A Project Report On

"An Approach to Automated Horizon Tracking and Fault Mapping using Machine Learning Algorithms"

Submitted in partial fulfilment of the
Requirement for the Award of degree of

Bachelor of Technology
of
University Of Petroleum and Energy Studies

Specialization in Cloud Computing and Virtualization Technology

Compiled by:

Manik Khurana

SAP ID: 500051975

Enrollment No.: R110216092

Under the guidance of:
Mr. Vivek Gaurav Saini

(Sr. Programming Officer)

GEOPIC, ONGC, Dehradun

DECLARATION

I hereby declare that the work presented in this project report entitled **"An Approach to Automated Horizon Tracking and Fault Mapping using Machine Learning Algorithms"** in partial fulfilment of requirement for the Award of Degree of Bachelor of Technology, submitted in the department of Computer Science Engineering, UPES, Dehradun is an authentic record of my work carried out during the Industrial Training from 16 December 2019 to 17 February 2020, under the guidance of **Mr.Vivek Gaurav Saini**, GEOPIC, ONGC, Dehradun.

Signature of the Student
Manik Khurana

CERTIFICATE

This is to certify that **Mr.** Manik Khurana, a student of Bachelors of Technology Computer Science Engineering spl. in Cloud Computing and Virtualization Technology of UPES, Dehradun has done his Winter Training starting from 16th December 2019 and ending on 17th February 2020 at **GEOPIC, ONGC Dehradun.** The project work entitled **"An Approach to Automated Horizon Tracking and Fault Mapping using Machine Learning Algorithms"** embodies the original work done by **Mr. Manik Khurana** during his, above mentioned, winter training period.

Signature of Project Guide

(Mr. Vivek Gaurav Saini)
Sr. Programming Officer

Signature of Training Coordinator

(Mr. P.R.Meena)
G.M. (Programming)

ACKNOWLEDGEMENT

The winter training at ONGC is a golden opportunity for learning and self-development. I consider myself very lucky and honoured to be able to be a part of it and have so many wonderful and experienced people lead me through the completion of this project. It gives me immense pleasure and a sense of satisfaction to have an opportunity to acknowledge and to express gratitude to those who were associated with me during my industrial training at GEOPIC, ONGC Dehradun.

I express my sincere thanks and gratitude to ONGC authorities for allowing me to undergo the training in this prestigious organization. I would like to thank **Mr. V.K Sharma, GGM (Programming) and Mr. Vivek Gaurav Saini (Sr. Programming officer)** for providing me the technical guidance, opportunity and infrastructure to work.

Finally, I would thank my parents for imparting me moral support and motivation during this project.

Contents

Abstract

All the time invested into the actual Fault Mapping and Horizon Tracking can now be considerably reduced with the approach presented in this project. There have been many researchers on the same mission to find something that can help the geophysicists and geoscientists make a few of their tasks automated. The application presents an idea of how we can change the "segy" file into a much more human readable format and use it for our further findings. The whole code is divided into various snippets for a much better comprehension of the entire piece of it. The actual Horizon Tracking will require this modified data to be put into a different module of code that can identify the faults in the Seismic Graphs and modify it accordingly. The code uses some sample data, links to which are provided as well.

Scope and Objective of the Project

The Scope of this project spans any corporation employing multiple geophysicists and geoscientists to identify faults and track horizon breaks manually. This approach leads to the part where it all can be automated, quicker and removes all chances of human errors.

The Objective of the project is to provide an advancement towards the present scenario in the field of Fault Mapping and Horizon Tracking by preparing the initial set of Data. The project aims to make the entire process automated and presents a cheaper and a much better alternative to the present systems. The learning to the model-to-be-prepared will be in form of Supervised Learning.

Introduction

The project entitled "An Approach to Automated Horizon Tracking and Fault Mapping using Machine Learning Algorithms" aims to present a proposal to the time-costly problem of finding the actual horizons and the faults or the shifts in these horizons in the Seismic Graphs of the location based data.

The program code is divided into small snippets to ensure the major objective of the project as to how it serves as a learning practice to data preparation for future Machine Learning models. The learning is supervised type and this is what makes it all the more efficient.

The true motivation of doing this project was the viability of it in the real market and how these small snippets of code can handle huge problems in the industry. The actual task of Fault Mapping and Horizon Tracking is done by the Geology Department Scientists and as much as experience matters, the automation of a few tasks would greatly help them in making better decisions and having time to think of better opportunities ahead. The project serves the purpose of having data categorized as per our requirements in terms of cross- lines and x-lines with their respective variations with amplitude values.

Resources used and Requirements

- **Python Programming Language**

  Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is a general purpose and high level programming language. You can use Python for developing desktop GUI applications, websites and web applications. Also, Python, as a high level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks.

- **Anaconda Distribution for Python and Navigator**

  Anaconda® is a package manager, an environment manager, a Python/R data science distribution, and a collection of over 7,500+ open-source packages. Anaconda is free and easy to install, and it offers free community support. Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and MacOS.

  Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and

manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.

- **Jupyter Notebook**

  Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension. A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell.

- **OpendTect Software by dGB Earth Sciences**

  OpendTect, a seismic interpretation software system for processing, visualizing and interpreting multi-volume seismic data, and for fast-track development of innovative interpretation tools. It is used to Analyze 2D, 3D, 4D pre- and post-stack seismic data. Data processing and visualization are rigorously integrated in the OpendTect system. Visualization elements can be moved freely through data space to interactively analyze data from stored volumes, or data that are calculated on-the-fly.

The horizon trackers support manual drawing, line tracking and 3D tracking of amplitudes and similarities. Horizons can be tracked on 2D and on 3D seismic data using the 3D scene and/or 2D viewers.

Fault planes and fault-sticks sets. Work flows are designed to draw fault sticks, which is very efficient if you work on a Wacom digitizing tablet.

- **TerraNubis Sample Data**

  Project Netherlands Offshore F3 Block – Complete Data packaged as an OpendTect software readable files with 3D Seismic Data, Acoustic Imdepence, Wells and Horizons.

- **Python Libraries:**
  - **segyio.py**

    Segyio is a small LGPL licensed C library for easy interaction with SEG-Y and Seismic Unix formatted seismic data, with language bindings for Python and Matlab. Segyio is an attempt to create an easy-to-use, embeddable, community-oriented library for seismic applications. Features are added as they are needed; suggestions and contributions of all kinds are very welcome.
  - **numpy.py**

    NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.
  - **matplotlib.py**

    Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Codes and Outputs

A. Tutorial Level Explanation to read the segy file

a. Import Required Libraries

```
import segyio
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from shutil import copyfile
from skimage import exposure
```

b. Make sure you are able to access the sample data

```
filename = '../../data/basic/F3_Similarity_FEF_subvolume_IL230-430_XL475-675_T1600-1800.sgy'
similarity = 1-segyio.tools.cube(filename)
filename1 = '../../data/basic/F3_Dip_steered_median_subvolume_IL230-430_XL475-675_T1600-1800.sgy'
seismic = segyio.tools.cube(filename1)
# Let's make sure we do indeed have ndarrays only.
```

c. Find the Similarity Type

```
print ('similarity type is: ' + str(similarity.__class__.__name__))
OUTPUT: similarity type is: ndarray.
```

d. Find the Seismic type

```
print ('seismic type is: ' + str(similarity.__class__.__name__))
OUTPUT: seismic type is: ndarray.
# Good. Now plotting slice 15 for both.
```

e. Plotting Slice 15 for both

```
fig = plt.figure(figsize=(14,6))
ax = fig.add_subplot(121)
sim = ax.imshow(similarity[:,:,15], cmap='gray_r');
fig.colorbar(sim, ax=ax)
ax.set_xticks([])
ax.set_yticks([])
ax.invert_xaxis()
ax1 = fig.add_subplot(122)
```

```
amp = ax1.imshow(seismic[:,:,15], cmap='gray_r');
fig.colorbar(amp, ax=ax1)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.invert_xaxis()
```

OUTPUT:

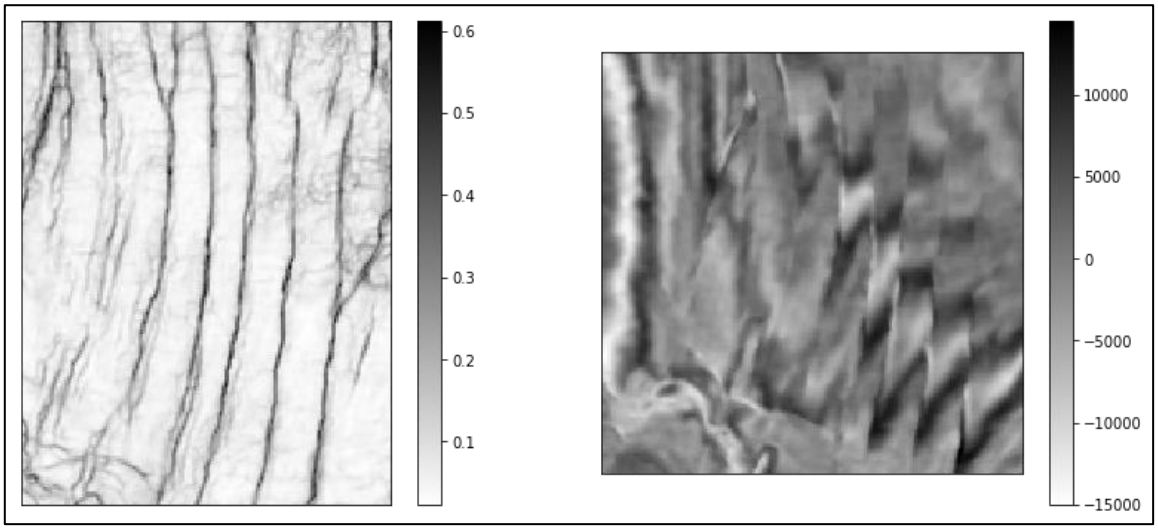

Figure 1: Amplitude slice and similarity slice

```
# That does not look right: the amplitude slice is wider than the similarity slice. Based on the filenames, they are supposed to have identical shape!
```

f. Check if these are equal, obviously they're not. It's visible

```
np.shape(seismic) == np.shape(similarity)
```

OUTPUT: False

g. Print Amplitudes

```
print('Amplitude IL/XL shape: ' + str(np.shape(seismic)[0]) + ' / '
+str(np.shape(seismic)[1]))
```

OUTPUT: Amplitude IL/XL shape: 191 / 146

h. Print Similarity

```
print('Similarity IL/XL shape: ' + str(np.shape(similarity)[0]) + ' / '
+str(np.shape(similarity)[1]))
```

OUTPUT: Similarity IL/XL shape: 191 / 146

```
# It looks like in spite of filenames, the amplitude volume has first two dimensions IL/XL=201/201; conversely the similarity, has first two dimensions 191/146! Let's use segyio to dig a bit deeper.!
```

i. Get inline and crossline statistics for the two volumes

```
with segyio.open(filename, "r") as segyfile:
    # Print inline and crossline ranges
    print('Amplitude Inline range: ' + str(np.amin(segyfile.ilines)) + ' - '
+str(np.amax(segyfile.ilines)))
    print('Amplitude Crossline range: ' + str(np.amin(segyfile.xlines)) + ' - '
+str(np.amax(segyfile.xlines)))
```

OUTPUT:

Amplitude Inline range: 230 - 420

Amplitude Crossline range: 475 - 620

Similarity

```
with segyio.open(filename1, "r") as segyfile:
    # Print inline and crossline ranges
    print('Similarity Inline range: ' + str(np.amin(segyfile.ilines)) + ' - '
+str(np.amax(segyfile.ilines)))
    print('Similarity Crossline range: ' + str(np.amin(segyfile.xlines)) + ' - '
+str(np.amax(segyfile.xlines)))
```

OUTPUT:

Similarity Inline range: 230 - 430

Similarity Crossline range: 475 – 675

```
# NB. That confirmed it: the amplitude volume has 10 extra inlines and 55 extra
crosslines
```

j. Trim seismic for display purposes

```
seismic = seismic[0:-10:1, 0:-55:1,:]
assert (np.shape(seismic) == np.shape(similarity))
fig = plt.figure(figsize=(14,6))
ax = fig.add_subplot(121)
sim = ax.imshow(similarity[:,:,15], cmap='gray_r');
fig.colorbar(sim, ax=ax)
ax.set_xticks([])
ax.set_yticks([])
ax.invert_xaxis()
ax1 = fig.add_subplot(122)
amp = ax1.imshow(seismic[:,:,15], cmap='gray_r');
```

```
fig.colorbar(amp, ax=ax1)
ax1.set_xticks([])
ax1.set_yticks([])
ax1.invert_xaxis()
```

OUTPUT:

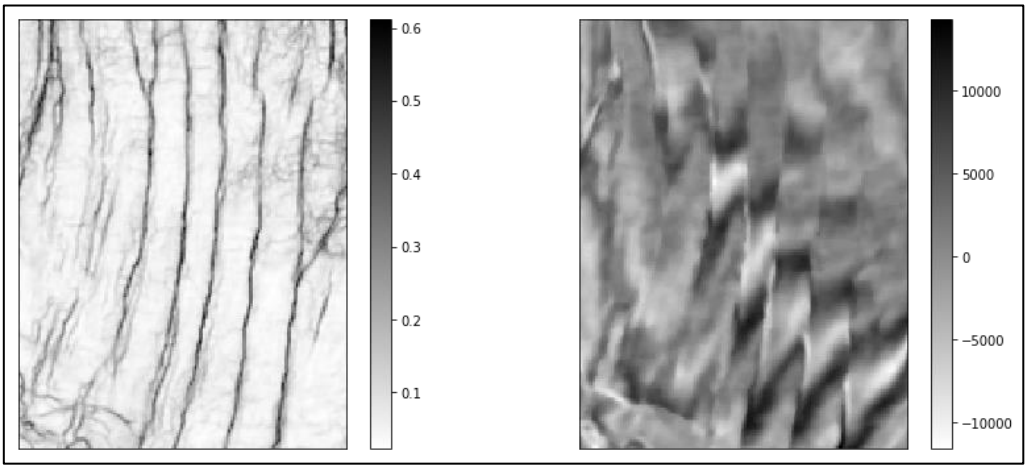

Figure 2: Trimmed Amplitude slice and similarity slice

```
# Perfect!!! Histogram analysis, threshold definition, threshold to make binary fault image
```

k. Let's calculate the square of the similarity to stretch its range. This will help with the following histogram-based thresholding

```
similarity = np.power(similarity,2)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(similarity[:,:,15], cmap='gray_r',  vmin =0, vmax=0.4);
plt.gca().invert_xaxis()
plt.colorbar();
```

OUTPUT:

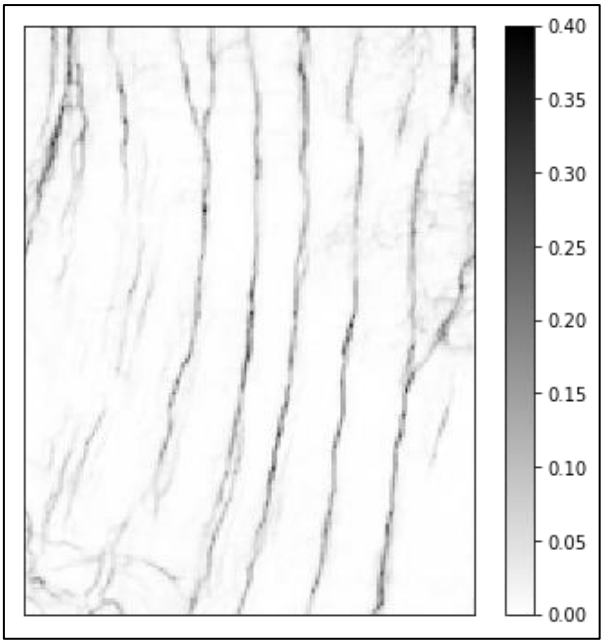

Figure 3: Square of the Similarity

l. Histogram

```
hi_sim = exposure.histogram(similarity)
plt.plot(hi_sim[1], hi_sim[0], 'brown')
plt.ylim(0, 300000)
plt.axvline(0.1, color='b', ls='--')
plt.axvline(0.05, color='g', ls='--')
plt.xlim(0,0.3);
```

OUTPUT:

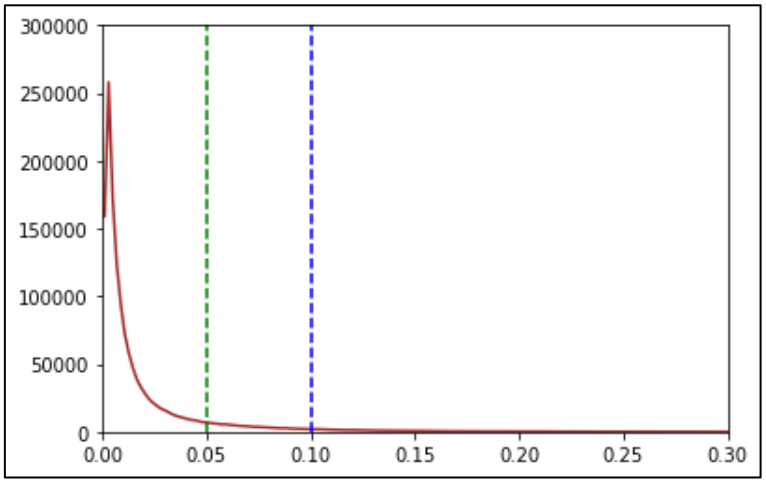

Figure 4: Histogram

```
# That shows a value of (squared) similarity of 0.05 is a good threshold to make the binary fault volume !!
```

m. Binary Fault Volume

```
binary = np.zeros(similarity.shape, dtype=np.uint8)
binary[similarity > 0.05] = 1
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(binary[:,:,15], cmap='gray_r');
plt.gca().invert_xaxis()
plt.colorbar();
```

OUTPUT:

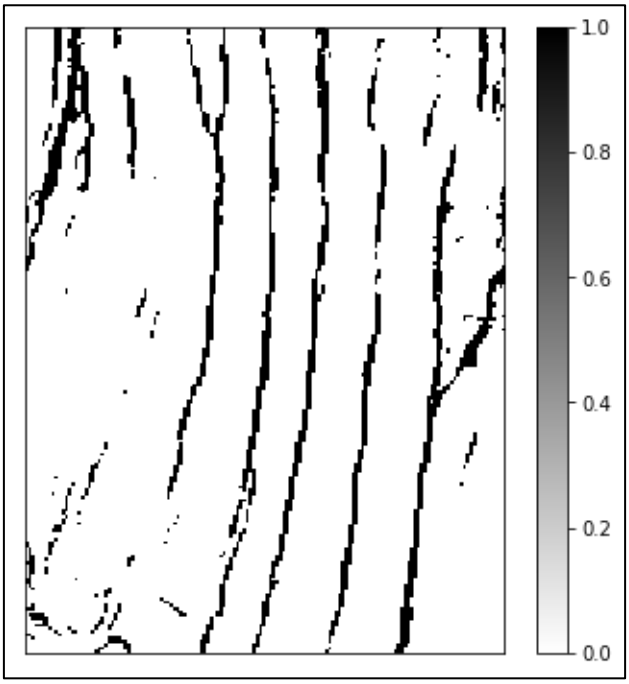

Figure 5: Binary Fault Volume

```
# That looks good. The next bit is to clean-up small minutia in the binary fault image. In this case I use 10 pixels after a bit of trial and error.
```

n. Binary Fault Image

```
label_objects, nb_labels = ndi.label(binary)
sizes = np.bincount(label_objects.ravel())
mask_sizes = sizes > 10
mask_sizes[0] = 0
cleaned = mask_sizes[label_objects]*1
fig = plt.figure(figsize=(6, 6))
```

```
ax = fig.add_subplot(1, 1, 1)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(cleaned[:,:,15], cmap='gray_r');
plt.gca().invert_xaxis()
plt.colorbar();
```

OUTPUT:

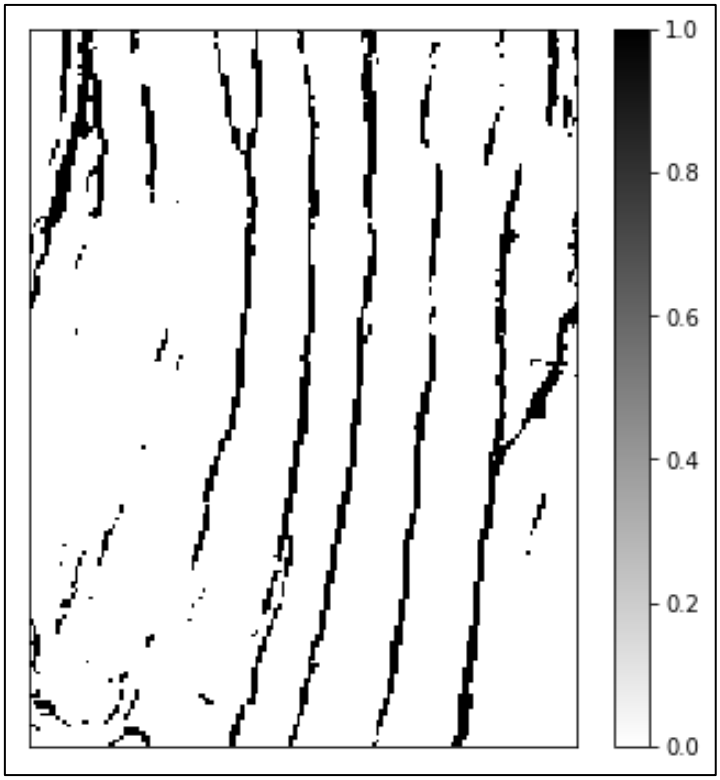

Figure 6: Binary Fault Image

o. For display purposes, make fault mask to display superimposed on amplitude. Display seismic with superimposed fault mask

```
masked = np.zeros((np.shape(cleaned)))
masked[cleaned == 0] = np.nan

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(1, 1, 1)
ax.set_xticks([])
ax.set_yticks([])
plt.imshow(seismic[:,:,15], cmap='gray_r')
plt.colorbar()
```

```
plt.imshow(masked[:,:,15], cmap='Reds')
plt.gca().invert_xaxis()
cb = plt.colorbar()
cb.set_ticks([])
```

OUTPUT:

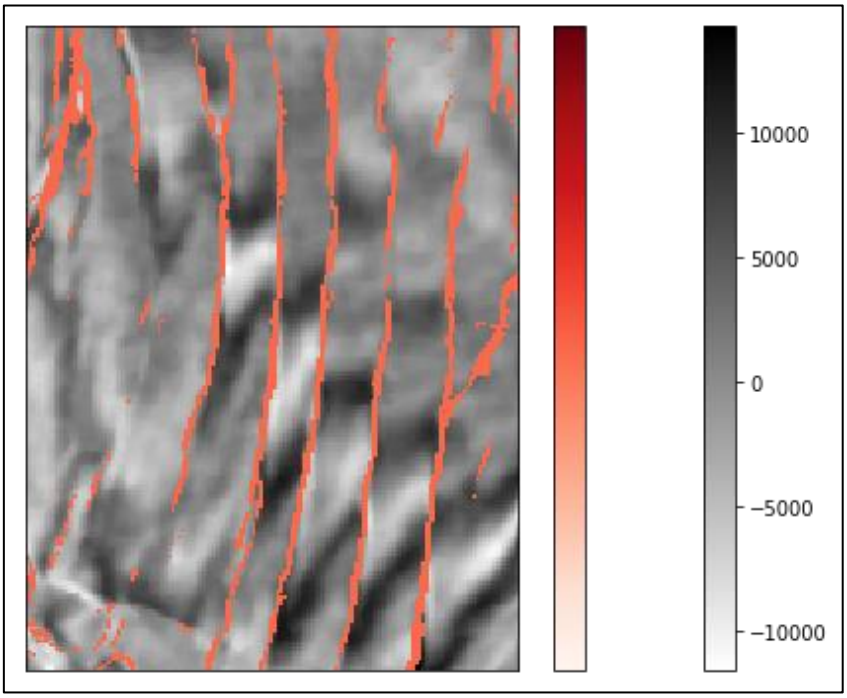

Figure 7: Fault Mask

p. Export Fault Volume

```
# To export re-using the input header, I make a copy of the input similarity SEGY file, then write to it the cleaned fault array. But first, let's test to ensure inlies are the fast mode. This is important to work with, especially with larger volumes. This currently works, but it is unsupported functionality
print('Is inline the fast mode? ' + str(segyfile.fast is segyfile.iline))
```

OUTPUT: Is inline the fast mode? True

```
# Export also requires a double index: One zero-based to iterate through the corresponding vertical slices in the cleaned fault array. And one based on inline range to iterate through inlines (the fast mode) in the output file
dim = np.shape(similarity)
idx = np.arange(0, dim[0])
print(np.amin(idx), np.amax(idx))
lns = np.arange(np.amin(segyfile.ilines), np.amax(segyfile.ilines)+1)
print(np.amin(lns), np.amax(lns))
```

```
OUTPUT:
0 190
230 430
# Finally Ready to export
```

q. Export

```
cleaned = cleaned.astype('float32')
input_file = '../../data/basic/F3_Similarity_FEF_subvolume_IL230-430_XL475-675_T1600-1800.sgy'
output_file = '../../data/basic/F3_Faults_subvolume_IL230-430_XL475-675_T1600-1800.sgy'
copyfile(input_file, output_file)
with segyio.open(output_file, "r+") as dst:
    for i,j in zip(lns,idx):
        dst.iline[i] = cleaned[j,:,:]
!rm ../../data/basic/F3_Faults_subvolume_IL230-430_XL475-675_T1600-1800.sgy
```

B. Segy file – Basic attributes, headers parsing, and plotting

```
# This notebook will go through the basics of segyio file handling. It covers: File reading, Bin, text and trace headers parsing, Trace headers quicklook, Section (assuming the segy is a 2D poststack section). The 2D seismic line in this post is from the USGS NPRA Seismic Data Archive, and are in the public domain. The line number is 3X_75_PR
```

a. Import Required Libraries

```
import re
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import segyio
filename = '../../data/basic/3X_75_PR.SGY'
```

b. Utility Functions

```
# The following functions are defined to split the workflow into independent parts. They allow for the parsing of both text and traces headers into a dict and a pandas dataframe respectively.
def parse_trace_headers(segyfile, n_traces):
```

```
    # Parse the segy file trace headers into a pandas dataframe. Column names are
defined from segyio internal tracefield One row per trace
        # Get all header keys
    headers = segyio.tracefield.keys
    # Initialize dataframe with trace id as index and headers as columns
    df = pd.DataFrame(index=range(1, n_traces + 1),
                      columns=headers.keys())
    # Fill dataframe with all header values
    for k, v in headers.items():
        df[k] = segyfile.attributes(v)[:]
    return df


def parse_text_header(segyfile):
    # Format segy text header into a readable, clean dict
    raw_header = segyio.tools.wrap(segyfile.text[0])
    # Cut on C*int pattern
    cut_header = re.split(r'C ', raw_header)[1::]
    # Remove end of line return
    text_header = [x.replace('\n', ' ') for x in cut_header]
    text_header[-1] = text_header[-1][:-2]
    # Format in dict
    clean_header = {}
    i = 1
    for item in text_header:
        key = "C" + str(i).rjust(2, '0')
        i += 1
        clean_header[key] = item
    return clean_header
```

c. Get basic attributes and parse headers

```
# The file reading is done within a with statement. The ignore_geometry=True
argument is passed for non 3D specific segys
with segyio.open(filename, ignore_geometry=True) as f:
    # Get basic attributes
    n_traces = f.tracecount
    sample_rate = segyio.tools.dt(f) / 1000
    n_samples = f.samples.size
```

```
    twt = f.samples
    data = f.trace.raw[:]  # Get all data into memory (could cause on big files)
    # Load headers
    bin_headers = f.bin
    text_headers = parse_text_header(f)
    trace_headers = parse_trace_headers(f, n_traces)
f'N Traces: {n_traces}, N Samples: {n_samples}, Sample rate: {sample_rate}ms'
OUTPUT: 'N Traces: 1592, N Samples: 751, Sample rate: 8.0ms'
# The twt variable stores the two-way time for each sample.
```

```
bin_headers
```

OUTPUT: {JobID: 1110580000, LineNumber: 3, ReelNumber: 1110579969, Traces: 1, AuxTraces: 0, Interval: 8000, IntervalOriginal: 0, Samples: 751, SamplesOriginal: 0, Format: 1, EnsembleFold: 1, SortingCode: 4, VerticalSum: 1, SweepFrequencyStart: 0, SweepFrequencyEnd: 0, SweepLength: 0, Sweep: 0, SweepChannel: 0, SweepTaperStart: 0, SweepTaperEnd: 0, Taper: 0, CorrelatedTraces: 0, BinaryGainRecovery: 1, AmplitudeRecovery: 4, MeasurementSystem: 0, ImpulseSignalPolarity: 0, VibratoryPolarity: 0, SEGYRevision: 0, TraceFlag: 0, ExtendedHeaders: 0}

```
text_headers
```

OUTPUT: {'C01': ' CLIENT/JOB ID   0 5 5 5 2 9 3 4 7 2 ',
 'C02': ' LINE L3X ',
 'C03': ' REEL NO 751023083601   DAY-START OF REEL 23   YEAR 1975 ',
 'C04': ' INSTRUMENT: MFG TI   MODEL ASC ',
 'C05': ' DATA TRACES/RECORD0001   AUXILIARY TRACES/RECORD 0 CDP FOLD 0001 ',
 'C06': 'SAMPLE INTERNAL 0000008000 US SAMPLES/TRACE 0751BITS/IN 1600 BYTES/SAMPLE 4 ',
 'C07': ' RECORDING FORMAT STDI   FORMAT THIS REEL SEG Y1 ',
 'C08': ' SAMPLE CODE: FLOATING PT ',
 'C09': ' GAIN TYPE: FLOATING PT C C C C C C C C C C C ',
 'C10': ' PROCESSING: ',

'C11': ' PROCESSING: ',
'C12': ' FIELD TAPE PROCESSING MACHINE NUMBER IS: TIMAP2 ',
'C13': ' INPUT TAPE FORMAT IS :   STDI ',
'C14': ' TMIN REQUESTED THIS REEL 00000000   TMAX REQUESTED THIS REEL 00006000 ',
'C15': ' INITIAL CHANNEL REQUESTED   0001   NUMBER OF CHANNELS REQUESTED 0001 ',
'C16': ' DELTA 0008 MILLIVOLT LEVEL 000000 TYPE INPUT 0001 ',
'C17': ' TRACE HEADER INFORMATION BY BYTE : ',
'C18': ' BYTES 181-182 : TMIN THIS TRACE   BYTES 183-184 : TMAX THIS TRACE ',
'C19': ' BYTES 185-186 : TRACE FLAG 1= GOOD   2 = MISSING OR DEAD TRACE INPUT ',
'C20': ' 3= REQUESTED DUMMY TRACE   BYTES 187-188 : TDS THIS TRACE ',
'C21': 'BYTES 189-192 : WATER DEPTH AT CDP   BYTES 193-196 : PRENMO DATUM CORR MS T ',
'C22': 'BYTES 197-200 : POSTNMO DATUM CORR MS T ',
'C23': 'BYTES 201-204 : DATUM TIME -- TIME TO WHICH TRACES HAVE BEEN CORRECTED ',
'C24': 'BYTES 205-208 : DATUM VELOCITY   - TO WHICH TRACES HAVE BEEN CORRECTED C ',
'C25': 'BINARY TAPE HEADER IDENTIFICATION :   JOB ID - 32 BIT FIXED POINT ',
'C26': 'REEL ID- 32 BIT PACKED BINARY BASE 10 BYTES 1-4 JOB ID BYTES 5-8 REEL ID ',
'C27': 'BYTES 95-96 SDOM: 0=SPC, 1= CDP ',
'C28': 'END PROCESSING: '}

trace_headers.columns

list the trace headers keys

OUTPUT:

Index(['TRACE_SEQUENCE_LINE', 'TRACE_SEQUENCE_FILE', 'FieldRecord',

'TraceNumber', 'EnergySourcePoint', 'CDP', 'CDP_TRACE',

'TraceIdentificationCode', 'NSummedTraces', 'NStackedTraces', 'DataUse',
'offset', 'ReceiverGroupElevation', 'SourceSurfaceElevation',
'SourceDepth', 'ReceiverDatumElevation', 'SourceDatumElevation',
'SourceWaterDepth', 'GroupWaterDepth', 'ElevationScalar',
'SourceGroupScalar', 'SourceX', 'SourceY', 'GroupX', 'GroupY',
'CoordinateUnits', 'WeatheringVelocity', 'SubWeatheringVelocity',
'SourceUpholeTime', 'GroupUpholeTime', 'SourceStaticCorrection',
'GroupStaticCorrection', 'TotalStaticApplied', 'LagTimeA', 'LagTimeB',
'DelayRecordingTime', 'MuteTimeStart', 'MuteTimeEND',
'TRACE_SAMPLE_COUNT', 'TRACE_SAMPLE_INTERVAL', 'GainType',
'InstrumentGainConstant', 'InstrumentInitialGain', 'Correlated',
'SweepFrequencyStart', 'SweepFrequencyEnd', 'SweepLength', 'SweepType',
'SweepTraceTaperLengthStart', 'SweepTraceTaperLengthEnd', 'TaperType',
'AliasFilterFrequency', 'AliasFilterSlope', 'NotchFilterFrequency',
'NotchFilterSlope', 'LowCutFrequency', 'HighCutFrequency',
'LowCutSlope', 'HighCutSlope', 'YearDataRecorded', 'DayOfYear',
'HourOfDay', 'MinuteOfHour', 'SecondOfMinute', 'TimeBaseCode',
'TraceWeightingFactor', 'GeophoneGroupNumberRoll1',
'GeophoneGroupNumberFirstTraceOrigField',
'GeophoneGroupNumberLastTraceOrigField', 'GapSize', 'OverTravel',
'CDP_X', 'CDP_Y', 'INLINE_3D', 'CROSSLINE_3D', 'ShotPoint',
'ShotPointScalar', 'TraceValueMeasurementUnit',
'TransductionConstantMantissa', 'TransductionConstantPower',
'TransductionUnit', 'TraceIdentifier', 'ScalarTraceHeader',
'SourceType', 'SourceEnergyDirectionMantissa',
'SourceEnergyDirectionExponent', 'SourceMeasurementMantissa',
'SourceMeasurementExponent', 'SourceMeasurementUnit', 'UnassignedInt1',

```
       'UnassignedInt2'],
      dtype='object')
```

```
trace_headers.head()
```

OUTPUT:

	TRACE_SEQUENCE_LINE	TRACE_SEQUENCE_FILE	FieldRecord	TraceNumber	EnergySourcePoint	CDP	CDP_TRACE	TraceIdentificationCode	NSummedTraces	NStackedTraces	...	TraceIdentifier	ScalarTraceHeader	SourceType	SourceEnergyDirectionMantissa	SourceEnergyDirectionExponent	SourceMeasurementMantissa	SourceMeasurementExponent	SourceMeasurementUnit	UnassignedInt1	UnassignedInt2
1	1	1	11	0	0	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
2	2	2	11	0	0	2	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
3	3	3	11	0	0	3	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
4	4	4	11	0	0	4	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0
5	5	5	11	0	0	5	1	1	1	1	...	0	0	0	0	0	0	0	0	0	0

5 rows × 91 columns

```
# Individual traces can then be accessed (e.g. get the FieldRecord number for the first trace)

trace_headers.loc[1, 'FieldRecord']
```

OUTPUT: 11

d. Plotting Headers

```
# Basic header QC might be achieved by plotting relevant values together. A quick example is shown here of creating a 2D header plot (without any meaning).
plt.style.use('ggplot')  # Use ggplot styles for all plotting
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(1, 1, 1)
ax.plot(trace_headers['FieldRecord'],
trace_headers['TRACE_SEQUENCE_FILE'], '-k')
```

```
ax.set_xlabel('Field Record Number (FFID)')
ax.set_ylabel('Trace sequence within file')
ax.set_title('Basic Header QC')
OUTPUT: Text(0.5, 1.0, 'Basic Header QC')
```

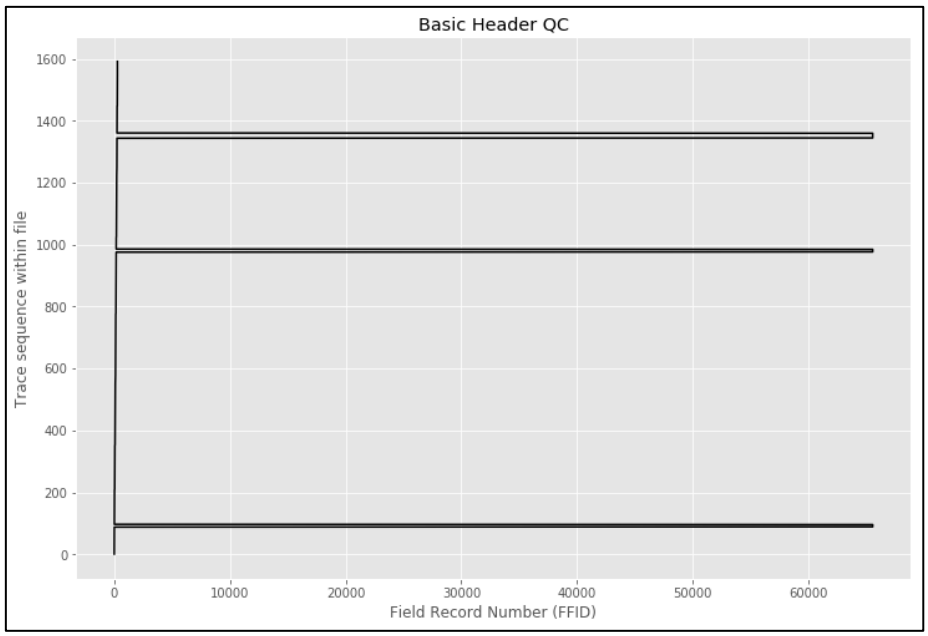

Figure 8: Basic Header QC

e. Plotting Section

```
# The straightforward way to display a segy file from segyio format is to use matplotlib.pyplot.imshow, which treats 2D arrays as images.
In order to scale and standardize the display, it is useful to get the nth percentile of the amplitudes
clip_percentile = 99

vm = np.percentile(data, clip_percentile)

f'The {clip_percentile}th percentile is {vm:.0f}; the max amplitude is {data.max():.0f}'

OUTPUT: 'The 99th percentile is 1802; the max amplitude is 1091224'

# Let's get to plotting!  Notice that we have to transpose the array to plot it like this. The reason is that we're storing things with traces in the first dimension ('rows' if you like), for convenience. This way data [0] refers to the first trace, not the first time sample. But imshow assumes that we're looking at a sort of image, with rows going across the image. The extent of the image is also populated from the number of trace to display and the samples two-way times

fig = plt.figure(figsize=(18, 8))
```

```
ax = fig.add_subplot(1, 1, 1)

extent = [1, n_traces, twt[-1], twt[0]]  # define extent

ax.imshow(data.T, cmap="RdBu", vmin=-vm, vmax=vm, aspect='auto',
extent=extent)

ax.set_xlabel('CDP number')

ax.set_ylabel('TWT [ms]')

ax.set_title(f'{filename}')

OUTPUT: Text(0.5, 1.0, '../../data/basic/3X_75_PR.SGY')
```

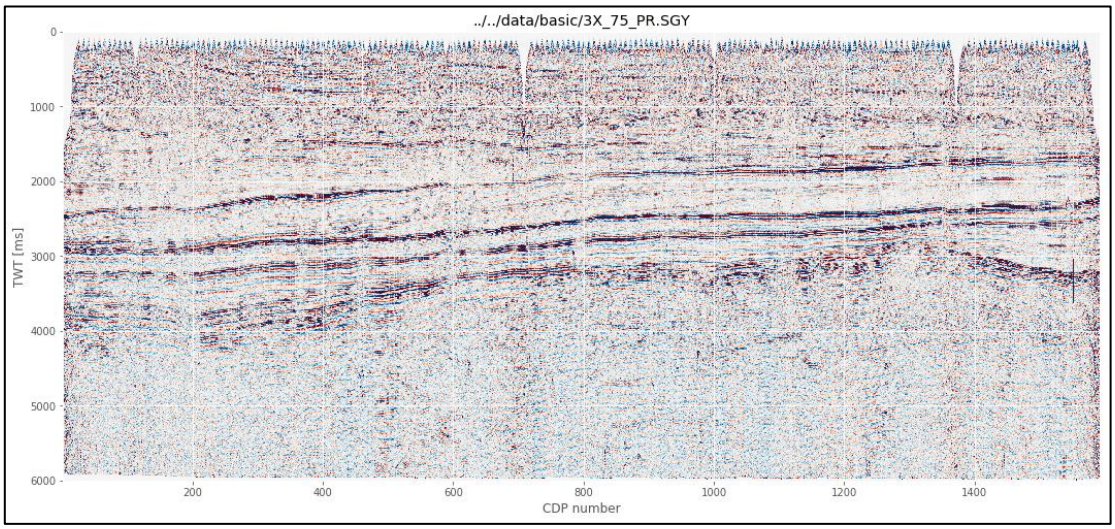

Figure 9: Plotting Section

Conclusion

This application was successful in this objective of helping the organization with the exquisite Jupyter Notebooks with the Python Code required for the seismic data preparation for future analytics. The code snippets when combined form the complete code that solves the purpose.

References

1. Kuhlman, Dave. "*A Python Book: Beginning Python, Advanced Python, and Python Exercises*"
2. https://github.com/equinor/segyio-notebooks
3. https://github.com/equinor/segyio
4. https://agilescientific.com/blog/2016/9/21/x-lines-of-python-read-and-write-seg-y
5. https://mycarta.wordpress.com/2019/03/12/working-with-3d-seismic-data-in-python-using-segyio-and-numpy-mostly/
6. https://jupyter.org/install
7. https://docs.anaconda.com/anaconda/install/