Investigating Civil Unrest

PSDS 4900 – Technical Report | Kaleb Kolb

Git Repo: https://github.com/kcky7d/psds-capstone

Contents

I. Introduction

Civil unrest is a topic that ruled the headlines in the United States over the past few years. From the Black Lives Matter protests to the Capitol riots and everything in between, it seems like there was a different form of civil unrest happening in the news every day. Civil conflict is far from a US-only problem though, so this project explores the topic of international civil unrest. This is an attempt to understand the factors that may foster an environment ripe for protests and rioting.

A. Questions to Answer

What factors contribute to civil unrest and can they be broken into categories? The following are some examples:

- Economic – income, gross domestic product, expenses, trade, inflation
- Education – school attendance, education expenditure, level of education, teacher stats
- Healthcare – fertility rates, healthcare expenditures, death rates, nutrition
- Infrastructure – access to electricity, internet, cell phones, gas prices
- Social Issues – labor force stats, poverty rates, unemployment, women's issues
- Other – corruption, homicides, migration, rural and urban populations

B. Data Sources

Armed Conflict Location and Event Data (ACLED) Project provides data on civil unrest across Africa, the Middle East, Latin America and the Caribbean, East Asia, South Asia, Southeast Asia, Central Asia and the Caucasus, Europe, and most recently the United States.[1] The data is aggregated from various new sources. This capstone project used worldwide protest and riot data from 2010 to 2019 (non-US).

The World Bank provides worldwide developmental indicator data compiled from officially recognized international sources.[2] The final project analyzed 92 factors (17x economic, 15x education, 10x healthcare, 19x infrastructure, 11x social issues, 20x other) over a 10-year span (2010-2019) for 216 countries and territories.

II. Data Acquisition and Carpentry

A. PostgreSQL Database

This project utilized the open-source PostgreSQL relational database management system.[3] The database, titled *unrestdatabase*, had a total of 93 tables filled with ACLED, World Bank, or data created from the two.

B. Armed Conflict Location and Event Data (ACLED) Project Data

The ACLED data was acquired using an issued API key with a query through the website's *Data Export Tool*. The queries were broken up by year to limit the file size of the exported Comma Separated Value (CSV) file. As the datasets became more robust the individual file sizes for the year increased over 10-fold. The 2010 CSV file was 3.44 mb and slowly increased to 6.27 mb in 2015 before doubling to 13.5 mb in 2016 and continuing to grow up to 49.4 mb in 2019, which could not be uploaded to

[1] (ACLED, 2021)
[2] (The World Bank, 2020)
[3] (The PostgreSQL Global Development Group, 2021)

GitHub due to size limitations. After analyzing the difference between the aggregated protest data from 2010 and 2019, the ACLED project accumulated 95 additional countries and territories over that time span, which accounts for the large increase in file size. The 2010 data contained 48 countries in the Middle East, Africa, and Southeast Asia in places that have had long histories of war and a weak or non-existent central government. Figure 1 is an example of a query using the tool. Figure 2 is an example of the output.

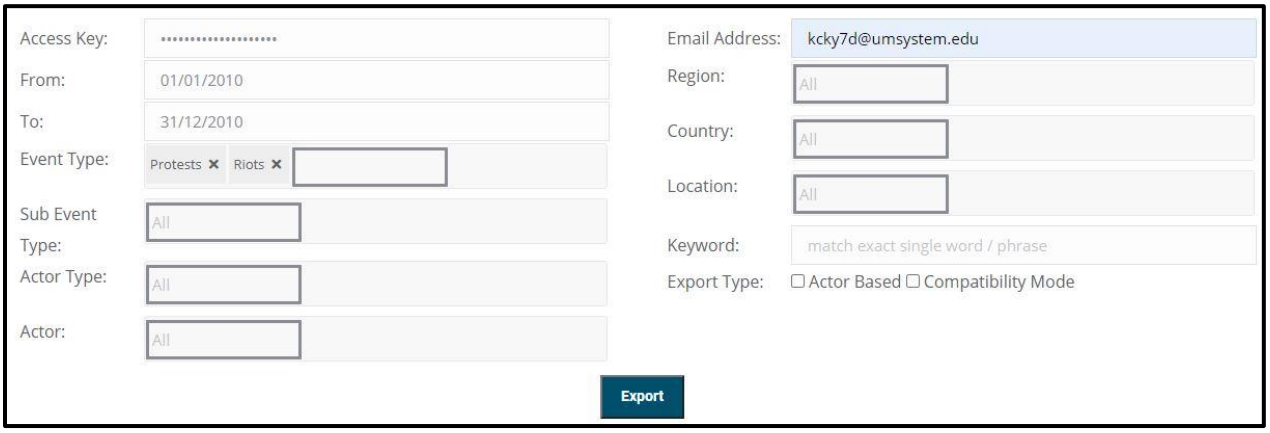

Figure 1: Data Export Tool Query, Protests and Riots for 2010

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	data_id	iso	event_id_	event_id_	event_date	year	time_prec	event_typ	sub_even	actor1	assoc_acto	inter1	actor2	assoc_actor_2	inter2
2	7263461	586	PAK5379	5379	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
3	7263430	586	PAK5383	5383	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
4	6059942	788	TUN97	97	31-Dec-10	2010	1	Protests	Protest wi	Police Forces of Tuni		1	Protesters (Tunisia)	UGTT: Tunisian G	6
5	6059916	788	TUN99	99	31-Dec-10	2010	1	Protests	Excessive	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
6	6059874	788	TUN89	89	31-Dec-10	2010	1	Protests	Protest wi	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
7	6059753	788	TUN88	88	31-Dec-10	2010	1	Protests	Protest wi	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
8	6059492	788	TUN98	98	31-Dec-10	2010	1	Protests	Peaceful p	Protesters	Journalist	6			0
9	6059387	788	TUN95	95	31-Dec-10	2010	1	Protests	Excessive	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
10	6059281	788	TUN96	96	31-Dec-10	2010	1	Protests	Protest wi	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
11	6059270	788	TUN93	93	31-Dec-10	2010	1	Protests	Excessive	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
12	6059177	788	TUN90	90	31-Dec-10	2010	1	Protests	Excessive	Police Forces of Tuni		1	Protesters (Tunisia)	Lawyers (Tunisia	6
13	5926514	50	BGD9735	9735	31-Dec-10	2010	1	Riots	Mob viole	Rioters (B	Bohoramp	5	Rioters (Bangladesh)	Bohorampur Cor	5
14	5590613	50	BGD9742	9742	31-Dec-10	2010	1	Riots	Mob viole	Rioters (B	Vigilante (	5	Civilians (Bangladesh)		7
15	5590611	50	BGD9740	9740	31-Dec-10	2010	1	Riots	Mob viole	Rioters (B	BCL: Bangl	5	Civilians (Bangladesl	BNP: Bangladesh	7
16	5590609	50	BGD9738	9738	31-Dec-10	2010	1	Riots	Mob viole	Rioters (Bangladesh)		5	Rioters (Bangladesh)		5
17	5590608	50	BGD9737	9737	31-Dec-10	2010	1	Riots	Mob viole	Rioters (Bangladesh)		5	Rioters (Bangladesh)		5
18	5533261	144	SRI1297	1297	31-Dec-10	2010	1	Riots	Mob viole	Rioters (Si	UPFA: Uni	5	Civilians (Sri Lanka)	UNP: United Nat	7
19	5529186	586	PAK5407	5407	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
20	5529185	586	PAK5406	5406	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
21	5529184	586	PAK5405	5405	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
22	5529183	586	PAK5404	5404	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
23	5529182	586	PAK5403	5403	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
24	5529181	586	PAK5402	5402	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
25	5529180	586	PAK5401	5401	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
26	5529179	586	PAK5400	5400	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
27	5529177	586	PAK5399	5399	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0
28	5529176	586	PAK5398	5398	31-Dec-10	2010	1	Protests	Peaceful p	Protesters (Pakistan		6			0

Figure 2: Protests and Riots ACLED Output for 2010

First, the ACLED data was pulled from the CSV file into a Pandas data frame using the *read_csv* function.[4] Next, the rows related to protests and riots were filtered separately, then the data was grouped by country and the entries were counted. The two tables were merged by year creating a

[4] https://github.com/kcky7d/psds-capstone/tree/main/acled

data frame with country, protest counts, and riot counts for the years 2010 – 2019. Another CSV file containing country codes, country names, and region was pulled in and merged with the previous data frame to provide a primary key (country code) between the ACLED data and the World Bank data. These data frames, one per year, were then written to tables in the *unrestdatabase* using the SQLAlchemy library. Figure 3 is a code snippet showing the transformation from CSV to database. Figure 4 shows an example from one of the ACLED tables, titled *ACLED_2010*.

```
import os
from sqlalchemy import create_engine

acled_cc = pd.read_csv('acled/acled_cc.csv')
acled_cc = [acled_cc["Country"], acled_cc["CC"], acled_cc["Region"]]
headers = ["country", "Country_Code", "Region"]
acled_df = pd.concat(acled_cc, axis=1, keys=headers)


for filename in os.listdir('acled'): ### Placed acled files in folder in directory 'acled'
    if filename.startswith('protests'):  ### acled exported csvs are titled like protests_riots_2010.csv
        year = filename[-8:-4]

        file = 'acled/' + filename
        df = pd.read_csv(file)

        protest_column = "Protest_Count_" + year
        df_protests = df[(df.event_type == 'Protests')]
        df_protests_country = df_protests.groupby("country")
        df_protests_counts = df_protests_country.count()
        df_protests_counts = df_protests_counts[['data_id']]
        df_protests_counts.reset_index(inplace=True)
        df_protests_counts = df_protests_counts.rename(columns = {'index':'country'})
        df_protests_counts = df_protests_counts.rename(columns = {"data_id": protest_column})

        riot_column = "Riot_Count_" + year
        df_riots = df[(df.event_type == 'Riots')]
        df_riots_country = df_riots.groupby("country")
        df_riots_counts = df_riots_country.count()
        df_riots_counts = df_riots_counts[['data_id']]
        df_riots_counts.reset_index(inplace=True)
        df_riots_counts = df_riots_counts.rename(columns = {'index':'country'})
        df_riots_counts = df_riots_counts.rename(columns = {"data_id": riot_column})

        df_counts = pd.merge(df_protests_counts, df_riots_counts, how='outer', sort=True, on='country')

        df_toDB = pd.merge(df_counts, acled_df, how='inner', sort=True, on='country')

        table_name = "acled_" + year
        engine = create_engine('postgresql://postgres:password@127.0.0.1:5432/unrestdatabase')
        df_toDB.to_sql(table_name, engine)
        print("ACLED Table for", year, "created.")
```

Figure 3: Code to loop through ACLED exported data, perform data carpentry, and write to database.

```
3  select * from acled_2010;
```

Data Output

	index bigint	country text	Protest_Count_2010 double precision	Riot_Count_2010 double precision	Country_Code text	Region text
1	0	Algeria	8	10	DZA	Africa
2	1	Banglade...	576	1138	BGD	Asia
3	2	Benin	9	[null]	BEN	Africa
4	3	Botswana	[null]	1	BWA	Africa
5	4	Burkina F...	3	1	BFA	Africa
6	5	Burundi	3	3	BDI	Africa
7	6	Cambodia	180	22	KHM	Asia
8	7	Cameroon	4	[null]	CMR	Africa
9	8	Central A...	4	1	CAF	Africa
10	9	Democra...	4	7	COD	Africa
11	10	Djibouti	2	[null]	DJI	Africa

Figure 4: Portion of acled_2010 table from unrestdatabase.

C. World Bank Development Indicators

The World Bank Development Indicator data was acquired using HTTP API calls. Figure 5 is an example of an API call for the United States (USA) pulling the "Income share held by highest 10%" (SI.DST.10TH.10) indicator. One call was made per factor per country and returned data back to the 1960s (or as far back as available) in a JSON string. Acquiring data over a 10-year span for 92 factors in 216 countries and territories required 19,872 (92x216) individual API calls – something clearly needing to be automated. Figure 6 is a sample of the returned JSON string from a World Bank query.

Figure 5: Example API Call using country code USA and indicator code SI.DST.10TH.10

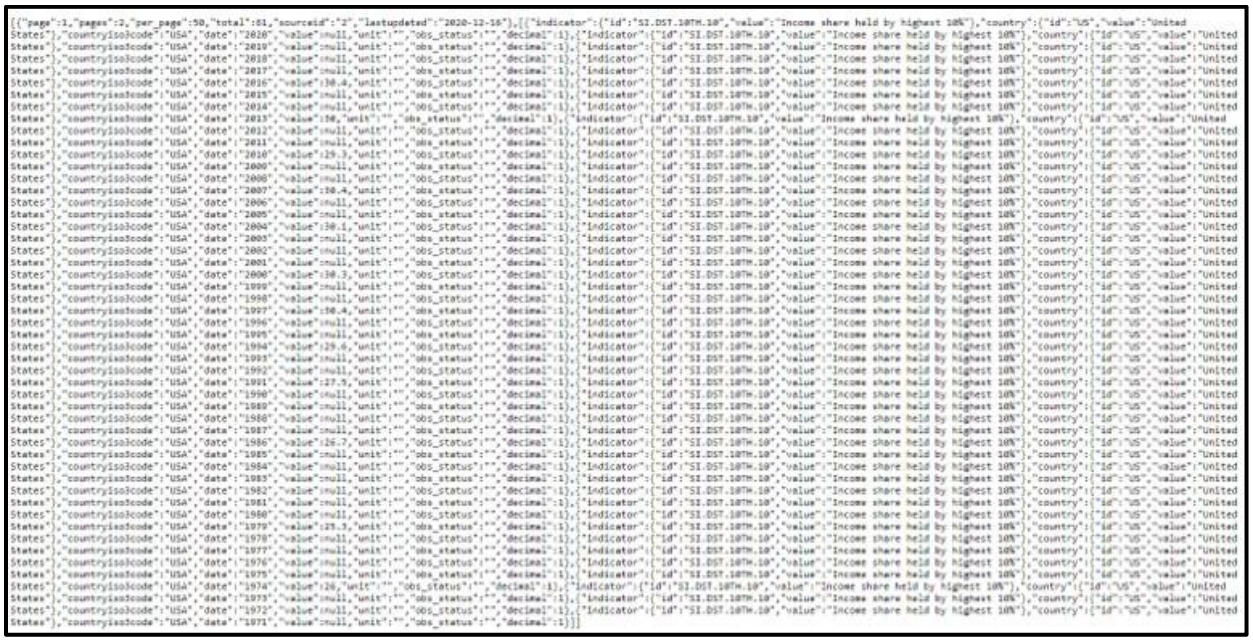

Figure 6: JSON string returned from World Bank API call

Due to the fact that nearly 20,000 API calls were going to be made, a script was created to loop through the different factors and countries to make the calls. Prior to making the API calls, the factors/indicators were downloaded into a Microsoft Excel file, organized into six categories (economic, education, healthcare, infrastructure, social issues, and other), and filtered down from 394 indicators to 92. The indicators are in a file named *factors_categories_short.xls*, with the factors broken into the different categories by worksheet.[5] These were loaded into a python dictionary with the indicator code as the key and the indicator name as the value and saved into their own tables in the *unrestdatabase* using the Psycopg2 library. Next, code was written with a nested loop to run through the list of countries and indicators, make the API call and pull the 10 years of data. This data was saved into a python dictionary, one for each year, then committed into the database once all of the factors for a single country were complete. Figure 7 is a snippet of that code and Figure 8 is the result.

[5] https://github.com/kcky7d/psds-capstone/blob/main/factors_categories_short.xls

```
start_time = time.time()
for cc in cc_list:

    ### As api calls time out, I don't want to repeat table entries, so checking if they exist and mov
    check_statement = 'select * from economic_2010 where "COUNTRY.CODE" =  ' + "'" + cc + "'"
    cursor.execute(check_statement)
    if bool(cursor.rowcount):
        continue

    values_dict_2010 = {'COUNTRY.CODE': cc}   ### dictionaries to hold values pulled from API
    values_dict_2011 = {'COUNTRY.CODE': cc}
    values_dict_2012 = {'COUNTRY.CODE': cc}
    values_dict_2013 = {'COUNTRY.CODE': cc}
    values_dict_2014 = {'COUNTRY.CODE': cc}
    values_dict_2015 = {'COUNTRY.CODE': cc}
    values_dict_2016 = {'COUNTRY.CODE': cc}
    values_dict_2017 = {'COUNTRY.CODE': cc}
    values_dict_2018 = {'COUNTRY.CODE': cc}
    values_dict_2019 = {'COUNTRY.CODE': cc}

    for indicator in econ_indicators:
        time.sleep(1)  ### Delay for API calls
        indicator_in = indicator[0].strip()
        url = ("http://api.worldbank.org/v2/country/%s/indicator/%s?format=json" % (cc, indicator_in))
        r = requests.get(url = url)
        if r.status_code == 200:
            data_pull = r.json()
            data_pull = json.dumps(data_pull)
            data_pull = json.loads(data_pull)

            ### years 2010 - 2019 correspond to indices 10 - 1, respectively
            values_dict_2010[indicator_in] = data_pull[1][10]['value']
            values_dict_2011[indicator_in] = data_pull[1][9]['value']
            values_dict_2012[indicator_in] = data_pull[1][8]['value']
            values_dict_2013[indicator_in] = data_pull[1][7]['value']
            values_dict_2014[indicator_in] = data_pull[1][6]['value']
            values_dict_2015[indicator_in] = data_pull[1][5]['value']
            values_dict_2016[indicator_in] = data_pull[1][4]['value']
            values_dict_2017[indicator_in] = data_pull[1][3]['value']
            values_dict_2018[indicator_in] = data_pull[1][2]['value']
            values_dict_2019[indicator_in] = data_pull[1][1]['value']

        else:
            print(r.status_code)
            continue

### 2010 Econ Data
insert_statement = """INSERT INTO economic_2010 ("""
insert_ind = ""
insert_val = ""
count = 0
for ind, val in values_dict_2010.items():
    count += 1
    if count < len(values_dict_2010):
        insert_ind = insert_ind + '"' + ind + '", '
        insert_val = insert_val + "'" + str(val) + "'" + ", "
    else:
        insert_ind = insert_ind + '"' + ind + '") '
        insert_val = insert_val + "'" + str(val) + "'" + ");"
insert_statement = insert_statement + insert_ind + " VALUES (" + insert_val
cursor.execute(insert_statement)
```

Figure 7: Code to make World Bank API call and save to database.

```
3  select * from economic_2010;
```

Data Output

	COUNTRY.CODE character (3)	NY.ADJ.NNTY.KD.ZG character (255)	NY.ADJ.NNAT.GN.ZS character (255)	MS.MIL.TOTL.TF.ZS character (255)	IC.FRM.BRIB.ZS character (255)
1	AFG	None	-9.67647720747311	4.27692086790267	None
2	ALB	5.60905979043629	5.17228114294992	1.20278488137335	None
3	DZA	15.8815062036008	43.7672570153693	2.86408843929482	None
4	ASM	None	None	None	None
5	AND	None	None	None	None
6	AGO	67.3975955790787	8.86069997627526	1.22379291917601	51.3
7	ATG	None	None	None	6.9
8	ARG	12.2232067957289	8.48310328504133	0.572577740708329	10.3

Figure 8: Example of World Bank data saved to database.

The code was designed to wait until all of the data for an entire country was saved into a dictionary before committing the data to the database. The reason for this is that The World Bank API would occasionally send an error message after timing out when too many calls were made consecutively. By waiting until all of the data for a country was complete, it made it easier to check if the data had already been pulled and pick up where the loop previously left off. It took approximately 34 seconds per country, per category to pull the data and fill the database. For example, to populate the economic data for all 216 countries and territories over 10 years took 2.05 hours (Figure 9). The entire database took approximately 10 hours to finish populating.

```
VUT  Economic Data input Complete
--- 7134.915424823761 seconds ---
VEN  Economic Data input Complete
--- 7170.6819360256195 seconds ---
VNM  Economic Data input Complete
--- 7203.347138643265 seconds ---
VIR  Economic Data input Complete
--- 7239.218963623047 seconds ---
PSE  Economic Data input Complete
--- 7275.497928619385 seconds ---
YEM  Economic Data input Complete
--- 7320.4866988658905 seconds ---
ZMB  Economic Data input Complete
--- 7357.522523641586 seconds ---
ZWE  Economic Data input Complete
--- 7392.47868180275 seconds ---
Econ Tables Complete
--- 7392.479679107666 seconds ---
```

Figure 9: Cumulative time to populate Econ Data.

D. Additional Data Carpentry

The data originally looked to be fairly complete and well-structured, but after some initial exploratory data analysis, I realized much of it was very incomplete (Figure 10).

Data Output

	COUNTRY.CODE character (3)	IC.TAX.GIFT.ZS character (255)
1	AFG	None
2	ALB	None
3	DZA	None
4	ASM	None
5	AND	None
6	AGO	34.2
7	ATG	6.1
8	ARG	8.7
9	ARM	None
10	ABW	None

Data Output

	COUNTRY.CODE character (3)	IC.FRM.BRIB.ZS character (255)
57	DOM	12.3
58	ECU	None
59	EGY	15.2
60	SLV	4.2
61	GNQ	None
62	ERI	None
63	EST	None
64	SWZ	6.7
65	ETH	None
66	FRO	None

Figure 10: Incomplete World Bank data examples

Pandas has an *interpolate* method which can be used to fill in NaN values using an interpolation method.[6] At this point, however, the way the data is saved in the database is not conducive to using the interpolate method because the data is saved in different tables by years. To interpolate, there must be values in the same category as those with the NaNs, so the data needed to be organized in such a way the values for each year are in a single data frame. In an effort to limit the amount of data to interpolate, it was decided to focus on the countries with the top five unrest counts (sum of protests and riots). For each year, the top five countries were added to a python set and from there those countries' data were pulled from each one of the tables, concatenated, interpolated over, and written back to the database (Figure 11, Figure 12).

```
for year in year_list:
    protest_ = "Protest_Count_" + year
    riot_ = "Riot_Count_" + year
    total_ = "Total_" + year
    acled_ = "acled_" + year
    query_countries = 'select *, ("' + protest_ + '" + "' + riot_ + '") as "' + total_ + '" from "' + acled_ + '";'
    acled = pd.read_sql_query(query_countries, con = engine)
    acled_top = acled.sort_values(by=[total_], ascending = False)[0:5]
    top_5 = acled_top.Country_Code
    for country in top_5:
        country_set.add(country)

for cc in country_set:
    table_name = cc.lower() + "_" + "data"
    df_cc = pd.DataFrame()
    for year in year_list:
        protest_ = "Protest_Count_" + year
        riot_ = "Riot_Count_" + year
        total_ = "Total_" + year
        acled_ = "acled_" + year
        df_year = pd.DataFrame()
        for category in categories:
            cat = category + '_' + year
            query = "select * from " + cat + " WHERE \"COUNTRY.CODE\" = '" + cc + "';"
            df = pd.read_sql_query(query, con = engine)
            df = df.replace(regex='^None', value = np.nan)
            float_columns = df.drop(["COUNTRY.CODE"], axis=1).columns
            df[float_columns] = df[float_columns].astype(float)
            df_year = pd.concat([df_year, df], axis=1)
        acled_query = 'select *, ("' + protest_ + '" + "' + riot_ + '") as "' + total_ + '" from ' + acled_ + ' WHE
        acled_df = pd.read_sql_query(acled_query, con = engine)
        df_year = pd.concat([df_year, acled_df], axis=1)
        df_year = df_year.rename(columns={protest_: "Protest_Count", riot_:"Riot_Count", total_:"Total_Count"})
        df_year = df_year.transpose()
        df_cc = pd.concat([df_cc, df_year], axis = 1)
    df_cc.columns = column_years
    df_cc = df_cc.drop(["COUNTRY.CODE", "Country_Code", "Region", "country"])
    df_cc = df_cc.astype(float)
    df_cc['Feature'] = df_cc.index
    df_cc['Country_Code'] = cc
    df_cc = df_cc.interpolate(method = 'linear', axis = 0, limit = 10, limit_direction='both')
    ### write to database
    df_cc.to_sql(table_name, engine)
```

Figure 11: Code to pull data, concatenate, and interpolate missing values.

[6] (Pandas, 2021)

```
select * from egy_data;
```

Output

index text	Year_2010 double precision	Year_2011 double precision	Year_2012 double precision	Year_2013 double precision
NY.AD...	3.68133559199863	3.94162490062675	3.62378124404825	3.88620037475637
NY.AD...	11.8301900914417	12.3182177685101	8.04918852188592	9.12370121294811
MS.MI...	3.04776020813631	3.0028399210298	2.92281256078261	2.84058243397422
IC.FR...	11.3408463194826	11.1551360885177	10.8152100400812	17.4
GC.DO...	19.6339324308289	19.3074322560057	18.7076075193797	23.5662754058838
GC.XP...	27.9270185421752	27.4597284234936	26.6000049986783	29.7325508117676
per_all...	36.4861094450378	35.232164474198	34.4924024779768	35.8988262176514
SL.EM...	45.0452003479004	43.0046005249023	42.3847999572754	42.0651016235352
GC.XP...	25.3537875020719	25.5952228137991	22.3054998769484	22.1252838391237
NY.GD...	5.14723485873294	1.76457194925801	2.22619979662137	2.18546605471226

Figure 12: Egypt data after interpolation

III. Exploratory Data Analysis

A. Correlations

The intent of this project was to try to identify the factors which create an environment ripe for civil unrest to occur. At this point, it seemed appropriate to use the Panda's correlate function since the data now seemed complete. The correlate method (pandas.DataFrame.corr) computes a pairwise correlation of columns.[7] Correlations report a value between -1 and 1. The closer they are to the upper and lower limits indicates higher positive or negative correlations, respectively. Because of this, a negative correlation (approaching -1) can be just as indicative as a positive correlation (approaching 1). A script was written to loop through the indicators and years to find factors that have a strong correlation (either positive or negative) with civil unrest. Indicators with an absolute correlation value greater than 0.75 were saved into a python set to DE duplicate for further investigation (Figure 13). This resulted in 10 indicators to further explore, which can be seen in Figure 14.

B. Examining Correlation Indicators

The indicators listed in Figure 14 were then plotted against protest, riot, and total (protest + riot) counts for each year (10 indicators x 10 years x 3 = 300 total plots). The results were not conclusive and did not appear to correlate as well as anticipated ($|x| > .75$). The issue likely arose because of outliers in the data. Even though the data points were legitimate, grouping them with the others likely skewed the correlation results. In the future finding better groups to compare would likely yield more reliable results. Figure 15 is an example of plots based off of the correlation indicators. Most of the other 299 had similar, dispersed results.

[7] (Pandas, 2021)

```
from sqlalchemy import create_engine
import pandas as pd
engine = create_engine('postgresql://postgres:password@127.0.0.1:5432/unrestdatabase')

top_countries = ['bgd', 'bhr', 'bra', 'egy', 'ind', 'irn', 'lbn', 'lka',
                 'mex', 'nga', 'npl', 'pak', 'tun', 'tur', 'ukr', 'zaf']

year_list = ["2010","2011","2012","2013","2014","2015","2016","2017","2018","2019"]

results_year_dict = {}

indicator_set = set()

for year in year_list:
    year_input = "Year_" + year
    df_year = pd.DataFrame(columns=['Feature'])
    for country in top_countries:
        dict_entry = country + "_" + year
        table_name = country + "_data"
        query = 'select "Feature", "' + year_input + '" as "' + dict_entry + '" from "' + table_name + '";'
        df_result = pd.read_sql_query(query, con=engine)

        df_year = pd.merge(left=df_year, right = df_result, how='right', on='Feature')
    results_year_dict[year_input] = df_year

### Do correlation for each year and combine to find top indicators over 10 years
for df in results_year_dict:
    df_transpose = results_year_dict[df].set_index('Feature').transpose()
    df_corr = df_transpose.corr()
    df_abs = abs(df_corr[["Total_Count"]])
    for index in df_abs[df_abs["Total_Count"]>=0.75].index:
        if(index != "Protest_Count" and index != "Riot_Count" and index != "Total_Count"):
            indicator_set.add(index)
```

Figure 13: Code to find correlation values greater than 0.75.

EN.POP.SLUM.UR.ZS - Population living in slums (% of urban population)	***SP.DYN.CDRT.IN - Death rate, crude (per 1,000 people)***
SH.DTH.1014 - Number of deaths ages 10-14 years	***SP.POP.TOTL - Population, total***
SH.DTH.1519 - Number of deaths ages 15-19 years	***SP.RUR.TOTL - Rural population***
SH.DTH.2024 - Number of deaths ages 20-24 years	***SP.URB.TOTL - Urban population***
SH.XPD.CHEX.PC.CD - Current health expenditure per capita (current USD)	***VC.PKP.TOTL.UN - Presence of peace keepers***

Figure 14: Correlation values greater than 0.75

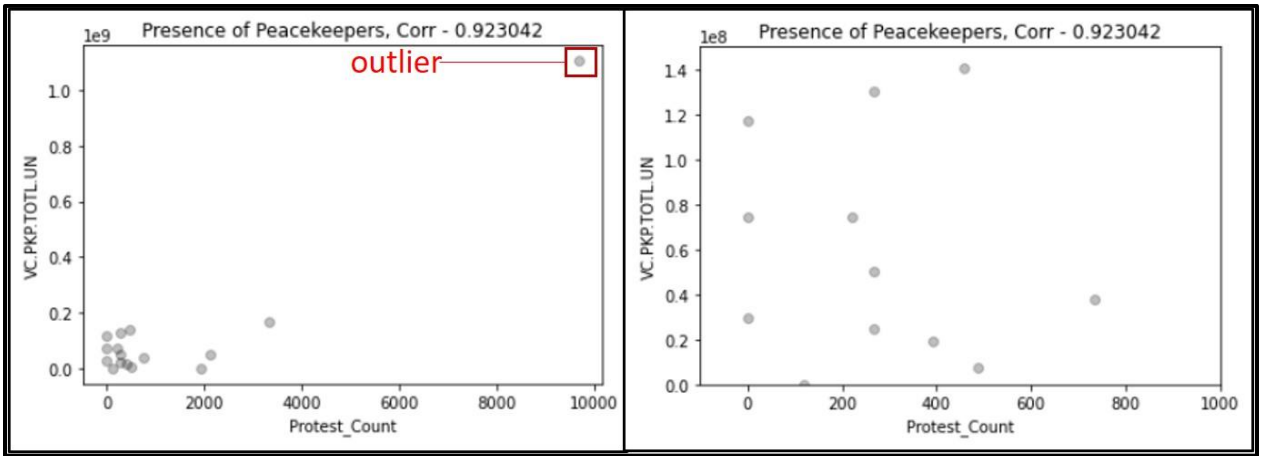

Figure 15: Presence of Peacekeepers vs. Protest Count. Left showing outlier, right zoomed into cluster.

IV. Statistical Modeling

A. Linear Regression

For statistical modeling, this project explored linear regression using Statsmodels API. This package allows estimation by ordinary least squares (OLS), weighted least squares (WLS), generalized least squares (GLS), and feasible generalized least squares with auto correlated AR(p) errors.[8] To accomplish this, the regressors (predictors) chosen were the 10 indicators found using correlation. The target chosen was the protest count and the data used was from 2016. Using the OLS method (Figure 16) produced the results in Figure 17.

```
import statsmodels.api as sm
X = df_predictors[regressors]
Y = target_protests_2016['Protest_Count']

model = sm.OLS(Y,X).fit()
predictions = model.predict(X)
model.summary()
```

Figure 16: Ordinary Least Squares estimation method

OLS Regression Results

Dep. Variable:	Protest_Count	**R-squared (uncentered):**	0.944
Model:	OLS	**Adj. R-squared (uncentered):**	0.872
Method:	Least Squares	**F-statistic:**	13.15
Date:	Fri, 15 Jan 2021	**Prob (F-statistic):**	0.00131
Time:	14:16:23	**Log-Likelihood:**	-125.91
No. Observations:	16	**AIC:**	269.8
Df Residuals:	7	**BIC:**	276.8
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
SP.RUR.TOTL	-0.0001	0.000	-0.822	0.438	-0.000	0.000
SP.URB.TOTL	-9.779e-06	8.28e-06	-1.181	0.276	-2.94e-05	9.79e-06
SH.XPD.CHEX.PC.CD	2.0665	0.867	2.383	0.049	0.016	4.117
VC.PKP.TOTL.UN	0.0003	0.000	0.922	0.387	-0.000	0.001
SH.DTH.1519	0.0477	0.090	0.532	0.611	-0.164	0.259
SH.DTH.1014	0.0201	0.078	0.256	0.805	-0.165	0.205
EN.POP.SLUM.UR.ZS	-1.319e-05	9.61e-05	-0.137	0.895	-0.000	0.000
SP.DYN.CDRT.IN	-16.0350	52.840	-0.303	0.770	-140.982	108.912
SP.POP.TOTL	-0.0001	0.000	-0.882	0.407	-0.000	0.000
SH.DTH.2024	-0.0882	0.085	-1.035	0.335	-0.290	0.113

Omnibus:	2.603	**Durbin-Watson:**	1.571
Prob(Omnibus):	0.272	**Jarque-Bera (JB):**	0.948
Skew:	0.538	**Prob(JB):**	0.622
Kurtosis:	3.515	**Cond. No.**	1.28e+17

Figure 17: Results of OLS regression analysis

[8] (statsmodel.org, 2020)

B. Interpreting Results[9,10]

Based on the **coefficients**, SP.URB.TOTL (urban population) and SP.DYN.CDRT.IN (death rate) seem to be the largest contributors to protest count. As urban population increases, so do the number of protests by a factor two. Additionally as death rate (per 1000 people) decreases, so do protests, both of which intuitively make sense.

Looking at the **P-Values**, SH.XPD.CHEX.PC.CD (current health expenditure per capita) appears to be the only factor that is statistically significant since the P-value is less than 0.05.

The **R-Squared Value** indicates 94.4% of the variation can be explained by the 10 indicators.

Additional results:
Omnibus – a value close to 0 indicates normalcy. A result of 2.603 is high indicating non -linear.
Prob(Omnibus) – a value close to 1 is normal. A result of 0.272 is low and indicates non-linear.
Skew – a value close to zero indicates a normal distribution. A result of 0.538 is relatively high.
Kurtosis – measures curvature of data with higher values indicating tighter clustering of residuals around zero and implies a better model with few outliers. A value of 3.515 is good.
Durbin-Watson – tests for homoscedasticity (consistent variants of errors) and should be between 1 and 2. A result of 1.571 is good.
Jarque-Bera (JB) / Prob(JB) – similar to Omnibus/Prob(Omnibus)
Condition Number – measures the sensitivity of a function's output compared to input. A relatively small number is expected (< 30). A value of 1.28×10^{17} is horrible.

Based on these results, a linear regression model is probably not the most appropriate and a nonlinear model would probably be better. Additionally, finding a better way to interpolate data and having more data points in the model would yield more meaningful results.

V. Conclusion

In the end, this project did not produce in any conclusive results. Although current health expenditures (SH.XPD.CHEX.PC.CD) was statistically significant based on the OLS linear regression model, it may be the result of faulty data practices. A better method for interpolation is definitely one thing that may drive better results. Additional ideas would be to use clustering to find indicators and countries that are better to compare. The methods chosen in this project may have been comparing apples to lawnmowers, so finding better ways to rank countries against one another would probably be helpful. There are various ways that this could be done, quality of life index, national wealth, or regionally may all work. Additionally, finding some way to group by a freedom of press index could be useful, since the ACLED data uses news articles to create their dataset. It's hard to compare two countries when one may be suppressing stories from getting out, thus having a lower tally of reported civil unrest. Although results from this project were not conclusive, it was an excellent data mining and data carpentry exercise.

[9] (Singh, 2019)
[10] (McCarty, 2018)

VI. Way Forward

The next step in this project would be to refine the data and consider better ways to compare the countries as discussed in the previous section. Once that is accomplished, additional statistical modeling could be performed (non-Linear regression, ANOVA, MANOVA) to find a way to better portray the data and predict civil unrest. From here a machine learning model could be built with some of the countries that have more complete data in an attempt to predict those with less complete data. Additionally, the model could be used to predict future countries that have an atmosphere ready for unrest. The final step would be to study social media posts and news articles then perform sentiment analysis to find keywords and triggers that may be able to forecast some kind of protest or riot to occur. Incorporating this into a dashboard that could monitor these things in real time would be the moonshot and could be a very useful tool for various security services and news outlets.

VII. Python Libraries Utilized

- JSON[11]
- Matplotlib[12]
- Numpy[13]
- OS[14]
- Pandas[15]
- PsycoPG2[16]
- Requests[17]
- SQLAlchemy[18]
- Statsmodels[19]
- Time[20]
- XLRD[21]

[11] (Python Software Foundation, 2021)
[12] (Matplotlib Development Team, 2020)
[13] (NumPy, 2020)
[14] (Python Software Foundation, 2021)
[15] (Pandas, 2021)
[16] (The Psycopg Team, 2020)
[17] (Python Software Foundation, 2021)
[18] (Bayer, 2021)
[19] (Statsmodels-Developers, 2020)
[20] (Python Software Foundation, 2021)
[21] (Python Software Foundation, 2021)

References

ACLED. (2021, January 15). *About ACLED*. Retrieved from The Armed Conflict Location and Event Data Project: https://acleddata.com/about-acled/

Bayer, M. (2021). *SQLAlchemy 1.3 Documentation*. Retrieved from SQLAlchemy: https://docs.sqlalchemy.org/en/13/

Matplotlib Development Team. (2020, November 12). *Documentation*. Retrieved from Matplotlib 3.3.3: https://matplotlib.org/3.3.3/contents.html

McCarty, K. (2018, June 18). *Interpretying Results from Linear Regression - Is the data appropriate?* Retrieved from Accelebrate: https://www.accelebrate.com/blog/interpreting-results-from-linear-regression-is-the-data-appropriate

NumPy. (2020). *NumPy Documentation*. Retrieved from NumPy: https://numpy.org/doc/

Pandas. (2021). *pandas.DataFrame.interpolate*. Retrieved from Pandas: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html

Python Software Foundation. (2021, January). *JSON encoder and decoder*. Retrieved from Python 3.9.1 Docs.

Python Software Foundation. (2021, January). *JSON Encoder and Decoder*. Retrieved from Python Documentation: https://docs.python.org/3/library/json.html

Singh, D. (2019, August 1). *Interpreting Data using Statistical Models with Python*. Retrieved from PLURALSIGHT: https://www.pluralsight.com/guides/interpreting-data-using-statistical-models-python

statsmodel.org. (2020, October 29). *Linear Regression*. Retrieved from statsmodels v0.12.1: https://www.statsmodels.org/stable/regression.html

Statsmodels-Developers. (2020, October 29). *API Reference*. Retrieved from statsmodel v0.12.1: https://www.statsmodels.org/stable/api.html

The PostgreSQL Global Development Group. (2021). Retrieved from PostgreSQL.

The Psycopg Team. (2020). *Psycopg 2.8.7 Documentation*. Retrieved from Psycopg.

The World Bank. (2020, December 16). *World Development Indicators*. Retrieved from The World Bank DataBank: https://databank.worldbank.org/source/world-development-indicators