Python and Data Analytics in Oil and Gas

Jaiyesh Chahar | jaiyesh0002@gmail.com

Linkedin: https://www.linkedin.com/in/jaiyesh-chahar-9b3642107/ (https://www.linkedin.com/in/jaiyesh-chahar-9b3642107/)

Github: https://github.com/jaiyesh (https://github.com/jaiyesh)

Petroleum From Scratch Youtube Channel: https://www.youtube.com/channel/UC_IT10npISN5V32HDLAklsw (https://www.youtube.com/channel/UC_IT10npISN5V32HDLAklsw)

Welcome Everyone!

List of Contents in this Notebook:

1. Core Python with oil and gas examples
2. Numpy, Pandas, Matplotlib
3. Projects: Vogel's IPR, Pressure Profile in reservoir by varying Parameters
4. Exploratory Data Analysis of Volve field Production Data

Why a Petroleum Engineer Should to learn Python?

1. Build your own simulation tool: Reservoir Simulation, EOR Simulation, Drilling, Production etc.

1. Industry 4.0: Data Volume in Oil and Gas Industry has been increasing exponentially because of advancement of technology, So why waste this data, if we can use it for building powerful Machine Learning, Deep Learning Algorithms for solving problems. For that python is best with its superpowers (Libraries)

Python from Scratch

- Python is a programing language not a software (like petrel)
- IDE (Integrated Development Environment) is an application to run python.
- Examples: Pycharm, Jupyter Notebook, Google Collab(online and Free)

In [1]:

```
# This is a comment, for explaining python code, preventing certain line of codes from executing
# Starting with print function
print(' Hello Buddies')
print(' How everyone doing')
```

```
 Hello Buddies
 How everyone doing
```

In [2]:

```
# Breaking the print statement : use \n
print('Hello \nUPES')
```

```
Hello 
UPES
```

Single values Data Types

1. Integer eg. - 7
2. Floats eg. - 7.0
3. Booleans: EIther True or False
4. Strings: set of characters that can also contain spaces,text and numbers, intialises by using "__" eg. - "6"

In [3]:

```
#getting data type using type() function
print(type(4))
print(type(4.0))
print(type("4"))
print(type("Petroleum Engineering"))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
```

Variables: Containers for storing values

- A variable is created the moment you first assign a value to it using = sign

In [4]:

```
a = 5
b = 'Petroleum'
```

In [5]:

```
a
```

Out[5]:

```
5
```

In [6]:

```
b
```

Out[6]:

```
'Petroleum'
```

In [7]:

```
b = "UPES" #Reintialize
b
```

Out[7]:

```
'UPES'
```

```
In [8]:  # casting Type
          x = '566'
          type(x)

Out[8]:   str
```

```
In [9]:   y = float(x)
          y

Out[9]:   566.0
```

```
In [10]:  type(y)

Out[10]:  float
```

Mathematical Operations

```
In [11]:  x = 45
          y = 21
          print(x+y)
          print(x-y)
          print(x*y)
          print(x/y)

          66
          24
          945
          2.142857142857143
```

```
In [12]:  ## Exponent: using **
          print(y**2)

          441
```

```
In [13]:  ## Quotient given by //
          print(y//5)

          4
```

```
In [14]:  ## Remainder using %
          print(y%5)

          1
```

```
In [15]:  print(x**2+y**2)

          2466
```

Strings

```
In [16]:  # surrounded by either single quotation marks, or double quotation mark
          s
```

6 : integer

6.0 : Float

"6" : String

```
In [17]: # Operations
         print('Petroleum'+'Engineering')
         print('Spam'*3)

         PetroleumEngineering
         SpamSpamSpam
```

```
In [18]: print(4*3)
         print(4*'3')

         12
         3333
```

```
In [19]: type(4*'3')

Out[19]: str
```

Input: Asking input from user

```
In [20]: porosity = input('Enter the Formation porosity: ')

         Enter the Formation porosity: 0.2
```

```
In [21]: porosity

Out[21]: '0.2'
```

```
In [22]: type('porosity')

Out[22]: str
```

```
In [24]: porosity = float(input('Enter the Formation porosity: '))

         Enter the Formation porosity: 0.5
```

```
In [25]: type(porosity)

Out[25]: float
```

```
In [26]: permeability = float(input(' Enter the formation\'s permeability(md):
         '))

          Enter the formation's permeability(md): 12
```

In [27]:

```
# String Formatting:
print(f'Formation porosity is {porosity} and permeability is {permeability}')  #Using fstring
print('Formation porosity is {} and permeability is {}'.format(porosity,permeability)) #Using .format
print('Formation porosity is', porosity, 'and permeability is', permeability )  # Using comma
```

```
Formation porosity is 0.5 and permeability is 12.0
Formation porosity is 0.5 and permeability is 12.0
Formation porosity is 0.5 and permeability is 12.0
```

Booleans and Comparisons

- Booleans represent one of two values: True or False.

In [28]:

```
print(2!= 3)
```

```
True
```

In [29]:

```
print(2 == 3 )
```

```
False
```

In [30]:

```
print(2 = 3 )
```

```
  File "<ipython-input-30-2981d28b369a>", line 1
    print(2 = 3 )
         ^
SyntaxError: keyword can't be an expression
```

- '=' is used for intializing variable values, '==' is used for comparison

In [31]:

```
print(5>3 or 3>5 )
```

```
True
```

In [32]:

```
print(5>3 and 3>5 )
```

```
False
```

If-else: To run a block of code only if certain condition holds true

Identation

- Indentation refers to the spaces at the beginning of a code line
- Python uses indentation to indicate a block of code.

In [33]:

```
if 5 > 2:
    print("Five is greater than two!")
    print('If is true')
print('This is outside')
```

```
Five is greater than two!
If is true
This is outside
```

In [35]:

```
Reservoir_Pressure = float(input('Enter the reservoir pressure(psi):
'))
Hydrostatic_pressure = float(input('Enter the Hydrostatic pressure of m
ud(psi): '))
Fracture_pressure = float(input('Enter the Fracture pressure of rock(ps
i): '))
```

```
Enter the reservoir pressure(psi): 1500
Enter the Hydrostatic pressure of mud(psi): 2200
Enter the Fracture pressure of rock(psi): 2000
```

In [36]:

```
if Hydrostatic_pressure > Reservoir_Pressure and Hydrostatic_pressure <
Fracture_pressure:
    print('Reservoir Pressure is lesser than Hydrostatic Pressure, so N
o Kick')
    print('Hydrostatic Pressure due to mud is lesser than Fracture Pres
sure of Formation, so no fracture')
    print('Safe Zone')
elif Hydrostatic_pressure > Reservoir_Pressure and Hydrostatic_pressure
> Fracture_pressure:
    print('Hydrostatic Pressure greater than Fracture pressure, may res
ult in Frature')
    print('Alert!!! Risk of formation Fracture')
else:
    print('Alert!!! Risk of kick')
```

```
Hydrostatic Pressure greater than Fracture pressure, may result in Fr
ature
Alert!!! Risk of formation Fracture
```

In [37]:

```
Pr = float(input('Enter the intial Reservoir pressure(psia):  '))
Pb = float(input('Enter the bubble point pressure(psia) of reservoir fl
uid:  '))
```

```
Enter the intial Reservoir pressure(psia):  1500
Enter the bubble point pressure(psia) of reservoir fluid:  2000
```

In [38]:

```
if Pr > Pb:  #Reservoir Pressure greater than Bubble Point Pressure: Un
dersqaturated Oil Reservoir
    print('Your reservoir is Undersaturated Oil Reservoir')
elif Pr==Pb:
    print('Your reservoir is Saturated Oil Reservoir')
else:
    print('Gas has been evolved from your reservoir, so your reservoir
has both phases: Oil and Gas')
```

```
Gas has been evolved from your reservoir, so your reservoir has both
phases: Oil and Gas
```

While Loop : To repeat a block of code again and again; until the condition satisfies

The code in body of while loop is executed repeatedly. This is called **Iterations**

In [39]:

```
pressure = 2000
while pressure < 2500:
    print(f'{pressure} is not abnormal pressure')
    pressure+=100
else:  # Else loop can also be used in while loop, it will run when while loop condition is not getting satisfied
    print(f'{pressure} is abnormal')
```

```
2000 is not abnormal pressure
2100 is not abnormal pressure
2200 is not abnormal pressure
2300 is not abnormal pressure
2400 is not abnormal pressure
2500 is abnormal
```

In [40]:

```
## While loop Can be used to stop iteration after a specific input
```

In [41]:

```
Password = input('Enter the Password: ')
while Password!= 'Petroleum':
    print('Wrong Password Enter Again')
    Password = input('Enter the Password: ')

else:
    print('Access Granted')
```

```
Enter the Password: hello
Wrong Password Enter Again
Enter the Password: sdsj
Wrong Password Enter Again
Enter the Password: Petroleum
Access Granted
```

break statement: for breaking the while loop prematurely

We can break an infinite loop if some condition is satisfied

In [42]:

```
pressure = 2500
while True: #while True is an easy way to make an infinite loop
    print(f'Formation Pressure is {pressure} psi')
    pressure +=100
    if pressure ==5000:
        print('Pressure Reached its Maximum value, so breaking the loop
')
        break
print('Finished')
```

```
Formation Pressure is 2500 psi
Formation Pressure is 2600 psi
Formation Pressure is 2700 psi
Formation Pressure is 2800 psi
Formation Pressure is 2900 psi
Formation Pressure is 3000 psi
Formation Pressure is 3100 psi
Formation Pressure is 3200 psi
Formation Pressure is 3300 psi
Formation Pressure is 3400 psi
Formation Pressure is 3500 psi
Formation Pressure is 3600 psi
Formation Pressure is 3700 psi
Formation Pressure is 3800 psi
Formation Pressure is 3900 psi
Formation Pressure is 4000 psi
Formation Pressure is 4100 psi
Formation Pressure is 4200 psi
Formation Pressure is 4300 psi
Formation Pressure is 4400 psi
Formation Pressure is 4500 psi
Formation Pressure is 4600 psi
Formation Pressure is 4700 psi
Formation Pressure is 4800 psi
Formation Pressure is 4900 psi
Pressure Reached its Maximum value, so breaking the loop
Finished
```

Continue : to jump back to top of the while loop, rather than stopping it.

Stops the current iteration and continue with the next one.

In [43]:

```
pressure = 3500
while pressure < 4500:
    pressure+=100
    if pressure ==4000:
        print('Skipping 4000 psi')
        continue
    print(f'Pressure is {pressure} psi')
```

```
Pressure is 3600 psi
Pressure is 3700 psi
Pressure is 3800 psi
Pressure is 3900 psi
Skipping 4000 psi
Pressure is 4100 psi
Pressure is 4200 psi
Pressure is 4300 psi
Pressure is 4400 psi
Pressure is 4500 psi
```

Multi Value Containers Data Structures and Sequences

Lists

- Used to store multiple items in a single variable
- Square Brackets are used [] for creating lists
- Mutable: Can Change length, elements, elements values

In [44]:

```
porosity = [0.3,0.41,0.51,0.11,0.34,0.67]
```

In [45]:

```
porosity
```

Out[45]:

```
[0.3, 0.41, 0.51, 0.11, 0.34, 0.67]
```

In [46]:

```
type(porosity)
```

Out[46]:

```
list
```

In [47]:

```
#Index = Address of Element in list
porosity[0]
```

Out[47]:

```
0.3
```

In [48]:

```
porosity[-1]
```

Out[48]:

```
0.67
```

In [49]:

```
porosity[0] = 0.5453
```

In [50]:

```
porosity
```

Out[50]:

```
[0.5453, 0.41, 0.51, 0.11, 0.34, 0.67]
```

In [51]:
```
#Len function: Calculate length of list
```

In [52]:
```
len(porosity)
```

Out[52]:
```
6
```

In [53]:
```
# Empty list are widely used for populating it later with certain calculations
specificgravity_of_crudes = [0.8,0.7,0.85,0.76,0.91,0.64,0.65]
denWater = 62.4
```

In [54]:
```
CrudeDensity = []
i = 0
while i < len(specificgravity_of_crudes):
    denO = denWater*specificgravity_of_crudes[i]
    CrudeDensity.append(denO)  #append: adding an item to the end of an existing list
    i+=1
CrudeDensity
```

Out[54]:
```
[49.92, 43.68, 53.04, 47.424, 56.784, 39.936, 40.56]
```

In [55]:
```
#insert: Like append but we can insert a new item at any position in list.
a = [1,2,3,4,5,6,7]
a.insert(4,'PETROLEUM')
a
```

Out[55]:
```
[1, 2, 3, 4, 'PETROLEUM', 5, 6, 7]
```

In [56]:
```
#Slicing. Just like the name says.
#It helps cut a slice (sub-part) off a List.
superset = [1,2,3,4,5,6,7,8,9]


#Slicing syntax - listname[start:stop:step] start is included. stop is not
subset1 = superset[0:5:2]

print(subset1)

subset2 = superset[:]

print(subset2) #Skipping a part also works for first and end indices.

subset3 = superset[:-1] #starting to ending -1
print(subset3)

subset4 = superset[:] #everything
print(subset4)
```

```
[1, 3, 5]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [57]: #Reversing the list
         reverse_set = superset[-1::-1]

         #Start at -1 th index, end at 0th, take negative steps.

         print(reverse_set)

         [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
In [58]: # List Operations:
         #addition(concating lists): Demerit of list, Numpy Array will handle th
         is issue
         a =[1,2,3]
         b =[4,5,6]
         c = a+b
         c

Out[58]: [1, 2, 3, 4, 5, 6]
```

```
In [59]: #Multiply
         a*3

Out[59]: [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
In [60]: a*b

         ---------------------------------------------------------------------
         ------
         TypeError                                 Traceback (most recent call
         last)
         <ipython-input-60-8ce765dcfa30> in <module>
         ----> 1 a*b

         TypeError: can't multiply sequence by non-int of type 'list'
```

Tuples

- A tuple is a collection which is ordered and unchangeable
- parenthesis are used ()
- Immutable: Cannot be change: can be used for storing important data that cannot be changed by anyone

```
In [61]: oilprd = (5000,1000,4322,5633,7892)
```

```
In [62]: #Ordered
         oilprd[1]

Out[62]: 1000
```

In [63]:

```
#Immutable
oilprd[3] = 2332
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-63-8ca89081a59e> in <module>
      1 #Immutable
----> 2 oilprd[3] = 2332

TypeError: 'tuple' object does not support item assignment
```

Dictionaries

Helps store data with labels

Syntax => { key : value }

Can be directly converted to DataFrames (tables)

In [64]:

```
rock_properties = {'poro' : 0.25, 'perm' : 150 , 'lithology' : 'Limestone'}

print(rock_properties)
```

```
{'poro': 0.25, 'perm': 150, 'lithology': 'Limestone'}
```

In [65]:

```
#Access
rock_properties['poro']
```

Out[65]:

```
0.25
```

In [66]:

```
#Change
rock_properties['lithology'] = 'Shale'
```

In [67]:

```
rock_properties
```

Out[67]:

```
{'poro': 0.25, 'perm': 150, 'lithology': 'Shale'}
```

In [68]:

```
#Saturation
rock_properties['WaterSat'] = 0.25
```

In [69]:

```
rock_properties
```

Out[69]:

```
{'poro': 0.25, 'perm': 150, 'lithology': 'Shale', 'WaterSat': 0.25}
```

```
In [70]: import pandas as pd
         pd.DataFrame({'poro' : [0.10,0.20,0.30,0.50], 'perm' : [50,100,150,20
         0],'Lithology':['S','S','C','S']})
```

Out[70]:

	poro	perm	Lithology
0	0.1	50	S
1	0.2	100	S
2	0.3	150	C
3	0.5	200	S

Sets

Curly braces are used just like dictionaries

Unordered: Can't be indexed

Can't contain duplicate values

Faster Than List

```
In [71]: a = {1,2,3,4,5,6,7,1,2,2}
```

```
In [72]: a
```

```
Out[72]: {1, 2, 3, 4, 5, 6, 7}
```

Summary of Data Structures

1. Dictionary - Key:Value, mutable
2. Lists - Mutable, Empty lists are used heavily to populate it later during the program
3. Set - Uniqueness of Elements
4. Tuples - Data cannot be changed

for loops

The tool with which we can utilize the power of computers(iterations)

We can perform repetition of a command a 1000 times in 1 second.

Iterations are always performed on Iterables (ordered-collections).

Examples of iterables - lists, strings etc

In [73]:

```
Specific_gravity = [0.2,0.3,0.4,0.87,0.9,1,0.2]
for i in Specific_gravity:
    api = (141.5/i) - 131.5
    print('API gravity corresponding to Specific Gravity', i, 'is', ap
i)
```

```
API gravity corresponding to Specific Gravity 0.2 is 576.0
API gravity corresponding to Specific Gravity 0.3 is 340.166666666666
7
API gravity corresponding to Specific Gravity 0.4 is 222.25
API gravity corresponding to Specific Gravity 0.87 is 31.143678160919
535
API gravity corresponding to Specific Gravity 0.9 is 25.7222222222222
3
API gravity corresponding to Specific Gravity 1 is 10.0
API gravity corresponding to Specific Gravity 0.2 is 576.0
```

Functions

Functions are very important.

1. Make the code clean.

2. Helps Prevent repetitive commands.

- Instead of writing code again and again we can create a function for different values, we can write a function and call that whenever we want to do the calculations

In [74]:

```
def Function():
    print('Use of Function')
```

In [75]:

```
#Calling Function
Function()
```

```
Use of Function
```

In [76]:

```
#Returning from a function
def add(x,y):
    return x+y
```

In [77]:

```
#Storing return in a variable
c = add(3,4)
```

In [78]:

```
c
```

Out[78]:

```
7
```

In [79]:

```
#Once we return from a function, it stops being executed, any code writ
en after the return will never be executed
def f(x,y,z):
    return x/y +z
    print('Hello')
```

In [80]:

```
f(3,4,5)
```

Out[80]:

```
5.75
```

In [81]:

```
def api(x):
    '''api function will take input as Specific Gravity and return dens
ity in API'''
    api = 141.5/x - 131.5
    print('The api gravity is',round(api))
```

In [82]:

```
api(0.9)
```

```
The api gravity is 26
```

In [83]:

```
def pythhyp(base,height):
    hyp = ((base**2)+(height**2))**0.5
    print('The value of hypotenuse by pythagoras theorem is: ', hyp)
    return hyp
```

In [84]:

```
pythhyp(3,4)
```

```
The value of hypotenuse by pythagoras theorem is:  5.0
```

Out[84]:

```
5.0
```

In [85]:

```
c = pythhyp(3,4)
```

```
The value of hypotenuse by pythagoras theorem is:  5.0
```

Lambda Function

Single line function

In [86]:

```
api_lambda = lambda x : 141.5/x - 131.5
```

In [87]:

```
api_lambda(0.9)
```

Out[87]:

```
25.72222222222223
```

Day 2

Numpy

1. Stands for Numerical Python,numerical package for python
2. Numpy is also incredibly fast, as it has bindings to C libraries. So, NumPy operations help in computational efficiency.
3. Entire mathematical package of Matlab is numpy
4. Data Manipulations of arrays and matix, Can be used for Multidimensional Data Preparations
5. Array doesn't exist in python core, there list is present

In [2]:

```
# importing numpy
import numpy as np
```

numpy arrays: numpy object used for storing data:

Faster than lists(upto 50 times) and takes lesser space

In [3]:

```
#Creating Numpy array from a python list
perm=[10,15,18,65,25]
permarr = np.array(perm)
permarr
```

Out[3]:

```
array([10, 15, 18, 65, 25])
```

In [4]:

```
type(permarr)
```

Out[4]:

```
numpy.ndarray
```

In [5]:

```
# Creating Numpy arrasy from tuples
import numpy as np
t = (1, 2, 3, 4, 5)

arr = np.array(t)

arr
```

Out[5]:

```
array([1, 2, 3, 4, 5])
```

DImensions in array

0-D array: Array having one element

In [6]:

```
ar0 = np.array(34)
```

In [7]:

```
ar0
```

Out[7]:

```
array(34)
```

```
In [8]:  #Checking Dimension of array using ndim attribute

         ar0.ndim

Out[8]:  0
```

1-D array : Array having 0-D arrays as its elements

- Most common and basic arrays
- Similar to basic lists

```
In [9]:  porosity = [0.4,0.5,0.64,0.73,0.544,0.65]

In [10]: arpor = np.array(porosity)

In [11]: arpor

Out[11]: array([0.4  , 0.5  , 0.64 , 0.73 , 0.544, 0.65 ])

In [12]: arpor.ndim

Out[12]: 1
```

2-D array: Array having 1-D arrays as its elements.

- Represents matrix or 2md order tensor

```
In [13]: porosity2d = np.array([[0.5,0.34,0.56,0.98,0.12],[0.34,0.65,0.12,0.87,
         0.23]])

In [14]: porosity2d

Out[14]: array([[0.5 , 0.34, 0.56, 0.98, 0.12],
                [0.34, 0.65, 0.12, 0.87, 0.23]])

In [15]: porosity2d.ndim

Out[15]: 2
```

3- D array : 2-D arrays behave as element

- Represents 3rd order tensor

```
In [16]: perm = np.array([[[1,2,3,5],[4,5,6,7]],[[1,2,3,2],[3,4,5,5]],[[43,45,6
         7,56],[32,54,65,64]]])

In [17]: perm.ndim

Out[17]: 3
```

```
In [18]: perm
```

```
Out[18]: array([[[ 1,  2,  3,  5],
                 [ 4,  5,  6,  7]],

                [[ 1,  2,  3,  2],
                 [ 3,  4,  5,  5]],

                [[43, 45, 67, 56],
                 [32, 54, 65, 64]]])
```

Shape: Checking shape

```
In [19]: porosity2d
```

```
Out[19]: array([[0.5 , 0.34, 0.56, 0.98, 0.12],
                [0.34, 0.65, 0.12, 0.87, 0.23]])
```

```
In [20]: porosity2d.shape
```

```
Out[20]: (2, 5)
```

```
In [21]: perm
```

```
Out[21]: array([[[ 1,  2,  3,  5],
                 [ 4,  5,  6,  7]],

                [[ 1,  2,  3,  2],
                 [ 3,  4,  5,  5]],

                [[43, 45, 67, 56],
                 [32, 54, 65, 64]]])
```

```
In [22]: perm.shape
```

```
Out[22]: (3, 2, 4)
```

Using inbuilt methods for generating arrays

Arange: Return an array with evenly spaced elements in the given interval.

```
- np.arange(start,stop,step)
```

```
In [23]: #Make an array of pressures ranging from 0 to 5000 psi with a step size
         of 500 psi
         pressures = np.arange(0,5500,500)
```

```
In [24]: pressures
```

```
Out[24]: array([   0,  500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 50
         00])
```

```
In [25]: pressures.ndim
```

```
Out[25]: 1
```

Linspace: Creates lineraly spaced array

- Evenly spaced numbers over a specified interval
- Creating n datapoints between two points

```
In [26]: # Create saturation array from 0 to 1 having 100 points
         saturations = np.linspace(0,1,100)
         saturations
```

```
Out[26]: array([0.        , 0.01010101, 0.02020202, 0.03030303, 0.04040404,
                0.05050505, 0.06060606, 0.07070707, 0.08080808, 0.09090909,
                0.1010101 , 0.11111111, 0.12121212, 0.13131313, 0.14141414,
                0.15151515, 0.16161616, 0.17171717, 0.18181818, 0.19191919,
                0.2020202 , 0.21212121, 0.22222222, 0.23232323, 0.24242424,
                0.25252525, 0.26262626, 0.27272727, 0.28282828, 0.29292929,
                0.3030303 , 0.31313131, 0.32323232, 0.33333333, 0.34343434,
                0.35353535, 0.36363636, 0.37373737, 0.38383838, 0.39393939,
                0.4040404 , 0.41414141, 0.42424242, 0.43434343, 0.44444444,
                0.45454545, 0.46464646, 0.47474747, 0.48484848, 0.49494949,
                0.50505051, 0.51515152, 0.52525253, 0.53535354, 0.54545455,
                0.55555556, 0.56565657, 0.57575758, 0.58585859, 0.5959596 ,
                0.60606061, 0.61616162, 0.62626263, 0.63636364, 0.64646465,
                0.65656566, 0.66666667, 0.67676768, 0.68686869, 0.6969697 ,
                0.70707071, 0.71717172, 0.72727273, 0.73737374, 0.74747475,
                0.75757576, 0.76767677, 0.77777778, 0.78787879, 0.7979798 ,
                0.80808081, 0.81818182, 0.82828283, 0.83838384, 0.84848485,
                0.85858586, 0.86868687, 0.87878788, 0.88888889, 0.8989899 ,
                0.90909091, 0.91919192, 0.92929293, 0.93939394, 0.94949495,
                0.95959596, 0.96969697, 0.97979798, 0.98989899, 1.        ])
```

```
In [27]: saturations.ndim
```

```
Out[27]: 1
```

Indexing Arrays

Accesing elements

```
In [28]: perm = np.array([[[1,2,3],[2,3,4]],[[34,45,56],[34,78,23]]])
```

In [29]:
```
perm
```

Out[29]:
```
array([[[ 1,  2,  3],
        [ 2,  3,  4]],

       [[34, 45, 56],
        [34, 78, 23]]])
```

In [30]:
```
perm[0]
```

Out[30]:
```
array([[1, 2, 3],
       [2, 3, 4]])
```

In [31]:
```
perm[0][1]
```

Out[31]:
```
array([2, 3, 4])
```

In [32]:
```
perm[0][1][1]
```

Out[32]:
```
3
```

Slicing Arrays

In [33]:
```
perm = np.array([[13,23,32,43,54],[43,55,60,7,8],[12,34,45,77,87],[2,5
5,39,82,49]])
```

In [34]:
```
perm
```

Out[34]:
```
array([[13, 23, 32, 43, 54],
       [43, 55, 60,  7,  8],
       [12, 34, 45, 77, 87],
       [ 2, 55, 39, 82, 49]])
```

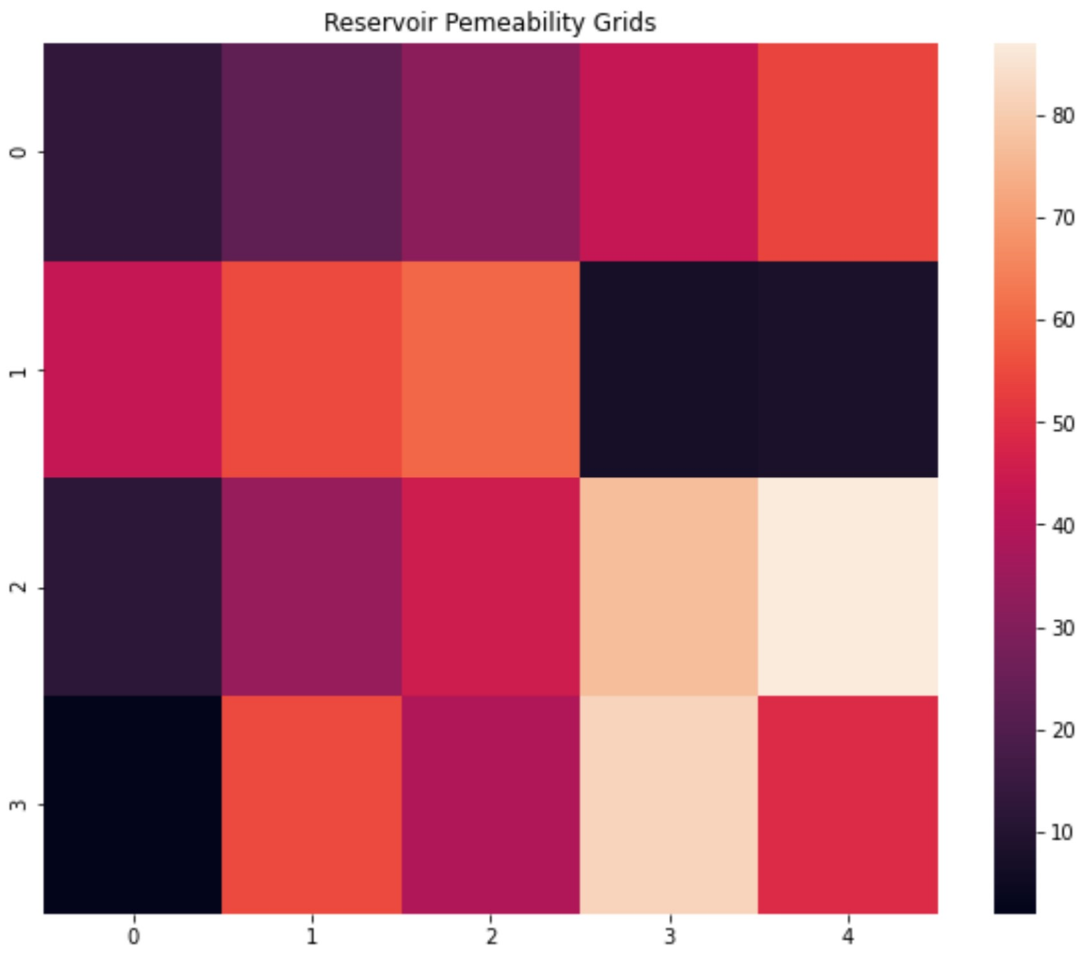
In [35]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

In [36]:

```
plt.figure(figsize = (10,8))
plt.title('Reservoir Pemeability Grids')
sns.heatmap(perm)
```

Out[36]:

```
<AxesSubplot:title={'center':'Reservoir Pemeability Grids'}>
```



In [37]:

```
perm
```

Out[37]:

```
array([[13, 23, 32, 43, 54],
       [43, 55, 60,  7,  8],
       [12, 34, 45, 77, 87],
       [ 2, 55, 39, 82, 49]])
```

In [42]:

```
perm[1:4]
```

Out[42]:

```
array([[43, 55, 60,  7,  8],
       [12, 34, 45, 77, 87],
       [ 2, 55, 39, 82, 49]])
```

In [40]:

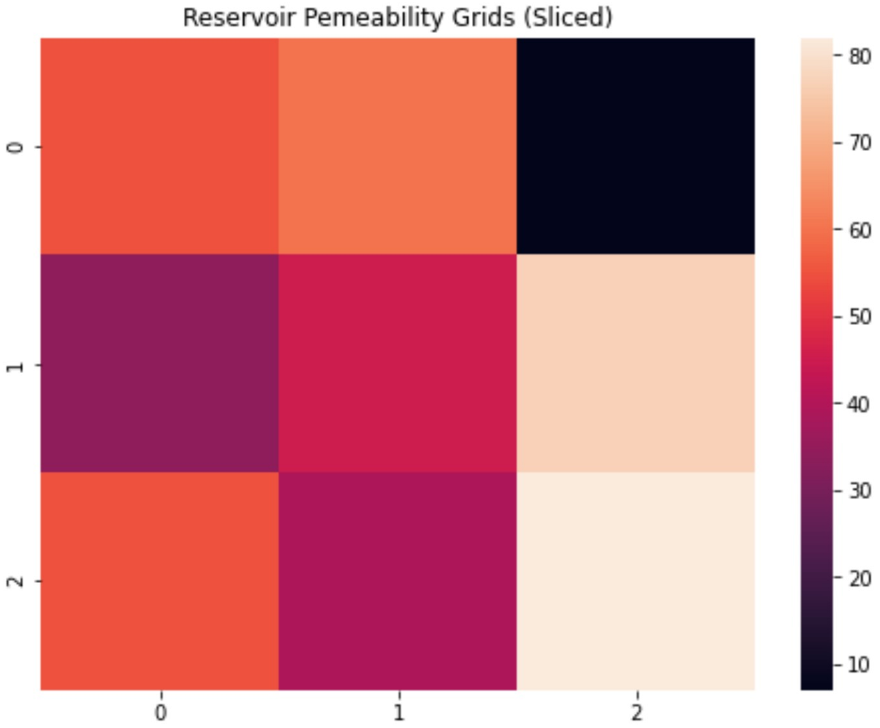
```
perm[1:4,1:4]
```

Out[40]:

```
array([[55, 60,  7],
       [34, 45, 77],
       [55, 39, 82]])
```

```
In [44]: plt.figure(figsize = (8,6))
         plt.title('Reservoir Pemeability Grids (Sliced)')
         sns.heatmap(perm[1:4,1:4])
```

```
Out[44]: <AxesSubplot:title={'center':'Reservoir Pemeability Grids (Sliced)'}>
```



Random Number Generation

Randint : Generating Random Integer

- np.random.randint(min,max,shape)

```
In [45]: np.random.randint(250,350,(3,4,3))
```

```
Out[45]: array([[[335, 302, 285],
                 [290, 335, 251],
                 [297, 279, 299],
                 [348, 303, 307]],

                [[332, 270, 310],
                 [343, 278, 257],
                 [291, 251, 321],
                 [258, 273, 345]],

                [[258, 325, 294],
                 [335, 292, 333],
                 [250, 311, 330],
                 [346, 254, 292]]])
```

rand: Generating Random FLoat between 0 and 1

- provide size

```
In [46]: porosity = np.random.rand(5,4,2)
```

```
In [47]: porosity
```

```
Out[47]: array([[[0.2034112 , 0.01688143],
                 [0.13384843, 0.05466519],
                 [0.55645963, 0.81959005],
                 [0.9601467 , 0.38045259]],

                [[0.53506026, 0.5134192 ],
                 [0.22483723, 0.50576969],
                 [0.61361707, 0.40238323],
                 [0.10464277, 0.2748988 ]],

                [[0.73888073, 0.61586745],
                 [0.24838684, 0.92115323],
                 [0.73716043, 0.06420523],
                 [0.3844945 , 0.83562484]],

                [[0.74130977, 0.32222591],
                 [0.05728948, 0.83661929],
                 [0.48840097, 0.68103858],
                 [0.11205602, 0.22901724]],

                [[0.05598509, 0.41528224],
                 [0.10596222, 0.93729523],
                 [0.93885962, 0.02993201],
                 [0.55515742, 0.46881714]]])
```

Additional*

```
In [48]: import matplotlib.pyplot as plt
         import matplotlib.colorbar
         from matplotlib import cm
```

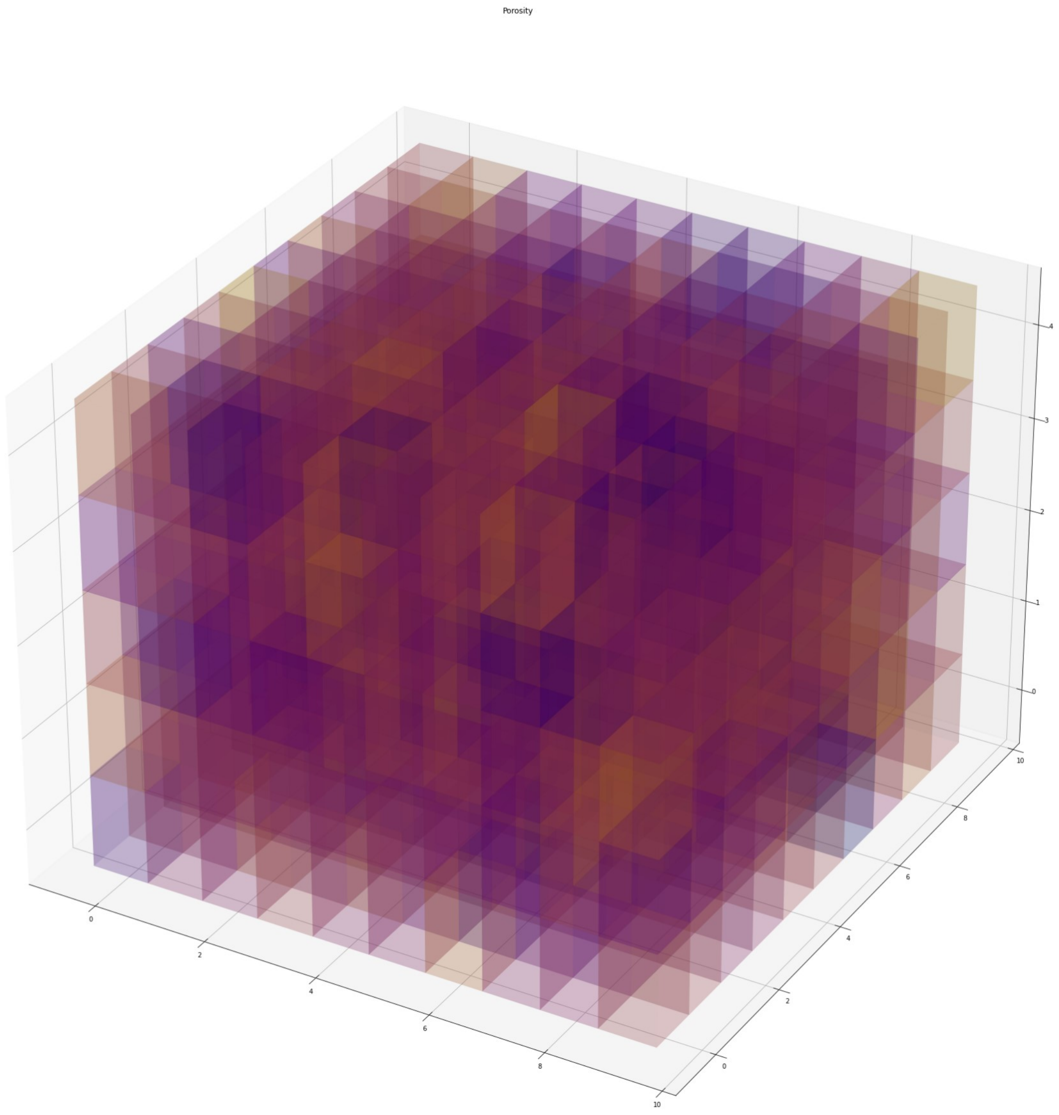
```
In [49]: import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colorbar
from matplotlib import cm

viridis = cm.get_cmap('plasma', 8) #Our color map

def cuboid_data(center, size=(1,1,1)):
    # suppose axis direction: x: to left; y: to inside; z: to upper
    # get the (left, outside, bottom) point
    o = [a - b / 2 for a, b in zip(center, size)]
    # get the length, width, and height
    l, w, h = size
    x =  np.array([[o[0], o[0] + l, o[0] + l, o[0], o[0]],      # x coo
rdinate of points in bottom surface
         [o[0], o[0] + l, o[0] + l, o[0], o[0]],                # x coo
rdinate of points in upper surface
         [o[0], o[0] + l, o[0] + l, o[0], o[0]],                # x coo
rdinate of points in outside surface
         [o[0], o[0] + l, o[0] + l, o[0], o[0]]])               # x coo
rdinate of points in inside surface
    y =  np.array([[o[1], o[1], o[1] + w, o[1] + w, o[1]],      # y coo
rdinate of points in bottom surface
         [o[1], o[1], o[1] + w, o[1] + w, o[1]],                # y coo
rdinate of points in upper surface
         [o[1], o[1], o[1], o[1], o[1]],                        # y coo
rdinate of points in outside surface
         [o[1] + w, o[1] + w, o[1] + w, o[1] + w, o[1] + w]])   # y coo
rdinate of points in inside surface
    z =  np.array([[o[2], o[2], o[2], o[2], o[2]],              # z coo
rdinate of points in bottom surface
         [o[2] + h, o[2] + h, o[2] + h, o[2] + h, o[2] + h],    # z coo
rdinate of points in upper surface
         [o[2], o[2], o[2] + h, o[2] + h, o[2]],                # z coo
rdinate of points in outside surface
         [o[2], o[2], o[2] + h, o[2] + h, o[2]]])               # z coo
rdinate of points in inside surface
    return x, y, z

def plotCubeAt(pos=(0,0,0), c="b", alpha=0.1, ax=None):
    # Plotting N cube elements at position pos
    if ax !=None:
        X, Y, Z = cuboid_data( (pos[0],pos[1],pos[2]) )
        ax.plot_surface(X, Y, Z, color=c, rstride=1, cstride=1, alpha=
0.1)

def plotMatrix(ax, x, y, z, data, cmap=viridis, cax=None, alpha=0.1):
    # plot a Matrix
    norm = matplotlib.colors.Normalize(vmin=data.min(), vmax=data.max
())
    colors = lambda i,j,k : matplotlib.cm.ScalarMappable(norm=norm,cmap
= cmap).to_rgba(data[i,j,k])
    for i, xi in enumerate(x):
            for j, yi in enumerate(y):
                for k, zi, in enumerate(z):
                    plotCubeAt(pos=(xi, yi, zi), c=colors(i,j,k), alpha
=alpha,  ax=ax)
```

```
    if cax !=None:
        cbar = matplotlib.colorbar.ColorbarBase(cax, cmap=cmap,
                                norm=norm,
                                orientation='vertical')
        cbar.set_ticks(np.unique(data))
        # set the colorbar transparent as well
        cbar.solids.set(alpha=alpha)
```

In [50]:
```
x = np.array(range(10))
y = np.array(range(10))
z = np.array(range(5))
por = np.random.normal(loc = 0.5,scale = 0.15,size =(len(x), len(y), le
n(z)) )
print(por.shape)
```

```
(10, 10, 5)
```

In [51]:
```
fig = plt.figure(figsize=(10,5))
ax = fig.add_axes([5,5,5,5], projection='3d')
plt.title('Porosity')
#ax_cb = fig.add_axes([1,1,1,1])
plotMatrix(ax, x, y, z, por, cmap=viridis)
```



Arithmetics Functions:

In [52]:

```
# Addition:
a = [1,2,3,44,5]
b = [4,5,6,7,8]
c= a+b
c
```

Out[52]:

```
[1, 2, 3, 44, 5, 4, 5, 6, 7, 8]
```

In [53]:

```
ara = np.array(a)
arb = np.array(b)
arc = ara +arb
arc
```

Out[53]:

```
array([ 5,  7,  9, 51, 13])
```

In [54]:

```
a*b
```

```
---------------------------------------------------------------------
------
TypeError                                 Traceback (most recent call
last)
<ipython-input-54-8ce765dcfa30> in <module>
----> 1 a*b

TypeError: can't multiply sequence by non-int of type 'list'
```

In [55]:

```
ara*arb
```

Out[55]:

```
array([  4,  10,  18, 308,  40])
```

In [56]:

```
4551/0
```

```
---------------------------------------------------------------------
------
ZeroDivisionError                         Traceback (most recent call
last)
<ipython-input-56-c82d36dbfcfb> in <module>
----> 1 4551/0

ZeroDivisionError: division by zero
```

In [57]:

```
np.array(42523454)/0
```

```
C:\Users\acer\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: Ru
ntimeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

Out[57]:

```
inf
```

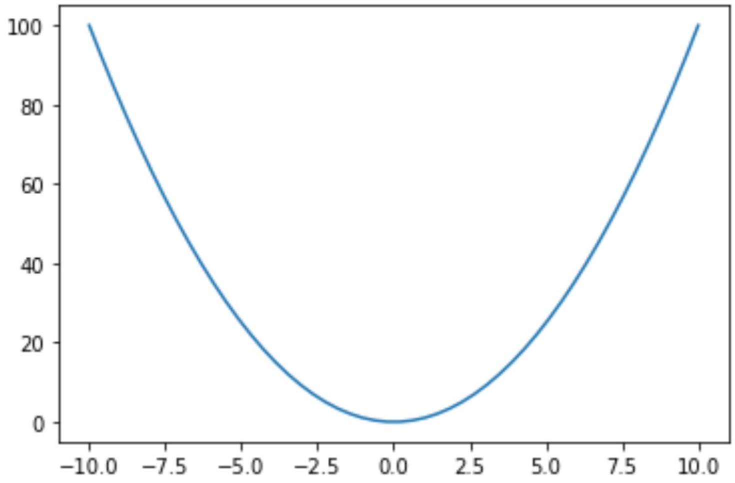
Line Plot using Matplotlib

In [58]:

```
#Step 1: Import the library(s).
import matplotlib.pyplot as plt

#Step 2: create numpy arrays for x and y.
x = np.linspace(-10,10)
y = x**2

#Step 3: Plot now.
plt.plot(x,y)
```

Out[58]:

```
[<matplotlib.lines.Line2D at 0x2171d3b6e08>]
```

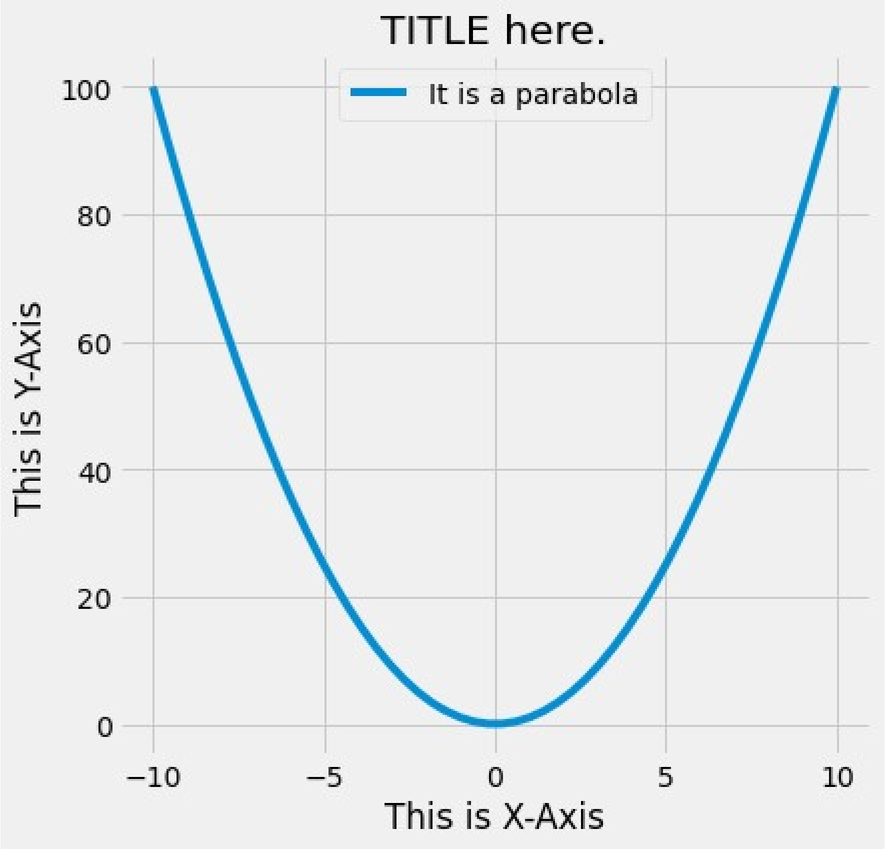


In [63]:

```
#Customization
plt.style.use('fivethirtyeight')
plt.figure(figsize=(6,6))  #6X6 canvas.
# plt.style.use('default')

# 1. generate the plot. Add a label.
plt.plot(x,y,label='It is a parabola')

#2. Set x axis label
plt.xlabel('This is X-Axis')

#3. Set y axis label.
plt.ylabel('This is Y-Axis')

#4. Set the title.
plt.title('TITLE here.')

#5. set the grid.
plt.grid(True)

#6. display the label in a legend.
plt.legend(loc='best')
```

Out[63]:

```
<matplotlib.legend.Legend at 0x2171e56e288>
```



In [60]:

```
plt.style.available
```

Out[60]:

```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

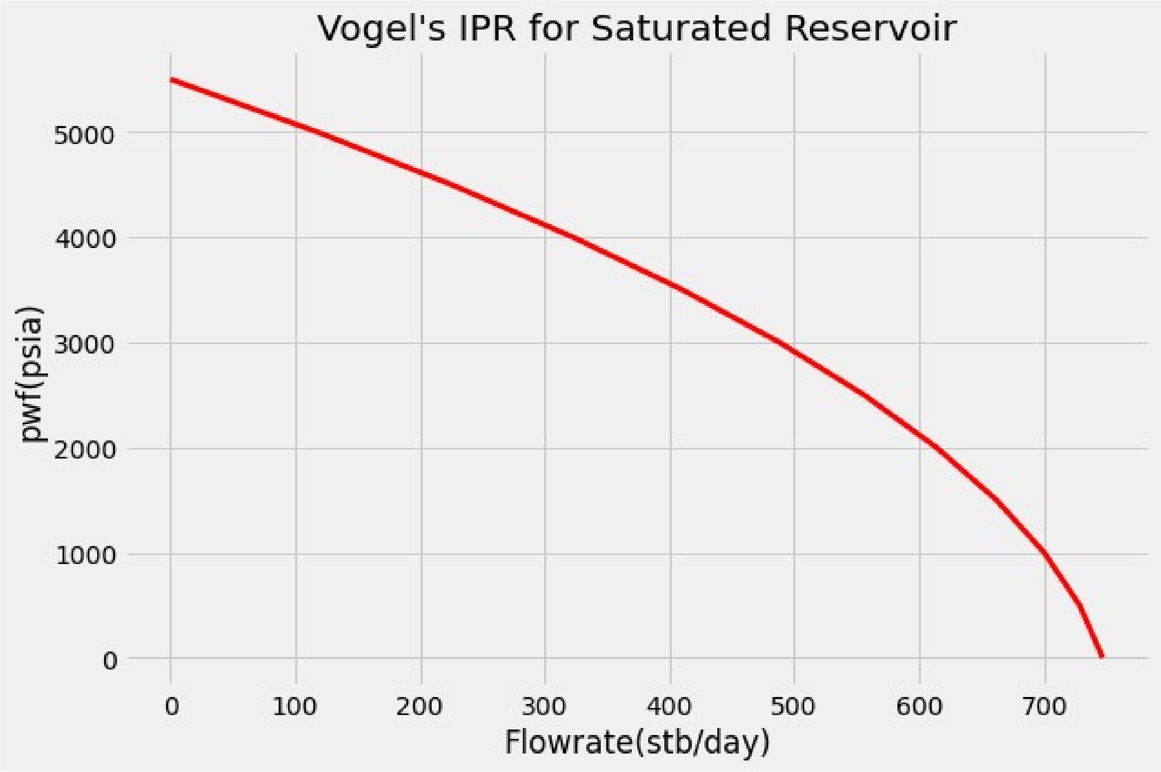
Mini Project: 1. Vogel IPR

In [64]:

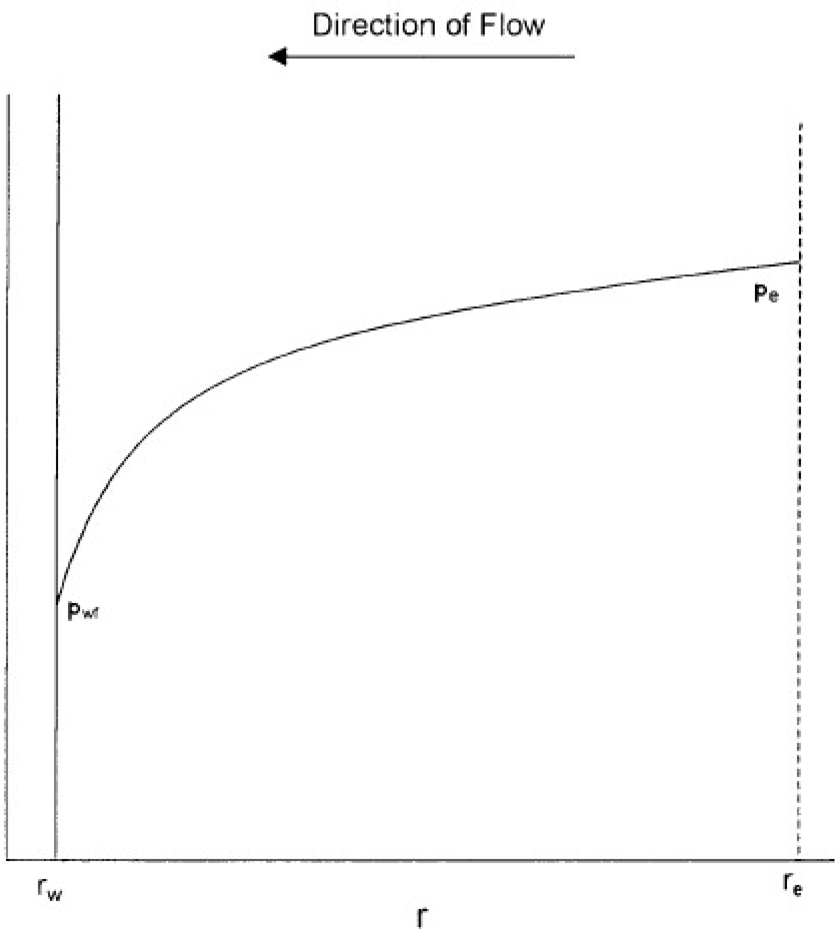
```python
def vogelipr():
    porosity = float(input("Enter Porosity: "))
    K = float(input("Enter Perm.(md): "))
    h = float(input("Enter pay zone thicknes(Feet): "))
    P = float(input("Enter  Reservoir Pressure(psi): "))
    Pb = float(input("Enter Bubble Point Pressure(psi): "))
    Bo = float(input("Enter Formation Volume Factor: "))
    Viscosity  = float(input("Enter fluid viscosity(cp): "))
    ct = float(input("Enter Total Compressibility(psi-1): "))
    A = float(input("Enter Drainage Area(Acres): "))
    re = np.sqrt(43560*A/3.14)
    rw = float(input("Enter Wellbore radius(ft): "))
    S = float(input("Enter Skin Factor: "))
##Calculation of productivity index
    J = K*h/(141.2*Bo*Viscosity*(np.log(re/rw)-0.75+S))
    print("The value of productivity index is", J)
##Calculation of Absolute open flow
    qmax = J*P/1.8
    print("The value of Absolute open flow is ", qmax, "stb/day")
##Pressures array
    a = np.arange(0,P,500)
    b = np.append(a,P)
    pwf = b[-1::-1]
##Calculation of flowrate
    flowrate = [] #empty list for occupying later
    for i in pwf:
        q = qmax*(1-0.2*(i/P)-0.8*((i/P)**2))
        flowrate.append(q)
    flowrates = np.array(flowrate)

##plotting IPR
    plt.figure(figsize = (9,6))
    plt.plot(flowrates,pwf,c = "red",linewidth=3)
    plt.xlabel("Flowrate(stb/day)")
    plt.ylabel("pwf(psia)")
    plt.grid(True)
    plt.title("Vogel's IPR for Saturated Reservoir")
```

```
In [65]: vogelipr()
```

```
Enter Porosity: 0.2
Enter Perm.(md): 9
Enter pay zone thicknes(Feet): 60
Enter  Reservoir Pressure(psi): 5500
Enter Bubble Point Pressure(psi): 5500
Enter Formation Volume Factor: 1.1
Enter fluid viscosity(cp): 1.7
Enter Total Compressibility(psi-1): 0.000129
Enter Drainage Area(Acres): 640
Enter Wellbore radius(ft): 0.328
Enter Skin Factor: 0
The value of productivity index is 0.24450472189461858
The value of Absolute open flow is  747.0977613446679 stb/day
```



2. Pressure Profile

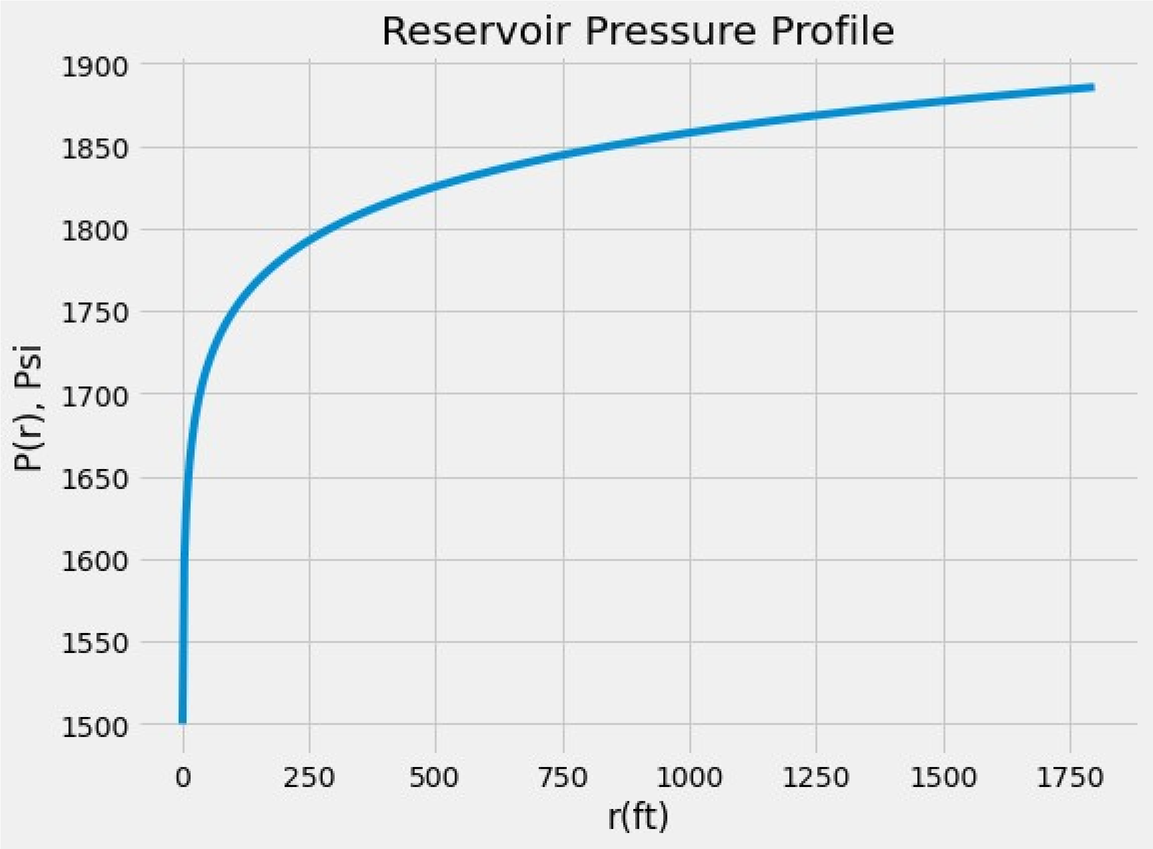


In [66]:

```python
def pressureprof():
    re = float(input('Outer radius of Reservoir(ft): '))
    rw = float(input('We4llbore Radius(ft): '))
    Pwf = float(input('Bottomhole Pressure(PSI): '))
    h = float(input('Net Pay Thickness(ft): '))
    k = float(input('Average Reservoir Permeability(mD): '))
    q = float(input('Flowrate(STB/Day): '))
    mu = float(input('Oil Viscosity: '))
    B = 1
    r = np.linspace(rw,re,500)
    Pressure = []
    for i in range(len(r)):
        P = Pwf + (141.2*q*mu*B*(np.log(r[i]/rw))/k/h)
        Pressure.append(P)
    plt.figure(figsize = [8,6])
    plt.plot(r,Pressure)
    plt.xlabel('r(ft)')
    plt.ylabel('P(r), Psi')
    plt.title('Reservoir Pressure Profile')
    plt.grid(True)
```

```
In [67]: pressureprof()
```

```
Outer radius of Reservoir(ft): 1800
We4llbore Radius(ft): 0.5
Bottomhole Pressure(PSI): 1500
Net Pay Thickness(ft): 60
Average Reservoir Permeability(mD): 150
Flowrate(STB/Day): 200
Oil Viscosity: 15
```



Dynamic Pressure Profile: Visualizing effect of Viscosity, Flowrate and permeability

```
In [68]: from ipywidgets import interact, interactive
         from IPython.display import clear_output, display, HTML
```

In [69]:

```
def flowprofile(k,mu,q):

  re = 3000
  rw = 0.5

  r = np.linspace(rw,re,500)

  pe = 4000

  B = 1

  h = 30 #ft

  P = pe - (141.2*q*mu*B*(np.log(re/r))/k/h)

  y_min = P[np.where(r==rw)]



  plt.plot(r,P,linewidth=4)
  plt.axhline(y_min,linewidth=3,color='red')

  plt.ylim(0,5000)

  plt.xlabel('r(ft)')
  plt.ylabel('P(r), Psi')

  plt.title('Reservoir Pressure Profile')

  plt.grid(True)

  return r,P
```

In [70]:

```
w = interactive(flowprofile, k = (200,1000), mu=(10,220), q = (100,20
0))
```

In [71]:

```
display(w)
```

Pandas

Ms Excel of Python but powerful This library helps us import | create | work with data in the form of tables.

The tables are called DataFrames.

1. We can directly convert a Dictionary into a DataFrame.
2. We can import excel-sheets or CSV files (most popular) into DF.
3. We can manipulate and use these tables in a user-friendly way.

In [72]:

```
#Converting Dictionary into DataFrame
#Step 1: Import Pandas with an alias 'pd'
import pandas as pd


#Step 2: Create your dictionary
Rock_Properties = {'phi': [0.2,0.40,0.30,0.25,0.270],
                   'perm': [100,20,150,130,145],
                   'lith': ['sandstone','shale','limestone','limestone','s
andstone']}

#Step 3: Create your Table.
rock_table = pd.DataFrame(Rock_Properties)

#Step 4: Print your table.
rock_table
```

Out[72]:

	phi	perm	lith
0	0.20	100	sandstone
1	0.40	20	shale
2	0.30	150	limestone
3	0.25	130	limestone
4	0.27	145	sandstone

In [73]:

```
#Adding New Column
rock_table['Saturation'] = [0.14,0.25,0.45,0.37,0.28]
rock_table
```

Out[73]:

	phi	perm	lith	Saturation
0	0.20	100	sandstone	0.14
1	0.40	20	shale	0.25
2	0.30	150	limestone	0.45
3	0.25	130	limestone	0.37
4	0.27	145	sandstone	0.28

In [74]:

```
#Importing from csv or excel files
volve = pd.read_csv('vpd.csv')
#Similarly excel file can be read by-
#df = pd.read_excel('\path\filename.csv')
```

In [75]:
```
volve
```

Out[75]:

	DATEPRD	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	ON_STREAM_HRS	AVG
0	07-Apr-14	7405	15/9-F-1 C	0.0	
1	08-Apr-14	7405	15/9-F-1 C	0.0	
2	09-Apr-14	7405	15/9-F-1 C	0.0	
3	10-Apr-14	7405	15/9-F-1 C	0.0	
4	11-Apr-14	7405	15/9-F-1 C	0.0	
...	...	...	...	...	
15629	14-Sep-16	5769	15/9-F-5	0.0	
15630	15-Sep-16	5769	15/9-F-5	0.0	
15631	16-Sep-16	5769	15/9-F-5	0.0	
15632	17-Sep-16	5769	15/9-F-5	0.0	
15633	18-Sep-16	5769	15/9-F-5	0.0	

15634 rows × 19 columns

In [76]:
```
volve.head()
```

Out[76]:

	DATEPRD	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	ON_STREAM_HRS	AVG_DO
0	07-Apr-14	7405	15/9-F-1 C	0.0	
1	08-Apr-14	7405	15/9-F-1 C	0.0	
2	09-Apr-14	7405	15/9-F-1 C	0.0	
3	10-Apr-14	7405	15/9-F-1 C	0.0	
4	11-Apr-14	7405	15/9-F-1 C	0.0	

In [77]:
```
#shape
volve.shape
```

Out[77]:
```
(15634, 19)
```

In [78]:
```
#columns to get output of columns name
volve.columns
```

Out[78]:
```
Index(['DATEPRD', 'NPD_WELL_BORE_CODE', 'NPD_WELL_BORE_NAME', 'ON_STR
EAM_HRS',
       'AVG_DOWNHOLE_PRESSURE', 'AVG_DOWNHOLE_TEMPERATURE', 'AVG_DP_T
UBING',
       'AVG_ANNULUS_PRESS', 'AVG_CHOKE_SIZE_P', 'AVG_CHOKE_UOM', 'AVG
_WHP_P',
       'AVG_WHT_P', 'DP_CHOKE_SIZE', 'BORE_OIL_VOL', 'BORE_GAS_VOL',
       'BORE_WAT_VOL', 'BORE_WI_VOL', 'FLOW_KIND', 'WELL_TYPE'],
      dtype='object')
```

In [79]:

```
volve['NPD_WELL_BORE_NAME'].value_counts()
```

Out[79]:

```
15/9-F-4       3327
15/9-F-5       3306
15/9-F-12      3056
15/9-F-14      3056
15/9-F-11      1165
15/9-F-15 D     978
15/9-F-1 C      746
Name: NPD_WELL_BORE_NAME, dtype: int64
```

In [80]:

```
#Conditional Dataframe Slicing
pf12 =  volve['NPD_WELL_BORE_NAME'] == '15/9-F-12' #Give Boolean
volve_pf12 = volve[pf12]
```

In [81]:

```
volve_pf12.head()
```

Out[81]:

	DATEPRD	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	ON_STREAM_HRS	AVG_
1911	12-Feb-08	5599	15/9-F-12	11.50	
1912	13-Feb-08	5599	15/9-F-12	24.00	
1913	14-Feb-08	5599	15/9-F-12	22.50	
1914	15-Feb-08	5599	15/9-F-12	23.15	
1915	16-Feb-08	5599	15/9-F-12	24.00	

In [82]:

```
#info: Information of datatypes and count of null values
volve_pf12.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3056 entries, 1911 to 4966
Data columns (total 19 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   DATEPRD                   3056 non-null   object 
 1   NPD_WELL_BORE_CODE        3056 non-null   int64  
 2   NPD_WELL_BORE_NAME        3056 non-null   object 
 3   ON_STREAM_HRS             3056 non-null   float64
 4   AVG_DOWNHOLE_PRESSURE     3050 non-null   float64
 5   AVG_DOWNHOLE_TEMPERATURE  3050 non-null   float64
 6   AVG_DP_TUBING             3050 non-null   float64
 7   AVG_ANNULUS_PRESS         3043 non-null   float64
 8   AVG_CHOKE_SIZE_P          3012 non-null   float64
 9   AVG_CHOKE_UOM             3056 non-null   object 
 10  AVG_WHP_P                 3056 non-null   float64
 11  AVG_WHT_P                 3056 non-null   float64
 12  DP_CHOKE_SIZE             3056 non-null   float64
 13  BORE_OIL_VOL              3056 non-null   float64
 14  BORE_GAS_VOL              3056 non-null   float64
 15  BORE_WAT_VOL              3056 non-null   float64
 16  BORE_WI_VOL               0 non-null      float64
 17  FLOW_KIND                 3056 non-null   object 
 18  WELL_TYPE                 3056 non-null   object 
dtypes: float64(13), int64(1), object(5)
memory usage: 477.5+ KB
```

In [83]:
```
volve_pf12.set_index(pd.to_datetime(volve_pf12['DATEPRD']),inplace = True)
```

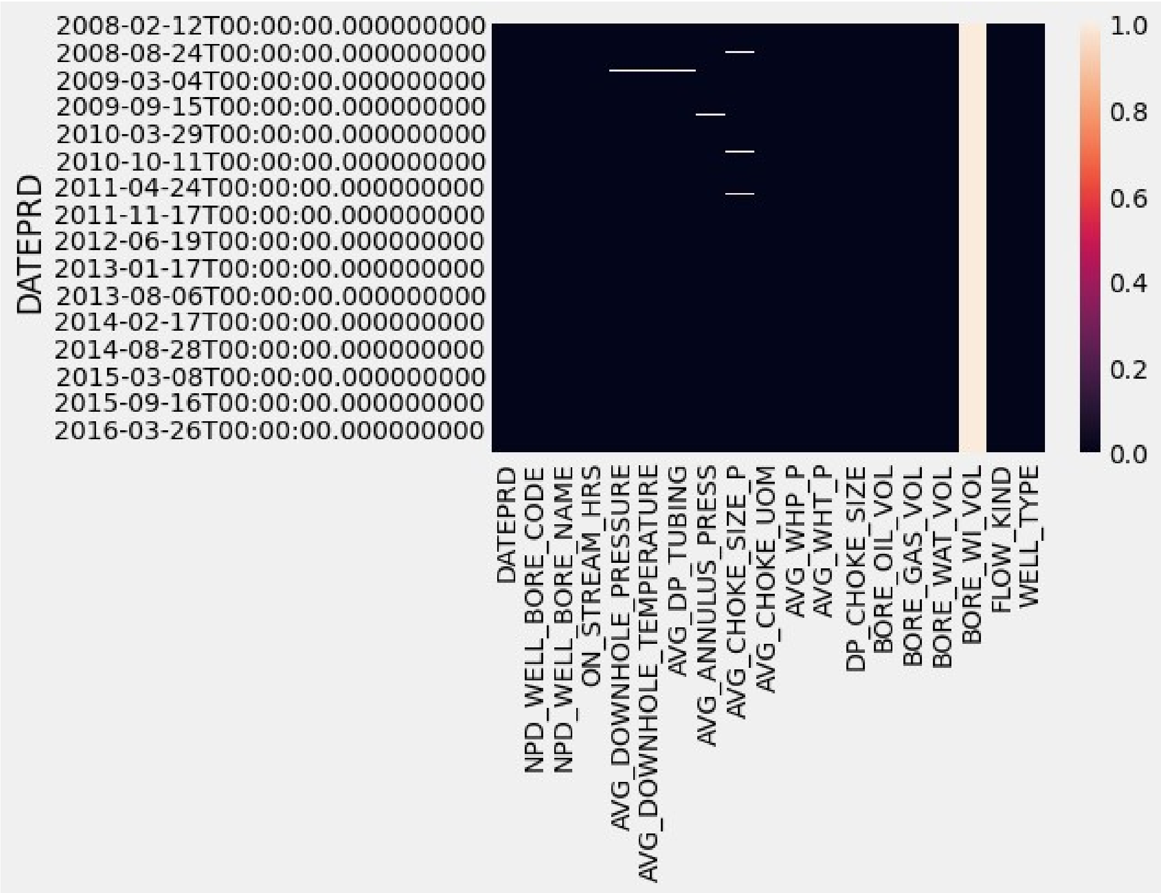
In [84]:
```
volve_pf12.head()
```

Out[84]:

	DATEPRD	NPD_WELL_BORE_CODE	NPD_WELL_BORE_NAME	ON_STREAM_HRS
DATEPRD				
2008-02-12	12-Feb-08	5599	15/9-F-12	11.50
2008-02-13	13-Feb-08	5599	15/9-F-12	24.00
2008-02-14	14-Feb-08	5599	15/9-F-12	22.50
2008-02-15	15-Feb-08	5599	15/9-F-12	23.15
2008-02-16	16-Feb-08	5599	15/9-F-12	24.00

In [85]:
```
import seaborn as sns
sns.heatmap(volve_pf12.isnull())
```

Out[85]:
```
<AxesSubplot:ylabel='DATEPRD'>
```



In [86]:

```
#Accessing a column
volve_pf12['AVG_DOWNHOLE_PRESSURE']
```

Out[86]:

```
DATEPRD
2008-02-12    308.056
2008-02-13    303.034
2008-02-14    295.586
2008-02-15    297.663
2008-02-16    295.936
               ...
2016-09-13      0.000
2016-09-14      0.000
2016-09-15      0.000
2016-09-16      0.000
2016-09-17      0.000
Name: AVG_DOWNHOLE_PRESSURE, Length: 3056, dtype: float64
```

In [87]:

```
volve_pf12[['AVG_DOWNHOLE_PRESSURE']]
```

Out[87]:

	AVG_DOWNHOLE_PRESSURE
DATEPRD	
2008-02-12	308.056
2008-02-13	303.034
2008-02-14	295.586
2008-02-15	297.663
2008-02-16	295.936
...	...
2016-09-13	0.000
2016-09-14	0.000
2016-09-15	0.000
2016-09-16	0.000
2016-09-17	0.000

3056 rows × 1 columns

In [88]:

```
a =volve_pf12[['AVG_DOWNHOLE_PRESSURE','BORE_OIL_VOL']]
```

In [89]:
```
a
```

Out[89]:

	AVG_DOWNHOLE_PRESSURE	BORE_OIL_VOL
DATEPRD		
2008-02-12	308.056	285.0
2008-02-13	303.034	1870.0
2008-02-14	295.586	3124.0
2008-02-15	297.663	2608.0
2008-02-16	295.936	3052.0
...	...	...
2016-09-13	0.000	0.0
2016-09-14	0.000	0.0
2016-09-15	0.000	0.0
2016-09-16	0.000	0.0
2016-09-17	0.000	0.0

3056 rows × 2 columns

In [90]:
```
#accessing through index number
volve_pf12.iloc[2]
```

Out[90]:
```
DATEPRD                        14-Feb-08
NPD_WELL_BORE_CODE                  5599
NPD_WELL_BORE_NAME             15/9-F-12
ON_STREAM_HRS                       22.5
AVG_DOWNHOLE_PRESSURE            295.586
AVG_DOWNHOLE_TEMPERATURE         105.775
AVG_DP_TUBING                    181.868
AVG_ANNULUS_PRESS                  12.66
AVG_CHOKE_SIZE_P                31.24997
AVG_CHOKE_UOM                          %
AVG_WHP_P                        113.718
AVG_WHT_P                         72.738
DP_CHOKE_SIZE                      80.12
BORE_OIL_VOL                      3124.0
BORE_GAS_VOL                    509955.0
BORE_WAT_VOL                         1.0
BORE_WI_VOL                          NaN
FLOW_KIND                     production
WELL_TYPE                             OP
Name: 2008-02-14 00:00:00, dtype: object
```

In [91]:

```
#index name
volve_pf12.loc['2008-02-14']
```

Out[91]:

```
DATEPRD                      14-Feb-08
NPD_WELL_BORE_CODE                5599
NPD_WELL_BORE_NAME           15/9-F-12
ON_STREAM_HRS                     22.5
AVG_DOWNHOLE_PRESSURE          295.586
AVG_DOWNHOLE_TEMPERATURE       105.775
AVG_DP_TUBING                  181.868
AVG_ANNULUS_PRESS                12.66
AVG_CHOKE_SIZE_P              31.24997
AVG_CHOKE_UOM                        %
AVG_WHP_P                      113.718
AVG_WHT_P                       72.738
DP_CHOKE_SIZE                    80.12
BORE_OIL_VOL                    3124.0
BORE_GAS_VOL                  509955.0
BORE_WAT_VOL                       1.0
BORE_WI_VOL                        NaN
FLOW_KIND                   production
WELL_TYPE                           OP
Name: 2008-02-14 00:00:00, dtype: object
```

In [92]:

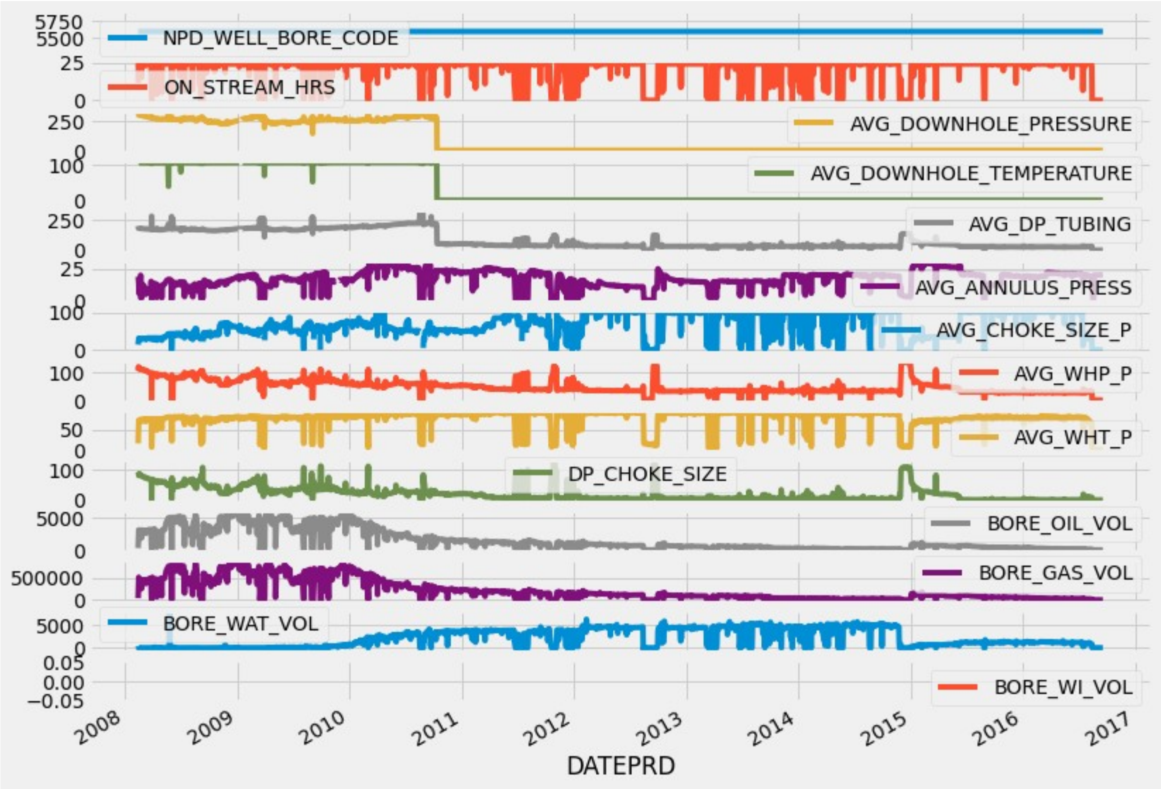
```
volve_pf12['AVG_DOWNHOLE_PRESSURE']['2008-02-14']
```

Out[92]:

```
295.586
```

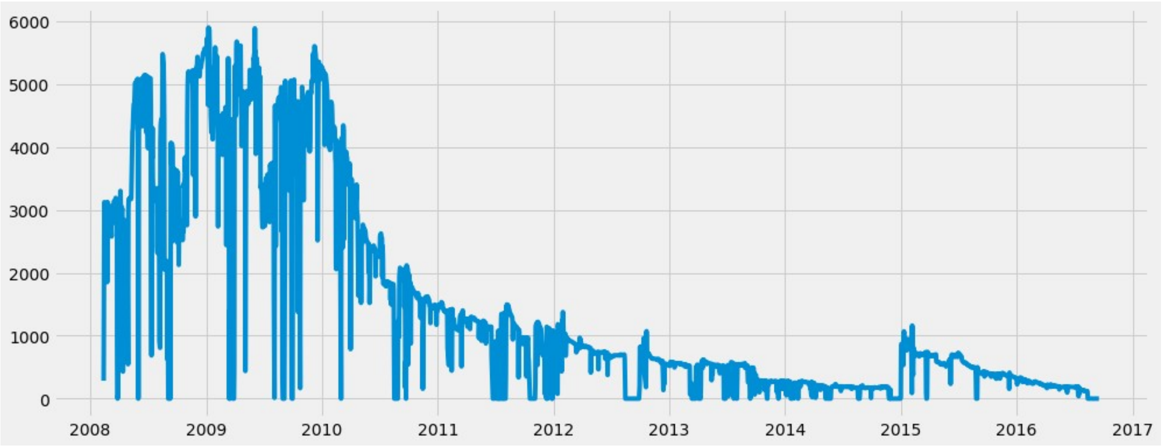
```
In [95]: #Plotting the values with value of date on x axis
         #inbuilt plot function of pandas
         volve_pf12.plot(figsize = (12,10),subplots= True)
```

```
Out[95]: array([<AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >,
                <AxesSubplot:xlabel='DATEPRD'>, <AxesSubplot:xlabel='DATEPRD'
         >],
               dtype=object)
```



```
In [97]: plt.figure(figsize=(15,6))
         plt.plot(volve_pf12.index,volve_pf12['BORE_OIL_VOL'])
```

```
Out[97]: [<matplotlib.lines.Line2D at 0x21721194f08>]
```



Thank You