My Master Code Book & FAQ's

- Edited and updated by Om Sarmalkar (recommend changes - omsarmalkar@gmail.com (mailto:omsarmalkar@gmail.com))

Table of contents

Basics

Python Basics

```
# To extract all keywords in python 3.6 use the below code

import keyword

# Import basic packages
import numpy as np                                          # Implemennts milti-
dimensional array and matrices
import pandas as pd                                         # For data manipulation and
analysis
import pandas_profiling
import matplotlib.pyplot as plt                             # Plotting library for Python
programming language and it's numerical mathematics extension NumPy
import seaborn as sns                                       # Provides a high level
interface for drawing attractive and informative statistical graphics
%matplotlib inline
sns.set()

from subprocess import check_output

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

```
More samples to upload images and videos in python file

from IPython.display import Image
Image(filename='fig/img_4926.jpg')

from IPython.display import YouTubeVideo
YouTubeVideo('iwVvqwLDsJo')
```

```
#borders for dataframe

%%HTML
<style type="text/css">
    table.dataframe td, table.dataframe th {
        border-style: solid;
        border: 3px solid lightgray;
    }
</style>
```

```
#TOGGLE YOUR RAW CODES

from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
 if (code_show){
 $('div.input').hide();
 } else {
 $('div.input').show();
 }
 code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here to toggle
on/off the raw code."></form>''')
```

```
%%HTML
#Writing Docstring

def square(num):
    """
    Function to square a number.
    Print this docstring using __doc__ attribute of the function
    """
    return num * num

print(square(20))
print(square.__doc__)
```

```
print(keyword.kwlist)        #all keyword list
print(id(Add your variable here))              #print address of variable:
```

```
Integers, floating point numbers and complex numbers falls under Python numbers category.
They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to
and the isinstance() function to check if an object belongs to a particular class.

Python Strings
String is sequence of Unicode characters. We can use single quotes or double quotes to
represent strings.
Multi-line strings can be denoted using triple quotes(single/double), ''' or """.
A string in Python consists of a series or sequence of characters - letters, numbers, and
special characters.
Strings can be indexed - often synonymously called subscripted as well.
The first character of a string has the index 0.
Slicing string
hey = "Om is great"
print("What do you know?:",hey)
print("who is great?:", hey[0:2])
print("Om is what? ", hey[6:12])
```

```
Python List
List is an ordered sequence of items. It is one of the most used datatype in Python and
is very flexible.
All the items in a list do not need to be of the same type.
Declaring a list is pretty straight forward. Items separated by commas are enclosed
within square brackets '[ ]'.

print(myList[2])                    # Print an element based on its index. Index starts
from 0
myList.remove(2.2)                           # Remove item from a particular index
print(myList)
LIST.pop(3)                                # Remove item from a particular index
myList.append(2.2)                           # Add item at the last index
```

```
Python Tuple

Tuple is an ordered sequence of items same as list.
The only difference is that tuples are immutable. Tuples once created cannot be modified.
Tuples are used to write-protect data and are usually faster than list as it cannot
change dynamically.
It is defined within parentheses () where items are separated by commas.

myTuple = (10,20,30,"Text")
```

```
print(myTuple[0:3])                # Read elements by their corresponding index values
```

```
Python Set
Set is an unordered collection of unique items.
Set is defined by values separated by comma inside curly braces { }.

mySet = {10, 20, 30, 40, 50, 0}
mySet = {10, 20, 20, 30, 30,40}       # Only unique values considered, duplicates removed
automatically
print(mySet)
OUT - {40, 10, 20, 30}
```

```
Python Dictionary
Dictionary is an unordered collection of key-value pairs.

It is generally used when we have a huge amount of data.
Dictionaries are optimized for retrieving data.
We must know the key to retrieve the value.

In Python, dictionaries are defined within curly braces {}
with each item being a pair in the form key:value.
Key and value can be of any type.
test = {'key1': "value1", 'key2': "value2", 'key3': "value3",}

Update a value in the dictionary
myDictionary['key2'] = "value2.2"
```

```
Conversion between Datatypes
To convert between different data types use different type conversion functions like
int(), float(), str() etc.

# 1> list to set
myList1 = ['a','b','c','c','d','D']
print(myList1)
print(type(myList1))
mySet1 = set(myList1)
print(type(mySet1))
print(mySet1)

# 2> String to list
myString1 = 'This is string to list'
print(type(myString1))

om = list(myString1)
print(om, type(om))
print(om[0:4])
om1 = str(om)
print(om1)
print(om[0])
```

```
List Comprehensions
List comprehensions provide a concise way to create lists.
This can be thought of a process that makes it easier to create lists.

Common applications are to make new lists where each element is the result of some
operations applied to each member of another sequence or iterable, or to create a
subsequence of those elements that satisfy a certain condition.

myList = []               # Normal way to create an empty List
```

```
for i in range(4,20): # Append the squared element to the list
    myList.append(i**3)
print(myList)

myList = [i**2 for i in range(4)]
print(myList)

samelist = []
for i in range(5):
    rowlist = []
    for j in range(1,6):
        rowlist.append(j)
    samelist.append(rowlist)
print(samelist)
```

```
Output Formatting
print("{} is half of {}".format(myInt1, myInt2), "is incorrect statement")

Input
userInput = input("Please enter some data: ")
print("You typed in: ",userInput)
```

```
variable1, variable2 = 5, 2
print(variable1 + variable2)  # Addition(+)
print(variable1 - variable2)  # Subtraction(-)
print(variable1 * variable2)  # Multiplication(*)
print(variable1 / variable2)  # Division(/)
print(variable1 % variable2)  # Modulo division (%)
print(variable1 // variable2) # Floor Division (//)
print(variable1 ** (variable2*3)) # Exponent (**)
```

```
variable1, variable2 = 3, 7           # Decimal to bitwise values -> 3 - 0000 0011 & 7 -
0000 0111
print(variable1 & variable2)          # Bitwise AND ->       0000 0011 & 0000 0111 ->
0000 0011 -> 3
print(variable1 | variable2)          # Bitwise OR ->        0000 0011 | 0000 0111 ->
0000 0111 -> 7
print(~variable2)                     # Bitwise NOT ->     ~ 0000 0111 -> 1111 1000 ->
-8
print(variable1 ^ variable2)          # Bitwise XOR ->       0000 0011 ^ 0000 0111 ->
0000 0100 -> 4
print(variable1>>2)                   # Bitwise rightshift   0000 0011>>2 -> 0000 0000 -
> 0
print(variable1<<2)                   # Bitwise Leftshift    0000 0011<<2 -> 0000 1100 -
> 12
```

```
Assignment operators
Assignment operators are used in Python to assign values to variables.

age = 50 is a simple assignment operator that assigns the value 50 on the right to the
variable (age) a on the left.
=,  +=,  -=,  *=,  /=,  %=,  //=,  **=, &=,  |=,  ^=,  >>=,  <<= are Assignment operators
age = 40
om = 55
age += 4        # Add AND  <- age = age + 4
age -= 7        # Subtract AND (-=)
age *= 4        # Multiply AND (*=)
age /= 4        # Divide AND (/=)
```

```
age %= 5       # Modulus AND (%=)
print(age)
om //= 11 # Floor Division (//=)
print(om)
om **= 2       # Exponent AND (**=)
print(om)
```

```
If else loop:

age = input()
age = int(age)
if age < 18:
    if age >= 12:
        print("Teen")
    else:
        print("child")
else:
    print("adult")
print('This will always get executed')
```

```
while loop: Use while loop to iterate over a block of code as long as the test expression
(also called test condition) is true.
Syntax
while test_expression:

    Body of while
The body of the loop is entered only if the test_expression evaluates to True.
After one iteration, the test expression is checked again.
This process continues until the test_expression evaluates to False.
While loop has an optional else block which one may use if one wishes to use it.
The else block gets executed when the condition in while statement is False.

The else can be skipped if we use a break command in the while block
myList = [1, 2, 3, 4, 5]

#iterating over the list
index = 0
while index < len(myList):
    print(myList[index])
    break
    index += 1
else:
    print('Completed iterating the list')

print('Eitherway printed')
```

```
for Loop Python: for loop in one of the most often used Python looping technque to
iterate over a sequence (list, tuple, string) or other iterable objects.
Iterating over a sequence is called traversal.
Syntax:
for element in sequence :
    'for' code block
Here, element is the variable that takes the value of the item inside the sequence on
each iteration.
Loop continues until we reach the last item in the sequence.

# Sum of all numbers in a list

myList = [1, 2, 3, 4, 5,2,5]
```

```
myList = [1, 2, 3, 4, 5,6,7]

sum = 0
for values in myList:
    sum += values
print(sum)


range(start,stop,step size)
# Print range of 5
for element in range(5,10,2):
    print(element)
```

```
Examples of break and continue

# Use of break
names = ['Suchit', 'Rakesh', 'Roshni']
for name in names:               # Let us iterate over the names list
    if name == 'Rakesh':
        break
    print(name)
else:
    print('For is completed and we are in else part')
print("Totally outside the for & else loop")

# Use of continue
names = ['Suchit', 'Rakesh', 'Roshni']

for name in names:               # Let us iterate over the names list
    if name == 'Rakesh':
        continue
    print(name)                  # If continue condition is satisfied we skip this line
and carry on with the next iteration
else:
    print('For is completed and we are in else part')
print("Totally outside the for & else loop")
```

```
Python Set Operations

Operations:
union
intersection
symmetric difference
subset

mySet1 = {1, 2, 3, 4, 5}
mySet2 = {3, 4, 5, 6, 7}
print(mySet1 | mySet2)              # union of 2 sets using | operator
print(mySet1.union(mySet2))         # Alternately use union() method

print(mySet1 & mySet2)              # Intersection of 2 sets using & operator
print(mySet1.intersection(mySet2))  # Alternately use intersection() method

print(mySet1 - mySet2)              # set Difference: set of elements that are only in
mySet1 but not in mySet2
print(mySet1.difference(mySet2))    # use differnce() function method

print(mySet1^mySet2)                        # For symmetric difference use ^ operator
print(mySet1.symmetric_difference(mySet2)) # Alternately use symmetric_difference
function
```

Function

"def" keyword notifies the start of function header
Arguments (parameters) through which we pass values to a function. These are optional
A colon(:) to mark the end of funciton header
Doc string describe what the function does. This is optional
"return" statement to return a value from the function. This is optional

```
def Facto(num):
    """
    Fact of number
    """
    fact = 1
    while(num>0):
        fact *= num
        num -= 1
    return fact
number = 6

print("Factorial of number {} is : {}".format(number, Facto(number)) )
```

Builtin functions

all()¶
The function all() retruns,
True: If all elements in an iterable data collection are true
False: If any element in an iterable data collection is false (Remember 0 & None are considered False)

dir()¶
The dir() tries to return a list of valid attributes of the object.
If the object has dir() method, the method will be called and must return the list of attributes.
If the object doesn't have dir() method, this method tries to find information from the dict attribute (if defined), and from type object. In this case, the list returned from dir() may not be complete.

divmod()
The divmod() method takes two numbers and returns a pair of numbers (a tuple) consisting of their quotient and remainder.

enumerate()
enumerate() method adds counter to an iterable data collection & returns it
Syntax: enumerate(iterable, start=0)
The enumerate() method takes two parameters:
iterable - a sequence, an iterator, or objects that supports iteration
start (optional) - enumerate() starts counting from this number. If start is omitted, 0 is taken as start.

filter()
The filter() method constructs an iterator from elements of an iterable for which a function returns true.
Syntax: filter(function, iterable)
The filter() method takes two parameters:
function - function that tests if elements of an iterable returns true or false If None, the function defaults to Identity function - which returns false if any elements are false
iterable - iterable which is to be filtered, could be sets, lists, tuples, or containers of any iterators

isinstance()

```
The isinstance() function checks if the object (first argument) is an instance or
subclass of classinfo class (second argument).
Syntax: isinstance(object, classinfo)

map()
Map applies a function to all the items in an input_list.
Syntax: map(function_to_apply, list_of_inputs)
```

```
Lambda or Anonymous Functions
In Python, anonymous function is a function that is defined without a name.
While normal functions are defined using def keyword,
in Python anonymous functions are defined using the lambda keyword.
Lambda functions are used extensively along with built-in functions like filter(), map()

square = lambda x: x**2          # Using lambe we write the squaring function as beside
print(square(10))

myList = [1, 2, 3, 4, 5]
oddList = list(filter(lambda x: (x%2 != 0), myList))
print(oddList)
```

Modules

```
Python provides a lot of standard modules that can be used for various purposes.
https://docs.python.org/3/py-modindex.html

import math
math.factorial(5)

import random
random.random()

from datetime import date
print(date.today())
```

```
File Handling  ##Pending
```

Numpy

Introduction to Numpy
Numpy is a library developed for Python which can handle large, multi-dimensional arrays and matrices. It has a large collections of mathematical functions to operate on these arrays.

Lets have a brief comparison of Numpy with Python Lists.
Numpy is remarkably faster than Python Lists for many reasons.

It was designed for efficient data storage. All the elements of numpy arrays are stored sequentially with a fixed width for each value. On the other hand Lists are pointers to data stored elsewhere. The number of separate reads the computer has to do is smaller for numpy.

Numpy has uniform datatypes instead of Lists. The computer performs a logic for each different element type. This is completely avoided with Numpy.

Numpy has optimized functions for many mathematical operations on arrays and matrices. This is why they are faster than regular math operations on lists.

```
import numpy as np

DATA = [[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
type(DATA)
mat.shape
mat.dtype

arange(start, stop, step)
np.arange(0,11,5)
```

```
numpy.linspace() returns evenly spaced numbers over a specified interval
numpy.eye is used to generate an identity matrix
numpy.zeros generates a matrix with all elements 0.

np.zeros((4,2))
np.ones((12,218))

Generate 10 points between 0 and 5.
np.linspace(0,5,10)      #linspace is used for making high resolution plot

Generate an array of 5 random numbers
np.random.rand(5)

Generate a 4*4 matrix where the elements are distributed in random distribition.
np.random.randn(4,4)

np.reshape(#,#)

Array Slicing
To access more than one element of the array use slicing.
```

```
IN: test = np.array([[0,1,2],[3,4,5],[6,7,8],[9,10,11]])
OUT: array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])

IN: test[0:2,0:2]
OUT: array([[0, 1],
       [3, 4]])

IN: arr_2d = np.array([[1,2,3],[5,6,7],[8,9,10],[12,13,14]])    # Creating numpy array
arr_2d

OUT: array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 8,  9, 10],
       [12, 13, 14]])

IN: scaler = 3
IN: arr_2d +  3                                          # operation with a scaler
OUT: array([[ 4,  5,  6],
       [ 8,  9, 10],
       [11, 12, 13],
       [15, 16, 17]])

IN: arr_1d = np.array([10,10,10])
IN: arr_2d + arr_1d                                       # operation with a array
of different shape
OUT: array([[11, 12, 13],
       [15, 16, 17],
       [18, 19, 20],
       [22, 23, 24]])
```

```
Numpy Mathematical Functions

Here you can see a lot of commonly used built-in functions of numpy for mathematical
operations. These functions are faster and optimized for large size arrays.

IN: arr = np.arange(1,11)  # Lets first create a numpy array
arr

OUT: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])

IN: arr.min()   = 2
IN: arr.max()  = 10

IN: arr.argmin()   = 0                                    # Index position of minimum
of array
IN: arr.argmax()   = 9                                   # Index position of maximum
of array

np.sqrt(arr)                                        # To calculate square root of all
elements in an array
arr.mean()                                          # To calculate mean of all the
values in an array
np.exp(arr)                                         # To calculate exponential value of
each element in an array


arr = np.arange(0,16)                               # Using reshape we can change the
dimensions of the array
```

```
dimensions of the array
arr_2D = arr.reshape(2,8)
arr_2D.flatten()                                    # Flatten is used to convert a 2D
array to 1D array
arr_2D.transpose()                                  # Transpose is used to convert the
rows into columns and vice-versa


Concatenate
arr_x = np.array([[1,2,3,4],[5,6,7,8]])             # Lets create 2 arrays
arr_y = np.array([[21,22,23,24],[25,26,27,28]])

IN: np.concatenate((arr_x, arr_y), axis=1)              # Join 2 arrays along columns
OUT: np.concatenate((arr_x, arr_y), axis=1)              # Join 2 arrays along columns
np.concatenate((arr_x, arr_y), axis=1)              # Join 2 arrays along columns
array([[ 1,  2,  3,  4, 21, 22, 23, 24],
       [ 5,  6,  7,  8, 25, 26, 27, 28]])

np.hsplit(arr, 2)                                  # It will split the array into 2
equal halves along the columns
np.vsplit(arr_z, 2)                                # It will split the array into 2
equal halves along the rows
```

Takeaways
Using concatenate function we can merge arrays columnwise and rowwise. Also arrays can be horizontally and vertically spliited using hsplit and vsplit.

Conclusion
Numpy is open source add on module to Python.

By using NumPy you can speed up your workflow and interface with other packages in the Python ecosystem that use NumPy under the hood.
A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays.
It provide common mathematical and numerical routines in pre-compiled, fast functions.
It provides basic routines for manipulating large arrays and matrices of numeric data.
Key Features

NumPy arrays have a fixed size decided at the time of creation. Changing the size of an ndarray will create a new array and delete the original.
The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory.
NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data.

Create a random vector of size 12 , sort it in decreasing order, make 4*3 array using it.

```
import numpy as np
def generate():
    rand_vect = np.random.random(12)
    rand_sort = np.array(sorted(rand_vect,reverse=True))
    rand_new = rand_sort.reshape(4,3)
    print(rand_new)
generate()
```

Below is Python dictionary data and Python list labels. Create a DataFrame df from this dictionary data which has index as labels. Select the rows where the age is missing, i.e. is NaN

```
import numpy as np
```

```
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog',
'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
import pandas as pd
def generate():
    df = pd.DataFrame(data, index=labels)
    print(df[df['age'].isnull()])
generate()
```

```
From the give dataframe df, in the 'animal' column change the 'dog' entries to
'Labrador', change the age in row 'd' to 2.7 and calculate the mean age (name it
mean_age) for each different animal in df. Print df and return mean_age

import pandas as pd
import numpy as np
data = {'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog',
'dog'],
        'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
        'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df = pd.DataFrame(data, index=labels)
def generate():
    df['animal'] = df['animal'].replace('dog', 'Labrador')
    df.loc['d', 'age'] = 2.7
    mean_age = df.groupby('animal')['age'].mean()
    print(df)
    return mean_age
generate()
```

```
Find the positions of numbers that are multiples of 4 from a series? Print series and
return position. Hint: use argwhere.

import pandas as pd
import numpy as np
def generate():
    series = pd.Series(np.random.randint(1, 10, 7))
    pos=np.argwhere(series % 4 == 0 )
    print(series)
    print(pos)
generate()
```

```
Create an numpy array sequence named num_seq when only the starting point, step size and
length of sequence is given

import numpy as np
length = 15
start = 5
step = 2

def generate(start, length, step):
    end = start + (step*length)
    num_seq = np.arange(start, end, step)
    return num_seq
```

```
generate(start, length, step)
```

```
reate a DatetimeIndex that contains each business day of 2016 and use it to index a
Series of random numbers. Let's call this Series bus_2k16. Hint: use pd.date_range.

import pandas as pd
import numpy as np
def generate():
    dti = pd.date_range(start='2016-01-01', end='2016-12-31', freq='B')
    bus_2k16 = pd.Series(np.random.rand(len(dti)), index=dti)
    print(bus_2k16)
generate()

For each calendar month in dataframe bus_2k16, from previous problem. Print out the mean
of values also find the sum of the values in bus_2k16 for every Monday.

import pandas as pd
import numpy as np
def generate():
    dti = pd.date_range(start='2016-01-01', end='2016-12-31', freq='B')
    bus_2k16 = pd.Series(np.random.rand(len(dti)), index=dti)
    print(bus_2k16)
    print(bus_2k16.resample('M').mean())
    print(bus_2k16[bus_2k16.index.weekday == 0].sum())
    return None
generate()
```

```
8x8 matrix 0 & 1 alternate

import numpy as np
def generate():
    Z = np.zeros((8,8),dtype=int)
    Z[1::2,::2] = 1
    Z[::2,1::2] = 1
    print(Z)
generate()
```

Pandas

```
import pandas as pd
from IPython.display import display

csv_df = pd.read_csv("http://........")   #  # read_csv is used to read csv file
```

```
Create a dataframe 4x200 with random values
```

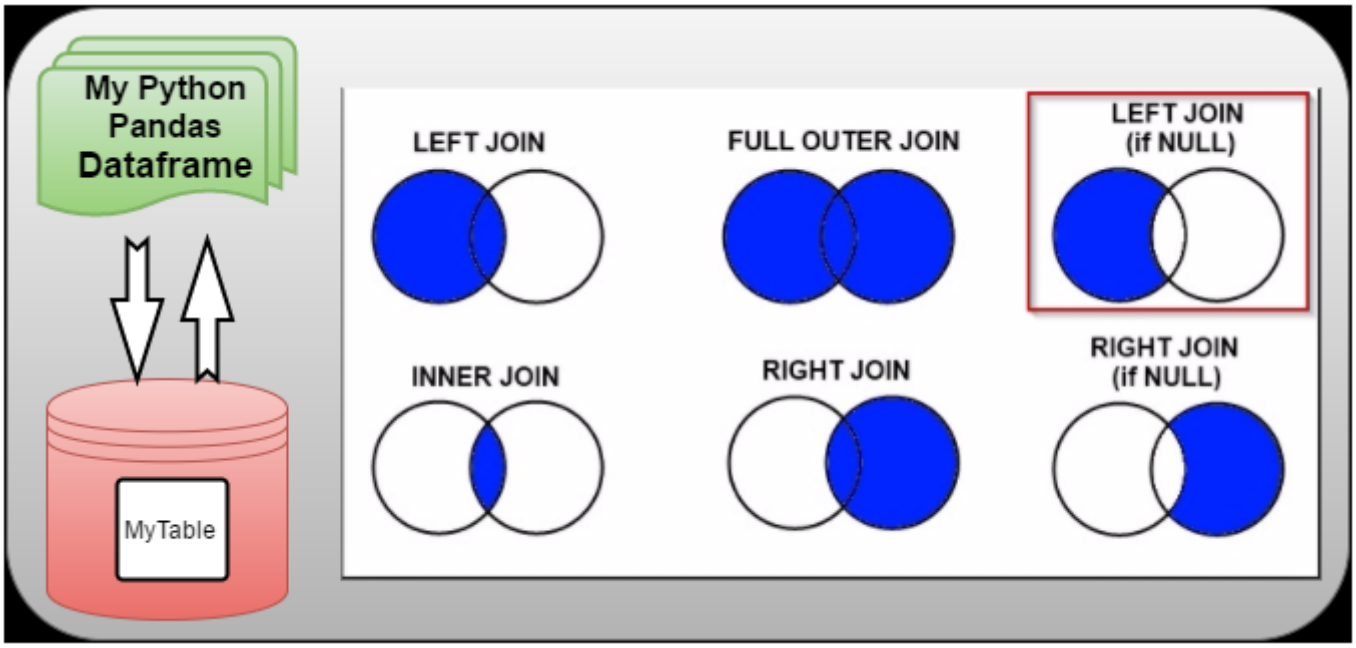
```
df = pd.DataFrame(columns=['COL_A','COL_B','COL_C','COL_D'], index=range(1,201))
#defining dataframe
df["COL_A"] = np.random.choice(["True", "False"], len(df))       #Creating random column
with True & False
df["COL_B"] = np.random.choice(["yes", "no"], len(df))           #Creating random column
with yes & no
df["COL_C"] = np.random.choice(["1", "0"], len(df))               #Creating random column
with 0 & 1
df['COL_D'] =  np.arange(len(df))                              # Adding a new column with
numbers 1-200
df["COL_X"] = np.random.randint(150,200, len(df)) *5       #Adding Radom numbers in range
- *5 is step size

df['Index'] =  np.arange(len(df))                              # Adding a new column with
numbers 1-200
df = df.set_index('Index')                                    #setting the index

df = pd.DataFrame({'COL_A':['True'],'COL_B':['yes'],'COL_C':['3']})    #Appending a row
on a DataFrame

df2 = df.append(df1, ignore_index = True)     #Appending a data frame by ignoring the
index
df2.reset_index(inplace=True)                #resetting the index
df2['index']                                 #deleting the index
```

```
df.shape
df.count()
df.index
df.columns
df = df.set_index('C')          #setting the index
df['COL_A'].unique()               #Unique values
df['COL_A'].nunique()              #count of Unique values
df['COL_A'].value_counts()         # Observe the number of counts (Of features)


df[3:7]                                                 # Accessing/Filtering all the
elements from 3rd to 7th index
df['COL_A']                                                  # Accessing the data using
labels
df[['COL_A','COL_D']]                                              # Accessing the data
using multiple labels
df[['1','2','3']]                                        # Accessing multiple elements in
a series using labels of dict
df['COL_E'] = df['COL_C'] + df['COL_D']                        # Addition of two columns.
You can perform any math operation.


df.drop('COL_E', axis=1, inplace=False)             # Drop or modify the dataframe
columns
df.drop("COL_E", axis=0, inplace =False)             # Drop or modify the dataframe row
after setting index
df.drop(['COL_E'], axis=1)

df.drop(df.index[1:5], axis=0, inplace =True)      # Drop or modify the dataframe rows
after setting index
del df['COL_E']                                        # Column Deletion using del

- Indexing using iloc
df.iloc[0:4,1:3]                                    # It will return rows from 0-4 and
columns from 0-5
df.iloc[:,0:5]                                      # Return the columns from 0-5
```

```
df.iloc[0:4,:]                                  # Return all the rows from 0-4

df.loc[3:5,"COL_C"]                               # Returns index "D" 1-4 and column
"C"
df.loc[1:4,"COL_B":"COL_C"]                        # Returns index "D" 1-4 and columns
B to C
df.loc[:,"COL_B":]                              # Return all the columns from "B"
onwards and all the rows
```

```
Merging, Concatenating and Appending
In the previous section we saw how to add rows or columns.
Here we will see how to merge two dataframes.

df1 = pd.DataFrame({                                    # Lets create 2 dataframes
    'id':[1,2,3,4,5],
    'name':['a','b','c','d','e'],
    'sub':['sub1','sub2','sub3','sub4','sub5']
})
df2 = pd.DataFrame({
    'id':[1,2,3,4,5],
    'name':['b','c','d','e','f'],
    'sub':['sub3','sub4','sub5','sub6','sub7']
})


pd.concat([df1, df2], axis=0)                           # Joining two DataFrame along
the rows
pd.concat([df1, df2], axis=1)                           # Joining 2 DataFrame along
the columns
pd.merge(left=df1, right=df2, on='sub')                 # Joining 2 DataFrame using
'sub' as keypd.merge(left=df1, right=df2, on='sub',how='left')
# on left
pd.merge(left=df1, right=df2, on='sub', how='outer')        # left, right, outer

another sample
IN:
Mylist1 = [(1,10),(2,20),(3,30),(4,40),(5,50)]
labels1 = ['ID','NUM']
df1_x = pd.DataFrame.from_records(Mylist1,columns= labels1)
df1_x
Mylist2 = [(1,'COL_A'),(6,'COL_B'),(3,'COL_C'),(8,'COL_D'),(10,'COL_E')]
labels2 = ['ID','ALPHA']
df2_x = pd.DataFrame.from_records(Mylist2,columns= labels2)
print(df1_x)
print(df2_x)

df_onlyleft = pd.merge(left=df1_x,right=df2_x,on='ID',how='left',indicator=
True).query('_merge=="left_only"').drop('_merge',1)
df_onlyleft

OUT
   ID  NUM
0   1   10
1   2   20
2   3   30
3   4   40
4   5   50
   ID ALPHA
0   1     A
1   6     B
2   3     C
```

```
3   8     D
4  10     E
ID  NUM ALPHA
1   2   20  NaN
3   4   40  NaN
4   5   50  NaN


IN:
df_left = df1_x
df_right = df2_x
df_onlyleft2 = df_left.query('ID not in @df_right.ID')
df_onlyleft2

OUT:
ID  NUM
1   2   20
3   4   40
4   5   50
```




```
SORT & Filter

df['COL_D'] >=15                                          # This returns a Boolean Series
df[~(df['COL_D'] >=15)]                                     # This returns all the rows
for which the condition is True

df1 = df[(df['COL_D'] >=15) & (df['COL_B'] != 'yes')]
df1 = df[(df.COL_B =='yes')]['COL_D'].count()                    #Sort and count
df1 = df[(df.COL_B !='yes')]['COL_D'].Sum()                     #Sort and sum
df1 = df.groupby(['COL_A','COL_B','COL_C'])['COL_D'].agg(['sum','mean','count'])
#aggregate


df1 = df.groupby(['COL_A','COL_B'])        #group A & B and count / mean /sum
df1.count()
df1.mean()
df1.sum()
```

```
df1 = df.groupby('COL_A').sum()      #group only values of A  and sum


df1 = df.set_index(['COL_A','COL_B'])                                    # Lets set 'sex'
and 'size' as our index
df1.sort_index(inplace = True)                                        # Sorting all the index
of A & B values in ascending order
df1 = df1.loc['true']                                                          # Accessing
records if index is A
df2 = df1.xs('yes', level='COL_B')                                     # Accessing records if
serving size is 2 (From second index)

df.sort_values('COL_C')                                                    # Sorting the
records based on a column

df.loc[:,df.notnull().all()]           #Select columns without NaN
```

```
Apply function

def times2(x):                                                                  # We are
going to apply this function
    num = x
    if x%6==0:
        num *= 2
    return num

df1 = df['COL_D'].apply(times2)

OR

df['COL_D'] = df['COL_D'].apply(lambda x: x * 2)
df
```

```
# DATE & time
df['Date'] = pd.to_datetime(df['Date'], format='%m/%d/%Y')
df['Time'] = pd.to_datetime(df['Time'], format = '%H:%M:%S')

year = df.Date.dt.year                              # Extracting Year from Date column
print(year.head())

month = df.Date.dt.month                            # Extracting Month from Date column
print(month.head())

day = df.Date.dt.day                                # Extracting Day from Date column
print(day.head())

day_of_week = air_df.Date.dt.dayofweek                 # Extracting the day of the week
number
print(day_of_week.head())

day_name = air_df.Date.dt.weekday_name                 # Extracting the name of the day
print(day_name.head())

day_of_year = air_df.Date.dt.dayofyear                 # Extracting the day of the year
print(day_of_year.head())

hour = air_df.Time.dt.hour                             # Extracting the hour from time
print(hour.head())
```

```
minute = air_df.Time.dt.minute                          # Extracting the minutes from the
time
print(minute.head())

second = air_df.Time.dt.second                          # Extracting the seconds from the
time
print(second.head())

 measure the number of records before 01/01/2005
datestamp = pd.to_datetime("01/01/2005", format='%d/%m/%Y')

from datetime import timedelta
df1 = df[air_df['Date'] < datestamp].tail()
```

```
Convert the first character of each element in a series to uppercase?And return the new
series. Hint: use map and lambda methods

import pandas as pd
series = pd.Series(['how', 'to', 'learn', 'data science?'])
def generate():
    new_series = series.map(lambda x: x[0].upper() + x[1:])
    return new_series
generate()
```

```
Given the dataframe df, In the From_To column split each string on the underscore
delimiter _ to give a new temporary DataFrame with the correct values. Also standardise
the strings so that only the first letter is uppercase (e.g. "londON" should become
"London".) Assign the correct column names to this temporary DataFrame named as temp.
Return temp

import numpy as np
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
                               'Budapest_PaRis', 'Brussels_londOn'],
              'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
              'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
                   'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
                               '12. Air France', '"Swiss Air"']})
import pandas as pd
def generate():
    temp = df.From_To.str.split('_', expand=True)
    temp.columns = ['From', 'To']
    temp['From'] = temp['From'].str.capitalize()
    temp['To'] = temp['To'].str.capitalize()
    return temp
generate()
```

```
Given a dataFrame df with a column 'int' of integers. Print out the dataframe with
filtered out rows which contain the same integer as the row immediately above. Also print
out the dataframe after subtracting the mean of all integers from each element.

import pandas as pd
df = pd.DataFrame({'int': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
def generate():
    print(df.drop_duplicates(subset='int'))
    print(df.sub(df.mean(axis=0), axis=1))
    return None
generate()
```

```
Filter out and capitalise all letters of words that contain atleast 2 vowels from a
series?

import pandas as pd
series = pd.Series(['Insaid', 'strives', 'for', 'bringing','out','the', 'best','in','you'
])
def generate():
    from collections import Counter
    mask = series.map(lambda x: sum([Counter(x.lower()).get(i, 0) for i in
list('aeiou')]) >= 2)
    s = series[mask].str.upper()
    print(s)
    return None
generate()
```

```
df.columns = map(str.lower, df.columns)           #mapping all headers to lower case
df.dtypes
```

```
#adding zeros in missing data Nan Values

df.c1.fillna('0', inplace=True)
df.c2.fillna('0', inplace=True)
```

```
Removing string prefix

df.c1 = cd.c1.loc[:,].replace(regex=True, to_replace="abc", value="")

df.c2 = df.c2.loc[:,].replace(regex=True, to_replace=("01. ","02. ","03. ","04. ","05.
","06. ","07. ","08. ","09. ","10. ","11. ","12. ","05A. ","05B. "), value="")

TO remove suffix from entire column
df.c2 = df.c2.loc[,:].replaceregex=True, to_replace=("___ ","___"), value="")


Replacing multiple features
df['c1'] = data['c1'].replace(to_replace=(0, 5, 6), value="___", regex=True)
df.c1.unique()
```

```
cols = ['c1', 'c2','c3']        #converting Float to integer
df[cols] = df[cols].applymap(np.int64)
```

```
df1 = df.copy(deep=True)                    #creating copy of dset
```

```
nlargest
df.nlargest(20,'c1').head(20)   #top20
```

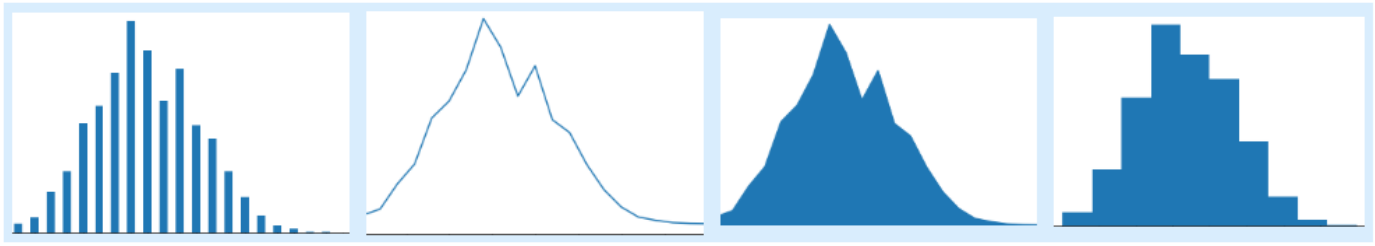
```
totals = df[['c1','c2','c3']]   #totals of selected columns
totals.sum(axis = 0)

#group total and sort by year
total_each_year = df.groupby(['year',])['XYZ'].sum().reset_index()
#grouping
total_each_year = total_each_year.sort_values(by = ['XYZ','year'], ascending=False)
    #sorting
total_each_year
```

```
df['percent'] = df.groupby('c1').apply(lambda s: s.C1.nunique()/s.C1).values
```

Data Visualization

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mat
import bokeh
```

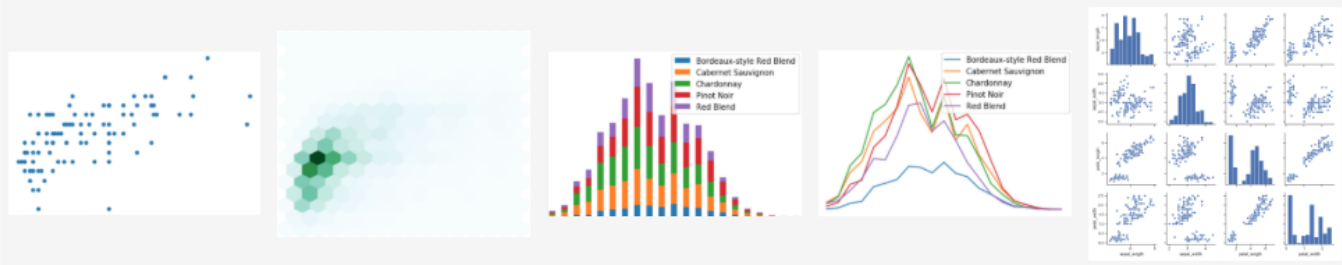
Univariate Plotting with pandas



Bar Chat	Line Chart	Area Chart	Histogram
df.plot.bar()	df.plot.line()	df.plot.area()	df.plot.hist()
Good for nominal and small ordinal categorical data.	Good for ordinal categorical and interval data.	Good for ordinal categorical and interval data.	Good for interval data.

```
Univariate Plotting with pandas

Bar charts

df['X'].value_counts().head(10).plot.bar()
(df['X'].value_counts().head(10) / len(df)).plot.bar()
df['X'].value_counts().sort_index().plot.bar()

Line charts
df['X'].value_counts().sort_index().plot.line()

Area Charts
df['X'].value_counts().sort_index().plot.area()

Histogram
df[df['X'] < 200]['X'].plot.hist()
```

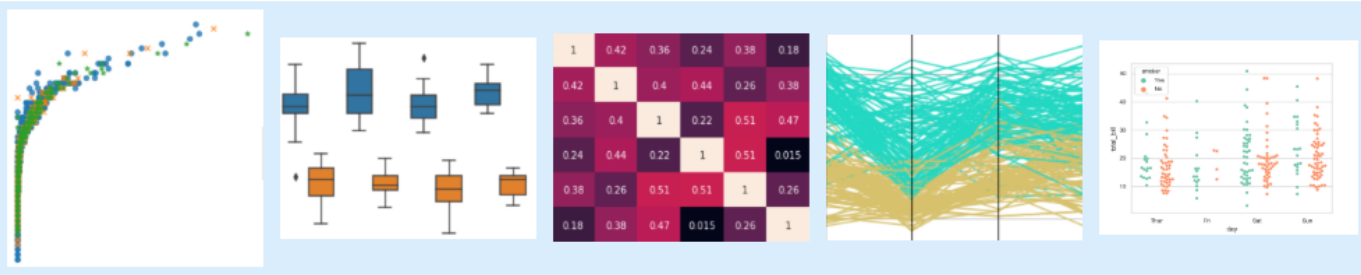
Bivariate plotting with pandas



Scatter Plot	Hex Plot	Stacked Bar Chart	Bivariate Line Chart	Pair Plot
df.plot.scatter()	df.plot.hex()	df.plot.bar(stacked=True)	df.plot.line()	df.sns.pairplot()
Good for interval and some nominal categorical data.	Good for interval and some nominal categorical data.	Good for nominal and ordinal categorical data.	Good for ordinal categorical and interval data.	Good for finding pairwise relationship in the data.

```
Bivariate plotting with pandas

Many pandas multivariate plots expect input data to be in this format, with:
one categorical variable in the columns
one categorical variable in the rows
counts of their intersections in the entries.

Scatter plot
df[df['X'] < 100].sample(100).plot.scatter(x='X', y='Y')

Hexplot
df[df['X'] < 100].sample(100).plot.hexbin(x='X', y='Y', gridsize=12)

Stacked plots
df.plot.bar(stacked=True, figsize=(20,8))

plot1 = df.groupby(["A", "B"])['C'].sum().unstack('B').fillna(0)
plot1.plot(kind='bar', stacked=True,figsize=(20,8) )

Area
df.plot.area()

Line
df.plot.line()

Pair Plot
sns.pairplot(df.drop("X", axis=1), hue="Y",palette="viridis", size=2)
sns.pairplot(df.drop("X", axis=1), hue="Y",palette="inferno", size=2)
```

Multivariate plotting



Multivariate Scatter Plot	Grouped Box Plot	Heatmap	Parallel Coordinates	Swarm plot
df.plot.scatter()	df.plot.box()	sns.heatmap	pd.plotting.parallel_coordinates	sns.swarmplot()
Good for interval and some nominal categorical data.	Good for interval and some nominal categorical data.	Good for nominal and ordinal categorical data.	Good for ordinal categorical and interval data.	Good for drawing categorical variable with non-overlapping points.

```
Multivariate scatter plots

sns.lmplot(x='X', y='Y', hue='ZZZZ', markers=['o', 'x', '*']
           data=df.loc[df['ZZZZ'].isin(['col_val', 'col_val', 'col_val'])],
           fit_reg=False)


Box Plot
sns.boxplot(x="X", y="Y", hue='ZZZZ', data=df)

heatmap
sns.heatmap(df, annot=True)

Parallel Coordinates
parallel_coordinates(df, 'X')

Swarm Plot
sns.swarmplot(x="x", y="Y", hue="ZZZZ", palette="gnuplot", data=df)
```

```
from bokeh.plotting import Figure,figure, output_file, show, output_notebook
from bokeh.layouts import column
from bokeh.models import ColumnDataSource, CustomJS, Slider, HoverTool
output_notebook()

plot = figure(plot_width=300, plot_height=300)
plot.circle(x=[1,2,3], y=[4,5,6], size=20,
                  color="#FB8072", fill_alpha=0.2, line_width=2)
slider = Slider(start=.1, end=1., value=.2, step=.1, title="delta-V")

show(plot)
```

```
CATPLOT

sns.catplot("C1", "C2", data=df, kind="bar" ,palette="PuBuGn_d",height=6, aspect=2)
plt.xlabel('C1')
plt.ylabel('C2')
locs, labels = plt.xticks()
plt.setp(labels, rotation=55)
plt.show()
```

```
# Distribution using Facetgrid

as_fig = sns.FacetGrid(df,hue='C1',aspect=5)

as_fig.map(sns.kdeplot,'C2',shade=True)

ABC = df['C2'].max()

as_fig.set(xlim=(0,ABC))

as_fig.add_legend()
plt.title('NAME_____')
```

Linear Regression

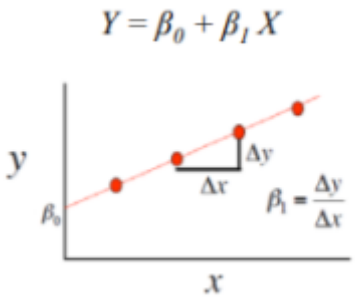
Linear regression is a *basic* and *commonly* used type of **predictive analysis**. The overall idea of regression is to examine two things:

- Does a set of **predictor variables** do a good job in predicting an **outcome** (dependent) variable?
- Which variables in particular are **significant predictors** of the outcome variable, and in what way they do **impact** the outcome variable?

These regression estimates are used to explain the **relationship between one dependent variable and one or more independent variables**. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula :
$y = \beta_0 + \beta_1 x$

What does each term represent?

- y is the response
- x is the feature
- β_0 is the intercept
- β_1 is the coefficient for x



Three major uses for **regression analysis** are:

- determining the **strength** of predictors,
  - Typical questions are what is the strength of **relationship** between *dose and effect*, *sales and marketing spending*, or *age and income*.
- **forecasting** an effect, and
  - how much **additional sales income** do I get for each additional $1000 spent on marketing?
- **trend** forecasting.
  - what will the **price of house** be in *6 months*?

Clean the data before applying any ML Algo

```
1. Write a code to understand the total count and percentage of missing values.
```

```
def missing_data(data):
    total = data.isnull().sum().sort_values(ascending = False)
    percent = (data.isnull().sum()/data.isnull().count()*100).sort_values(ascending =
False)
    return pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data(DATASET)
```

2. Impute the missing value using "median" groupby Column1.

```
def Miss():
    data["Column_missing_values"].fillna(data.groupby("Column1")
["Column_missing_values"].transform("median"), inplace=True)
    return
Miss()
print (data.isnull().sum())
```

3. Plot outliers

```
def plotoutliers():
    import seaborn as sns
    sns.catplot(data="____", orient="h", palette="Set2", kind="box", height=6, aspect=2)
    return None
plotoutliers()
```

4. Fix outliers

```
def outliers(data):
    import pandas as pd
    Q1 = data.quantile(0.05)
    Q3 = data.quantile(0.95)
    Q_diff = Q3 - Q1
    XYZ = data[~((data < (Q1 - Q_diff))|(data > (Q3 + Q_diff))).any(axis=1)]
    print(data[((data < (Q1 - Q_diff))|(data > (Q3 + Q_diff))).any(axis=1)])
    return XYZ
data = outliers(data)
data
```

OR

Write a user defined function to calculate the Inter quartile range for quantile values outside 25 to 75 range. And do the outlier capping for lower level with min value and for upper level with 'q3=1.5*iqr' value.

```
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25)
    q3 = df_in[col_name].quantile(0.75)
    iqr = q3-q1
    lower_bound = df_in[col_name].min()
    upper_bound = q3+1.5*iqr
    print('Column',col_name,'IQR lower bound and upper bound are', lower_bound, 'and',
upper_bound, 'respectively')
    df_out = df_in.loc[(df_in[col_name] > lower_bound) & (df_in[col_name] < upper_bound)]
    return df_out
remove_outlier(DATASET, 'col_name')
```

5. Basic Plots for comparision of EV vs TV

```
f, axes = plt.subplots(2, 2, figsize=(7, 7), sharex=True)                    # Set up the
matplotlib figure
sns.despine(left=True)

sns.distplot(DATASET.TV, color="b", ax=axes[0, 0])

sns.distplot(DATASET.EV1, color="r", ax=axes[0, 1])

sns.distplot(DATASET.EV2, color="g", ax=axes[1, 0])

sns.distplot(DATASET.EV3, color="m", ax=axes[1, 1])
```

*********OR*********

```
sns.pairplot(DATASET, x_vars=['EV1', 'EV2', 'EV3'], y_vars='TV', size=5, aspect=1,
kind='reg')
```

OR

```
JG1 = sns.jointplot("EV1", "TV", data=DATASET, kind='reg')       #EV = EXPLAINATORY
VARIABLE
JG2 = sns.jointplot("EV2", "TV", data=DATASET, kind='reg')       #TV = TARGET VARIABLE
JG3 = sns.jointplot("EV3", "TV", data=DATASET, kind='reg')
#subplots migration
f = plt.figure()
for J in [JG1, JG2,JG3]:
    for A in J.fig.axes:
        f._axstack.add(f._make_key(A), A)
```

6. If there are categorical variables present use ONE HOT ENCODING - dummyfication ; Make sure you drop one of the feature to avoid dummy variable trap or multicollinearity

```
dummy_1 = pd.get_dummies(DATASET, columns=['Column1','column2'], drop_first=True)
#creating dummies
dummy_1.rename(columns=
{'Column1_1':'Add_new_name1','Column1_2':''Add_new_name2','Column2_1':''Add_new_name3'},
inplace=True)   #Replacing column headers
dummy_2 = dummy_1[['new_name1','new_name2','new_name3']]   #selecting only required
columns for analysis
dummy_2.sample(4)

**merge the dummydataset with exisitng**
new = pd.concat([DATASET, dummy_2], axis=1)
_new.head()
```

7. If you have Multiple categorial variables then split data into
        - numerical_dataset
        - categorical_dataset

```
Label-encode categorical dataset
def catds(bank_cat):
    from sklearn.preprocessing import LabelEncoder
    return categorical_dataset.apply(LabelEncoder().fit_transform)
```

```
categorical_dataset = catds(categorical_dataset)
categorical_dataset

Combine both dataset
DATASET_final= pd.concat([numerical_dataset, categorical_dataset], axis = 1)


******OR***********

# manually standardize numeric columns intead of using standard scaler

col_to_norm = ['col1', 'col2',.... 'col_n']
DATASET[col_to_norm]=DATASET[col_to_norm].apply(lambda x: (x-np.mean(x))/np.std(x))


StandardScaler : It transforms the data in such a manner that it has mean as 0 and
standard deviation as 1. In short, it standardizes the data. Standardization is useful
for data which has negative values. It arranges the data in normal distribution. It is
more useful in classification than regression.

Normalizer : It squeezes the data between 0 and 1. It performs normalization. Due to the
decreased range and magnitude, the gradients in the training process do not explode and
you do not get higher values of loss. Is more useful in regression than classification.
```

Learn more about transformers

https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py (https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py)

```
Assumption validation before starting

****MAKE SURE YOUR TARGET VALRIABLE IS CONTINIOUS*******

1. There should be a linear and additive relationship between dependent (response)
variable and independent (predictor) variable(s). A linear relationship suggests that a
change in response Y due to one unit change in X¹ is constant, regardless of the value of
X¹. An additive relationship suggests that the effect of X¹ on Y is independent of other
variables.

    CHECK WITH
#sns.pairplot(DATASET, size = 2, aspect = 1.5)

2. There should be no correlation between the residual (error) terms. Absence of this
phenomenon is known as Autocorrelation.


3. The independent variables should not be correlated. Absence of this phenomenon is
known as multicollinearity.

    CHECK WITH
corr = DATASET.corr()
plt.figure(figsize=(18,18))
sns.heatmap(corr,vmax=.8,linewidth=.01, square = True, annot =
True,cmap='YlGnBu',linecolor ='black')
plt.title('Correlation between features')

OR
```

```
sns.heatmap(DATASET.corr(), annot=True );

4. The error terms must have constant variance. This phenomenon is known as
homoskedasticity. The presence of non-constant variance is referred to
heteroskedasticity.

****For checking , we need to plot a scatter plot with residuals on y axis and predicted
value on x axis****



5. The error terms must be normally distributed.

*****For checking , we need to plot histogram of the residuals.***********


        If your TARGET VARIABLE is not normally distributed... LOG transform or SQUARE
ROOT Transform. I my personal experiance this method did not work left skew data became
right skew... check the article below which summaries risk of Log transform...
    https://stats.stackexchange.com/questions/130262/why-not-log-transform-all-variables-
that-are-not-of-main-interest
```

Linear Regression coding

1. Standardize features by removing the mean and scaling to unit standard deviation.

#EV = EXPLAINATORY VARIABLE

#TV = TARGET VARIABLE

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(DATASET)
DATASET_R = pd.DataFrame(scaler.transform(DATASET), index=DATASET.index,
columns=data.columns)
DATASET_R = pd.DataFrame(DATASET_R)
DATASET_R.head()
```

2. adding back the columns names

```
DATASET_R.columns = ['TV','EV1','EV2','EV3']
DATASET_R.head()
```

3. Preparing X and y

```
feature_cols = ['EV1', 'EV2', 'EV3']
X = DATASET_R[feature_cols]                          # use the list to select a subset of
the original DataFrame-+
y = DATASET_R.TV
```

4. Splitting X and y into training and test datasets.

```
from sklearn.model_selection import train_test_split  #old name cross_validation

def split(X,y):
    return train_test_split(X, y, test_size=0.20, random_state=1)

X_train, X_test, y_train, y_test=split(X,y)
print('Train cases as below')
print('X_train shape: ',X_train.shape)
print('y_train shape: ',y_train.shape)
print('\nTest cases as below')
print('X_test shape: ',X_test.shape)
print('y_test shape: ',y_test.shape)
```

5. Linear regression in scikit-learn

To apply any machine learning algorithm on your dataset, basically there are 4 steps:

1. Load the algorithm
2. Instantiate and Fit the model to the training dataset
3. Prediction on the test set
4. Calculating Root mean square error The code block given below shows how these steps are carried out:

```
from sklearn.linear_model import LinearRegression
    linreg = LinearRegression()
    linreg.fit(X_train, y_train)
    RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, y_pred_test))
```

```
def linear_reg( X, y, gridsearch = False):

    X_train, X_test, y_train, y_test = split(X,y)

    from sklearn.linear_model import LinearRegression
    linreg = LinearRegression()

    if not(gridsearch):
        linreg.fit(X_train, y_train)

    else:
        from sklearn.model_selection import GridSearchCV
        parameters = {'normalize':[True,False], 'copy_X':[True, False]}
        linreg = GridSearchCV(linreg,parameters, cv = 10,refit = True)
        linreg.fit(X_train, y_train)                              # fit the model to the
training data (learn the coefficients)
        print("Mean cross-validated score of the best_estimator : ", linreg.best_score_)


    y_pred_test = linreg.predict(X_test)
  # make predictions on the testing set

    RMSE_test = (metrics.mean_squared_error(y_test, y_pred_test))
# compute the RMSE of our predictions
    print('RMSE for the test set is {}'.format(RMSE_test))

    return linreg
```

6. Checking the RMSE

```
X = DATASET_R[feature_cols]
y = DATASET_R.TV
linreg = linear_reg(X,y)
```

7. Interpreting Model Coefficients

```
feature_cols.insert(0,'Intercept')
coef = linreg.coef_.tolist()
coef.insert(0, linreg.intercept_)
eq1 = zip(feature_cols, coef)
for c1,c2 in eq1:
    print(c1,c2)
```

Write your equation

y(TV) = 0.00116 + 0.7708 * EV1 + 0.508 * EV2 + 0.010 * EV3

Make initial interpretation 1. 2.

Important Notes:

- This is a statement of **association**, not **causation**.
- E.g. If an increase in television ad spending was associated with a **decrease** in sales, β1 would be **negative.**

8. Using the Model for Prediction

```
X_train, X_test, y_train, y_test = split(X,y)
y_pred_train = linreg.predict(X_train)
y_pred_test = linreg.predict(X_test)
print(y_pred_train)
print(y_pred_test)

#We need an evaluation metric in order to compare our predictions with the actual values.
```

9. Model evaluation

Error is the *deviation* of the values *predicted* by the model with the *true* values.
For example, if a model predicts that the price of apple is Rs75/kg, but the actual price of apple is Rs100/kg, then the error in prediction will be Rs25/kg.
Below are the types of error we will be calculating for our *linear regression model*:

- Mean Absolute Error
- Mean Squared Error
- Root Mean Squared Error

9.1 Model Evaluation using metrics.

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Computing the MAE for our TV predictions

```
MAE_train = metrics.mean_absolute_error(y_train, y_pred_train)
MAE_test = metrics.mean_absolute_error(y_test, y_pred_test)
print('MAE for training set is {}'.format(MAE_train))
print('MAE for test set is {}'.format(MAE_test))
```

9.2 *Mean Squared Error*(MSE) is the mean of the squared errors:__

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Computing the MSE for our TV predictions

```
MSE_train = metrics.mean_squared_error(y_train, y_pred_train)
MSE_test = metrics.mean_squared_error(y_test, y_pred_test)
print('MSE for training set is {}'.format(MSE_train))
print('MSE for test set is {}'.format(MSE_test))
```

9.3 Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Computing the RMSE for our TV predictions

```
RMSE_train = np.sqrt( metrics.mean_squared_error(y_train, y_pred_train))
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, y_pred_test))
print('RMSE for training set is {}'.format(RMSE_train))
print('RMSE for test set is {}'.format(RMSE_test))
```

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the **average error.**
- **MSE** is more popular than MAE, because MSE "punishes" larger errors.
- **RMSE** is even more popular than MSE, because RMSE is *interpretable* in the "y" units.
    - Easier to put in context as it's the same units as our response variable.

10. Model Evaluation using Rsquared value.

- There is one more method to evaluate linear regression model and that is by using the **Rsquared** value.
- R-squared is the **proportion of variance explained**, meaning the proportion of variance in the observed data that is explained by the model, or the reduction in error over the **null model**. (The null model just predicts the mean of the observed response, and thus it has an intercept and no slope.)
- R-squared is between 0 and 1, and higher is better because it means that more variance is explained by the model. But there is one shortcoming of Rsquare method and that is **R-squared will always increase as you add more features to the model**, even if they are unrelated to the response. Thus, selecting the

model with the highest R-squared is not a reliable approach for choosing the best linear model.

There is alternative to R-squared called **adjusted R-squared** that penalizes model complexity (to control for overfitting).

```
yhat = linreg.predict(X_train)
SS_Residual = sum((y_train-yhat)**2)
SS_Total = sum((y_train-np.mean(y_train))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y_train)-1)/(len(y_train)-X_train.shape[1]-1)
print("For Train data R-Square value is {} & adjusted R-Square values is
{}".format(r_squared, adjusted_r_squared))
```

```
yhat = linreg.predict(X_test)
SS_Residual = sum((y_test-yhat)**2)
SS_Total = sum((y_test-np.mean(y_test))**2)
r_squared = 1 - (float(SS_Residual))/SS_Total
adjusted_r_squared = 1 - (1-r_squared)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("For Test data R-Square value is {} & adjusted R-Square values is
{}".format(r_squared, adjusted_r_squared))
```

Whereas R-squared is a relative measure of fit, RMSE is an absolute measure of fit. As the square root of a variance, RMSE can be interpreted as the standard deviation of the unexplained variance, and has the useful property of being in the same units as the response variable. Lower values of RMSE indicate better fit. & R-square should be high

11. Feature Selection

At times some features do not contribute much to the accuracy of the model, in that case its better to discard those features.

- Let's check whether **dropping one of the EV** improve the quality of our predictions or not.
  To check this we are going to take all the features other than "EV" and see if the error (RMSE) is reducing or not.
- Also Applying **Gridsearch** method for exhaustive search over specified parameter values of estimator.

```
feature_cols = ['EV1','EV2']                # create a Python list of feature names AND
DROPPING EV3
X = DATASET_R[feature_cols]
y = DATASET_R.TV
linreg=linear_reg(X,y, gridsearch=True)
```

In [ ]:

```
feature_cols = ['EV1','EV2']                # create a Python list of feature names AND
X = DATASET_R[feature_cols]
y = DATASET_R.TV
linreg=linear_reg(X,y,)
```

In [ ]:

```
feature_cols.insert(0,'Intercept')
coef = linreg.coef_.tolist()
coef.insert(0, linreg.intercept_)

eq1 = zip(feature_cols, coef)

for c1,c2 in eq1:
    print(c1,c2)
```

** Then follow steps 7 to 10 **

12. Decision Tree Regresser

In [ ]:

```
from sklearn.tree import DecisionTreeRegressor
decisiontreereg = DecisionTreeRegressor()

decisiontreereg.fit(X_train, y_train)

y_pred_train_dtr = decisiontreereg.predict(X_train)
y_pred_test_dtr = decisiontreereg.predict(X_test)
```

** Then follow steps 7 to 10 **

13. Random Forest Regresser

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
randomforestreg = RandomForestRegressor()

randomforestreg.fit(X_train, y_train)

y_pred_train_rfr = randomforestreg.predict(X_train)
y_pred_test_rfr = randomforestreg.predict(X_test)
```

** Then follow steps 7 to 10 **

Note:

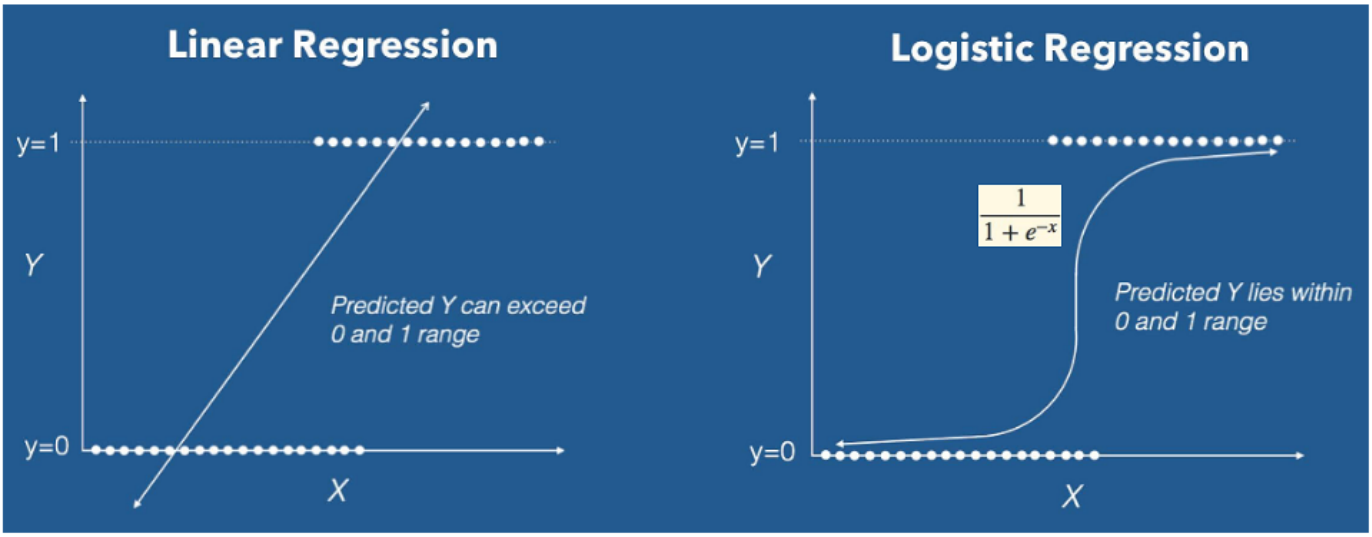
If there are categorical variables present use ONE HOT ENCODING - dummyfication ; Make sure you drop one of the feature to avoid dummy variable trap or multicollinearity

Logistic Regression

Logistic regression is a techinque used for solving the **classification problem**.
And Classification is nothing but a problem of **identifing** to which of a set of **categories** a new observation belongs, on the basis of *training dataset* containing observations (or instances) whose categorical membership is known.
For example to predict:
Whether an email is spam (1) or not (0) or,
Whether the tumor is malignant (1) or not (0)
Below is the pictorial representation of a basic logistic regression model to classify set of images into *happy or sad.*

Both Linear regression and Logistic regression are **supervised learning techinques**. But for the *Regression* problem the output is **continuous** unlike the *classification* problem where the output is **discrete**.

- Logistic Regression is used when the **dependent variable(target) is categorical**.
- **Sigmoid function** or logistic function is used as *hypothesis function* for logistic regression. Below is a figure showing the difference between linear regression and logistic regression, Also notice that logistic regression produces a logistic curve, which is limited to values between 0 and 1.



ASSUMPTIONS OF LOGISTIC REGRESSION

1. ASSUMPTION OF APPROPRIATE OUTCOME STRUCTURE To begin, one of the main assumptions of logistic regression is the appropriate structure of the outcome variable. Binary logistic regression requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal.

2. ASSUMPTION OF OBSERVATION INDEPENDENCE Logistic regression requires the observations to be independent of each other. In other words, the observations should not come from repeated measurements or matched data. ASSUMPTION OF THE ABSENCE OF MULTICOLLINEARITY Logistic regression requires there to be little or no multicollinearity among the independent variables. This means that the independent variables should not be too highly correlated with each other.

3. ASSUMPTION OF LINEARITY OF INDEPENDENT VARIABLES AND LOG ODDS Logistic regression assumes linearity of independent variables and log odds. Although this analysis does not require the dependent and independent variables to be related linearly, it requires that the independent variables are linearly related to the log odds.

4. ASSUMPTION OF A LARGE SAMPLE SIZE Finally, logistic regression typically requires a large sample size. A general guideline is that you need at minimum of 10 cases with the least frequent outcome for each independent variable in your model. For example, if you have 5 independent variables and the expected probability of your least frequent outcome is .10, then you would need a minimum sample size of 500 (10*5 / .10).

Logistic regression is quite different than linear regression in that it does not make several of the key assumptions that linear and general linear models (as well as other ordinary least squares algorithm based models) hold so close:

> - Logistic regression does not require a linear relationship between the dependent and independent variables,
> - The error terms (residuals) do not need to be normally distributed,
> - Homoscedasticity is not required, and
> - The dependent variable in logistic regression is not measured on an interval or ratio scale.

_source # https://www.lexjansen.com/wuss/2018/130_Final_Paper_PDF.pdf (https://www.lexjansen.com/wuss/2018/130_Final_Paper_PDF.pdf)

- Clean, Scaler Transform & Dummify Data Before you start

```
Comparision plots can be used for visual representation
1. pairplot
2. Corr Plot
```

Logistic Regression Coding

1. Preparing X and y using pandas

```
X = DATASET.loc[:,DATASET.columns != 'TV']         #TV is target variable
y = DATASET.TV
```

2. Splitting X and y into training and test datasets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
print(X_train.shape)
print(y_train.shape)
```

3. Logistic regression in scikit-learn

To apply any machine learning algorithm on your dataset, basically there are 4 steps:

1. Load the algorithm
2. Instantiate and Fit the model to the training dataset
3. Prediction on the test set
4. Calculating the accuracy of the model

The code block given below shows how these steps are carried out:

```
from sklearn.linear_model import LogisticRegression
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)
    accuracy_score(y_test,y_pred_test))
```

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train,y_train)
```

4. Using the Model for Prediction

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1
y_pred_train = logreg.predict(X_train)
y_pred_test = logreg.predict(X_test)
```

5. Model evaluation

Error is the *deviation* of the values *predicted* by the model with the *true* values.
We will use **accuracy score __ and __confusion matrix** for evaluation.

5.1 Model Evaluation using accuracy classification score

```
from sklearn.metrics import accuracy_score
print('Accuracy score for test data is:', accuracy_score(y_test,y_pred_test))
```

5.2 Model Evaluation using confusion matrix

to learn more about Confusion Matrix - please check [https://github.com/omsarmalkar/Machine-Learning/tree/Confusion-Matrix (https://github.com/omsarmalkar/Machine-Learning/tree/Confusion-Matrix)](https://github.com/omsarmalkar/Machine-Learning/tree/Confusion-Matrix)

```
from sklearn.metrics import confusion_matrix

confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

print(confusion_matrix)
```

```
confusion_matrix.index = ['Actual __NO(0)_','Actual _YES(1)__']
confusion_matrix.columns = ['Predicted __NO(0)_','Predicted _YES(1)__']
print(confusion_matrix)
```

6. Adjusting Threshold

- We have used, **.predict** method for classification. This method takes 0.5 as the default threshhod for prediction.
- Now, we are going to see the impact of changing threshold on the accuracy of our logistic regression model.
- For this we are going to use **.predict_proba** method instead of using .predict method.

```
preds1 = np.where(logreg.predict_proba(X_test)[:,1]> 0.75,1,0)        #Threshold to 0.75
~ 75%
print('Accuracy score for test data is:', accuracy_score(y_test,preds1))
```

```
preds2 = np.where(logreg.predict_proba(X_test)[:,1]> 0.25,1,0)         #Threshold to 0.25
~ 25%
print('Accuracy score for test data is:', accuracy_score(y_test,preds2))
```

```
```

Decision Tree

A **decision tree** is one of most frequently and widely used supervised machine learning algorithms that can perform both **regression and classification tasks.**
The intuition behind the decision tree algorithm is simple, yet also very powerful.

Everyday we need to make numerous **decisions**, many smalls and a few big.
So, Whenever you are in a dilemna, if you'll keenly observe your thinking process. You'll find that, you are unconsciously using **decision tree approcah** or you can also say that decision tree approach is based on our thinking process.

- A decision tree **split the data into multiple sets**.Then each of these sets is further split into subsets to arrive at a **decision**.
- It is a very natural decision making process asking a series of question in a nested if then else statement.
- On each node you ask a question to further split the data held by the node.

So, lets understand what is a decision tree with a help of a real life example.

Consider a scenario where a person asks you to lend them your car for a day, and you have to make a decision whether or not to lend them the car. There are several factors that help determine your decision, some of which have been listed below:

1. **Is this person a close friend or just an acquaintance?**
    - If the person is just an acquaintance, then decline the request;
    - if the person is friend, then move to next step.

2. **Is the person asking for the car for the first time?**
    - If so, lend them the car,
    - otherwise move to next step.
3. **Was the car damaged last time they returned the car?**
    - If yes, decline the request;
    - if no, lend them the car.

The structure of decision tree resembles an **upside down tree**, with its roots at the top and braches are at the bottom. The end of the branch that doesnt split any more is the decision or leaf.

Now, lets see what is **Decision tree algorithm.**
Decision tree is a type of **supervised learning algorithm** (having a pre-defined target variable) that is mostly used in classification problems.

- It works for both **categorical and continuous** input and output variables.
- In this technique, we **split the population** or sample into two or more homogeneous sets (or sub-populations) based on most **significant splitter / differentiator** in input variables.

Important Terminology related to Decision Trees

Let's look at the basic terminology used with Decision trees:

- **Root Node**:
  It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting**:
  It is a process of dividing a node into two or more sub-nodes.
- **Decision Node**:
  When a sub-node splits into further sub-nodes, then it is called decision node.
- **Leaf/ Terminal Node**:
  Nodes do not split is called Leaf or Terminal node.
- **Pruning**:
  When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
- **Branch / Sub-Tree**:
  A sub section of entire tree is called branch or sub-tree.
- **Parent and Child Node**:
  A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

Types of Decision Trees¶

Types of decision tree is based on the **type of target variable** we have. It can be of two types:

- **Categorical Variable Decision Tree**:
    - Decision Tree which has **categorical target variable** then it called as categorical variable decision tree.
- **Continuous Variable Decision Tree**:
    - Decision Tree has **continuous target variable** then it is called as Continuous Variable Decision Tree.

Example:

- Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (**Yes/ No**).

company (Yes/ No).
For this we are predicting values for categorical variable. So, the decision tree approach that will be used is **Categorical Variable Decision Tree.**

- Now, suppose insurance company does not have income details for all customers. But, we know that this is an important variable, then we can build a decision tree to predict customer income based on occupation, product and various other variables.
In this case, we are predicting values for continuous variable. So , This approach is called **Continuous Variable Decision Tree**.

Concept of Homogenity

Homogenous populations are **alike** and **heterogeneous** populations are **unlike**.

- A heterogenous population is one where individuals are **not similar** to one another.
- For example, you could have a heterogenous population in terms of humans that have migrated from different regions of the world and currently live together. That population would likely be heterogenous in regards to height, hair texture, disease immunity, and other traits because of the varied background and genetics.

Note: In real world you would never get this level of homogeniety. So out of the hetrogenous options you need to select the one having maximum homoginiety. To select the feature which provide maximum homoginety we use **gini & entropy** techniques.

What Decision tree construction algorithm will try to do is to **create a split in such a way that the homogeneity of different pieces must be as high as possible.**

Let's say we have a sample of **30 students** with three variables:

1. Gender (Boy/ Girl)
2. Class (IX/ X) and,
3. Height (5 to 6 ft).

15 out of these 30 play cricket in leisure time. Now, I want to **create a model to predict who will play cricket during leisure period**? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.

This is where decision tree helps, it will segregate the students based on all values of three variables and identify the variable, which creates the best homogeneous sets of students (which are heterogeneous to each other). In the snapshot below, you can see that variable **Gender** is able to identify best homogeneous sets compared to the other two variables.

As mentioned above, decision tree identifies the most significant variable and it's value that gives best homogeneous sets of population. Now the question which arises is, how does it identify the variable and the split? To do this, decision tree uses various algorithms, which we will shall discuss in the following section.

How does a tree decide where to split?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria is different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable. Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on type of target variables. Let's look at the most commonly used algorithms in decision tree:

Gini Index

4.5.1 Gini Index

Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

- It works with categorical target variable "Success" or "Failure".
- It performs only Binary splits
- Higher the value of Gini higher the homogeneity.
- CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure ($1 - p^2 - q^2$).
2. Calculate Gini for split using weighted Gini score of each node of that split

Example: – Referring to example used above, where we want to segregate the students based on target variable (playing cricket or not). In the snapshot below, we split the population using two input variables Gender and Class. Now, I want to identify which split is producing more homogeneous sub-nodes using Gini index.

Gini for Root node:

- 1 - (0.5 * 0.5) - (0.5 * 0.5) = 0.50

Split on Gender:

1. Gini for sub-node **Female**
    - 1 - (0.2 * 0.2) - (0.8 * 0.8) = 0.32
2. Gini for sub-node **Male**
    - 1 - (0.65 * 0.65) - (0.35 * 0.35) = 0.45
3. Weighted Gini for Split **Gender**
    - (10/30) * 0.32 + (20/30) * 0.45 = 0.41

Split on Class :

1. Gini for sub-node **Class IX** =
    - 1 - (0.43 * 0.43) - (0.57 * 0.57) = 0.49
2. Gini for sub-node **Class X** =
    - 1 - (0.56 * 0.56) - (0.44 * 0.44) = 0.49
3. Calculate weighted Gini for Split **Class**
    - (14/30) * 0.51 + (16/30) * 0.51 = 0.49

Above, you can see that: **Gini score** for Split on **Gender** < Gini score for Split on **Class**.
Also, **Gini score** for **Gender** < Gini score for **root node**.
Hence, the **node split will take place on Gender.**

Information Gain:

Look at the image below and think which node can be described easily.
I am sure, your answer is C because it requires less information as all values are similar. On the other hand, B requires more information to describe it and A requires the maximum information.
In other words, we can say that **C is a Pure node, B is less Impure and A is more impure.**

Now, we can build a conclusion that:

- less impure node requires less information to describe it.
- more impure node requires more information.

Information theory is a measure to define this degree of disorganization in a system by a parameter known as **Entropy**.

- If the sample is completely **homogeneous**, then the **entropy is zero** and
- If the sample is an **equally divided** (50% – 50%), it has **entropy of one**.

Entropy can be calculated using formula: where,
p__ & __q is **probability of success and failure** respectively in that node.

- **Information Gain = 1 - Entropy**.
- The model will choose the split which facilitates **maximum information gain**, which in turn means **minimum Entropy**.
- So, it chooses the split which has **lowest entropy** compared to parent node and other splits.
- **The lesser the entropy, the better it is.**

Steps to calculate entropy for a split:

1. Calculate entropy of parent node
2. Calculate entropy of each individual node of split and
3. Calculate weighted average of all sub-nodes available in split.
4. Caluclate the Information Gain in various split options w.r.t parent node
5. Choose the split with highest Information Gain.

Example: Let's use this method to identify best split for student example.

- **Entropy for parent node**
  - - (15/30) log2 (15/30) − (15/30) log2 (15/30) = 1.
    Here 1 shows that it is a impure node.
- **Entropy for Female node**
  - - (2/10) log2 (2/10) − (8/10) log2 (8/10) = 0.72
- **Entropy for male node**
  - - (13/20) log2 (13/20) − (7/20) log2 (7/20) = 0.93
- **Entropy for split Gender** = Weighted entropy of sub - nodes
  - (10/30) * 0.72 + (20/30) * 0.93 = 0.86

- **Information Gain for split Gender** = Entropy of Parent Node - Weighted entropy for Split Gender

- 1 - 0.86 = 0.14

- **Entropy for Class IX node**,
  - - (6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99
- **Entropy for Class X node**,
  - - (9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
- **Entropy for split Class**,
  - (14/30) * 0.99 + (16/30) * 0.99 = 0.99

- **Information Gain for split Class** = Entropy of Parent Node - Weighted entropy for Split Class
  - 1 - 0.99 = 0.01

Observe that:
Information Gain for Split on Gender > Information Gain for Split on Class,
So, **the tree will split on Gender.**

Advantages of using Decision Tree

- **Easy to Understand**:
  - Decision tree output is very easy to understand even for people from non-analytical background. It does not require any statistical knowledge to read and interpret them.
  - Its graphical representation is very intuitive and users can easily relate their hypothesis.
- **Less data cleaning required**:
  - It requires less data cleaning compared to some other modeling techniques.
  - It is not influenced by outliers and missing values to a fair degree.
- **Data type is not a constraint**:
  - It can handle both numerical and categorical variables.
- **Non Parametric Method**:
  - Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

Shortcomings of Decision Trees

- **Over fitting**:
  - Over fitting is one of the most practical difficulty for decision tree models. This problem gets solved by setting constraints on model parameters and pruning (discussed in detailed below).
- **Not a great contributor for regression**:
  - While working with continuous numerical variables, decision tree looses information when it categorizes variables in different categories.

- Clean, Scaler Transform & Dummify Data Before you start

1. Preparing X & y

```
X = DATASET.loc[:,DATASET.columns != 'rating']
y = dt.rating
```

2. Splitting X and y into training and test datasets.__

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
print(X_train.shape)
print(y_train.shape)
```

3. Decision Tree in scikit-learn

To apply any machine learning algorithm on your dataset, basically there are 4 steps:

1. Load the algorithm
2. Instantiate and Fit the model to the training dataset
3. Prediction on the test set
4. Calculating the accuracy of the model

The code block given below shows how these steps are carried out:

```
from sklearn import tree
    model = tree.DecisionTreeClassifier(criterion='gini')
    model.fit(X, y)
    predicted= model.predict(x_test)
```

```
IN []
from sklearn import tree
model = tree.DecisionTreeClassifier(random_state = 0)
model.fit(X_train, y_train)


OUT []
# DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
#             max_features=None, max_leaf_nodes=None,
#             min_impurity_decrease=0.0, min_impurity_split=None,
#             min_samples_leaf=1, min_samples_split=2,
#             min_weight_fraction_leaf=0.0, presort=False, random_state=0,
#             splitter='best')
```

Instantiate Decision Tree Classifer using scikit learn having (criterion='entropy', max_leaf_nodes=10, max_depth=3, min_samples_split=5, min_samples_leaf=4).

In [ ]:

```
dtree1 = DecisionTreeClassifier(criterion = 'entropy', max_leaf_nodes=10, max_depth=3, m
dtree1.fit(X_train,y_train)
```

```
#code for Graphviz
import sys
!{sys.executable} -m pip install graphviz
!{sys.executable} -m pip install pydotplus
!{sys.executable} -m pip install Ipython
```

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/bin/'  #where
file is stored
```

4. Plot Decision Tree

```
import pydotplus
from IPython.display import Image
dot_tree = tree.export_graphviz(model, out_file=None,filled=True, rounded=True,
                                special_characters=True, feature_names=X.columns)
graph = pydotplus.graph_from_dot_data(dot_tree)

Image(graph.create_png())
```

5. Using the Model for Prediction

```
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)
```

6. Model Evaluation using accuracy_score

```
from sklearn.metrics import accuracy_score
print('Accuracy score for test data is:', accuracy_score(y_test,y_pred_test))
```

7. Model Evaluation using confusion matrix

```
from sklearn.metrics import confusion_matrix

confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

confusion_matrix.index = ['Actual __NO(0)_','Actual _YES(1)__']
confusion_matrix.columns = ['Predicted __NO(0)_','Predicted _YES(1)__']
confusion_matrix
```

8. Decision Tree with Gridsearch

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

decision_tree_classifier = DecisionTreeClassifier(random_state = 0)


tree_para = [{'criterion':['gini','entropy'],'max_depth': range(2,60),     #Gini or entr
                            'max_features': ['sqrt', 'log2', None] }]



grid_search = GridSearchCV(decision_tree_classifier,tree_para, cv=10, refit='AUC')
grid_search.fit(X_train, y_train)
```

8.1. Using the model for predictior

```
y_pred_test1 = grid_search.predict(X_test)
```

8.2. Model Evaluation using accuracy_score

```


from sklearn.metrics import accuracy_score
print('Accuracy score for test data is:', accuracy_score(y_test,y_pred_test1))

from sklearn.metrics import confusion_matrix

confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test1))
```

8.3. Model Evaluation using confusion matrix

```
confusion_matrix.index = ['Actual __NO(0)_','Actual _YES(1)__']
confusion_matrix.columns = ['Predicted __NO(0)_','Predicted _YES(1)__']
confusion_matrix
```

Random Forest

Random Forest is considered to be the **panacea** of all data science problems. On a funny note, when you can't think of any algorithm (irrespective of situation), use random forest!

In Random Forest, we grow **multiple trees** as opposed to a single tree in CART model . To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. **The forest chooses the classification having the most votes** (over all the trees in the forest) and in case of **regression**, it takes the **average of outputs by different trees.**

Random Forest is a versatile machine learning method capable of performing **both regression and classification tasks**. It also undertakes dimensional reduction methods, treats missing values, outlier values and other essential steps of data exploration, and does a fairly good job. It is a type of **ensemble learning** method, where **a group of weak models combine to form a powerful model.**

Real Life Analogy:

Imagine a guy named Andrew, that want's to decide, to which places he should travel during a one-year vacation trip. He asks people who know him for advice. First, he goes to a friend, tha asks Andrew where he traveled to in the past and if he liked it or not. Based on the answers, he will give Andrew some advice.

This is a typical **decision tree algorithm approach**. Andrews friend created rules to guide his decision about what he should recommend, by using the answers of Andrew.

Afterwards, Andrew starts asking more and more of his friends to advise him and they again ask him different questions, where they can derive some recommendations from. Then he chooses the places that where recommend the most to him, which is the typical **Random Forest algorithm approach**.

Wisdom of Crowd

"The Wisdom of Crowds" is an idea, summarized in the 2004 book by **James Surowiecki** by the same name, which states that **the aggregate information in a group often leads to a better decision than any single member of the group.**

- It's something that's been empirically *observed* in many different areas of *social science*, and if some basic initial conditions are met, it usually holds up pretty well in the real world.
- The premise is this – if you take a large enough group of people, all with independent judgments, and all with access to different levels and amounts of information, the overall group's average judgment is usually better than any single individual judgment.
- There have been many famous cases of Wisdom of Crowds at play – from guessing the weight of an ox at a county fair to asking the audience in the popular game show, **Who Wants To Be A Millionaire**.

If you are running a Random Forest model in your job or class, and you sift through some of the statistics and mathematics behind it, you are in effect applying some of the core concepts of the Wisdom of Crowds in your work. In his book, James Surowiecki lays out some of the basic elements that are required for the Wisdom of Crowds to work. Here's how it compares to what Random Forest is actually doing.

	Wisdom of Crowds Theory	Random Forest
Basic Unit	Individual	Decision Tree
Diversity of Opinion	Each person should have private information	Each decision tree is built off of a randomly selected subset of the data, and each time the tree has to choose a split, it is allowed to only select from a randomly selected subset of predictors. Therefore, each tree is built based on completely different information from every other tree (ie, private information)
Independence	People's opinions aren't determined by the opinions of those around them.	By utilizing different training sets and randomly selecting the subset of predictors at each split, the algorithm ensures that each tree is independent from every other. The later actually has the effect of *decorrelating* the trees
Decentralization	People are able to specialize and draw on local knowledge	Inherent in the fact that each tree is built with different training data and different predictors to choose from at each split
Aggregation	Some mechanism exists to turn private judgments into a collective decision	The last step of the algorithm is to take either the mean (regression) or mode (classification)

- Statistically speaking, Random Forest is simply trying to **reduce the variance in prediction by averaging a large number of independent, uncorrelated, individual predictions**
- Philosophically, Random Forest is simply applying many of the ideas behind the **Wisdom of Crowds**.

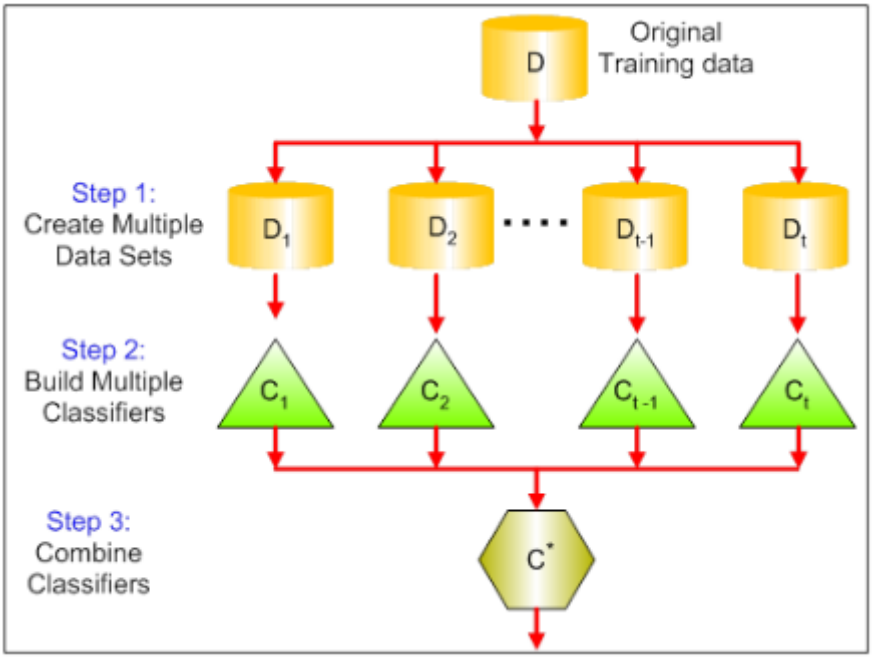
Concept behind random forest

The random forest is a model made up of many decision trees. Rather than just being a forest though, this model is random because of two concepts:

1. Random sampling of data points
2. Splitting nodes based on subsets of features

Random Sampling

- One of the keys behind the random forest is that **each tree trains on random samples** of the data points.
- The samples are drawn with *replacement* (known as **bootstrapping**) which means that some samples will be trained on in a single tree multiple times (we can also disable this behavior if we want).
- The idea is that by training each tree on different samples, although **each tree** might have **high variance** with respect to a particular set of the training data, overall, the **entire forest __will have __low variance**.
- This procedure of training each individual learner on different subsets of the data and then averaging the predictions is known as **bagging**, short for bootstrap aggregating.



To more clearly understand bagging summarised below are the steps to follow:

1. **Create Multiple DataSets**:
    - Sampling is done with replacement on the original data and new datasets are formed.
    - The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model
    - Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
2. **Build Multiple Classifiers**:
    - Classifiers are built on each data set.
    - Generally the same classifier is modeled on each data set and predictions are made.
3. **Combine Classifiers**:
    - The predictions of all the classifiers are combined using either mean or mode value depending on the problem at hand.
    - Generally **mean** are used for **regression** problems and **mode** is used for **classification** problems.
    - The combined values are generally more robust than a single model.

Random Subsets of Features

- Another concept behind the random forest is that only a **subset** of all the **features** are considered for splitting each node in each decision tree. Generally this is set to **sqrt(n_features)** meaning that at each node, the decision tree considers splitting on a sample of the features totaling the square root of the total number of features.
- The random forest *can* also be trained considering **all the features** at every node. (These options can be controlled in the Scikit-Learn random forest implementation).

The random forest combines hundreds or **thousands of decision trees**, trains each one on a **slightly different set of the observations** (sampling the data points with replacement) and also **splits nodes in each tree considering only a limited number of the features.** The final predictions made by the random forest are made by **averaging the predictions of each individual tree.**

Advantages and Disadvantages:

Advantages

- It can be used for **both regression and classification** tasks and that it's easy to view the relative importance it assigns to the input features.
- Random forest classifier **handle the missing values** on its own.
- Random Forest is also considered as a very handy and easy to use algorithm, because it's **default hyperparameters often produce a good prediction result**. The number of hyperparameters is also not that high and they are straightforward to understand.
- One of the big problems in machine learning is overfitting, but most of the time this won't happen that easy to a random forest classifier. That's because if there are **enough trees** in the forest, the classifier **won't overfit** the model.

Shortcomings

- The main limitation of Random Forest is that a **large number of trees** can make the algorithm **slow and ineffective for real-time predictions.** In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.
- Random Forest is a predictive modeling tool and **not a descriptive tool**. That means, if you are looking for a description of the relationships in your data, other approaches would be preferred.
- Random Forest can feel like a **black box approach for statistical modelers** – you have **very little control** on what the model does. You can at best – try different parameters and random seeds!

Use Cases:

The random forest algorithm is used in a lot of different fields like:

- **Banking**
  - In Banking it is used for example to detect customers who will use the bank's services more frequently than others and repay their debt in time. In this domain it is also used to detect fraud customers who want to scam the bank.
- **Stock Market**,
  - In finance, it is used to determine a stock's behaviour in the future.
- **Medicine**
  - In the healthcare domain it is used to identify the correct combination of components in medicine and to analyze a patient's medical history to identify diseases.
- **E-Commerce**

- To determine whether a customer will actually like the product or not.

- Clean, Scaler Transform & Dummify Data Before you start

Random Forest Coding

1. Prepare X & y

```
X = DATASET.loc[:,DATASET.columns != 'rating']
y = dt.rating
```

2. Splitting X and y into training and test datasets.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
print(X_train.shape)
print(y_train.shape)
```

3. Random Forest in scikit-learn

To apply any machine learning algorithm on your dataset, basically there are 4 steps:

1. Load the algorithm
2. Instantiate and Fit the model to the training dataset
3. Prediction on the test set
4. Calculating the accuracy of the model

The code block given below shows how these steps are carried out:

```
from sklearn.ensemble import RandomForestClassifier
    model = RandomForestClassifier()
    model.fit(X, y)
    predicted= model.predict(x_test)
```

- **Model without parameter specification**

```
IN[]

from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(random_state = 0)
model.fit(X_train, y_train)

OUT[]

# RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
#             max_depth=None, max_features='auto', max_leaf_nodes=None,
#             min_impurity_decrease=0.0, min_impurity_split=None,
#             min_samples_leaf=1, min_samples_split=2,
```

```
#             min_weight_fraction_leaf=0.0, n_estimators=15, n_jobs=1,
#             oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Instantitate Random Forest Model using scikit learn having (criterion='entropy',n_estimators = 100, random_state = 0, max_depth = 2, min_samples_split=4, min_samples_leaf=3, max_leaf_nodes=5).

rfc_new = RandomForestClassifier(criterion='entropy',n_estimators = 100, random_state = 0, max_depth = 2, min_samples_split=4, min_samples_leaf=3, max_leaf_nodes=5) rfc_new.fit(X_train,y_train)

4. Using the Model for Prediction

```
y_pred_train = model.predict(X_train)
y_pred_train1 = model1.predict(X_train)
y_pred_test = model.predict(X_test)                                    # make predictions
on the testing set
y_pred_test1 = model1.predict(X_test)
```

5. Model evaluation

Error is the *deviation* of the values *predicted* by the model with the *true* values.
We will use **accuracy score __ and __confusion matrix** for evaluation.

5.1 Model Evaluation using accuracy_score

```
from sklearn.metrics import accuracy_score
print('Accuracy score for test data using the model without parameter specification:',
accuracy_score(y_test,y_pred_test))
print('Accuracy score for test data using the model with parameter specification:',
accuracy_score(y_test,y_pred_test1))
```

5.2 Model Evaluation using confusion matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

confusion_matrix.index = ['Actual __NO(0)_','Actual _YES(1)__']
confusion_matrix.columns = ['Predicted __NO(0)_','Predicted _YES(1)__']
confusion_matrix
```

6. Random forest with RandomizedsearchCV

Applying __RandomizedsearchCV__ method for __exhaustive search over specified parameter values__ of estimator.
To know more about the different parameters in random forest classifier, refer the [documentation](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html).
Below we will apply gridsearch over the following parameters: - criterion - max_depth - n_estimators - min_samples_split - min_samples_leaf

You can change other parameters also and compare the impact of it via calculating **accuracy score & confusion matrix**

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
# parameters for GridSearchCV
# specify parameters and distributions to sample from
param_dist = {"max_depth": range(2,5),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "n_estimators": [100, 400, 700, 1000, 1500],
              "criterion" : ["gini", "entropy"],
              'max_features': ['sqrt', 'log2', None]
             }
# run randomized search
n_iter_search = 50
random_search = RandomizedSearchCV(model, param_distributions = param_dist,
                                   n_iter = n_iter_search,
                                   n_jobs = -1)
```

```
random_search.fit(X_train, y_train)
```

6.1 Using the model for prediction

```
y_pred_test1 = random_search.predict(X_test)
```

6.2 Model Evaluation using accuracy_score

```
from sklearn.metrics import accuracy_score
print('Accuracy score on test data with RandomizedSearchCV is:',
accuracy_score(y_test,y_pred_test1))
```

6.3 Model Evaluation using confusion matrix

```
from sklearn.metrics import confusion_matrix
confusion_matrix = pd.DataFrame(confusion_matrix(y_test, y_pred_test))

confusion_matrix.index = ['Actual __NO(0)_','Actual _YES(1)__']
confusion_matrix.columns = ['Predicted __NO(0)_','Predicted _YES(1)__']
confusion_matrix
```

```
predictions = classifier.predict(X_test)
# True Positive (TP): we predict a label of 1 (positive), and the true label is 1.
TP = np.sum(np.logical_and(predictions == 1, y_test == 1))

# True Negative (TN): we predict a label of 0 (negative), and the true label is 0.
TN = np.sum(np.logical_and(predictions == 0, y_test == 0))

# False Positive (FP): we predict a label of 1 (positive), but the true label is 0.
FP = np.sum(np.logical_and(predictions == 1, y_test == 0))

# False Negative (FN): we predict a label of 0 (negative), but the true label is 1.
FN = np.sum(np.logical_and(predictions == 0, y_test == 1))
```

```
print('TP: {}, FP: {}, TN: {}, FN: {}'.format(TP,FP,TN,FN))
```

Miscellaneous

In [ ]:

```
from sklearn.metrics import roc_curve, auc, classification_report, confusion_matrix, acc
```

Gridsearch

```
from sklearn.model_selection import GridSearchCV
def gridsearch(model, params):
    gs = GridSearchCV(model, params, scoring='roc_auc', n_jobs=-1)
    gs.fit(X_train, y_train)
    print ('Best params: ', gs.best_params_)
    print ('Best auc on training set: ', gs.best_score_)
    print ('Best auc on test set: ', gs.score(X_test, y_test))
    return gs.predict(X_test), gs.decision_function(X_test)
```

Gradient descent

#DO NOT USE TESTING IN PROGRESS AS ITS THERE IS SOME DELAY WITH KERNAL (UPDATED SEPT 12 BY OM SARMALKAR)

```
# try using stochastic gradient descent with logistic loss function
# specify lasso regularization to select features and address multicollinearity issues

from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier(loss='log', penalty='l1', learning_rate='optimal')

# use grid search to optimize parameters
sgd_params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1.0, 5.0], 'class_weight': [None,
'balanced']}

sgd_pred, sgd_prob = gridsearch(sgd, sgd_params)
```

Accuracy score

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
sgd_params = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1.0, 5.0], 'class_weight': [None,
'balanced']}
sgd_pred, sgd_prob = gridsearch(sgd, sgd_params)
sgd = SGDClassifier(loss='log', penalty='l1', learning_rate='optimal', alpha=0.001)
print ('accuracy score on training set: ', cross_val_score(sgd, X_train, y_train,
n_jobs=-1).mean())
print ('accuracy score on testing set: ', accuracy_score(sgd_pred, y_test))
```

Classification report

```
print(classification_report(y_test, sgd_pred, target_names=['not default', 'default']))
```

Confusion Matrix

```
def plot_confusion(prediction):
    conmat = np.array(confusion_matrix(y_test, prediction, labels=[1,0]))
    confusion = pd.DataFrame(conmat, index=['default', 'not default'],
                             columns=['predicted default', 'predicted not default'])
    return confusion
```

```
plot_confusion(sgd_pred)
```

```
from sklearn.tree import DecisionTreeClassifier   #or use
classifier = DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=20,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=20,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
classifier.fit(X_train, y_train)
predictions = classifier.predict(X_test)
accuracy_score(y_true = y_test, y_pred = predictions)
```

```
predictions = classifier.predict(X_test)
# True Positive (TP): we predict a label of 1 (positive), and the true label is 1.
TP = np.sum(np.logical_and(predictions == 1, y_test == 1))

# True Negative (TN): we predict a label of 0 (negative), and the true label is 0.
TN = np.sum(np.logical_and(predictions == 0, y_test == 0))

# False Positive (FP): we predict a label of 1 (positive), but the true label is 0.
FP = np.sum(np.logical_and(predictions == 1, y_test == 0))

# False Negative (FN): we predict a label of 0 (negative), but the true label is 1.
FN = np.sum(np.logical_and(predictions == 0, y_test == 1))

print('TP: {}, FP: {}, TN: {}, FN: {}'.format(TP,FP,TN,FN))
```

ROC Curve

```
from sklearn.metrics import roc_curve, auc
def plot_roc(prob):
    y_score = prob
```

```
    fpr = dict()
    tpr = dict()
    roc_auc=dict()
    fpr[1], tpr[1], _ = roc_curve(y_test, y_score)
    roc_auc[1] = auc(fpr[1], tpr[1])

    plt.figure(figsize=[9,7])
    plt.plot(fpr[1], tpr[1], label='Roc curve (area=%0.2f)' %roc_auc[1], linewidth=4)
    plt.plot([1,0], [1,0], 'k--', linewidth=4)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('false positive rate', fontsize=18)
    plt.ylabel('true positive rate', fontsize=18)
    plt.title('ROC curve for credit default', fontsize=18)
    plt.legend(loc='lower right')
    plt.show()
```

```
# plot roc curve and calculate auc
plot_roc(sgd_prob)
```

PCA - Principle Component Analysis

Introduction to Principal Component Analysis

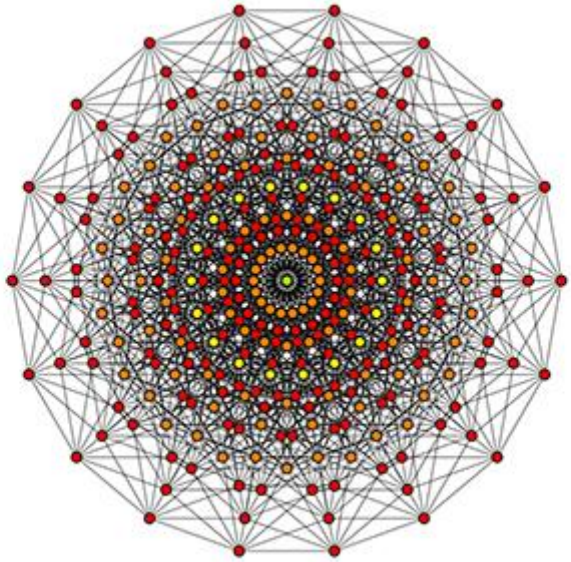
What is Data Dimensionality?

In real world, **number of columns** is the number of **dimensions of data**.However, some columns are **similar**, some are **correlated**, some are **duplicates** in some way, some are **junk**, some are **useless**, etc. so the actual number of dimensions can be unknown. Its a knotty problem.

What is high dimensionality?

Suppose we have **500 variables** in a data set. As it a very huge number, so it is quite difficult to read and understand the data. This is known as high dimensionality. Anything which **can't be read and understood without any use of external resources** is an example of high dimensionality.

Lost in high dimensional space:

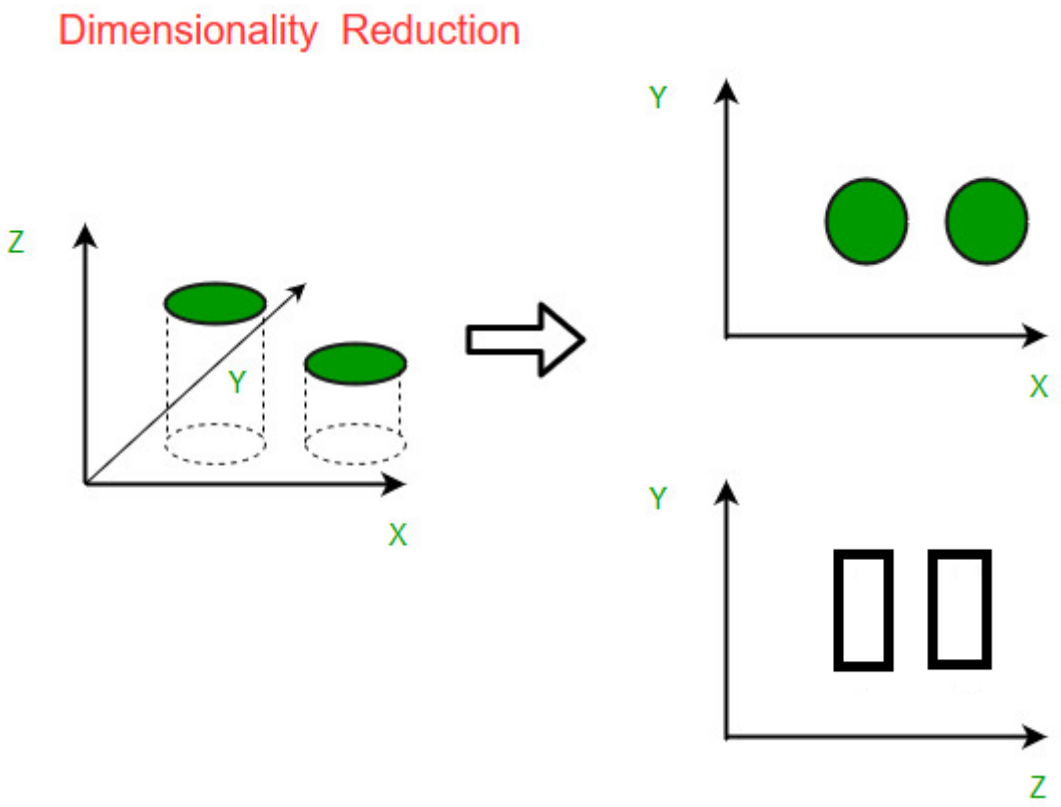


MOTIVATION

When dealing with real problems and real data we often deal with **high dimensional** data that can go up to **millions**.

- Sometime we might need to deal with data having large number of columns/variables, so we need to **reduce its dimensionality**.
- The need to reduce dimensionality is often **associated with visualizations** (reducing to 2–3 dimensions so we can plot it) but *that is not always the case*.
- Sometimes we might value **performance over precision** so we could reduce *1,000 dimensional data to 10 dimensions so we can manipulate it faster* (eg. calculate distances).
- Find essential **attributes/variables**.

The need to reduce dimensionality at times is **real and has many applications**. For the same, there are **various techinques**.
This sheet is entirely focused on **PCA(Principal Component Analysis)**



The picture above explains a simple dimension reduction in which a **3-D** figure is compressed to a **2-D** figure. This helps in better **visualisation** and better **understanding** of data points.

PRINCIPAL COMPONENT ANALYSIS

> *Too many variables? Should you be using all possible variables to generate model?*

In order to handle **'curse of dimensionality'** and avoid issues like **over-fitting** in high dimensional space, methods like **Principal Component analysis** is used.

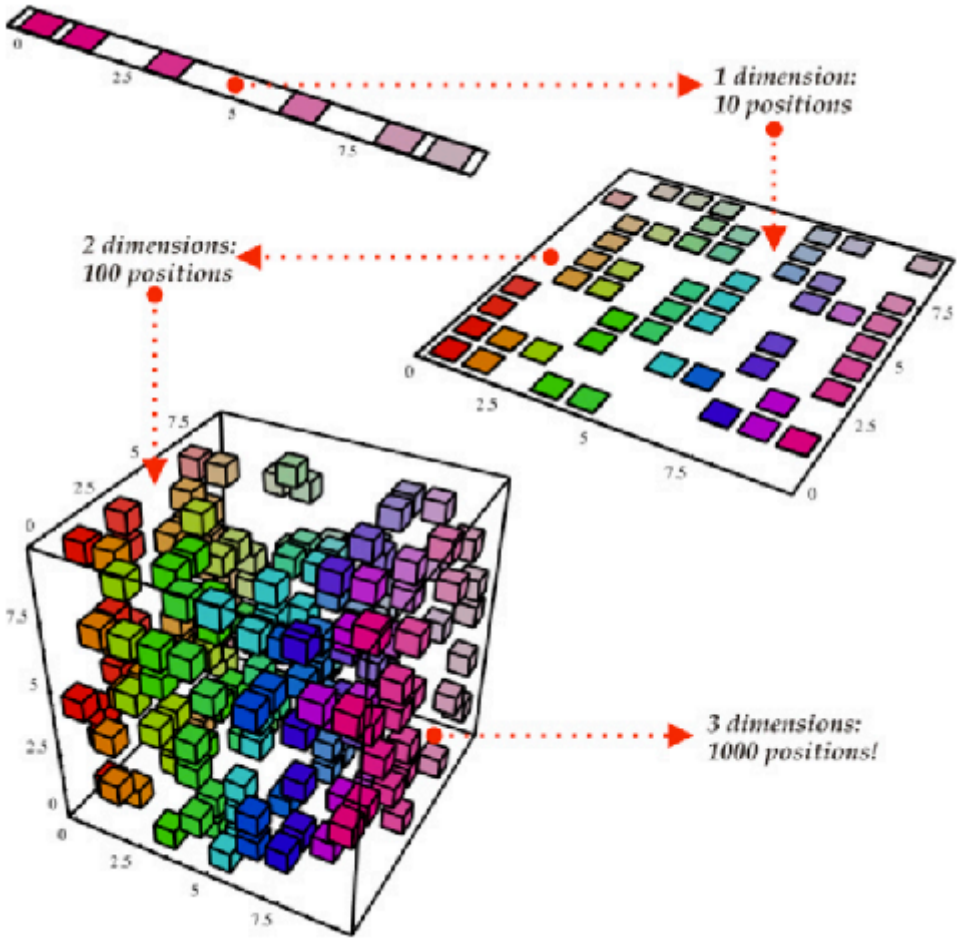
PCA is a method used for **compressing** a lot of data into something that captures the **essence** of the *original data*.

- It reduces the dimension of your data with the **aim of retaining** as *much information as possible*.
- Calculated efficiently with computer programs
- This method combines **highly correlated variables** together to form a smaller number of an artificial set of variables.
- These artificial set of variables are called **'principal components'** that account for **most variance** in the data.

This image below is an example for **visualization**, as how *different dimensions are arranged*. As the **dimensionality increase**, the **complexity in visualization increases**. In the image below, we can see that in

- *1 dimension we have 10 positions which is easy to read and understand*.
- *2 dimensions is having 100 positions, it is still good*.
- *3 dimensions is having 1000 posiitons, it is now a bit difficult to read, as we have to check through 3 corners to understand the data well*.

Note : Though we can go for **N-Dimensions** (N=1,2,3,.....,1000,....,N), but **4-D and above** cannot be drawn on a piece of paper as 1-D, 2-D and 3-D.



PCA IS NOTHING BUT COORDINATE SYSTEM TRANSFORMATION.

The output model uses three axes:
L (Length), W (Width) and H (Height) that perpendicular to each other to represent the 3-D world. So each data point on that object can be written as a function of three variables:

```
Data(i) = f(L(i), W(i), H(i))          [function 1]
```

In the new coordinate system, each data point on that ellipse can be re-written as a function of two variables:

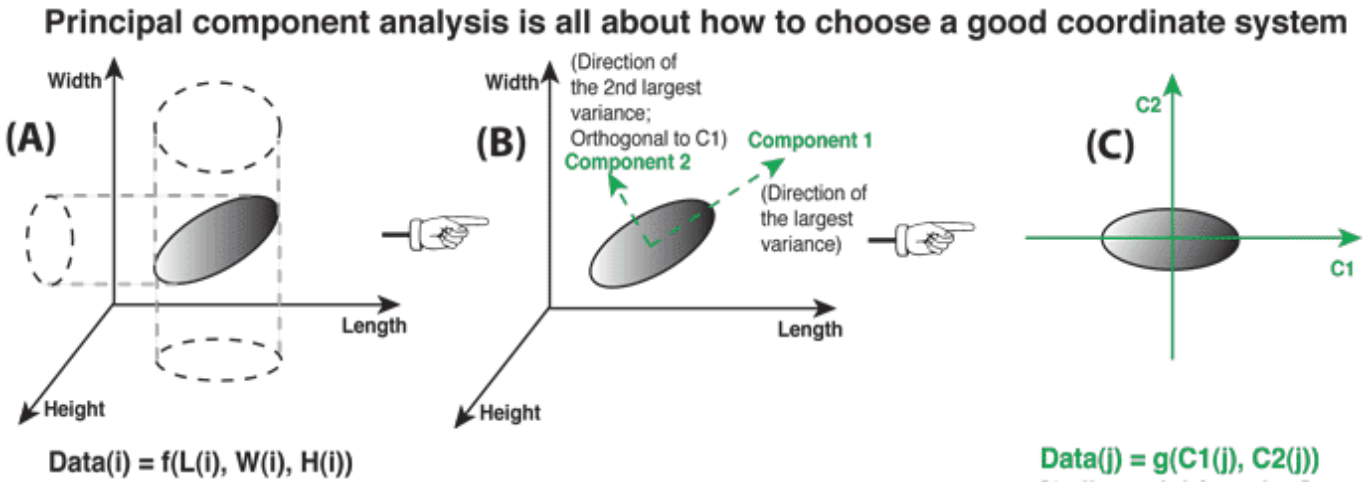
```
Data(j) = g(C1(j), C2(j))              [function 2]
```

- Fewer variables (or lower dimensions of variables) of function 2 compared to function 1.

  ```
  (L, W, H) --> (C1, C2)
  ```

- No information lost.

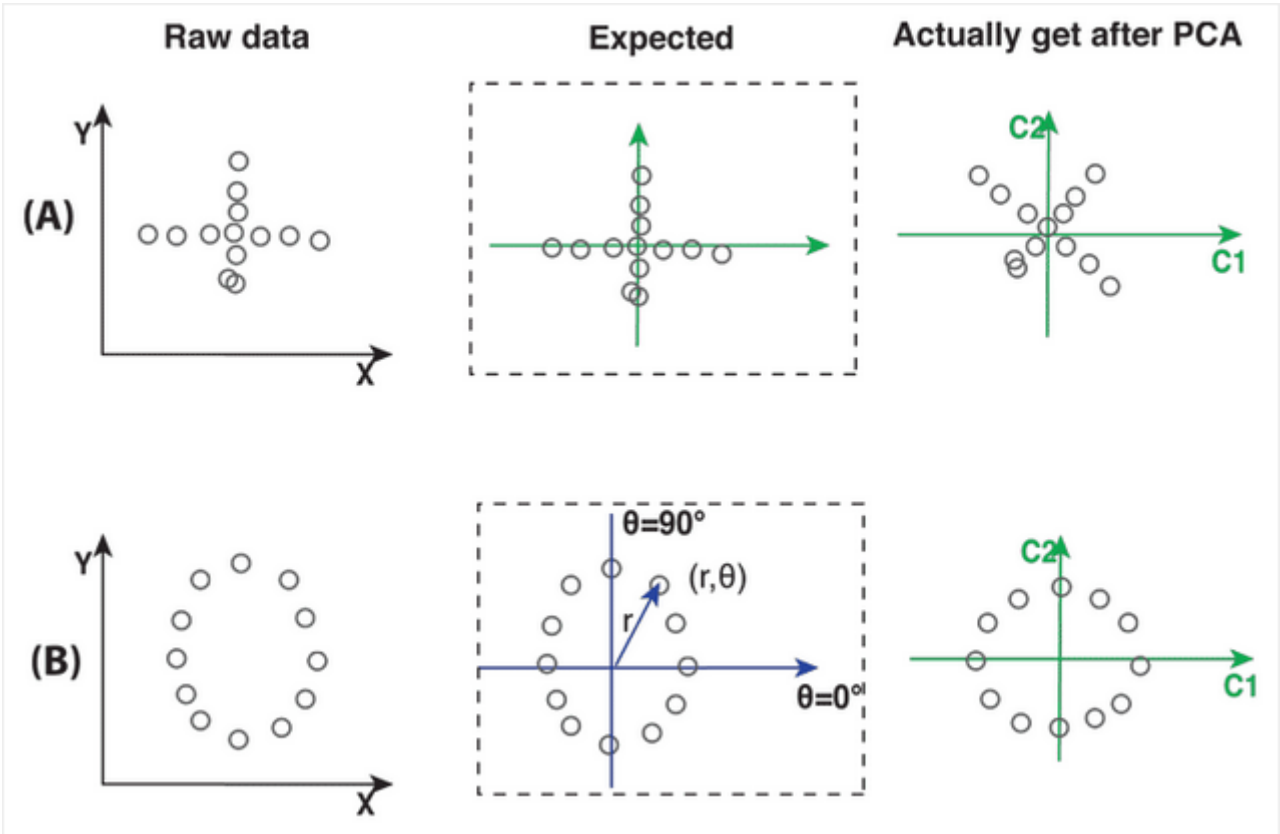
  ```
  function 1 == function 2
  The relative geometric positions of all data points remain unchanged.
  ```

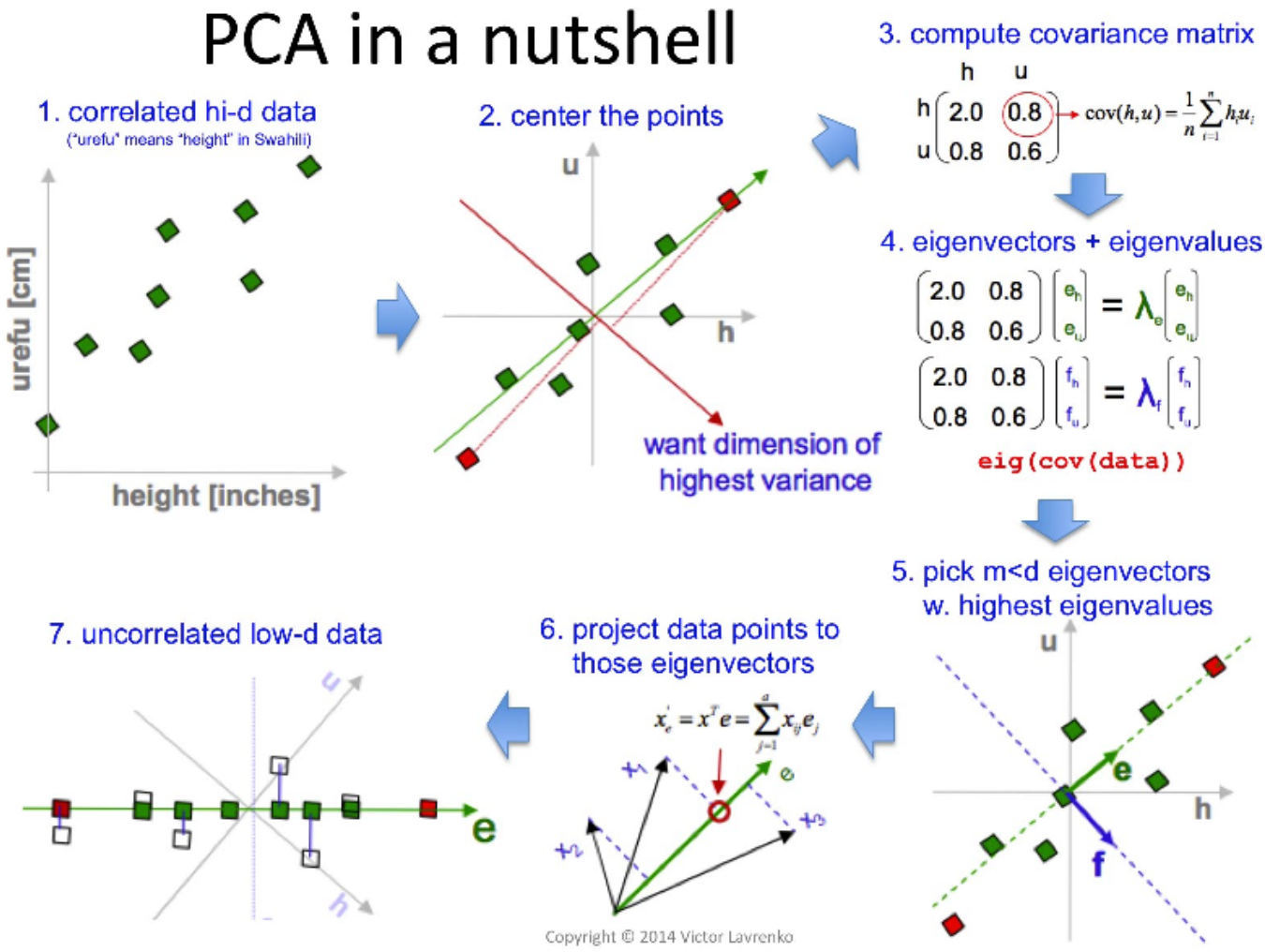


PCA has limitations : example of failures

Any algorithm could **fail** when its **assumption is not satisfied**.

- PCA makes the **"largest variance"** assumptions.
- If the data does not follow a multidimensional normal distribution
- PCA may not give the best principal components.





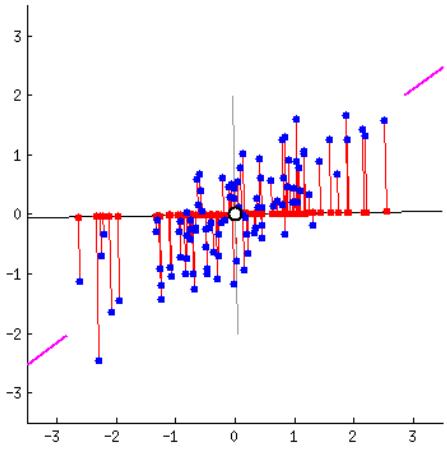
PCA explanation through animation

PCA will find the **"best"** line according to **two different criteria** of what is the "best".

- First, the variation of values along the line should be **maximal**.
  - Pay attention to how the **"spread" (variance)** of the *red dots* changes while the line rotates.
  - **can you see when it *reaches maximum*?**
- Second, if we **reconstruct** the original two characteristics (**position of a blue dot**) from the new one (**position of a red dot**), the **reconstruction error** will be given by the *length of the connecting red line*.
- Observe how the length of these red lines changes while the line rotates.
  - **Can you see when the total length *reaches minimum*?**

If you stare at this animation for some time,

- You will notice that **"the maximum variance"** and **"the minimum error"** are reached at the **same time**, namely when the line points to the magenta ticks I marked on both sides of the data cloud.
  - This line corresponds to the *new data property that will be constructed by PCA*.



Conclusion

Thus PCA is a method that brings together:

1. A measure of how each variable is associated with one another. (Covariance matrix.)
2. The directions in which our data are dispersed. (Eigenvectors.)
3. The relative importance of these different directions. (Eigenvalues.)

PCA combines our predictors and allows us to drop the eigenvectors that are relatively unimportant.

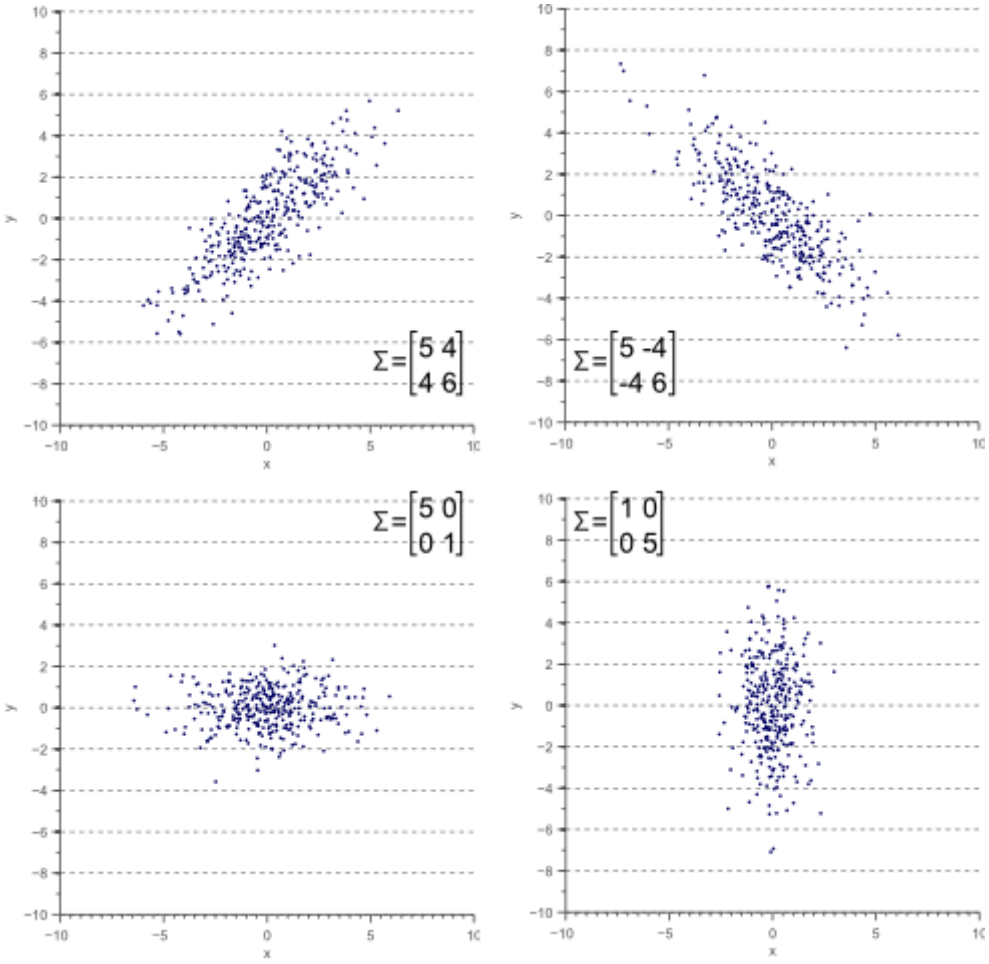
- A square matrix of numbers that describe the **variance of the data, and the covariance among variables** is called covariance matrix.
- It is an **empirical description** of data we observe.
- For a 2 x 2 matrix, a covariance matrix might look like this:

$$\Sigma = \begin{bmatrix} 5 & 4 \\ 4 & 6 \end{bmatrix}$$

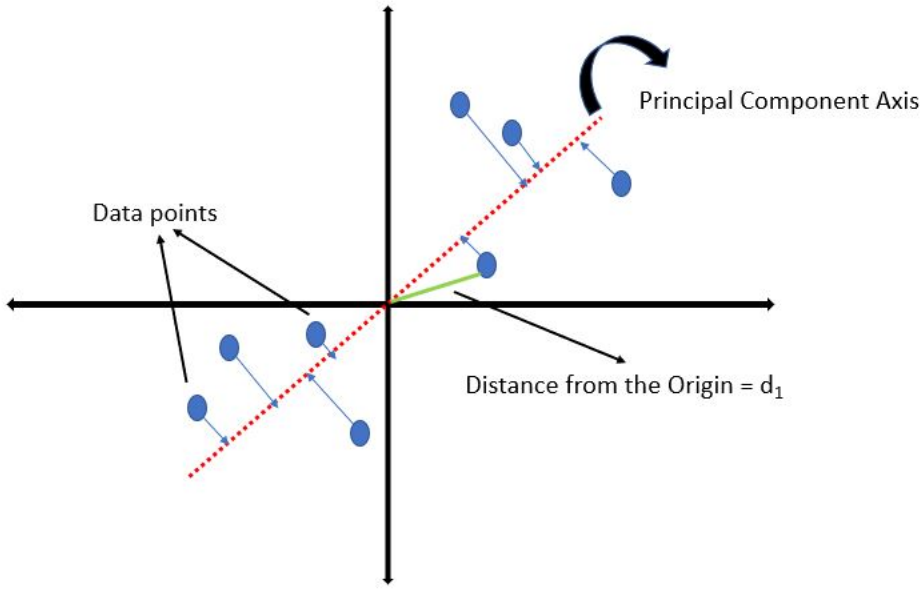
- The numbers on the upper left and lower right represent the **variance of the x and y** variables.
- While the identical numbers on the lower left and upper right represent the **covariance between x and y**.

Graphical Representation:

- If two variables **increase and decrease together** (a line going up and to the right), they have a **positive covariance**.
- If one **decreases while the other increases**, they have a **negative covariance** (a line going down and to the right).



Let us understand the concept of Eigenvectors and Eigenvalues



- Similarly, **distance from origin** is calculated for other data points as well.
- **Sum of Squared distances** = **SS**(distances) **= $d_1^2+d_2^2+d_3^2+d_4^2+d_5^2+d_6^2+d_7^2+d_8^2$**
- The **best pc line** is the one with the **largest sum of squared distance** between the projected points and the origin.

- An **Eigen vector** is a vector whose direction remains unchanged when a linear transformation is applied to it.
- The submission of squared distances from origin of all data points is called **Eigen Value**.
- The eigenvector with the highest eigenvalue is therefore the **principal component**.

- Clean & Scaler Transform Data Before you start

1. Checking and dropping constant columns

```
drop_cols=[]
for cols in data.columns:
    if data[cols].std()==0:
        drop_cols.append(cols)
print("Number of constant columns to be dropped: ", len(drop_cols))
print(drop_cols)
data.drop(drop_cols,axis=1, inplace = True)
data
```

2. Scaler Transformation

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(DATASET)
PCA = pd.DataFrame(scaler.transform(DATASET), index=DATASET.index, columns=data.columns)
PCA = pd.DataFrame(PCA)
PCA.head()
```

3. Preparing X and y using pandas.

```
X = value_sc.loc[:,value_sc.columns != TV]
y = value_sc.loc[:,value_sc.columns == TV]
```

4. Use any Regressers

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor
randomforestreg = RandomForestRegressor()

randomforestreg.fit(X_train, y_train)

y_pred_train_rfr = randomforestreg.predict(X_train)
y_pred_test_rfr = randomforestreg.predict(X_test)
```

5. Model Selection

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=1)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

6. Use the model for predictions

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

7. Use RMSE for model evaluation

```
from sklearn import metrics
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, predictions))
print('RMSE for test set is {}'.format(RMSE_test))
```

DIMENSIONALITY REDUCTION

- One of the major problems with this dataset is that it has too many predictors . To go through each of these predictors and see which ones are significant for the model is going to be a tedious task. Instead, we can use one of the all-time favourite dimensionality reduction technique - Principle Component Analysis.
- Before we can use PCA, we need to **STANDARDISE** the data (Standardisation and Normalization are used inter-dependently. Standardisation is moulding the data to between -1 and +1 data points. Normalisation is normalising the data so that the data points lie along the mean.)

8. Now that the data is scaled, we shall use PCA

```
from sklearn.decomposition import PCA
pca = PCA(0.95).fit(X)
```

9. Variance Graph

```
var=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=3)*100)
plt.ylabel('% Variance Explained')
plt.xlabel('Number of Features')
plt.title('PCA Analysis')
plt.ylim(30,100.5)
plt.style.context('seaborn-whitegrid')
plt.plot(var)
```

```
print('%d components explain 95%% of the variation in data' % pca.n_components_)
```

10. We can see that the first xxx Principal Components attribute for about 95% variation in the data. We shall use these xxx for our prediction

```
pca = PCA(n_components=____, random_state = 0)
pca.fit(X)
X_number__ = pca.transform(X)
print(X_number__.shape)
```

```
Split the random variable in train and test data
```

In [ ]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_250, y, test_size=0.25, random_sta
print(X_train.shape)
print(X_test.shape)
```

In [ ]:

```
pca = PCA(n_components=100, random_state = 0)
pca.fit(X)
X_100 = pca.transform(X)
print(X_number__.shape)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_100, y, test_size=0.25, random_sta

import time
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()

rf.fit(X_train, y_train)


predictions_pca = rf.predict(X_test)
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, predictions_pca))
print('RMSE for test set is {}'.format(RMSE_test))

predictions_pca_exp = np.expm1(predictions_pca)
```

In [ ]:

```
predictions_pca = rf.predict(X_test)
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, predictions_pca_exp))
print('RMSE for test set is {}'.format(RMSE_test))
```

In [ ]:

```
X_number__
```

11. Using 'exponents' to get back to the original value of target variable.

In [ ]:

```
predictions_pca_exp = np.expm1(predictions_pca)
```

In [ ]:

```
print(len(predictions_pca_exp))
```

In [ ]:

```
print(predictions_pca_exp[:20])
```

12. Mean Absolute Error / Mean Squared Error / Root mean squared error

In [ ]:

```
from sklearn import metrics
```

In [ ]:

```
MAE_test = metrics.mean_absolute_error(y_test, predictions_pca_exp)
print('MAE for test data set is {}'.format(MAE_test))
MSE_test = metrics.mean_squared_error(y_test, predictions_pca_exp)
print('MSE for test set is {}'.format(MSE_test))
RMSE_test = np.sqrt(metrics.mean_squared_error(y_test, predictions_pca))
print('RMSE for test set is {}'.format(RMSE_test))
```

In [ ]:

```

```

In [ ]:

```

```

```

```