Predicting Demonstration Size on Salience and Demographic Characteristics

GW DATS 6103: Introduction to Data Mining Final Project

Alexander D. Silberman

2025-12-13

Table of contents

Introduction . . . 2
Data Acquisition and Wrangling . . . 3
Results Dataset: Crowd Counting Consortium Phase 1 (2017-2020) . . . 3
American Community Survey (ACS) 1-Year Estimates . . . 11
Most Important Problem Dataset (MIPD), Second Release (2024) . . . 15
Merge the Datasets . . . 18
Modeling . . . 25
Generalized Linear Models (GLMs) . . . 27
Tree . . . 34
Random Forest . . . 36
K Nearest Neighbors (KNN) . . . 40
Model Evaluation . . . 41
Classifier Evaluation . . . 41
Regressor Evaluation . . . 47
Residuals . . . 47
Coefficients of Determination . . . 52
Conclusions . . . 52
Next Steps . . . 53
References . . . 53


```
# import cuml

# %load_ext cuml.accel

import numpy as np
```

```
import pandas as pd
pd.set_option('display.max_colwidth', 1000)

import matplotlib.pyplot as plt
import seaborn as sns
import re

# from census import Census
# from us import states

import censusdis.data as ced
from censusdis.datasets import ACS1_PROFILE
# from censusdis import states

from statsmodels.formula.api import ols # for linear regression

from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.linear_model import LinearRegression, LogisticRegression, LassoCV
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.decomposition import PCA
from sklearn.metrics import classification_report, roc_curve, auc

seed = 42
```

Introduction

In recent years, demonstrations in the US have grown in both number and scale (*Crowd Counting Consortium*, 2024). Being able to predict the scale of future demonstrations would allow for a more efficient distribution of resources and would enable protesters, government institutions, first responders, and news organizations—among others—to better prepare for what is to come.

In an analysis of Chilean demonstrations between 2000 and 2012, Somma & Medel found that demonstrations targeting broader issues had larger crowd sizes (2018). In other words, demonstrations surrounding issues that larger portions of the population care about tend to be larger in scale. An opposing arrow of causality was drawn by Carey, Jr. et al. (2014), which pointed to a rise in the salience of immigration among U.S. Latinos following the 2006 immigration protests. In either case, there is some evidence to suggest that increased salience corresponds with increased crowd sizes, and may thereby be useful in predicting such.

In general, however, in spite of the importance of this topic, little work has been done on predicting the size of demonstrations, as noted by Somma & Medel (2018, pp. 2–3).

The SMART question at the heart of this analysis is as follows: is it possible to predict the size of crowds at demonstrations—as measured in either raw number or magnitude—for the years 2017 and 2018 based on salience of the issue(s) at hand, as indicated by the percent of the population who labeled that category Most Important Problem that year, and demographic data at the county level for that year, as sourced from the American Community Survey (ACS) 1-year estimates?

Data Acquisition and Wrangling

Results Dataset: Crowd Counting Consortium Phase 1 (2017-2020)

The dataset which originates the target variables of this study is that of the Crowd Counting Consortium (CCC) Phase 1 (2017-2020). This dataset, published by Harvard Kennedy School's Ash Center for Democratic Governance and Innovation and the University of Connecticut, derives information on gatherings from news articles, recording high, low, and mean estimates for crowd size, accounting for vagaries in descriptions, as well as their claimed purpose(s) (Chenoweth et al., 2025). Other information is also collected, such as whether the events resulted in any arrests and the actors forming such crowds, but those are irrelevant for the purposes of this study.

```
crowd_data_orig = pd.read_csv("C:/Users/alexa/Code/GW DATS/6103 12 Intro to Data Mining/Final
crowd_data_orig.head(5)
```

```
C:\Users\alexa\AppData\Local\Temp\ipykernel_16472\1640917472.py:1: DtypeWarning:

Columns (6,22,31,32,33,34,35,36,37,38,39,40,41) have mixed types. Specify dtype option on imp
```

	date	locality	state	location_detail	online	type	m
0	2017-01-01	Washington	DC	Lafayette Square Park	0.0	vigil	Na
1	2017-01-01	Mankato	MN	NaN	0.0	vigil	Na
2	2017-01-01	Minneapolis	MN	U.S. Bank Stadium	0.0	protest; banner drop	Na
3	2017-01-01	Little Compton	RI	Town Green	0.0	vigil	Na
4	2017-01-01	Oak Ridge	TN	Y-12 National Security Complex	0.0	vigil	Na

Only data for crowds that aren't online with a crowd size, information about what the crowd gathered for, and a FIPS code is kept. Also, the data is rid of `source` columns, as they are irrelevant for our purposes.

```
crowd_data = crowd_data_orig[(crowd_data_orig["issues"].notna()) &
                             (crowd_data_orig["size_mean"].notna()) &
                             (crowd_data_orig["fips_code"].notna()) &
                             (crowd_data_orig["online"] == 0) &
                             (crowd_data_orig["size_mean"] != 0) &
                             (crowd_data_orig["size_mean"].notna()) &
                             (crowd_data_orig["type"].notna())]
crowd_data = crowd_data.drop(columns=np.array(["source_"]*30)+np.arange(1,31).astype(str))
crowd_data.isna().sum()[crowd_data.isna().sum() > 0]
```

```
location_detail       2749
macroevent           32293
actors                1220
valence                 10
size_text             6536
size_high               33
arrests               3927
injuries_crowd        3985
injuries_police       3973
property_damage       3956
chemical_agents      30723
notes                29202
lat                      2
lon                      2
resolved_locality      340
resolved_county       1923
resolved_state          10
dtype: int64
```

```
# source_cols = np.array(["source_"]*30) + np.arange(1,31).astype(str)
# crowd_data.drop(columns=source_cols, inplace=True)
# crowd_data.head(5)
# crowd_data["fips_code"] = crowd_data["fips_code"].astype(np.int16)
# crowd_data["fips_code"]

crowd_data["year"] = crowd_data["date"].apply(lambda x:x[:4]).astype(int)
crowd_data["year"]
```

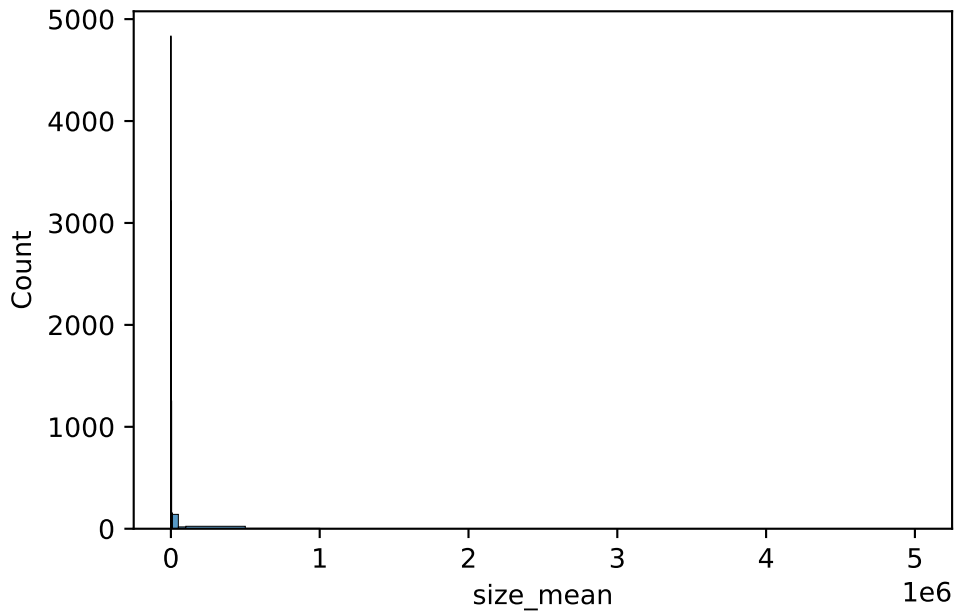
```
2        2017
8        2017
13       2017
```

```
14        2017
16        2017
          ...
72155     2020
72156     2020
72167     2020
72168     2020
72172     2020
Name: year, Length: 33467, dtype: int64
```

The data is limited to the years 2017 and 2018, as that is the overlap available to us between these three disparate datasets.

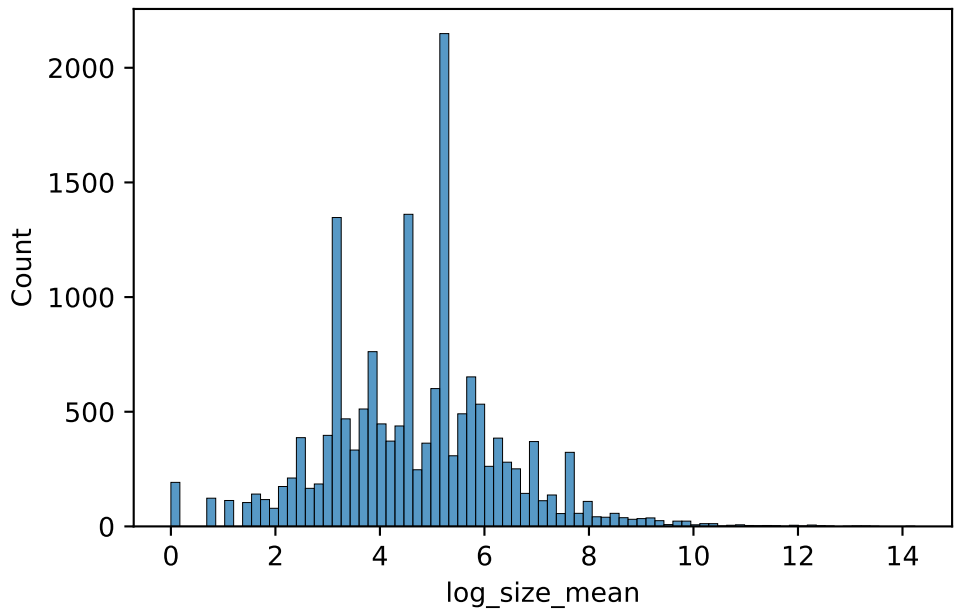
```
crowd_data = crowd_data[(crowd_data["year"] == 2017) | (crowd_data["year"] == 2018)]
```

```
sns.histplot(data=crowd_data, x="size_mean", bins=[0, 5, 10, 20, 50, 100, 250, 500, 1000, 500
plt.show()
```



As is to be expected, the data is extremely right-skewed. A column is added for `log_size_mean`, for a target variable that more closely approximates a normal distribution (as is assumed by e.g. linear regression).

```
crowd_data["log_size_mean"] = crowd_data["size_mean"].apply(np.log)
sns.histplot(data=crowd_data, x="log_size_mean")
plt.show()
```



It is still right-skewed, but much better.

Next, the FIPS codes are corrected.

```
print(crowd_data["fips_code"].isna().sum(), "out of", len(crowd_data))
```

```
0 out of 16703
```

There are 0 unresolved counties out of 16703; we need not discard any data and may proceed to cast the column as strings.

```
# crowd_data = crowd_data[crowd_data["fips_code"].notna()]
crowd_data["fips_code"] = crowd_data["fips_code"].astype(int).astype(str).str.rjust(5,"0").as
crowd_data["fips_code"]
```

```
2         27053
8         12086
13        26161
```

```
14        38085
16        36061
          ...
32405     37081
32413     01101
32415     17031
32424     30087
32426     39035
Name: fips_code, Length: 16703, dtype: category
Categories (1316, object): ['01003', '01007', '01015', '01041', ..., '56041', '66000', '72000
```

Many of these columns are unused in the final product; the data is thereby reindexed to the requisite columns.

```
crowd_data = crowd_data[["year","fips_code","type","issues","claims","size_mean", "log_size_m
crowd_data.head(5)
```

	year	fips_code	type	issues
2	2017	27053	protest; banner drop	banking and finance; economy; energy; environment; indigen
8	2017	12086	demonstration	policing
13	2017	26161	protest	education; racism
14	2017	38085	protest; occupying land	energy; environment
16	2017	36061	protest	presidency

Mislabeled and misspelled `type`s are corrected, then the strings are split into lists of `type`s.

For a Harvard dataset, there is much that must be cleaned up. Perhaps such problems were fixed in later phases of the research.

```
crowd_data_type = crowd_data["type"].str.lower()#.str.replace(r"\s+", " ", regex=True)

crowd_data_type_repl_dict = {"nat'l":"national",
                             ",":";", "/":";", " and ":";", "'":";", ":":";",
                             "block highways":"block highway",
                             "counter-protest":"counterprotest",
                             "counterprotesting families belong together march":"counterprote
                             "demonstraton":"demonstration", "demonstation":"demonstration",
                             "demonstratin":"demonstration", "deomnstration":"demonstration"
                             "flashmob":"flash mob",
                             "marach":"march",
                             "protests":"protest", "protestors":"protest", "protesst":"protes
```

```
                              "protest hike":"protest; hike",
                              "protest resignation":"protest; resignation",
                              "protest (walk-out)":"protest; walk-out",
                              "protest walk":"protest; walk",
                              "raly":"rally", "rallies":"rally", "rallying":"rally",
                              "sit-in demonstration":"sit-in",
                              "walk in":"walk-in","walkin":"walk-in",
                              "walk out":"walk-out", "walkout":"walk-out","walkut":"walk-out"
                              }

for key, value in crowd_data_type_repl_dict.items():
    crowd_data_type = crowd_data_type.str.replace(key, value)

crowd_data_type = crowd_data_type.str.split(";").explode().astype(str).str.lstrip().str.rstri
crowd_data_type = crowd_data_type.loc[(crowd_data_type != "nan") & (crowd_data_type != "m") &
crowd_data_type.unique()
```

```
array(['protest', 'banner drop', 'demonstration', 'occupying land',
       'strike', 'vigil', 'rally', 'direct action', 'march',
       'flag burning', 'block highway', 'sign held on crane',
       'counterprotest', 'sit-in', 'standoff', 'dance', 'blockade',
       'nonviolent blockade', 'road blockade', 'mass mooning',
       'kindness march', 'civil disobedience', 'walk-out',
       'human barricade', 'block streets', 'peace march',
       'protest (armed)', 'hunger strike', 'sick out', 'boycott',
       'swim protest', 'die-in', 'boycott classes', 'resignation',
       'turn backs to speaker', 'die in', 'art projection', 'sing-in',
       'skits', 'flash mob', 'flotilla', 'counter protest',
       'block interactions', 'light rail stop', 'superhero rally',
       'aircraft flyover', 'death train', 'street blockade', 'encampment',
       '"drive"', 'railroad blockade', 'pray-in', 'picket',
       'railway blockade', 'hike', 'festival', 'walk', 'dinner', 'event',
       'mass', 'parade', 'standout', 'protest vandalism',
       'silent protest', 'parents', 'support walk-out', 'walk-in',
       'postcard writing to public officials', 'science fair',
       'mock execution', 'teach in', 'on-campus discussion',
       'kayak protest', 'iftar', 'moment of silence', 'prayer rally',
       'self-immolation protest', 'convoy', 'kneeling protest',
       'procession', 'rent strike', 'hunger-strike', '"kookout"',
       'protest in trees', 'educational event',
       'kneeling during national anthem', 'non-partisan rally', 'demo',
       'speeches', 'drive by mar-a-lago', 'prayer vigil',
```

```
       'press conference', 'grade-strike'], dtype=object)
```

```
crowd_data_orig[crowd_data_orig["type"]=="parents"]
```

	date	locality	state	location_detail	online	type	macroevent	actors
13414	2018-03-01	Muskogee	OK	Muskogee High School	0.0	parents	NaN	teachers, stu

It is unclear as to what it means for a crowd to be of `type` "parents" having `actors` that are teachers and students.

```
type_onehot = pd.get_dummies(crowd_data_type, prefix="type", drop_first=True).groupby(crowd_
# type_onehot

crowd_data = crowd_data.drop(columns="type").join(type_onehot)
crowd_data = crowd_data[crowd_data.notna().all(axis=1)]
crowd_data.head(5)
```

	year	fips_code	issues	claims
2	2017	27053	banking and finance; economy; energy; environment; indigenous peoples	against the
8	2017	12086	policing	in remembra
13	2017	26161	education; racism	against racia
14	2017	38085	energy; environment	for environm
16	2017	36061	presidency	against Trun

`type` is one-shot encoded. To prevent multicollinearity, the first category—`"drive"`—is dropped.

```
crowd_data["size_cat"] = crowd_data["size_cat"].astype("category").cat.as_ordered()
crowd_data["size_mean"] = crowd_data["size_mean"].astype(int)
crowd_data[["size_cat","size_mean"]]
```

	size_cat	size_mean
2	1	2
8	1	18
13	2	200
14	2	300
16	1	2

	size_cat	size_mean
...	...	...
32405	2	150
32413	1	12
32415	1	20
32424	2	100
32426	1	30

`size_cat` is typed as a category and `size_mean` as an integer.

```
crowd_data_problems = pd.concat([crowd_data["issues"].str.split("; "),crowd_data["year"]],ax
crowd_data_problems = crowd_data_problems.explode(column="issues")
crowd_data_problems.head(20)
```

	issues	year
2	banking and finance	2017
2	economy	2017
2	energy	2017
2	environment	2017
2	indigenous peoples	2017
8	policing	2017
13	education	2017
13	racism	2017
14	energy	2017
14	environment	2017
16	presidency	2017
21	presidency	2017
26	education	2017
28	racism	2017
34	women's rights	2017
37	energy	2017
37	environment	2017
38	democracy	2017
38	judiciary	2017
38	presidency	2017

`issues` is prepared for merging by splitting and exploding, segmenting it off into its own dataframe (with `year`) to save space and time when merging.

American Community Survey (ACS) 1-Year Estimates

The American Community Survey is a comprehensive yearly survey conducted by the U.S. Census Bureau. For the purposes of this project, we will be looking at the 1-year estimates for the most up-to-date demographic data, and analyzing all variables found in the five Data Profiles, as they are selected subsets that capture various potentially-relevant information for our purposes: social, economic, housing, and demographic characteristics (U.S. Census Bureau, 2017, 2018).

This data is aggregated at the county level, as that is the smallest unit for which the majority of the locations found in the CCC dataset have data.

```
api_key = "29e7dfea2f8b253a0a10ccd9626f78e49f4f0a4f"

def regex_filter(string, myregex):
    if string:
        mo = re.search(myregex, string)
        if mo:
            return True
        else:
            return False
    else:
        return False

census_vars_orig = ced.variables.search(ACS1_PROFILE, 2017)
census_vars = census_vars_orig[(census_vars_orig["GROUP"] != "N/A")]
census_vars_pcts = census_vars[census_vars["VARIABLE"].apply(regex_filter,myregex=r"PE$")]
census_vars_pcts_no_ratio = census_vars_pcts[~census_vars_pcts["LABEL"].apply(regex_filter,my

census_vars_agg = census_vars_pcts_no_ratio[census_vars_pcts_no_ratio["LABEL"].apply(regex_fi
# census_vars_tot = census_vars_pcts_no_ratio[census_vars_pcts_no_ratio["LABEL"].apply(regex_
# census_vars_tot[["VARIABLE","LABEL"]]

# census_vars_agg[["VARIABLE","LABEL"]]

census_vars_pcts_no_ratio_agg = census_vars_pcts_no_ratio.drop(census_vars_agg.index)

census_vars_agg_ests = census_vars_agg["VARIABLE"].apply(lambda x:x.replace("PE","E"))
```

The variables are limited to percent estimates—those variables ending in `PE`—excepting those that have no percent estimates, e.g. aggregate variables like medians; for those, the estimates—variables ending solely in `E`—are taken instead. Variables representing population totals—that

would have percent estimates of 100%—instead have percent estimates equivalent to their estimates, so these do not need to be altered.

The variables representing ratios are also eliminated.

```
census_vars_to_drop = ["DP05_0033PE", # same as DP05_0001PE (Total Population)
                       "DP05_0105PE", # total housing units-replace with DP05_0105E, which is
                       "DP04_0006PE", # same as DP04_0001PE
                       "DP04_0016PE", # same as DP04_0001PE
                       "DP04_0027PE", # same as DP04_0001PE
                       "DP04_0038PE" # same as DP04_0001PE
                       ]

# census_vars_to_add = []

census_vars_pcts_no_ratio_agg_drop = census_vars_pcts_no_ratio_agg["VARIABLE"][~census_vars_

census_vars_final = pd.concat([pd.Series(["NAME"]),
                               pd.concat([census_vars_pcts_no_ratio_agg_drop,
                                          census_vars_agg_ests,
                                        #   pd.Series(census_vars_to_add)
                                        ], ignore_index=True).sort_values()],
                              ignore_index=True)

census_vars_final
```

```
0               NAME
1      DP02PR_0001PE
2      DP02PR_0002PE
3      DP02PR_0003PE
4      DP02PR_0004PE
           ...
661      DP05_0085PE
662      DP05_0086PE
663      DP05_0087PE
664      DP05_0088PE
665      DP05_0089PE
Length: 666, dtype: object
```

Variables are eliminated which repeat information seen elsewhere in the data. After doing so, there are 666 still remaining.

```
census_data_orig = [ced.download(ACS1_PROFILE, year, census_vars_final,
                                 state = "*", county ="*",
                                 api_key=api_key
                                 ).set_index(["STATE","COUNTY"])
                    for year in range(2017,2019)]


census_data_orig = pd.concat(census_data_orig, keys = np.arange(2017,2019), names=["year"])
census_data_orig
```

year	STATE
2017	47
	48
...	...
2018	48

Data is pulled for each of the two years for which we have data in all of the datasets, then concatenated into one dataframe, with their respective years as their keys.

Notably, data for the year of 2020 is absent, due to difficulties surrounding the COVID-19 Pandemic.

Though `DP02PR` originates from the Puerto Rico Community Survey (PRCS) and not the American Community Survey (ACS), with some differences as to how answers are collected and the exact language used, the columns are identical to the ones found in `DP02`. As such, rather than discard data from the 16 crowds in the CCC dataset located in Puerto Rico, each is mapped to their corresponding `DP02` variable, with the `DP02PR` columns then being dropped.

```
pr_cols = census_data_orig.columns[census_data_orig.columns.str.startswith("DP02PR_")]

census_data_pr = census_data_orig[pr_cols][census_data_orig["DP02PR_0001PE"].isna() == False]
census_data_pr = census_data_pr.rename(columns=dict(zip(pr_cols, pr_cols.str.replace("PR","")
```

```
census_data_no_pr = census_data_orig.drop(columns=pr_cols)

census_data = census_data_no_pr.combine_first(census_data_pr)
census_data[(census_data_orig["DP02PR_0001PE"].isna() == False).reindex(census_data.index)]
```

year	STATE
2017	72
2018	72

As per Prof. Darcy Steeg Morris' recommendation, to limit the columns to a reasonable number, all columns with any NAs are eliminated. Largely, these are small variables, such as `DP05_0042PE`: the percent of the population in Navajo tribes. She suggested that these may be left out intentionally due to privacy concerns.

```
census_data = census_data.drop(columns=census_data.columns[census_data.isna().any()])
census_data.columns.size
```

```
246
```

After eliminating, there are 246 columns remaining.

Most Important Problem Dataset (MIPD), Second Release (2024)

The second release of the Most Important Problem Dataset (MIPD) contains data from roughly 850 surveys and numerous institutions from between 1939 and 2020. Though reliability will vary by survey, as the question (or variants thereof) have been asked consistently across so many decades, it is a rich source from which to mine to understand salience among the American public (Heffington et al., 2017; "The 'Most Important Problem' Dataset (MIPD): A New Dataset on American Issue Importance," n.d.; Yildirim & Williams, 2025).

The data comprises a list of 12 general quasi-responses, as well as one representing "other", one representing "NaN", and one representing those who refused to answer, for each year. Each of these are granted descriptions, issues, and examples, as well as the percent of total respondents asked the question in the given year who gave that answer (Heffington et al., 2017; "The 'Most Important Problem' Dataset (MIPD): A New Dataset on American Issue Importance," n.d.; Yildirim & Williams, 2025).

The aggregated data may be found here: https://williamslaro.github.io/talks/dataset2. Previously, the individual data was used, necessitating much preprocessing.

```
issue_data = pd.read_csv("C:\\Users\\alexa\\Code\\GW DATS\\6103 12 Intro to Data Mining\\Fina
                         index_col=["year","problem"])
issue_data
```

year	problem	cat
	1	1
	2	1
1939	3	1
	4	1
	5	1
...	...	...
	11	1
	12	1
2018	13	1
	14	1
	15	1

The data is limited to the relevant years. Though the paper covers years ranging from 1939 to 2020, data for the years 2019 and 2020 are absent from the aggregated dataset. To quote Laron K. Williams: "Unfortunately, it is a product of polling companies choosing to monetize the data rather than posting them in a free public repository (like the Roper Center)" (personal communication, November 20, 2025).

```
issue_data = issue_data.loc[[2017,2018]]
issue_data.head(5)
```

year	problem	cat
2017	1	1
	2	1
	3	1
	4	1
	5	1

The data is limited to the relevant columns, including those describing the problems, for reference.

```
issue_data_problems = issue_data[["problemname","problemdesc","problemissues","problemexample
issue_data_problems
```

year	problem	prob
2017	1	Econ
	2	Soci
	3	Righ
	4	Publ
	5	Fisc
	6	Fore
	7	Inte
	8	Envi
	9	Mor
	10	Polit
	11	Yout
	12	Grou
	13	Othe
	14	NaN
	15	Don
2018	1	Econ
	2	Soci
	3	Righ
	4	Publ
	5	Fisc

year	problem	prob
	6	Fore
	7	Inter
	8	Envi
	9	Mora
	10	Polit
	11	Yout
	12	Grou
	13	Othe
	14	NaN
	15	Don

`problemname` is set as a categorical variable.

```
issue_data_problems["problemname"] = issue_data_problems["problemname"].astype("category")
issue_data_problems["problemname"]
```

```
C:\Users\alexa\AppData\Local\Temp\ipykernel_16472\1646687161.py:1: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
year  problem
2017  1                                 Economy
      2                            Social Policy
      3                                   Rights
      4                            Public Safety
      5                            Fiscal Policy
      6                           Foreign Policy
      7          International Economic Relations
      8                             Environment
      9                                Morality
      10                               Politics
      11                           Youth Issues
      12                                 Groups
```

```
      13                            Other and All
      14                                      NaN
      15                     Don't Know/Refused
2018  1                                 Economy
      2                           Social Policy
      3                                  Rights
      4                           Public Safety
      5                           Fiscal Policy
      6                          Foreign Policy
      7          International Economic Relations
      8                             Environment
      9                                Morality
      10                               Politics
      11                           Youth Issues
      12                                 Groups
      13                          Other and All
      14                                      NaN
      15                     Don't Know/Refused
Name: problemname, dtype: category
Categories (14, object): ['Don't Know/Refused', 'Economy', 'Environment', 'Fiscal Policy', .
```

Merge the Datasets

Merge Crowd and Issue Data

Some crowds are gathered for issues that fit into multiple categories, organized in alphabetical order.

```
crowd_data["issues"]
```

```
2        banking and finance; economy; energy; environment; indigenous peoples
8                                                                      policing
13                                                        education; racism
14                                                      energy; environment
16                                                               presidency
                                          ...
32405                                                     foreign affairs
32413                                                   policing; racism
32415                                                            housing
32424                                                    women's rights
32426                                                   criminal justice
Name: issues, Length: 16701, dtype: object
```

It is unclear how these should best be merged. Perhaps solely the issue with the highest salience should be accounted for. However, for the purposes of this project, it was decided the percents should be added, representing the percent of the population who would judge at least one of the issues at play to be a part of the Most Important Problem for that year. As the MIPDV2 Codebook states: "The percentages can therefore be interpreted as the percentage of the electorate who answered the question and selected that category" (p. 5).

```
np.sort(crowd_data_problems["issues"].unique())
```

```
array(['animal rights', 'banking and finance', 'civil rights',
       'corruption', 'covid', 'criminal justice', 'democracy',
       'development', 'disability rights', 'drugs', 'economy',
       'education', 'energy', 'environment', 'foreign affairs',
       'free speech', 'guns', 'healthcare', 'housing', 'immigration',
       'indigenous peoples', 'judiciary', 'labor', 'legislative',
       'lgbtqia', 'military', 'patriotism', 'policing', 'presidency',
       'racism', 'religion', 'reproductive rights', 'science',
       'sexual violence', 'sports', 'taxes', 'transportation',
       "women's rights"], dtype=object)
```

Every issue present in the CCC data is presented above. Each is to be mapped to a problem code in the MIPD data. This will be done by hand.

The presence of `covid` is confusing, however, as this should be limited to the years 2017 and 18.

```
crowd_data[crowd_data["issues"].str.contains("covid")].reindex(columns=["issues","claims"])
```

	issues	claims
8278	covid; judiciary	Rally against court's decesion to reopen Rocky Flats
23918	covid; healthcare	Protesting CDC definition of lyme disease
26845	covid; immigration	against ICE; against reopening immigrant detention facility
26968	covid; energy	protesting reopening of Japanese nuclear power plants

According to the data dictionary, "These are generated after data compilation by running a series of regular expressions over the claims description text" (p. 3). These claims all include the words "reopen", "reopening", or "CDC"; these may thereby be taken to be false positives. As such, `covid` will manually be removed from these four so as to not impact findings.

```
crowd_data_problems = crowd_data_problems[crowd_data_problems["issues"] != "covid"]
```

Once more, the data is checked for unique issues.

```
np.sort(crowd_data_problems["issues"].unique())
```

```
array(['animal rights', 'banking and finance', 'civil rights',
       'corruption', 'criminal justice', 'democracy', 'development',
       'disability rights', 'drugs', 'economy', 'education', 'energy',
       'environment', 'foreign affairs', 'free speech', 'guns',
       'healthcare', 'housing', 'immigration', 'indigenous peoples',
       'judiciary', 'labor', 'legislative', 'lgbtqia', 'military',
       'patriotism', 'policing', 'presidency', 'racism', 'religion',
       'reproductive rights', 'science', 'sexual violence', 'sports',
       'taxes', 'transportation', "women's rights"], dtype=object)
```

```
issue_code_dict = {'animal rights':13,
                   'banking and finance':1,
                   'civil rights':3,
                   'corruption':10,
                   'criminal justice':3,
                   'democracy':10,
                   'development':1,
                   'disability rights':3,
                   'drugs':4,
                   'economy':1,
                   'education':2,
                   'energy':8,
                   'environment':8,
                   'foreign affairs':6,
                   'free speech':3,
                   'guns':4,
                   'healthcare':2,
                   'housing':2,
                   'immigration':5,
                   'indigenous peoples':12,
                   'judiciary':10,
                   'labor':1,
                   'legislative':10,
                   'lgbtqia':3,
                   'military':6,
```

```
                    'patriotism':9,
                    'policing':3,
                    'presidency':10,
                    'racism':3,
                    'religion':9,
                    'reproductive rights':3,
                    'science':10,
                    'sexual violence':4,
                    'sports':13,
                    'taxes':1,
                    'transportation':2,
                    "women's rights":3
}
```

```
crowd_data_problems["issues"] = crowd_data_problems["issues"].apply(np.vectorize(lambda x:iss
crowd_data_problems_to_merge = crowd_data_problems.reset_index().drop_duplicates() # neccessa
crowd_data_problems_to_merge
```

	index	issues	year
0	2	1	2017
2	2	8	2017
4	2	12	2017
5	8	3	2017
6	13	2	2017
...	...	...	...
25901	32405	6	2018
25902	32413	3	2018
25904	32415	2	2018
25905	32424	3	2018
25906	32426	3	2018

The mapping is applied, but some lists of `issues` will have multiple map to the same `problem`. As such, before they can be re-grouped, duplicates must be deleted; to that end, `index` is turned into a column with `reset_index()`, so that only those which also have duplicate indices are removed.

`index` is also necessary to merge this dataframe back with `crowd_data`.

```
crowd_problems_data_exploded = crowd_data_problems_to_merge.merge(issue_data_problems[["perc
crowd_problems_data_exploded
```

	index	issues	year	perc	problemname
0	2	1	2017	19.669001	Economy
1	2	8	2017	2.157000	Environment
2	2	12	2017	0.000000	Groups
3	8	3	2017	4.969000	Rights
4	13	2	2017	16.531000	Social Policy
...	...	...	...	...	...
24846	32405	6	2018	12.685000	Foreign Policy
24847	32413	3	2018	8.407000	Rights
24848	32415	2	2018	11.224000	Social Policy
24849	32424	3	2018	8.407000	Rights
24850	32426	3	2018	8.407000	Rights

```
crowd_problems_data_exploded["problemname"] = crowd_problems_data_exploded["problemname"].cat
crowd_problems_data_exploded["problemname"].unique()
```

```
['Economy', 'Environment', 'Groups', 'Rights', 'Social Policy', ..., 'Public Safety', 'Fiscal
Length: 11
Categories (11, object): ['Economy', 'Environment', 'Fiscal Policy', 'Foreign Policy', ...,
```

Unused categories must be removed before one-hot encoding, as, otherwise, categories that are not used will get their own columns, filled completely with `False`s.

```
problems_onehot_to_group = pd.get_dummies(crowd_problems_data_exploded[["index","problemname"

problems_onehot_cols = np.setdiff1d(problems_onehot_to_group.columns,["perc","index"])
problems_onehot_grouped = problems_onehot_to_group.groupby(by="index").agg({"perc":"sum"} | c
problems_onehot_grouped

# crowd_problems_perc = crowd_data.drop(columns="type").join(type_onehot)
# crowd_problems_perc
```

index	perc	problemname_Economy	problemname_Environment	problemname_Fiscal Policy	pr
2	21.826001	True	True	False	Fa
8	4.969000	False	False	False	Fa
13	21.500000	False	False	False	Fa
14	2.157000	False	True	False	Fa
16	14.678000	False	False	False	Fa

index	perc	problemname_Economy	problemname_Environment	problemname_Fiscal Policy	pr
...	...	...	...	...	...
32405	12.685000	False	False	False	Tr
32413	8.407000	False	False	False	Fa
32415	11.224000	False	False	False	Fa
32424	8.407000	False	False	False	Fa
32426	8.407000	False	False	False	Fa

Much like `type` was before, the one-hot encoded columns and `perc` are grouped so that, once again, each one `index` corresponds to exactly one row.

Lastly, merge back with original data.

```
crowd_problems_data = crowd_data.merge(problems_onehot_grouped, how="inner", left_index=True
crowd_problems_data.head(5)
```

	year	fips_code	issues	claims
2	2017	27053	banking and finance; economy; energy; environment; indigenous peoples	against the
8	2017	12086	policing	in remembra
13	2017	26161	education; racism	against racia
14	2017	38085	energy; environment	for environm
16	2017	36061	presidency	against Trun

Merge with Census Data

```
# cbsa_fips_crosswalk = pd.read_excel("C:\\Users\\alexa\\Code\\GW DATS\\6103 12 Intro to Data
# cbsa_fips_crosswalk.head()
```

```
# cbsa_fips_crosswalk["FIPS Code"] = cbsa_fips_crosswalk["FIPS State Code"] + cbsa_fips_cross
# cbsa_fips_crosswalk["FIPS Code"] = cbsa_fips_crosswalk["FIPS Code"].astype("category")
```

```
# cbsa_fips_crosswalk["FIPS Code"] = cbsa_fips_crosswalk["FIPS Code"].astype("category")
# cbsa_fips_crosswalk["CBSA Code"] = cbsa_fips_crosswalk["CBSA Code"].astype("category")
# cbsa_fips_crosswalk["FIPS State Code"] = cbsa_fips_crosswalk["FIPS State Code"].astype("cat
```

```
# cbsa_fips_crosswalk = cbsa_fips_crosswalk[cbsa_fips_crosswalk["FIPS Code"].notna()]
# cbsa_fips_crosswalk["FIPS Code"]
```

```
# crowd_problems_cbsa_data = crowd_problems_data.merge(cbsa_fips_crosswalk[["FIPS Code", "CBS
#                                                     how="inner", left_on="fips_code", righ
# crowd_problems_cbsa_data
```

```
census_data = census_data.reset_index()
```

```
census_data["fips_code"] = census_data["STATE"]+census_data["COUNTY"]
census_data["fips_code"]
```

```
0       01003
1       01015
2       01043
3       01049
4       01051
        ...
1670    72113
1671    72127
1672    72135
1673    72137
1674    72139
Name: fips_code, Length: 1675, dtype: object
```

A single `fips_code` column is created so that the datasets may be merged one last time.

```
demo_data = crowd_problems_data.merge(census_data, how="inner", on = ["year","fips_code"])
demo_data.head(5)
```

	year	fips_code	issues	claims
0	2017	27053	banking and finance; economy; energy; environment; indigenous peoples	against the D
1	2017	12086	policing	in remembran
2	2017	26161	education; racism	against racial
3	2017	36061	presidency	against Trum
4	2017	11001	presidency	against the ce

```
demo_data.columns[demo_data.isna().any()]
```

```
Index([], dtype='object')
```

As the columns with NAs have been eliminated previously, the data is clean.

```
# demo_data_clean_na_cols = demo_data_clean.columns[demo_data_clean.isna().sum() > 0]

# demo_data_clean_na_dict = dict(zip(demo_data_clean_na_cols,
#                                    demo_data_clean[demo_data_clean_na_cols].mean()))
# demo_data_clean_na_dict
```

```
# demo_data_clean = demo_data_clean.fillna(demo_data_clean_na_dict)
# (demo_data_clean.isna().sum() > 0).any()
```

One final step to preprocess: eliminate unused categories, then one-hot encode `NAME` and `STATE` and eliminate all other string variables—including `fips_code`, which, post-merging, is no longer necessary (and is perfectly collinear with `NAME`.). As before, multicollinearity is avoided.

```
demo_data["NAME"] = demo_data["NAME"].astype("category")
demo_data["STATE"] = demo_data["STATE"].astype("category")

demo_data_preprocessed = pd.get_dummies(demo_data, columns=["NAME","STATE"], drop_first=True)
demo_data_preprocessed
```

	year	size_mean	log_size_mean	size_cat	type_"kookout"	type_aircraft flyover	type_art pr
0	2017	2	0.693147	1	False	False	False
1	2017	18	2.890372	1	False	False	False
2	2017	200	5.298317	2	False	False	False
3	2017	2	0.693147	1	False	False	False
4	2017	3	1.098612	1	False	False	False
...	...	...	...	...	...	...	...
15128	2018	2	0.693147	1	False	False	False
15129	2018	150	5.010635	2	False	False	False
15130	2018	12	2.484907	1	False	False	False
15131	2018	20	2.995732	1	False	False	False
15132	2018	30	3.401197	1	False	False	False

```
# np.setdiff1d(demo_data["NAME"], demo_data["NAME"].drop_duplicates(keep=False))
```

Modeling

```
target_cont = "log_size_mean"
target_cat = "size_cat"

targets = [target_cont, target_cat, "size_mean"]

demo_train, demo_test = train_test_split(demo_data_preprocessed,
                                         train_size=0.8,
                                         random_state=seed,
                                         stratify=demo_data_preprocessed[target_cat])

# reset the index
demo_train, demo_test = demo_train.reset_index(drop=True), demo_test.reset_index(drop=True)

feature_names = np.setdiff1d(demo_train.columns, targets)

X_train = demo_train[feature_names].values
X_test = demo_test[feature_names].values

# Target
y_train_cont = demo_train[target_cont].values
y_test_cont = demo_test[target_cont].values

y_train_cat = demo_train[target_cat].values
y_test_cat = demo_test[target_cat].values
```

The train and test datasets are prepared. Note that two targets have been chosen: `log_size_mean` and `size_cat`; the former will be used as the target for regression, and the latter the target for categorization.

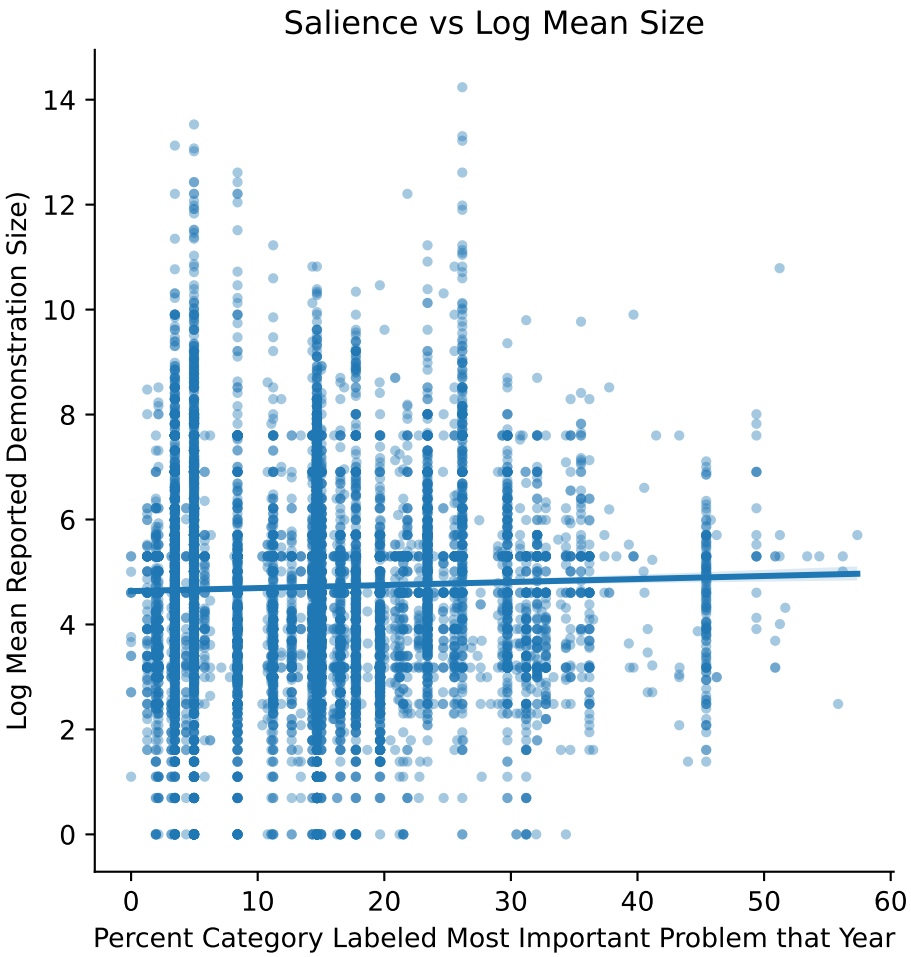
```
ss = StandardScaler()

# training data
X_train = ss.fit_transform(X_train)

# test data
X_test = ss.transform(X_test)
```

The features are standardized.

Generalized Linear Models (GLMs)

Linear Regression

```
sns.lmplot(x = 'perc', y = target_cont,
           data = demo_data,
           fit_reg = True, scatter_kws={'alpha': 0.4, 's': 8 })
plt.xlabel("Percent Category Labeled Most Important Problem that Year")
plt.ylabel("Log Mean Reported Demonstration Size)")
plt.title("Salience vs Log Mean Size")
plt.show()
```



```
ols_log_size_mean_by_perc = ols(formula = 'log_size_mean ~ perc', data = demo_data)
ols_log_size_mean_by_perc_fit = ols_log_size_mean_by_perc.fit()
print( ols_log_size_mean_by_perc_fit.summary() )
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          log_size_mean   R-squared:                       0.001
Model:                            OLS   Adj. R-squared:                  0.001
Method:                 Least Squares   F-statistic:                     11.77
Date:                Fri, 12 Dec 2025   Prob (F-statistic):           0.000604
Time:                        23:57:25   Log-Likelihood:                 -
29713.
No. Observations:               15133   AIC:                          5.943e+04
Df Residuals:                   15131   BIC:                          5.945e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.6357      0.028    164.644      0.000       4.581       4.691
perc           0.0058      0.002      3.431      0.001       0.002       0.009
==============================================================================
Omnibus:                      525.906   Durbin-Watson:                   1.491
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              983.246
Skew:                           0.273   Prob(JB):                     3.10e-
214
Kurtosis:                       4.123   Cond. No.                         33.8
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Evidence accumulates in support of a positive response to the SMART question as early as a simple linear regression with the percent of the population who would judge at least one of the issues at play to be a part of the Most Important Problem for that year being the only predictor.

Despite having an R^2 of 0.001, the F-statistic is at 11.77—highly significant. Indeed, both the intercept and `perc` have very small p-values.

In other words, though the salience of a category of issues does not explain nearly any of the variance in the log mean reported size of demonstrations, it is nevertheless explaining some of it.

```
ols_log_size_mean_by_perc_pop = ols(formula = 'log_size_mean ~ perc + DP05_0001PE', data = de
ols_log_size_mean_by_perc_pop_fit = ols_log_size_mean_by_perc_pop.fit()
print( ols_log_size_mean_by_perc_pop_fit.summary() )
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          log_size_mean   R-squared:                       0.005
Model:                            OLS   Adj. R-squared:                  0.005
Method:                 Least Squares   F-statistic:                     36.42
Date:                Fri, 12 Dec 2025   Prob (F-statistic):           1.66e-
16
Time:                        23:57:25   Log-Likelihood:                 -
29683.
No. Observations:               15133   AIC:                          5.937e+04
Df Residuals:                   15130   BIC:                          5.939e+04
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      4.5692      0.029    155.623      0.000       4.512       4.627
perc           0.0055      0.002      3.271      0.001       0.002       0.009
DP05_0001PE  5.901e-08   7.55e-09      7.812      0.000    4.42e-08    7.38e-
08
==============================================================================
Omnibus:                      494.969   Durbin-Watson:                   1.487
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              920.540
Skew:                           0.258   Prob(JB):                     1.28e-
200
Kurtosis:                       4.092   Cond. No.                     4.63e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.63e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Adding in DP05_0001PE—total population—greatly boosts the predictive power, though, with two factors, the R^2 remains very low at 0.005.

```
# ols_log_size_mean_by_all = ols(formula = 'log_size_mean ~ .', data = demo_data) # formula c
# ols_log_size_mean_by_all_fit = ols_log_size_mean_by_all.fit()
# print( ols_log_size_mean_by_all.summary() )
```

```
demo_lm = LinearRegression()  # instantiate
demo_lm.fit(X_train, y_train_cont)
print("R^2 (with the test set):", demo_lm.score(X_test, y_test_cont))
print("R^2 (with the train set):", demo_lm.score(X_train, y_train_cont))
```

```
R^2 (with the test set): 0.04276461592441927
R^2 (with the train set): 0.2813385596802962
```

The linear model with all variables overfits; with an R^2 of 0.04, it is barely better than the null model. If linear regression is to be used, dimensionality reduction must be performed.

LASSO Regression

For the purposes of LASSO regression, `LassoCV` is used in place of `GridSearchCV`: "The advantage of using a cross-validation estimator over the canonical estimator class along with grid search is that they can take advantage of warm-starting by reusing precomputed results in the previous steps of the cross-validation process. This generally leads to speed improvements" (scikit-learn, 2025).

```
demo_lasso = LassoCV(n_alphas=100, max_iter=10000, random_state=seed, n_jobs=6)
demo_lasso.fit(X_train, y_train_cont)

print('LASSO Regression model accuracy (with the test set):', demo_lasso.score(X_test, y_test
print('LASSO Regression model accuracy (with the train set):', demo_lasso.score(X_train, y_tr
```

```
LASSO Regression model accuracy (with the test set): 0.17273548327115795
LASSO Regression model accuracy (with the train set): 0.23152201789832516
```

The `train` set could not be fit as tightly, but this is more than made up for by the massive improvement in the Coefficient of Determination with the `test` data.

```
# pd.DataFrame([feature_names, demo_lasso.coef_]).T.rename(columns={0:"var_name",1:"coef"}).s

lasso_coefs = pd.Series(demo_lasso.coef_,index=feature_names, name="coef")
lasso_coefs_sort = pd.DataFrame([lasso_coefs,lasso_coefs.abs().rename("abs_coef")]).T.sort_va
```

```
lasso_coefs_sort = lasso_coefs_sort[lasso_coefs_sort["coef"] != 0].reset_index().rename(colu
lasso_coefs_sort_merged = lasso_coefs_sort.merge(census_vars[["VARIABLE","LABEL"]], how="lef
lasso_coefs_sort_merged.head(20)
```

	feature	coef	abs_coef	LABEL
0	type_march	0.438333	0.438333	
1	type_protest	-0.304985	0.304985	
2	problemname_Public Safety	0.168577	0.168577	
3	problemname_Rights	0.143290	0.143290	
4	problemname_Fiscal Policy	0.141170	0.141170	
5	type_parade	0.128362	0.128362	
6	problemname_Politics	0.103969	0.103969	
7	problemname_Foreign Policy	-0.096764	0.096764	
8	type_counterprotest	-0.095642	0.095642	
9	type_walk-out	0.094778	0.094778	
10	DP02_0061PE	-0.078669	0.078669	Percent!!EDUCATIONAL ATTAINME
11	DP04_0080PE	0.058032	0.058032	Percent!!VALUE!!Owner-occupied unit
12	type_hunger strike	-0.057869	0.057869	
13	perc	0.056460	0.056460	
14	type_kneeling protest	-0.055056	0.055056	
15	NAME_Kanawha County, West Virginia	0.044903	0.044903	
16	year	0.042981	0.042981	
17	NAME_Bergen County, New Jersey	-0.040055	0.040055	
18	NAME_Talladega County, Alabama	-0.037649	0.037649	
19	DP04_0002PE	0.037046	0.037046	Percent!!HOUSING OCCUPANCY!!To

It is clear that the type of demonstration and the relevant Most Important Problem category are very important to the success of this regression. `perc` itself also appears, lower on the list; however, it is clear that the topics themselves—outside of salience (as measured in this way)—hold the power.

As for demographic variables, the total number of occupied housing units, as well as the percent that are occupied by their owners, both appear. Perhaps homeowners are more likely to demonstrate because they feel more integrated into their respective communities.

There are also a few counties who are particularly demonstration-inclined or declined.

The most prominent demographic variable is the percent of the adult population whose highest educational attainment is graduation from high school or its equivalent. Evidently, the lower this percent (and thereby the higher-educated the population at large is), the more people turn out. Perhaps those who have gone to college have a cultural impetus to protest.

Logistic Regression

```
demo_logit = LogisticRegression(penalty="l2", C=1, random_state=seed, max_iter=1000)
demo_logit.fit(X_train, y_train_cat)
# Cross-validation
kfold = KFold(5, random_state = seed, shuffle = True)
grid_logit = GridSearchCV(demo_logit, {'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}, cv = kfold, s
grid_logit.fit(X_train, y_train_cat)

print( grid_logit.best_params_ )
print( grid_logit.cv_results_[('mean_test_score')] )

demo_logit = grid_logit.best_estimator_

print('Logit model accuracy (with the test set):', demo_logit.score(X_test, y_test_cat))
print('Logit model accuracy (with the train set):', demo_logit.score(X_train, y_train_cat))
```

```
{'C': 0.001}
[0.55798745 0.55749127 0.55567429 0.55162671 0.54914874 0.54749663
 0.54667083]
Logit model accuracy (with the test set): 0.5609514370664024
Logit model accuracy (with the train set): 0.6141582686271271
```

Multinomial logistic regression is not particularly accurate at predicting the magnitude of a crowd. Evidently, it is easier to approach the correct number than to get its magnitude exactly.

Even so, it is interesting to note that the model is less overfit here than for linear regression without LASSO.

Primary Component Analysis (PCA)

```
# # PCA Computations
# X_avg = np.mean(X_train, axis = 0)
# B = X_train - np.tile(X_avg, (len(X_train), 1))

# U, S, VT = np.linalg.svd(B / np.sqrt(len(X_train)), full_matrices=True)
# V = VT.T

# #t = np.arange(0, 1, 0.1)
# # plt.plot(t, V[1, 0]/V[0, 0] * t)
```

```
# # plt.plot(t, V[1, 1]/V[0, 1] * t)
# plt.quiver([X_avg[0], X_avg[0]], [X_avg[1],X_avg[1]], V[0,:], V[1,:], scale=8, zorder=2, co
# plt.show()

# theta = 2 * np.pi * np.arange(0, 1, 0.01)
# Xstd = np.array([np.cos(theta), np.sin(theta)]).T @ np.diag(S) @ VT

# plt.plot(X_avg[0] +     Xstd[:, 0], X_avg[1] +     Xstd[:,1])
# plt.plot(X_avg[0] + 2 * Xstd[:, 0], X_avg[1] + 2 * Xstd[:,1])
# plt.plot(X_avg[0] + 3 * Xstd[:, 0], X_avg[1] + 3 * Xstd[:,1])
# plt.show()
```

```
demo_pca = PCA(n_components=0.95)  # Retain 95% of variance
demo_pca.fit(X_train)

X_train_pca = demo_pca.transform(X_train)
X_test_pca = demo_pca.transform(X_test)
```

Linear Regression with PCA

```
pca_lm = LinearRegression()
pca_lm.fit(X_train_pca,y_train_cont)

print("R^2 (with the test set):", pca_lm.score(X_test_pca, y_test_cont))
print("R^2 (with the train set):", pca_lm.score(X_train_pca, y_train_cont))
```

```
R^2 (with the test set): 0.07789234638581732
R^2 (with the train set): 0.18730111311333475
```

Though nearly doubling the Coefficient of Determination, linear regression with PCA nevertheless still overfits. In terms of dimensionality reduction, LASSO does a much better job.

Logistic Regression with PCA

```
pca_logit = LogisticRegression(penalty="l2", C=1, random_state=seed, max_iter=10000)
pca_logit.fit(X_train_pca,y_train_cat)

# Cross-validation
```

```
grid_logit = GridSearchCV(pca_logit, {'C': [0.001, 0.01, 0.1, 1, 5, 10, 100]}, cv = kfold, sc
grid_logit.fit(X_train_pca, y_train_cat)

print( grid_logit.best_params_ )
print( grid_logit.cv_results_[('mean_test_score')] )

pca_logit = grid_logit.best_estimator_

print('Logit model accuracy (with the test set):', pca_logit.score(X_test_pca, y_test_cat))
print('Logit model accuracy (with the train set):', pca_logit.score(X_train_pca, y_train_cat)
```

```
{'C': 1}
[0.53981464 0.54956162 0.55137943 0.5531137  0.55022247 0.54881803
 0.54914789]
Logit model accuracy (with the test set): 0.5523620746613809
Logit model accuracy (with the train set): 0.6225012390550141
```

Here, the accuracy with the test data is worse with PCA than without it.

Tree

Regression Tree

```
demo_dtr = DecisionTreeRegressor(max_depth=3, random_state=seed)

param_grid_dtr = {'max_depth': range(3, 15)}
                 #'criterion':('absolute_error', 'squared_error', 'friedman_mse', 'poisson')}
# Cross-validation
grid_dtr = GridSearchCV(demo_dtr, param_grid_dtr, cv = kfold, scoring = None, n_jobs=6)
grid_dtr.fit(X_train, y_train_cont)

print( grid_dtr.best_params_ )
print( grid_dtr.cv_results_[('mean_test_score')] )
demo_dtr = grid_dtr.best_estimator_
```

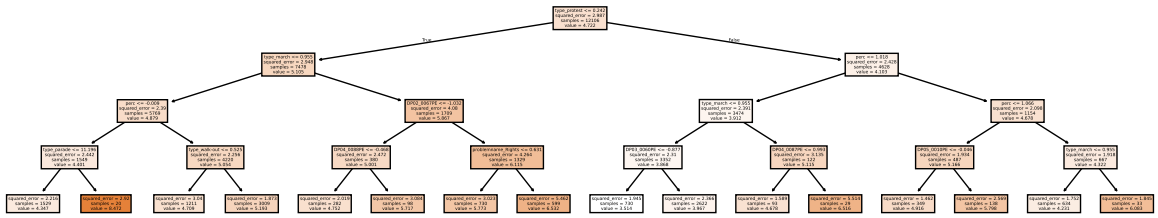
```
{'max_depth': 4}
[ 0.15427641  0.17541476  0.17179642  0.16702332  0.14898642  0.11507915
  0.07250315  0.03277081 -0.00644088 -0.04971834 -0.07389028 -0.11400164]
```

```
print("R^2 (with the test set):", demo_dtr.score(X_test, y_test_cont))
print("R^2 (with the train set):", demo_dtr.score(X_train, y_train_cont))
```

```
R^2 (with the test set): 0.17649969590118297
R^2 (with the train set): 0.1956188098385594
```

The regression tree does well, even on the test data.

```
plt.figure(dpi=1000,figsize=(15,3))
plot_tree(demo_dtr, feature_names = np.setdiff1d(demo_train.columns, targets), filled=True)
plt.show()
```



`type_protest`, `type_march`, and `perc` were the three most important variables, with `DP02_0067PE` not far behind. `DP02_0067PE` is the percent of the adult population whose highest educational attainment is graduation from high school (or equivalent) or beyond, and the less-educated a population is, the smaller demonstrations in the area are likely to be. This lends further weight to educational attainment's high position in the LASSO regression.

Classification Tree

```
demo_dtc = DecisionTreeClassifier(max_depth=3, random_state=seed)
demo_dtc.fit(X_train,y_train_cat)
demo_dtc.score(X_test,y_test_cat)
```

```
0.5850677238189627
```

```
param_grid_dtc = {'max_depth': range(3, 15, 2),
                  "criterion":("gini","entropy","log_loss")}
# Cross-validation
grid_dtc = GridSearchCV(demo_dtc, param_grid_dtc, cv = kfold, scoring = "accuracy", n_jobs=6)
grid_dtc.fit(X_train, y_train_cat)
```

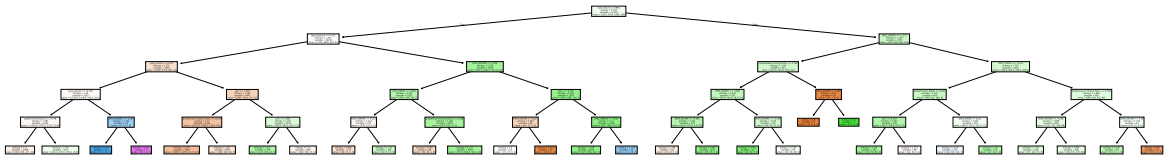
```
print( grid_dtc.best_params_ )
print( grid_dtc.cv_results_[('mean_test_score')] )
demo_dtc = grid_dtc.best_estimator_
```

```
{'criterion': 'entropy', 'max_depth': 5}
[0.56996457 0.56889002 0.5721939  0.55856473 0.54890057 0.54642321
 0.55583995 0.57392957 0.56335577 0.55972053 0.54510014 0.53898727
 0.55583995 0.57392957 0.56335577 0.55972053 0.54510014 0.53898727]
```

```
demo_dtc.score(X_test,y_test_cat)
```

```
0.5910142054839775
```

```
plt.figure(dpi=1000,figsize=(21,3))
plot_tree(demo_dtc, feature_names=np.setdiff1d(demo_train.columns, targets), filled=True)
plt.show()
```



`type_march`, `type_walk-out`, and `DP02_0064PE` are the three most important variables. Again, an educational attainment variable is significant: this time, it is the percent of the adult population whose highest educational attainment is an Associate's degree.

Other important variables include other `types`, `problems`, `DP05_0029PE`, and `DP03_0109PE`. `DP05_0029PE` is the percent of the population who are 65 years old or older, with higher levels of those of advanced age associated with larger protests, while `DP03_0109PE` is the percent of the population who are unemployed, with higher levels of unemployment associated with larger protests. Perhaps those who are unemployed or retired have more free time to protest.

`perc` does also make an appearance, but lower down.

Random Forest

Random Forest Regression

```
demo_rfr = RandomForestRegressor(n_estimators=100, max_depth=3, random_state=seed, n_jobs=6)
demo_rfr.fit(X_train, y_train_cont)

print("R^2 (with the test data):", demo_rfr.score(X_test, y_test_cont))
```

```
R^2 (with the test data): 0.16889189209025635
```

```
param_grid_rf = {
    'max_depth': [3, 5, 7, 9, 11],
    'n_estimators': [5, 10, 15, 20, 25, 30]
}
# Cross-validation
grid_rfr = GridSearchCV(demo_rfr, param_grid_rf, cv = kfold, scoring = None, n_jobs=6)
grid_rfr.fit(X_train, y_train_cont)

print( grid_rfr.best_params_ )
print( grid_rfr.cv_results_[('mean_test_score')] )
demo_rfr = grid_rfr.best_estimator_
```

```
{'max_depth': 7, 'n_estimators': 30}
[0.16530795 0.16770009 0.16790283 0.16854292 0.16871062 0.1683927
 0.19821715 0.20514605 0.20707459 0.2083553  0.20810933 0.20910525
 0.19486511 0.20848912 0.2131116  0.21583924 0.2165475  0.2181381
 0.18017232 0.19947351 0.20693605 0.21016008 0.21338016 0.21534464
 0.15067159 0.18217935 0.19052761 0.19475678 0.19996667 0.20249005]
```

Best: max depth 7, 30 estimators

```
print("R^2 (with the test data):", demo_rfr.score(X_test, y_test_cont) )
```

```
R^2 (with the test data): 0.2231426252546519
```

```
rfr_feature_imp = pd.DataFrame({'Feature': feature_names, 'Gini Importance': demo_rfr.feature
    'Gini Importance', ascending=False)
rfr_feature_imp = rfr_feature_imp.merge(census_vars[["VARIABLE","LABEL"]], how="left", left_
rfr_feature_imp.head(20)
```

	Feature	Gini Importance	LABEL
0	type_protest	0.211848	
1	type_march	0.150337	
2	perc	0.107702	
3	type_parade	0.027757	
4	type_walk-out	0.027299	
5	problemname_Rights	0.023090	
6	problemname_Fiscal Policy	0.019479	
7	DP02_0067PE	0.018525	Percent!!EDUCATIONAL ATTAINMENT!!Percent ba
8	DP05_0010PE	0.013239	Percent!!SEX AND AGE!!25 to 34 years
9	type_rally	0.009304	
10	type_counterprotest	0.007802	
11	problemname_Politics	0.007586	
12	DP02_0061PE	0.006558	Percent!!EDUCATIONAL ATTAINMENT!!Population
13	DP05_0029PE	0.006492	Percent!!SEX AND AGE!!65 years and over
14	DP03_0114PE	0.006269	Percent!!HEALTH INSURANCE COVERAGE!!Civilia
15	DP04_0054PE	0.006246	Percent!!YEAR HOUSEHOLDER MOVED INTO UN
16	problemname_Public Safety	0.005976	
17	DP02_0064PE	0.005715	Percent!!EDUCATIONAL ATTAINMENT!!Population
18	problemname_Foreign Policy	0.005683	
19	DP04_0042PE	0.005675	Percent!!BEDROOMS!!Total housing units!!3 bedroom

When the variables are ranked by importance, again, `type`s head the list, with `perc` in third place and different `problem`s not far below that. Educational attainment again makes an appearance, with the strongest support yet for the college culture hypothesis: this time, with the highest R^2 thus far, the variable chosen is those with a Bachelor's degree or higher. In addition, those of advanced age and not in the labor force are also important.

The percent of the population between 25 and 34 years old ranks quite highly, as well; perhaps those of that age are another population likely to protest.

Random Forest Classification

```
demo_rfc = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=seed, n_jobs=6)
demo_rfc.fit(X_train, y_train_cat)

print('RFC model accuracy (with the test set):', demo_rfc.score(X_test, y_test_cat))
print('RFC model accuracy (with the train set):', demo_rfc.score(X_train, y_train_cat))
```

```
RFC model accuracy (with the test set): 0.5229600264288073
RFC model accuracy (with the train set): 0.5431191144886833
```

```
# Cross-validation
grid_rfc = GridSearchCV(demo_rfc, param_grid_rf, cv = kfold, scoring = None, n_jobs=6)
grid_rfc.fit(X_train, y_train_cat)

print( grid_rfc.best_params_ )
print( grid_rfc.cv_results_[('mean_test_score')])
demo_rfc = grid_rfc.best_estimator_
```

```
{'max_depth': 9, 'n_estimators': 30}
[0.50693928 0.51164634 0.51652097 0.52205516 0.52238536 0.52800236
 0.51338096 0.5236235  0.53023247 0.53287577 0.5332893  0.53675812
 0.51362971 0.52263336 0.52428571 0.53048133 0.52990363 0.5358507
 0.5246989  0.52932403 0.53271072 0.53221526 0.53395022 0.53700667
 0.51585954 0.5232937  0.52585439 0.53163716 0.53213275 0.53403225]
```

```
print("Accuracy (with the test data):", demo_rfc.score(X_test, y_test_cat) )
```

```
Accuracy (with the test data): 0.5490584737363726
```

```
rfc_feature_imp = pd.DataFrame({'Feature': feature_names, 'Gini Importance': demo_rfc.feature
    'Gini Importance', ascending=False)
rfc_feature_imp = rfc_feature_imp.merge(census_vars[["VARIABLE","LABEL"]], how="left", left_
rfc_feature_imp.head(20)
```

	Feature	Gini Importance	LABEL
0	type_protest	0.062471	
1	type_walk-out	0.048473	
2	perc	0.038086	
3	type_march	0.035311	
4	problemname_Social Policy	0.029801	
5	problemname_Public Safety	0.024623	
6	DP04_0051PE	0.009591	Percent!!YEAR HOUSEHOLDER MOVED INTO UNI
7	problemname_Rights	0.007978	
8	type_rally	0.007440	
9	DP04_0052PE	0.006895	Percent!!YEAR HOUSEHOLDER MOVED INTO UNI
10	problemname_Fiscal Policy	0.006775	
11	problemname_Economy	0.006578	
12	problemname_Politics	0.006418	
13	DP04_0054PE	0.006228	Percent!!YEAR HOUSEHOLDER MOVED INTO UNI

	Feature	Gini Importance	LABEL
14	DP04_0055PE	0.005819	Percent!!YEAR HOUSEHOLDER MOVED INTO UNI
15	DP03_0087E	0.005785	Estimate!!INCOME AND BENEFITS (IN 2017 INFLA
16	DP04_0053PE	0.005723	Percent!!YEAR HOUSEHOLDER MOVED INTO UNI
17	DP03_0067E	0.005605	Estimate!!INCOME AND BENEFITS (IN 2017 INFLA
18	DP04_0109E	0.005207	Estimate!!SELECTED MONTHLY OWNER COSTS (S
19	DP03_0094E	0.004954	Estimate!!INCOME AND BENEFITS (IN 2017 INFLA

For the categorical random forest, `types`, `problems`, and `perc` again are found towards the top, but we see entirely different demographic variables: for some reason, the percent of occupied housing units moved into in different decades is significant, as is mean family income and median income for female full-time workers. Perhaps those who are more settled feel more comfortable protesting. As for income, no conclusion can readily be made; it is unclear from importance whether or not the correlation is positive or negative.

K Nearest Neighbors (KNN)

K Nearest Neighbors Regression

```
demo_knnr = KNeighborsRegressor()
param_grid_knn = {'n_neighbors': range(1,31)}
# Cross-validation
grid_knnr = GridSearchCV(demo_knnr, param_grid_knn, cv = kfold, scoring = None, n_jobs=6)
grid_knnr.fit(X_train, y_train_cont)

print( grid_knnr.best_params_ )
print( grid_knnr.cv_results_[('mean_test_score')] )
demo_knnr=grid_knnr.best_estimator_
```

```
{'n_neighbors': 29}
[-6.93281057e-01 -2.92231569e-01 -1.66642711e-01 -1.04816958e-01
 -6.80145105e-02 -4.39656171e-02 -2.74694373e-02 -1.98939267e-02
 -1.33587513e-02 -6.79451491e-04  3.43898735e-03  7.85792948e-03
  1.11042459e-02  1.27858748e-02  1.49968302e-02  1.94702889e-02
  2.08537607e-02  2.06767612e-02  2.20331481e-02  2.29842905e-02
  2.28393239e-02  2.26654688e-02  2.26495905e-02  2.31892275e-02
  2.24552154e-02  2.46215018e-02  2.42134857e-02  2.61699761e-02
  2.62851297e-02  2.59752936e-02]
```

```
print("R^2 (with the test data):", demo_knnr.score(X_test, y_test_cont) )
```

```
R^2 (with the test data): 0.00835429019177314
```

With a Coefficient of Determination of 0.008, KNN is simply not the correct model to use here.

K Nearest Neighbors Classification

```
demo_knnc = KNeighborsClassifier()
# Cross-validation
grid_knnc = GridSearchCV(demo_knnc, param_grid_knn, cv = kfold, scoring = "accuracy", n_jobs=
grid_knnc.fit(X_train, y_train_cat)

print( grid_knnc.best_params_ )
print( grid_knnc.cv_results_[('mean_test_score')] )
demo_knnc=grid_knnc.best_estimator_
```

```
{'n_neighbors': 17}
[0.46555446 0.48777505 0.49397066 0.49818271 0.49545683 0.49917465
 0.50107486 0.5046267  0.50289144 0.50826136 0.50718677 0.50826129
 0.50867397 0.50966492 0.51148218 0.50941739 0.51214231 0.50867379
 0.50569975 0.50594785 0.51049143 0.50784745 0.50974742 0.50916918
 0.51090366 0.50660778 0.50925111 0.50619476 0.50776467 0.50636019]
```

```
print("Accuracy (with the test data):", demo_knnc.score(X_test, y_test_cat) )
```

```
Accuracy (with the test data): 0.49289725801123224
```

Here, KNN is worse than pure chance.

Model Evaluation

Classifier Evaluation

Performance Metrics

```
print("Logit Classification\n", classification_report( y_test_cat, demo_logit.predict(X_test)
print("Logit Classification with PCA\n", classification_report( y_test_cat, pca_logit.predict
print("Decision Tree Classification\n", classification_report( y_test_cat, demo_dtc.predict(X
print("Random Forest Classification\n", classification_report( y_test_cat, demo_rfc.predict(X
print("K Nearest Neighbors Classification\n", classification_report( y_test_cat, demo_knnc.pr
```

```
Logit Classification
               precision    recall  f1-score   support

           1       0.58      0.57      0.57      1291
           2       0.55      0.66      0.60      1428
           3       0.41      0.06      0.11       271
           4       0.00      0.00      0.00        37

    accuracy                           0.56      3027
   macro avg       0.39      0.32      0.32      3027
weighted avg       0.54      0.56      0.54      3027

Logit Classification with PCA
               precision    recall  f1-score   support

           1       0.58      0.56      0.57      1291
           2       0.55      0.62      0.59      1428
           3       0.33      0.17      0.22       271
           4       0.33      0.22      0.26        37

    accuracy                           0.55      3027
   macro avg       0.45      0.39      0.41      3027
weighted avg       0.54      0.55      0.54      3027

Decision Tree Classification
               precision    recall  f1-score   support

           1       0.59      0.73      0.65      1291
           2       0.61      0.57      0.59      1428
           3       0.29      0.07      0.11       271
           4       0.40      0.05      0.10        37

    accuracy                           0.59      3027
   macro avg       0.47      0.36      0.36      3027
weighted avg       0.57      0.59      0.57      3027
```

```
Random Forest Classification
              precision    recall  f1-score   support

           1       0.55      0.56      0.56      1291
           2       0.55      0.65      0.60      1428
           3       1.00      0.00      0.01       271
           4       0.00      0.00      0.00        37

    accuracy                           0.55      3027
   macro avg       0.52      0.31      0.29      3027
weighted avg       0.58      0.55      0.52      3027

K Nearest Neighbors Classification
              precision    recall  f1-score   support

           1       0.47      0.51      0.49      1291
           2       0.51      0.59      0.55      1428
           3       0.00      0.00      0.00       271
           4       0.00      0.00      0.00        37

    accuracy                           0.49      3027
   macro avg       0.25      0.27      0.26      3027
weighted avg       0.44      0.49      0.47      3027
```

Recall of the biggest demonstration sizes is more important here, as it is better to allocate resources where they are not needed than to not when they are necessary. As such, the best decision tree classifier, with the highest accuracy and the second-best recall of Classes 3 and 4, looks to be the second-best model; although the logistic regression with PCA resulted in slightly worse accuracy than without it and than the best decision tree classifier, due to its better results for those larger crowd sizes, that should be taken as the best model for classification.

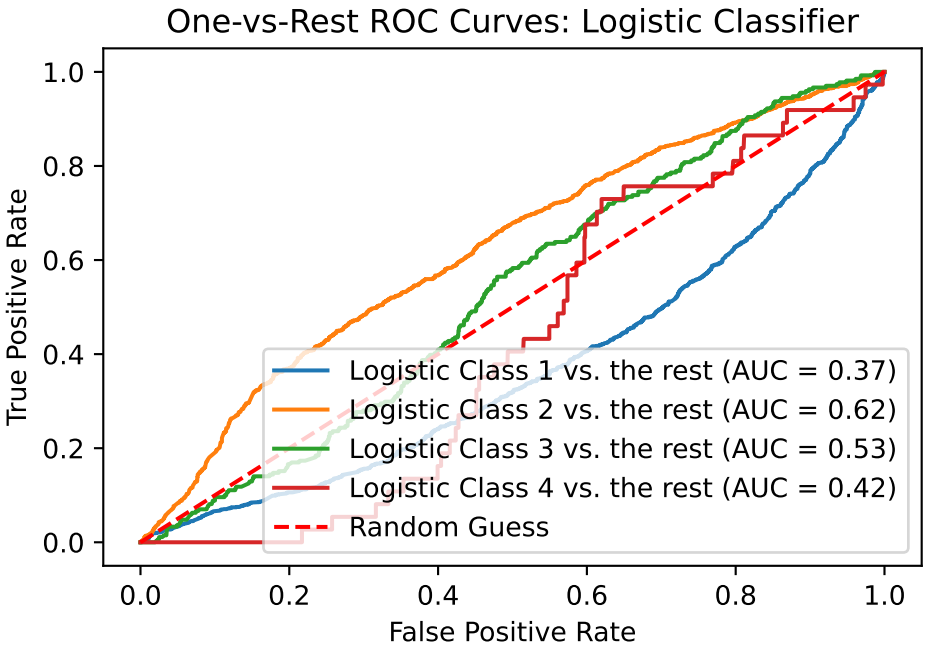
One vs. Rest multiclass ROC-AUC curves

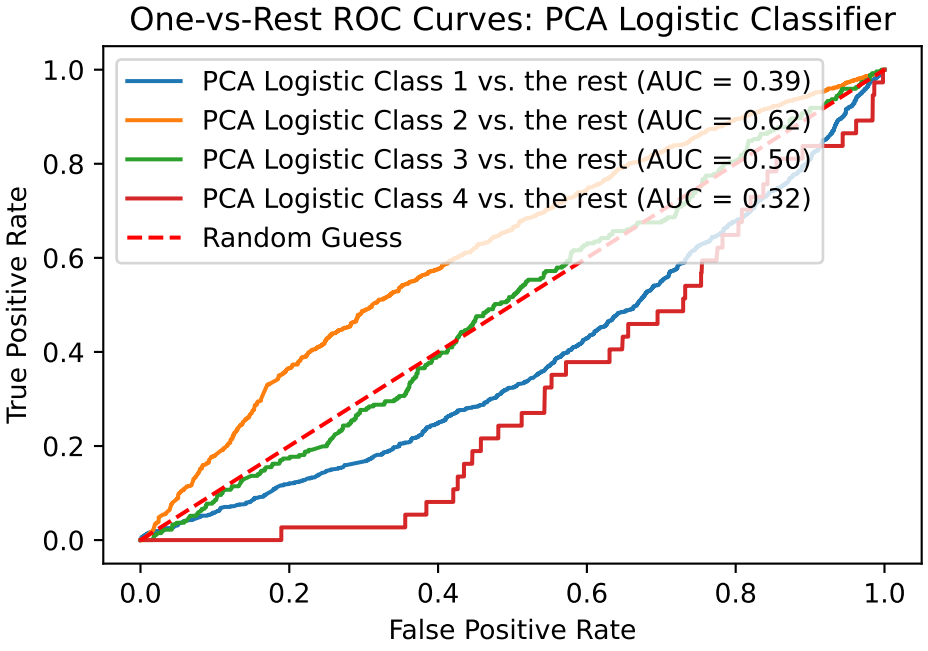
ROC-AUC can only be applied to binary classifiers. As such, a One vs. Rest strategy may be used, plotting multiple curves: one for each class, where a failure is a classification of any of the others.

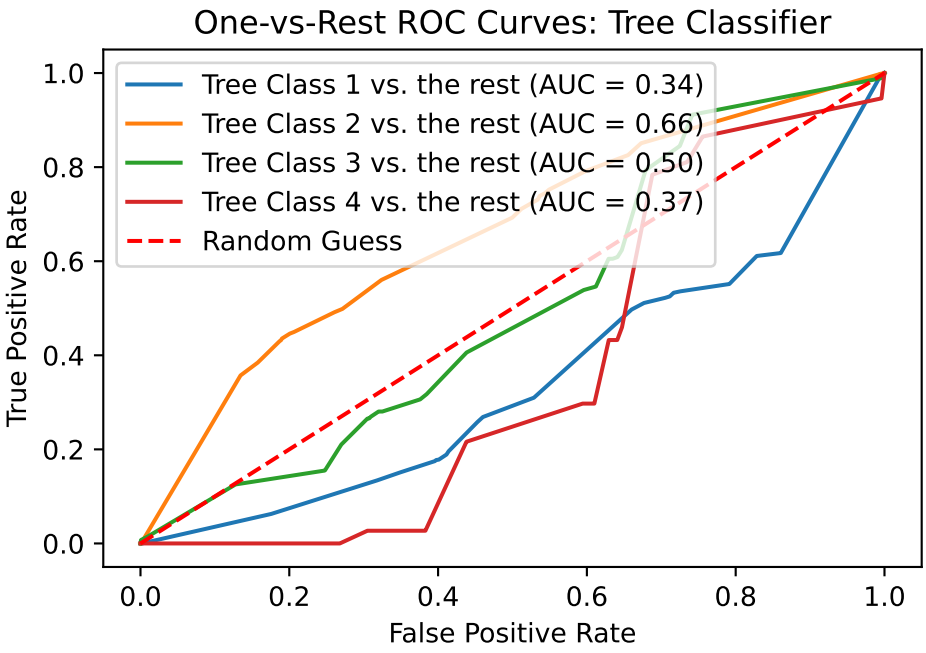
```
label_binarizer = LabelBinarizer().fit(y_train_cat) # sourced from https://scikit-learn.org/s
y_test_cat_onehot = label_binarizer.transform(y_test_cat)
# y_test_cat_onehot.shape  # (n_samples, n_classes)

class_names=y_test_cat.unique().sort_values()
```

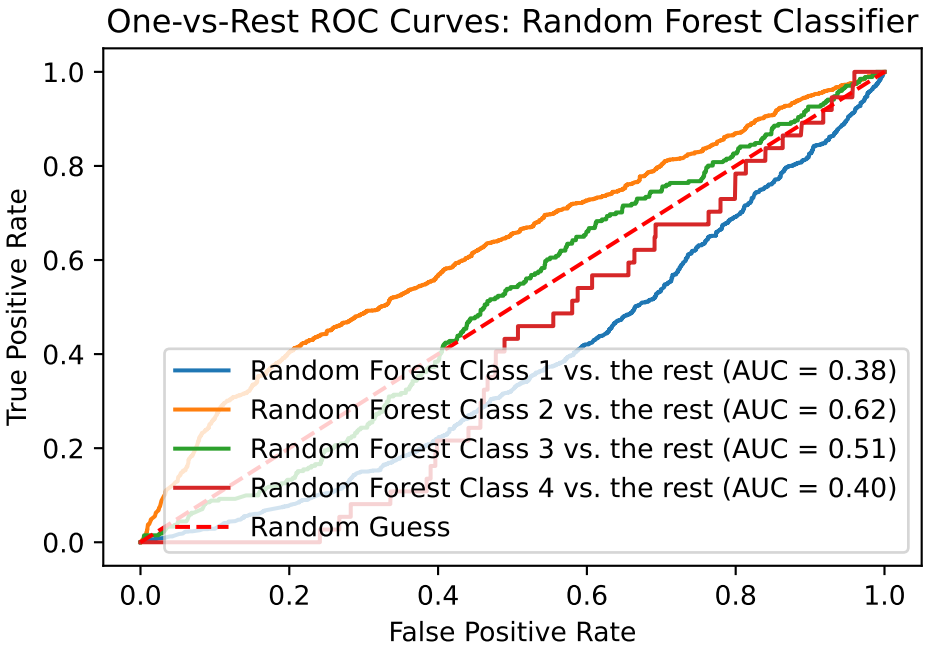
```
def OvR_ROC_AUC(X_test, y_test_onehot, cat_model, cat_model_name, classes, class_names):
    for class_of_interest in classes:
        class_id = np.flatnonzero(label_binarizer.classes_ == class_of_interest)[0]

        fpr, tpr, _ = roc_curve(y_test_onehot[:, class_id], cat_model.predict_proba(X_test)[
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f"{cat_model_name} Class {class_names[class_id]} vs. the res

    plt.plot([0, 1], [0, 1], 'r--', label='Random Guess')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'One-vs-Rest ROC Curves: {cat_model_name} Classifier')
    plt.legend()
    plt.show()

OvR_ROC_AUC(X_test, y_test_cat_onehot, cat_model=demo_logit, cat_model_name="Logistic", class
OvR_ROC_AUC(X_test_pca, y_test_cat_onehot, cat_model=pca_logit, cat_model_name="PCA Logistic"
OvR_ROC_AUC(X_test, y_test_cat_onehot, cat_model=demo_dtc, cat_model_name="Tree", classes=lab
OvR_ROC_AUC(X_test, y_test_cat_onehot, cat_model=demo_rfc, cat_model_name="Random Forest", cl
OvR_ROC_AUC(X_test, y_test_cat_onehot, cat_model=demo_knnc, cat_model_name="KNN", classes=lab


# # Logit
# fpr, tpr, _ = roc_curve(y_test_cat, demo_logit.predict_proba(X_test)[:, 1])
# roc_auc = auc(fpr, tpr)
# plt.plot(fpr, tpr, label=f'Logistic (AUC = {roc_auc:.2f})')
# # Decision Tree
# fpr, tpr, _ = roc_curve(y_test_cat, demo_dtc.predict_proba(X_test)[:, 1])
# roc_auc = auc(fpr, tpr)
# plt.plot(fpr, tpr, label=f'Tree (AUC = {roc_auc:.2f})')
# # Random Forest
# fpr, tpr, _ = roc_curve(y_test_cat, demo_rfc.predict_proba(X_test)[:, 1])
# roc_auc = auc(fpr, tpr)
# plt.plot(fpr, tpr, label=f'Random Forest (AUC = {roc_auc:.2f})')
# # KNN
# fpr, tpr, _ = roc_curve(y_test_cat, demo_knnc.predict_proba(X_test)[:, 1])
# roc_auc = auc(fpr, tpr)
# plt.plot(fpr, tpr, label=f'KNN (AUC = {roc_auc:.2f})')

# plt.plot([0, 1], [0, 1], 'r--', label='Random Guess')
# plt.xlabel('False Positive Rate')
# plt.ylabel('True Positive Rate')
```

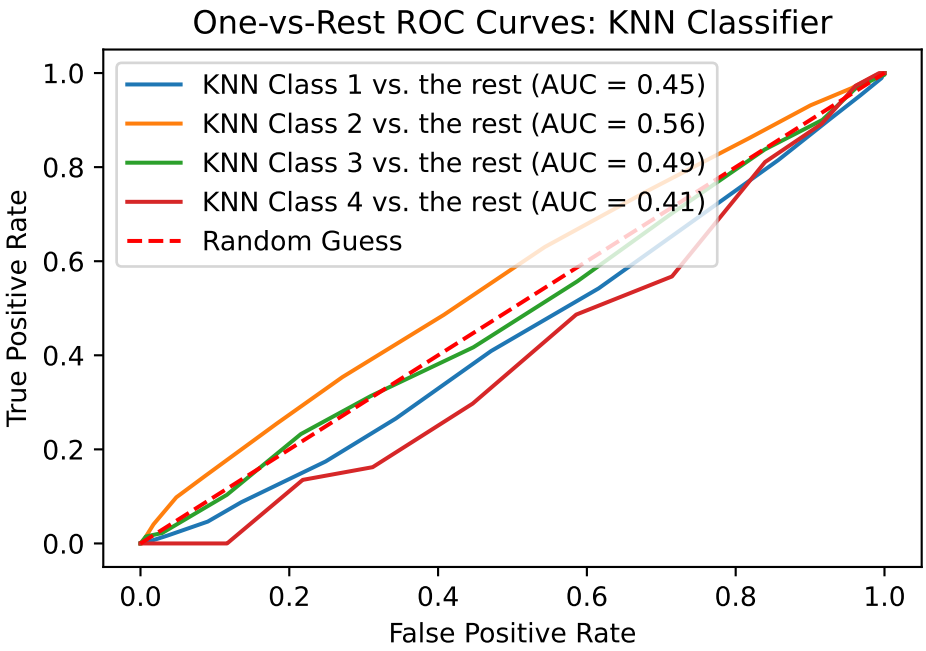
```
# plt.title('ROC Curves for Two Models')
# plt.legend()
# plt.show()
```






One-vs-Rest ROC Curves: Tree Classifier
Tree Class 1 vs. the rest (AUC = 0.34)
Tree Class 2 vs. the rest (AUC = 0.66)
Tree Class 3 vs. the rest (AUC = 0.50)
Tree Class 4 vs. the rest (AUC = 0.37)
Random Guess
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0


One-vs-Rest ROC Curves: Random Forest Classifier
Random Forest Class 1 vs. the rest (AUC = 0.38)
Random Forest Class 2 vs. the rest (AUC = 0.62)
Random Forest Class 3 vs. the rest (AUC = 0.51)
Random Forest Class 4 vs. the rest (AUC = 0.40)
Random Guess
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0



Judging by the ROC-AUC curves, the best decision tree classifier would appear to be the best model. It is the worst at predicting Class 1s, but as that is the smallest class, it is the least essential to distribute resources appropriately for.

It is worth comparing the logistic regression with the full dataset vs. the one flattened by PCA. Despite PCA being useful in the case of linear regression, for logistic regression, it instead produces worse results here for Classes 3 and 4, contradicting the results from the classification tables above.

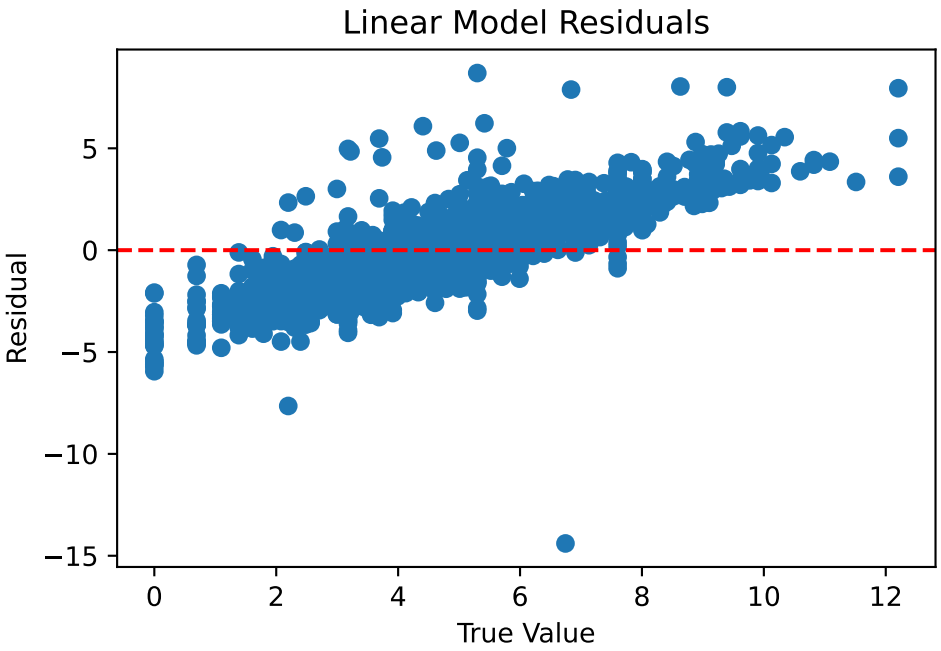
Between these two evaluations, the best decision tree classifier could be judged the best classification model.

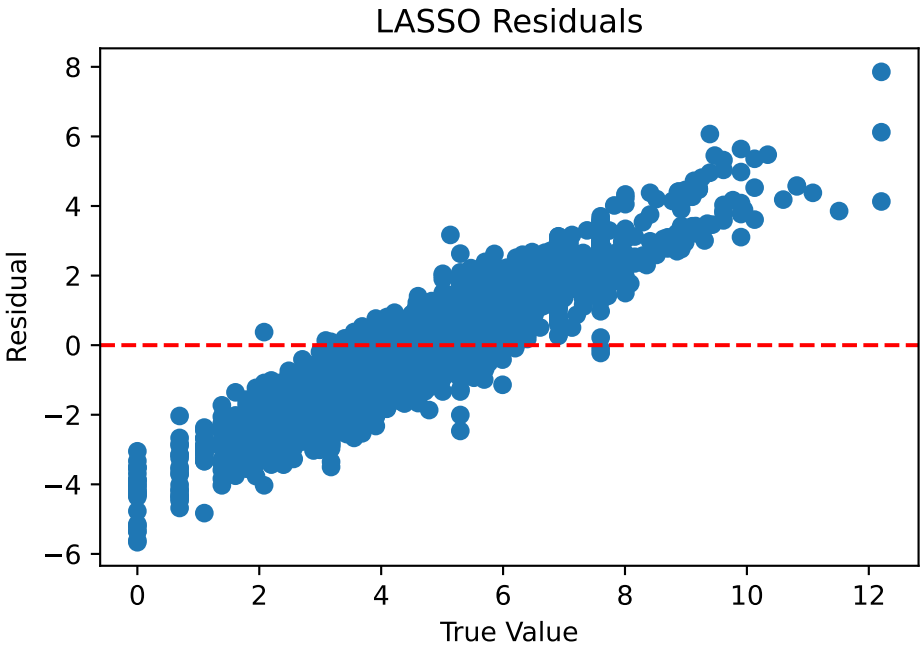
Regressor Evaluation

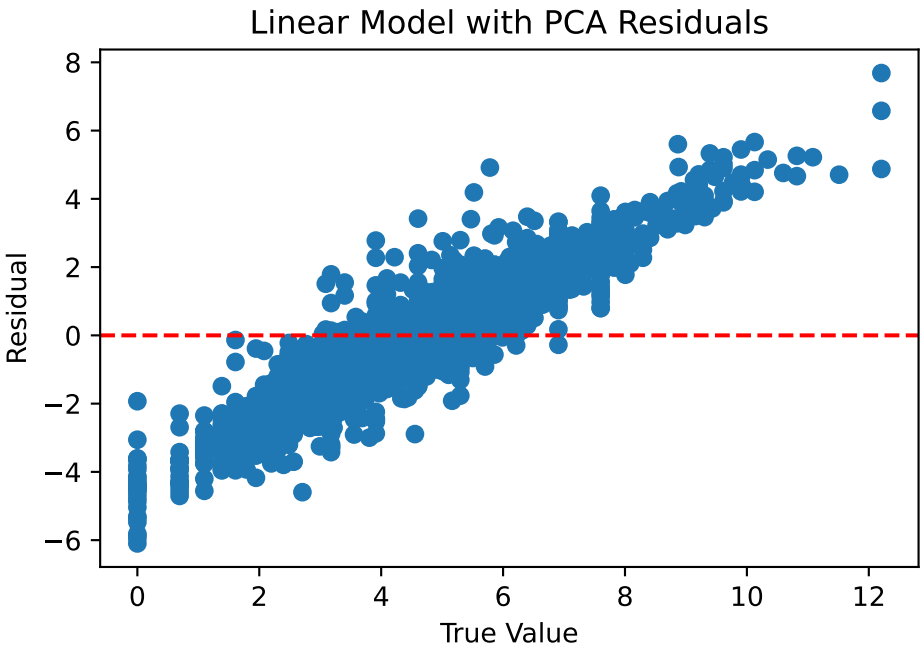
Residuals

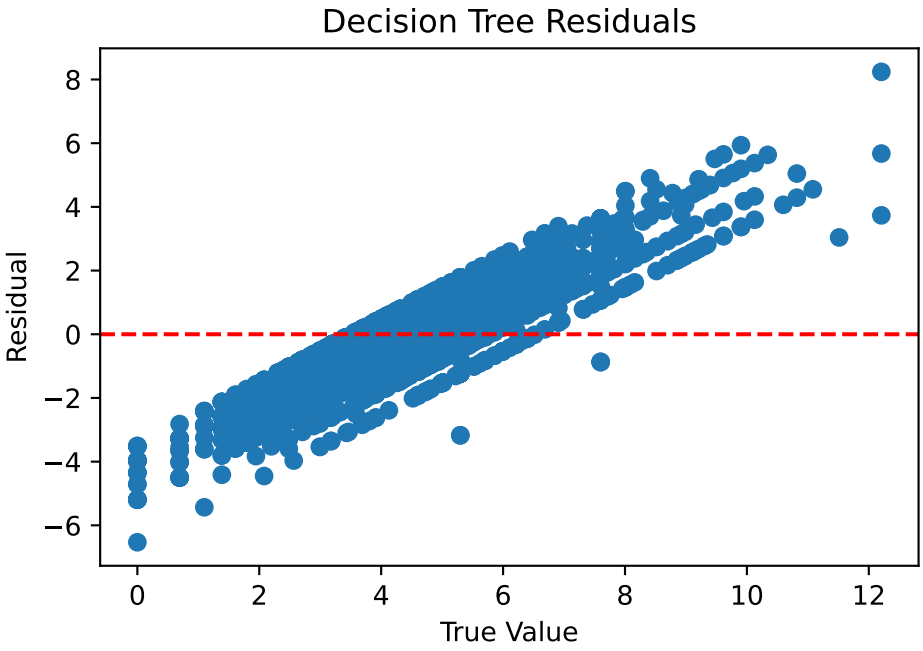
```
# linear model
plt.scatter(y_test_cont, y_test_cont - demo_lm.predict(X_test))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("Linear Model Residuals")
```

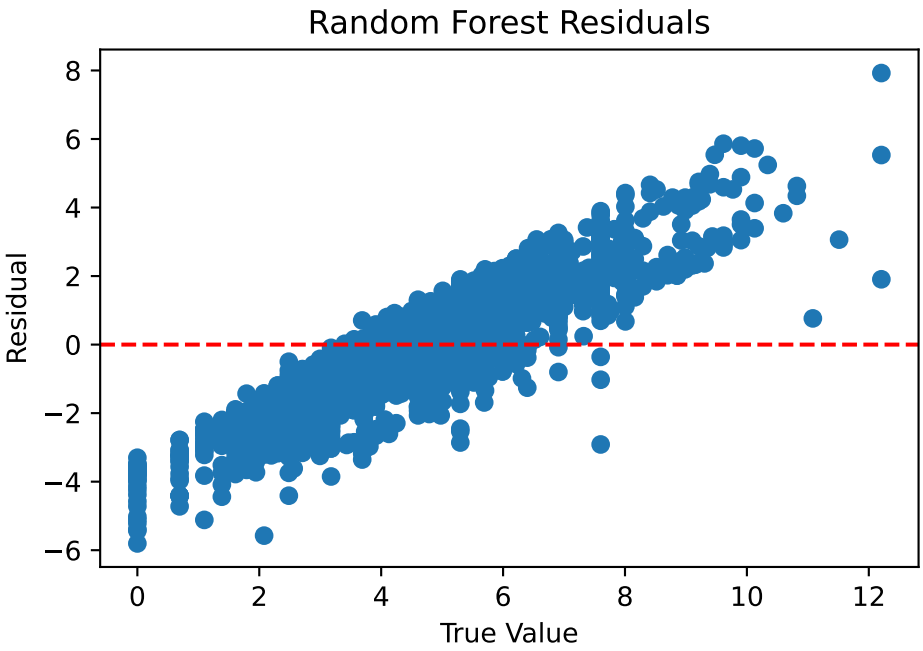
```
plt.show()

# LASSO
plt.scatter(y_test_cont, y_test_cont - demo_lasso.predict(X_test))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("LASSO Residuals")
plt.show()

# linear model with PCA
plt.scatter(y_test_cont, y_test_cont - pca_lm.predict(X_test_pca))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("Linear Model with PCA Residuals")
plt.show()

# tree
plt.scatter(y_test_cont, y_test_cont - demo_dtr.predict(X_test))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("Decision Tree Residuals")
plt.show()

# random forest
plt.scatter(y_test_cont, y_test_cont - demo_rfr.predict(X_test))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("Random Forest Residuals")
plt.show()

# knn
plt.scatter(y_test_cont, y_test_cont - demo_knnr.predict(X_test))
plt.axhline(y=0, color='r', linestyle='--')
plt.xlabel("True Value")
plt.ylabel('Residual')
plt.title("K Nearest Neighbors Residuals")
plt.show()
```

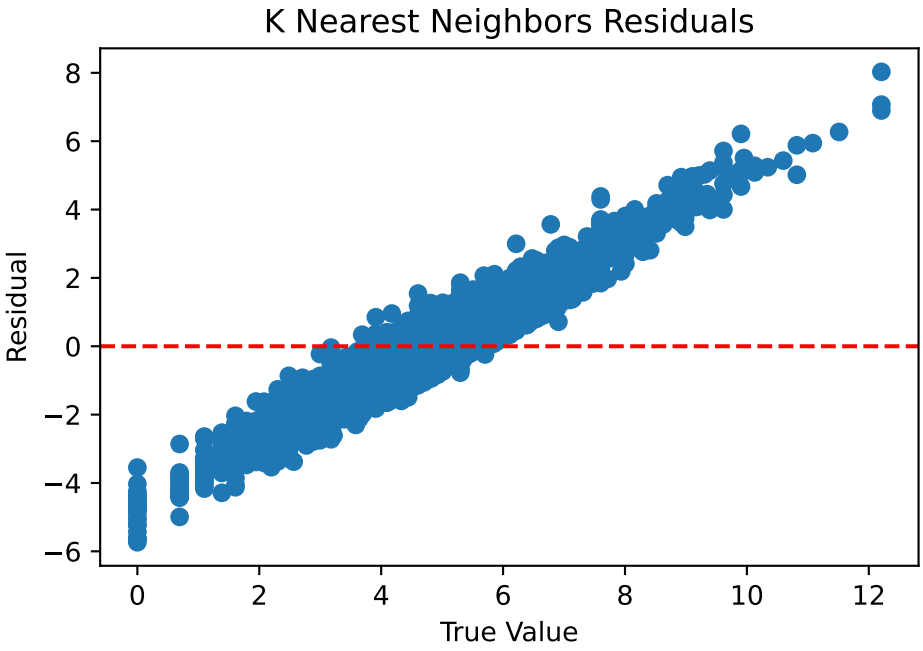

Linear Model Residuals
Residual
True Value
5
0
−5
−10
−15
0
2
4
6
8
10
12


LASSO Residuals
Residual
True Value
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12


Linear Model with PCA Residuals
Residual
True Value
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12


Decision Tree Residuals
Residual
True Value
8
6
4
2
0
−2
−4
−6
0
2
4
6
8
10
12





The residuals all have a clear direction. The linear model appears flattest, but only because it is squashed to show all of the residuals.

Coefficients of Determination

```
print("R^2 With the Test Data")
print(f"Linear Model: {demo_lm.score(X_test,y_test_cont):.2f}")
print(f"LASSO: {demo_lasso.score(X_test,y_test_cont):.2f}")
print(f"Linear Model with PCA: {pca_lm.score(X_test_pca,y_test_cont):.2f}")
print(f"Decision Tree: {demo_dtr.score(X_test,y_test_cont):.2f}")
print(f"Random Forest: {demo_rfr.score(X_test,y_test_cont):.2f}")
print(f"K Nearest Neighbors: {demo_knnr.score(X_test,y_test_cont):.2f}")
```

```
R^2 With the Test Data
Linear Model: 0.04
LASSO: 0.17
Linear Model with PCA: 0.08
Decision Tree: 0.18
Random Forest: 0.22
K Nearest Neighbors: 0.01
```

The best Random Forest model explains the most variance. The linear model with PCA explains twice the variance of the linear model without it, though LASSO did better in the category of dimensionality reduction. The KNN regressor, meanwhile, barely accounts for anything.

Based on these results, the best random forest regressor may be judged the best regression model.

Conclusions

At the beginning of this project, it was unknown whether or not the SMART question at hand could be answered. Now, at its end, though success was middling, success was found nonetheless. It is, indeed, possible to predict demonstration size with some success based on salience (as represented through the question of the Most Important Problem) and demographic information in the years 2017 and 2018.

Some of the important demographic variables were expected: larger crowds were associated with higher percents of the popualation being of retirement age and/or unemployed. Others were less so: different levels of educational attainment repeatedly showed that, the more educated a populace is, the larger demonstrations are likely to be. This may be due to the cultural influence of university, but there are many hypotheses that could be made. For others, their absence is intriguing: total population was never close to the top, whereas it may naively be assumed to be the most important factor.

Lastly, the abject failure of K Nearest Neighbors is notable. Perhaps it is due to the fact that, though protesters may gather in the same places, that is no indication of that gathering's size.

Next Steps

Several ideas present themselves. The different types of crowds were the strongest predictor, but there is a chance that they are being assigned post-hoc by the journalists themselves. For instance, perhaps a small crowd is deemed a demonstration, whereas a large one is labeled a protest. If that is the case, such information would be useless in predicting the size of crowds in advance; these models should be redone without those factors.

Interaction terms would be another avenue to pursue: for instance, year by problem or problem by county. Such would necessitate dimensionality reduction or a non-parametric model; but, as those are already the most successful models, this should not be a problem, though the increased compute time may be.

Additionally, the Most Important Problem may be broken down further. These categories are general, but the second version of the dataset extends to 110 different sub-categories ("The 'Most Important Problem' Dataset (MIPD): A New Dataset on American Issue Importance," n.d.; Yildirim & Williams, 2025); such analysis would also be interesting to perform.

Most importantly, however, such information is quite delayed. The original interest in this subject came from the protests this year; 2018 was 7 years ago, and, importantly, pre-COVID-19. If predictions are to be made in any form of reasonable time period, more recent data must be used as a proxy for salience; for instance, data from Google Trends could be an interesting avenue to go down.

References

Carey, Jr., T. E., Branton, R. P., & Martinez-Ebers, V. (2014). The Influence of Social Protests on Issue Salience among Latinos. *Political Research Quarterly, 67*(3), 615–627. https://doi.org/10.1177/1065912914534074

Chenoweth, E., Pressman, J., & Ulfelder, J. (2025). *Crowd counting consortium U.S. protest event data, 2017-2020.* https://doi.org/10.7910/DVN/6OPP7H

Crowd Counting Consortium. (2024, May 30). Ash Center. https://ash.harvard.edu/programs/crowd-counting-consortium/

Heffington, C., Park, B. B., & Williams, L. K. (2017). The "Most Important Problem" Dataset (MIPD): a new dataset on American issue importance. *Conflict Management and Peace Science, 36*(3), 312-335. https://doi.org/10.1177/0738894217691463 (Original work published 2019)

scikit-learn. (2025). *Glossary of common terms and API elements.* Scikit-Learn. https://scikit-learn.org/stable/glossary.html#term-cross-validation-estimator

Somma, N. M., & Medel, R. M. (2019). What makes a big demonstration? Exploring the impact of mobilization strategies on the size of demonstrations. *Social Movement Studies, 18*(2), 233–251. https://doi.org/10.1080/14742837.2018.1532285

The 'Most Important Problem' Dataset (MIPD): A New Dataset on American Issue Importance, Release 1.0 REPLICATION, 1939 [Dataset]. Roper #31094159, Version 2. Not applicable [producer]. Cornell University, Ithaca, NY: Roper Center for Public Opinion Research [distributor]. doi:10.25940/ROPER-31094159

U.S. Census Bureau. (2017). *American Community Survey 1-year estimates: Data Profiles* [Data set]. https://api.census.gov/data/2017/acs/acs1/profile (Accessed December 12, 2025).

U.S. Census Bureau. (2018). American Community Survey 1-year estimates: Data Profiles [Data set]. https://api.census.gov/data/2017/acs/acs1/profile (Accessed December 12, 2025).

Yildirim, T. M., & Williams, L. K. (2025). Problem importance across time and space: updating the "Most Important Problem Dataset." *Journal of Elections, Public Opinion and Parties, 35*(4), 517–532. https://doi.org/10.1080/17457289.2024.2337424