ENPM 808W Data Science

Final Project Report

Fake News Data Analysis to predict party, demographics and more!

by

Garett Hill, Prasanna Thirukudanthai Raghavan, Denesh Nallur Narasimman

UNIVERSITY OF MARYLAND

1856

Introduction

This project is inspired by the BuzzFeed article "Most Americans Who See Fake News Believe It, New Survey Says" . It is also claimed to be the first large-scale public opinion survey on fake news conducted which also had a major effect on politics on a global level. According to the article, nearly 75% of the American adult population thinks that the fake news printed is accurate. Further, it also mentions that a majority of the adult population also use Facebook as the source for the news which shouldn't be the case as anyone is free to post whatever they like without any supervision which implies that there is more chance to mislead the consumers with fake news.

With the given data, we plan on visualizing the correlation between the demographic data and the willingness to believe the fake news, and we also plan to train a model to predict a person's willingness to believe fake news based on the demographic information provided in the survey.

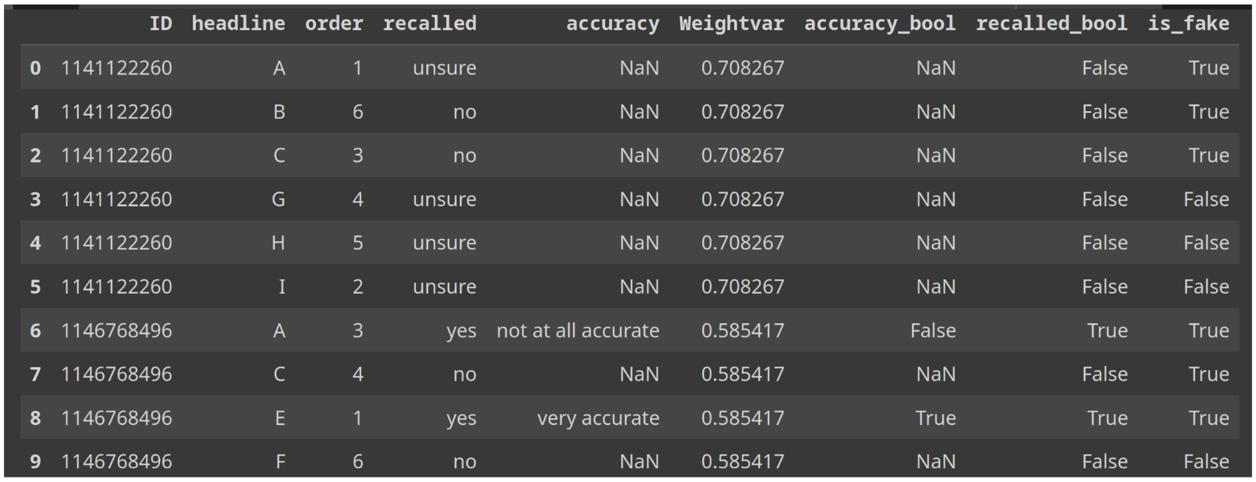
Why this is a Good Idea

As mentioned previously, a huge proportion of the US population is falling for fake news. This has dangerous consequences for democracy, as democracy is fundamentally based on a well informed and engaged citizenry. If we are able to better understand what factors best indicate a person's willingness to believe fake news, we will be able to make more informed decisions about policies and programs to help combat misinformation and disinformation.

What we did

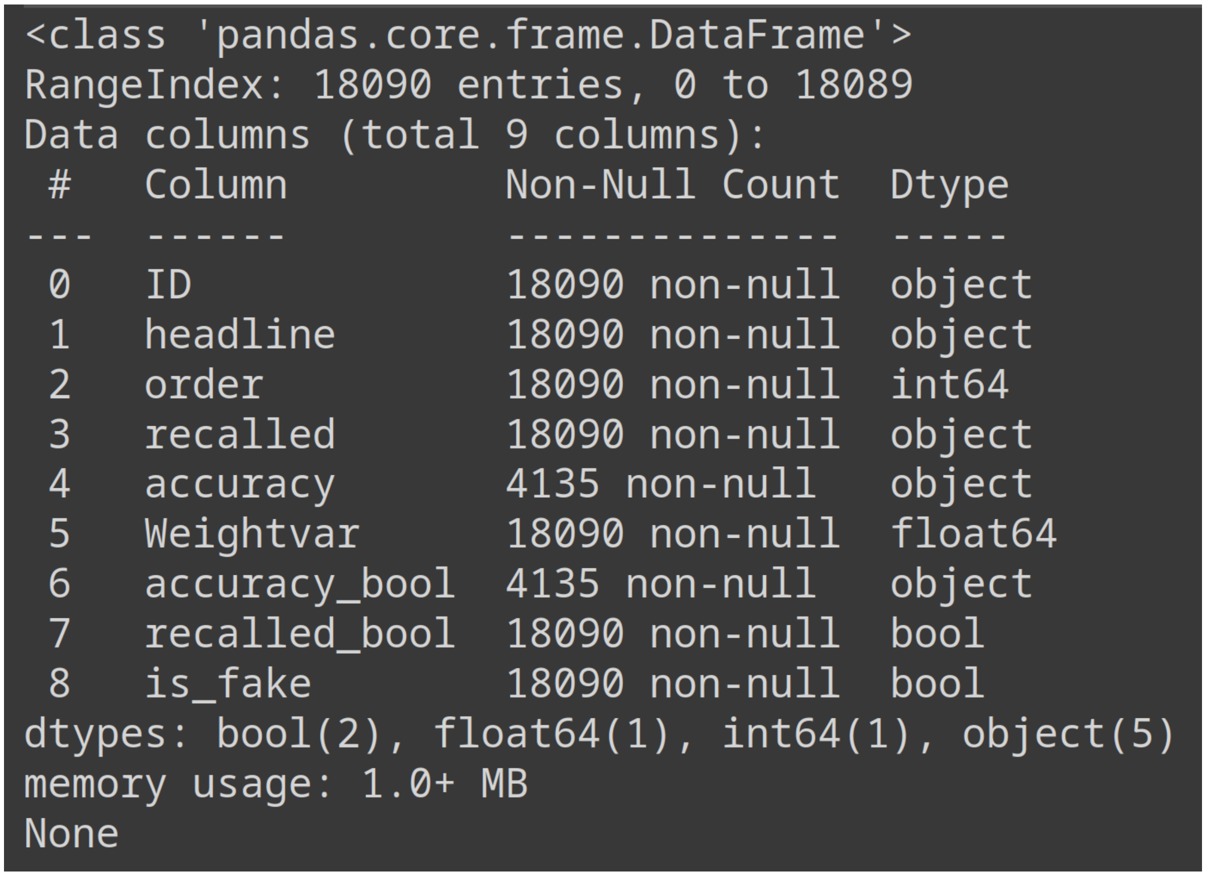
To start, we downloaded the two csv files from the github repository linked in our Sources section below (the Jupyter notebook will download the data from github for you). We needed to understand the structure of the data first because the csv's were not in the same table format, despite representing overlapping information.

Headline-responses.csv, first 10 rows:



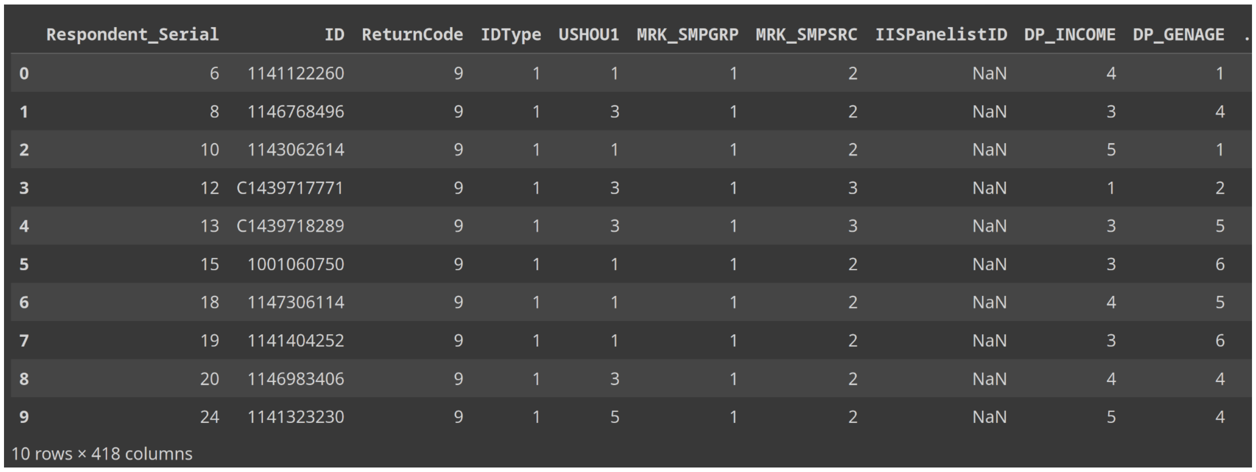
	ID	headline	order	recalled	accuracy	Weightvar	accuracy_bool	recalled_bool	is_fake
0	1141122260	A	1	unsure	NaN	0.708267	NaN	False	True
1	1141122260	B	6	no	NaN	0.708267	NaN	False	True
2	1141122260	C	3	no	NaN	0.708267	NaN	False	True
3	1141122260	G	4	unsure	NaN	0.708267	NaN	False	False
4	1141122260	H	5	unsure	NaN	0.708267	NaN	False	False
5	1141122260	I	2	unsure	NaN	0.708267	NaN	False	False
6	1146768496	A	3	yes	not at all accurate	0.585417	False	True	True
7	1146768496	C	4	no	NaN	0.585417	NaN	False	True
8	1146768496	E	1	yes	very accurate	0.585417	True	True	True
9	1146768496	F	6	no	NaN	0.585417	NaN	False	False

Pandas info() of headline-responses.csv (now called headline_resps) loaded into a DataFrame:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18090 entries, 0 to 18089
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ID             18090 non-null  object
 1   headline       18090 non-null  object
 2   order          18090 non-null  int64
 3   recalled       18090 non-null  object
 4   accuracy       4135 non-null   object
 5   Weightvar      18090 non-null  float64
 6   accuracy_bool  4135 non-null   object
 7   recalled_bool  18090 non-null  bool
 8   is_fake        18090 non-null  bool
dtypes: bool(2), float64(1), int64(1), object(5)
memory usage: 1.0+ MB
None
```

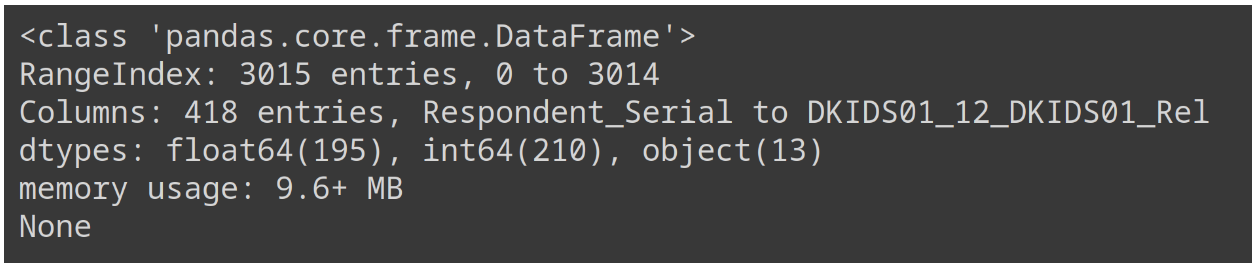
Raw_data.csv, first 10 rows:



	Respondent_Serial	ID	ReturnCode	IDType	USHOU1	MRK_SMPGRP	MRK_SMPSRC	IISPanelistID	DP_INCOME	DP_GENAGE
0	6	1141122260	9	1	1	1	2	NaN	4	1
1	8	1146768496	9	1	3	1	2	NaN	3	4
2	10	1143062614	9	1	1	1	2	NaN	5	1
3	12	C1439717771	9	1	3	1	3	NaN	1	2
4	13	C1439718289	9	1	3	1	3	NaN	3	5
5	15	1001060750	9	1	1	1	2	NaN	3	6
6	18	1147306114	9	1	1	1	2	NaN	4	5
7	19	1141404252	9	1	1	1	2	NaN	3	6
8	20	1146983406	9	1	3	1	2	NaN	4	4
9	24	1141323230	9	1	5	1	2	NaN	5	4

10 rows × 418 columns

Pandas info() of raw_data.csv (now called raw_data) loaded into a DataFrame:

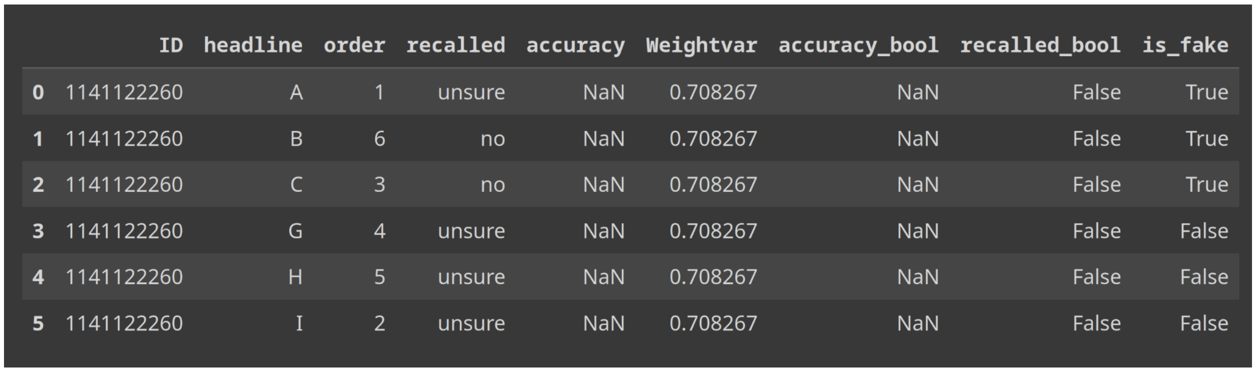


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3015 entries, 0 to 3014
Columns: 418 entries, Respondent_Serial to DKIDS01_12_DKIDS01_Rel
dtypes: float64(195), int64(210), object(13)
memory usage: 9.6+ MB
None
```

Now that we know what our tables look like, what important information do we pay attention to? Notably, the ID's in headline_resps and the ID's in raw_data are the same! However, in headline_resps, there are multiple lines per unique ID, but there is only a single line per unique ID in raw_data. Why is this?

According to the referenced github, headline_resps is a sort of summarized version of raw_data, with each row of headline_resps corresponding to a single survey participant's answer to a single question. While this is nice for looking at one person's responses to the survey (shown below), it is less helpful if we want to treat each survey participant as one observation.

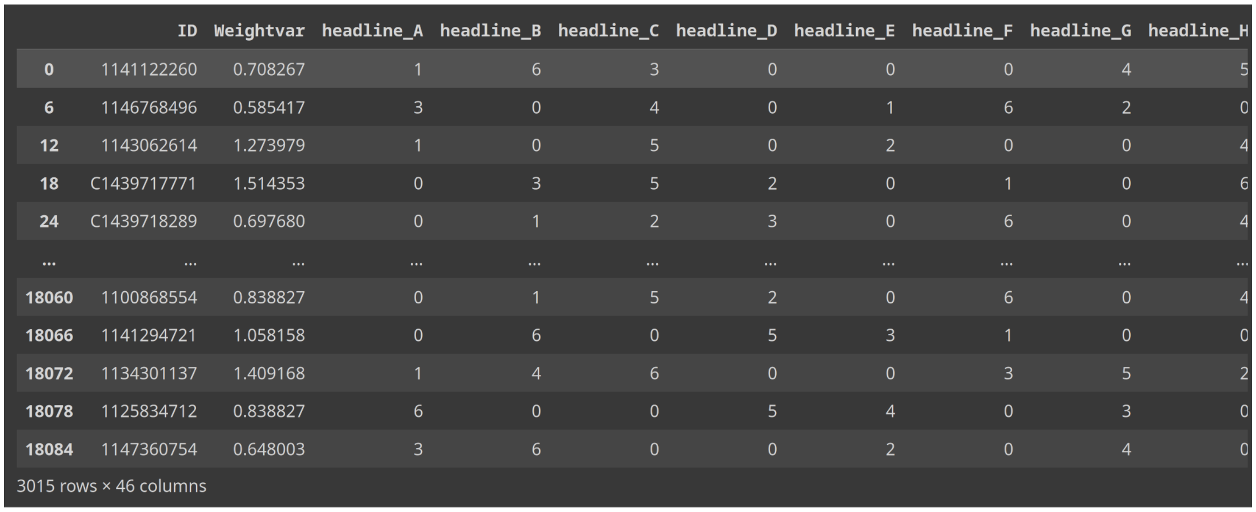
All survey responses for participant ID 1141122260



	ID	headline	order	recalled	accuracy	Weightvar	accuracy_bool	recalled_bool	is_fake
0	1141122260	A	1	unsure	NaN	0.708267	NaN	False	True
1	1141122260	B	6	no	NaN	0.708267	NaN	False	True
2	1141122260	C	3	no	NaN	0.708267	NaN	False	True
3	1141122260	G	4	unsure	NaN	0.708267	NaN	False	False
4	1141122260	H	5	unsure	NaN	0.708267	NaN	False	False
5	1141122260	I	2	unsure	NaN	0.708267	NaN	False	False

So, how do we merge this easier-to-read information of headline_resps into raw_data? In order to match raw_data, we need headline_resps to have all rows with the same ID merged into one row, but without losing any helpful information. Python has some libraries that help in this process, and the final output is shown below.

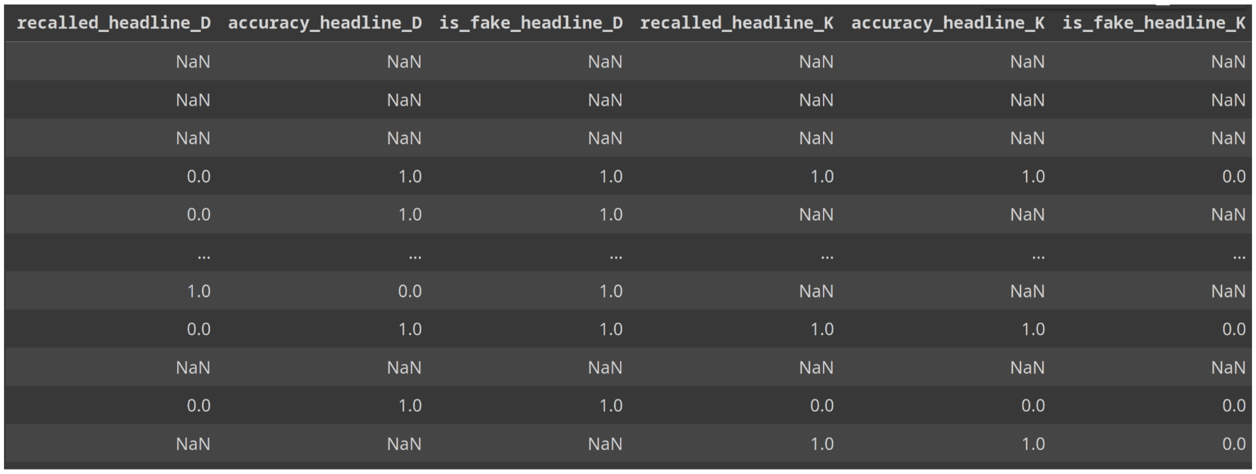
The beginning of the one-id-per-row table



	ID	Weightvar	headline_A	headline_B	headline_C	headline_D	headline_E	headline_F	headline_G	headline_H
0	1141122260	0.708267	1	6	3	0	0	0	4	5
6	1146768496	0.585417	3	0	4	0	1	6	2	0
12	1143062614	1.273979	1	0	5	0	2	0	0	4
18	C1439717771	1.514353	0	3	5	2	0	1	0	6
24	C1439718289	0.697680	0	1	2	3	0	6	0	4
...	...	...	...	...	...	...	...	...	...	...
18060	1100868554	0.838827	0	1	5	2	0	6	0	4
18066	1141294721	1.058158	0	6	0	5	3	1	0	0
18072	1134301137	1.409168	1	4	6	0	0	3	5	2
18078	1125834712	0.838827	6	0	0	5	4	0	3	0
18084	1147360754	0.648003	3	6	0	0	2	0	4	0

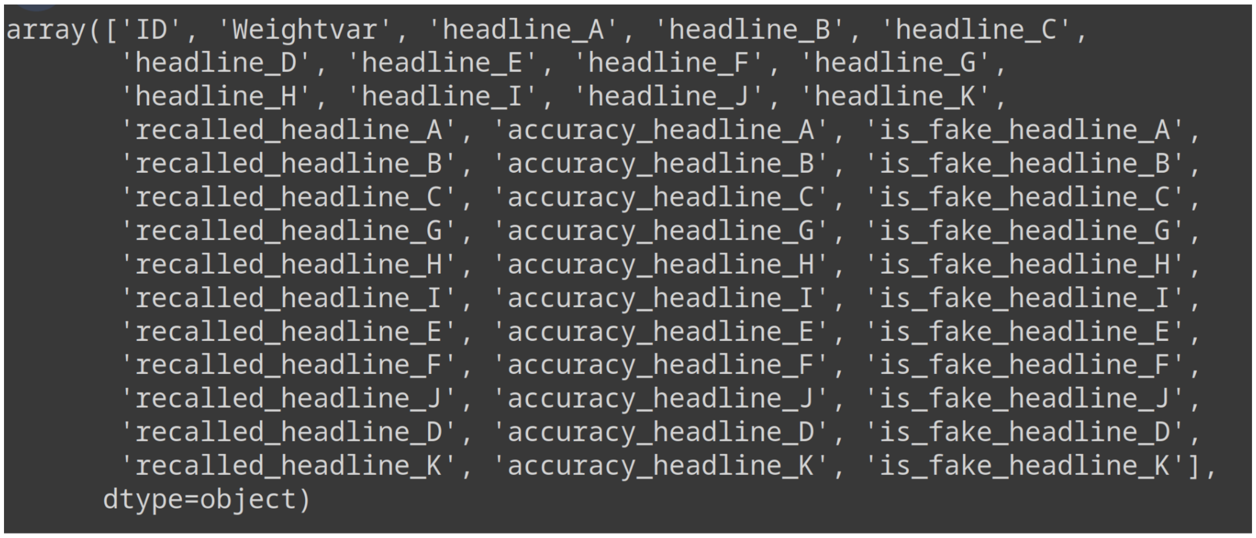
3015 rows × 46 columns

The end of the one-id-per-row table



recalled_headline_D	accuracy_headline_D	is_fake_headline_D	recalled_headline_K	accuracy_headline_K	is_fake_headline_K
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN
0.0	1.0	1.0	1.0	1.0	0.0
0.0	1.0	1.0	NaN	NaN	NaN
...	...	...	...	...	...
1.0	0.0	1.0	NaN	NaN	NaN
0.0	1.0	1.0	1.0	1.0	0.0
NaN	NaN	NaN	NaN	NaN	NaN
0.0	1.0	1.0	0.0	0.0	0.0
NaN	NaN	NaN	1.0	1.0	0.0

All columns of the one-id-per-row table

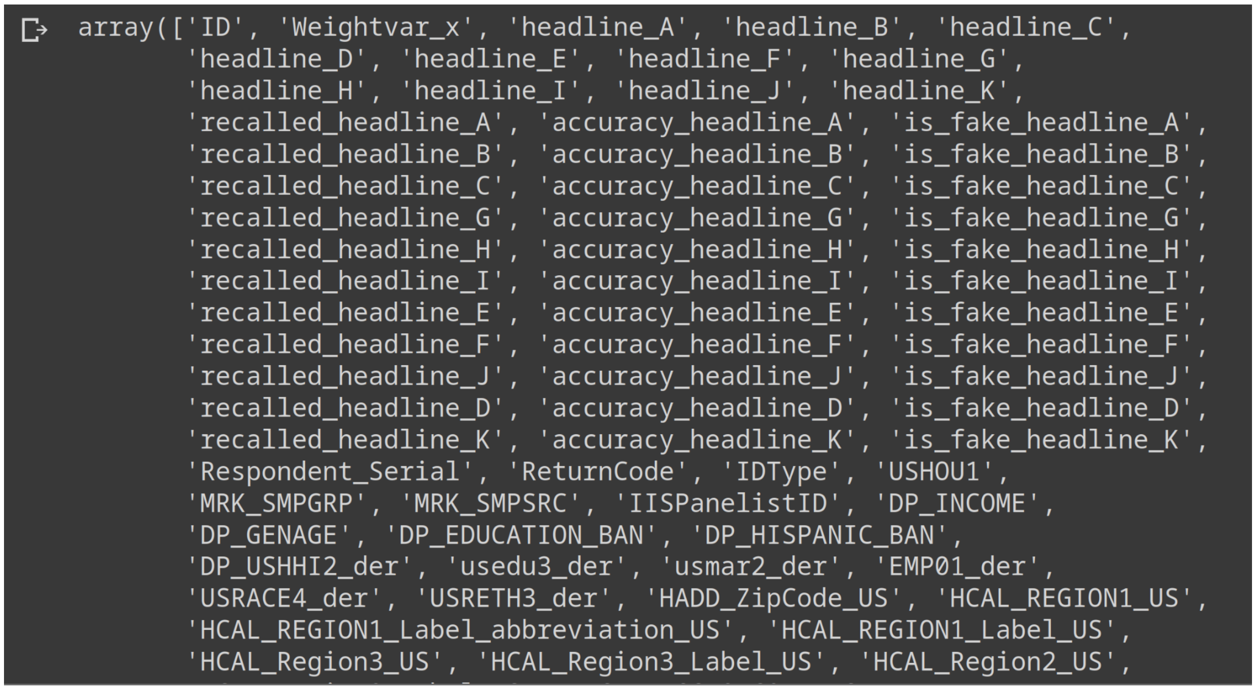


```
array(['ID', 'Weightvar', 'headline_A', 'headline_B', 'headline_C',
       'headline_D', 'headline_E', 'headline_F', 'headline_G',
       'headline_H', 'headline_I', 'headline_J', 'headline_K',
       'recalled_headline_A', 'accuracy_headline_A', 'is_fake_headline_A',
       'recalled_headline_B', 'accuracy_headline_B', 'is_fake_headline_B',
       'recalled_headline_C', 'accuracy_headline_C', 'is_fake_headline_C',
       'recalled_headline_G', 'accuracy_headline_G', 'is_fake_headline_G',
       'recalled_headline_H', 'accuracy_headline_H', 'is_fake_headline_H',
       'recalled_headline_I', 'accuracy_headline_I', 'is_fake_headline_I',
       'recalled_headline_E', 'accuracy_headline_E', 'is_fake_headline_E',
       'recalled_headline_F', 'accuracy_headline_F', 'is_fake_headline_F',
       'recalled_headline_J', 'accuracy_headline_J', 'is_fake_headline_J',
       'recalled_headline_D', 'accuracy_headline_D', 'is_fake_headline_D',
       'recalled_headline_K', 'accuracy_headline_K', 'is_fake_headline_K'],
      dtype=object)
```

With our headline_responses now in the form that we want, they are referred to as "clean_headlines". Now we can combine clean_headlines with our raw_data with a single line of code!

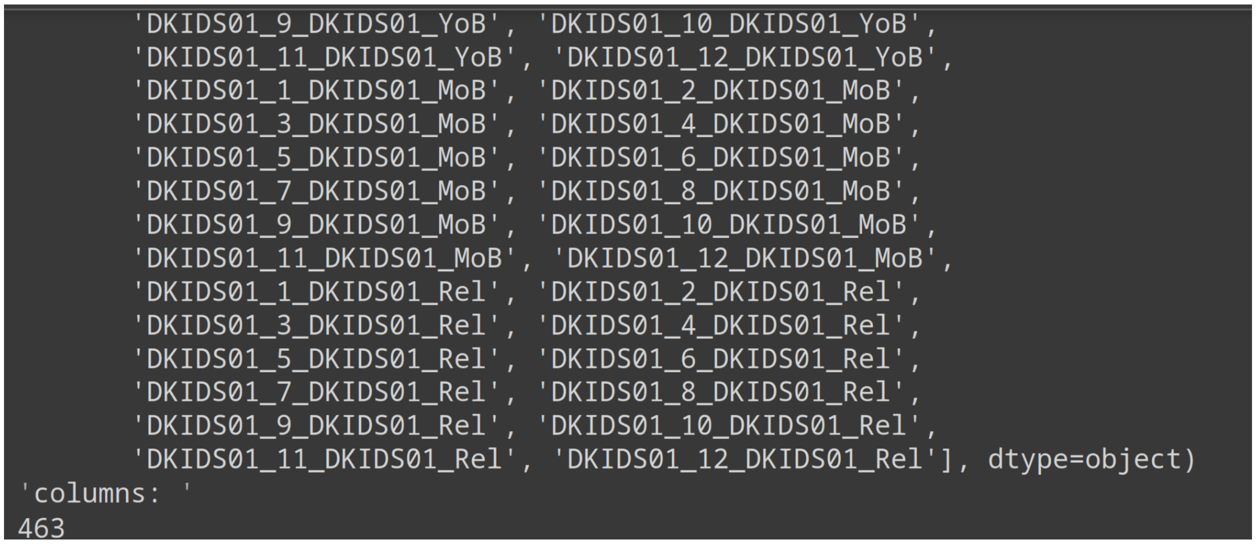
```
combined = clean_headlines.merge(raw_data, how='inner', on='ID')
```

Now what does the combined DataFrame look like?



```
array(['ID', 'Weightvar_x', 'headline_A', 'headline_B', 'headline_C',
        'headline_D', 'headline_E', 'headline_F', 'headline_G',
        'headline_H', 'headline_I', 'headline_J', 'headline_K',
        'recalled_headline_A', 'accuracy_headline_A', 'is_fake_headline_A',
        'recalled_headline_B', 'accuracy_headline_B', 'is_fake_headline_B',
        'recalled_headline_C', 'accuracy_headline_C', 'is_fake_headline_C',
        'recalled_headline_G', 'accuracy_headline_G', 'is_fake_headline_G',
        'recalled_headline_H', 'accuracy_headline_H', 'is_fake_headline_H',
        'recalled_headline_I', 'accuracy_headline_I', 'is_fake_headline_I',
        'recalled_headline_E', 'accuracy_headline_E', 'is_fake_headline_E',
        'recalled_headline_F', 'accuracy_headline_F', 'is_fake_headline_F',
        'recalled_headline_J', 'accuracy_headline_J', 'is_fake_headline_J',
        'recalled_headline_D', 'accuracy_headline_D', 'is_fake_headline_D',
        'recalled_headline_K', 'accuracy_headline_K', 'is_fake_headline_K',
        'Respondent_Serial', 'ReturnCode', 'IDType', 'USHOU1',
        'MRK_SMPGRP', 'MRK_SMPSRC', 'IISPanelistID', 'DP_INCOME',
        'DP_GENAGE', 'DP_EDUCATION_BAN', 'DP_HISPANIC_BAN',
        'DP_USHHI2_der', 'usedu3_der', 'usmar2_der', 'EMP01_der',
        'USRACE4_der', 'USRETH3_der', 'HADD_ZipCode_US', 'HCAL_REGION1_US',
        'HCAL_REGION1_Label_abbreviation_US', 'HCAL_REGION1_Label_US',
        'HCAL_Region3_US', 'HCAL_Region3_Label_US', 'HCAL_Region2_US',
```

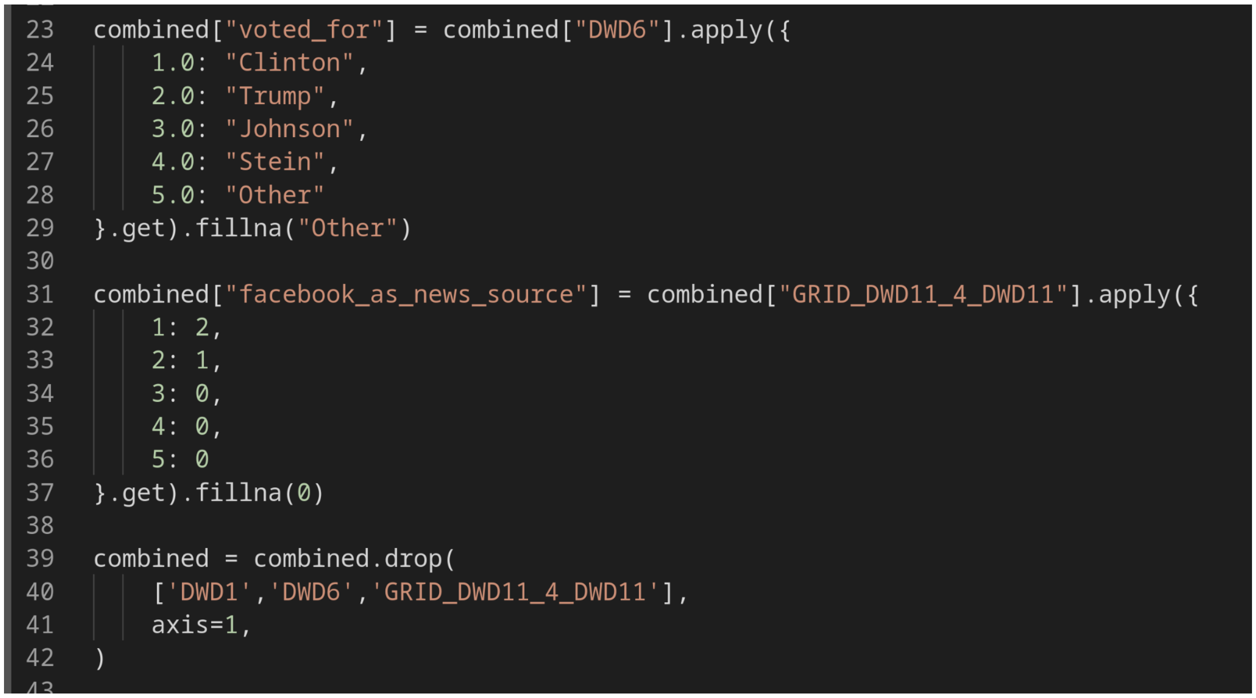
…



```
       'DKIDS01_9_DKIDS01_YoB', 'DKIDS01_10_DKIDS01_YoB',
       'DKIDS01_11_DKIDS01_YoB', 'DKIDS01_12_DKIDS01_YoB',
       'DKIDS01_1_DKIDS01_MoB', 'DKIDS01_2_DKIDS01_MoB',
       'DKIDS01_3_DKIDS01_MoB', 'DKIDS01_4_DKIDS01_MoB',
       'DKIDS01_5_DKIDS01_MoB', 'DKIDS01_6_DKIDS01_MoB',
       'DKIDS01_7_DKIDS01_MoB', 'DKIDS01_8_DKIDS01_MoB',
       'DKIDS01_9_DKIDS01_MoB', 'DKIDS01_10_DKIDS01_MoB',
       'DKIDS01_11_DKIDS01_MoB', 'DKIDS01_12_DKIDS01_MoB',
       'DKIDS01_1_DKIDS01_Rel', 'DKIDS01_2_DKIDS01_Rel',
       'DKIDS01_3_DKIDS01_Rel', 'DKIDS01_4_DKIDS01_Rel',
       'DKIDS01_5_DKIDS01_Rel', 'DKIDS01_6_DKIDS01_Rel',
       'DKIDS01_7_DKIDS01_Rel', 'DKIDS01_8_DKIDS01_Rel',
       'DKIDS01_9_DKIDS01_Rel', 'DKIDS01_10_DKIDS01_Rel',
       'DKIDS01_11_DKIDS01_Rel', 'DKIDS01_12_DKIDS01_Rel'], dtype=object)
'columns: '
463
```

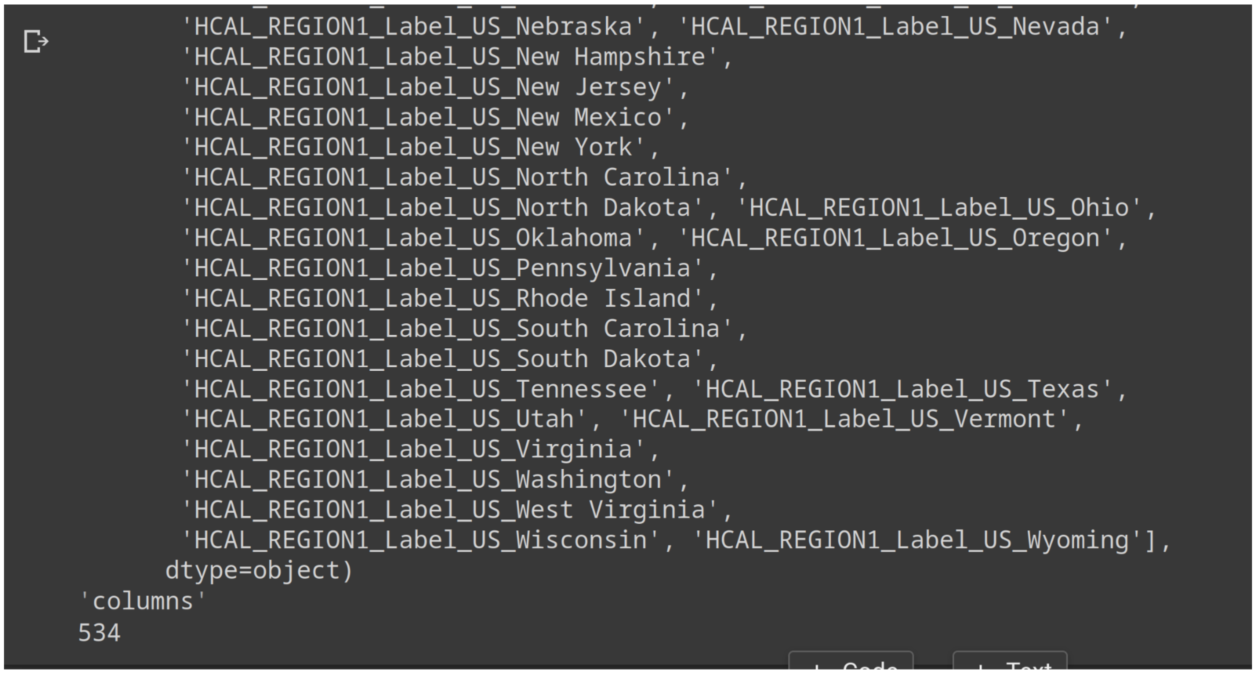
Clearly, there are a lot of columns here, 463 to be exact. We can also decode some of the known columns (features) as demonstrated in the original notebook provided by buzzfeed.

Some of the feature engineering/decoding



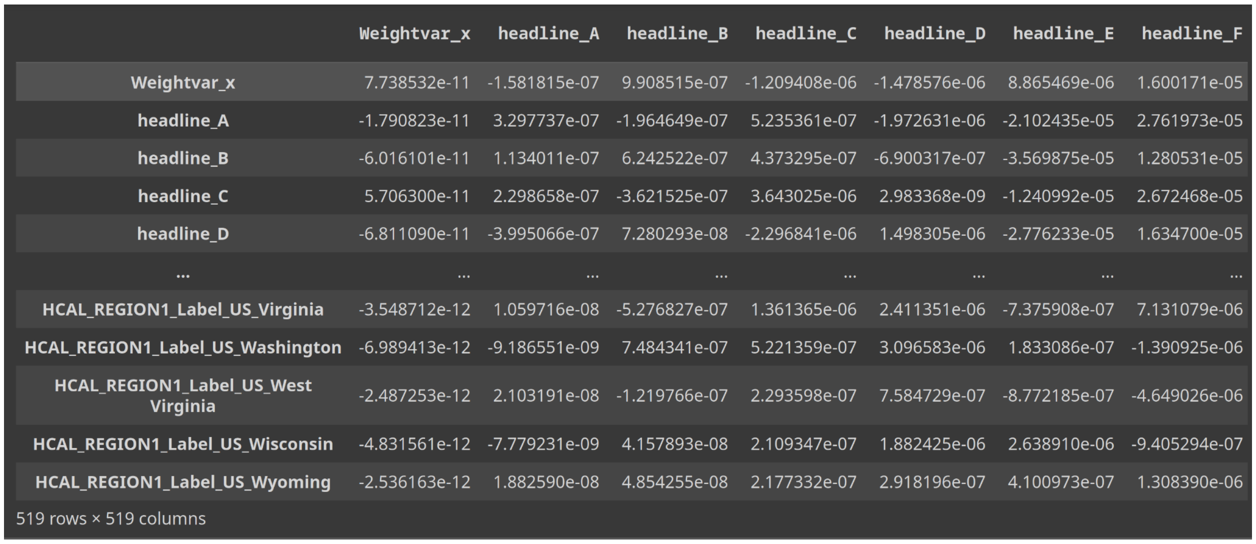
```
combined["voted_for"] = combined["DWD6"].apply({
        1.0: "Clinton",
        2.0: "Trump",
        3.0: "Johnson",
        4.0: "Stein",
        5.0: "Other"
}.get).fillna("Other")

combined["facebook_as_news_source"] = combined["GRID_DWD11_4_DWD11"].apply({
        1: 2,
        2: 1,
        3: 0,
        4: 0,
        5: 0
}.get).fillna(0)

combined = combined.drop(
        ['DWD1','DWD6','GRID_DWD11_4_DWD11'],
        axis=1,
)
```

New columns including (but not limited to) one-hot encoding for state/party/who participant voted for



```
       'HCAL_REGION1_Label_US_Nebraska', 'HCAL_REGION1_Label_US_Nevada',
       'HCAL_REGION1_Label_US_New Hampshire',
       'HCAL_REGION1_Label_US_New Jersey',
       'HCAL_REGION1_Label_US_New Mexico',
       'HCAL_REGION1_Label_US_New York',
       'HCAL_REGION1_Label_US_North Carolina',
       'HCAL_REGION1_Label_US_North Dakota', 'HCAL_REGION1_Label_US_Ohio',
       'HCAL_REGION1_Label_US_Oklahoma', 'HCAL_REGION1_Label_US_Oregon',
       'HCAL_REGION1_Label_US_Pennsylvania',
       'HCAL_REGION1_Label_US_Rhode Island',
       'HCAL_REGION1_Label_US_South Carolina',
       'HCAL_REGION1_Label_US_South Dakota',
       'HCAL_REGION1_Label_US_Tennessee', 'HCAL_REGION1_Label_US_Texas',
       'HCAL_REGION1_Label_US_Utah', 'HCAL_REGION1_Label_US_Vermont',
       'HCAL_REGION1_Label_US_Virginia',
       'HCAL_REGION1_Label_US_Washington',
       'HCAL_REGION1_Label_US_West Virginia',
       'HCAL_REGION1_Label_US_Wisconsin', 'HCAL_REGION1_Label_US_Wyoming'],
      dtype=object)
'columns'
534
```

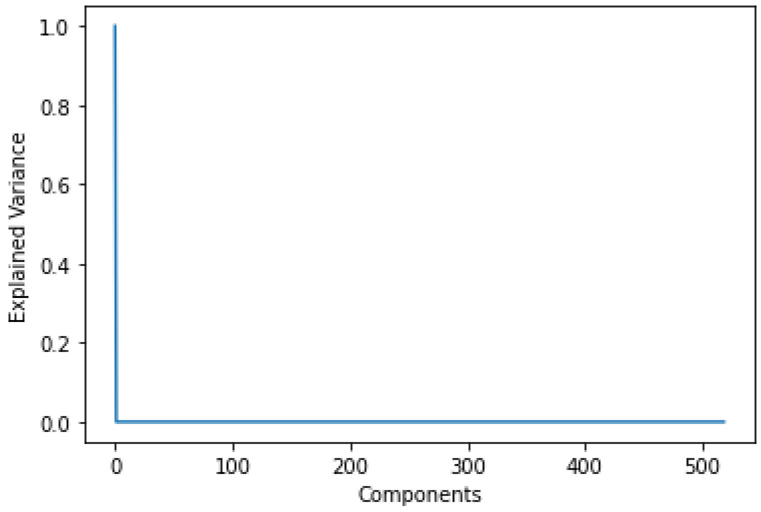
After filtering our features to just numerical values using *df.select_dtypes(include=np.number)*, we can examine the correlation matrix between each variable using sklearn’s Primary Component Analysis library, shown here.



	Weightvar_x	headline_A	headline_B	headline_C	headline_D	headline_E	headline_F
Weightvar_x	7.738532e-11	-1.581815e-07	9.908515e-07	-1.209408e-06	-1.478576e-06	8.865469e-06	1.600171e-05
headline_A	-1.790823e-11	3.297737e-07	-1.964649e-07	5.235361e-07	-1.972631e-06	-2.102435e-05	2.761973e-05
headline_B	-6.016101e-11	1.134011e-07	6.242522e-07	4.373295e-07	-6.900317e-07	-3.569875e-05	1.280531e-05
headline_C	5.706300e-11	2.298658e-07	-3.621525e-07	3.643025e-06	2.983368e-09	-1.240992e-05	2.672468e-05
headline_D	-6.811090e-11	-3.995066e-07	7.280293e-08	-2.296841e-06	1.498305e-06	-2.776233e-05	1.634700e-05
...	...	...	...	...	...	...	...
HCAL_REGION1_Label_US_Virginia	-3.548712e-12	1.059716e-08	-5.276827e-07	1.361365e-06	2.411351e-06	-7.375908e-07	7.131079e-06
HCAL_REGION1_Label_US_Washington	-6.989413e-12	-9.186551e-09	7.484341e-07	5.221359e-07	3.096583e-06	1.833086e-07	-1.390925e-06
HCAL_REGION1_Label_US_West Virginia	-2.487253e-12	2.103191e-08	-1.219766e-07	2.293598e-07	7.584729e-07	-8.772185e-07	-4.649026e-06
HCAL_REGION1_Label_US_Wisconsin	-4.831561e-12	-7.779231e-09	4.157893e-08	2.109347e-07	1.882425e-06	2.638910e-06	-9.405294e-07
HCAL_REGION1_Label_US_Wyoming	-2.536163e-12	1.882590e-08	4.854255e-08	2.177332e-07	2.918196e-07	4.100973e-07	1.308390e-06

519 rows × 519 columns

Now examine the explained variance plot for the components (below), showing that we need less than 10 features to accurately predict another.



If we don’t need all of our components to make predictions, then we should try to find the most important components to make our predictions. Let’s try doing that with a training pipeline of sklearn’s SelectKBest into a Support Vector Classifier (rbf kernel).

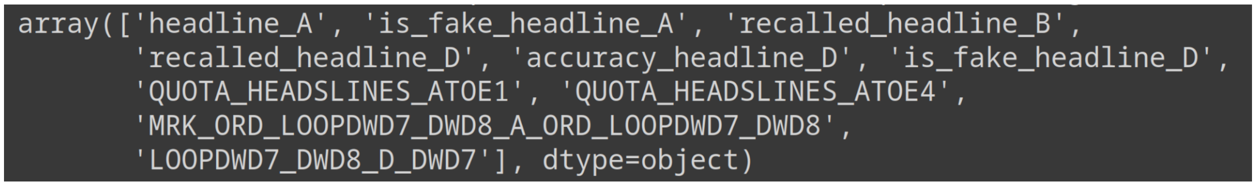
Additional Features:

We needed to check if there were additional features that can be used or if new features could be manufactured from the dataset. For this we took the addition of geographical features to see if they will give us a better prediction but the geo-data did not help but in turn also made the predictions worse which meant that the location of the individual did not matter in the likelihood of them believing in false news as well as voting or supporting a particular party. For the models; state, city or tri-state area had the same amount of result in the final prediction.

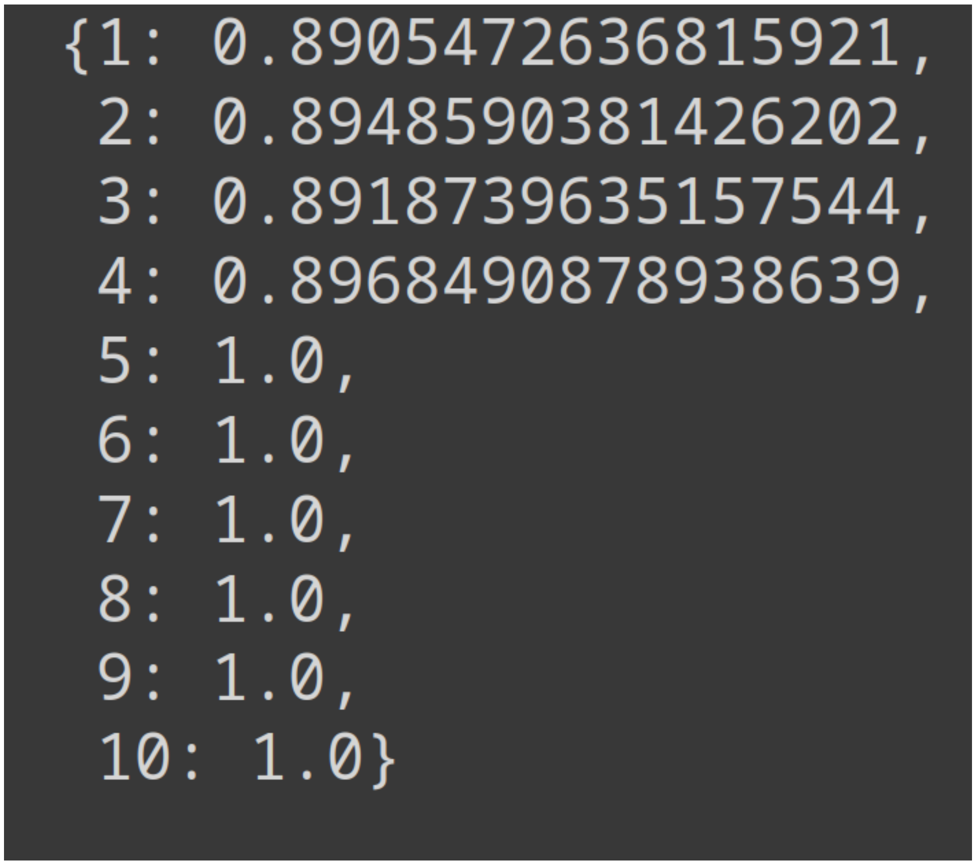
Baseline Measurement:

For the baseline measurement, we used a machine learning library that helped us in choosing the right feature before implementing our bespoke feature picking function as the dataset came with around 450 features and we did not require all of them for the prediction. This helped us in predicting the 10 best features and also gave us an idea of if and how the model is overfitting the data. The library did perform well with a constant prediction of 80% - 90% but started to overfit from k=5. Detailed analysis of why this is happening is given below.

Here's what sklearn thinks are the most important features when trying to predict if a person will correctly classify "Headline A" from the survey, as well as how well the model performed on a 20% train/test split for k values 1-10:



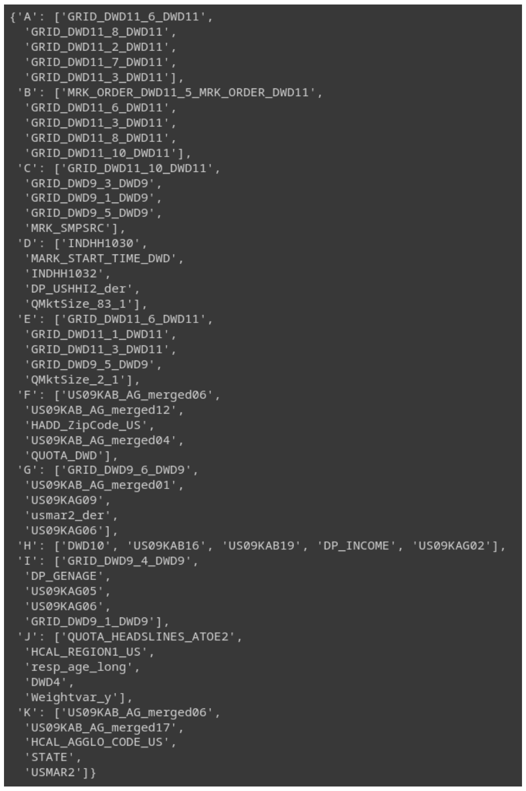
```
array(['headline_A', 'is_fake_headline_A', 'recalled_headline_B',
       'recalled_headline_D', 'accuracy_headline_D', 'is_fake_headline_D',
       'QUOTA_HEADSLINES_ATOE1', 'QUOTA_HEADSLINES_ATOE4',
       'MRK_ORD_LOOPDWD7_DWD8_A_ORD_LOOPDWD7_DWD8',
       'LOOPDWD7_DWD8_D_DWD7'], dtype=object)
```



```
{1: 0.8905472636815921,
 2: 0.8948590381426202,
 3: 0.8918739635157544,
 4: 0.8968490878938639,
 5: 1.0,
 6: 1.0,
 7: 1.0,
 8: 1.0,
 9: 1.0,
 10: 1.0}
```

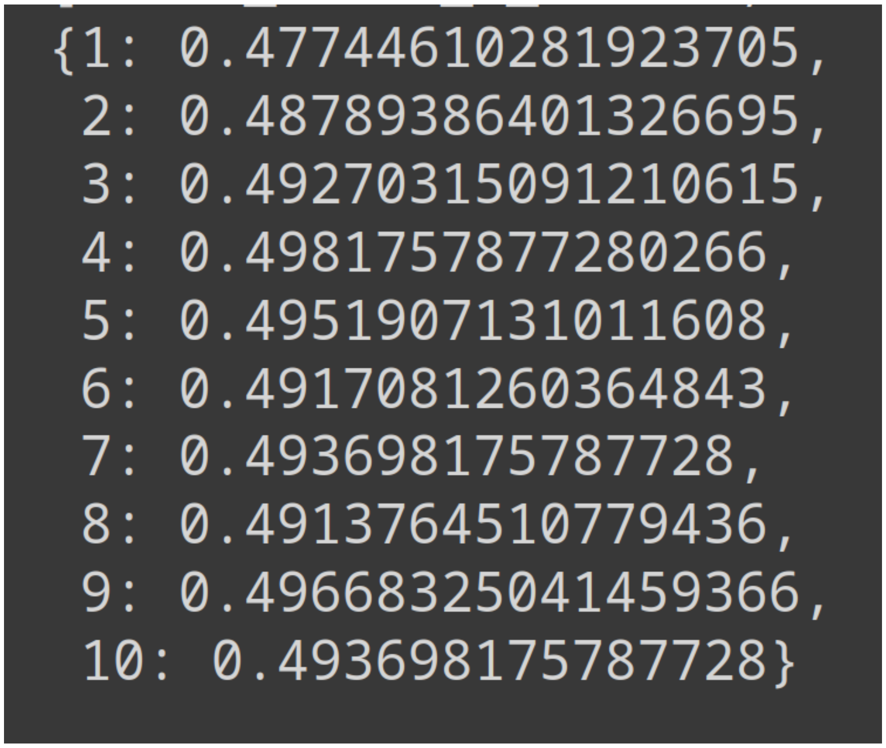
Since the feature headline_A here tells us what order the headline was presented to the participant in, we can see that the order matters when being presented headlines, or alternatively, what matters is that the question was presented to the participant at all (non-zero value)! Additionally, we can see that a person’s response to other questions (headlines B and D here) impacts their expected response to headline A, which makes sense intuitively.

Perhaps against better judgment, I tried using my own implementation of K-best feature picking based on the PCA table we created before. Below is the list of features that are most impactful for each question according to my model.



```
{'A': ['GRID_DWD11_6_DWD11',
  'GRID_DWD11_8_DWD11',
  'GRID_DWD11_2_DWD11',
  'GRID_DWD11_7_DWD11',
  'GRID_DWD11_3_DWD11'],
 'B': ['MRK_ORDER_DWD11_5_MRK_ORDER_DWD11',
  'GRID_DWD11_6_DWD11',
  'GRID_DWD11_3_DWD11',
  'GRID_DWD11_8_DWD11',
  'GRID_DWD11_10_DWD11'],
 'C': ['GRID_DWD11_10_DWD11',
  'GRID_DWD9_3_DWD9',
  'GRID_DWD9_1_DWD9',
  'GRID_DWD9_5_DWD9',
  'MRK_SMPSRC'],
 'D': ['INDHH1030',
  'MARK_START_TIME_DWD',
  'INDHH1032',
  'DP_USHHI2_der',
  'QMktSize_83_1'],
 'E': ['GRID_DWD11_6_DWD11',
  'GRID_DWD11_1_DWD11',
  'GRID_DWD11_3_DWD11',
  'GRID_DWD9_5_DWD9',
  'QMktSize_2_1'],
 'F': ['US09KAB_AG_merged06',
  'US09KAB_AG_merged12',
  'HADD_ZipCode_US',
  'US09KAB_AG_merged04',
  'QUOTA_DWD'],
 'G': ['GRID_DWD9_6_DWD9',
  'US09KAB_AG_merged01',
  'US09KAG09',
  'usmar2_der',
  'US09KAG06'],
 'H': ['DWD10', 'US09KAB16', 'US09KAB19', 'DP_INCOME', 'US09KAG02'],
 'I': ['GRID_DWD9_4_DWD9',
  'DP_GENAGE',
  'US09KAG05',
  'US09KAG06',
  'GRID_DWD9_1_DWD9'],
 'J': ['QUOTA_HEADSLINES_ATOE2',
  'HCAL_REGION1_US',
  'resp_age_long',
  'DWD4',
  'Weightvar_y'],
 'K': ['US09KAB_AG_merged06',
  'US09KAB_AG_merged17',
  'HCAL_AGGLO_CODE_US',
  'STATE',
  'USMAR2']}
```

If we try a similar training pipeline to before (k-best into SCV), how well do we do? Unfortunately, basically no better than random chance for k values 1-10.



```
{1: 0.47744610281923705,
 2: 0.48789386401326695,
 3: 0.49270315091210615,
 4: 0.4981757877280266,
 5: 0.4951907131011608,
 6: 0.4917081260364843,
 7: 0.493698175787728,
 8: 0.4913764510779436,
 9: 0.49668325041459366,
 10: 0.493698175787728}
```

Did our technique work?

While our library-based model had an impressive success rate at predicting fake news beliefs (when it isn't overfitting), the bespoke model performed roughly equivalent to random guesses. However, we have a leading theory regarding the limitations of the bespoke model, which might explain such poor results.

We believe that our bespoke model is overfitting to whatever its training data is due to some sort of re-encoding of information across features. Perhaps someone's response to the survey question is contained in a variable we don't expect, or their lack of responding to a randomly chosen half of the questionnaire is causing the model to predict their response based only on whether they were asked the question at all.

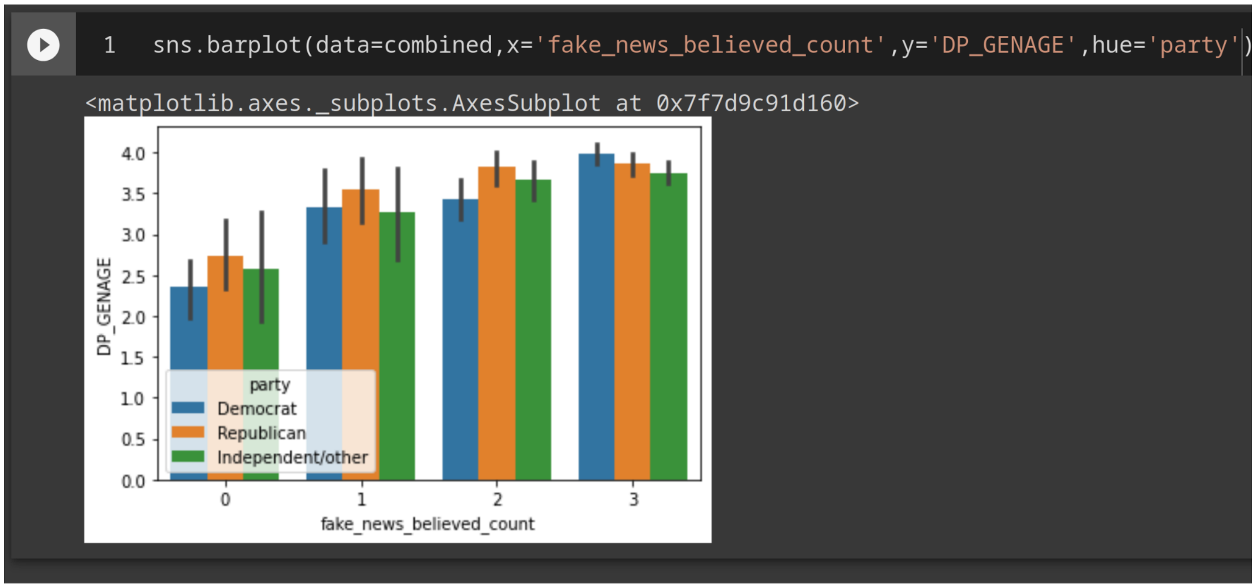
Future Research

With unlimited time and resources, I would absolutely proceed with decoding all the variables we were not able to understand yet. I think that there is a lot of very valuable information that we do not have access to in our predictions, much of which would undoubtedly improve our models.

Additionally, I'd like to add additional features regarding the text of the headlines. At the professor's mention, I went back and looked at homework 2 with the bigrams and trigrams, which seem like something that could be very useful in predicting someone's response to an individual fake news headline. Currently, we are able to see which components correlate most with individual headline response accuracies, but I hypothesize adding some natural language processing could considerably improve our model.

Closing remarks

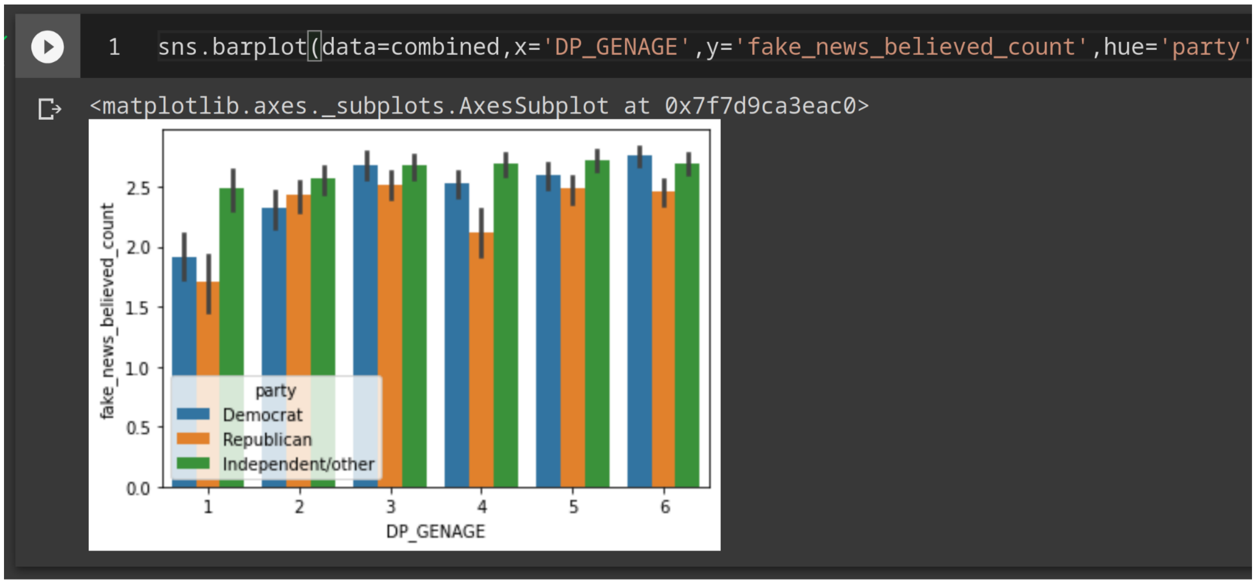
Despite our bespoke model not performing well, I was still able to find a strong visualized correlation between generation age and the *amount* of fake news believed.



```
sns.barplot(data=combined,x='fake_news_believed_count',y='DP_GENAGE',hue='party')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7d9c91d160>
```

Here we can clearly see the lower expected generation age for someone who does not fall for fake news. How about the reverse?



```
sns.barplot(data=combined,x='DP_GENAGE',y='fake_news_believed_count',hue='party'
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7d9ca3eac0>
```

Again, the youngest generation (except for independants/non-D/R's) is least likely to believe fake news.

__Instructions to run the code:__

Follow the colab link provided under the "Resources" heading and execute the cells one by one for the output. It is not necessary for you to download the dataset and save in your system as we have extracted the data directly from the github repo to make things easier.

Who did what

Garrett:

- Python notebook
- Slides 6-7
- Project writeup ‘Why this is a good idea’ - ‘Resources’

Prasanna:

- Slide 4-5
- Project write up ‘Alternative features’

Denesh:

- Slides 2-3
- Project writeup ‘Introduction’ - ’Baseline Measurement’ - ’Instructions to run the code’

Resources

- Github Repo for Data Source : https://github.com/BuzzFeedNews/2016-12-fake-news-survey
- Our Notebook (Shared with professor): https://colab.research.google.com/drive/1s_VtQ_y5j87fJZw4M3tYnBbV-WEK5WUV#scrollTo=97C3fN5hdiG1