DASH FINAL PROJECT DOCUMENTATION

KENYAN FARMERS ARC2 SATELLITE PIXELS ALLOCATION ¶

INTRODUCTION

About ARC2 DATASET

> Africa Rainfall Climatology version2 (ARC2) is a data that is as a result of a project to create satellite-estimated precipitation climatology over the Africa domain over various timescales. Files in the ARC2 dataset consist of binary-formatted grids at a 0.1 degree resolution. The spatial extent of all estimates spans 20.0W,-55.0E and 40.0S,-40.0N. Daily data files are written in compressed binary data format and consist of one record (one array) of floating point rainfall estimates in mm. Each daily array equals 751*801 elements, corresponding to 751 pixels in the X direction and 801 pixels in the Y direction.

ARC2 SATELLITE PIXEL ALLOCATION DASH PROJECT

> While insuring farmers considering rainfall as the risk, the insurer will be interested in some historical rainfall information of the location of the farmer. To aid in this, the insurer will be interested in knowing on which pixel the farmer falls, so as to use that pixel's historical data in calculating and setting premium for the client. The objective of my project is to place a farmer on one of the pixels from the ARC2 dataset. I have concentrated on Kenya as a country, thus the dataset I've used is only for Kenya, and is a subset of the ARC2 dataset. Based on the farmer's latitude and longitude inputs, and with the ARC2 data for Kenya loaded, the farmer will be matched to the nearest pixel and the finer details of his/her area together with a plot of his/her location on the map will also be displayed as part of the output.

```
### THE CODE
>Below is a summary explanation of the code:

>Load all the required packages
<code>
import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import os
import numpy as np
```

```
from scipy import spatial
import plotly.graph_objs as go
from dash.dependencies import Input, Output, State
import plotly.express as px
</code>

>Initialize the app
<code>
external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
colors = {
    'background': '#87CEEB',
    'text': '#7FDBFF',
    'subheadingsText': "red",
    'outputText': "blue",
    'submitText': 'green'
}
</code>

>Load the ARC2 dataset for Kenya
<code>
grid = pd.read_csv('C:/Users/HP/OneDrive/wycliffe/Africa Data
School/Assignments/Project/the.grid.Kenya_pixelboundaries.csv')
</code>

>The app layout
>The app will first prompt the user to enter the latitude and longitude of
his/her location. The locational attributes of the farmer will be displayed on
the right side, and the plot will be on the left.
<code>
app.layout = html.Div(children =[
    html.Div(children=[
                html.H1("KENYAN FARMERS ARC2 SATELLITE PIXELS ALLOCATION",
                style={'textAlign': 'center','color': colors['text']}
                )]),
    html.Div(children =[
        html.Div(children=[
            html.Div(children=[
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["Latitude"]),
                    html.Div(dcc.Input(id='latitude', type='number', min=-4.5,
max=4.5, step=0.001, value=0))
                ], style={'width': '150px'}),
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']},children=["Longitude"]),
                    html.Div(dcc.Input(id='longitude', type='number', min=34,
max=42, step=0.001, value=0))
                   ], style={'width': '150px'})
            ], style={'display': 'flex', 'flex-direction': 'row'}),
            html.Div(children=[
                html.Button('Submit', id='submit-button', n_clicks=0, style=
{'textAlign': 'center','backgroundColor': colors['submitText']})
            ]),
            html.Div(children=[
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["PixelName"]),
                    html.Div(id='Nearest-Pixel', style = {"color":
colors['outputText']})
                ], style={'width': '150px'})
            ]),
            html.Div(children=[
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["PixelLatitude"]),
                    html.Div(id='PixelCenterPoint(Lat)', style = {"color":
colors['outputText']})
                ], style={'width': '150px'}),
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["PixelLongitude"]),
                    html.Div(id='PixelCenterPoint(Lon)', style = {"color":
colors['outputText']})
                ], style={'width': '150px'})
            ], style={'display': 'flex', 'flex-direction': 'row'}),
            html.Div(children=[
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["Country"]),
                    html.Div(id='Country', style = {"color":
colors['outputText']})
                ], style={'width': '150px'}),
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["County"]),
                    html.Div(id='County', style = {"color":
colors['outputText']})
                ], style={'width': '150px'})
            ], style={'display': 'flex', 'flex-direction': 'row'}),
            html.Div(children=[
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["Constituency"]),
                    html.Div(id='Constituency', style = {"color":
colors['outputText']})
                ], style={'width': '150px'}),
                html.Div(children=[
                    html.Br(),
                    html.Label(style={"font-weight": "bold", 'color':
colors['subheadingsText']}, children=["Ward"]),
                    html.Div(id='Ward', style = {"color":
colors['outputText']})
                ], style={'width': '150px'})
            ], style={'display': 'flex', 'flex-direction': 'row'})
```

```
        ], style={'backgroundColor': colors['background'], 'padding': 10,
'flex': 1}),
        html.Div(children=[
            html.Label(style={"font-weight": "bold", 'textAlign': 'center',
'color': colors['subheadingsText']}, children=["LOCATION OF THE FARMER ON THE
MAP"]),
            dcc.Graph(id='my-graph', figure={})
        ], style={'border': 'solid 2px blue'})
    ], style={'display': 'flex', 'flex-direction': 'row'})
])
</code>

>The app server
>I have used KDTree function to match the user to the nearest pixel based on
the user inputs and output the locational features. In addition to this, I
have used mapbox to render a plot of the location on a map.
<code>
@app.callback(Output('Nearest-Pixel', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    pixel = str(grid.loc[index, ["PixelName"]].values)[2:-2]
    return pixel
@app.callback(Output('Country', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    country = str(grid.loc[index, ["Country"]].values)[2:-2]
    return country
@app.callback(Output('PixelCenterPoint(Lat)', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    lat = str(grid.loc[index, ["Latitude"]].values)[1:-1]
    return lat
@app.callback(Output('PixelCenterPoint(Lon)', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    lon = str(grid.loc[index, ["Longitude"]].values)[1:-1]
    return lon
```

```
@app.callback(Output('County', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    county = str(grid.loc[index, ["county"]].values)[2:-2]
    return county
@app.callback(Output('Constituency', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    constituency = str(grid.loc[index, ["constituency"]].values)[2:-2]
    return constituency
@app.callback(Output('Ward', 'children'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_output(n_clicks, input1, input2):
    the_grid = grid[['Latitude', 'Longitude']].to_numpy()
    pixelpoints = spatial.KDTree(the_grid)
    distance, index = pixelpoints.query((input1,input2), k=1)
    ward = str(grid.loc[index, ["ward"]].values)[2:-2]
    return ward
@app.callback(Output('my-graph', 'figure'),
    [Input('submit-button', 'n_clicks')],
    [State('latitude', 'value'),
    State('longitude', 'value')])
def update_graph(n_clicks, input1, input2):
    data = pd.DataFrame([
    {"Latitude" : input1, "Longitude": input2}
    ])
    fig = px.scatter_mapbox(data, lat="Latitude", lon="Longitude", zoom=3,
mapbox_style='open-street-map')
    return fig
</code>

>Run app
<code>
if __name__ == '__main__':
    app.run_server(debug=False, use_reloader=False)
</code>

>I have deployed the app to production through heroku.
```

PROJECT FLOW

Inputs

1. Latitude
2. Longitude

Outputs

1. Allocated pixel name based on the user(farmer) inputs(geographical location)
2. The center points of the allocated pixel, i.e. pixel's latitude and longitude
3. Country
4. County
5. Constituency
6. Ward

Outputs(plots/visualization)

1. The location of the user on the map

CONCLUSION

> With the pixel identified, the insurer will now narrow down to the data for that specific pixel, and use it to process the insurance policy, i.e. calculate premium.

In [ ]: