March 28, 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 published articles, even when using data provided with the paper (Nuijten, Hartgerink, et al. 2015; Nuijten and Polanin 2020; Eubank 2016). In many cases, failures to replicate are due to coding errors or mistakes in transcribing the results of a calculation into the published manuscript (Eubank 2016).

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). I use Kintr to integrate statistical calculations into the paper, eliminating the possibility of transcription errors (Xie 2014).

Finally, I provide a Docker image (Merkel 2014) with the reproducibility materials to ensure others can replicate the figures in this paper on their own systems (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 on their own systems from the reproducibility materials.

Figure 1: Sinclair News Anchors Reading a “Must-Run” Script (May 2018)

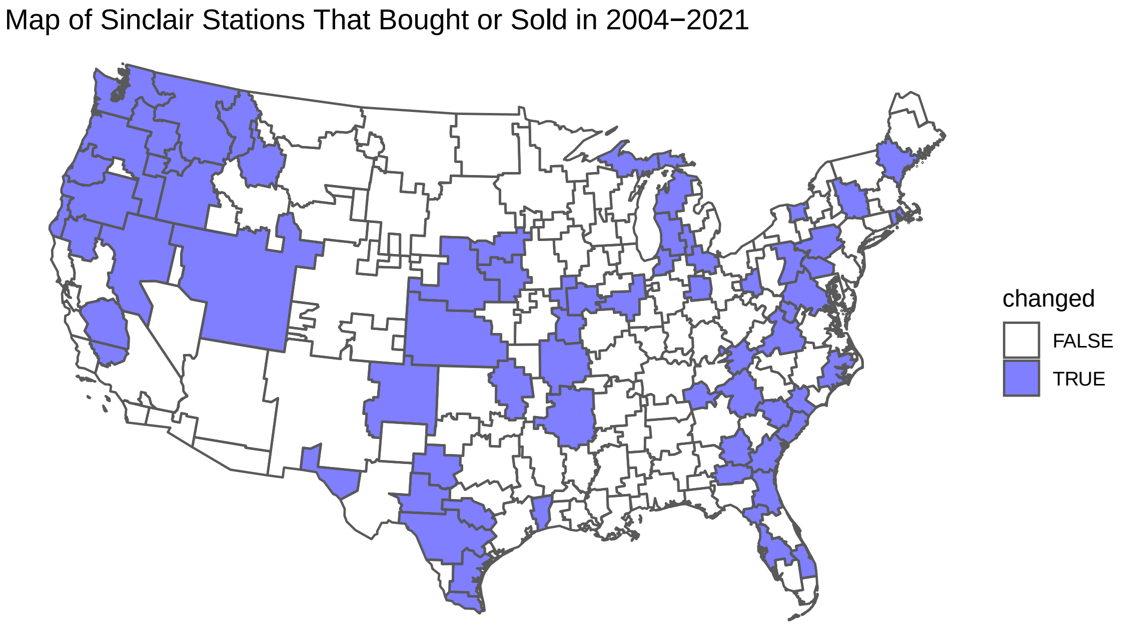
(a) Images from 30 of 210 Stations Reading the Script (Burke 2018)

**(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)



## 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.

# A Codings for Offensive Words

| Code | Word |
|---|---|
| Word 1 | nigger |

# B Code In This Document

```r
## ----echo=FALSE, warn=FALSE, include=FALSE------------------------------------
# Set options to knit this document, load libraries
knitr::opts_chunk$set(out.width="100%", message=F, cache=T,echo=F, warn=F, include=F)
library(tidyverse)
library(sf)


## -----------------------------------------------------------------------------
# Load and create data to show markets Sinclair has expanded to / moved out of
sinclair_expansions <- read_csv("../data/clean_sinclair_data.csv") %>%
    nest(-code) %>%
    mutate(
           any_true =  data %>% map_lgl(~any(.$sinclair_present)),
           any_false =  data %>% map_lgl(~any(!.$sinclair_present)),
           changed = any_true & any_false
   )

dma_boundaries <- st_read("../data/dma_boundaries/dma_boundary.shp")
dma_boundaries <- merge(dma_boundaries, sinclair_expansions, by.x="dma0", by.y="code")


## ----include = T--------------------------------------------------------------
# Plot markets Sinclair has expanded to / moved out of
ggplot() +
    geom_sf(data=dma_boundaries, aes(fill=changed), alpha=.5) +
    scale_fill_manual(values  = c(NA, "blue")) +
    ggtitle("Map of Sinclair Stations That Bought or Sold in 2004-2021") +
    theme_void()


## ----echo=FALSE, include=T----------------------------------------------------
# Pull R code out of this document to put into the appendix
code <- knitr::purl("diss.Rnw", quiet=T)
```

# 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 get_region_trend
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("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 = get_region_trend(keyword, region, True)

                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("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)))[1]
    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


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)
```

## C.3 Collecting Within-Region Data

```python
#!/usr/bin/python
from pytrends.request import TrendReq
from utils import censor_string
import time

def get_region_trend(query, region, censor):
    pytrends = TrendReq(hl='en-US', tz=360)
    try:
        pytrends.build_payload(kw_list=[query],
                               geo=f"{region}",
                               timeframe="all")
        df = pytrends.interest_over_time()
        if not df.empty:
            df.columns = ["n", "ispartial"]
            df.index.name = 'date'
            df.reset_index(inplace=True)
            df["region"] = region

            df["query"] = query

            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)
        get_region_trend(query, region, censor)
```

## C.4 Utility Functions

```python
import itertools as it
import requests
import random


# 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

```
# Load Libraries for Analysis
library(tidyverse)
library(lubridate)
library(broom)

# Load utility functions
source("utils.R")

# Creat tidy dataframe of years / markets in which sinclair was present
sinclair_present <- tibble(filename = list.files(path = "../data/sinclair_data/", full.names = T, pattern = "*.csv")) %>%
  mutate(
    year = str_extract(filename, "[0-9]+") %>% as.double(),
    data = filename %>% map(~ read_csv(.x, locale = locale(encoding = "UTF-8"))),
    data = map(data, ~ rename_with(.x, ~ gsub("\n", "", .x))),
    data = map(data, ~ .x %>% rename_with(tolower)),
    data = map(data, ~ .x %>% dplyr::select(market))
  ) %>%
  unnest(data) %>%
  select(-filename) %>%
  mutate(
    sinclair_present = T,
    market = gsub("\n", "", market)
  )

# Loads in Traslation dictionary of market names in the Sinclair dataset to DMA
  # codes
sinclair_codes <- read_csv("../data/sinclair_names.csv")

# Loads in Traslation dictionary of DMA codes to Standardized names
dma_names <- read_csv("../data/dma_list.csv")

# Adds codes to sinclair present datset
sinclair_present <- sinclair_present %>%
  full_join(sinclair_codes) %>%
  dplyr::select(-market)

# Fills in data to include stations where sinclair was not present
sinclair_data <- sinclair_present %>%
  complete(code = dma_names$code, year = 2005:2020) %>%
  arrange(code) %>%
  replace_na(list(sinclair_present = F)) %>%
  left_join(dma_names)

write_csv(sinclair_data, "../data/clean_sinclair_data.csv")

sinclair_data <- read_csv("../data/clean_sinclair_data.csv")

# Not done annotating / updating after here
################################################################################
between_data <- read_csv("../data/google_trends_data/set_1_between_regions.csv") %>%
  pivot_longer(-code, names_to = "term", values_to = "overall") %>%
  mutate(term = censor_string(term))

search_data <- read_csv("../data/google_trends_data/set_1_time_serires.csv") %>%
  mutate(year = year(date)) %>%
  mutate(term = term) %>%
  group_by(year, code, term) %>%
  nest() %>%
  mutate(
    score = data %>% map_dbl(~ mean(.x$score, na.rm = T)),
    code = str_extract(code, "[0-9]+") %>% as.numeric()
  )

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))


full_data <- search_data %>%
  right_join(sinclair_data) %>%
  filter(year != 2021) %>%
  group_by(market) %>%
  mutate(years_before = years_before(sinclair_present))


full_data %>%
  ggplot(aes(x = years_before, y = n____r, group = market)) +
  geom_line()


full_data %>%
    nest(-code) %>%
    mutate(
           any_true =  data %>% map_lgl(~any(.$sinclair_present)),
           any_false =  data %>% map_lgl(~any(!.$sinclair_present)),
           changed = any_true & any_false
    ) %>%
    unnest() %>%
  ggplot(aes(x = year, y = overall_score, col = sinclair_present, group = market)) +
  scale_color_manual(values = c("grey", "black")) +
  geom_line() +
  facet_wrap(~market) +
  theme(
    plot.background = element_rect(fill = "transparent", colour = NA),
    panel.border = element_rect(colour = "black", fill = NA, size = 0.1),
    panel.background = element_blank(),
    legend.key = element_rect(fill = "transparent", colour = NA),
    legend.background = element_rect(fill = "transparent", colour = NA),
    legend.position = "bottom",
    panel.grid = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major = element_blank()
  )

lm(overall_score ~ as.factor(year) + as.factor(code) + sinclair_present, data = full_data)
    %>%
  summary()

lm(overall_score ~ as.factor(year) + as.factor(code) + sinclair_present + year:as.factor(code
    ), data = full_data) %>%
  summary()

lm(overall_score ~ as.factor(year) + as.factor(code) + as.factor(years_before), data = full_
    data) %>%
  tidy() %>%
  filter(grepl("years_before", term)) %>%
  mutate(years_before = as.numeric(str_extract(term, "-?[0-9]+"))) %>%
  ggplot(aes(x = years_before, y = estimate)) +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(x = years_before, ymin = estimate - 1.96 * std.error, ymax = estimate +
    1.96 * std.error)) +
  geom_hline(yintercept = 0, linetype = 2, col = "grey") +
  geom_vline(xintercept = 0, linetype = 1)
```

## D.1 Utility Functions

```
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) & 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

```
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(self):

        result_1 = between_region(["socks"],
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")
        result_2 = between_region(["socks", "shoes", "fish"],
                                  timeframe="2016-12-14 2017-01-25",
                                  geo="US",
                                  gprop="")

        expected_1 = pd.read_parquet("tests/test_data/between_region_1.parquet")
        expected_2 = pd.read_parquet("tests/test_data/between_region_2.parquet")

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

```
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))
})
```