



FEE.org Visitor Analysis & Content Recommendations

David Veksler

Trending
Stories



The Olympic Ideal



Bitcoin Wins in Court



The One Question Ruining
the Song of Liberty

Technology

You Don't Actually Own Your Securities



Caitlin Long
Aug 11, 2016

Latest Stories



Policy

Why We Trade

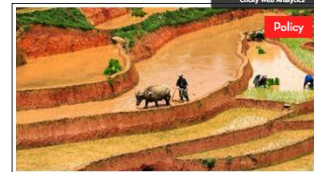
Daniel J. Ikenson - Aug 11, 2016



Technology

Bitcoin Wins in Court

Larry White - Aug 11, 2016



Policy

Poor Countries Need Market
Access, Not "Assistance"

Nate Mason - Aug 11, 2016

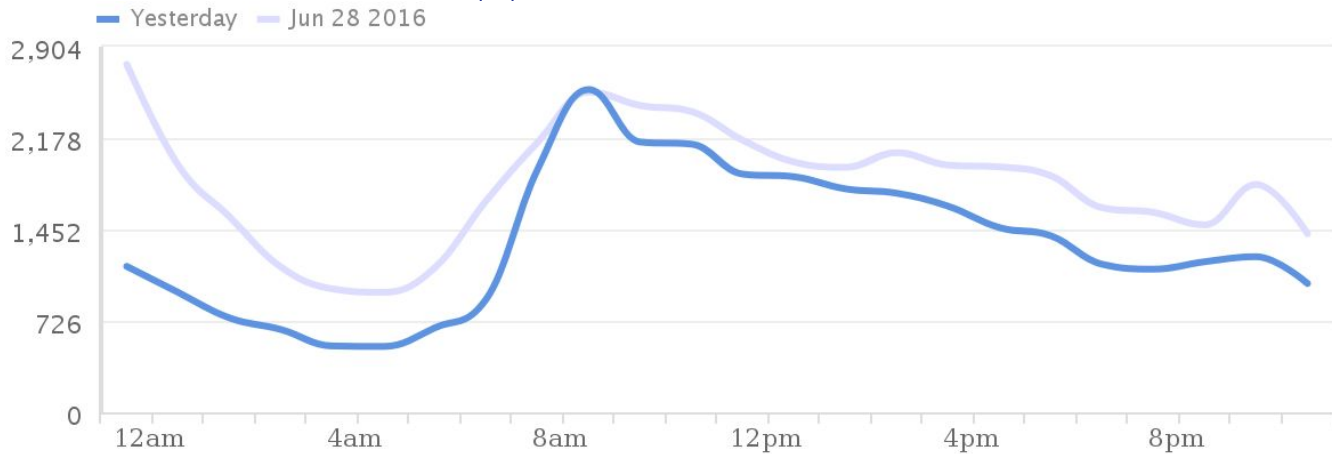
The problem

Who are our visitors?

- What content engages users?
- What does a typical session look like?

Can we improve engagement?

- Can we push relevant content to visitors?



Solution

Build content recommendation
algorithm

- Analyze content on FEE.org
- Suggest related articles to visitors

Part 1: Content & Visitor Analysis

<https://github.com/DavidVeksler/DS3-Projects/blob/master/Final%20Project/Part%203%20-%20Exploratory%20Analysis.ipynb>

Content Analysis

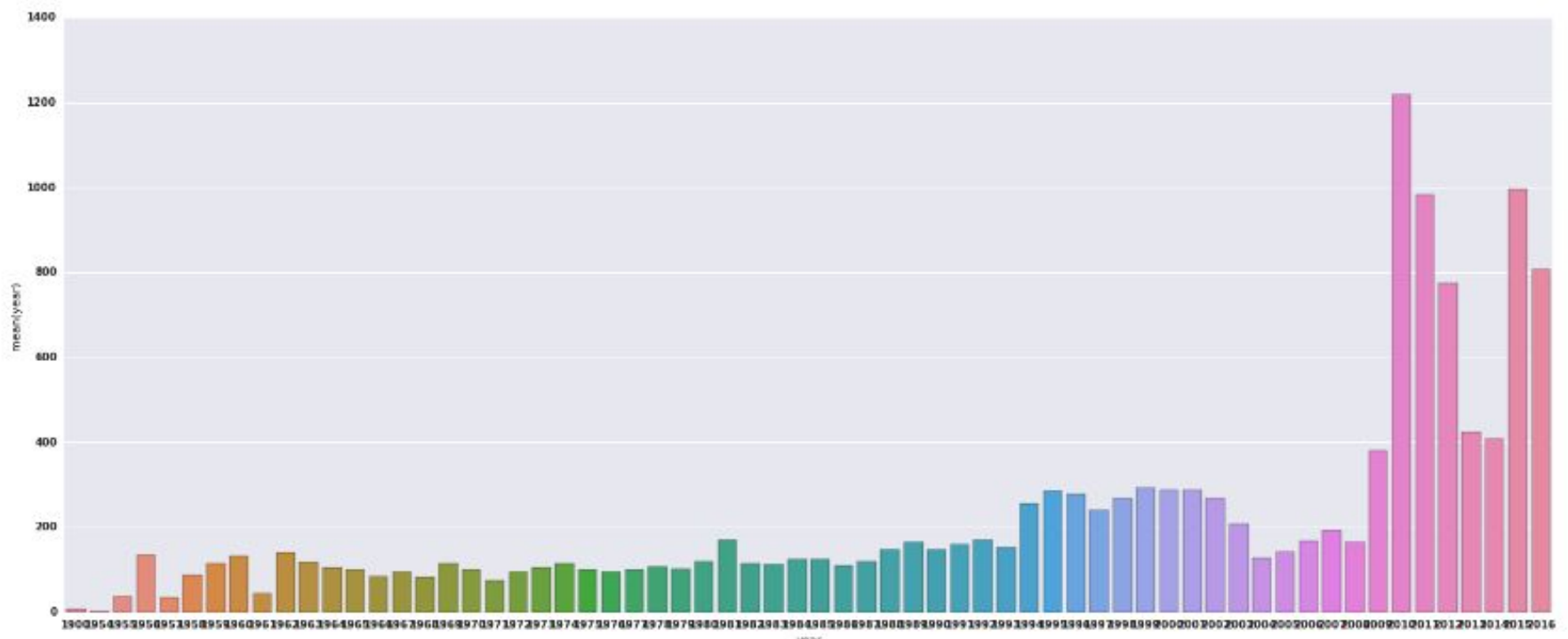
- 13000 articles over 66 years
- Source: content management system

Visitor Analysis

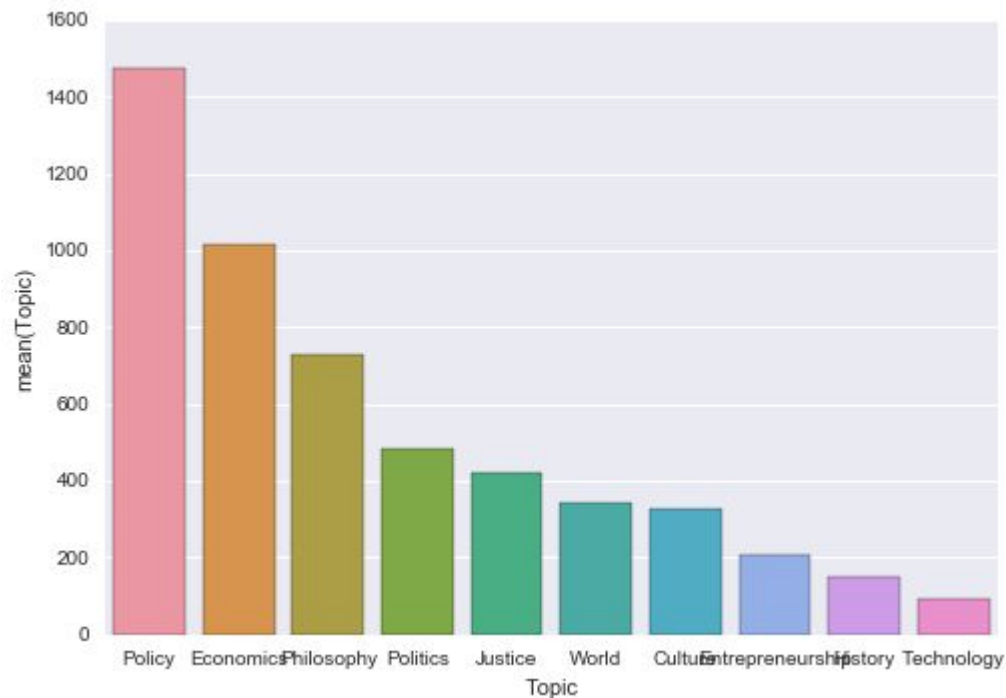
- 50,000 actions (page views) over 11 hours
- Source: Clicky web traffic logs



Articles per year, 1952-2016



Top Categories

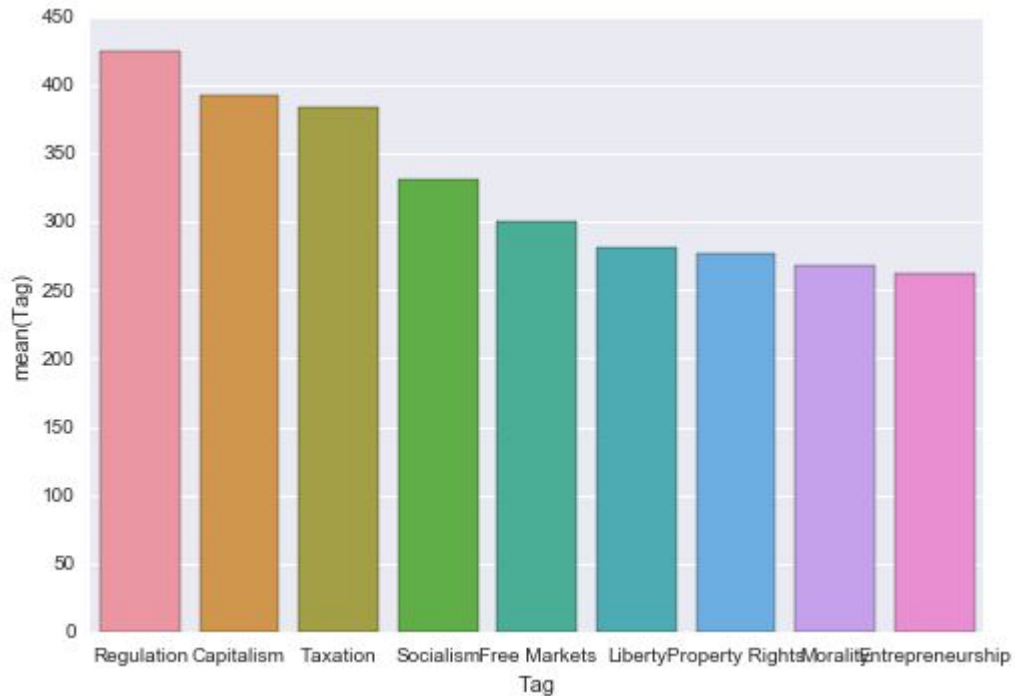


```
articles_by_topic=data.groupby('Topic').Topic.c  
articles_by_topic
```

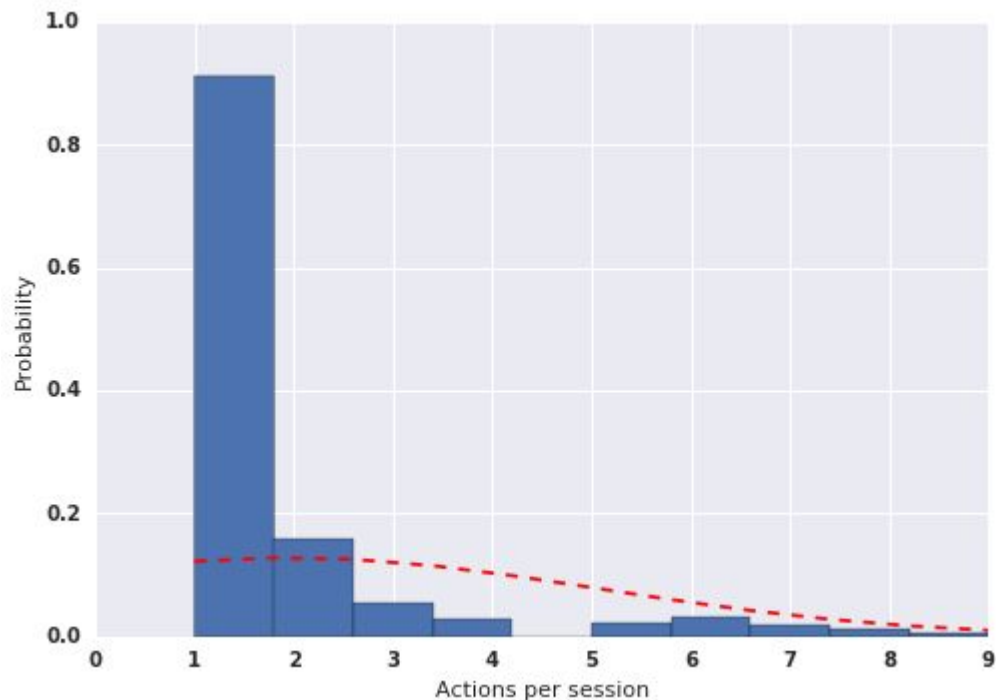
```
Topic  
Culture          329  
Economics       1018  
Entrepreneurship 207  
History          152  
Justice          420  
Philosophy       728  
Policy          1473  
Politics         486  
Technology        92  
World           341  
Name: Topic, dtype: int64
```

```
g = sns.factorplot(x="Topic", data=data, kind='  
g.set_xticklabels(step=2)  
g.set_xticklabels(rotation=90)
```


Top Editorial Tags



Actions per session

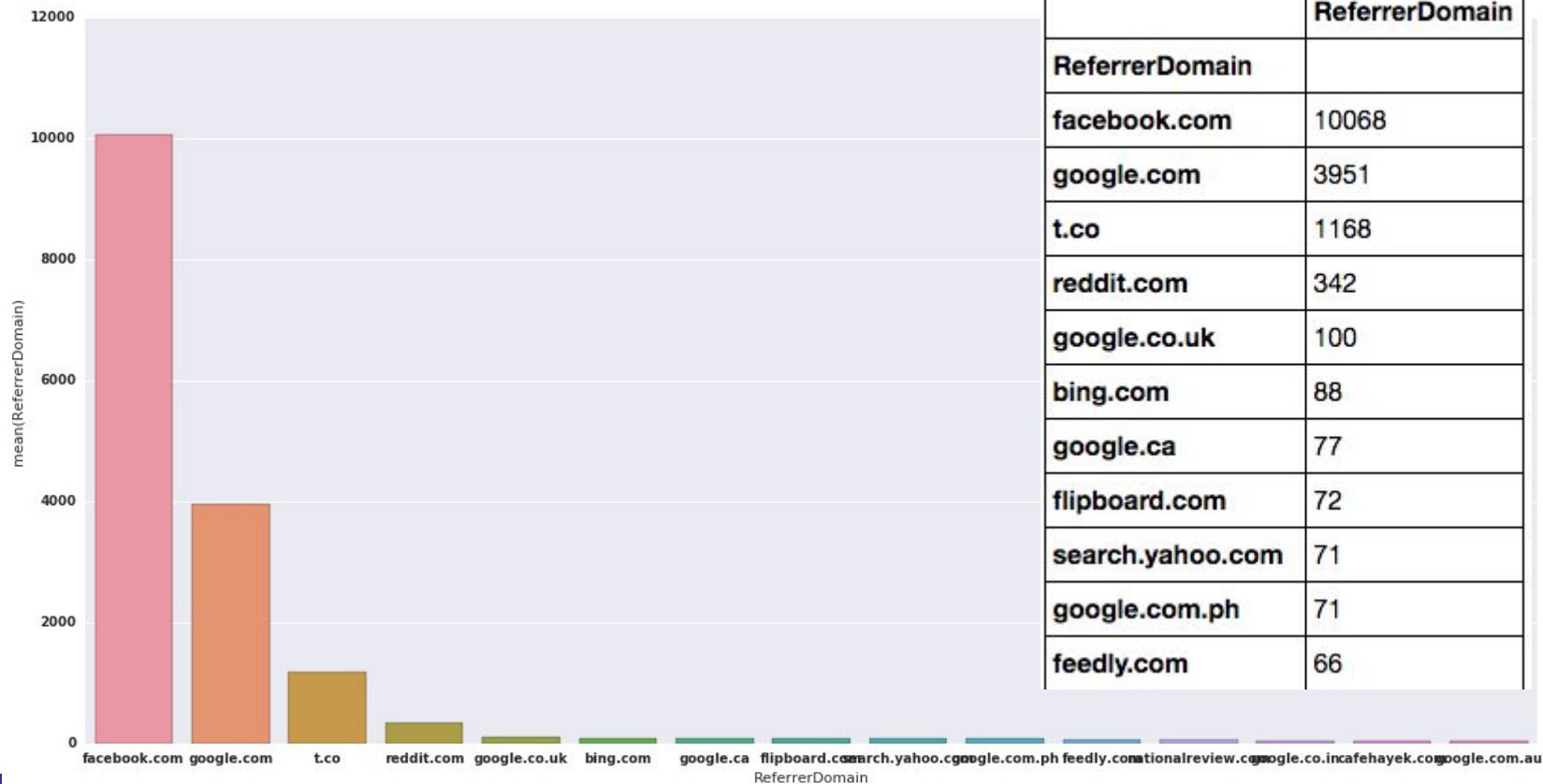


```
sigma = float(np.std(sessions_data))
print("std: %f" % sigma)
mu = float(np.mean(sessions_data))
print("mean: %f" % mu)
num_bins = 10
n, bins, patches = plt.hist(sessions_data[sessions_data > 0],
                             num_bins, normed=1)
plt.xlabel("Actions per session")
plt.ylabel('Probability')
## add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)

plt.plot(bins, y, 'r--')
```

std: 3.113452
mean: 1.962015

Top Referring domains



Part 2: Similar Article Recommendations

Policy Money Payday Loans Supply and Demand

What If You Needed a Loan, Like Now?



Abigail Blanco

Thursday, August 11, 2016



{...}

Related Articles



A Payday Loan Can Be A Lifeline

Paige Marta Skiba - June 07, 2016

👍 0 Comments



The Hidden Tax That Costs Households up to \$1,500 a Year

Salim Furth - April 18, 2016

👍 3 Comments



9 Ways Austin Blocks New Housing in Central City

Dan Keshet - April 07, 2016

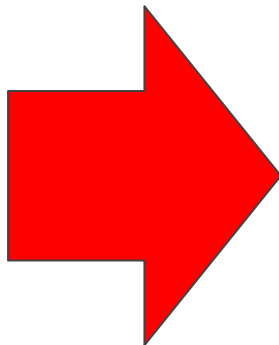
👍 0 Comments



Feds' Crazy Plan: Make Risky Loans, Don't Charge for Them

Ike Brannon - March 17, 2016

👍 0 Comments



Process

Extract metadata

**Build script which
extracts all content from
the CMS**

Extract features

**Convert all text fields in
each article to a term
frequency-inverse
document frequency
feature matrix**

Find related content

**Given a URL, return 5
similar articles**

Step 1: Extract all text fields & URLs

1. Build command-line interface to Umbraco Content Management System
2. Strip HTML.
3. Encode text to JSON.
4. Extract all values to a CSV file.

```
var articles = access.Services.ContentService.GetChildren((int)CmsTopLevelContentNodes.Articles);
var metas = new List<ArticleMetadata>();
    articles.ForEach(article =>
    {
        Console.WriteLine("{0}: {1}", article.Name, article.ToString());
        metas.Add(GetArticleMetadata(article));
    });

WriteMetaDataToFile(metas);
```

Step 2: Pre-process CSV to Pandas Pickle

1. Convert CSV to Pandas DataFrame
2. Parse dates and convert tag to list.
3. Decode JSON strings
4. Strip HTML from content `from html.parser import HTMLParser`
5. Save DataFrame to Pandas pickle `data.to_pickle('assets\dataset\ArticleMetadata.pkl')`

```
import pandas as pd
import json

articles = pd.read_pickle('ArticleMetadata.pkl')
articles.DatePublished = pd.to_datetime(articles.DatePublished)
articles.Tags = articles.Tags.map(lambda x: str(x))
articles.TagArray = articles.Tags.map(lambda x: x.split(','))
articles.TagArray[0]
articles.head(1)
```

	Url	Title	Tags	Topic	DatePublished	Abstract	FullText
ArticleId							
12897	/articles/amc-s-halt-and-catch-fire-is-capital...	AMC's "Halt and Catch Fire" Is Capitalism's Fl...	Capitalism,Competition,Property Rights,Entrepre...	Economics	2015-09-02 10:56:24	"The show is a vibrant look at the early PC in...	"AMC's Halt and Catch Fire is a brilliant ...

Step 3: TfidfVectorizer

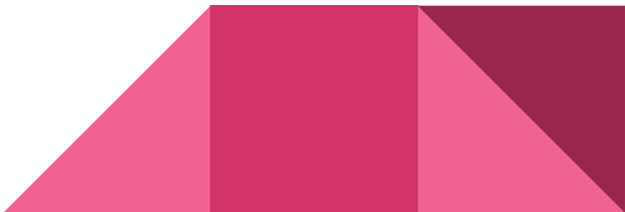
1. Extract words as features with TfidfVectorizer

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf = TfidfVectorizer(stop_words='english',min_df=3).fit_transform(articles.RawText.dropna())
```

```
# no need to normalize, since Vectorizer will return normalized tf-idf
```

```
pairwise_similarity = tfidf * tfidf.T
```



Step 4: Compute Cosine Similarity

1. Compute the linear kernel between X and Y.

```
from sklearn.metrics.pairwise import linear_kernel
```

```
def FindSimilarArticles(url, tfidf_matrix, articles):
```

```
    matches = articles.loc[articles['Url'] == url].index.tolist()
```

```
    originalArticleIndex = int(matches[0])
```

```
    print("original index: %s" % originalArticleIndex)
```

```
    cosine_similarities = linear_kernel(tfidf_matrix[originalArticleIndex], tfidf_matrix).flatten()
```

```
    print("cosine_similarities: %s" % cosine_similarities)
```

```
    related_docs_indices = cosine_similarities.argsort()[:-5:-1]
```

```
    print('related articles: ' % related_docs_indices)
```

```
    related_articles = []
```

```
    [related_articles.append(articles.iloc[index]) for index in related_docs_indices]
```

```
    return related_articles
```



Demo

```
related = FindSimiliarArticles('/articles/how-america-can-keep-the-entrepreneurs-we-train/',tfidf,articles)
```

```
print(related)
```

```
Original index: 13795
```

```
cosine_similarities: [ 0.05772269  0.04216487  0.04971372 ...,  0.00795316  0.03272186  0.024919 ]
```

```
related articles:
```

```
Url          /articles/how-america-can-keep-the-entrepreneu...
```

```
Title        How America Can Keep the Entrepreneurs We Train
```

```
Name: 13795, dtype: object, ArticleId          132805
```

```
Url          /articles/immigrants-are-twice-as-likely-to-st...
```

```
Title        Immigrants Are Twice as Likely to Start a Busi...
```

```
Name: 671, dtype: object, ArticleId          108006
```

```
Url          /articles/why-government-jobs-programs-destroy...
```

```
Title        Why Government Jobs Programs Destroy Jobs
```

```
Name: 1076, dtype: object, ArticleId          129584
```

```
Url          /articles/5-charts-that-show-trumps-immigratio...
```

```
Title        5 Charts that Show Trump's Immigration Paper I...
```

```
Name: 338, dtype: object]
```



Part 3: Other Classifier & Clustering Experiments

Other experiments:

- Keyword modeling with Word2Vec Neural Network
 - Useful for tag recommendations when tagging articles.
- Document clustering - K means
 - Incomplete, too complex
- Topic modeling with latent Dirichlet allocation
 - Not very descriptive results:

```
In [12]: num_topics= 10
num_words_per_topic= 10

for ti, topic in enumerate(lda_model.show_topics(num_topics,num_words_per_topic)):
    print("Topic:      %d" % (ti))
    print(topic)
    print()

Topic:  0
(0, '0.001*government + 0.000*tax + 0.000*people + 0.000*moore + 0.000*tubman + 0.000*soto + 0.
*education + 0.000*trade + 0.000*new + 0.000*economic')

Topic:  1
(1, '0.001*government + 0.001*venezuela + 0.001*market + 0.001*free + 0.001*state + 0.001*madur
0.001*erhard + 0.001*economic + 0.001*jury + 0.001*amazon')

Topic:  2
(2, '0.002*government + 0.002*people + 0.002*market + 0.001*economic + 0.001*world + 0.001*free
0.001*state + 0.001*new + 0.001*money + 0.001*percent')
```

```
In [18]: model.most_similar(positive=
```

```
Out[18]: [('entrepreneurship', 0.7297
('protectionism', 0.7233507
('global', 0.70528000593185
('capitalism', 0.6833494901
('instability', 0.667334139
('technological', 0.6574413
('liberalization', 0.652382
('overpopulation', 0.638936
('sustainable', 0.634416639
('specialization', 0.627989
```



The End