Example Book

V1.0

Nashat Jumaah Omar

Python For Oil & Gas

Python for Oil & Gas

Example Guide

Nashat Jumaah Omar

Table Of Content

Chapter 1: Reservoir Engineering -RE

Example 1: Chan Diagnostic Plot.

This Type of plot is used to distinguish the water production source in oil producing wells. Refer to paper (SPE-30775) for more information about this methodology.

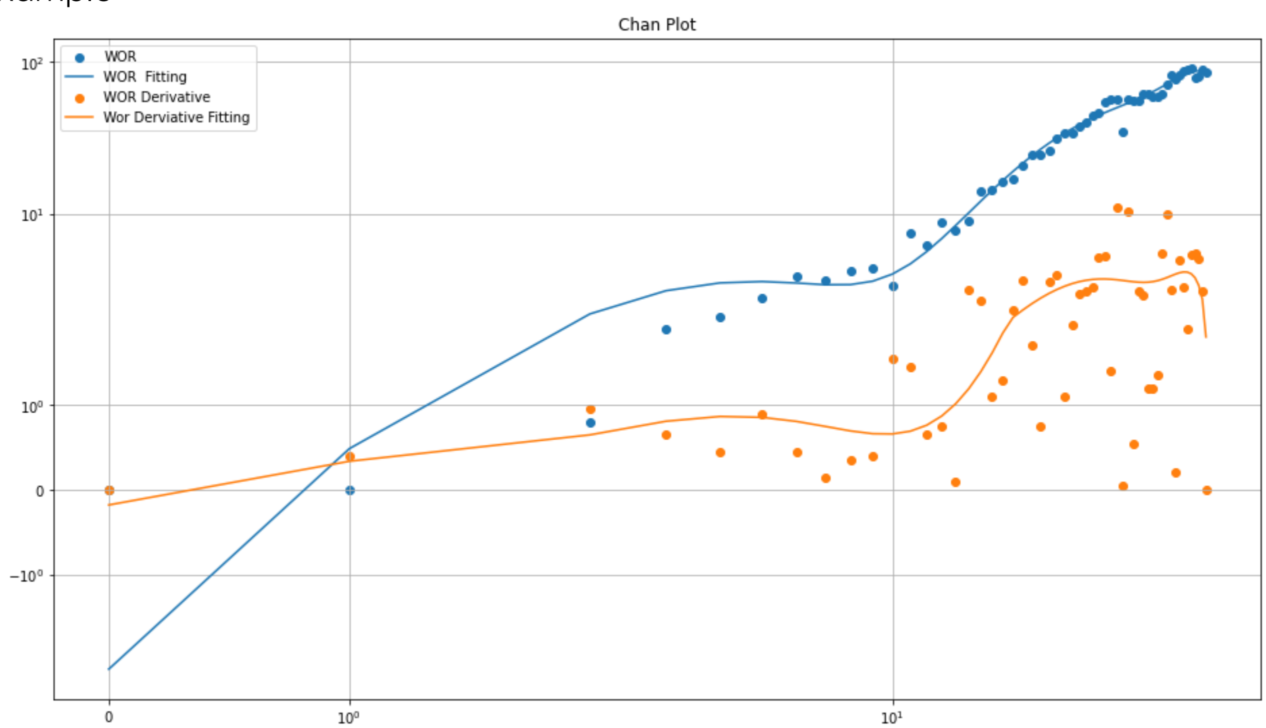
Required Data:

- Water Oil Ratio. Vs Date, Days or Months

What is Plotted:

- Water Oil Ratio.
- Water Oil Ratio Derivative.

Plot Example



Python Code

```
from matplotlib import pyplot as plt
import pandas as pd
plt.figure(figsize=(10,10))
import numpy as np

from matplotlib import pyplot as plt

import pandas as pd
plt.figure(figsize=(10,10))
import numpy as np
import math
#Import Data
data = pd.read_csv(r'wor.txt',sep='\t')
#Create Wor Derivative Column and assign it to zero value
data["Wor_"] = [0] * len(data)
#calculate derivative
for i in range(1, len(data)-1):
    data["Wor_"][i] = abs((data['Wor'][i+1] - data['Wor'][i-1])/(data['Month'][i+1] -
data['Month'][i-1]))
#Plot Historical Wor
plt.scatter(data['Month'],data['Wor'],label='WOR')
plt.yscale('symlog')
plt.xscale('symlog')
plt.grid()
polyfunc = np.poly1d(np.polyfit(data['Month'],data['Wor'],deg=6))
plt.plot(data.Month,polyfunc(data.Month),label='WOR  Fitting')
#Plot Derivative
plt.scatter(data['Month'],data['Wor_'],label='WOR Derivative')
polyfunc = np.poly1d(np.polyfit(data['Month'],data['Wor_'],deg=6))
plt.plot(data.Month,polyfunc(data.Month),label='Wor Derviative Fitting')
#Finishup the plot Title
plt.title("Chan Plot")#Set Title
plt.legend()#Show Legend

import math
#Import Data
data = pd.read_csv(r'wor.txt',sep='\t')
#Create Wor Derivative Column and assign it to zero value
data["Wor_"] = [0] * len(data)
#calculate derivative
for i in range(1, len(data)-1):
    data["Wor_"][i] = abs((data['Wor'][i+1] - data['Wor'][i-1])/(data['Month'][i+1] -
data['Month'][i-1]))
#Plot Historical Wor
plt.scatter(data['Month'],data['Wor'],label='WOR')
plt.yscale('symlog')
plt.xscale('symlog')
plt.grid()
polyfunc = np.poly1d(np.polyfit(data['Month'],data['Wor'],deg=6))
plt.plot(data.Month,polyfunc(data.Month),label='WOR  Fitting')
#Plot Derivative
plt.scatter(data['Month'],data['Wor_'],label='WOR Derivative')
polyfunc = np.poly1d(np.polyfit(data['Month'],data['Wor_'],deg=6))
plt.plot(data.Month,polyfunc(data.Month),label='Wor Derviative Fitting')
#Finishup the plot Title
plt.title("Chan Plot")#Set Title
plt.legend()#Show Legend
```

Example 2: Reverse Injectivity Index

Reverse injectivity index is a diagnostics plot to evaluate the efficiency of an injection well over a period of time, loss of injectivity is usually to skin or water being injected in an appropriate formation.

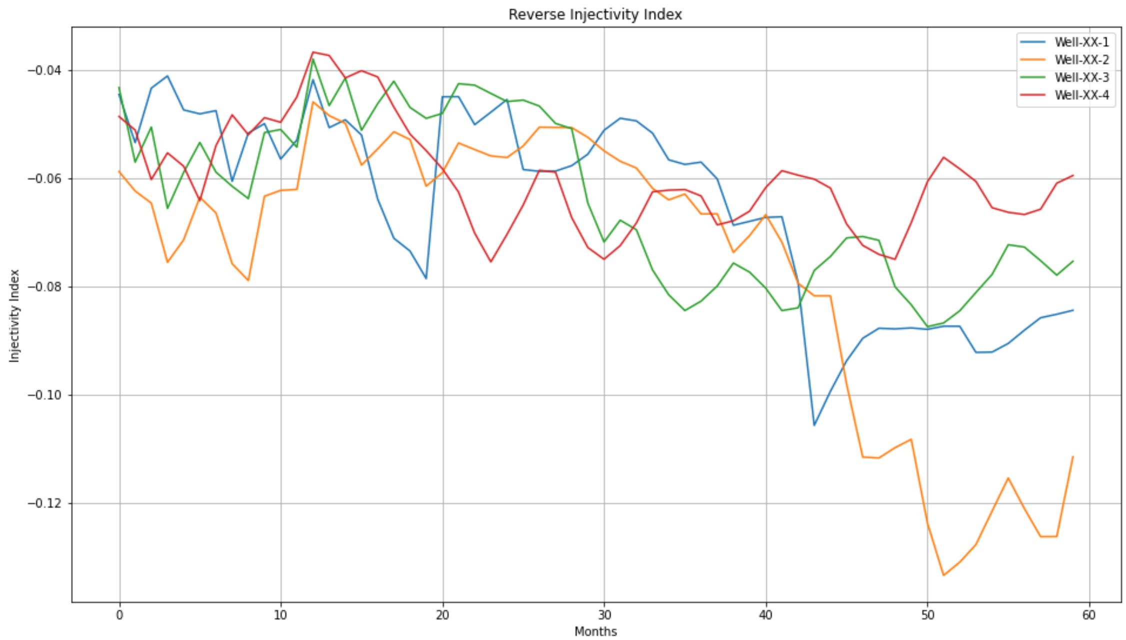
Required Data:

- Reservoir Pressure(psig)
- Pwi(Injection BHP) (psig)
- Injection Rate(stb)

What is Plotted:

- Reverse of injective index (= 1/Inj.Index)

Plot Example



Python Code

```
from matplotlib import pyplot as plt
import pandas as pd
plt.figure(figsize=(16,9))
import numpy as np
import math
#Import Data
data = pd.read_csv(r'inj.txt',sep='\t')
print(data.head())
#Read Reservoir Pressure
p_res = 1080 #psig
#Create New column
data['ii'] = 1/(data['Winj_bbl']/(data['InjP_psig'] - p_res))
#Get Unique Well Names
well_names = data['Well'].unique()
print(well_names)
for well in well_names:
    df = data[data['Well'] == well]
    plt.plot(range(0,len(df)), df['ii'],label=well)
plt.legend()
plt.grid(True)
plt.xlabel('Months')
plt.ylabel('Injectivity Index')
plt.title("Reverse Injectivity Index")
```

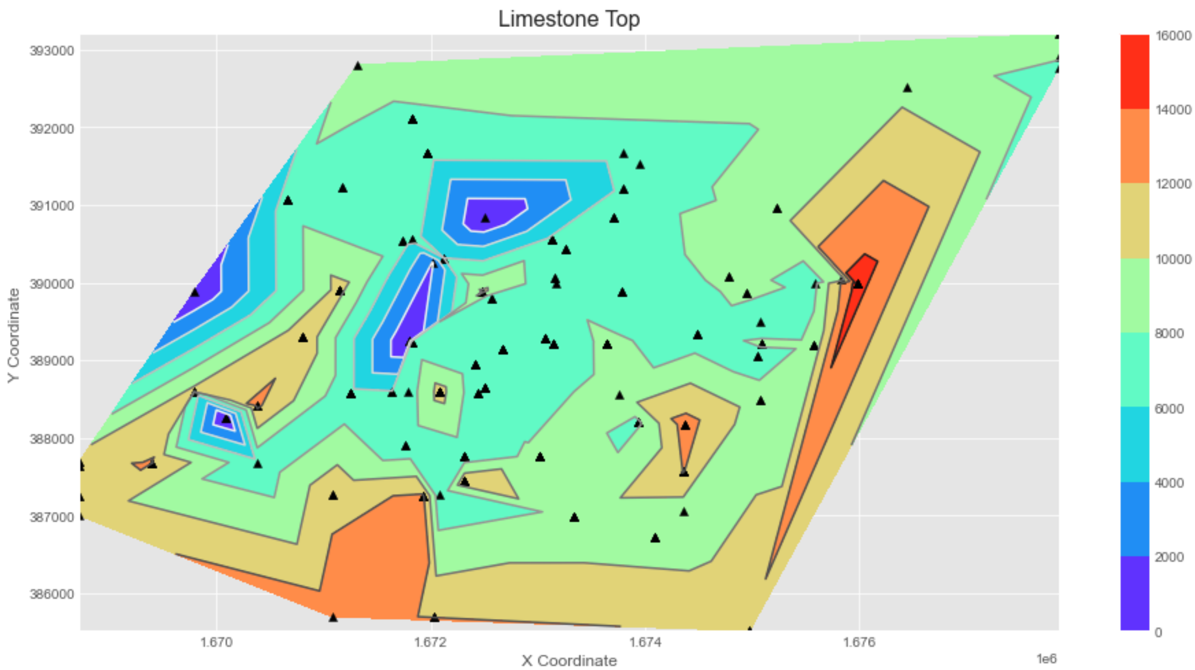
Example 3: Reservoir Mapping

Knowing how to create maps (generally speaking) is an essential skill for petroleum engineers. The process of creating maps was somewhat tedious, however python makes it easier and cheaper to produce maps of any desired variable, without using commercial reservoir management packages.

Required Data:

- X Coordinate.
- Y Coordinate.
- Any Other variables (Depth, Production, Sw, etc.)

Plot Example



Python Code

```
import matplotlib.pyplot as plt
plt.style.use('ggplot')
#Set figure size
fig = plt.figure(figsize=(16,8))
import pandas as pd
#import file
df = pd.read_csv('demoxy.txt', sep='\t')
#Get X, Y As List
x= df['XCoordinate']
y= df['YCoordinate']
Z = df['TotalDepth']
df = df.replace({'NaN': 0})
#Plot Triangular Color Filled Contour
plt.tricontourf(x,y,Z,cmap='rainbow')
plt.colorbar()
plt.tricontour(x,y,Z)
#Set Well Shapes
plt.scatter(x,y,color='black',marker='^')
plt.xlabel("X Coordinate")#Plot labels
plt.ylabel("Y Coordinate")#Plot labels
plt.title("Limestone Top",fontdict=({'size':16}))
```

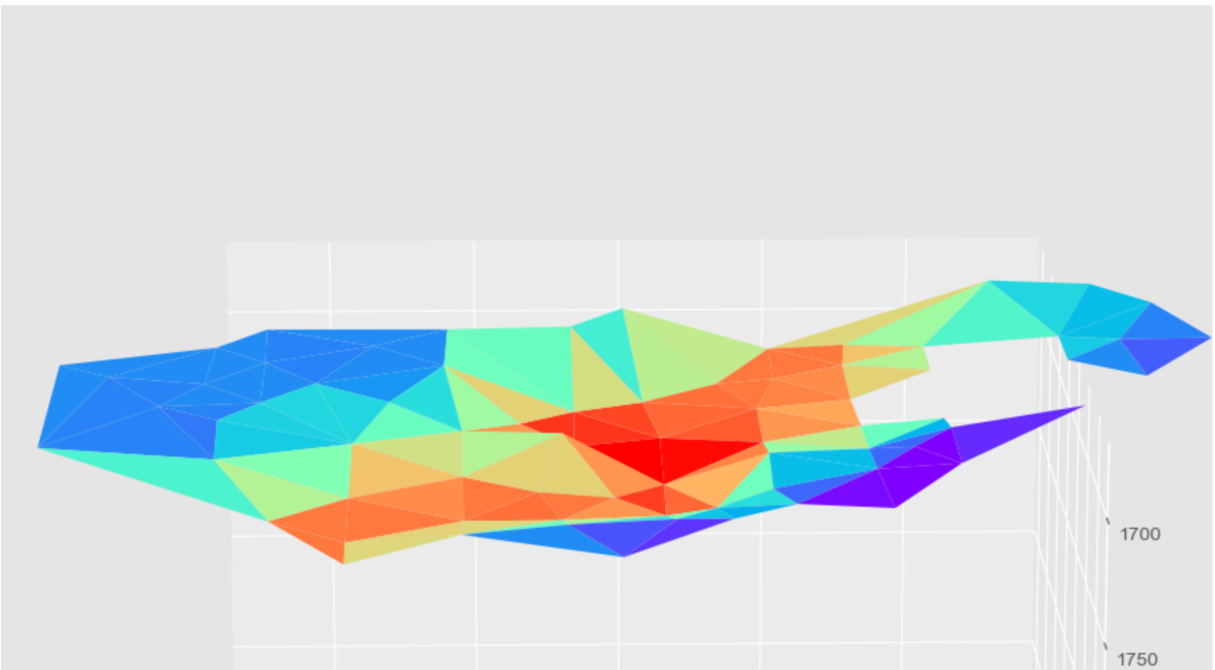
Example 4: Water Oil Contact 3D Map

This example illustrates how to plot 3D Oil water contact map, as values are populated from drilled wells. Knowing OWC is necessary to know if a well is penetrating the water leg thus derive conclusion for various reservoir management/workover activities.

Required Data:

- X Coordinate.
- Y Coordinate.
- OWC Depth

Plot Example



Python Code

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib qt #View Plot Outside (separate window)
plt.style.use('ggplot')
#Set figure size
fig = plt.figure(figsize=(16,8))
data = pd.read_csv('resdata.txt',sep='\t')
#prepair  an empty 3d figure
surf_plot = fig.add_subplot(projection='3d')
#Prep the data
x= data['Xcoord']
y= data['Ycoord']
z= data['OWC']
#plot 3d surface
surf_plot.plot_trisurf(x,y,z,cmap='rainbow',linewidth=0.01,label='Water Table')
#Set depth limits
surf_plot.set_zlim([1600,2000])
surf_plot.invert_zaxis()
```

Example 5: Primary Recovery Estimation

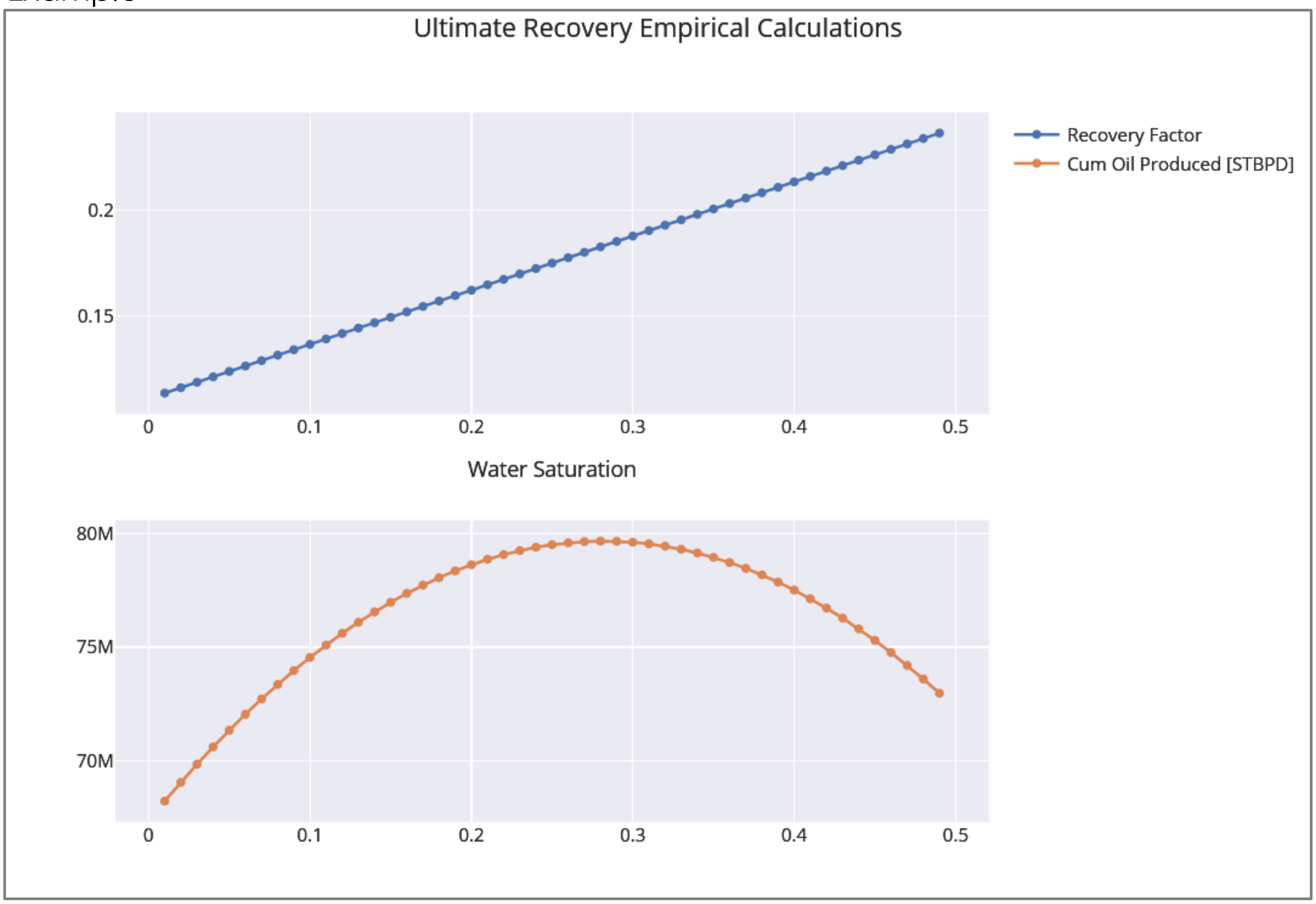
This example uses an empirical calculation proposed by Guthrie et. Al(ref API-1955 study), although no proved empirical methodology exist in with high accuracy, this method is to approximate the primary recovery factor

$$R_o = 0.114 + 0.272\log(k) + 0.256(S_w) - 0.136\log(\mu_o) - 1.538(\varnothing) - 0.00035(h)$$

Required Data:

- Water saturation.
- Oil Viscosity.(cp)
- Permeability(md)
- Payzone thickness
- Porosity

Plot Example



Python Code

```
from math import log10
import numpy as np
import plotly.express as ex
import plotly.graph_objects as go
from plotly.subplots  import make_subplots
import plotly
def calc_oil_prod_np(perm,sw,oil_visc,poro,h_pay,area,boi_fvf):
    """Using Guthrie et al 1955 - Empirical -Water Drive Reservoir"""
    recovery_fact = 0.2719*log10(perm) + 0.255*sw - 0.1355 * log10(oil_visc) - 1.538*poro
-  0.00035*h_pay + 0.11403
    ooip = 7758 * area * h_pay * poro* (1-sw)/boi_fvf
    return (recovery_fact * ooip),recovery_fact,ooip#return 3 parameters
# establish saturation range
sat_list = np.arange(0.01,0.5,0.01)
prod_list = []
rf_list = []
ooip_list = []
#Input data valus
grid = make_subplots(rows=2,cols=1)
poro = 0.18;perm = 30 ;oil_visc = 8; boi_fvf = 1.21 ;sw = 0.2; area = 35000;h_pay=15;
for w_sat in sat_list:
    prod = calc_oil_prod_np(perm, w_sat, oil_visc, poro, h_pay, area, boi_fvf)
    #add data to lists
    prod_list.append(prod[0])
    rf_list.append(prod[1])
    ooip_list.append(prod[2])

#add figures
grid.add_trace( go.Scatter(x=sat_list,y=rf_list,name='Recovery
Factor',mode='lines+markers')   ,row=1,col=1)
grid.add_trace( go.Scatter(x=sat_list,y=prod_list,name='Cum Oil Produced
[STBPD]',mode='lines+markers'),row=2,col=1)
#update plots
grid.update_layout(template='seaborn',title='Ultimate Recovery Empirical Calculations')
grid.update_layout(xaxis_title='Water Saturation')
#show plot
plotly.offline.plot(grid)
```

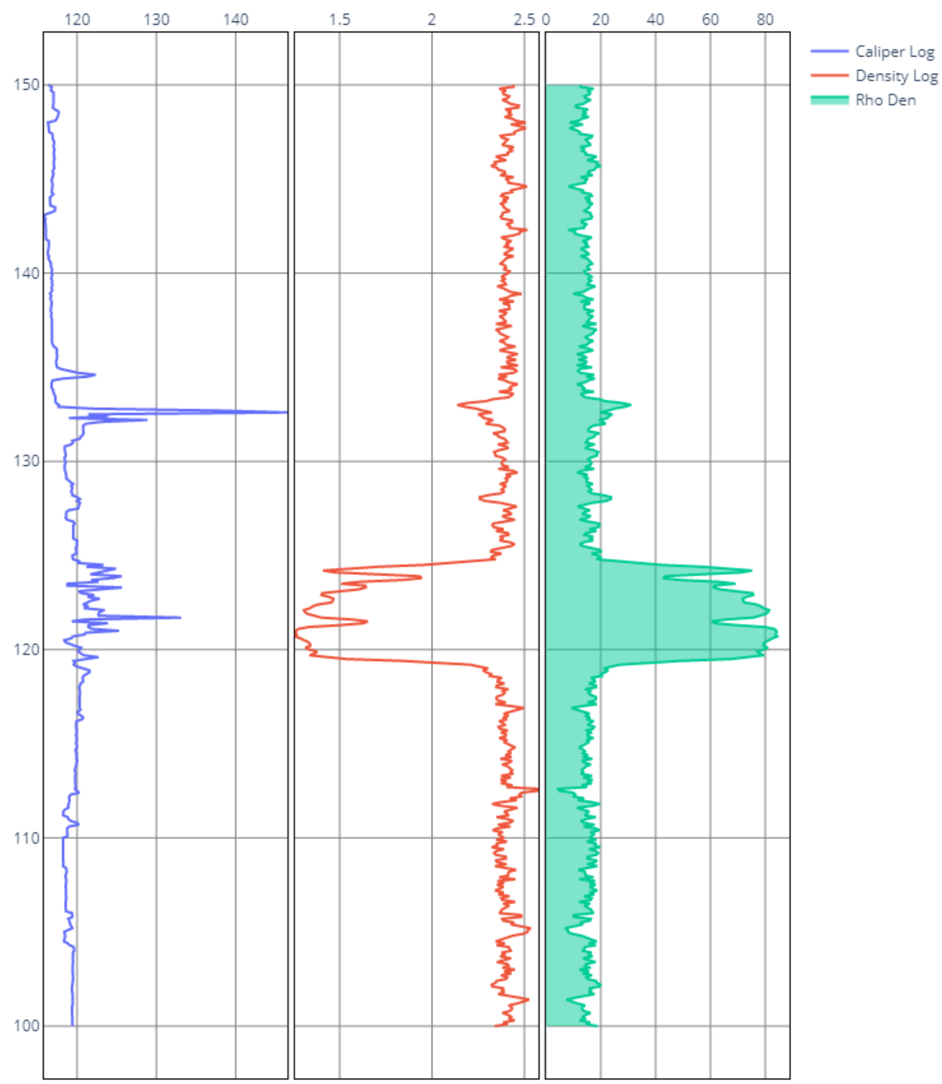
Example 6: Porosity From density Log

In this exercise, we will try to calculate porosity by utilizing density log(using las file), basic assumptions are made about the matrix density and fluid property. This example is intended to illustrate how to construct well log tracks using python and a set of 3rd party libraries.

Required Data:

- Las file
- Matrix Density
- In-pore fluid density

Plot Example



Python Code

```python
import lasio
logfile = lasio.read('caliper.las')
import plotly.graph_objects as go
from plotly.subplots import make_subplots
#Calculate the porosity for limestone
# limestone matrix density assumed = 2.65
# fluid density =1 g/cc assuming Water
#Make Subplot Grid
fig = make_subplots(rows=1, cols=3,shared_yaxes=True,subplot_titles=[' ',' ', ' '],horizontal_spacing=0.01)
#Set Properties
matrix_density = 2.65
fluid_density = 1
#Calculate Density-Porostity
logfile['DenRho'] = 100*(matrix_density - logfile ['DENS'])/(matrix_density - fluid_density)
#Create traces for each curve/series
density = go.Line(x=logfile['DENS'],y=logfile['DEPT'],name='Density Log')
caliper= go.Line(x=logfile['CAL'],y=logfile['DEPT'],name='Caliper Log')
porosity= go.Line(x=logfile['DenRho'],y=logfile['DEPT'],name='Rho Den',fill='tozerox')
#Add plots to subplot grid
fig.add_trace(caliper,row=1,col=1)
fig.add_trace(density,row=1,col=2)
fig.add_trace(porosity,row=1,col=3)
fig.update_layout(width=800,height=1000,template='plotly_white')
#update Axes
fig.update_xaxes(side='top',showline=True,linewidth=1,linecolor='black',mirror=True,gridcolor='grey')
fig.update_yaxes(showline=True,linewidth=1,linecolor='black',mirror=True,gridcolor='grey')
```

Chapter 2: Production Engineering -PE

Example 1: Liquid Loading Calculation

Liquid loading calculation is used for gas wells (High GOR wells) were it's desired to know the liquid loading efficiency of a given well, if Gas Velocity > Critical velocity then the well is unloading efficiently and vise versa.

Required Data:

- Wellhead Pressure(psig)
- Gas Flowrate (psig)
- Tubing ID = 2.99 inches

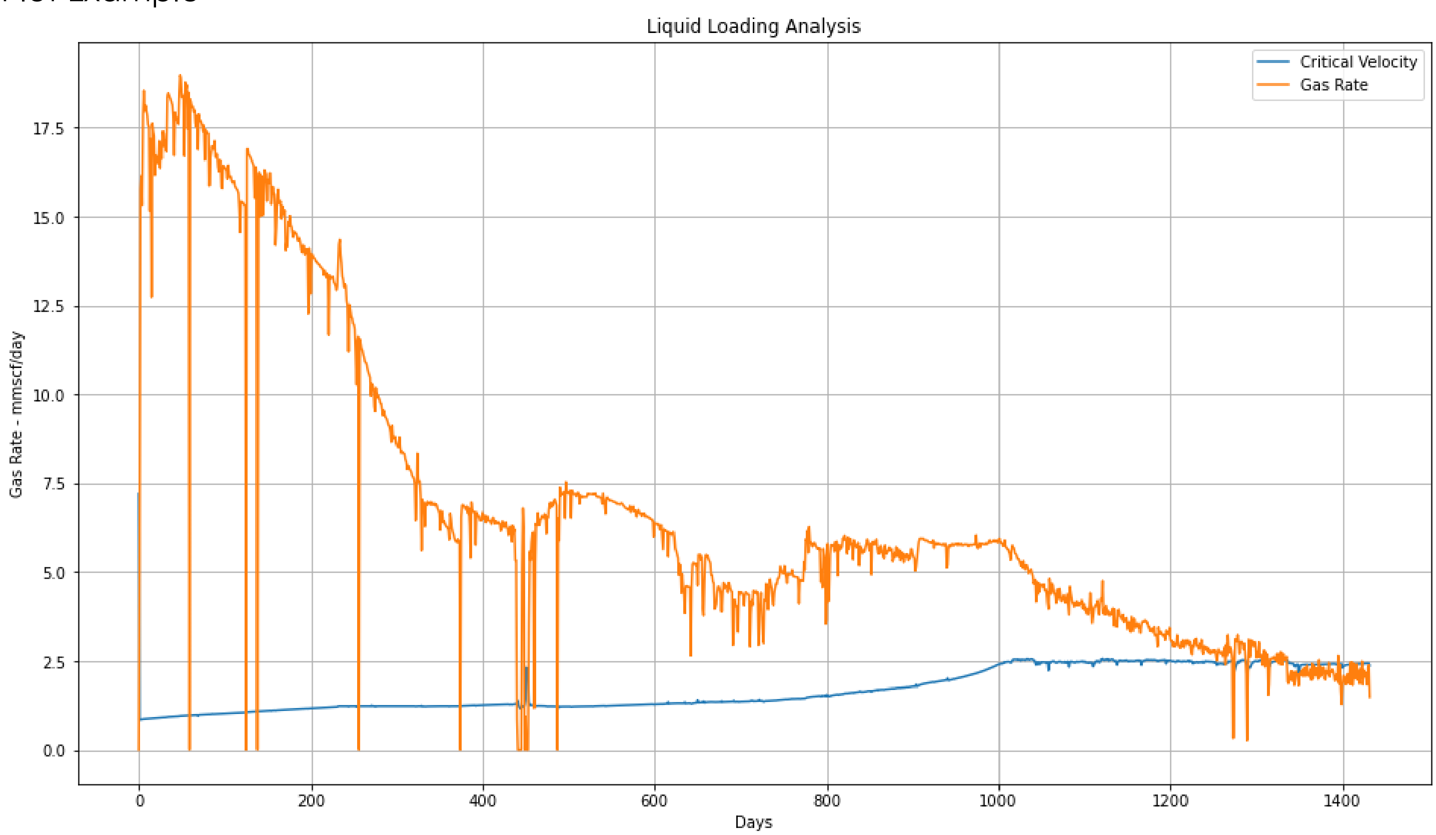
What is Plotted:

- Critical Velocity. (From turner equation)
- Actual Gas Velocity (from Qg)
- Unloading Efficiency (Actual Velocity / Critical)

$$v_{gwater} = \frac{5.62(67 - 0.0031p)^{1/4}}{(0.0031p)^{1/2}} \, ft/\sec$$

P is WHP in psig

Plot Example



Python Code

```
from matplotlib import pyplot as plt
import pandas as pd
plt.figure(figsize=(16,9))
import math
#load data
data = pd.read_csv('gasflow.txt',sep='\t')
#Calculate Tubing Area
tubing_id = 2.99 #internal diameter
tbg_area = math.pow(tubing_id,2)*3.14 / 144 #144 is the convesion from in to ft
#calculate turner velocity
data['Turner'] = 5.62*(67 - 0.0031*data['Whp_psig'])**0.25 /
((0.0031*data['Whp_psig']**0.5))
data['Turner'] = tbg_area * data['Turner'] * 86400/1000000 # Convert Velocity to
Qg_critical
#Plot data
plt.plot(data['Days'],data['Turner'], label="Critical Velocity")
plt.plot(data['Days'],data['Qgas_mmscf'], label="Gas Rate")
plt.legend()
plt.grid(True)
plt.xlabel('Days')
plt.ylabel('Gas Rate - mmscf/day')
plt.title("Liquid Loading Analysis")
#Add Limit to Xaxis(optional)
#plt.xlim(1000,1500)
```

Example 2: Production Back Allocation

Back allocation is an industry wide practice used to devide production from a given separator or surface processing facility to it's connected wells, this will provide daily production by multiplying well test values to a calculation allocation factor(something similar to tuning factor).

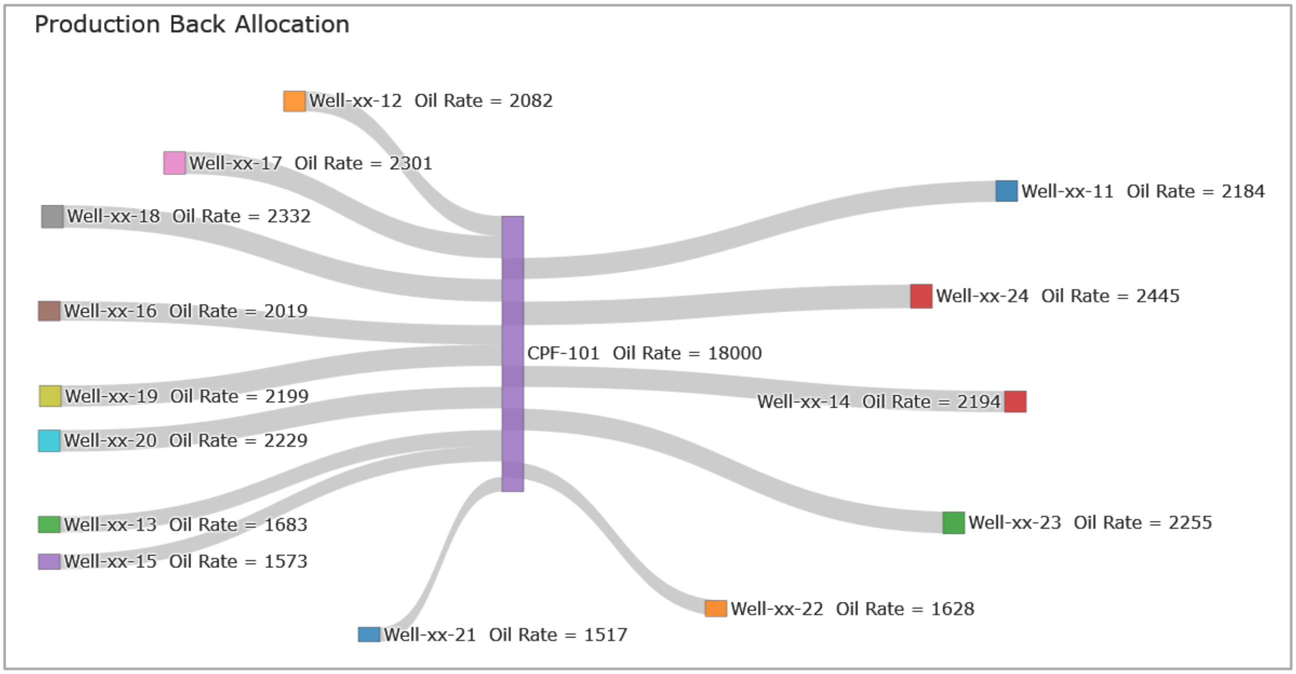
Required Data:

- Separator Production (Qo,Qg,Qw)
- Well Test for each well
- Allocation Factor (Calculated)

Allocation Factor can be calculated using :

$$Allocation\ Factor = \frac{Actual\ Production}{Theoretical\ Production} = \frac{Separator\ Rate(s)}{\sum Well\ Test\ Rates}$$

Plot Example



Libraries used:

- Plotly.

Figure Types:

- Sankey

Python Code

```
#If you don't have plotly install it : PIP install Plotly
import pandas as pd
import plotly.graph_objects as go
import plotly
df = pd.read_csv('testrates.txt', sep='\t')
#Calculate Total Oil and Water Rate
qo_total = df['OilRate'].sum()
qw_total = df['WaterRate'].sum()
oil_allocation_factor = 26500/qo_total
water_allocation_factor = 13000/qw_total
#Correct Allocated Rates
df['OilCorrected'] = df['OilRate'] * oil_allocation_factor
df['WaterCorrected'] = df['WaterRate'] * water_allocation_factor
#Populate Facility data
facitlity_data = {'Asset':'CPF-
101','OilRate':26500,'WaterRate':13000,'OilCorrected':26500,'WaterCorrected':13000}
#Insert Facility Data
df = df.append(facitlity_data,ignore_index=True);
nodes = list(df.index)
target = [14] * 14
values = list(df['OilRate'])
#Create Labels
labels = []
for i in df.index:
    labels.append(df['Asset'][i]+" "+" Oil Rate (STBPD) = "+
str(int(df["OilCorrected"][i])))
#Create Plotly Figure
fig = go.Figure(data=[go.Sankey(
    node = dict(
      label = labels
    ),
    link = dict(
      source = nodes,
      target = target,
      value = values
  ))])
#Setup Plot and Show figure
fig.update_layout(title_text="Production Back Allocation",
font_size=16,template='simple_white')
plotly.offline.plot(fig)
```

Example 3: Fetkovitch IPR

This example illustrates the usage of IPR where multiple well test points are required.

Required Data:

- Well Test Pwf, Q
- Reservoir Pressure

Fetkovitch's Equation:

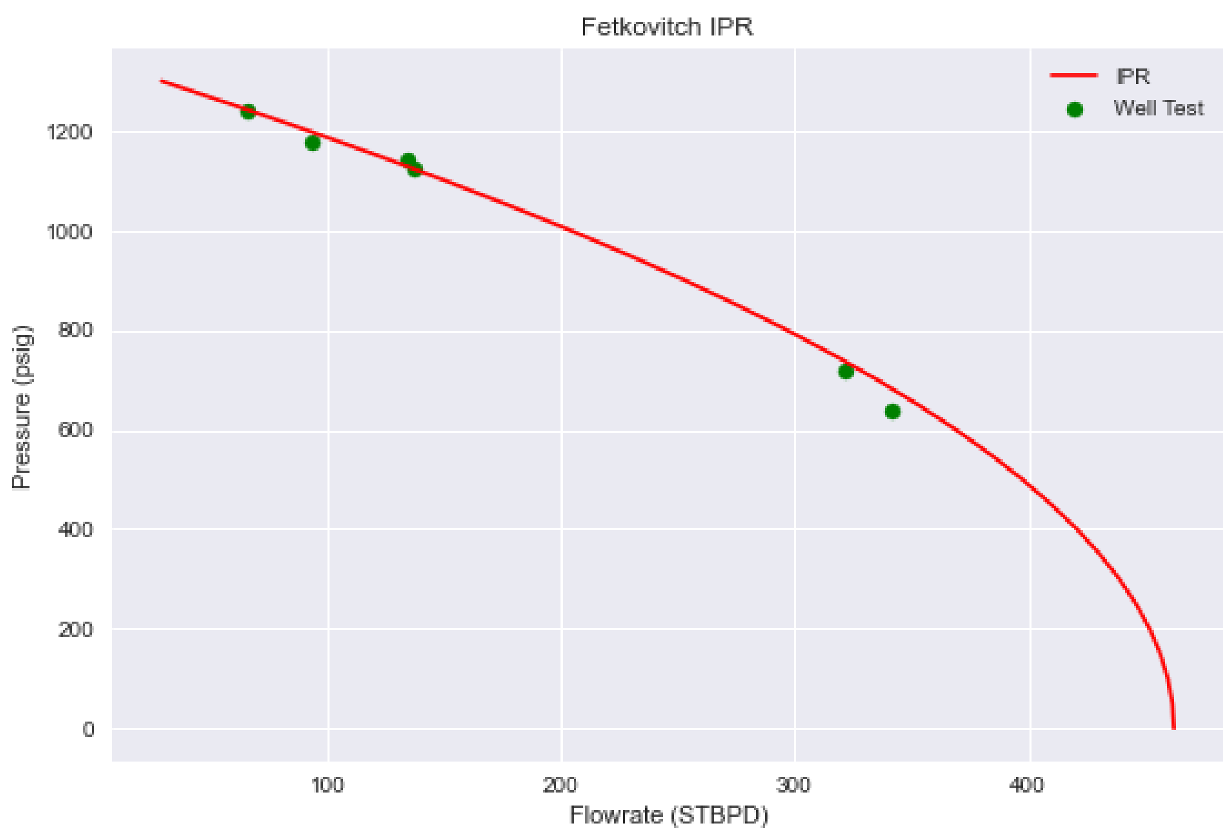
$$Q_o = C * (Pr^2 - Pwf^2)^n$$

where n = 1/slope when fitted in log-log plot
where C can be calculated once n Is known.

Libraries Used :

- Matplotlib

Plot Example



Python Code:

```
import matplotlib.pyplot as plt1
import matplotlib.pyplot as plt
import pandas as pd
import math
import numpy as np
#populate well test data
q_list =   [66,134,137,93,321,341]
pwf_list = [1242,1142,1123,1178,719,638]
p_res = 1345 #psig
df_init = {"q_stb":q_list,"pwf_psig":pwf_list}
df_well_test = pd.DataFrame(df_init)
df_well_test['delta^2'] = p_res**2 - df_well_test['pwf_psig']**2
df_well_test['delta_log'] =np.log10(df_well_test['delta^2'])
df_well_test['q_log'] = np.log10(df_well_test['q_stb'])
#plt.scatter(x=df_well_test['q_log'],y=df_well_test['delta_log'])
poly_func = np.poly1d(np.polyfit(df_well_test['q_log'], df_well_test['delta_log'],
deg=1))
pwf_pred = poly_func(df_well_test['q_log'])
#plt.plot(df_well_test['q_log'],list(pwf_pred))
#plt.xscale('linear'),plt.yscale('linear')
n = 1/poly_func.coefficients[0]
c = q_list[0]/(p_res**2 - pwf_list[0]**2)**n
#Construct IPR Data
pwf_range = range(0,p_res,50)
q_range = []

for pwf in pwf_range:
    q_range.append(c*(p_res**2 - pwf**2)**n)
plt.style.use('seaborn')
plt.plot(q_range,pwf_range,c='red')
plt.scatter(q_list,pwf_list,c='green')
plt.xlabel('Flowrate');plt.ylabel('Pressure')
plt.title('Fetkovitch IPR')
```

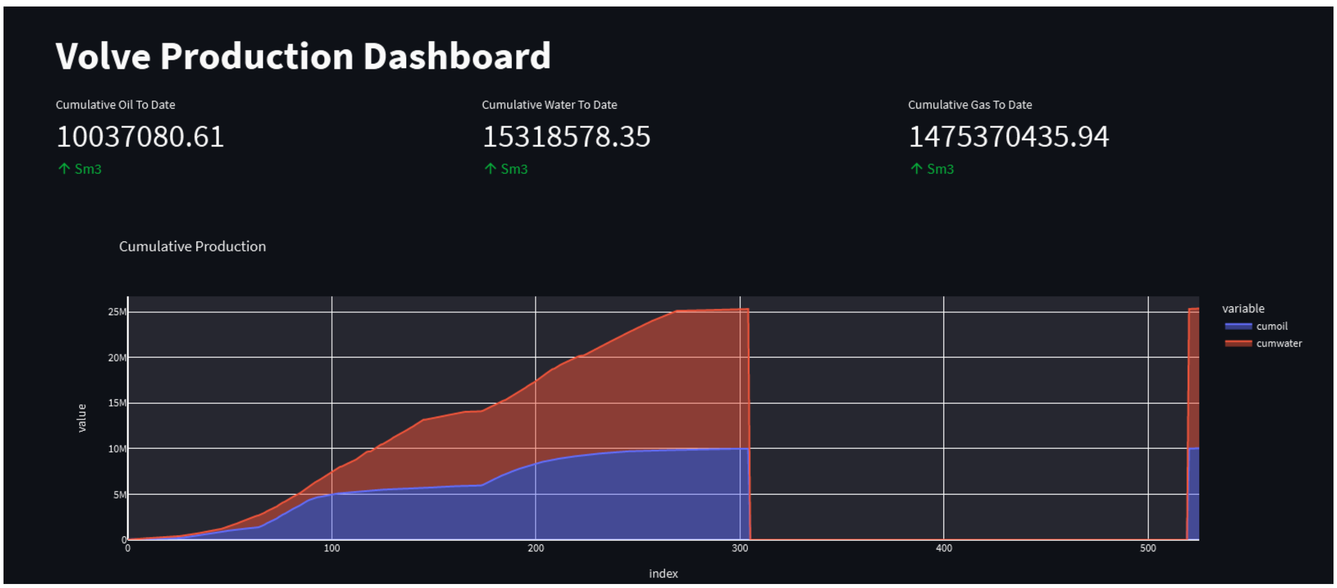
Example 4: Volve Production Dashboard Using Streamlit

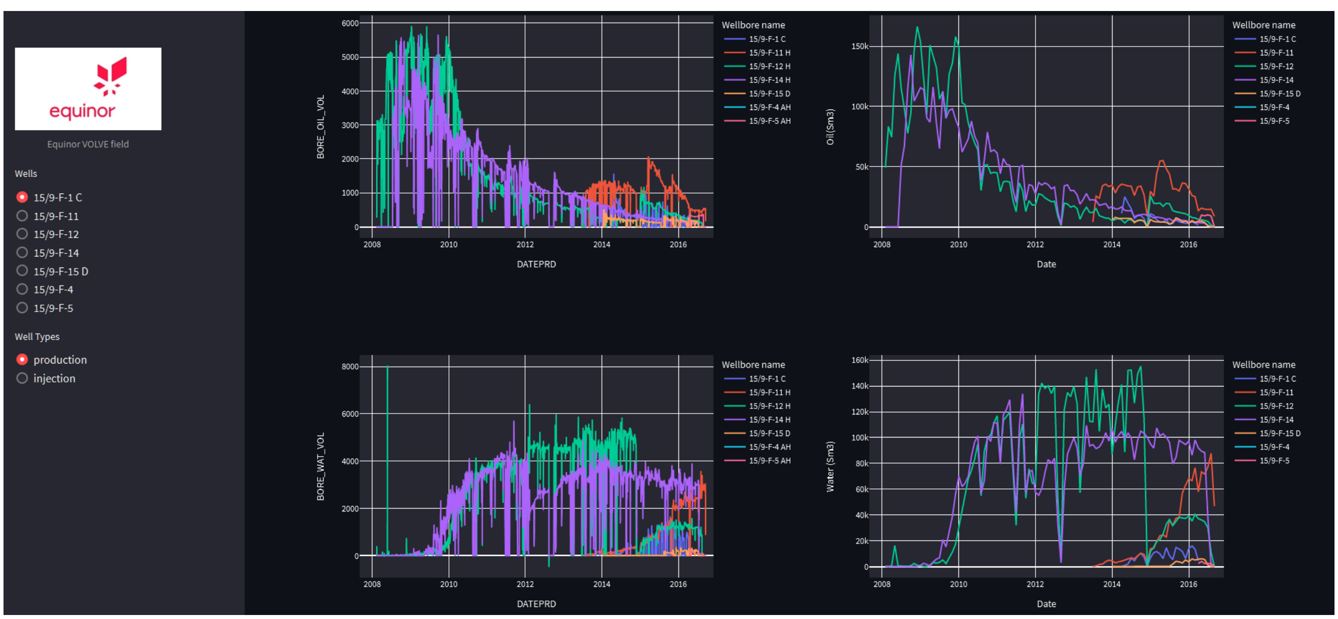
Production dashboards are very effective in monitoring production over a given period of time (Monthly, Daily , etc).Dashboards give insight and understanding to the operators of what their field is performing. Python provides strong dashboarding capabilities. In this example we use streamlit to create dashboard for VOLVE field data.

Libraries Used:

- Plotly Express.
- Streamlit
- Pandas

Plot Examples





Python Code

```
import streamlit as st
import pandas as pd
import plotly.express as ex
from PIL import Image
#Import Excel Data
volve_daily = pd.read_excel('volve.xlsx')
volve_montly = pd.read_excel('volve.xlsx',sheet_name='Monthly Production Data')
# App Details
st.title('Volve Production Dashboard') # set title
volve_montly['cumoil'] = volve_montly['Oil(Sm3)'].cumsum()
volve_montly['cumgas'] = volve_montly['Gas (Sm3)'].cumsum()
volve_montly['cumwater'] = volve_montly['Water (Sm3)'].cumsum()
img = Image.open('logo.jpg')
img = img.resize([int(img.width/5),int(img.height/5)])#resize image 5 times smaller
# Side bar
st.sidebar.image(img,caption='Equinor VOLVE field',)#set image on top of sidebar
st.sidebar.radio('Wells', volve_montly['Wellbore name'].unique())
st.sidebar.radio('Well Types', volve_daily['FLOW_KIND'].unique())
#Add some data metrics to the chart
total_oil = volve_montly['Oil(Sm3)'].sum()
total_gas = volve_montly['Gas (Sm3)'].sum()
total_water = volve_montly['Water (Sm3)'].sum()
c1,c2,c3 = st.columns(3)
c1.metric(label='Cumulative Oil To Date',value = total_oil,delta=" Sm3")
c2.metric(label='Cumulative Water To Date',value = total_water,delta=" Sm3")
c3.metric(label='Cumulative Gas To Date',value = total_gas,delta=" Sm3")
#Add Injection Metrics
inj1, inj2 = st.columns(2)
total_gas_inj = volve_montly['GI (Sm3)'].sum()
total_water_inj = volve_montly['WI (Sm3)'].sum()
#inj1.metric(label = "Total Gas
Injected",value=total_gas_inj,delta="Sm3",delta_color="inverse")
#inj2.metric(label= "Total Water Injected",value =total_water_inj,delta="Sm3")
st.plotly_chart(ex.area(volve_montly,y=['cumoil','cumwater'],title='Cumulative
Production'),use_container_width=True)
col1, col2 = st.columns(2)
# Set header for the two columns
col1.subheader('Daily Production')
col2.subheader('Montly Production')
# Add chart to first column
daily_top_chart = ex.line(volve_daily,x='DATEPRD', y='BORE_OIL_VOL',color='Wellbore
name')
daily_bottom_chart = ex.line(volve_daily,x='DATEPRD', y='BORE_WAT_VOL',color='Wellbore
name')
col1.write(daily_top_chart);col1.write(daily_bottom_chart)
# Add chart to second column
month_top_chart = ex.line(volve_montly,x='Date', y='Oil(Sm3)',color='Wellbore name')
month_bottom_chart = ex.line(volve_montly,x='Date', y='Water (Sm3)',color='Wellbore
name')
col2.write(month_top_chart);col2.write(month_bottom_chart)
```

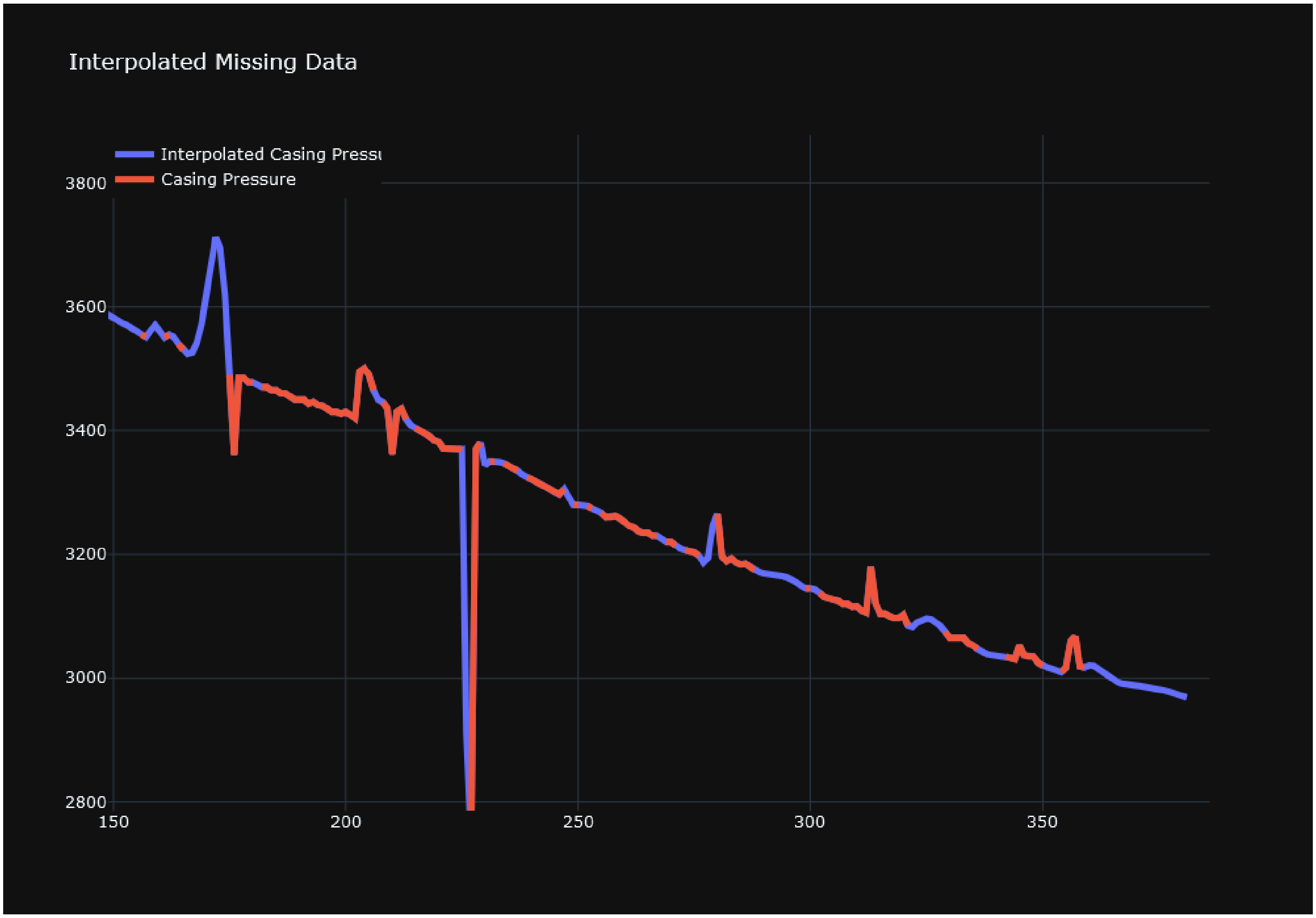
Example 5: Missing Casing Head Pressure Data

A lot of missing data present in oil and gas industry data warehouses, and the need to be fixed before even making use of. This example deals with casing head pressure where in contains tens of missing values.

Libraries Used:

- Plotly.
- Pandas

Plot Examples



Python Code:

```
# Import Libraries
import pandas as pd
import plotly.graph_objects as go
import plotly
# import file as DataFrame
prod_df = pd.read_csv('5-missing data.txt', sep='\t')
prod_df.index = prod_df['Time (Days)']
#Create New Column for Interpolation
prod_df['CSG-P-Inter'] = prod_df['Casing Pressure ']
#Interpolate over the data using 2nd order polynomial
prod_df['CSG-P-Inter'].interpolate(method='polynomial', order=2,inplace=True)
#Create a plotly figure
fig = go.Figure()
# Add Curves to plotly graph
fig.add_trace(go.Scatter(x=prod_df['Time (Days)'],y=prod_df['CSG-P-Inter'],
name='Interpolated Casing Pressure',line=dict(width=5)))
fig.add_trace(go.Scatter(x=prod_df['Time (Days)'],y=prod_df['Casing Pressure '],
name='Casing Pressure',line=dict(width=5)))
# Setup the plot
fig.update_layout(title='Interpolated Missing Data',template='plotly_dark')
# Setup legend
fig.update_layout(legend=dict(x=0,y=1))
# show the plot
fig.show()
```

Chapter 3: Drilling and Workover

Example 1: Openhole Volume Calculation

Openhole section during drilling are made using drill bit, however the hole size is not exactly the same diameter as bit diameter due to many reasons. It's usually beneficial to know the exact volume of openhole volume to account for different calculations like cement volume required behind casing.

$$Vol_{openhole} = 3.14\ x\ (\frac{BitSize}{2})^2\ x\ Height$$

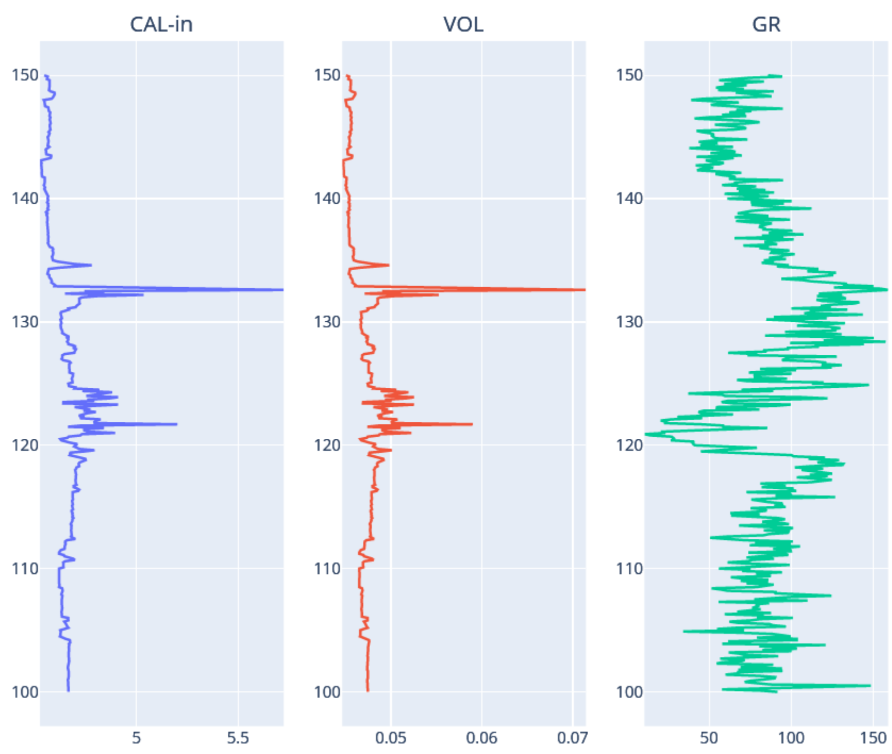
Required Data:

- Well log (las file)
- Bit size
- Depth
- Caliper readings

Libraries used:

- Plotly.
- Lasio.

Plot Example



Python Code:

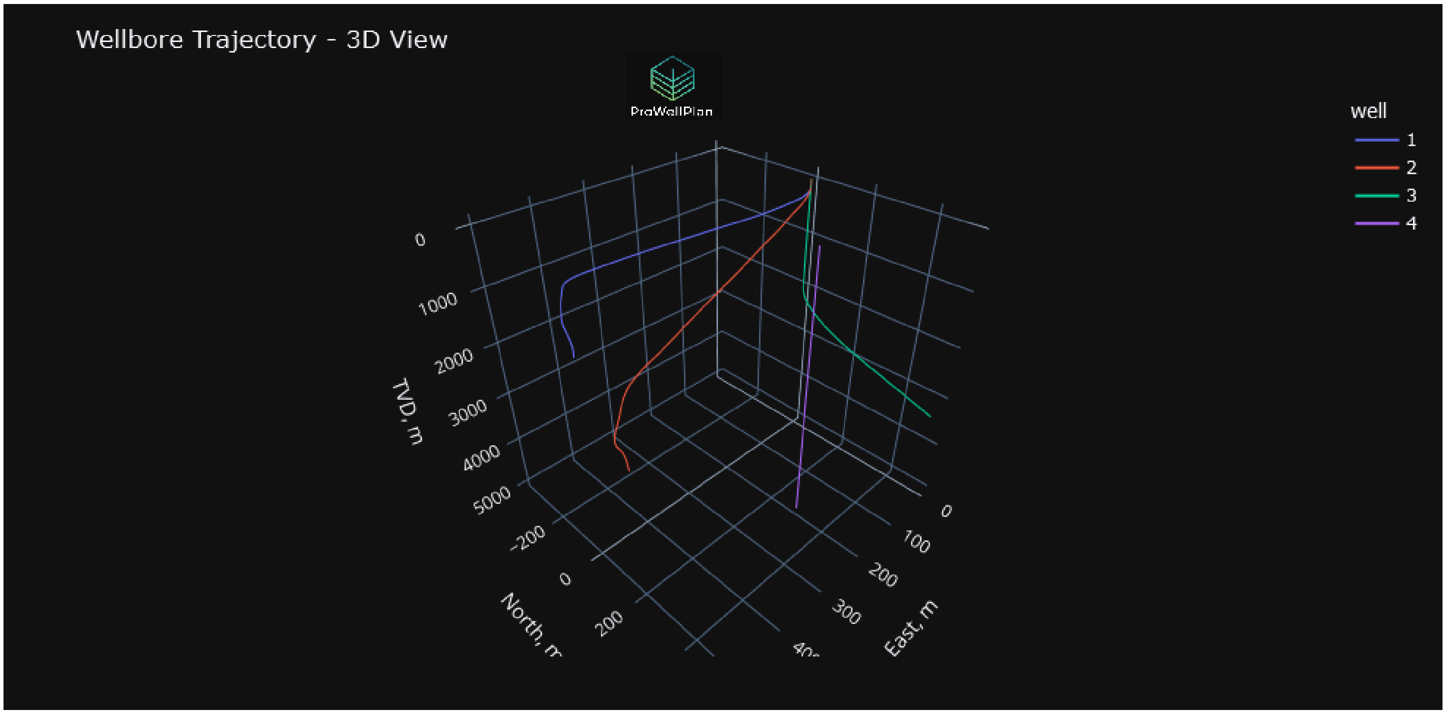
```
# Import Libraries
import lasio
import plotly.graph_objects as go
import plotly
from plotly.subplots import make_subplots
las = lasio.read('caliper.las')     # Import Sample Las File
#Creat new curve for caliper in inches
las['CAL-in'] = las['CAL']/(25.6)
#Create Volume curve
interval_step = las.well['step'].value
las['VOL'] = 3.14*las['CAL-in'] * las['CAL-in'] * interval_step /144
tracks = ['CAL-in','VOL','GR']  # Track we wish to plot
subplots = make_subplots(rows=1,cols=6,subplot_titles=tracks)
#Iterate through tracks
for t in tracks:

subplots.add_trace(go.Line(y=las.depth_m,x=las[t],name=t),row=1,col=tracks.index(t)+1)
#Calculate openhole section volume
volume_log = las['VOL'].sum()
#calculate theoretical section volume
depth_start = las.well['STRT'].value
depth_end =las.well['stop'].value
interval =  depth_end - depth_start #net interval(in ft )
bit_size = las.well['bit'].value
volume_bit = (3.14 * (bit_size/2)**2 / 144 ) * interval
print("Volume from log = {0:.2f} , Volume From Bitsize
{1:.2f}".format(volume_log,volume_bit))
plotly.offline.plot(subplots)
```

Example 2: 3D Well Survey Visualization

Commercial grade software that are provided by companies like (Schlumberger, Haliburton, etc) have the capability to plot and calculate trajectories for already existing wells or wells that are being planned. Python offers a simple yet power full library that can do what is mentioned above. The input files must be in xlsx file type and in the following column format :

md	inclination	azimuth	azimuth	tvd

Plot Example



Generated Well Profile Example:

Out[49]:

	md	inc	azi	north	east	tvd	dl	sectionType	dls
0	1	0.00	0	0.000000	0.0	1.000000	0.00	vertical	0.0
1	101	0.00	0	0.000000	0.0	101.000000	0.00	vertical	0.0
2	201	0.00	0	0.000000	0.0	201.000000	0.00	vertical	0.0
3	301	0.00	0	0.000000	0.0	301.000000	0.00	vertical	0.0
4	401	0.00	0	0.000000	0.0	401.000000	0.00	vertical	0.0
5	501	0.00	0	0.000000	0.0	501.000000	0.00	vertical	0.0
6	601	0.00	0	0.000000	0.0	601.000000	0.00	vertical	0.0
7	701	0.00	0	0.000000	0.0	701.000000	0.00	vertical	0.0
8	801	0.00	0	0.000000	0.0	801.000000	0.00	vertical	0.0
9	901	0.00	0	0.000000	0.0	901.000000	0.00	vertical	0.0

Python Code:

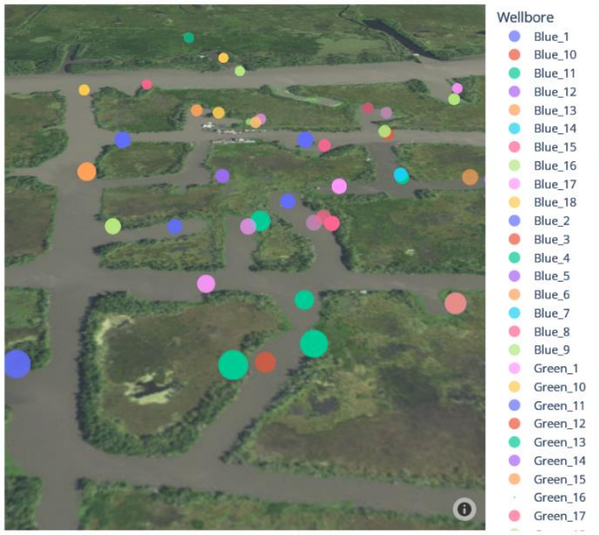
Note This Example only runs in Jupyter Notebooks

```
import pandas as pd
# import profile
import well_profile as wp
#import 2 welss from excel sheet
well1 = wp.load('well2.xlsx')
well2= wp.load('well1.xlsx')
#create 3rd well by entering data
well3 = wp.get(3500,profile='J', kop=2000, eob=3000 , build_angle=30)
#Create 4th Vertical well
well4 = wp.get(5000,profile='V',set_start={'north':300,'east':200})
#plot all wells
well2.plot(add_well=[well1,well3,well4],style={'darkMode':True,'size':3}).show()
#get survey calculations for well3
survey_list = []
for i in range(1,3500,100):
    survey_list.append(well3.get_point(i))#add single depth property to the list
#Create Pandas dataframe
df =pd.DataFrame(survey_list)
```

Example 3: GIS Mapping of Well Locations

This example shows how to map multiple wells location on GIS map using plotly graphs.

Example Plot:



Python Code:

```
import pandas as pd
import plotly.express as ex
import plotly
#read data
xy_df = pd.read_csv('demoxy.txt',sep='\t')
fig =
ex.scatter_mapbox(xy_df,lat='Latitude',lon='Longitude',hover_name='WellboreID',zoom=13,
                         text='WellboreID',color='Wellbore',size='TotalDepth')
#Copy paste this part
fig.update_layout(
    mapbox_style="white-bg",
    mapbox_layers=[
        {
            "below": 'traces',
            "sourcetype": "raster",
            "sourceattribution": "United States Geological Survey",
            "source": [

"https://basemap.nationalmap.gov/arcgis/rest/services/USGSImageryOnly/MapServer/tile/{z}/
{y}/{x}"
            ]
        },
        {
            "sourcetype": "raster",
            "sourceattribution": "Government of Canada",
            "source": ["https://geo.weather.gc.ca/geomet/?"
                       "SERVICE=WMS&VERSION=1.3.0&REQUEST=GetMap&BBOX={bbox-epsg-
3857}&CRS=EPSG:3857"

"&WIDTH=1000&HEIGHT=1000&LAYERS=RADAR_1KM_RDBR&TILED=true&FORMAT=image/png"],
        }
      ])
plotly.offline.plot(fig)
```

Chapter 4: Operational

Example 1: NORSOK M-506 Sweet Corrosion Monitoring

NORSOK Corrosion model is an industry best practice for calculation CO2 induced corrosion in Water/Gas Systems. The model relies multiple inputs. NORSOK M-506 2005 is used for the basis of calculation. Main equation is shown below. However above standard should be consulted for the companion equations

$$CR_t = K_t \text{ x } fCO_2^{0,62} \text{ x } (S/19)^{0,146 + 0,0324 \log(fCO2)} \text{ x } f(pH)_t \quad \text{(mm/year)}$$

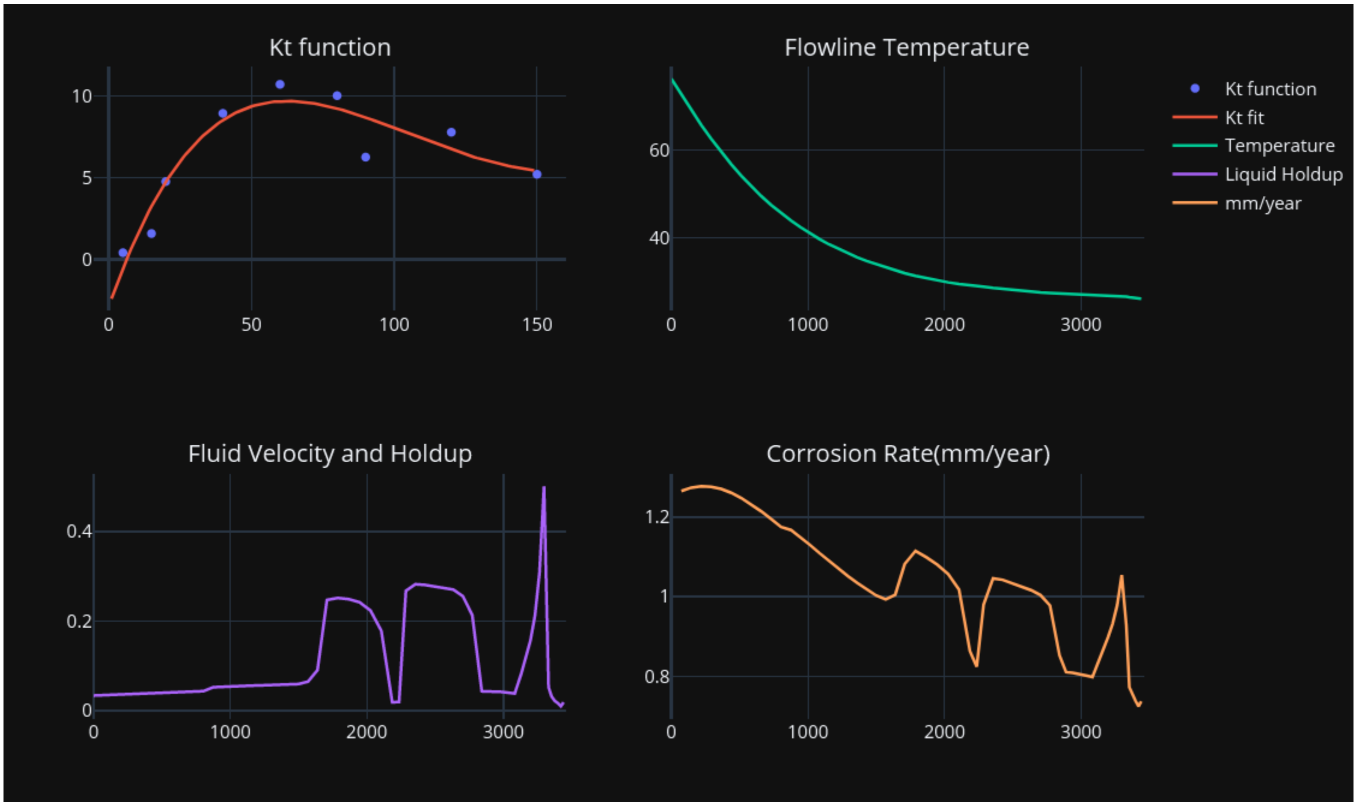
where:

K_t	- Constant for the temperature t
fCO_2	- the fugacity of CO_2 (bar)
S	- Wall shear stress (Pa)
$f(pH)_t$	- The pH factor at temperature t

Required Data:

- Pressure.
- Temperature.
- Mixture Velocity.
- Mixture Density.
- Aqueous System pH.
- CO2 Properties

Plot Example



Python Code

```
#This python code written based on NORSOK M-506 Standard
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots
df = pd.read_csv('1.flowdata.txt',sep='\t')
pipe_diameter = 16.36 #inches , 0.41 meters
water_visc = 0.001# =1cp = 0.001 N s/m2
water_ph = 5.5
co2_percent = 2 # %
f_pH = 0.25 # pH function
#corrosion Cooficient fitting
temp = [5,15,20,40,60,80,90,120,150]
kt = [0.42,1.59,4.76,8.92,10.69,10,6.25,7.77,5.203]
kt_ft = np.poly1d(np.polyfit(temp,kt,deg=4))
temperature_prediction = list(range(1,150,1))
#Step 1 Calculate Renolds Number
df['Re'] = (df['VELOCITY(M/S)'] * df['Density [kg/m3]
']*(pipe_diameter*0.0254))/water_visc
#Step 2 : Calculate Friction factor (blasius)
df['ff'] = 0.3164*df['Re']**(-0.25)
#Step 3 : Calculate Wall Shear Stress
df['ss'] = 0.5 * df['Density [kg/m3] '] * (df['VELOCITY(M/S)']**2) * df['ff']
#Step 4: Calculate Co2 properties
df['co2_pp'] = co2_percent*0.01 * df['PT[PSIG]']/14.5 # pressure in bars
df['co2_a_fact'] = 10**((df['PT[PSIG]']/14.5)*(0.0031 - (1.4/(df['TM[c]']+273))))
df['co2_fug'] = df['co2_pp'] * df['co2_a_fact']
#Step 5 :Calculate corrosion rate mm/year
df['corr_rate'] = kt_ft(df['TM[c]']) * ((df['co2_fug'] ** 0.62)) * ((df['ss']/19) **
(0.146 + 0.0324 * np.log10( df['co2_fug']))  )* f_pH
#Create Array of Visualization Plots
fig = make_subplots(rows=2,cols=2,
                    subplot_titles=(['Kt function','Flowline Temperature','Fluid Velocity
and Holdup','Corrosion Rate(mm/year)']))
#Create Kt factor fitting plot
fig.add_trace(go.Scatter(x=temp,y=kt,name='Kt function',mode='markers'),row=1,col=1)
fig.add_trace(go.Scatter(x=temperature_prediction,y=kt_ft(temperature_prediction),name='K
t fit'),row=1,col=1)
#Create pressure and temperature plots
fig.add_trace(go.Line(x=df['length [m]'],y=df['TM[c]'],name='Temperature'),row=1,col=2)
fig.add_trace(go.Line(x=df['length [m]'],y=df['LIQUIDHOL'],name='Liquid
Holdup'),row=2,col=1)
fig.add_trace(go.Line(x=df['length [m]'],y=df['corr_rate'],name='mm/year'),row=2,col=2)
#Change figure template
fig.update_layout(template='plotly_dark')
plotly.offline.plot(fig)
```

Example 2: Flow Stability Advisor

This example calculates/figures out flow regime type in a set of given well data using Taitel and Duckler flow regime map.

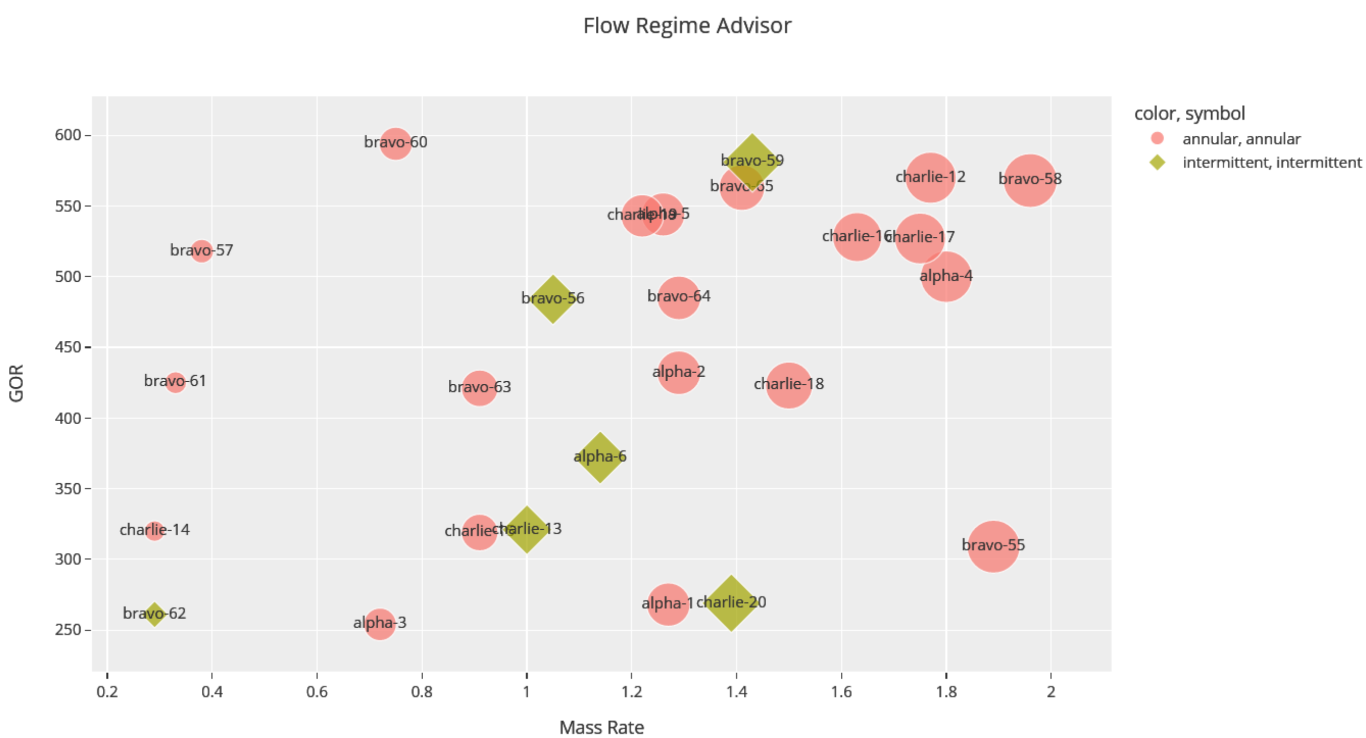
Required Data:

- Liquid Holdup.
- Mass Rate.
- Temperature.
- Mixture Density.
- etc

Libraries used:

- Fluids.
- Plotly.

Plot Example



Python Code;

```
#Well Stability advisor
import pandas as pd
import plotly.express as ex
import plotly
import math
from math import log10
import fluids.two_phase as fd
from plotly.subplots import make_subplots
#define fluid properties calculations
def calculate_deadoil_vicosity(API, temperature_R):
    '''Calculate Dead Oil viscosity using Glaso'''
    A = 10.313*(log10(temperature_R-460))-36.447
    viscosity = ((temperature_R-460)**-3.444)*(3.141*10**10)*(math.log10(API))**A
    return viscosity
def calculate_satoil_viscosity(Rs,API,temperature_R):
    '''Calculate Saturated Oil Viscosity, using Beggs-Robinson'''
    a = 10.715*(Rs+100)**-.515
    b = 5.44*(Rs+150)**-0.338
    dead_viscosity = calculate_deadoil_vicosity(API,temperature_R)
    viscosity = a*dead_viscosity**b
    return viscosity
def calculate_gas_viscosity(gas_density,gas_spgr,temp_r):
    """Lee-Gonzalez 1966 - gas viscosity in cp"""
    gas_mw = 29 * gas_spgr #29 is air mw
    x = 3.5 + (986/temp_r) + 0.01 * gas_mw
    y= 2.4 - 0.2 * x
    k = (9.4 + 0.02 * gas_mw)*(temp_r**1.5) / (209 + (19 * gas_mw) + temp_r)
    gas_visc = 0.0001 * k * math.exp(x*(gas_density/62.4)**y)
    return gas_visc

#read data
plotgrid = make_subplots(rows=2,cols=2,)
data = pd.read_csv('2.flowstabilitydata.txt', sep='\t')
data['type'] = 0
for i in range(0,len(data)):
    mass = data['mass_rate_kg/s'][i]
    gas_hol = data['gas_hol'][i]
    rohl = data['rohl_kg/m3'][i]
    gor = data['gor_scf/stb'][i]
    wht_r = data['wht_c'][i] * 1.8 + 491
    api = data['api'][i]
    liq_visc = calculate_satoil_viscosity(gor,api,wht_r)
    gas_visc = calculate_gas_viscosity(8.11,0.71,wht_r)
    data['type'][i] = fd.Taitel_Dukler_regime(mass,1-gas_hol ,rohl, 8.1,
                                              liq_visc,gas_visc,1.99/25.6,0)[0]
#using express plot features
fig  = ex.scatter(x=data['mass_rate_kg/s'],
                         y=data['gor_scf/stb'],symbol=data['type'],color=data['type'],
                         text=data['well_name'],size=data['mass_rate_kg/s'],
                         title='Flow Regime Advisor',size_max=30,template='ggplot2',
                         labels={'x':"Mass Rate",'y':'GOR'})

plotly.offline.plot(fig)
```

Example 3: ESP Recommendation Dashboard

This example illustrates the process of building ESP recommendation dashboard based on a design rate and fluid properties. This example imports a full ESP catalog from a text database.

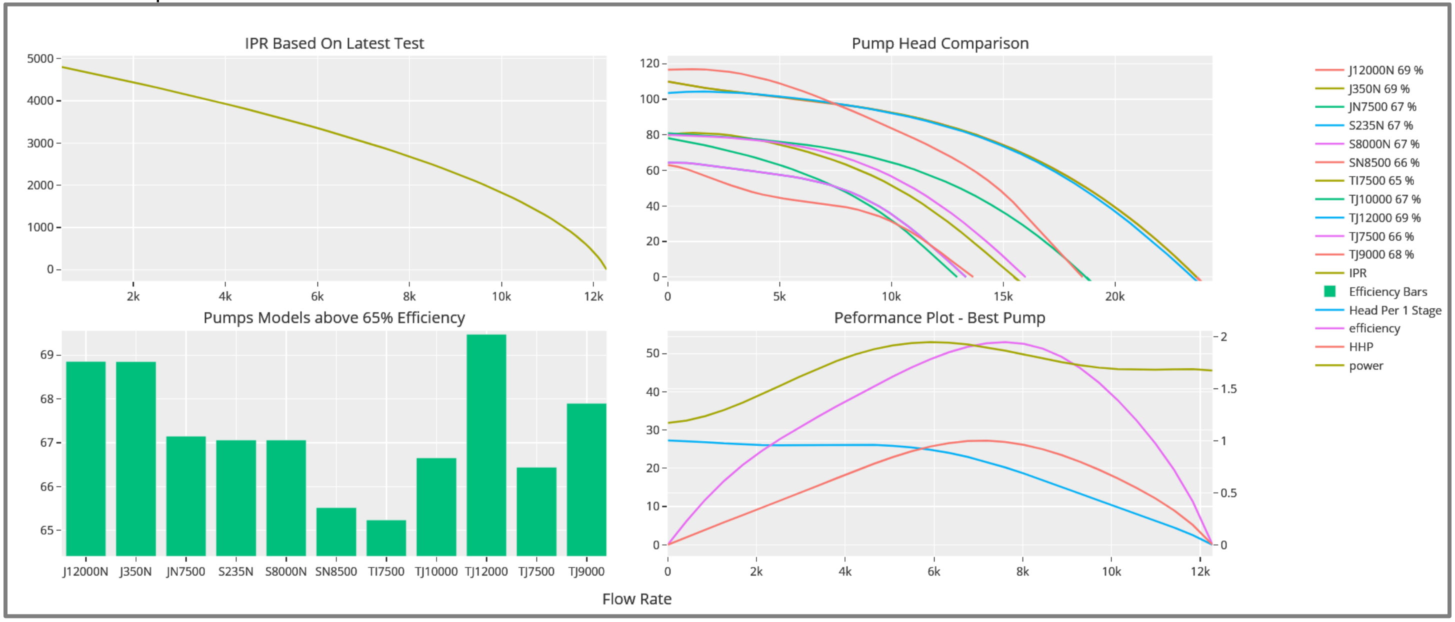
Required Data:

- Esp catalog.
- Design rate.
- Fluid Properties.

Libraries Used:

- Plotly

Plot Example



Python Code:

```
from IPython.display import display, HTML
import pandas as pd
import plotly
import plotly.graph_objects as go
import plotly.express as ex
from  plotly.subplots import make_subplots
#Open ESP Catalog
esp_df = pd.read_csv('3-esp_catalog.csv',sep=',')
#valuate HHP and efficiency %
rhol = 0.88#produced fluid density
esp_df['hhp'] =  esp_df['flow_rate'] * esp_df['head_per_stage']*7.4*60*rhol/3956
esp_df['eff'] =  100 * esp_df['hhp'] / esp_df['power']
print('Unique Models : ', len(esp_df['model'].unique()))
#Get IPR Data and Create Desired Rate:
liquid_rate = 5000 # stbpd
pwf =3650 # psig
pres = 4900 # psig
#Calculate Maximumrate by Vogel
p_ratio = pwf / pres
qo_max_vogel = liquid_rate/(1-0.2*(p_ratio)-0.8*(p_ratio**2))
#Create lambda function for vogel
vogel_liq_rate = lambda pwf : (qo_max_vogel*
                               (1-0.2*(pwf/pres)-0.8*(((pwf/pres))**2)))
#Entered Desired Rate
design_rate = 8000 #stb
#Create IPR
#Create pwf list (assumed)
pwf_list = list(range(0,pres,100))
liquid_rate_list = []
for pressure in pwf_list:
    liquid_rate_list.append(vogel_liq_rate(pressure))#add value to liquid list
#ESP Pump Recommender
#Step 1 : Convert Design Rate to ft3/sec
design_rate_cfps = design_rate * 0.000065 #Conversion factor
#Step 2 : Filter ESP Models
esp_df_filter = esp_df[esp_df['max_rate']>design_rate_cfps ]
esp_df_filter = esp_df_filter[esp_df_filter['min_rate']<design_rate_cfps]
#Group data by model
esp_grouped = esp_df_filter.groupby('model')
#Suggested ESP models
esp_models = list(esp_grouped.groups.keys())
# Establish Plot Template
specss = specs=[[{"secondary_y": True}, {"secondary_y": True}],
                           [{"secondary_y": True}, {"secondary_y": True}]]
plot_fig = make_subplots(rows = 2, cols=2,specs=specss,subplot_titles=['IPR Based On
Latest Test',
                                                           'Pump Head Comparison',
                                                           'Pumps Models above 65%
Efficiency','Peformance Plot - Best Pump'],
                         x_title='Flow
Rate',horizontal_spacing=0.05,vertical_spacing=0.1)
top_pumps = {}
for model in esp_models:
    esp_df_model = esp_df[esp_df['model'] == model]
    #only consider efficiency factors above
    if esp_df_model['eff'].max()<65:
        continue ;
    plot_fig.add_trace(go.Scatter(x=esp_df_model['flow_rate']/0.000065,
```

```
                                  y=esp_df_model['head_per_stage'],name = model+"
"+'{:.0f} %'.format(esp_df_model['eff'].max()),
                       row=1,col=2)
    top_pumps[model] = esp_df_model['eff'].max()

bar_plot = go.Bar(x=list(top_pumps.keys()),y=list(top_pumps.values()),name='Efficiency
Bars')
ipr_plot = go.Scatter(x=liquid_rate_list,y=pwf_list,name='IPR')
plot_fig.add_trace(ipr_plot,row=1,col=1)
#best performing pump
esp_best = esp_df[esp_df['model'] == esp_models[0]]
perf_plot1 =
go.Scatter(x=esp_best['flow_rate']/0.000065,y=esp_best['head_per_stage'],name='Head Per 1
Stage')
perf_plot2 =
go.Scatter(x=esp_best['flow_rate']/0.000065,y=esp_best['eff'],name='efficiency')
perf_plot3 = go.Scatter(x=esp_best['flow_rate']/0.000065,y=esp_best['hhp'],name='HHP')
perf_plot4 =
go.Scatter(x=esp_best['flow_rate']/0.000065,y=esp_best['power'],name='power')
plot_fig.add_trace(bar_plot,row=2,col=1)
plot_fig.add_trace(perf_plot1,row=2,col=2)
plot_fig.add_trace(perf_plot2,row=2,col=2)
plot_fig.add_trace(perf_plot3,row=2,col=2,secondary_y=True)
plot_fig.add_trace(perf_plot4,row=2,col=2,secondary_y=True)
#set plotting style
plot_fig.update_layout(template='ggplot2')
#plot
plotly.offline.plot(plot_fig)
```

Example 4: Wellhead Pressure Smoothing

This example illustrates various techniques might be used to smooth wellhead pressure data that is obtained from DOF projects on seconds/minute bases. Or the pressures that are obtained from continuous monitoring on wellhead pressure during surface well tests.

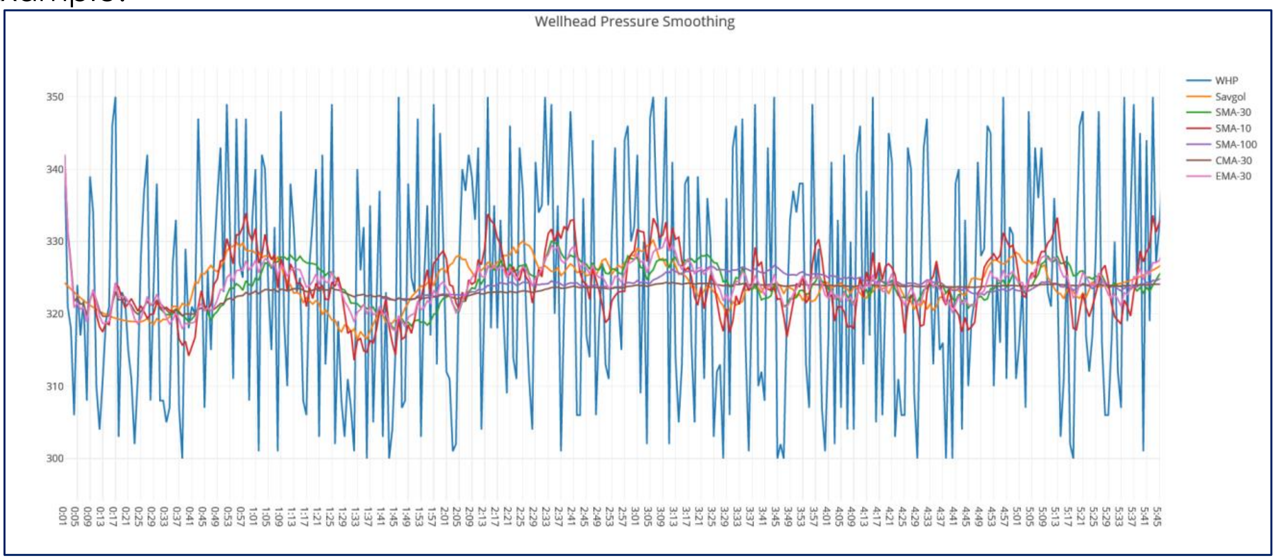
Techniques used for data smoothing:

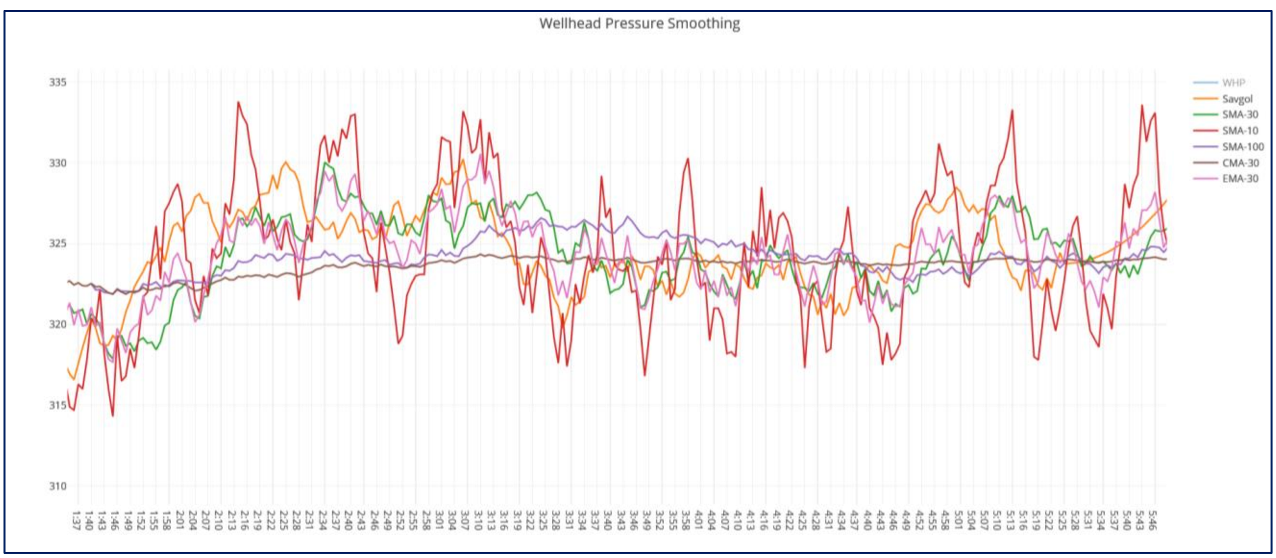
- Savgol Filter.
- SMA(Simple Moving Average)
- CMA (Continuous moving Average)
- EMA (Exponential Moving Average)

Required Data:

- WHP.

Plot Example:





Python Code:

```
import plotly
import plotly.graph_objects as go
from  plotly.subplots import make_subplots
from scipy.signal import *
import pandas as pd
# Read WHP data from file
whp_df = pd.read_csv('4-whp_data.txt',sep='\t')
# Calculate Stavitzky Golay Filter
whp_savgol = savgol_filter(whp_df['WHP_psig'], 49, 2)
whp_df['whp_savgol'] = whp_savgol
#Calculate Simple moving average filter using various spans 30,10,100
whp_df['whp_sma_30'] = whp_df['WHP_psig'].rolling(30).mean()
whp_df['whp_sma_10'] = whp_df['WHP_psig'].rolling(10).mean()
whp_df['whp_sma_100'] = whp_df['WHP_psig'].rolling(100).mean()
#Calculate Cumulative Moving Average
whp_df['whp_cma'] = whp_df['WHP_psig'].expanding().mean()
#Calculate Exponential Moving Average
whp_df['whp_ema'] = whp_df['WHP_psig'].ewm(span=30).mean()
#Create Figure
fig = make_subplots(rows=1,cols=1)
#create Plots
fig.add_trace(go.Scatter(x=whp_df['Time'], y =
whp_df['WHP_psig'],name='WHP'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y =
whp_df['whp_savgol'],name='Savgol'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y = whp_df['whp_sma_30'],name='SMA-
30'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y = whp_df['whp_sma_10'],name='SMA-
10'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y = whp_df['whp_sma_100'],name='SMA-
100'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y = whp_df['whp_cma'],name='CMA-
30'),row=1,col=1)
fig.add_trace(go.Scatter(x=whp_df['Time'], y = whp_df['whp_ema'],name='EMA-
30'),row=1,col=1)
#Setup Plot Rage and template
fig.update_layout(yaxis_range = [0,whp_df['WHP_psig'].max()+100],template='none',
                  title='Wellhead Pressure Smoothing')
#Show Plot
plotly.offline.plot(fig)
```

Chapter 5: Fluid Properties PVT

Example 1: Gas Solubility Using Standing's Method

Gas solubility can be calculating using Standing's infamous correlation:

$$R_s = \gamma_g \left[\left(\frac{p}{18.2} + 1.4 \right) 10^x \right]^{1.2048}$$

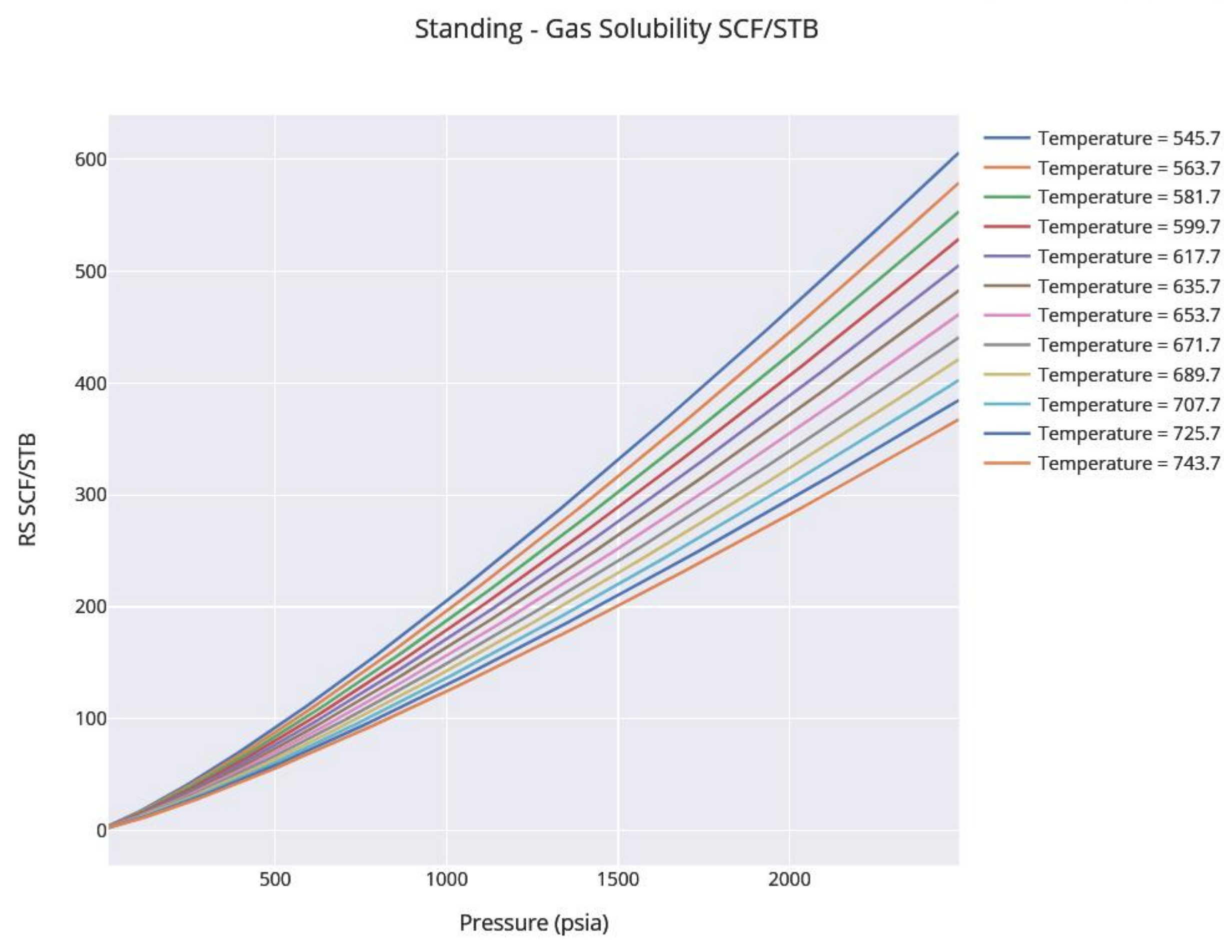
with

$$x = 0.0125\text{API} - 0.00091(T - 460)$$

where:

R_s = gas solubility, scf/STB
T = temperature, °R
p = system pressure, psia

Example Plot:



Python Code:

```
import plotly.graph_objects as go
import plotly
#define solubility correlation
def calculate_solubility(gas_spgr, pressure_psia,API,temperature_R):
    '''Calculate Rs in Scf/STB - Standing'''
    x = 0.0125*API - 0.00091*(temperature_R-460)
    term1=((pressure_psia/18.2)+1.4)*10**x
    Rs = gas_spgr*term1**1.2048
    return Rs
api = 30
temperature_c = 90 #deg C
gas_spgr = 0.7 # From Lab Tests
figure = go.Figure()
#Creating a loop for effect of Pressure and temperature
temp_range = range(30,150,10)
press_range = range(14,2500,20)
rs_list = []
for temp in temp_range:
    for press in press_range:#loop through pressure range
        temp_c = temp * 1.8 + 491.7
        rs_list.append(calculate_solubility(gas_spgr, press, api, temp_c))
    #plot curve @ given temperature
    figure.add_trace(go.Scatter(x=list(press_range),y=rs_list,name='Temperature =
'+str(temp_c)))
    #Clear all items from the list
    rs_list.clear()
#Update graph and show the plot
figure.update_layout(title='Standing - Gas Solubility SCF/STB ',template='seaborn'
                     ,yaxis=dict(title='RS SCF/STB'),xaxis=dict(title='Pressure (psia)'))
plotly.offline.plot(figure)
```

Example 2: Oil Density – Standing's Correlation

Standing Proposed the following correlation to calculate oil density at any give pressure and temperature.

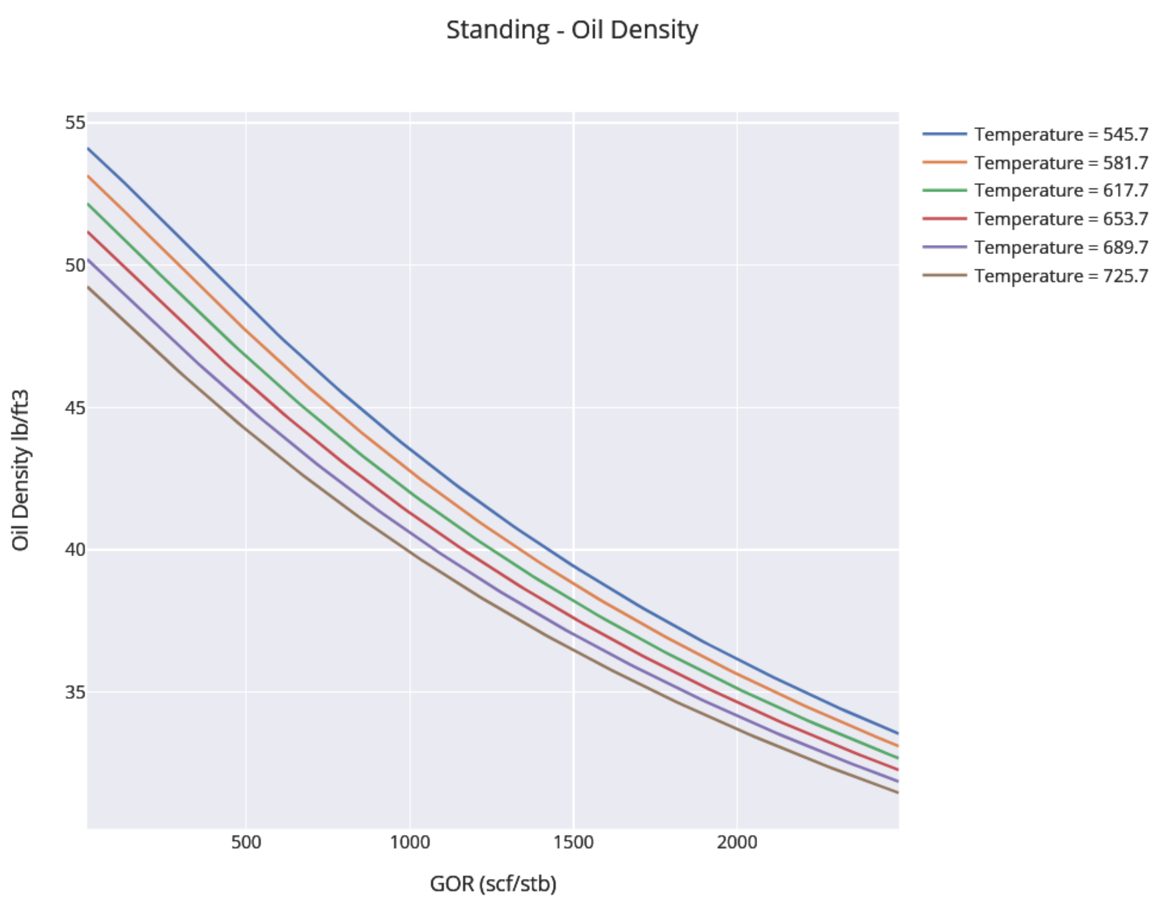
Density can be plotted agaist key influencing factors(e.g Gas contained in oil phase, temperature or Dead Oil Density)

$$\rho_o = \frac{62.4\gamma_o + 0.0136 R_s \gamma_g}{0.972 + 0.000147\left[R_s\left(\frac{\gamma_g}{\gamma_o}\right)^{0.5} + 1.25(T-460)\right]^{1.175}}$$

where

T = system temperature, °R
γ_o = specific gravity of the stock-tank oil, 60°/60°
γ_g = specific gravity of the gas
R_s = gas solubility, scf/STB
ρ_o = oil density, lb/ft^3

Plot Example:



Python Code:

```
import plotly.graph_objects as go
import plotly
from math import sqrt
def calculate_oil_spgr(API):
    spgr = (141.5/(API+131.5))
    return spgr
def calculate_oil_density(API,gas_spgr,Rs,temperature_R):
    '''Calculate Oil Density in lb/ft3'''
    oil_spgr = calculate_oil_spgr(API)
    nominator = 62.4*oil_spgr+.013*Rs*gas_spgr
    denominator = .972 + .000147*(Rs*sqrt(gas_spgr/oil_spgr)+1.25*(temperature_R-
460))**1.175
    density = nominator/denominator
    return density

api = 30
temperature_c = 90 #deg C
gas_spgr = 0.7 # From Lab Tests
solubility = 500 #scf/stb
figure = go.Figure()
#Creating a loop for effect of Pressure and temperature
temp_range = range(30,150,20)
gor_list = range(14,2500,20)
rho_list = []
for temp in temp_range:
    for gor in gor_list:#loop through pressure range
        temp_r = temp * 1.8 + 491.7
        rho_list.append(calculate_oil_density(api,gas_spgr,gor,temp_r))
    #plot curve @ given temperature
    figure.add_trace(go.Scatter(x=list(gor_list),y=rho_list,name='Temperature =
'+str(temp_r)))
    #Clear all items from the list
    rho_list.clear()
#Update graph and show the plot
figure.update_layout(title='Standing - Oil Density ',template='seaborn'
                     ,yaxis=dict(title='Oil Density lb/ft3'),xaxis=dict(title='GOR
(scf/stb)'))
plotly.offline.plot(figure)
```