Intoduction to Machine Learning for Petroleum Engineers.

> Divyanshu Vyas | Oil & Gas Data Scientist.

> LinkedIn : https://www.linkedin.com/in/divyanshu-vyas/

> Email : dvyas13ad@gmail.com

Agenda :

1. Python : Lists, Dictionaries, Loops, Functions.
2. Data Analysis : NumPy & Pandas.
3. Data Visualization : Matplotlib
4. A Brief Talk on Machine Learning Algorithms.
5. Hands on Excercises :-

> A. Hands on Practice : DCA with Python

> B. Hands on Practice : Machine Learning a Popular $\phi - K$ correlation.

> C. Hands on Practice : Machine Learning based Artificial Lift Selection.

1. Python Lists

> Mutable: You can index and change elements.

```
#The Following are different types of lists.

name = 'divyanshu'
#list of strings.
litho_names = ['SST' , 'LST' , 'SH' , 'DOLO']

#list of porosities
porosities = [0.25 , 0.13 , 0.38 , 0.17]

#list of permeabilities.
perms_md = [100 , 2 , 0.5 , 3.5 ]
```

Accessing the Data in Lists.

```
numbers = [1,2,4,5,6,7,8,12,12,13,45,56,200]

len(numbers)
```

```
13
```

```
numbers[0] , numbers[1] , numbers[2] , numbers[3] #... and so on
```

```
(1, 2, 4, 5)
```

The folliwing is called List Slicing. listname[start:st0p:step]

```
numbers[0:5]
```

```
[1, 2, 4, 5, 6]
```

```
#List reversal
# NAMAN

original = [1,2,3,4,5,6]

reversed_list = original[-1::-1]
```

```
reversed_list
```

```
[6, 5, 4, 3, 2, 1]
```

2. Python Dictionaries

{key : value}

```
rock_details = {'sst1': 0.25 , 'sst2': 0.30 , 'lst1': 0.12 , 'lst2': 16}

rock_details
```

```
{'lst1': 0.12, 'lst2': 16, 'sst1': 0.25, 'sst2': 0.3}
```

```
reservoir_data = {'porosities': [0.24, 0.25 , 0.34 , 0.44] ,
                  'perms_md' : [120, 100, 60 , 500]}

reservoir_data['porosities']
```

```
[0.24, 0.25, 0.34, 0.44]
```

3. Python : Loops & Iterations.

```
porosities
```

```
[0.25, 0.13, 0.38, 0.17]
```

```
for i in porosities:

  print(i)

# print
```

```
0.25
0.13
0.38
0.17
```

```
litho_names
```

```
['SST', 'LST', 'SH', 'DOLO']
```

```
for i in range(len(litho_names)):

  print(f'Name{i+1} is {litho_names[i]}')
```

```
Name1 is SST
Name2 is LST
Name3 is SH
Name4 is DOLO
```

```
for i in range(10):
  print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

```
mylist = []
for i in range(100):
  mylist.append(i)

print(mylist)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70
```

4. Python Functions - VVVVIMP!

```
def funcname( parameter1 , parameter2):

  res = parameter1 + parameter2

  return res
```

```
funcname(1,2)
```

```
3
```

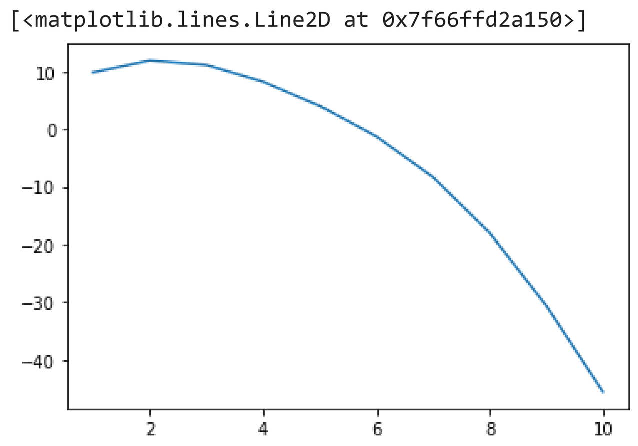
```
# Suppose we have a function f(x) = -x^2 + 5x + sinx + 5
# We want to create a function that returns y for each x according to this mapping.

def f(x):

  import math

  y = -x**2 + 5*x + math.sin(x) + 5

  return y
```

```
f(3)
```

```
11.141120008059868
```

```
#Optional. What if we are interested in seeing the graph of this function.
import matplotlib.pyplot as plt

X = [1,2,3,4,5,6,7,8,9,10]

Y= []

for i in X:

  Y.append(f(i))

plt.plot(X,Y)
```

```
[<matplotlib.lines.Line2D at 0x7f66ffd2a150>]
```



5. Data Analysis (NumPy and Pandas) & Visualization (Matplotlib)

> NumPy : Numerical Python : Built on C++. Superfast. "NumPy Arrays"

> Pandas : Excel of Python : Built on Top of NumPy. "Pandas DataFrame".

1. NumPy

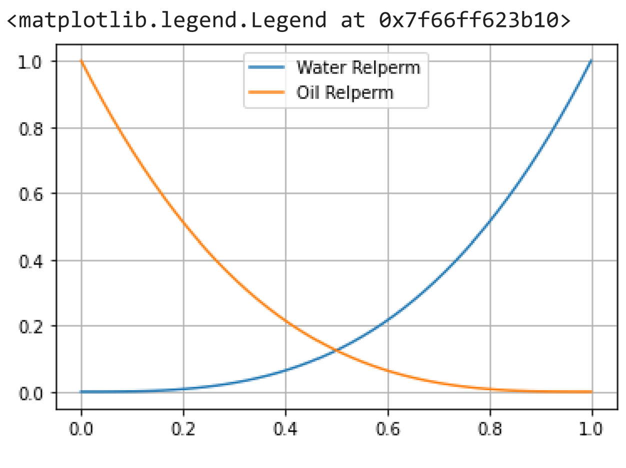
```
import numpy as np
```

```
num1 = [1,2,3,4,5]
num2 = [5,6,7,8,9]

#Lets first Directly convert the list to NumPy arrays -> use np.array(listname)
arr1 = np.array(num1)
arr2 = np.array(num2)

#And Let's now solve the List Disadvantages using NumPy
print(arr1)
print(arr2)
print(arr1*arr2)
print(arr1 + arr2)
```

```
[1 2 3 4 5]
[5 6 7 8 9]
[ 5 12 21 32 45]
[ 6  8 10 12 14]
```

```
#linspace creates number of points between a start and an end point.
Sw = np.linspace(0,1,50)

krw = Sw**3

kro = (1-Sw)**3

#plotting
plt.plot(Sw,krw,label='Water Relperm')
plt.plot(Sw,kro,label='Oil Relperm')
plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f66ff623b10>
```




```
Sw
```

```
array([0.        , 0.02040816, 0.04081633, 0.06122449, 0.08163265,
       0.10204082, 0.12244898, 0.14285714, 0.16326531, 0.18367347,
       0.20408163, 0.2244898 , 0.24489796, 0.26530612, 0.28571429,
       0.30612245, 0.32653061, 0.34693878, 0.36734694, 0.3877551 ,
       0.40816327, 0.42857143, 0.44897959, 0.46938776, 0.48979592,
       0.51020408, 0.53061224, 0.55102041, 0.57142857, 0.59183673,
       0.6122449 , 0.63265306, 0.65306122, 0.67346939, 0.69387755,
       0.71428571, 0.73469388, 0.75510204, 0.7755102 , 0.79591837,
       0.81632653, 0.83673469, 0.85714286, 0.87755102, 0.89795918,
       0.91836735, 0.93877551, 0.95918367, 0.97959184, 1.        ])
```

Suppose you want to create an array x where each value is center of a grid block.

```
dx = 10 #ft

np.arange(0,110,dx)
```

```
array([  0,  10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

```
#Will be useful for Reservoir simulation. Space-Time Arrays.

#Suppose you want to build an array for days. from t = 0th day to t= 500th day.
#Time step = 20 days.

t = np.arange(0,520,20)

t
```

```
array([  0,  20,  40,  60,  80, 100, 120, 140, 160, 180, 200, 220, 240,
       260, 280, 300, 320, 340, 360, 380, 400, 420, 440, 460, 480, 500])
```

Universal Functions Make Scientific computations easy using NumPy : https://www.w3schools.com/python/numpy_ufunc.asp

```
np.log2(t)
```

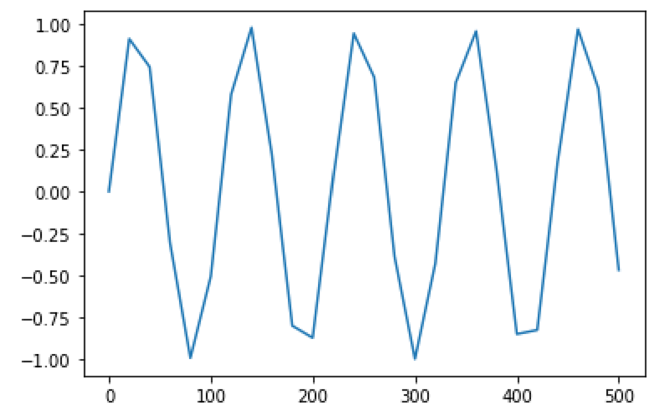
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in log2
  """Entry point for launching an IPython kernel.
array([      -inf, 4.32192809, 5.32192809, 5.9068906 , 6.32192809,
       6.64385619, 6.9068906 , 7.12928302, 7.32192809, 7.4918531 ,
       7.64385619, 7.78135971, 7.9068906 , 8.02236781, 8.12928302,
       8.22881869, 8.32192809, 8.40939094, 8.4918531 , 8.56985561,
       8.64385619, 8.71424552, 8.78135971, 8.84549005, 8.9068906 ,
       8.96578428])
```

```
np.sin(t)
```

```
array([ 0.        ,  0.91294525,  0.74511316, -0.30481062, -0.99388865,
       -0.50636564,  0.58061118,  0.98023966,  0.21942526, -0.80115264,
       -0.8732973 ,  0.08839871,  0.94544515,  0.6832397 , -0.38780942,
       -0.99975584, -0.42815543,  0.65031074,  0.95891572,  0.13232187,
       -0.85091936, -0.82681172,  0.17610529,  0.97054255,  0.61601671,
       -0.46777181])
```

```
plt.plot(t,np.sin(t))
```

```
[<matplotlib.lines.Line2D at 0x7f66ff5cdb90>]
```



2. Pandas

```
#Step 1: Import Pandas with an alias 'pd'
import pandas as pd


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

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

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

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

```
df1 = pd.read_csv('/content/sample_data/california_housing_train.csv')


#Similarly excel file can be read by-
#df = pd.read_excel('\path\filename.csv')

df1.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Description-

Note that the Pandas DF has two parts-

The vertical Columns. The Horizontal Rows.

```
df1.shape
```

```
(17000, 9)
```

Descriptive Statistics of each feature.

```
df1.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Accessing Columns

```
df1.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

```
rock_table.columns
```

```
Index(['phi', 'perm', 'lith'], dtype='object')
```

```
rock_table[['phi','perm']]
```

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

Accessing Rows.

```
#Accessing 5th to 9th index rows, and all columns.
df1.iloc[5:10, :]
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
5	-114.58	33.63	29.0	1387.0	236.0	671.0	239.0	3.3438	74000.0
6	-114.58	33.61	25.0	2907.0	680.0	1841.0	633.0	2.6768	82400.0
7	-114.59	34.83	41.0	812.0	168.0	375.0	158.0	1.7083	48500.0

loc and iloc

" l for label , i for index"

```
df1.iloc[0:5 , 0:3]
```

	longitude	latitude	housing_median_age
0	-114.31	34.19	15.0
1	-114.47	34.40	19.0
2	-114.56	33.69	17.0
3	-114.57	33.64	14.0
4	-114.57	33.57	20.0

```
df1.loc[0:4, 'longitude':'housing_median_age']
```

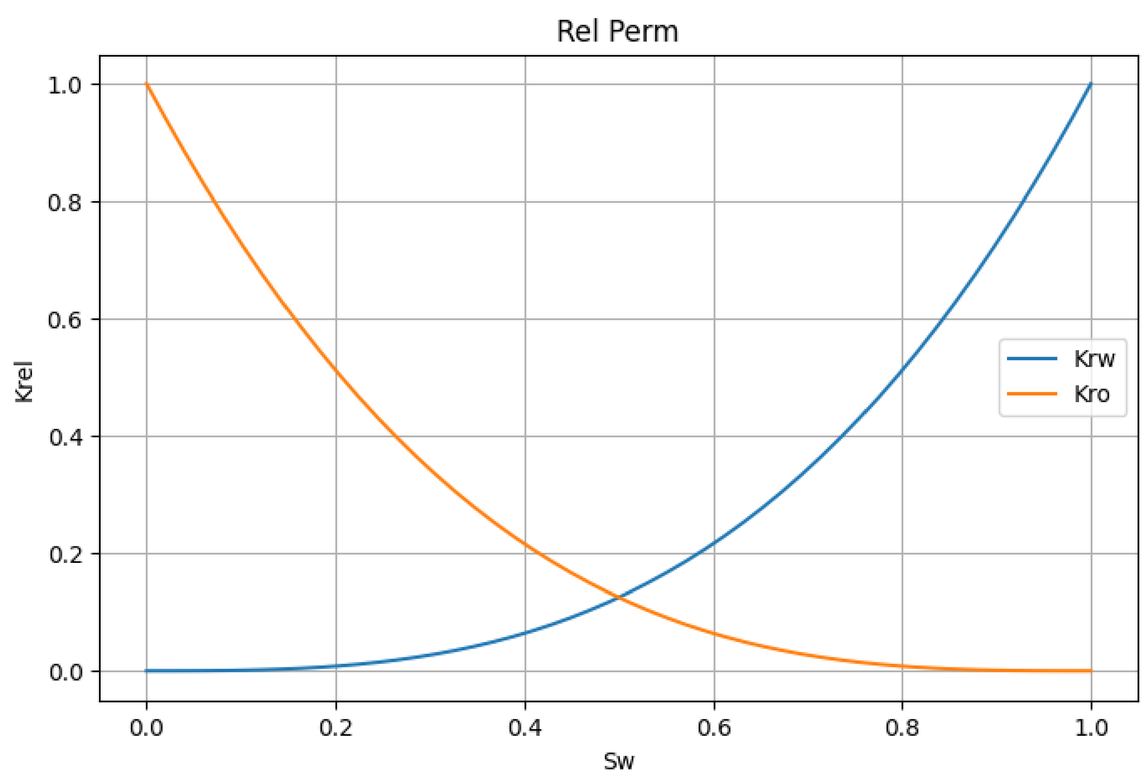
	longitude	latitude	housing_median_age
0	-114.31	34.19	15.0
1	-114.47	34.40	19.0
2	-114.56	33.69	17.0
3	-114.57	33.64	14.0
4	-114.57	33.57	20.0

3. Plotting : F S P L L G

1. F- Figure (figsize=(h,v))
2. S- Style (default)
3. P - Plot(plt.plot(x,y,label=')
4. L- Labels for axes (plt.xlabel)
5. L - Legend (plt.legend)
6. G - Grid (plt.grid())

```
#1. F : figsize
plt.figure(figsize=(8,5))

#2. S : Style
plt.style.use('default')

#3. P : Plot
plt.plot(Sw,krw,label='Krw')
plt.plot(Sw,kro,label='Kro')

#4. L : Labels
plt.xlabel('Sw') ; plt.ylabel('Krel')

#5. Legend
plt.legend()

#6. Grid
plt.grid()

#7. title
plt.title('Rel Perm')
```

```
Text(0.5, 1.0, 'Rel Perm')
```



Data Analysis Excercise : Fractional Flow

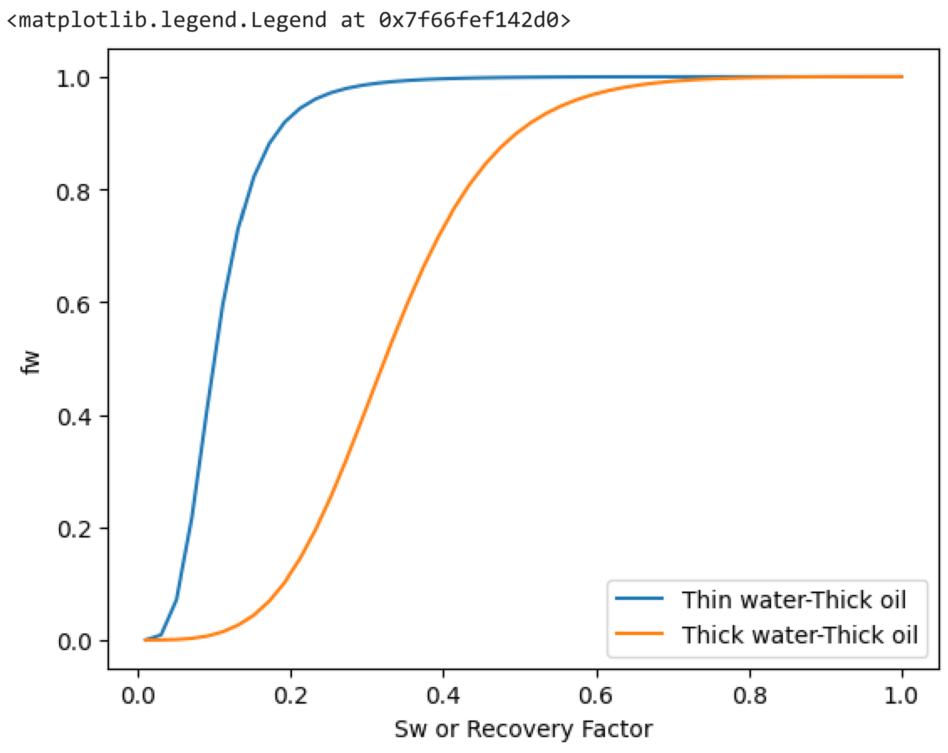
> fw = 1/(1 + 1/M) M = (krw/mu_w)/(kro/mu_o)

Assume mu_o = 1000 cp (heavy oil) mu_w = 1 cp

```
import warnings
warnings.filterwarnings('ignore')
```

```
sw = np.linspace(0.01,1,50)

mu_o = 1000

mu_w = 1

M1 = (krw/mu_w)/(kro/mu_o)

fw = 1/(1 + (1/M1))

plt.plot(sw,fw,label = 'Thin water-Thick oil')

M2 = (krw/100)/(kro/mu_o)

fw2 = 1/(1+(1/M2))

plt.plot(sw,fw2,label='Thick water-Thick oil')


plt.xlabel('Sw or Recovery Factor') ; plt.ylabel('fw')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f66fef142d0>
```



6. Machine Learning : Layman Introduction.

1. Supervised - Regression

```
df_example = pd.DataFrame({'phi':[0.1,0.15,0.2,0.25,0.3,0.35],
 'pore radius': [5,10,15,20,25,30],
 'k': [80,100,120,150,180,'?']})


df_example

#this becomes a regression problem
```

	phi	pore radius	k
0	0.10	5	80
1	0.15	10	100
2	0.20	15	120
3	0.25	20	150
4	0.30	25	180
5	0.35	30	?

Resulting model : $K = m_1\phi + m_2r + c$

> 2. Supervised - Classification

```
rock_labels = pd.DataFrame({'k_md':[100,150,200,5,90,6,80,4],
 'phi': [0.25,0.27,0.22,0.44,0.26,0.38,0.21,0.34],
 'RT(ohm-m)': [100,110,120,10,89,15,80,20],
 'lith':['sst','sst','sst','shale','sst','shale','sst','???']})


rock_labels
#KNN | K-Means
```

	k_md	phi	RT(ohm-m)	lith
0	100	0.25	100	sst
1	150	0.27	110	sst
2	200	0.22	120	sst
3	5	0.44	10	shale
4	90	0.26	89	sst
5	6	0.38	15	shale
6	80	0.21	80	sst
7	4	0.34	20	???

Steps in ML :-

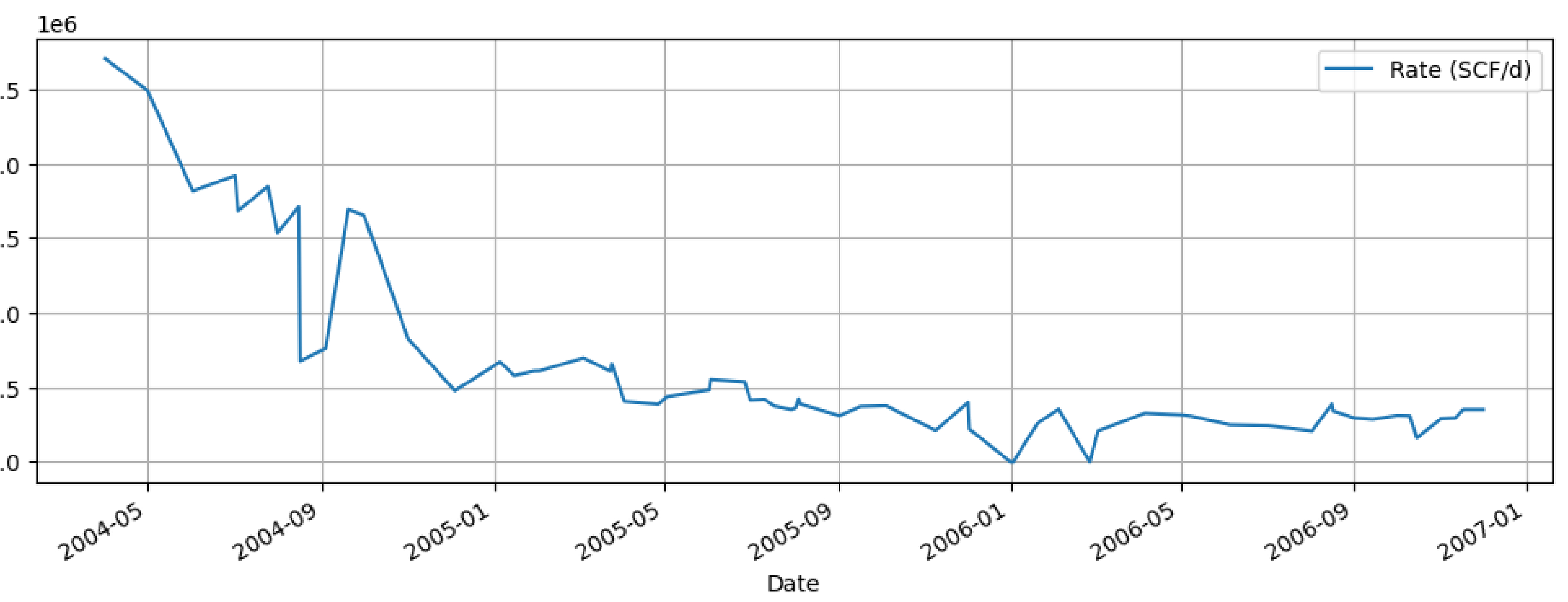
1. Import Data.
2. Check for missing values and abnormal values.
3. Accordingly process the data and make it usable. STEPS in a Machine Learning Project-
4. Perform EDA - Exploratory Data Analysis. Check for which features are not important and which can be excluded.
5. Record Visual stories of your data, to be presented.
6. now pick a ML algrithm suitable to your case.
7. Split the Data into Training, Validation and Test Data.
8. Fit the ML-model into the training data.
9. Perform predictions and validate, and make modifications based on the validation performance.
10. Finally Use real, unseen data and make predictions (Production)

Hands on Excercise 1 : DCA with Python

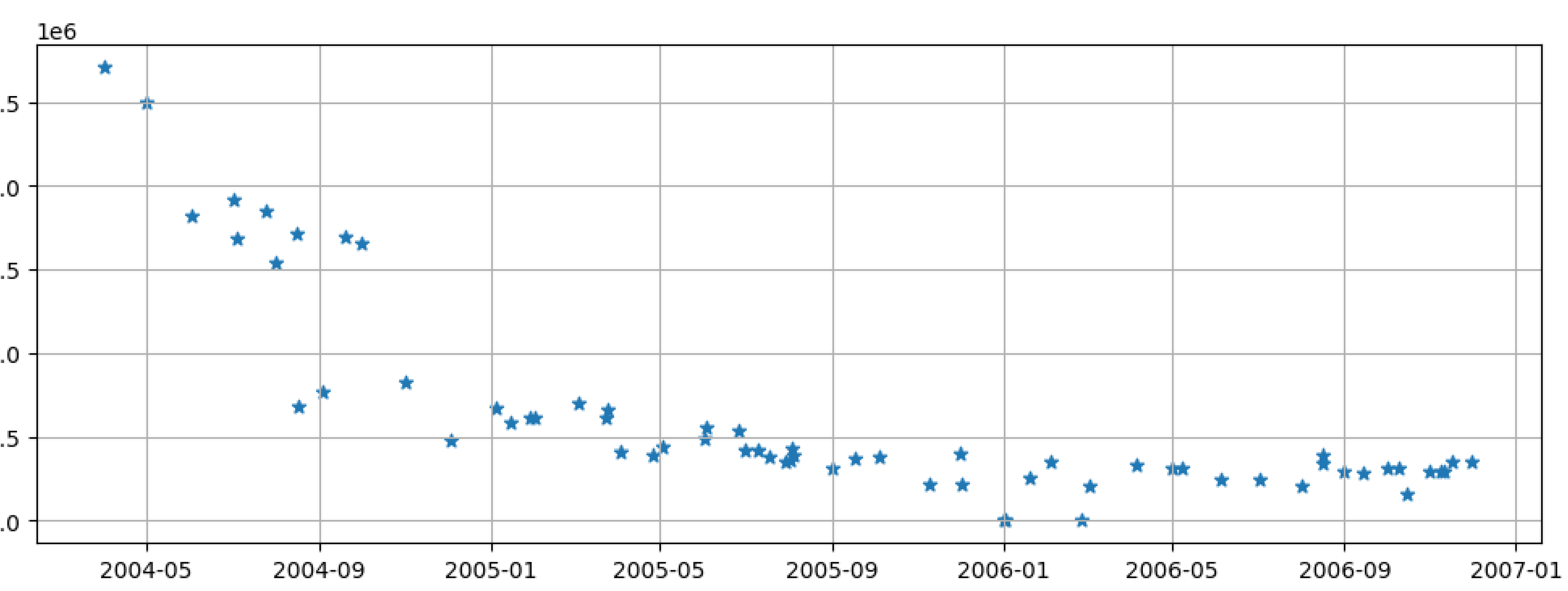
```
dca_df = pd.read_csv('https://raw.githubusercontent.com/yohanesnuwara/pyreservoir/master/data/norne_production_rate_sample.csv',
                     index_col = 0 , parse_dates = True)

dca_df.head()
```

	Rate (SCF/d)
Date	
2004-04-01	2706039.0
2004-05-01	2492086.2
2004-06-02	1816846.1
2004-07-02	1920207.4
2004-07-04	1683521.4

```
dca_df.plot(figsize=(12,4))
plt.grid()
```




```
plt.figure(figsize=(12,4))
plt.scatter(y=dca_df['Rate (SCF/d)'],x=dca_df.index,marker='*')

# plt.ylabel('Rate (SCF/d)')
plt.grid()
```



Step 1 : Convert Dates into Days (t)

```
def day_maker(df):
    '''
    Pass a Time-Series DataFrame to it and it will
```

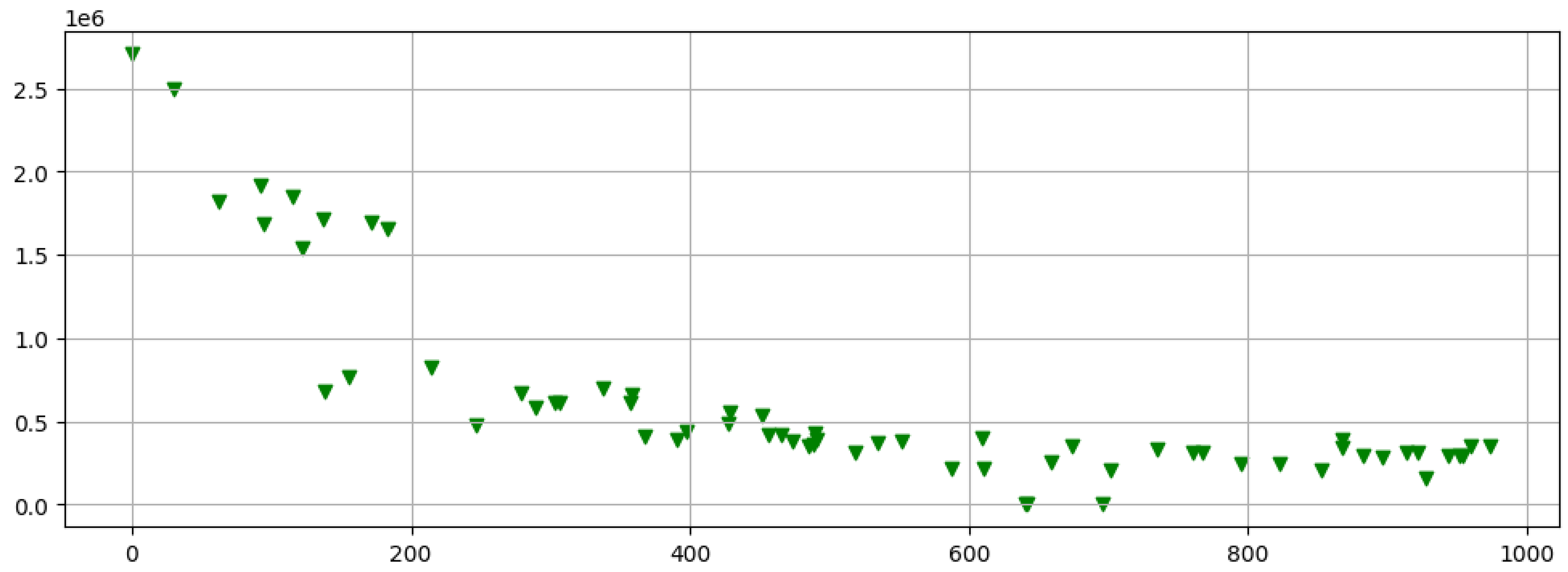
```
    return a days column. Subtracts dates and makes days.

    Returned is a days (np array).
    '''

    days = []

    for d in range(len(df)):

        delta = df.index[d] - df.index[0]

        days.append(delta.days)

    days = np.array(days)

    return days
```

```
dca_df['days'] = day_maker(dca_df)
```

```
dca_df.head()
```

	Rate (SCF/d)	days
Date		
2004-04-01	2706039.0	0
2004-05-01	2492086.2	30
2004-06-02	1816846.1	62
2004-07-02	1920207.4	92
2004-07-04	1683521.4	94

```
plt.figure(figsize=(12,4))
plt.scatter(y=dca_df['Rate (SCF/d)'],x=dca_df['days'],marker='v',color='green')

# plt.ylabel('Rate (SCF/d)')
plt.grid()
```



Step 2 : Hyperbolic Model Function & Curve Fitting methodology.

```
from scipy.optimize import curve_fit
```

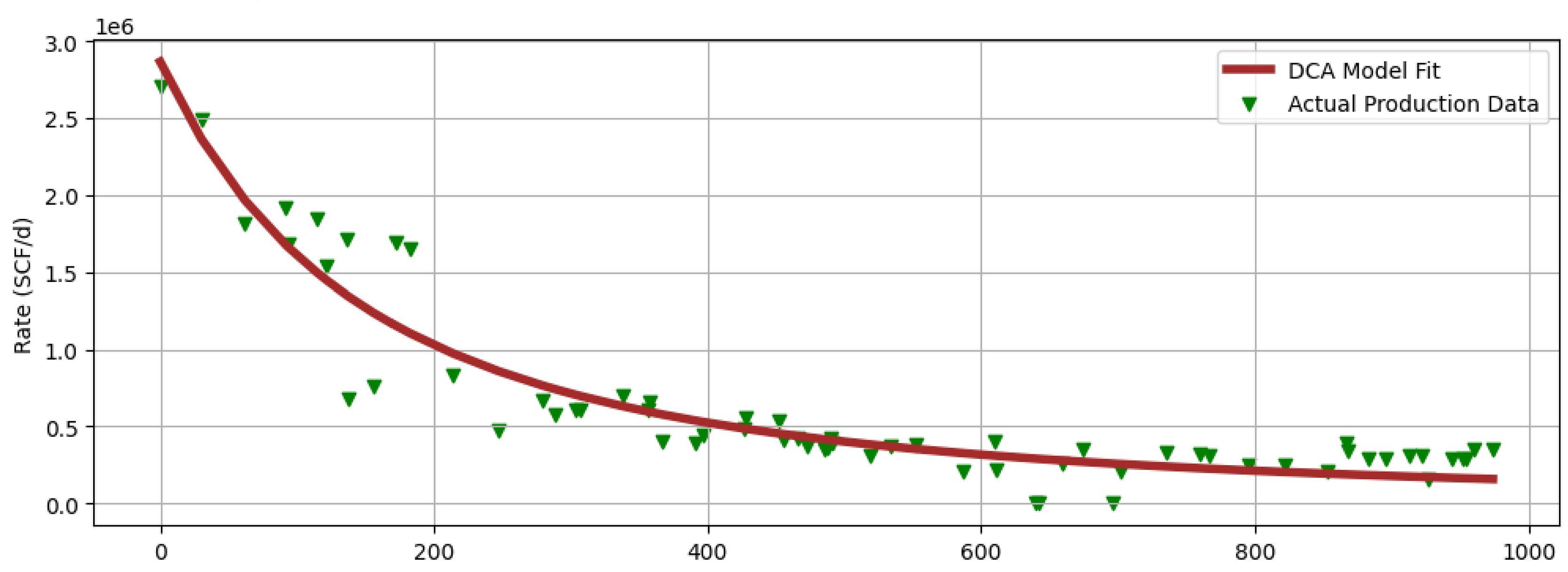
```
def q_hyp(t,qi,b,d):

    qfit = qi/(np.abs((1 + b * d* t))**(1/b))

    return qfit


def hyp_fitter(q,t):

    #First we have to Normalize so that it converges well and quick.
    q_n = q/max(q)
    t_n = t/max(t)

    #curve-fit (optimization of parameters)
    params = curve_fit(q_hyp,t_n,q_n)
    [qi,b,d] = params[0]

    #These are for normalized t and q.
    #We must re-adjust for q and t (non-normalized)
    d_f = d/max(t)
    qi_f = qi*max(q)

    #Now we can use these parameters.
    q_hyp_fit = q_hyp(t,qi_f,b,d_f)

    return q_hyp_fit,params
```

```
#This stepwise approach gives you a very nice picture about how linear regression is implemented.
```

```
q = dca_df['Rate (SCF/d)'] ; t = dca_df['days']
q_fit ,params = hyp_fitter(q,t)
```

```
plt.figure(figsize=(12,4))
plt.scatter(t,q,marker='v',color='green',label='Actual Production Data')
plt.plot(t,q_fit,color='brown',lw=4,label='DCA Model Fit')

plt.ylabel(df.columns[0])

plt.grid()
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f66efc9d6d0>
```



Notice that in this DCA technique (Physics base, Arps) we MUST know the equation prior to the project. We need that info otherwise we cannot do anything.

Whereas in Data Driven approaches, we normally start with ONLY DATA and fit the model that suits the best, use this model for Production Forecasting.

Best Suited Model for this : Time Series Forecasting (ARIMA/Prophet etc.)

Hands on Excercise 2 : Machine Learning for Phi-K relationship modelling.

> A Supervised Regression Problem

Step 1 : Import the Dataset

```
df = pd.read_csv('https://raw.githubusercontent.com/Divyanshu-ISM/Machine-Learning-Deep-Learning/main/PhiK.csv',
                 index_col=0)
```

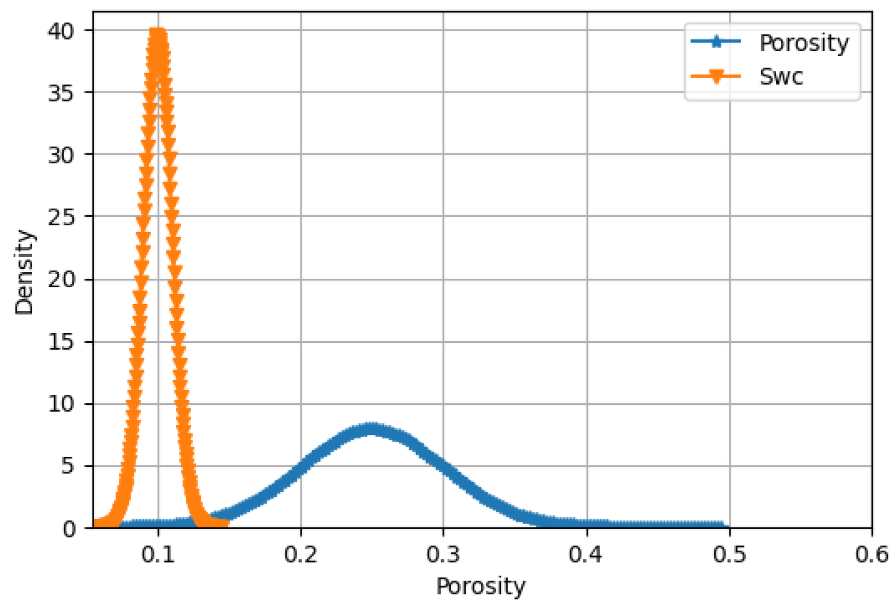
```
df.head()
```

	Porosity	Swc	Permeability(D)
0	0.269158	0.114209	2.042529
1	0.324275	0.072078	11.639989
2	0.218003	0.101849	1.015917
3	0.211875	0.099354	0.941715
4	0.322281	0.083444	8.452433

Step 2 : Exploratory Data Analysis

```
import seaborn as sns
plt.style.use('default')

plt.figure(figsize=(6,4))

sns.kdeplot(df['Porosity'],label='Porosity',marker='*')
sns.kdeplot(df['Swc'],label='Swc',marker='v')

plt.xlim(min(df['Swc']), 0.6)

plt.grid()

plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7f66e44f3550>
```



Step 3 : Train-Test split

```
from sklearn.model_selection import train_test_split

X = df[['Porosity', 'Swc']]

y = df['Permeability(D)']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=101)
```

Step 4 : Machine Learning Implementation : sklearn

```
from sklearn.tree import DecisionTreeRegressor
```

```
model = DecisionTreeRegressor(random_state=1)
```

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

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', 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='deprecated',
                      random_state=1, splitter='best')
```

5. Step 5 : Model Predictions

```
y.head()
```

```
0     2.042529
1    11.639989
2     1.015917
3     0.941715
4     8.452433
Name: Permeability(D), dtype: float64
```

```
print(model.predict(X.head()))
```

```
[ 2.04252923 11.69904489  1.02501243  0.94171488  8.45243294]
```

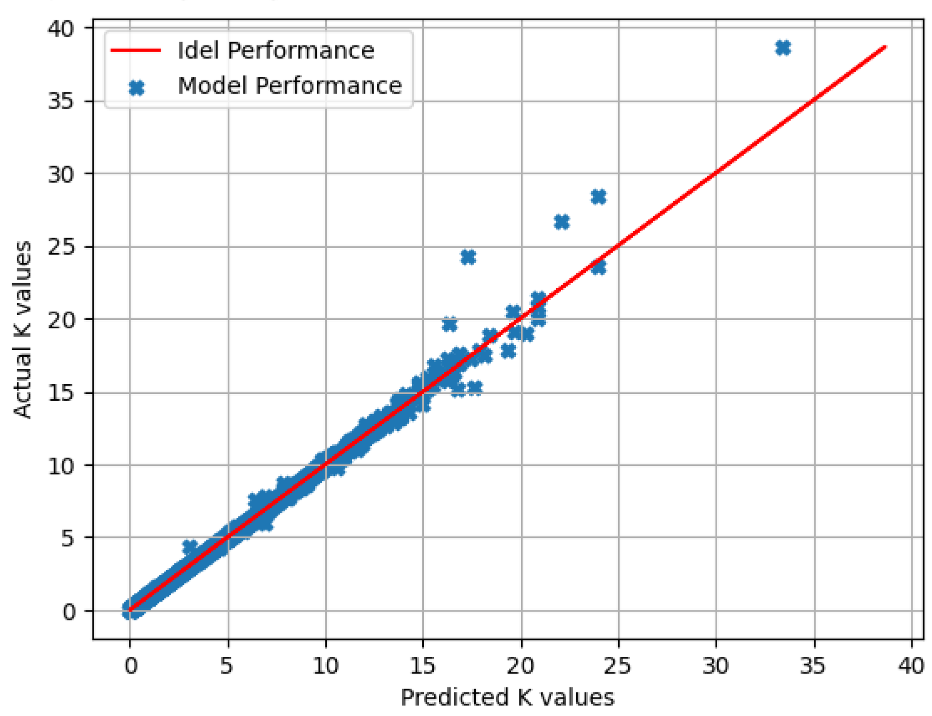
Performance evalauation : Visual and Quantitative.

```
from sklearn.metrics import mean_squared_error as mse
```

```
y_p = model.predict(X_test)
```

```
plt.grid()
plt.scatter(y_p,y_test,marker='X',label='Model Performance')

plt.plot(y_test,y_test, color='red',label='Idel Performance')

plt.xlabel('Predicted K values')
plt.ylabel('Actual K values')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f66df90fa10>
```




```
from sklearn import metrics
MAE =metrics.mean_absolute_error(y_test,y_p)
MSE = metrics.mean_squared_error(y_test,y_p)
RMSE = np.sqrt(MSE)

evaluation = pd.DataFrame(data =[MAE*100,MSE*100,RMSE*100], index='MAE(%) MSE(%) RMSE(%)'.split(), columns = ['Evaluation Values'])
evaluation
```

	Evaluation Values
MAE(%)	2.155681
MSE(%)	1.322553
RMSE(%)	11.500231

Hands on Excercise 3 : ALS Selection with ML

> A Supervised Classification Problem.

Source Code : https://github.com/Divyanshu-ISM/Machine-Learning-Deep-Learning/blob/main/Machine%20Learning%20for%20ALS.ipynb

✓ 0s completed at 3:58 PM