April 1, 2021

This paper uses Google search terms containing offensive language as a proxy measure for racial animus. Following the practice of similar works (Stephens-Davidowitz 2014; Chae, Clouston, Hatzenbuehler, et al. 2015; Chae, Clouston, Martz, et al. 2018; Isoya and Yamada 2021), I use coded language to refer to these terms. The words themselves can be found in Table **??**

1 Introduction

2 Methods

2.1 Tools Used

Surveys of major social science journals routinely fail to reproduce the findings of a plurality or majority of papers from the supplementary code and data provided (Nuijten, Hartgerink, et al. 2015; Nuijten and Polanin 2020; Eubank 2016).

Failures to replicate are often due to coding errors or mistakes in transcribing the results of a calculation into a published manuscript (Eubank 2016, p. 276).

I use Kintr to integrate statistical calculations into the paper, eliminating the possibility of transcription errors (Xie 2014). To ensure that the methods of this paper have been properly implemented and the finding are reproducible, I tested the analysis routines using the *testthat* package in R (Wickham 2011) and the *unittest* module in Python (Van Rossum and Drake 2009). But I won't make you take my word for it – I provide a Docker image with the reproducibility materials to ensure others can replicate the calculations on their own systems (Merkel 2014; Boettiger 2015). The net result is "one-click reproducibility" (Nüst et al. 2020); readers can reproduce this exact paper with the push of a button from the linked materials. [1]

2.2 Preregistration

To avoid the possibility of fitting hypotheses to the data after results are known, I created a preregistration plan of my analysis. The plan can be seen in section **??**.

I have made one significant deviation from the preregistration plan. In my preregistration, I describe a strategy to back out a ratio-level measure of the number of searches for [Word 1] in an area from Google trends data. This strategy is based on a misunderstanding of the format of Google trends data, and does not actually produce the desired measure.

In the analysis I perform, I correct this mistake. I describe the correct scaling procedure in section **??**.

[1] Replication materials available here

Figure 1: Illustration of Scaling Algorithm

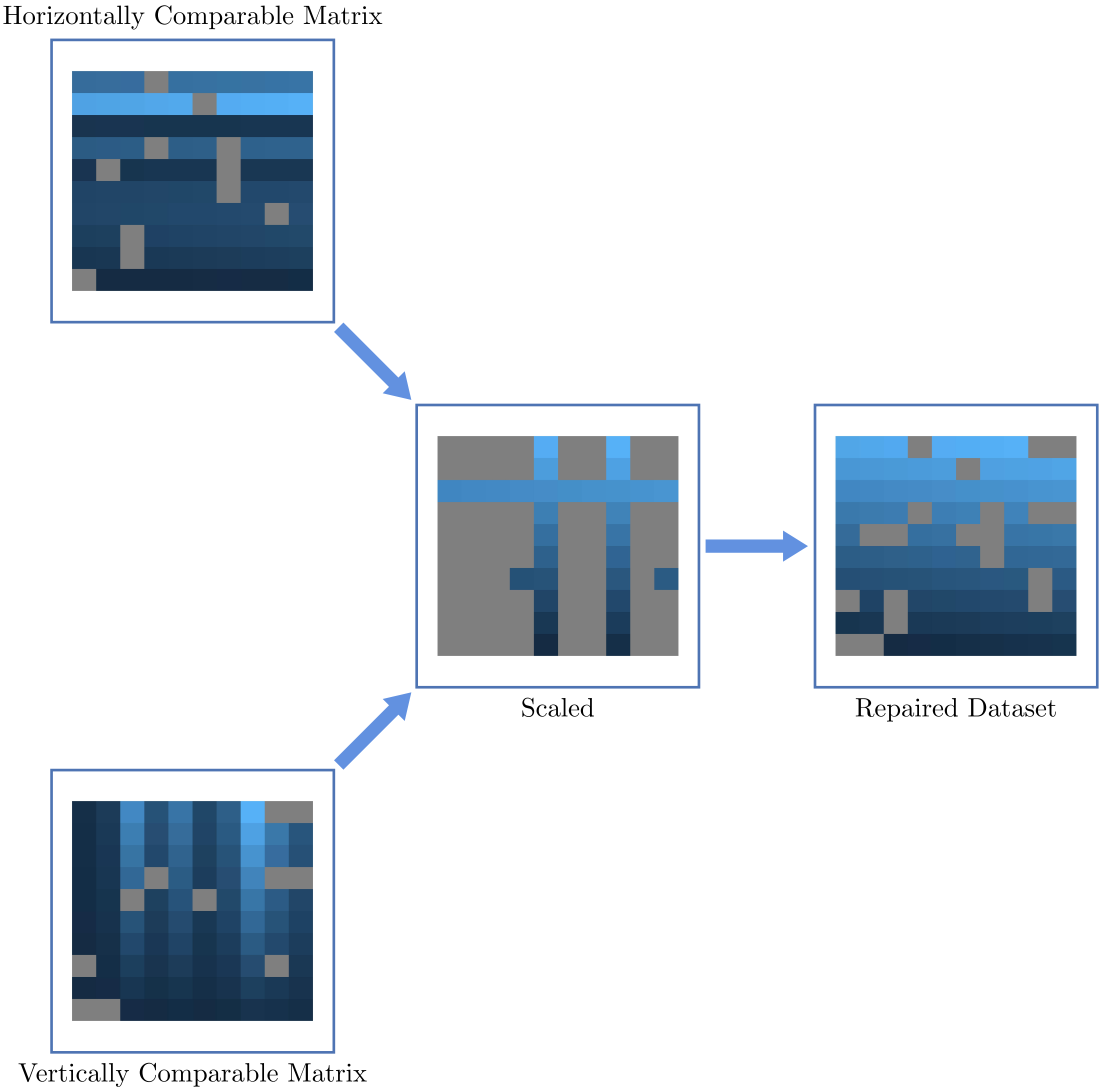


Figure 2: Sinclair News Anchors Reading a "Must-Run" Script (May 2018)

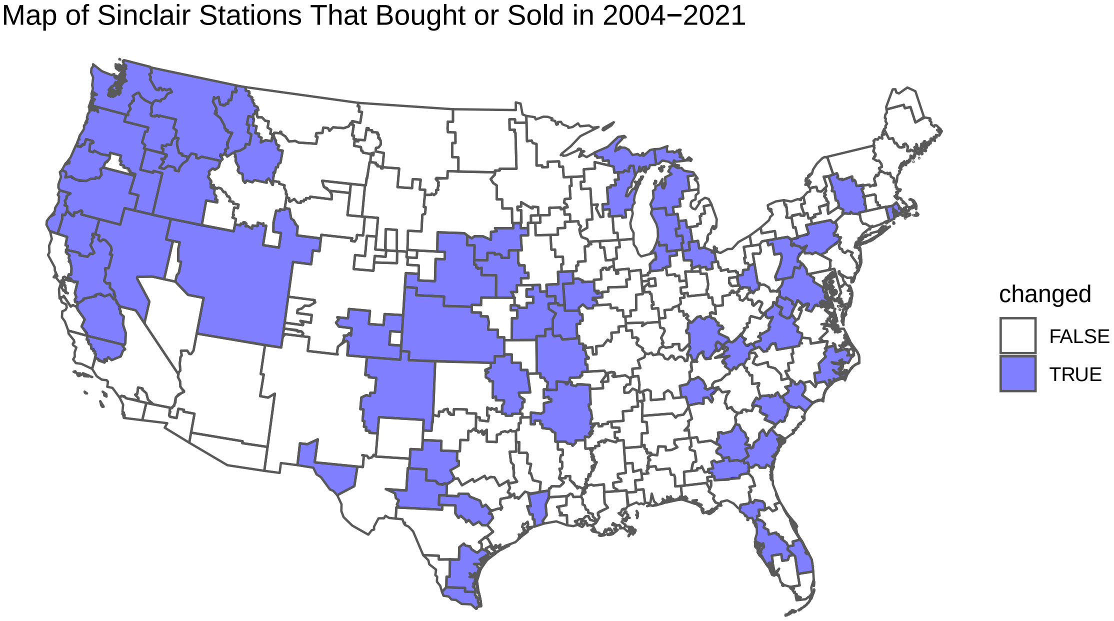
(a) Images of 30 of 210 Sinclair Station Hosts Reading a Must-Run Script (Burke 2018)

(A): But we're concerned about the troubling trend of irresponsible, one sided news stories plaguing our country. The sharing of biased and false news has become all too common on social media.

(B): More alarming, some media outlets publish these same fake stories... stories that just aren't true, without checking facts first.

(A): Unfortunately, some members of the media use their platforms to push their own personal bias and agenda to control 'exactly what people think'...This is extremely dangerous to a democracy.

(b) Transcript of Segment (Cohen 2018)



3 Results

$$\text{Racially Charged Search Rate} = \beta_1(\text{ Sinclair Present }) + \beta_2(\text{ DMA fixed effects }) + \beta_3(\text{ year fixed effects })$$

Table 1: Fixed-Effect Model Results

	Dependent variable:	
	Frequency of Google Searches for [Word 1]	
	(1)	(2)
Sinclair Present	-0.122^{*}	-0.068
	(0.065)	(0.107)
Constant	-0.212	41.959
	(0.234)	(126.128)
Year Fixed Effects	Yes	Yes
Region Fixed Effects	Yes	Yes
Region / Year Fixed Effects	Yes	No
Observations	3,570	3,570
R^2	0.240	0.309
Adjusted R^2	0.188	0.214
Residual Std. Error	0.901 (df = 3343)	0.887 (df = 3134)
F Statistic	4.668^{***} (df = 226; 3343)	3.229^{***} (df = 435; 3134)
Note:		*p<0.1; **p<0.05; ***p<0.01

3.1 Indentification Asssumption

In this section, I test the indefication assumption, the assumption that the treated and control units would have the outcomes if the treatment were absent.

$$\text{Racially Charged Search Rate} = \beta_1(\text{ Sinclair Present }) + \beta_2(\text{ DMA fixed effects }) + \beta_3(\text{ year fixed effects }) + \beta_4(\text{Year / D}$$

Figure 3: Fixed-Effects Estimates of the effect of Sinclair Acquisition on Rate of Racially Charged Google Searches

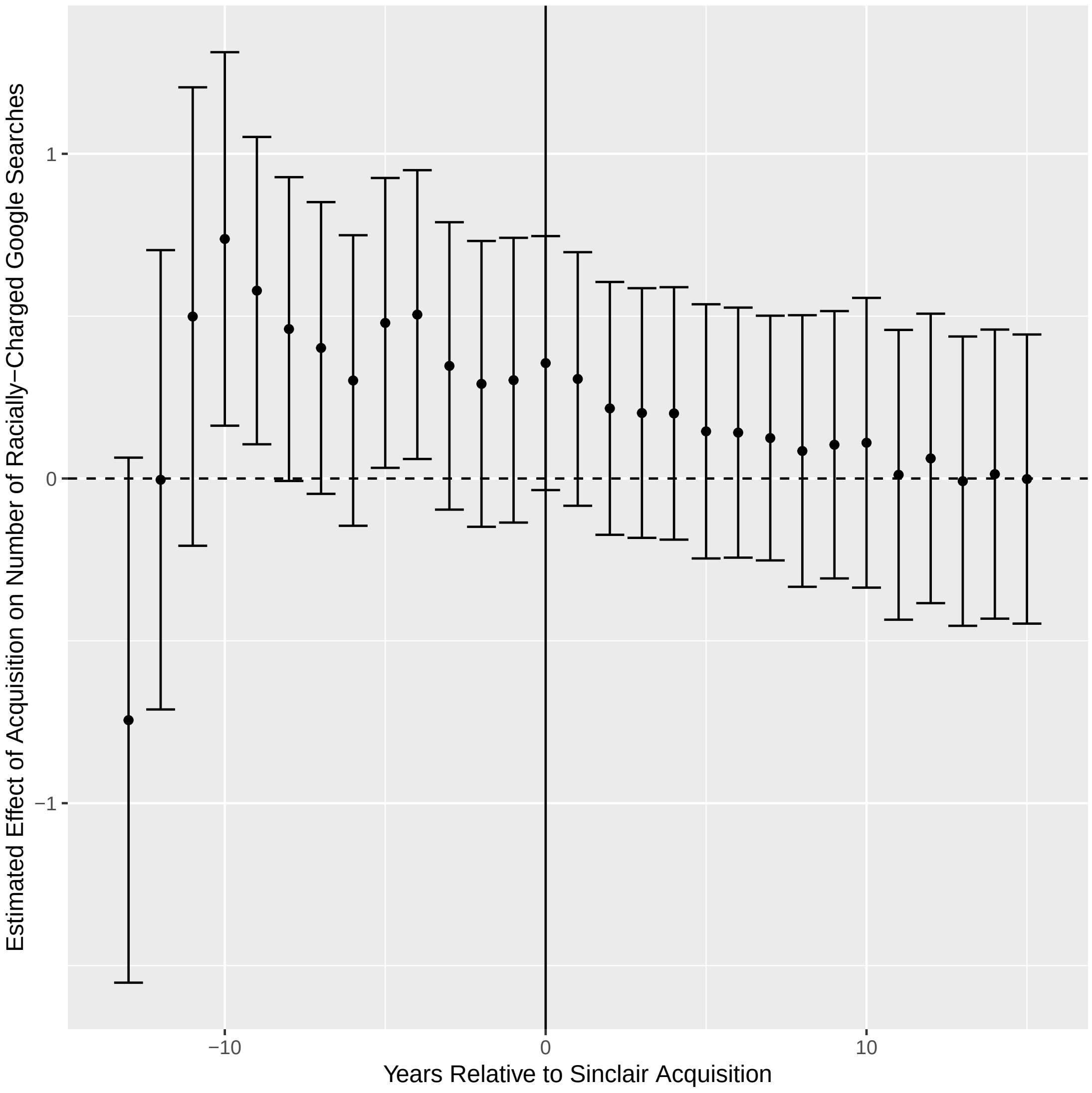
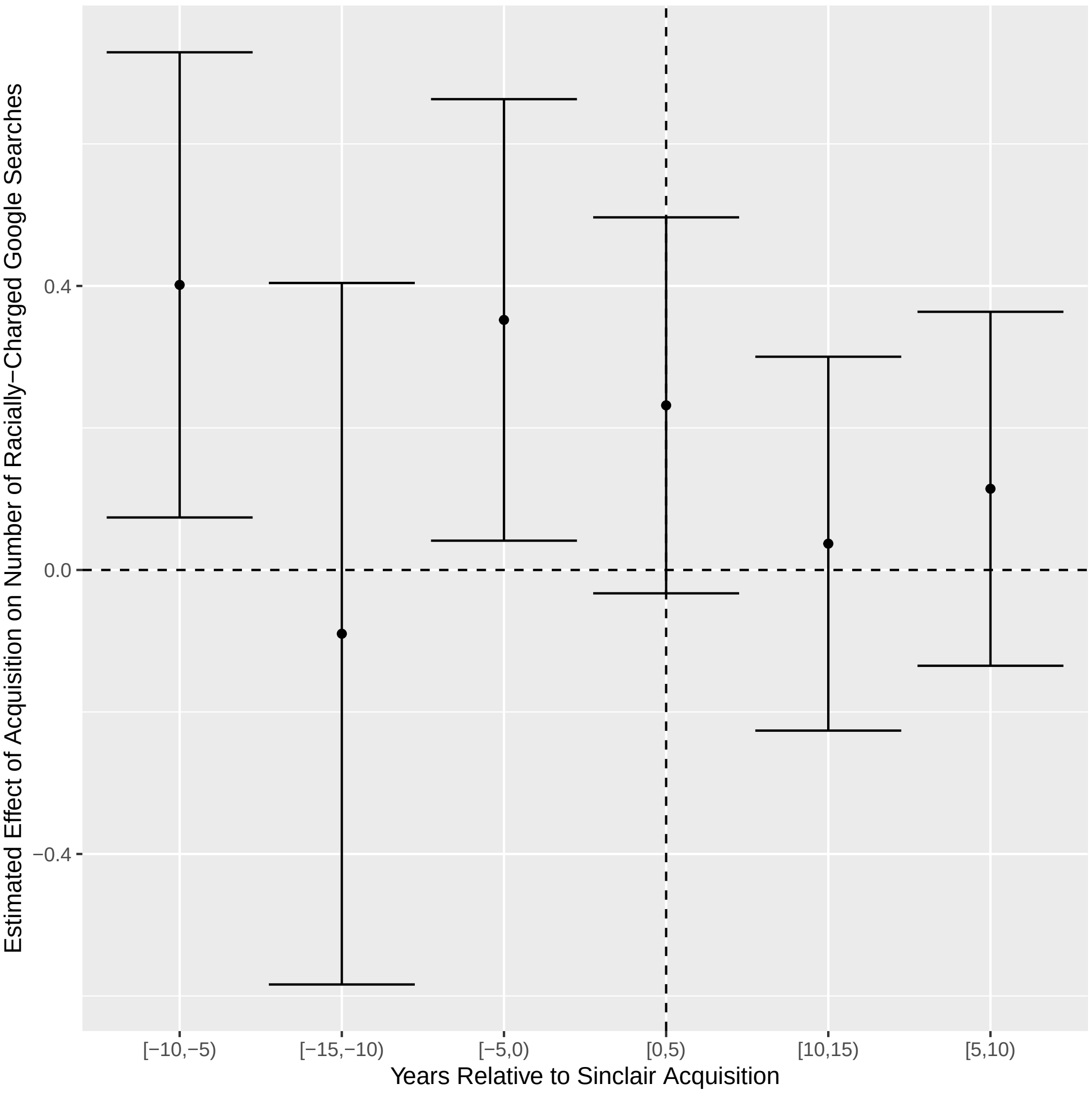


Figure 4: Fixed-Effects Estimates of the effect of Sinclair Acquisition on Rate of Racially Charged Google Searches [Larger Bins]



Code	Word
Word 1	nigger

A Codings for Offensive Words

References

Boettiger, Carl (Jan. 2015). “An introduction to Docker for reproducible research”. In: *ACM SIGOPS Operating Systems Review* 49.1, pp. 71–79.

Burke, Timothy (2018). *How I Made A Dumb Video Making Fun Of Sinclair Broadcasting And Somehow Started A Media War.* URL: https://deadspin.com/how-i-made-a-dumb-video-making-fun-of-sinclair-broadcas-1825106452 (visited on 03/23/2020).

Chae, David H., Sean Clouston, Mark L. Hatzenbuehler, et al. (Apr. 2015). “Association between an Internet-Based Measure of Area Racism and Black Mortality”. In: *PLOS ONE* 10.4. Ed. by Hajo Zeeb, e0122963.

Chae, David H., Sean Clouston, Connor D. Martz, et al. (Feb. 2018). “Area racism and birth outcomes among Blacks in the United States”. In: *Social Science & Medicine* 199, pp. 49–55.

Cohen, Stephen (2018). *KOMO attacks 'biased and false news' in Sinclair-written promos.* URL: https://www.seattlepi.com/seattlenews/article/KOMO-fake-news-Sinclair-promos-12792032.php (visited on 03/23/2020).

Eubank, Nicholas (Apr. 2016). “Lessons from a Decade of Replications at the Quarterly Journal of Political Science”. In: *PS: Political Science & Politics* 49.02, pp. 273–276.

Isoya, Hidehito and Hiroyuki Yamada (Mar. 2021). “Hidden internet-based measure of discrimination and mental health of Black people”. In: *Social Science & Medicine* 273, p. 113776.

Merkel, Dirk (2014). “Docker: lightweight linux containers for consistent development and deployment”. In: *Linux journal* 2014.239, p. 2.

Nuijten, Michèle B., Chris H. J. Hartgerink, et al. (Oct. 2015). “The prevalence of statistical reporting errors in psychology (1985–2013)”. In: *Behavior Research Methods* 48.4, pp. 1205–1226.

Nuijten, Michèle B. and Joshua R. Polanin (Apr. 2020). ““ statcheck ”: Automatically detect statistical reporting inconsistencies to increase reproducibility of meta-analyses”. In: *Research Synthesis Methods.*

Nüst, Daniel et al. (Nov. 2020). “Ten simple rules for writing Dockerfiles for reproducible data science”. In: *PLOS Computational Biology* 16.11. Ed. by Scott Markel, e1008316.

Stephens-Davidowitz, Seth (Oct. 2014). “The cost of racial animus on a black candidate: Evidence using Google search data”. In: *Journal of Public Economics* 118, pp. 26–40.

Van Rossum, Guido and Fred L. Drake (2009). *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace.

Wickham, Hadley (2011). “testthat: Get Started with Testing”. In: *The R Journal* 3, pp. 5–10.

Xie, Yihui (2014). “knitr: A Comprehensive Tool for Reproducible Research in R”. In: *Implementing Reproducible Computational Research.* Ed. by Victoria Stodden, Friedrich Leisch, and Roger D. Peng. ISBN 978-1466561595. Chapman and Hall/CRC.

B Code In This Document

C Web Scraping Code

C.1 Master Web Scraping Script

```python
#!/usr/bin/python
import glob
import os
import pickle as pkl
from between_regions import between_reigion_many
from in_region import in_region
from utils import itr_split_overlap
import tqdm


def extract_keywords(filename):
    with open(filename, "r") as f:
        keywords = f.read().splitlines()
    return keywords


def get_basename(filename):
    name_w_ext = os.path.basename(filename)
    basename = os.path.splitext(name_w_ext)[0]
    return (basename)


def run_keywords(name, keywords):
    regions = pkl.load(open("data/dma_abbreviations.pkl", "rb"))
    filename = f"data/google_trends_data/{name}_time_serires.csv"
    with open(filename, "w") as f:
        f.write("row,date,score,ispartial,code,term\n")
        for keyword in keywords:
            for region in tqdm.tqdm(regions):
                df = in_region(keyword, region, True, timeframe="all")

                if df is not None:
                    df.to_csv(f, header=False)

    filename = f"data/google_trends_data/{name}_between_regions.csv"
    with open(filename, "w") as f:
        df = between_reigion_many(itr_split_overlap(keywords, 5, 1),
                                  censor=True,
                                  timeframe="all",
                                  geo="US",
                                  gprop="")
        df.to_csv(f, header=True)


if __name__ == "__main__":
    keyword_files = sorted(glob.glob("data/keywords/*.csv"))
    base_names = [get_basename(filename) for filename in keyword_files]
    keywords = [extract_keywords(filename) for filename in keyword_files]
    first = (list(zip(base_names, keywords)))[0]
    run_keywords(first[0], first[1])
```

C.2 Collecting Data for Between-Region Comparisons

```python
#!/usr/bin/env python
from pytrends.request import TrendReq
from utils import censor_string
import pandas as pd


def between_region(query, censor, **kwargs):
    pytrends = TrendReq(hl='en-US', tz=360)
    pytrends.build_payload(query, cat=0, **kwargs)
    df = pytrends.interest_by_region(resolution='DMA',
                                     inc_low_vol=True,
                                     inc_geo_code=True)
    df = df.set_index("geoCode")
    df.index.name = 'code'
    if censor:
        df = df.rename(censor_string, axis="columns")
    return (df)


def between_reigion_many(iterable, censor, **kwargs):
    iterable = iter(iterable)
    chunk1 = next(iterable)
    df1 = between_region(chunk1, censor, **kwargs)

    for chunk in iterable:
        chunk2 = chunk
        shared = list(set(chunk1) & set(chunk2))[0]
        if censor:
            shared = censor_string(shared)
        df2 = between_region(chunk2, censor, **kwargs)

        mean1 = df1[shared].mean()
        mean2 = df2[shared].mean()
        normaliation_factor = mean1 / mean2

        df2 = df2 * normaliation_factor
        df2 = df2.drop(columns=[shared])

        df1 = df1.join(df2)

        chunk1 = chunk2

    return (df1)


def create_v_df(term, year):
    timeframe = f"{year}-01-01 {year}-12-31"
    df = between_region([term], False, timeframe=timeframe, geo="US")
    df = df.rename(columns={term: str(year)})
    return df


if __name__ == "__main__":
    dfs = tuple(create_v_df("hello", year) for year in range(2004, 2021))
    df_c = pd.concat(dfs, axis=1)

    df_c.to_parquet("v_matrix.parquet")
```

C.3 Collecting Within-Region Data

```python
#!/usr/bin/python
from pytrends.request import TrendReq
from utils import censor_string
import time
import pandas as pd
import re
import pickle as pkl


def in_region(query, region, censor, **kwargs):
    """ Returns a set of data showing the popularity of a search term over time

    :query: The search term that time-series data is collected for
    :region: The geographic region to retrieve time-series data for.
    :censor: boolean should the search terms be censored?
    :**kwargs: kwargs to be passed to pytrends.build_payload()
    :returns: DataFrame giving time-series data for the popularity of a
        search term in a given region

    """
    pytrends = TrendReq(hl='en-US', tz=360)
    try:
        pytrends.build_payload(kw_list=[query], geo=f"{region}", **kwargs)
        df = pytrends.interest_over_time()
        if df.empty:

            df = pd.DataFrame()
            df['date'] = pd.period_range(start='2004-01-01',
                                         end='2021-01-01',
                                         freq='M').to_timestamp()
            df['n'] = 0
            df['ispartial'] = pd.Series([True]).bool()

        else:

            df.columns = ["n", "ispartial"]
            df.index.name = 'date'
            df.reset_index(inplace=True)

        df["query"] = query
        df['code'] = re.findall("\d+", region)[0]
        if censor:
            df["query"] = df["query"].apply(censor_string)

        return df
    except:
        if censor:
            print(f"Rate error: {censor_string(query)} in {region}")
        else:
            print(f"Rate error: {query} in {region}")

        time.sleep(60)
        return in_region(query, region, censor, **kwargs)


def to_wide(df):
    """TODO: Turns time-series search popularity data into a 'wide' dataframe to be used in scaling

    :df: 'long' dataframe of search data, as from in_region()
    :returns: 'wide' DataFrame of search data, averaged by year

    """
    print(df['date'])
    df['year'] = pd.DatetimeIndex(df['date']).year
    df['year'] = df['year'].apply(str)
    df = df.groupby(['year', 'code'])["n"].mean()
    df = df.unstack(level=0)
    return (df)


# with open("data/dma_abbreviations.pkl", "rb") as f:
#     dmas = pkl.load(f)

# dmas = dmas

# in_region_dfs = tuple(
#     in_region("economist", dma, True, timeframe="all") for dma in dmas)
# wide_dfs = map(to_wide, in_region_dfs)
# h_df = pd.concat(wide_dfs).sort_index()
# print(h_df)
```

C.4 Utility Functions

```python
#!/usr/bin/python
import itertools as it
import requests
import random


# returns mean of nonzero values in iterator
def mean_nonzero(iterator):
    nonzero = tuple(filter(lambda x: x != 0, iterator))
    if len(nonzero) == 0:
        return None
    elif None in nonzero:
        return None
    else:
        return sum(nonzero) / len(nonzero)


# Returns list of N random words from MIT dictionary
def random_words(n):
    word_site = "https://www.mit.edu/~ecprice/wordlist.10000"

    response = requests.get(word_site)
    words = response.text.splitlines()

    keywords = random.sample(words, n)

    return (keywords)


# Tests if computer is connected to internet (used in tests)
def connected():
    url = "http://google.com"
    timeout = 5
    try:
        requests.get(url, timeout=timeout)
        return (True)
    except (requests.ConnectionError, requests.Timeout):
        return (False)


# Censors strings so that senstive words aren't uploaded to github / used in
# scripts
def censor_string(string):
    return (string[0] + "_" * (len(string) - 2) + string[-1])


# credit to Ilja Everila for this implimentation
# https://stackoverflow.com/questions/48381870/a-better-way-to-split-a-sequence-in-chunks-with-overlaps
def itr_split_overlap(iterable, size, overlap):

    if overlap >= size:
        raise ValueError("overlap must be smaller than size")

    itr = iter(iterable)

    next_ = tuple(it.islice(itr, size))

    yield next_

    prev = next_[-overlap:] if overlap else ()

    while True:
        chunk = tuple(it.islice(itr, size - overlap))

        if not chunk:
            break

        next_ = (*prev, *chunk)
        yield next_

        if overlap:
            prev = next_[-overlap:]
```

D Analysis Code

```
#!/usr/bin/Rscript
# Load Libraries for Analysis
library(tidyverse)
library(lubridate)
library(broom)
# Load utility functions
source("utils.R")

search_data <- read_csv("data/google_trends_data/word_1.csv") %>%
  pivot_longer(-code, names_to = "year", values_to = "word1") %>%
  mutate_all(as.numeric)

stopifnot(nrow(search_data) == 3570)
stopifnot(all(!is.na(search_data)))

write_csv(search_data, "../data/google_trends/word1.csv")

sinclair_data <- read_csv("../data/clean_sinclair_data.csv")
stopifnot(nrow(sinclair_data) == 3570)
stopifnot(all(!is.na(sinclair_data)))

```

```r
# search_data <- search_data %>%
#   dplyr::select(-data) %>%
#   distinct(term, code, year, .keep_all = TRUE) %>%
#   pivot_wider(names_from = term, values_from = score) %>%
#   mutate(overall_score = rowSums(across(everything()), na.rm = T))
dma_names <- read_csv("data/dma_list.csv")
stopifnot(nrow(dma_names) == 210)

full_data <- search_data %>%
  right_join(sinclair_data) %>%
  full_join(dma_names) %>%
  filter(year != 2021) %>%
  group_by(code) %>%
  mutate(years_before = years_before(sinclair_present)) %>%
  ungroup() %>%
  mutate(sword1 = (word1-mean(word1))/sd(word1)) %>%
  mutate(years_before = relevel(as.factor(years_before),"-99"))

stopifnot(nrow(full_data) == 3570)
stopifnot(all(!is.na(full_data)))

write_csv(full_data, "../data/full_data.csv")

model_1 <- lm(sword1 ~ as.factor(year) + as.factor(code) + sinclair_present, data = full_data)
model_1 %>% summary()

model_2 <- lm(sword1 ~ as.factor(year) + as.factor(code) + year:as.factor(code)+ sinclair_present, data = full_data)
model_2 %>% summary()

model_3 <- lm(sword1 ~ as.factor(year) + as.factor(code) + as.factor(years_before), data = full_data)
model_3 %>% summary()

model_3 %>%
  tidy() %>%
  filter(grepl("years_before", term)) %>%
  mutate(term = as.numeric(gsub("[^0-9\\-]+", "", term))) %>%
  ggplot(aes(x = term, y = estimate)) +
  geom_point() +
  geom_errorbar(aes(ymin = estimate - 1.96 * std.error, ymax = estimate + 1.96 * std.error)) +
  geom_hline(aes(yintercept=0), linetype=2) +
  geom_vline(aes(xintercept=0))
```

D.1 Utility Functions

```r
#!/usr/bin/Rscript
censor_string <- function(string) {
  substr(string, 2, nchar(string) - 1) <- paste0(rep("_", nchar(string) - 2), collapse = "")
  names(string) <- NULL
  return(string)
}
censor_string <- Vectorize(censor_string, USE.NAMES=F)

years_before <- function(bool) {
  if (any(bool)) {
    out <- numeric(length = length(bool))
    start <- min(which(bool))

    before <- (seq(start, 1) - 1) * -1
    if (start != length(bool)) {
      after <- seq(1, length(bool) - start)
    } else {
      after <- c()
    }


    out <- c(before, after)
  } else {
    out <- rep(-99, length(bool))
  }

  return(out)
}
```

E Unit tests

E.1 For Python Code

```python
#!/usr/bin/python
import unittest
import pandas as pd
from between_regions import between_region
from utils import connected


@unittest.skipIf(not connected(), "not connected to the internet")
class TestBetweenRegion(unittest.TestCase):
    def test_between_region_uncensored(self):

        result_1 = between_region(["socks"],
                                  censor=False,
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")

        result_2 = between_region(["socks", "shoe", "fish"],
                                  censor=False,
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")

        expected_1 = pd.read_parquet(
            "tests/test_data/between_region_1_uc.parquet")
        expected_2 = pd.read_parquet(
            "tests/test_data/between_region_2_uc.parquet")

        self.assertTrue(expected_1.equals(result_1))
        self.assertTrue(expected_2.equals(result_2))

    def test_between_region_censored(self):
        result_1 = between_region(["socks"],
                                  censor=True,
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")

        result_2 = between_region(["socks", "shoe", "fish"],
                                  censor=True,
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")

        expected_1 = pd.read_parquet(
            "tests/test_data/between_region_1_c.parquet")
        expected_2 = pd.read_parquet(
            "tests/test_data/between_region_2_c.parquet")

        self.assertTrue(expected_1.equals(result_1))
        self.assertTrue(expected_2.equals(result_2))
```

```python
#!/usr/bin/python
import unittest
from utils import itr_split_overlap, censor_string


class TestItrSplitOverlap(unittest.TestCase):
    def test_itr(self):
        test_list = ["one", "two", "three", "four", "five"]

        expected_result = [('one', 'two'), ('two', 'three'), ('three', 'four'),
                           ('four', 'five')]
        self.assertEqual(expected_result,
                         list(itr_split_overlap(test_list, 2, 1)))

        expected_result = [('one', 'two', 'three', 'four'),
                           ('two', 'three', 'four', 'five')]
        self.assertEqual(expected_result,
                         list(itr_split_overlap(test_list, 4, 3)))

    def test_exceptions(self):
        with self.assertRaises(ValueError):
            test_list = ["one", "two", "three", "four", "five"]

            list(itr_split_overlap(test_list, 2, 3))


class CensorString(unittest.TestCase):
    def test_censor(self):
        self.assertEqual("h_y", censor_string("hey"))
        self.assertEqual("f____r", censor_string("fender"))
        self.assertEqual("fr", censor_string("fr"))
```

E.2 For R Code

```
library(testthat)
source("../../utils.R")

test_that("censor string works", {
      expect_equal(censor_string("hey"),"h_y")
      expect_equal(censor_string("hy"),"hy")
      expect_equal(censor_string("watermelon"),"w________n")
      expect_equal(censor_string("fantastic"),"f_______c")
})

test_that("censor string vectorized correctly", {
      expect_equal(censor_string(c("hello","there")),c("h___o","t___e"))
      expect_equal(censor_string(c("watermelon","paper")),c("w________n","p___r"))
})

test_that("years before works", {
  expect_equal(years_before(c(F, F, F, T, T)), c(-3, -2, -1, 0, 1))
  expect_equal(years_before(c(F, F, F, F, T)), c(-4, -3, -2, -1, 0))
  expect_equal(years_before(c(T, T, T, T, T)), c(0, 1, 2, 3, 4))
})
```