## Horizontal Bar Plot

```
plt.barh(names, values, color-’g’)
plt.title(“Bar PLot”)
plt.xlabel(“Names”)
plt.ylabel(“Marks”)
plt.grid(True)
plt.show()
```

## Adding Subplots

```
x=np.arange(1,11)
y1=2*x
y2=3*x
plt.subplot(1,2,1)
plt.plot(x,y1,color=’g’, linestyle=’:’,linewidth=2)
plt.plot(x,y2,color=’g’,
linestyle=’:’,linewidth=2)
pl.show()
```

## Boxplot

```
one=[1,2,3,4,5,6,7,8,9]
two=[1,2,3,4,5,4,3,2,1]
Three[6,7,8,9,8,7,6,5,4]
data=list([one,two,three])
plot.boxplot(data)
plt.show()

plot.violinplot(data)
plt.show()
```

## Pie Chart

```
fruit=[‘apple’, ‘orange’, ‘guava’]
quantity=[50,60,70]
plot.pie(quantity,labels=fruit))
plt.show()
```

# MACHINE LEARNING

# Machine Learning

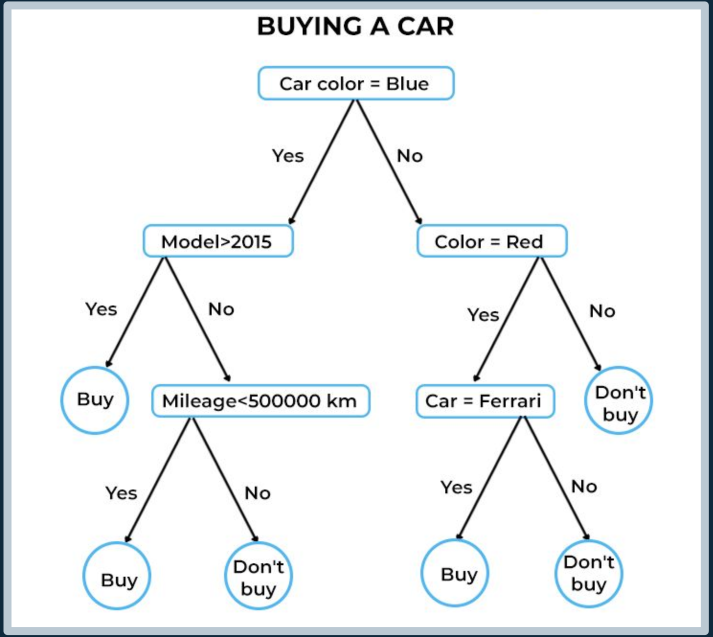
- **Supervised Learning**
  - *Task Driven*
  - *Predict Next Value*
  - *House Price Prediction*
  - *Medical Imaging*
- **Unsupervised Learning**
  - *Data Driven*
  - *Identify Cluster*
  - *Customer Segmentation*
  - *Market Basket Analysis*
- **Reinforcement Learning**
  - *Learn from Mistakes*
  - *Optimised Marketing*
  - *Driverless Cars*

ML is a type of (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. ML algorithms use historical data as input to predict new output values.

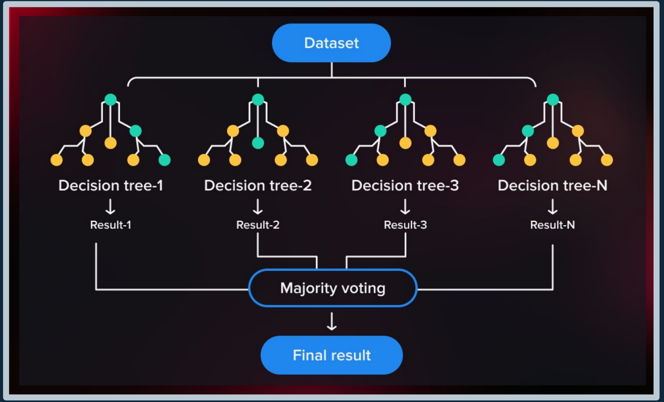
# Decision Tree

A Decision Tree is a Flow Chart, and can help you make decisions based on previous experience.We can perform both classification and regression in decision tree.

```
import pandas
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pandas.read_csv("data.csv")

d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)
```



# Random Forest



Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

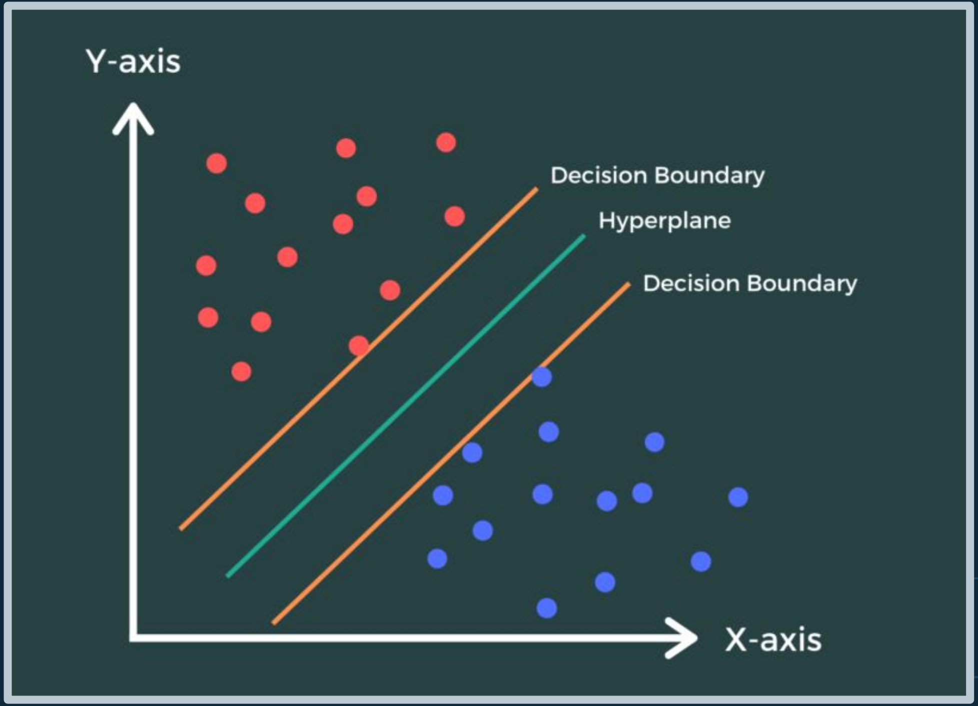
```
Import pandas  as pd
df = pd.read_csv(‘test.csv’)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=44)

from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=50, max_features="auto",
random_state=44)rf_model.fit(X_train, y_train)

predictions = rf_model.predict(X_test)
predictions
```

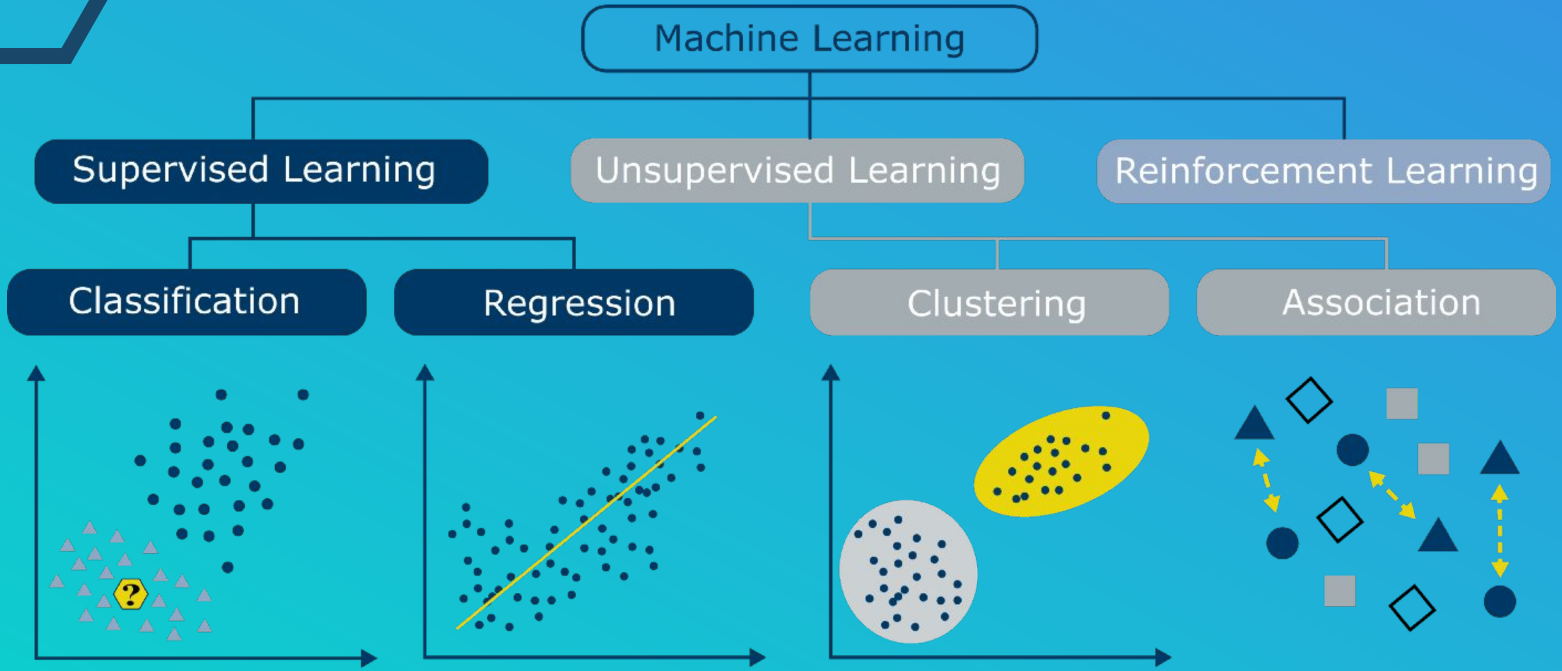
# Support Vector Machine

SVM is a classification based algorithm, where we have a margin (Line) dividing the data.

SVM

```
import pandas as pd
Import matplot.pyplot as plt
Import seaborn as sns
df = pd.read_csv('TEST.csv')
x=df[['A1']]
y=df[['A2']]
from sklearn.model selection import train_test_split
train_test_split (x,y, test_size = 0.25)
from sklearn.svm import SVC
svc=SVC
svc.fit(x_train,y_train)
y_pred = svc.predict(x_test)
from sklearn.metrics import confusion_matrix
confusion_matrix (x_test, y_pred)
```





- ◇ **Supervised Learning (Labeled Data) :**
  - Input Variable (x) and Output Variable (y)
  - Regression : Continuous Numeric Prediction : y = f(x)
  - Classification : Dependent = *Categorical*, Independent = *Categorical or Numerical*
- ◇ **Unsupervised Learning (Custering) :**
  - Only Input Variable
  - Increase Intra Cluster Similarity and Inter Cluster Dissimilarity

# Linear Regression

*A predictive model used for finding the linear relationship between dependent variable and one or more independent variables.*

- Dependent Variable : Continuous Numerical
- Residuals : Actual Y -  Predicted Y
- Best Fit Line : Sum of residual square should be minimum

# Logistic Regression

- Dependent Variable : Categorical
- Sigmoid Curve (S Curve)

# Regression in Python

1. For Reading CSV File

```
import pandas as pd
df = pd.read_csv (file.csv)
df.head()
```

2. For Scatter Plot

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.scatterplot (x = ‘length’, y = ‘width’, data=file)
plt.show()
```

3. Define and Store Variables:

```
y = file[[ ‘length’]]
x = file[[ ‘width’, ‘area’]]
```

3. For Train Test Split

```
From sklearn.model selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split (x,y, test_size = 0.3)
# To see the test set
x_test.head()
```

4. For Model Fit

```
From sklearn.linear model import linear regression
lr=linear regression
lr.fit(x_train, y_train)
lr.predict(x_test)
y_test.head()
```

5. Error in Prediction

```
From sklearn.metrics import mean squared error
Mean_squared_error (y_test, y_pred)
```

8

# Time Series Analysis

- Every business operated under risk and understanding forecasting helps us to assess these risks.
- Y = f(X), where Y = Dependent Variable (Future), X = Independent Variable (Past) i.e Time
- Measurements are taken at regular time interval (Between Identical Gaps)
- Features of Time Series Data
  - *Data cannot be independent (Relation between time intervals)*
  - *Ordering Matters*
  - *Missing data is not allowed (Sequence Break)*

# DECOMPOSITION OF TIME SERIES

TREND
Long Term

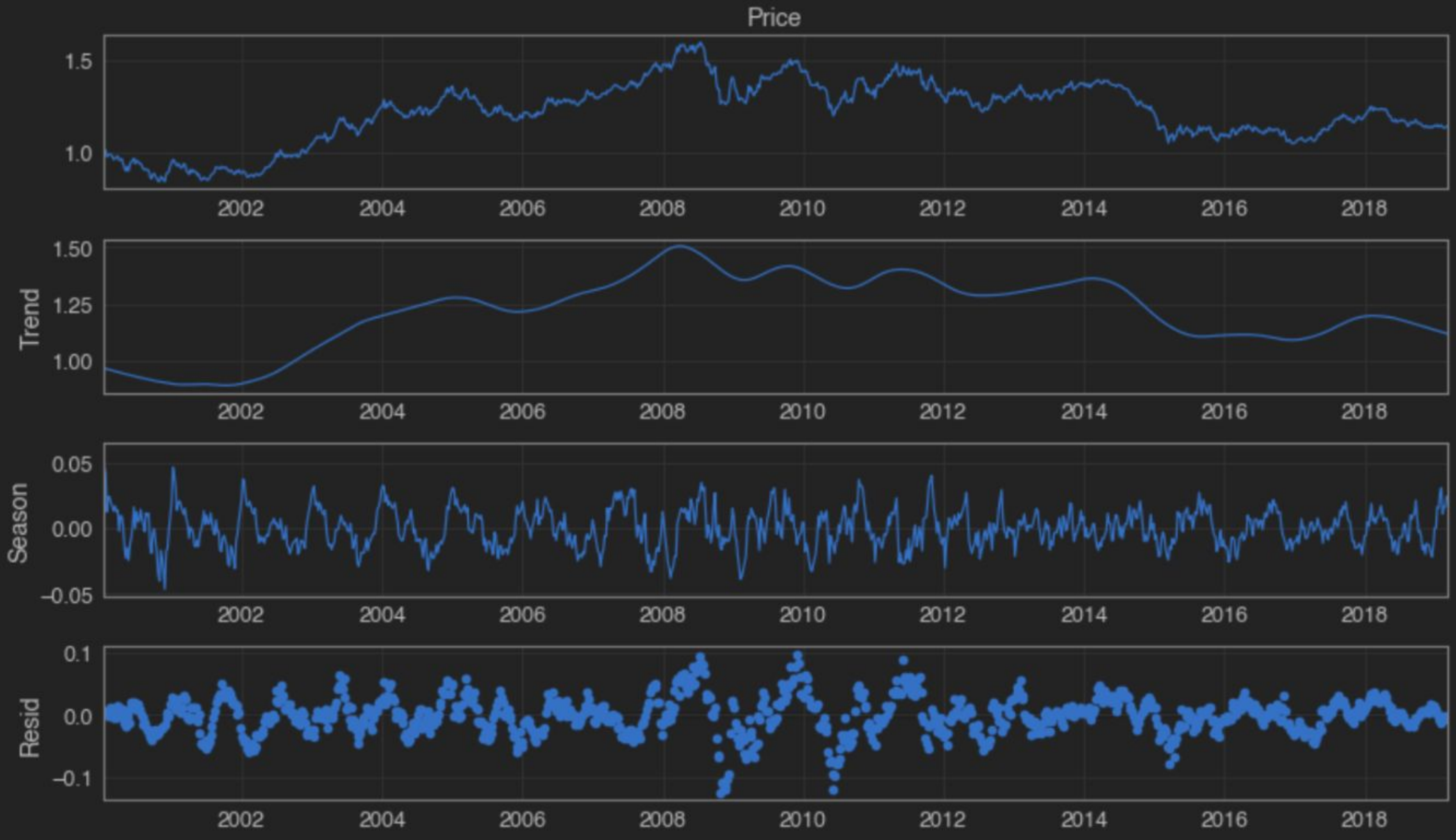
SEASONAL
Ups & Downs

IRREGULAR
Covid 19

Breaking down a time series data into trend, seasonality and irregular components. Compare the long term movements of series w.r.t short term movements.

# Decomposition Model

- Additive
  - Observation = Trend + Seasonality + Error
  - Y = T + S + I
  - When Seasonality Pattern is Constant
- Multiplicative
  - Observation = Trend x Seasonality x  Error
  - Y = T x S x I
  - When Seasonality is not Constant



Price
1.5
1.0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Trend
1.50
1.25
1.00
Season
0.05
0.00
−0.05
Resid
0.1
0.0
−0.1

# Python Code (Time Series)

```
import numpy a np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal decompose

# Air Passenger.csv
Decompose the time series multiplicative

df = seasonal_decompose (df,model=”multiplicative”)
df.plot()
plt.show()
```

# Thanks!