TARGETED POLITICAL ADS ON FACEBOOK: MODELING RUSSIA'S 2016 ELECTION INTERFERENCE

FINAL PROJECT SUBMISSION: COMPUTER SCIENCE 105, PRIVACY AND TECHNOLOGY

Tejal Patwardhan
Harvard University
tejal_patwardhan@college.harvard.edu

December 13, 2019

ABSTRACT

Facebook is facing increasing scrutiny for its lax regulation of political advertisements. Critics worry about the potential of using personal information on Facebook to target voters in ways that undermine our democracy, citing Russian interference in the 2016 U.S. presidential election as an example of this threat. This project seeks to quantitatively understand the effectiveness of Russia's 2016 election Facebook advertisement campaign and determine the importance of targeting with personal information. Using a government release of Russian Facebook advertisements, I trained and validated supervised machine learning methods to select the best predictive model of advertisement effectiveness. The best model, a random forest model, predicts ad effectiveness in a held out test set with an R^2 of over 85%. Further analyzing the most important features in ad effectiveness yields that the most effective ads do not require much effort or money. This analysis demonstrates how vulnerable the public would be to a similar Facebook interference strategy for future political cycles.

***K*eywords** Facebook · 2016 United States presidential election · Internet Research Agency · election interference

1 Background

Facebook faces mounting pressure to reform its political advertisement policies. Recent scrutiny began after Facebook refused to take down an October 2019 Trump re-election advertisement which made false claims about Democratic presidential hopeful Joe Biden [1]. In response, Elizabeth

Warren, a prominent critic of Facebook, ran her own targeted advertisements falsely claiming Mark Zuckerberg endorsed Trump's re-election [2]. As criticism brewed, in late October, Congress questioned Zuckerberg about his plan to address the implications of Facebook's lax political advertisement monitoring. He said, "in a democracy people should be able to see for themselves what politicians are saying," explaining his company's position to not regulate political advertisements [1]. On November 15, 2019, Twitter's ban of political advertisements went into effect, a move that placed Facebook's decision to not regulate advertisements under further scrutiny.

Understanding why politicians are so concerned about Facebook's political advertisement policies requires understanding what happened in the 2016 United States Presidential election. Russian interference in the election stoked fear about the underlying health of the U.S. democratic process. On February 16, 2018, Special Counsel Robert Mueller announced the indictment of the Internet Research Agency (IRA) for 2016 presidential election interference [3]. The IRA is a Russian online influence company with known ties to Vladimir Putin and Russian intelligence. The Grand Jury for the District of Columbia charged them for violating criminal laws with intent to interfere with U.S. elections and political processes. The indictment included evidence of the IRA's explicit goal to "spread distrust towards the candidates and the political system in general" [3]. A leaked document revealed that IRA specialists were instructed to post content that focused on "politics in the USA" and to "use any opportunity to criticize Hillary and the rest (except Sanders and Trump—we support them)" [3]. The report goes further, tying social media campaigns in particular to these foreign interference operations. The indictment report notes that in 2015, the IRA "began to purchase advertisements on online social media sites to promote IRA-controlled social media groups, spending thousands of U.S. dollars every month." [3]. More importantly, "To measure the impact of their online social media operations, defendants and their co-conspirators tracked the performance of content they posted over social media, [including] size of the online U.S. audiences reached through posts" [3]. This evidence explains that the IRA certainly were involved in political advertising on social media and with a mission to undermine democracy. Because the IRA themselves modeled engagement, it's interesting to determine what they found to be an effective interference strategy. This project involved creating a model for the effectiveness of Russia's 2016 election Facebook advertisement campaign and determine the importance of targeting with personal information.

Question: How to design an effective election interference strategy using targeted Facebook advertisements?

2 Methods

I built a series of supervised machine learning models to predict advertisement effectiveness. To proxy advertisement effectiveness, I used the number of advertisement clicks as the outcome variable. An advertisement click occurs when an individual sees an advertisement and doesn't just scroll past it, but instead actively engages with the linked media (e.g. goes to the page sponsoring the advertisement). Clicks are a good proxy for whether the advertisement was powerful enough to make an individual act on that information. The IRA actually used advertisement clicks as a metric to optimize their advertisement campaigns, which is why I was particularly interested

in predicting this metric [3]. One reason for this is because the actual votes of individuals are anonymous and unverifiable, while clicks can be attributed to particular people for particular advertisements. Similarly, most individuals see so many advertisements that that the effect of particular advertisement on a vote would be hard to determine even if we had individual voter data. Because of these problems with using votes as an outcome variable, clicks are considered the most generalizable metric to determine the effectiveness of a particular advertisement [4]. Below I walk through how I collected the data to predict ad effectiveness, what tools I used to process the data, which models I constructed, and how I evaluated my models.

2.1 Data

In May of 2018, the Democratic Minority in the U.S. House of Representatives Permanent Select Committee on Intelligence released all Facebook advertisements linked to the Internet Research Agency [5]. This data includes 3.5 thousand individual Facebook advertisements paid for by the IRA, as were exposed to over 11.4 million people. The data were released as a series of PDFs, so were converted into JSON and then CSV format for use in this analysis. For each advertisement, disclosed information included the full text of that advertisement, attached media or links, location targeting, interest targeting, amount paid, dates of advertisement campaign, and ad effectiveness metrics (e.g. number of impressions, number of clicks).

2.1.1 Exhibit

Below is a representative exhibit of what the data for an ad looks like. This is a November 2016 IRA-sponsored Facebook advertisement targeted towards African-Americans, which urges individuals to vote for third-party candidate Jill Stein. Below is the JSON listing for this advertisement as well as the attached image of what the advertisement looked like, which were extracted from the Permanent Select Committee on Intelligence release.

```
{
"language_categories": ["English"],
"placement_categories": ["Facebook"],
"interests_categories": ["African American"],
"location_categories": [],
"ad_id": 1183.0,
"ad_copy": "Choose peace and vote for Jill Stein.
Trust me, it's not a wasted vote.",
"ad_landing_page": "www.facebook.com/Blacktivist-128371547505950/",
"ad_targeting_location": "United States",
"interest_expansion": "Pan-Africanism, African-American, Civil Rights
Movem(1954-68), African-American history or Black (Color)",
"age_lower": 16.0,
"age_upper": 999,
"age": "16-65+",
"language": "English (UK)or English (US)",
"placements": "News Feed on desktop computers or
News Feed on mobile devices ",
"ad_impressions": 18888.0,
"ad_clicks": 1635.0,
"ad_spend_rub": 500.0,
"ad_creation_date": "11/03/2016",
"month": ["11"],
"year": ["2016"],
"day": "03",
"ad_end_date": "11/10/2016",
"pdf_filepath": "2016-11/P10003004.pdf",
"image_filepath": "2016-11/P10003004.-000.png",
"excluded_connections": null,
"ad_spend_usd": 7.84,
"efficiency_impressions": 2409.18,
"efficiency_clicks": 208.55,
"conversion_rate": 0.09,
"interests_categories_regex": "African American",
"location_categories_regex": "",
"date_order_index": 2166
}
```


Listing 1: JSON exhibit

Figure 1: Corresponding advertisement image for JSON exhibit

This advertisement links to a page urging a vote for Jill Stein. Among the features given for this particular advertisement, one can notice the interest in African American issues, including interests in "Pan-Africanism, African-American, Civil Rights Movement, African-American history or Black (Color)." These indicate that this ad is targeted towards African-American supporters, presumably a group of people that would have been more likely to vote for Clinton than Trump. By encouraging these would-be Clinton voters to vote for Jill Stein, the IRA could further its purpose of promoting the election prospects of Trump. This strategy of targeting voters to sow political distrust and promote Trump's election chances are shared across advertisements in the dataset. All features listed in the JSON file for this advertisement exhibit, including interest and location targeting, were given for all advertisements in the dataset, and used in my modeling.

2.1.2 Feature Engineering

I also did some feature engineering to increase the size of the feature set. This included scouring the ad text itself for key words (e.g. mentions of "Trump", "vote", and "viral"). I also appended a feature for post length, which was based on the number of characters in the post. Another appended feature was length of campaign. Other features added include binary variables for location targeting in particular states (e.g. "Pennsylvania") and particular languages (e.g. "Spanish" and Arabic").

2.2 Tools

The data were analyzed in Python primarily using the following packages:

- Data handling: pandas, numpy
- Models: scikit-learn
- Visualization: matplotlib, seaborn

2.3 Model evaluation

To evaluate which model was most accurate, the data were randomly split 80/20 into a training dataset and a representative held out test set. All models were trained and tuned only using the training set, and later were evaluated in the test set. R^2 scores in the test set were compared across models to determine which model could most accurately predict data in the holdout test set.

2.4 Models

Ad effectiveness metrics were predicted using supervised machine learning methods, including a linear regression, decision tree, boosted decision tree, and random forest.

- The first model tried was an ordinary least squares (OLS) linear regression. This is a parametric regressor that assigns coefficients to each of our features and constructs a prediction as a linear combination of those weighted features. The linear regression was L2 regularized.
- The next model is a decision tree. This is a nonparametric regressor that can better account for nonlinear relationships between our features and our outcome variable [6]. It breaks our data into subsets by constructing trees with decision nodes that cut across our features (like location, interests, and age). I tuned the hyperparameters using cross-validation in the training set.

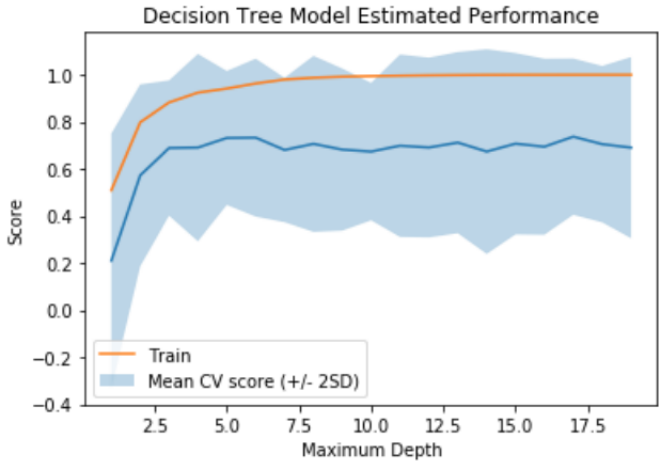

Figure 2: Plot for tuning maximum depth of decision tree

At a maximum depth of 5, the cross-validation score plateaus, so a maximum depth 5 was set as the optimal depth for the decision tree.

- Next, a boosted decision tree was created. Boosting is an ensemble method that builds on a decision tree. Trees are constructed iteratively to address the errors in the previous trees [7]. This involves building a decision tree from the training data, and then adding on decision trees that attempt to fix the errors from the previous trees. I tuned the hyperparameters using cross-validation in the training set.

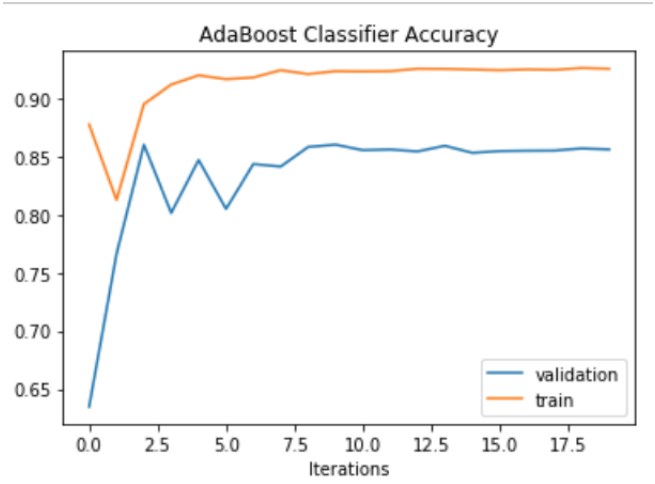

Figure 3: Plot for tuning number of iterations

At 10 iterations, the cross-validation score plateaus, so an iteration number of 10 was used for the boosted decision tree model.

- Finally, a random forest model was created. A random forest is another ensemble method to address overfitting of decision trees. For each tree, a subset of features are selected, and a

decision tree is fit to just that subset of features. The random forest then averages over all the constructed trees to create a more generalizable model than a single tree.

These models were all trained and tuned in the training set, and then evaluated in the test set.

3 Results

3.1 Determining most accurate model

Below are the R^2 values for each of our models in both the training and the held-out test sets.

Model	Train R^2	Test R^2
Linear Regression	0.8534	0.7453
Decision Tree	0.9632	0.5781
Boosted Decision Tree	0.9152	0.7568
Random Forest	0.9755	0.8654

The best model was the random forest model, which had the highest R^2 in the test set. I used accuracy in the test set to determine which model was best, because the model that performs best in the training may just be overfit to the particular training points. Since the purpose of this model is to predict how effective a theoretical advertisement for a future campaign would be, I was more interested in the accuracy in the test set as a proxy for how generalizably accurate the models were. The random forest had an R^2 in the test set of 0.8654, which means it could capture over 85% of the variation in the held-out test set. If we were to just predict the mean number of advertisement clicks, the R^2 would be 0, so having this high of an R^2 in the test set indicates that we have accurately picked up on some generalizable signal. This leads us to **takeaway 1:** it is possible to very accurately predict the exact number of advertisement clicks generated for a particular ad, with an R^2 of over 85%.

3.2 Most important features of an effective advertisement

We then used recursive feature elimination (RFE) to identify which features were most critical in determining advertisement effectiveness. Below is the ranking of the top 10 most important features in determining advertisement effectiveness.

1. Amount spent
2. Campaign length
3. Time until election
4. Length of advertisement
5. Lower age limit
6. Interest in African-American issues
7. Post content including word “Trump”

8. Political leaning
   (a) Interest in conservative issues
   (b) Interest in progressive issues
9. Hyper-targeting (requiring >3 targeted interests to display post)

It is relatively intuitive to understand why the listed features are important. The first two most important features make sense because they both correlate with how many impressions an advertisement received (even though number of impressions was not in the model, amount spent and campaign length are both direct components of number of impressions). Users cannot click on an advertisement if they do not see it, which is why spending more money to show the advertisement to more people and running the advertisement for longer is associated with more clicks. The rest of the features revealed are more interesting, because they show what matters after controlling for the effects of increased numbers of impressions. Considering time until the election, some political strategists claim that campaigns should "front-load" their advertisements early in an election cycle to get early support [9]. While that may be true, this analysis indicates that voters are much more likely to click on advertisements closer to the election. Examining length of the advertisement, shorter advertisements fared much better than longer ones. Keep it quick and people will click. They want to read more if you don't give away everything right away. Finally, what makes Facebook special, its ability to tailor advertisements using personal data, also mattered quite a bit. Tailoring by age, interest in African-American issues, and political leaning, both conservative and progressive, made users much more likely to click. Using multiple forms of targeting was also an important feature in increasing clicks.

Below is a sampling of the least important features in determining advertisement effectiveness, which had no bearing on effectiveness whatsoever.

- Location targeting, including by individual state
- Language, including Arabic and Spanish
- Using sensationalist content identifiers, including "viral" (e.g. "let's make this go viral!") and exclamation points, (e.g. "Justice!!!")
- Including videos as opposed to pictures or just text

Combining this with the earlier analysis reveals that the most effective campaign involves well-timed, short pieces of content. Tailoring can be simple, just based on age, interest in race, and political leaning. There is no need for a campaign to bother with sensationalist language, putting effort into using videos as opposed to links, or focusing on in-depth or quality content. This leads to **takeaway 2**: it's relatively easy to create effective political interference advertisements on Facebook. Strategies that take time, like focusing on video or writing a lot of content, do not affect advertisement effectiveness that much. The only feature that slightly contradicts this is that it also always helps to spend money and keep the campaign going longer so more people see the ad, which is the gateway to clicks. However, given how relatively cheap targeted Facebook advertisements already are, the features that make each dollar go further (like targeting) make Facebook particularly susceptible to abuse. The next section considers the financial cost in more depth.

3.3 Analyzing cost

Looking deeper into the advertisement spend features revealed another interesting result, which is **takeaway 3**: effective political interference advertisements can be super cheap. The entire advertisement campaign conducted by the IRA cost less than $\$95$k, and yet they still managed to reach over 11 million Americans [3]. Diving deeper into the data, each dollar spent acquired over 437 views and 40 clicks on average. This translates into a cost of $\$0.002$ per view and $\$0.025$ per click. Compared to the average cost per click for traditional advertisements, this is incredible return-on-investment. For traditional advertisement campaigns, the cost per click and impression is an order of magnitude larger, costing over $\$0.20$ on average per click, for example [8]. It is possible that the level of interest in the 2016 U.S. presidential election combined with the inflammatory nature of these posts made them much more likely to be clicked. However, it is also likely that the IRA's continuous optimization for clicks, using models like the ones I built here, helped make their campaign so cheaply successful.

4 Future Work

There are a number ways to improve this analysis and future avenues to explore.

- Try alternative outcome variables. Because the ultimate goal of an election campaign is to affect votes (for the IRA, this meant either suppressing votes or swinging a vote for a particular candidate), trying different proxies to measure effect on vote could be useful.
  - Consider different efficiency metrics, including conversion rate and quality of interaction (for example, separate likes from hearts from angry reactions), and replicate this analysis with new outcome variables. These outcome variables may come closer to actually determining whether someone will vote.
  - Consider targeted outcome variables, e.g. best conversion rate for specific political group. If the goal of the campaign is to suppress just the vote of progressives, for example, the prediction that might be more useful is one tailored by individual group.
- Improve the data used to model.
  - Increase size (number of rows) of dataset. Because this dataset is still relatively small, some of the models may not be training as well as they theoretically could. It would be useful to have access to the 80K posts associated with the IRA to increase the power of the predictions by an order of magnitude, which Facebook has not released. Another way to increase the dataset size is manually correcting the PDF to JSON conversion errors and appending these previously dropped rows to the dataset. Finally, including related advertisements, like from the IRA's Ukraine intervention campaigns, may help if those advertisements share critical characteristics with the 2016 political election dataset.
  - Increase features (number of columns) of dataset. Plenty of features not analyzed here may be relevant to advertisement performance. For example, including sentiment

analysis of the advertisement text could be useful. Similarly, analyzing attached media, like video length, could be useful too.

- Improve the model. Perhaps alternative models can generalize better than the ones I tested (like k-nearest-neighbors regressors, or if our dataset was eventually large enough, a neural network). Similarly, better regularizing our existing models may help them improve test set performance even with the existing dataset.
- Combine with qualitative features. For example, non-advertisement-related effects, like current poll numbers, probably also sway the effectiveness of an advertisement. Similarly, the content that is galvanizing a political base at a given time, be it gun control legislation or economic policy, may also influence the effectiveness of an advertisement.

5 Conclusion

Conducting an election interference campaign using just Facebook advertisements is quite easy because of the cheap and easy-to-target platform provided by Facebook. Using the model developed here, advertisements can be tailored to maximize advertisement effectiveness. Reflecting on this analysis gives richer context to the broader political debate currently occurring, where politicians are wary of Facebook's lack of political advertisement regulation. Russia exploited a real vulnerability that exists on Facebook, a platform that can target voters using private information. Defending ourselves against similar covert operations in the future will require regulating the ability to conduct these Facebook advertisement campaigns, because as demonstrated by this project, Facebook's combination of cheap and targeted advertising makes political interference too easy.

References

[1] Emily Stewart. "Watch AOC ask Mark Zuckerberg if she can run fake Facebook ads, too." In *Vox*. 23 October 2019.

[2] Madeline Carslyle. "Elizabeth Warren Targets Facebook's Ad Policy With a Fake Ad Claiming Mark Zuckerberg Endorsed Trump's Re-Election." In *Time*. 12 October 2019.

[3] UNITED STATES DISTRICT COURT FOR THE DISTRICT OF COLUMBIA. "Internet Research Agency Indictment." 16 February 2018. https://www.justice.gov/file/1035477/download

[4] Dominic Field, Shilpa Patel, and Henry Leon "The Dividends of Digital Marketing Maturity." In *Boston Consulting Group Perspectives*. 18 February 2019. https://www.bcg.com/en-us/publications/2019/dividends-digital-marketing-maturity.aspx

[5] U.S. House of Representatives Permanent Select Committee on Intelligence. "Exposing Russia's Effort to Sow Discord Online: The Internet Research Agency and Advertisements," May 2018. https://intelligence.house.gov/social-media-content

[6] Chirag Sehra "Decision Trees Explained Easily," 19 January 2018. https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248

[7] Jason Brownlee "Boosting and AdaBoost for Machine Learning." In *Machine Learning Algorithms*. 12 August 2019. https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/

[8] Consumer Acquisition. "How Much Does It Cost to Advertise on Facebook?" https://www.consumeracquisition.com/faq/how-much-does-facebook-advertising-cost/

[9] Lonna Rae Atkeson and Cherie D. Maestas "Racing To the Front: The Effect of Frontloading on Presidential Primary Turnout." http://myweb.uiowa.edu/bhlai/caucus/atkeson.pdf

6 Appendix: Python code

```
#!/usr/bin/env python
# coding: utf-8

# # Final Project: How personal information made rigging the election cheap
# **Cs105 Privacy and Technology**<br/>
# **Project By**: Tejal Patwardhan
# <hr style="height:2pt">


# import packages

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import accuracy_score, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.utils import resample
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LogisticRegressionCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
import json
import csv


# access data


with open("russian_ads.json") as f:
    data = json.load(f)



clicks = []
for ad in data:
    clicks.append(ad["ad_clicks"])



plt.hist(clicks,bins=1000)
plt.show()



np.max(clicks)



categories = [
    "ad_id",
    "language_categories",
    "placement_categories",
    "interests_categories",
    "location_categories",
    "ad_copy",
    "ad_landing_page",
    "ad_targeting_location",
    "age_lower",
```

```
    "age_upper",
    "age",
    "placements",
    "ad_impressions",
    "ad_clicks",
    "ad_creation_date",
    "month",
    "year",
    "day",
    "ad_end_date",
    "extra_data",
    "interest_expansion",
    "excluded_connections",
    "language",
    "ad_spend_usd",
    "ad_spend_rub",
    "efficiency_clicks",
    "conversion_rate",
    "interests_categories_regex",
    "location_categories_regex",
    "date_order_index"
    ]

writer = csv.writer(open("full_data.csv", "w", newline=''))
writer.writerow(categories)

for ad in data:
    row = list()
    for category in categories:
        entry = ad[category]
        if type(entry) == type([]):
            entry = "\t".join(entry)
        row.append(entry)
    writer.writerow(row)


full_data = pd.read_csv("full_data.csv")

with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more o
    #display(full_data["ad_targeting_location"])
    display(full_data["placement_categories"])
```

```
# feature engineering


# campaign start and end
start_date = []
for row in full_data["ad_creation_date"]:
    if type(row) == float and np.isnan(row):
        start_date.append(0)
    else:
        start_date.append(datetime.strptime(row, '%m/%d/%Y'))
full_data["start_date"] = start_date

end_date = []
for row in full_data["ad_end_date"]:
    if type(row) == float and np.isnan(row):
        end_date.append(0)
    else:
        end_date.append(datetime.strptime(row, '%m/%d/%Y'))
full_data["end_date"] = end_date

# length of campaign
diff = []
for i in range(len(start_date)):
    if(type(end_date[i]) == int or type(start_date[i]) == int):
        diff.append(0)
    else:
        delta=(end_date[i] - start_date[i])
        diff.append(delta.days)
print(diff)

full_data["campaign_length"] = diff


# number of interests
n_interests = []
for row in full_data["interests_categories"]:
    if type(row) == float and np.isnan(row):
        n_interests.append(0)
    else:
        n_interests.append(1+row.count("\t"))
full_data["n_interests"] = n_interests
```

```
# length of post
post_length = []
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        post_length.append(0)
    else:
        post_length.append(len(row))
full_data["post_length"] = post_length


# pennsylvania
pennsylvania = []
for row in full_data["ad_targeting_location"]:
    if type(row) == float and np.isnan(row):
        pennsylvania.append(0)
    else:
        pennsylvania.append(row.count("Pennsylvania"))
full_data["pennsylvania"] = pennsylvania


# wisconsin
wisconsin = []
for row in full_data["ad_targeting_location"]:
    if type(row) == float and np.isnan(row):
        wisconsin.append(0)
    else:
        wisconsin.append(row.count("Wisconsin"))
full_data["wisconsin"] = wisconsin


# michigan
michigan = []
for row in full_data["ad_targeting_location"]:
    if type(row) == float and np.isnan(row):
        michigan.append(0)
    else:
        michigan.append(row.count("Michigan"))
full_data["michigan"] = michigan


# florida
```

```
florida = []
for row in full_data["ad_targeting_location"]:
    if type(row) == float and np.isnan(row):
        florida.append(0)
    else:
        florida.append(row.count("Florida"))
full_data["florida"] = florida



np.unique(full_data["placement_categories"],return_counts=True)

# spend_real
real_spend = []
for row in full_data["ad_spend_usd"]:
    if type(row) == float and np.isnan(row):
        real_spend.append(0)
    else:
        real_spend.append(row)
full_data["real_spend"] = real_spend



# video
video = []
for row in full_data["placement_categories"]:
    if type(row) == float and np.isnan(row):
        video.append(0)
    else:
        video.append(row.count("Video"))
full_data["video"] = video

# third party apps
apps = []
for row in full_data["placement_categories"]:
    if type(row) == float and np.isnan(row):
        apps.append(0)
    else:
        apps.append(row.count("Third"))
full_data["apps"] = apps
```

```
# analyze direct relationships

np.unique(full_data["conversion_rate"],return_counts=True)
len(full_data["conversion_rate"]) - sum(np.isnan(full_data["conversion_rate"]))


plt.scatter(full_data["ad_spend_rub"],full_data["ad_clicks"])
plt.show()


np.correlate(full_data["ad_spend_rub"],full_data["ad_clicks"])



plt.scatter(full_data["ad_spend_usd"],full_data["ad_clicks"])
plt.show()



plt.scatter(full_data["ad_spend_usd"],full_data["ad_spend_rub"])
plt.show()


plt.scatter(full_data["ad_spend_usd"],full_data["ad_impressions"])
plt.show()


plt.scatter(full_data["ad_spend_usd"],full_data["conversion_rate"])
plt.show()



plt.scatter(full_data["n_interests"],full_data["ad_clicks"])
plt.show()


# messing with whether more precise targeting by number of interests meant more conversi


new_df = pd.concat([full_data["n_interests"],full_data["conversion_rate"],full_data["ad_
sm_df = new_df.dropna()
sm_df
```

```
X = np.array(sm_df["n_interests"]).reshape(-1, 1)
y = np.array(sm_df["conversion_rate"]).reshape(-1, 1)
reg = LinearRegression().fit(X,y)
print(reg.score(X,y))
print(reg.coef_)


X = np.array(sm_df["n_interests"]).reshape(-1, 1)
y = np.array(sm_df["ad_clicks"]).reshape(-1, 1)
reg = LinearRegression().fit(X,y)
print(reg.score(X,y))
print(reg.coef_)


plt.scatter(full_data["ad_clicks"],full_data["conversion_rate"])

# analyze costs

np.sum(full_data["ad_spend_usd"])


np.sum(full_data["ad_impressions"])

np.sum(full_data["ad_clicks"])


np.sum(full_data["ad_spend_rub"])


np.sum(full_data["ad_spend_usd"])/np.sum(full_data["ad_clicks"])

np.sum(full_data["ad_spend_usd"])/np.sum(full_data["ad_impressions"])



# a click cost less than 2.5 cents, an impression was less than .2 cents


# are ads targeted to african americans more clicked on
```

```
# afam
afam = []
for row in full_data["interests_categories_regex"]:
    if type(row) == float and np.isnan(row):
        afam.append(0)
    else:
        afam.append(row.count("African American"))
full_data["afam"] = afam


new_df_2 = pd.concat([full_data["afam"],full_data["conversion_rate"],full_data["ad_click
sm_df_2 = new_df_2.dropna()
sm_df_2

X = np.array(sm_df_2["af_am"]).reshape(-1, 1)
y = np.array(sm_df_2["conversion_rate"]).reshape(-1, 1)
reg = LinearRegression().fit(X,y)
print(reg.score(X,y))
print(reg.coef_)


# what about length of ad


new_df_3 = pd.concat([full_data["post_length"],full_data["conversion_rate"],full_data["a
sm_df_3 = new_df_3.dropna()
#sm_df_3


X = np.array(sm_df_3["post_length"]).reshape(-1, 1)
y = np.array(sm_df_3["ad_clicks"]).reshape(-1, 1)
reg = LinearRegression().fit(X,y)
print(reg.score(X,y))
print(reg.coef_)


plt.hist(sm_df_3["ad_clicks"],bins = 100)


# what about location
```

```
new_df_f = pd.concat([full_data["pennsylvania"],
                      full_data["wisconsin"],
                      full_data["michigan"],
                      full_data["florida"],
                      full_data["age_lower"],
                      full_data["age_upper"],
                      full_data["ad_spend_usd"],
                      full_data["afam"],
                      full_data["n_interests"],
                      full_data["post_length"],
                      full_data["conversion_rate"],
                      full_data["efficiency_clicks"],
                      full_data["ad_clicks"]], axis=1, keys=['pennsylvania',
                                                             'wisconsin',
                                                             'michigan',
                                                             'florida',
                                                             'age_lower',
                                                             'age_upper',
                                                             'ad_spend_usd',
                                                             'afam',
                                                             'n_interests',
                                                             'post_length',
                                                             'conversion_rate',
                                                             'efficiency_clicks',
                                                             'ad_clicks'])
sm_df_f = new_df_f.dropna()
X = pd.concat([sm_df_f["pennsylvania"],
                      sm_df_f["wisconsin"],
                      sm_df_f["michigan"],
                      sm_df_f["florida"],
                      sm_df_f["age_lower"],
                      sm_df_f["age_upper"],
                      sm_df_f["ad_spend_usd"],
                      sm_df_f["afam"],
                      sm_df_f["n_interests"],
                      sm_df_f["post_length"]], axis=1, keys=['pennsylvania',
                                                             'wisconsin',
                                                             'michigan',
                                                             'florida',
                                                             'age_lower',
                                                             'age_upper',
```

```
                                                        'ad_spend_usd',
                                                        'afam',
                                                        'n_interests',
                                                        'post_length'])
y = np.array(sm_df_f["conversion_rate"]).reshape(-1, 1)


plt.hist(sm_df_f["conversion_rate"],bins=100)
plt.title("Conversion rate")
plt.show()
plt.hist(sm_df_f["efficiency_clicks"],bins=100)
plt.title("Efficiency clicks")
plt.show()
plt.hist(sm_df_f["ad_clicks"],bins=100)
plt.title("Ad clicks")
plt.show()




reg = LinearRegression().fit(X,y)
print(reg.score(X,y))
print(np.sqrt(reg.score(X,y)))
print(reg.coef_)


# **add even more features**



# ad copy
# viral
viral = []
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        viral.append(0)
    else:
        viral.append(row.count("viral")+row.count("Viral")+row.count("VIRAL"))
full_data["viral"] = viral

# trump
trump = []
```

```
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        trump.append(0)
    else:
        trump.append(row.count("trump")+row.count("Trump")+row.count("TRUMP"))
full_data["trump"] = trump


# vote
vote = []
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        vote.append(0)
    else:
        vote.append(row.count("vote")+row.count("Vote")+row.count("VOTE"))
full_data["vote"] = vote


# death
death = []
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        death.append(0)
    else:
        death.append(row.count("death")+row.count("Death")+row.count("DEATH")+row.count(
full_data["death"] = death

# exclam
exclam = []
for row in full_data["ad_copy"]:
    if type(row) == float and np.isnan(row):
        exclam.append(0)
    else:
        exclam.append(row.count("!"))
full_data["exclam"] = exclam

# language
# arabic
arabic = []
for row in full_data["language"]:
    if type(row) == float and np.isnan(row):
        arabic.append(0)
```

```
    else:
        arabic.append(row.count("Arabic"))
full_data["arabic"] = arabic

# spanish
spanish = []
for row in full_data["language"]:
    if type(row) == float and np.isnan(row):
        spanish.append(0)
    else:
        spanish.append(row.count("Spanish"))
full_data["spanish"] = spanish


# conservative
conservative = []
for row in full_data["interests_categories_regex"]:
    if type(row) == float and np.isnan(row):
        conservative.append(0)
    else:
        conservative.append(row.count("Conservative")+row.count("Christian")+row.count("
full_data["conservative"] = conservative

# progressive
progressive = []
for row in full_data["interests_categories_regex"]:
    if type(row) == float and np.isnan(row):
        progressive.append(0)
    else:
        progressive.append(row.count("Progressive"))
full_data["progressive"] = progressive

# latinx
latinx = []
for row in full_data["interests_categories_regex"]:
    if type(row) == float and np.isnan(row):
        latinx.append(0)
    else:
        latinx.append(row.count("Latinx"))
full_data["latinx"] = latinx

# gun
```

```
gun = []
for row in full_data["interests_categories_regex"]:
    if type(row) == float and np.isnan(row):
        gun.append(0)
    else:
        gun.append(row.count("Gun"))
full_data["gun"] = gun



sum(full_data["gun"])



# pair plots
ax = sns.pairplot(sm_df_f, diag_kind="kde")
ax
plt.show()



# **okay....let's just get a predictive model all up in here**

# BUILD MODELS TO ACTUALLY TRAIN AND TEST


new_df_f = pd.concat([full_data["pennsylvania"],
                      full_data["wisconsin"],
                      full_data["michigan"],
                      full_data["florida"],
                      full_data["age_lower"],
                      full_data["age_upper"],
                      full_data["real_spend"],
                      #full_data["ad_spend_usd"],
                      full_data["afam"],
                      full_data["n_interests"],
                      full_data["post_length"],
                      full_data["viral"],
                      full_data["trump"],
                      full_data["vote"],
                      full_data["death"],
                      full_data["exclam"],
```

```
full_data["year"],
full_data["arabic"],
full_data["spanish"],
full_data["conservative"],
full_data["progressive"],
full_data["latinx"],
full_data["gun"],
full_data["video"],
full_data["apps"],
full_data["campaign_length"],
full_data["date_order_index"],
full_data["conversion_rate"],
full_data["efficiency_clicks"],
full_data["ad_clicks"]], axis=1, keys=['pennsylvania',
                                       'wisconsin',
                                       'michigan',
                                       'florida',
                                       'age_lower',
                                       'age_upper',
                                       'real_spend',
                                       #'ad_spend_usd',
                                       'afam',
                                       'n_interests',
                                       'post_length',
                                       'viral',
                                       'exclam',
                                       'trump',
                                       'vote',
                                       'death',
                                       'year',
                                       'arabic',
                                       'spanish',
                                       'conservative',
                                       'progressive',
                                       'latinx',
                                       'gun',
                                       'video',
                                       'apps',
                                       'campaign_length',
                                       'date_order_index',
                                       'conversion_rate',
                                       'efficiency_clicks',
```

```
                                                        'ad_clicks'])

# drop empty rows
sm_df_f = new_df_f.dropna()

# also get rid of the one outlier in conversion_rate since it must be less than 1
for index, row in new_df_f.iterrows():
    if(row["conversion_rate"] > 1):
        sm_df_f.loc[index,"conversion_rate"] = 1

sm_df_f = sm_df_f[sm_df_f.conversion_rate < 1]


X = pd.concat([sm_df_f["pennsylvania"],
                      sm_df_f["wisconsin"],
                      sm_df_f["michigan"],
                      sm_df_f["florida"],
                      sm_df_f["age_lower"],
                      sm_df_f["age_upper"],
                    sm_df_f["real_spend"],
                      #sm_df_f["ad_spend_usd"],
                      sm_df_f["afam"],
                      sm_df_f["n_interests"],
                      sm_df_f["post_length"],
                      sm_df_f["viral"],
                      sm_df_f["trump"],
                      sm_df_f["vote"],
                      sm_df_f["death"],
                      sm_df_f["exclam"],
                      sm_df_f["year"],
                      sm_df_f["arabic"],
                      sm_df_f["spanish"],
                      sm_df_f["conservative"],
                      sm_df_f["progressive"],
                      sm_df_f["latinx"],
                      sm_df_f["gun"],
                      sm_df_f["video"],
                      sm_df_f["apps"],
                      sm_df_f["campaign_length"],
                      sm_df_f["date_order_index"]], axis=1, keys=['pennsylvania',
                                                        'wisconsin',
                                                        'michigan',
```

```
                                                    'florida',
                                                    'age_lower',
                                                    'age_upper',
                                                    'ad_spend_usd',
                                                    'afam',
                                                    'n_interests',
                                                    'post_length',
                                                    'viral',
                                                    'exclam',
                                                    'trump',
                                                        'vote',
                                                    'death',
                                                    'year',
                                                   'arabic',
                                                   'spanish',
                                                   'conservative',
                                                   'progressive',
                                                   'latinx',
                                                   'gun',
                                                   'video',
                                                   'apps',
                                                   'campaign_length',
                                                   'date_order_index'])
y = np.array(sm_df_f["ad_clicks"]).reshape(-1, 1)



len(y)


# train/val split on all data, 80-20
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)


print(X_train.shape,X_test.shape)

# regression

reg = LinearRegression().fit(X_train,y_train)
print(reg.score(X_train,y_train))
print(reg.score(X_test,y_test))
```

```
print(np.sqrt(reg.score(X_train,y_train)))
print(reg.coef_)



from sklearn.feature_selection import RFE
selector = RFE(reg, 5, step=1)
selector = selector.fit(X_train, y_train)
first = selector.ranking_
second = X.columns
print(selector.ranking_)
print(X.columns)
feat_rank = pd.DataFrame([first,second])
for row in feat_rank.iterrows():
    print(row)

reg2 = ElasticNet().fit(X_train,y_train)
print(reg2.score(X_train,y_train))
print(reg2.score(X_test,y_test))
print(np.sqrt(reg2.score(X_train,y_train)))
print(reg2.coef_)



selector = RFE(reg2, 5, step=1)
selector = selector.fit(X_train, y_train)
first = selector.ranking_
second = X.columns
print(selector.ranking_)
print(X.columns)
feat_rank = pd.DataFrame([first,second])
for row in feat_rank.iterrows():
    print(row)

# baseline
npm = np.mean(y)
p_base = []
for i in range(len(y_train)):
    p_base.append(npm)
print(r2_score(p_base,y_train))
p_base2 = []
for i in range(len(y_test)):
    p_base2.append(npm)
```

```
print(r2_score(p_base2,y_test))



# decision tree

# classify by depth
def treeRegressorByDepth(depth, X_train, y_train, cvt = 5):
    model = DecisionTreeRegressor(max_depth=depth).fit(X_train, y_train)
    return cross_val_score(model, X_train, y_train, cv = cvt)


# 5-fold CV
means = []
lower = []
upper = []
sds = []
trains = []
for i in range(1, 20):
    # fit model
    tc = treeRegressorByDepth(i, X_train, y_train)
    # calc mean and sd
    cur_mean = np.mean(tc)
    cur_sd = np.std(tc)
    train_val = DecisionTreeRegressor(max_depth=i).fit(X_train, y_train).score(X_train,y
    # add to lists
    trains.append(train_val)
    means.append(cur_mean)
    lower.append(cur_mean - 2*cur_sd)
    upper.append(cur_mean + 2*cur_sd)


# plot
plt.plot(range(1,20),means)
plt.fill_between(range(1,20), lower, upper, alpha = 0.3, label = "Mean CV score (+/- 2SD
plt.plot(range(1,20), trains, label="Train")
plt.title("Decision Tree Model Estimated Performance")
plt.xlabel("Maximum Depth")
plt.ylabel("Score")
plt.legend()
plt.show()
```

```
# cross validation performance
train_score = means[0]
print("Mean score train: ",train_score)
print("Mean +/- 2 SD: (", lower[0],",",upper[0],")")


# holdout set performance
model_dec_tree = DecisionTreeRegressor(max_depth=5).fit(X_train, y_train)
train_score = model_dec_tree.score(X_train, y_train)
print("Mean score train: ", train_score)
val_score = model_dec_tree.score(X_test, y_test)
print("Mean score val: ", val_score)



plt.scatter(full_data["post_length"],full_data["efficiency_clicks"])


# boosted decision tree

# just training the adaboost
X_train_ada, X_val_ada, y_train_ada, y_val_ada = train_test_split(X_train, y_train,
                                                  test_size=0.2, random_state=42)



# evaluate on validaton
abc = AdaBoostRegressor(n_estimators=20)
abc.fit(X_train, y_train)
abc_predicts_train = list(abc.staged_score(X_train_ada, y_train_ada))
# staged_score test to plot
abc_predicts_test = list(abc.staged_score(X_val_ada,y_val_ada))
plt.plot(abc_predicts_test, label = "validation");
plt.plot(abc_predicts_train, label = "train");
plt.legend()
plt.title("AdaBoost Classifier Accuracy")
plt.xlabel("Iterations")
plt.show()
print("Maximum training accuracy is "+str(max(abc_predicts_train))+" at "+str(abc_predic
     "Maximum validation accuracy is "+str(max(abc_predicts_test))+" at "+str(abc_predic
```

```
# evaluate on test
abc = AdaBoostRegressor(n_estimators=20)
abc.fit(X_train, y_train)
abc_predicts_train = list(abc.staged_score(X_train, y_train))
plt.plot(abc_predicts_train, label = "train");
# staged_score test to plot
abc_predicts_test = list(abc.staged_score(X_test,y_test))
plt.plot(abc_predicts_test, label = "test");
plt.legend()
plt.title("AdaBoost Classifier Accuracy")
plt.xlabel("Iterations")
plt.show()
print("Maximum training accuracy is "+str(max(abc_predicts_train))+" at "+str(abc_predic
     "Maximum validation accuracy is "+str(max(abc_predicts_test))+" at "+str(abc_predic
```

```
abc.feature_importances_
```

```
first = abc.feature_importances_
second = X.columns
print(selector.ranking_)
print(X.columns)
feat_rank = pd.DataFrame([first,second])
for row in feat_rank.iterrows():
    print(row)
```

```
for i in range(len(first)):
    print(first[i],second[i])
```

```
# random forest

rf = RandomForestRegressor().fit(X_train,y_train)
print("Training Accuracy:",r2_score(rf.predict(X_train), y_train))
print("Validation Accuracy:",r2_score(rf.predict(X_test), y_test))
```

```
selector = RFE(rf, 1, step=1)
selector = selector.fit(X_train, y_train)
first = selector.ranking_
second = X.columns
print(selector.ranking_)
print(X.columns)
feat_rank = pd.DataFrame([first,second])
for row in feat_rank.iterrows():
    print(row)

for i in range(len(first)):
    print(first[i],second[i])


plt.scatter(rf.predict(X_train), y_train)


plt.scatter(rf.predict(X_test), y_test)
```