STUDY OF THE ELECTORAL SYSTEM AND VOTING BEHAVIOUR AT THE ELECTION COMMISSION OF INDIA

Dissertation submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

Computer Science with specialization in Data Analytics

By

RITIKA SINGH

Department of Computer Science

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

(Declared Deemed to be University U/S 3 of UGC Act)

A-10, SECTOR-62, NOIDA, INDIA

May 2019

TABLE OF CONTENTS

DECLARATION BY THE SCHOLAR

I hereby declare that the work reported in the MTech. Dissertation entitled **“STUDY OF THE ELECTORAL SYSTEM AND VOTING BEHAVIOUR AT THE ELECTION COMMISSION OF INDIA”** submitted at **Jaypee Institute of Information Technology, Noida, India,** is an authentic record of my work carried out under the supervision of Dr. **Satish Chandra.** I have not submitted this work elsewhere for any other degree or diploma. I am fully responsible for the contents of my MTech Thesis.

(Ritika Singh)

Department of Computer Science

Jaypee Institute of Information Technology, Noida, India

20-05-2019

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M. Tech Dissertation entitled "**STUDY OF THE ELECTORAL SYSTEM AND VOTING BEHAVIOUR AT THE ELECTION COMMISSION OF INDIA**", submitted by **Ritika Singh** at **Jaypee Institute of Information Technology, Noida**, **India,** is a bonafide record of her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

(Dr. Satish Chandra)

Associate Professor

20-05-2019

ABSTRACT

My dissertation reflects my work conducted at the Election Commission of India Headquarters, New Delhi as the Statistical Expert for Lok Sabha Elections 2019.

This dissertation explores the first election to use big data analytics. The focus is to establish an understanding of the impact of analytics and machine learning algorithms amongst both voters and candidates alike. In the progressing general decisions 2019 in India, innovation hosts made it simpler for political gatherings to measure the temperament of the voters and put their best message forward. Ideological groups are utilizing huge information examination to assemble bits of knowledge into voter inclinations dependent on their financial status, rank, nearby issues, and different parameters. In light of the voter conclusion and fragments, modified decision crusades with the most significant messages and recordings are being made and pitched to the particular target gatherings. When the crusade is propelled, information is assembled to dissect its viability and change it further as indicated by the reaction it creates.

KEYWORDS

sentiment analysis, artificial neural networks, feed-forward back propagation neural networks, opinion mining, social media platforms, Internet, organizations, python library, TextBlob, twitter, Facebook, news Websites, ANN, min-max approach, prediction accuracy, backpropagation, data mining, feedforward neural nets, Internet, minimax techniques, Python, sentiment analysis, social networking (online)

INTRODUCTION

The Election Commission of India is a self-sufficient sacred specialist in charge of controlling race forms in India. The body manages decisions to the Lok Sabha, Rajya Sabha, state Legislative Assemblies, state administrative Councils, and the workplaces of the President and Vice President of the nation.

Over the decades, the result of the political elections was analyzed and predicted by pundits and political analysts. These predictions were often biased as they were produced using the person's personal experiences and had traces of mere intuitions. Keeping the effect of personal biases in mind nowadays, the trends have shifted to more of scientific approaches. The election results, voter turnout percentages and party performances are now predicted statistically using a poll. Sample of voter data is collected to develop test cases and from that the result of the elections are extrapolated.

Political races are known to be hard to anticipate, because of their results being the result of an assortment of components. Just in this present decade have purported "surveyors" or analysts picked up the capacity to precisely foresee Lok Sabha decisions. Given the limit of AI to learn associations between components, we apply a combination of AI strategies to the errand of envisioning race results from money related and factual data. All the more explicitly, we attempt to foresee and clarify the outcomes from the Lok Sabha 1951-2014 decisions, and to assess the impact of different factors on their results.

From this information, we extract different statistic and financial highlights, for example, PC Number name, PC Type, Candidate Name, Candidate Sex, Candidate Category, Candidate Age, Party Abbreviation, Total Votes Polled and Position. In spite of the fact that the information we wish to utilize as of now exists, it is spread crosswise over numerous documents and arrangements. We in this manner perform critical preprocessing and solidification arranges so as to acquire our last informational index.

This report incorporates work accomplished for the Lok Sabha races led by ECI crosswise over 17 Lok Sabha Elections from 1951-2019. The data used was scraped from approved interior sources, converted to a database reasonable for long run. Enormous information and investigation additionally helped drive the battle's advertisement purchasing choices, which brought about obtaining promotions amid offbeat programming and schedule vacancies. Here again the group depended on big data analytics, enormous information examination instead of on outside media specialists and specialists to choose where and when advertisements should run. At last, this data-driven approach demonstrated fruitful in getting the messages out to the focused-on watchers and driving the turnout in states.

The expression "Big Data" came into spotlight when conventional database devices like "RDBMS" become unable to deal with huge, unstructured information which is described by high volume, speed, and assortment. Separating needful data from this huge information is one of the primary difficulties for the two experts and database. Consistently around 2.5 quintillion bytes of information are made. With this bounty of information, we can without much of a stretch create some meaningful data by applying reasonable strategies to the informational index. Decisions are held all through the world covering practically all the whole countries. The decision is a procedure by which general public can pick their delegate by throwing their votes. Each country has various terms and standards for the election procedure. Social Media is an online application stage which encourages interaction, joint effort, and sharing of substance. Both open, just as political pioneers, utilize internet-based life like Twitter, Facebook, and Google+ and so on for the battle, discourse, forecast, and investigation of the race. These web based life particularly Twitter and Facebook create a tremendous measure of crude information which is extremely helpful for both ideological groups and overall population especially amid race times. Consequently, ideological groups utilize web-based life since Politicians with higher online life commitment got moderately more votes inside most ideological groups and furthermore it builds their battle since fan base of driving political pioneers expanded with the beginning of their advanced crusade during the decisions. So Political gatherings, even in creating nations, try cognizant endeavors to oversee internet-based life appropriately amid their crusading stage. Significance of web-based life comes into the spotlight when Barrack Obama won 2008 Presidential decision of America by utilizing web-based life (Twitter) in his battling.

DATA COLLECTION

A critical component to great research is the precise and effective gathering and readiness of data for analysis. Scholastic specialists have to spend exorbitant time cleaning data and training data models preventing inclusion of garbage data and recording mistakes. The execution of straightforward rules dependent on procedures utilized by expert information supervisory groups will spare analysts time and cash and result in an informational collection more qualified to address investigate questions. Since Microsoft Excel is regularly utilized by analysts to gather information, explicit methods that can be actualized in Excel are exhibited.

For our dataset, we utilize an assortment of socio-economic information acquired from data.gov.in, www.nic.in, http://eci.gov.in, https://eci.gov.in/statistical-report/statistical-reports and http://www.indiavotes.com joined with the area wide decision results from the 1951 to 2019 races. So as to make the information as updated as could be allowed, we use information from the internal sources of Election Commission of India. (any dissemination, use, review, distribution, printing or copying of the information contained in this report and/or attachments to it are strictly prohibited.). These reviews are performed yearly from a smaller bunch of regions and incorporate an abundance of data, as opposed to the national statistics which is played out at regular intervals and just gathers populace numbers. This choice empowers us to prepare on various information for the 1951-2019 elections.

We presently examine the source of our data inside and out. To acquire our information, we utilize an information gathering highlight to crawl and scrape data and APIs, which enables the access of different datasets without a moment's delay and their downloading as a bundle. Through this strategy, we downloaded the crude information for the 1951-2014 years. All the more explicitly, we download informational indexes relating to the significant subjective classes of public sentiments.

In the wake of acquiring the crude informational collections as our basic raw datasets, we start the way toward cleaning and consolidating them. To do as such, we previously changed over the informational collections to csv documents. Afterwards, we make a technique that joins numerous different csv records into a solitary Pandas data frame. Following this, we use Pandas to expel numerous repetitive sections, which compare to highlights, for example, deface gins of mistake, definitions of passages in substitute units or rates, and segments containing generally or every unfilled passage. Following this, we join the records and match the information passages with their relating region level outcomes. We treat missing qualities as NaNs in Pandas for the motivations behind our analysis.

SOURCE

Election Results - Full Statistical Reports

STATISTICAL REPORTS OF GENERAL ELECTION TO LOKSABHA

STATISTICAL REPORTS OF GENERAL ELECTION TO STATE LEGISLATIVE ASSEMBLY (VIDHANSABHA)

STATES	YEARS
Andhra Pradesh	[illegible]
Arunachal Pradesh	[illegible]
Assam	[illegible]
Bihar	[illegible]
Chhattisgarh	[illegible]

Madhya Pradesh Legislative El...

Mizoram Legislative Election 20...

Karnataka 2018

Goa General Legislative Electio...

Figure 2.1: Source of data extraction

SCRAPPER FOR THE DATA COLLECTION

In [1]:

```
import os
import re
import urllib
from hashlib import sha256
from lxml.html import parse
```

In [2]:

```
def get(url):
    """Retrieves a URL as an lxml tree, cached where possible"""
    filename = '.cache.' + sha256(url).hexdigest()
    if not os.path.exists(filename):
        html = urllib.urlretrieve(url, filename)
    return parse(filename)
```

In [3]:

```
def constituencies(url):
    """Yields dicts with state, state_code, constituency, constituency_code."""
    tree = get(url)
    statecode = re.findall('st.value *= *\'([^\']+).*?HdnFld([^\']+)',
                           tree.findall('.//script')[0].text, re.S)
    statecode = {state:code for code, state in statecode}

    # Constituency codes are in hidden input fields. Format is:
    # code,constituency; code,constituency; ...
    for el in tree.findall('.//input[@id]'):
        id = el.get('id', '').strip()
        if id.startswith('HdnFld'):
            state = id.replace('HdnFld', '')
            for row in el.get('value').split(';'):
                row = row.strip()
                if row:
                    cells = row.split(',')
                    yield {
                        'state': state,
                        'statecode': statecode.get(state),
                        'constituency': cells[1],
                        'constituencycode': cells[0]
                    }
```

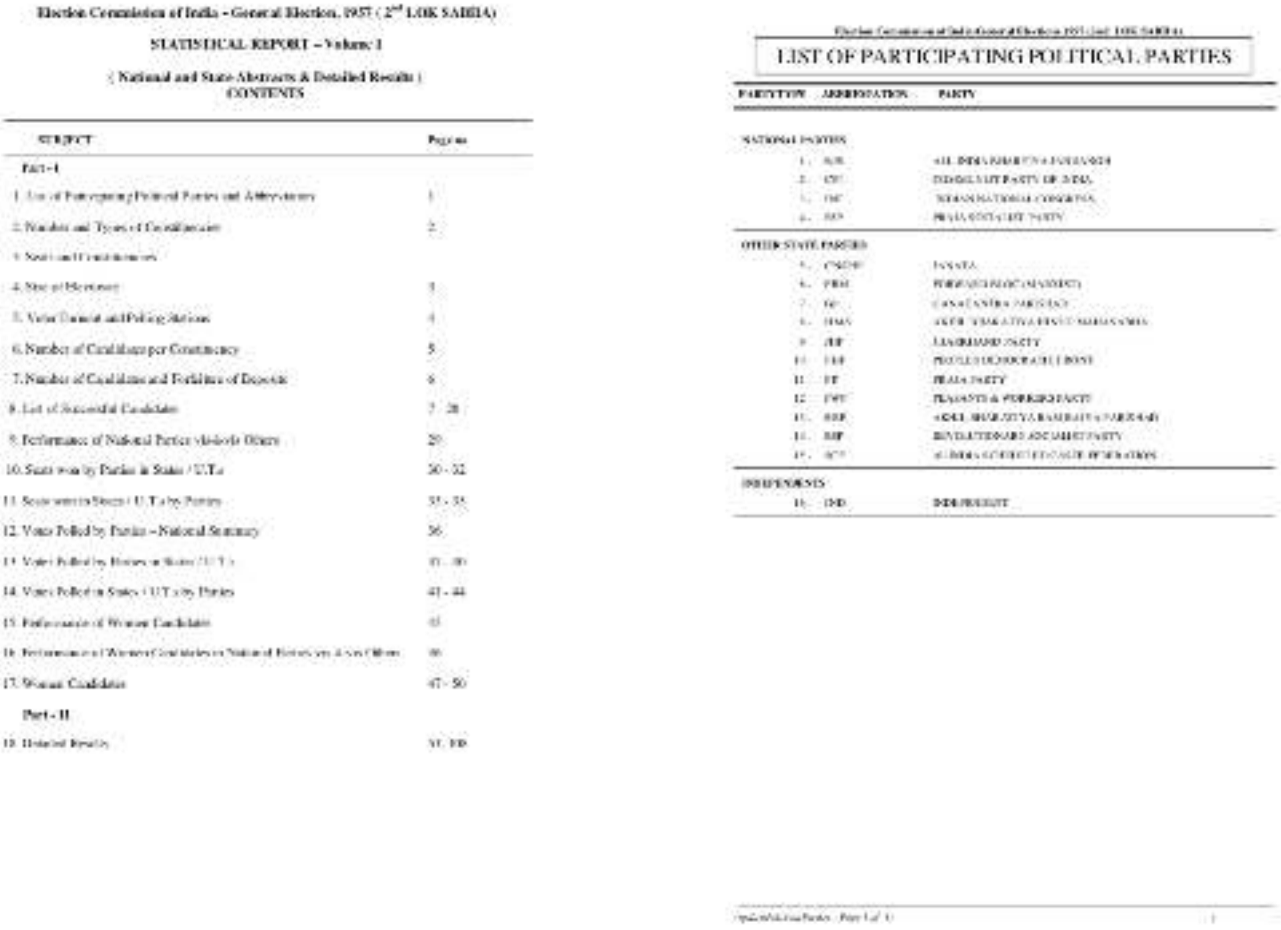
In [4]:

```
def results(url):
    """For a constituency URL, yields dicts with candidate, party, votes."""
    tree = get(url)

    # Results are inside a table in a <div id="div1">
    for row in tree.findall('.//*[@id="div1"]//tr'):
        cells = row.findall('td')
        if len(cells) >= 3:
            yield {
                'candidate': cells[0].text.strip(),
                'party': cells[1].text.strip(),
                'votes': cells[2].text.strip(),
            }
```

In [7]:

```
dataset = []
for place in constituencies('http://eciresults.nic.in/ConstituencywiseS2653.htm'):
    url = "http://eciresults.nic.in/Constituencywise{:s}{:s}.htm?ac={:s}".format(
        place['statecode'], place['constituencycode'], place['constituencycode'])
    # print 'Debug: scraping', place['state'], place['constituency']
    for result in results(url):
        result.update(place)
        dataset.append(result)

with open('2013-result.txt', 'wb') as out:
    fields = ['state', 'constituency', 'votes', 'candidate', 'party']
    out.write('\t'.join(fields) + '\n')
    for row in dataset:
        out.write('\t'.join(row[f] for f in fields).encode('utf-8') + '\n')
```

SAMPLE RESULT FROM DATA SCRAPPER



Election Commission of India - General Election, 1957 (2nd LOK SABHA)

STATISTICAL REPORT – Volume I

(National and State Abstracts & Detailed Results)
CONTENTS

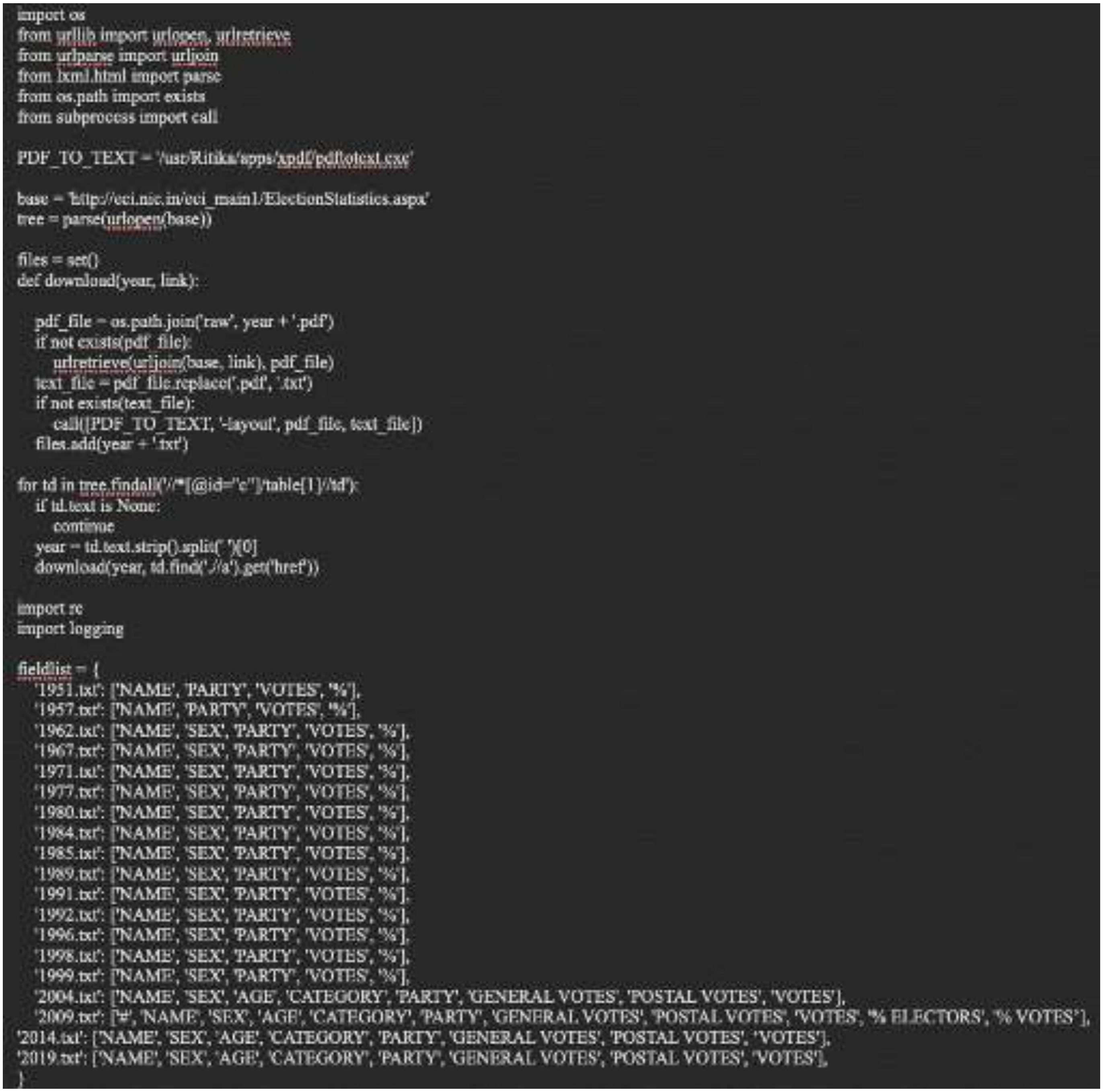
SUBJECT	Pages
Part - I	
1. List of Participating Political Parties and Abbreviations	1
2. Number and Types of Constituencies	2
3. [illegible]	
4. Size of Electorate	3
5. Voter Turnout and Polling Stations	4
6. Number of Candidates per Constituency	5
7. Number of Candidates and Forfeiture of Deposits	6
8. List of Successful Candidates	7 - 28
9. Performance of National Parties vis-a-vis Others	29
10. Seats won by Parties in States / U.T.s	30 - 32
11. Seats won in States / U.T.s by Parties	33 - 35
12. Votes Polled by Parties – National Summary	36
13. Votes Polled by Parties in States / U.T.s	37 - 40
14. Votes Polled in States / U.T.s by Parties	41 - 44
15. Performance of Women Candidates	45
16. Performance of Women Candidates in National Parties vis-a-vis Others	46
17. Women Candidates	47 - 50
Part - II	
18. Detailed Results	51 - [illegible]

LIST OF PARTICIPATING POLITICAL PARTIES

PARTY TYPE	ABBREVIATION	PARTY
NATIONAL PARTIES		
1 .	ABJS	ALL INDIA BHARTIYA JAN SANGH
2 .	CPI	COMMUNIST PARTY OF INDIA
3 .	INC	INDIAN NATIONAL CONGRESS
4 .	PSP	PRAJA SOCIALIST PARTY
OTHER STATE PARTIES		
5 .	CNSPJP	JANATA
6 .	FBL(M)	FORWARD BLOC (MARXIST)
7 .	GP	GANATANTRA PARISHAD
8 .	HMS	AKHIL BHARATIYA HINDU MAHASABHA
9 .	JHP	JHARKHAND PARTY
10 .	PDF	PEOPLES DEMOCRATIC FRONT
11 .	PP	PRAJA PARTY
12 .	PWP	PEASANTS & WORKERS PARTY
13 .	RRP	AKHIL BHARTIYA RAM RAJYA PARISHAD
14 .	RSP	REVOLUTIONARY SOCIALIST PARTY
15 .	SCF	ALL INDIA SCHEDULED CASTE FEDERATION
INDEPENDENTS		
16 .	IND	INDEPENDENT

Figure 2.2: Result of the data collected

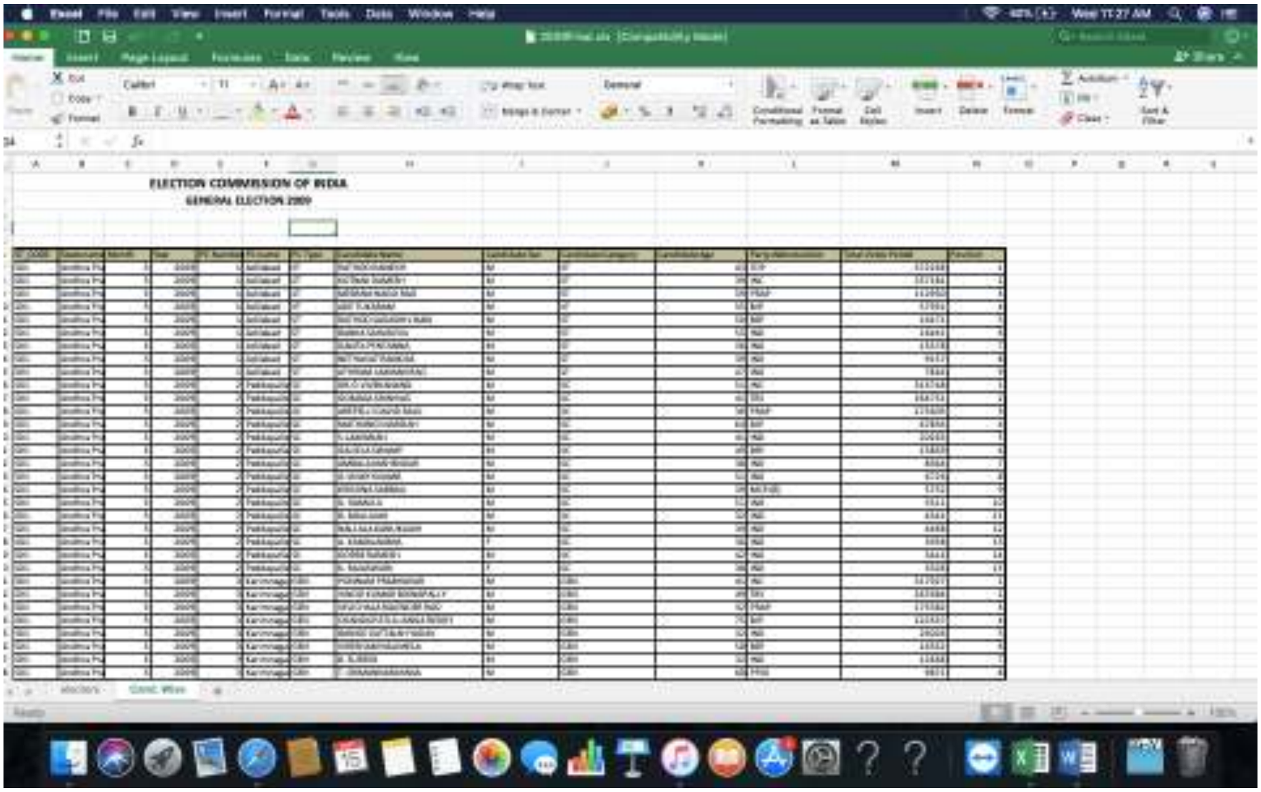
ECI OPEN DATA- CRAWL THE ECI ELECTION STATISTICS



```
import os
from urllib import urlopen, urlretrieve
from urlparse import urljoin
from lxml.html import parse
from os.path import exists
from subprocess import call

PDF_TO_TEXT = '/usr/Ritika/apps/xpdf/pdftotext.exe'

base = 'http://eci.nic.in/eci_main1/ElectionStatistics.aspx'
tree = parse(urlopen(base))

files = set()
def download(year, link):

    pdf_file = os.path.join('raw', year + '.pdf')
    if not exists(pdf_file):
        urlretrieve(urljoin(base, link), pdf_file)
    text_file = pdf_file.replace('.pdf', '.txt')
    if not exists(text_file):
        call([PDF_TO_TEXT, '-layout', pdf_file, text_file])
    files.add(year + '.txt')

for td in tree.findall('//*[@id="c"]/table[1]//td'):
    if td.text is None:
        continue
    year = td.text.strip().split(' ')[0]
    download(year, td.find('.//a').get('href'))

import re
import logging

fieldlist = {
    '1951.txt': ['NAME', 'PARTY', 'VOTES', '%'],
    '1957.txt': ['NAME', 'PARTY', 'VOTES', '%'],
    '1962.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1967.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1971.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1977.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1980.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1984.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1985.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1989.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1991.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1992.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1996.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1998.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '1999.txt': ['NAME', 'SEX', 'PARTY', 'VOTES', '%'],
    '2004.txt': ['NAME', 'SEX', 'AGE', 'CATEGORY', 'PARTY', 'GENERAL VOTES', 'POSTAL VOTES', 'VOTES'],
    '2009.txt': ['#', 'NAME', 'SEX', 'AGE', 'CATEGORY', 'PARTY', 'GENERAL VOTES', 'POSTAL VOTES', 'VOTES', '% ELECTORS', '% VOTES'],
'2014.txt': ['NAME', 'SEX', 'AGE', 'CATEGORY', 'PARTY', 'GENERAL VOTES', 'POSTAL VOTES', 'VOTES'],
'2019.txt': ['NAME', 'SEX', 'AGE', 'CATEGORY', 'PARTY', 'GENERAL VOTES', 'POSTAL VOTES', 'VOTES'],
}
```

```
def old_text_parse(filename):
    if filename.startswith('1'):
        re_state = re.compile(r'^ {25,}[A-Za-z].*')
        re_electors = re.compile(r'ELECTORS *: *(\d+)')
    else:
        re_state = re.compile(r'^[A-Z][A-Za-z& ]+$')
        re_electors = re.compile(r'Total Electors *(\d+)(.*)')

    re_constituency = re.compile(r'Constituency *:? *(\d+) *\.? *(.*)', re.IGNORECASE)
    re_name = re.compile(r'^\d+ *\. *')
    re_scst = re.compile(r' *\((SC|ST)\)')

    fields = fieldlist[filename]
    results, electors = [], {}
    state, constituency = None, None
    for ln, line in enumerate(open(filename)):
        match = re_constituency.match(line)
        if match:
            constituency = match.group(2).split('  ')[0].upper()
            constituency = re_scst.sub('', constituency)
            continue

        match = re_state.match(line)
        if match:
            state = line.strip().upper()
            continue

        match = re_electors.match(line)
        if match:
            electors[state, constituency] = match.group(1)
            continue

        parts = re.split(r'  +', line.strip())
        if len(parts) == len(fields):
            row = dict(zip(fields, parts))
        elif len(parts) == 1:
            row['NAME'] = row['NAME'] + ' ' + line.strip()
            continue
        else:
            logging.warn('%s:%d: %d parts, not %d: %s',
                         filename, ln + 1, len(parts), len(fields), line)
            continue

        row['STATE'] = state
        row['PC'] = constituency
        row['NAME'] = re_name.sub('', row['NAME'])
        results.append(row)
```

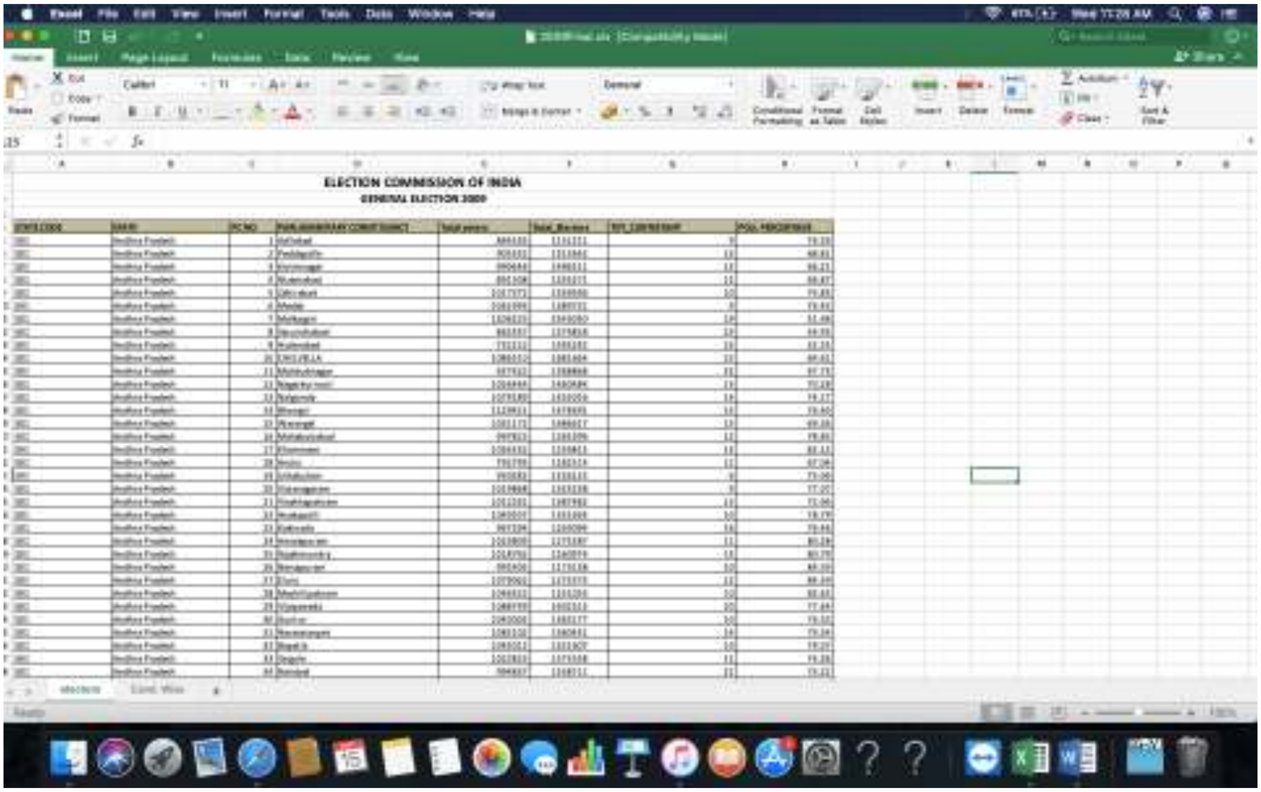
```
    results = pd.DataFrame(results).set_index(['STATE', 'PC'])
    results['YEAR'] = filename.split('.')[0]
    results['ELECTORS'] = pd.Series(electors)
    if '%' in results:
        del results['%']
    return results.reset_index()

logging.basicConfig(level=logging.INFO)

results = []
for filename in sorted(fieldlist):
    results.append(old_text_parse(filename))

results = pd.concat(results, ignore_index=True)['YEAR STATE PC NAME SEX PARTY AGE CATEGORY VOTES ELECTORS'.split(' ')]

rename = pd.read_csv('ELECTORS.csv').set_index(['Field', 'Source'])['Target']
for col in rename.index.get_level_values(0).unique():

    results[col].replace(rename.ix[col].to_dict(), inplace=True)

results['VOTES'] = results['VOTES'].astype(float)
results['#'] = results.groupby(['YEAR', 'STATE', 'PC'])['VOTES'].rank(method='min', ascending=False)
results.sort(['YEAR', 'STATE', 'PC', 'VOTES'], ascending=(True, True, True, False), inplace=True)
results.to_csv('parliament.csv', index=False, float_format='%.0f')
```



ELECTION COMMISSION OF INDIA

GENERAL ELECTION 2009

Figure 2.3: Result of candidate data scraped

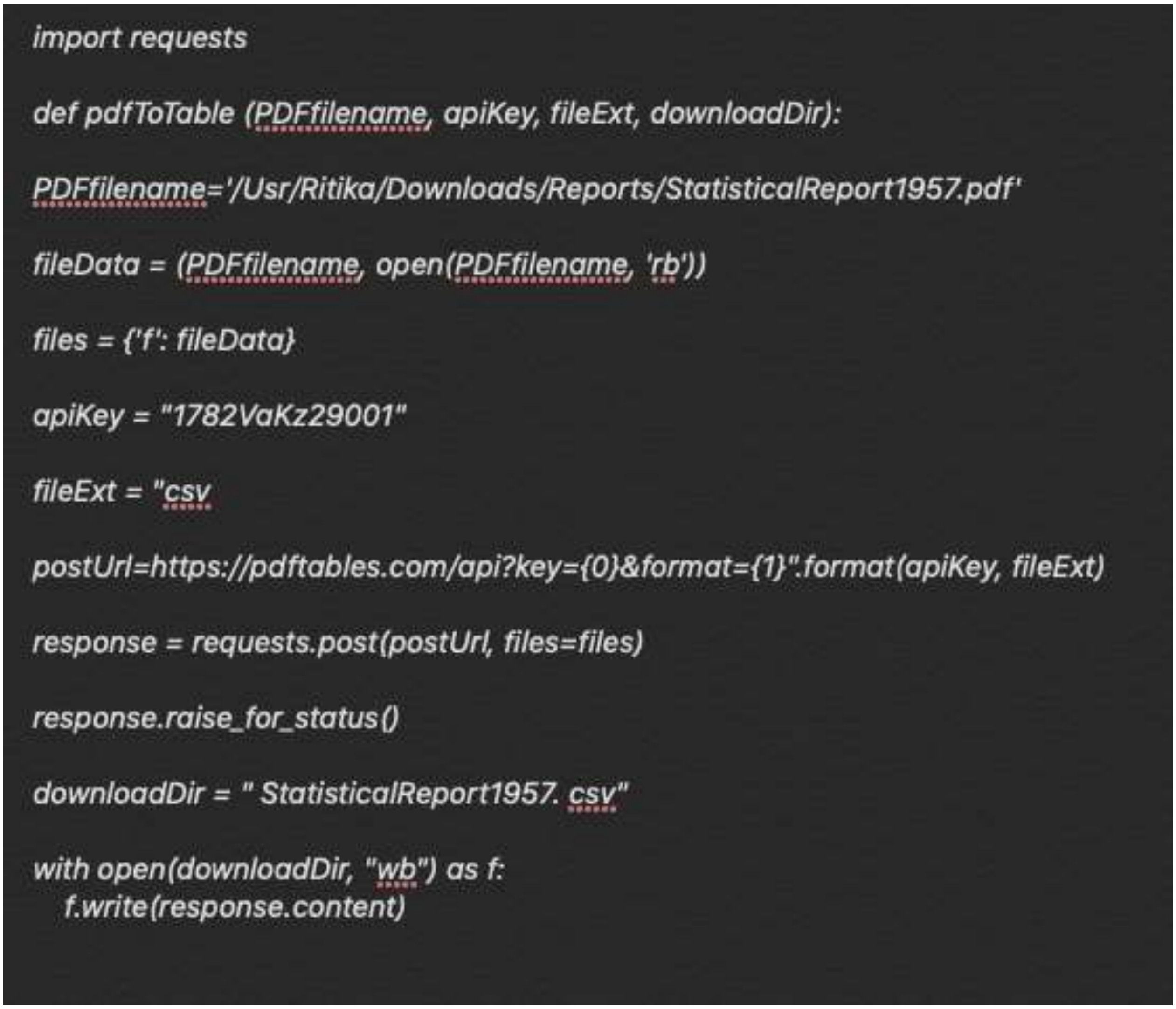


ELECTION COMMISSION OF INDIA

GENERAL ELECTION 2009

Figure 2.4: Result of electors' data scraped

PDF - EXCEL: PYTHON CODE



```
import requests

def pdfToTable (PDFfilename, apiKey, fileExt, downloadDir):

PDFfilename='/Usr/Ritika/Downloads/Reports/StatisticalReport1957.pdf'

fileData = (PDFfilename, open(PDFfilename, 'rb'))

files = {'f': fileData}

apiKey = "1782VaKz29001"

fileExt = "csv

postUrl=https://pdftables.com/api?key={0}&format={1}".format(apiKey, fileExt)

response = requests.post(postUrl, files=files)

response.raise_for_status()

downloadDir = " StatisticalReport1957. csv"

with open(downloadDir, "wb") as f:
    f.write(response.content)
```

DATABASE IN ANACONDA

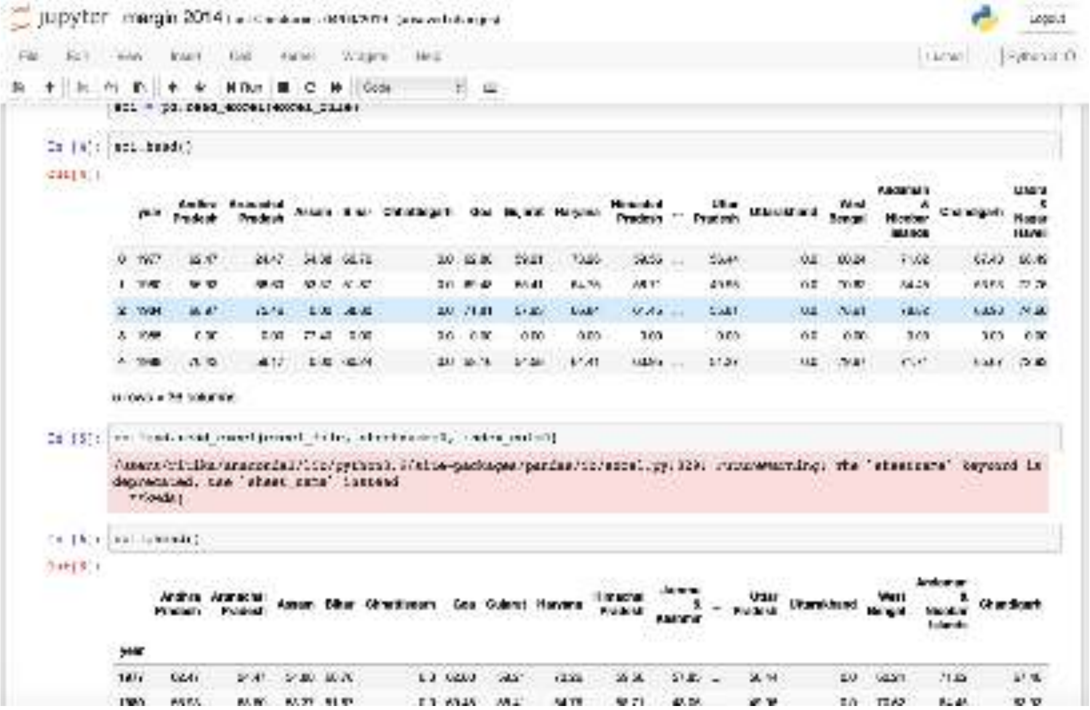


Figure 2.5: Result of the final database state-wise created in Anaconda for analysis

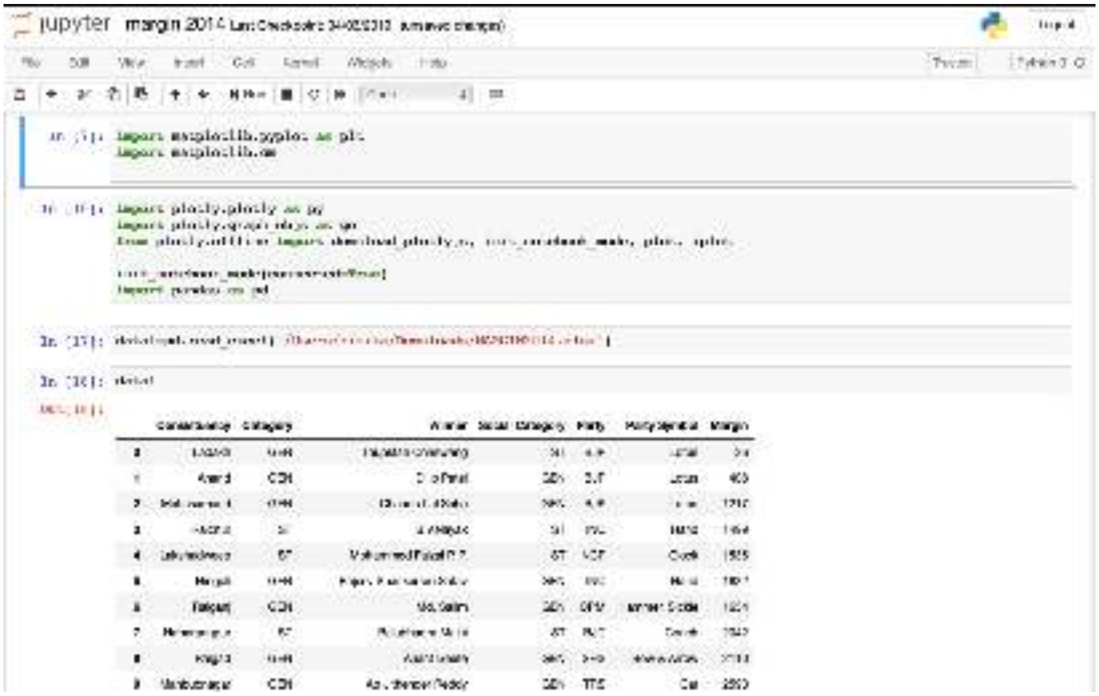


Figure 2.6: Result of the final database PC-wise created in Anaconda for analysis

TABLEAU

Data Visualization is a craft of showing the information in a way that even a non-analyst can get it. An ideal mix of tasteful components like hues, measurements, marks can make visual gems, henceforth uncovering astonishing business bits of knowledge which thusly causes organizations to settle on educated choices.

Tableau is one of the fastest evolving Business Intelligence (BI) and data visualization tool.

DATA EXTRACTION

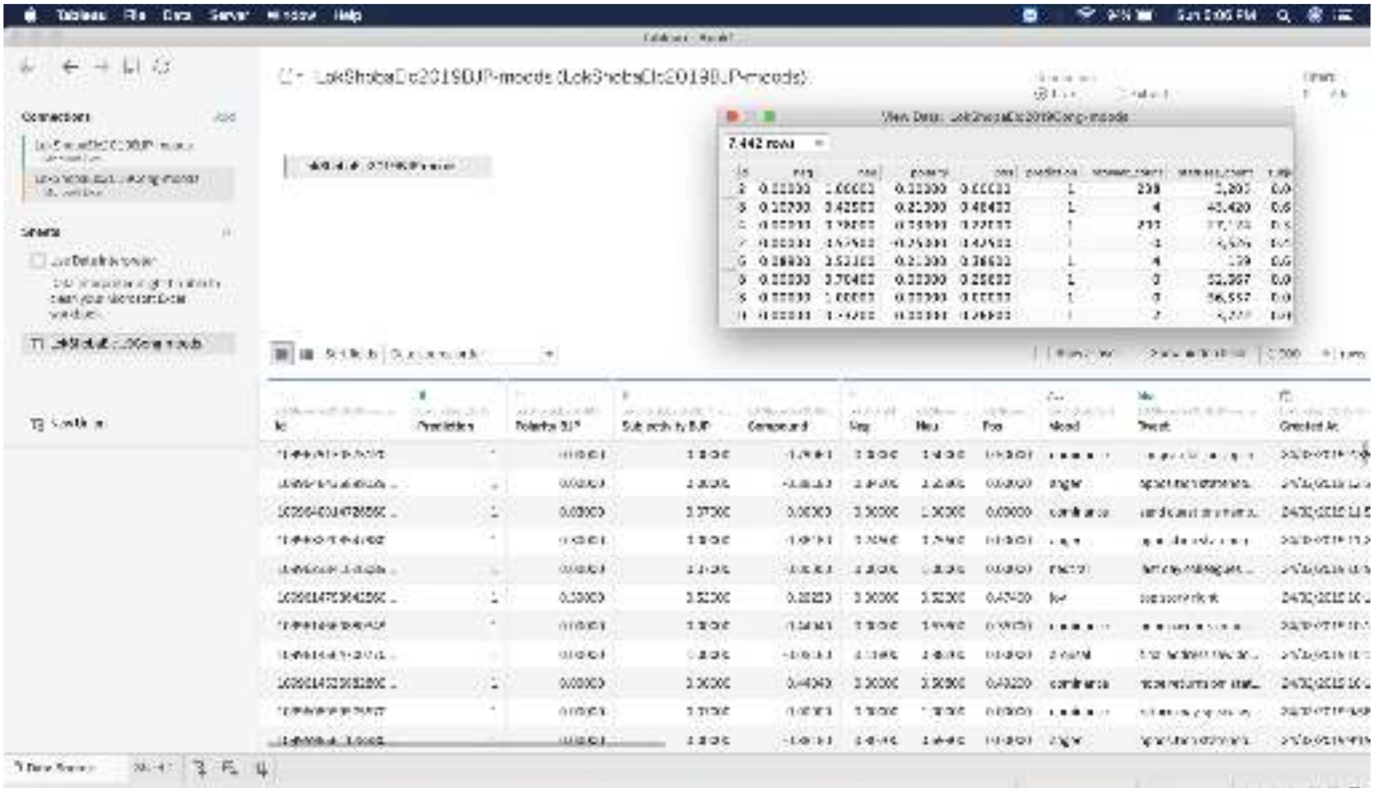


Figure 3.1: Result of data extraction in Tableau

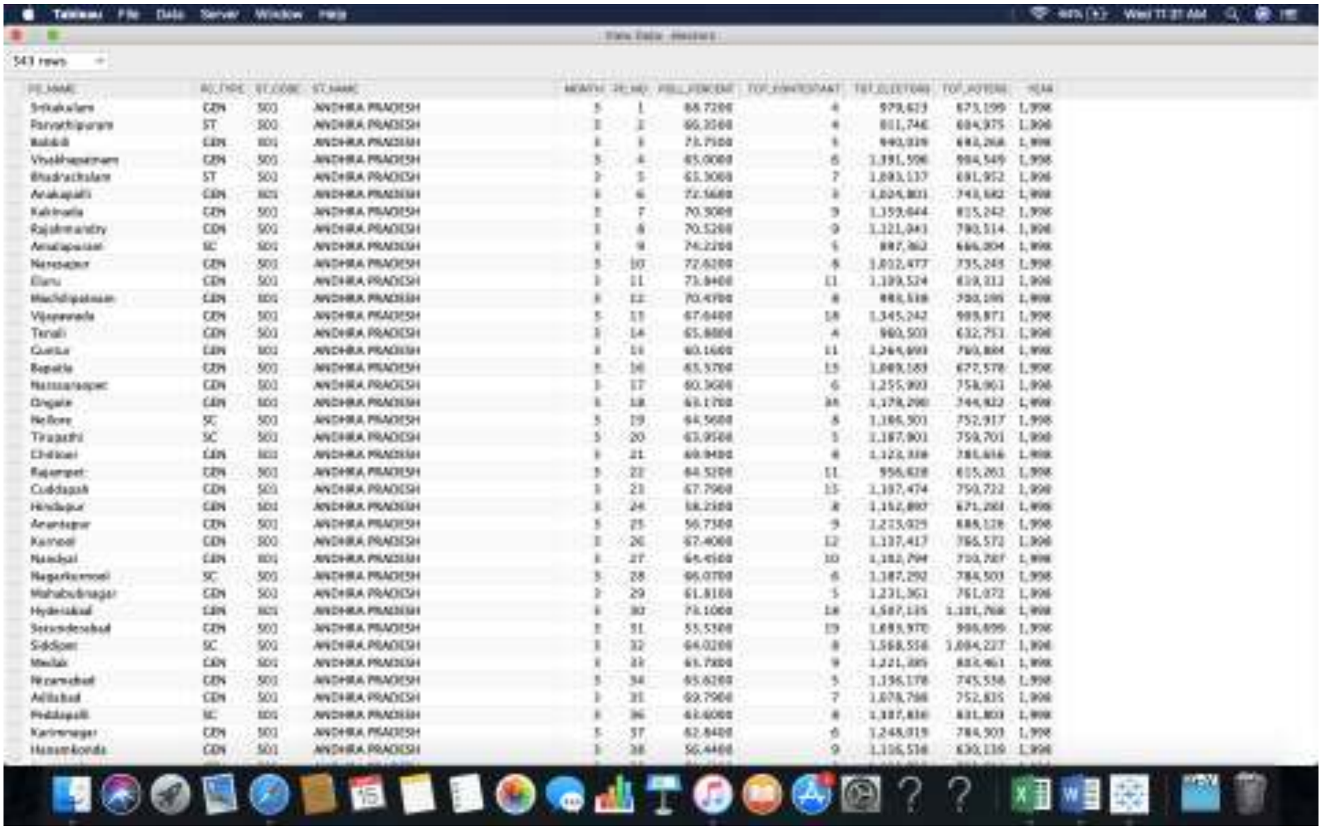


Figure 3.2: Result of data extraction PC-wise in Tableau

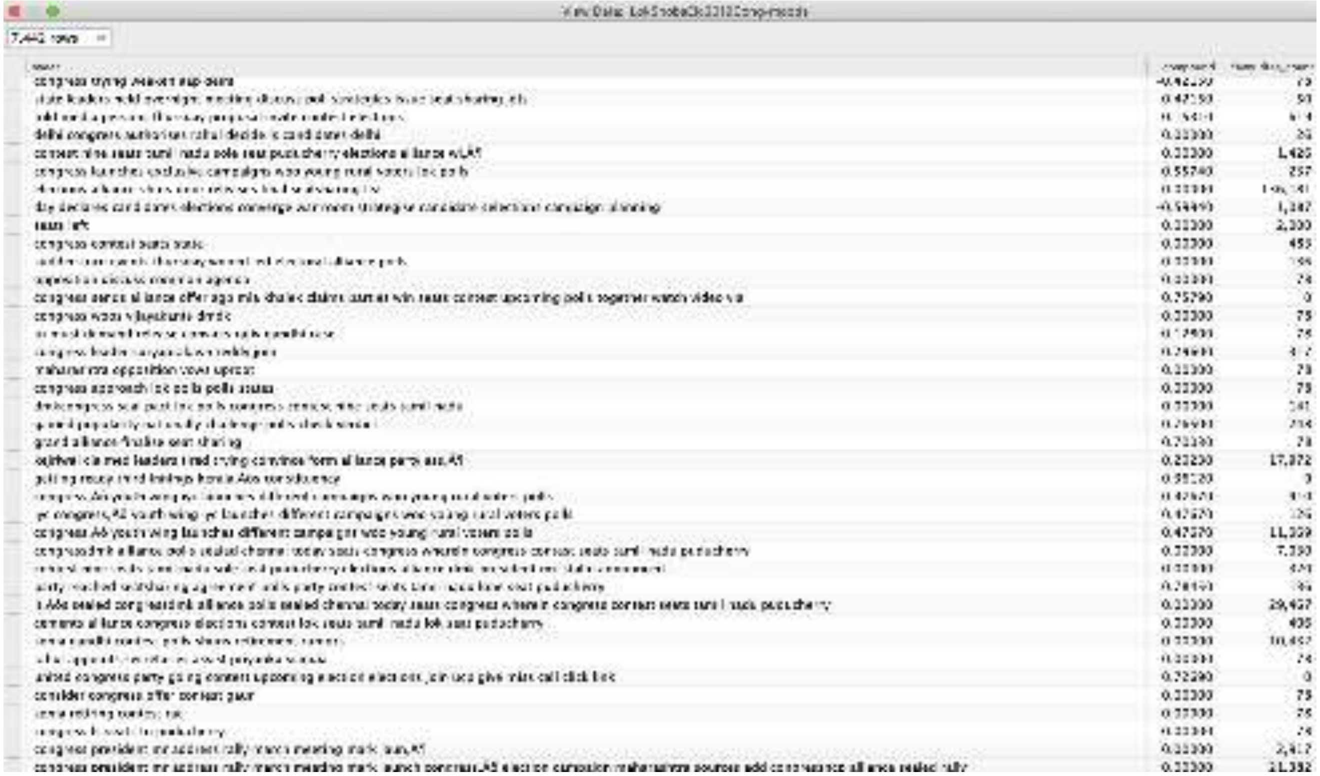
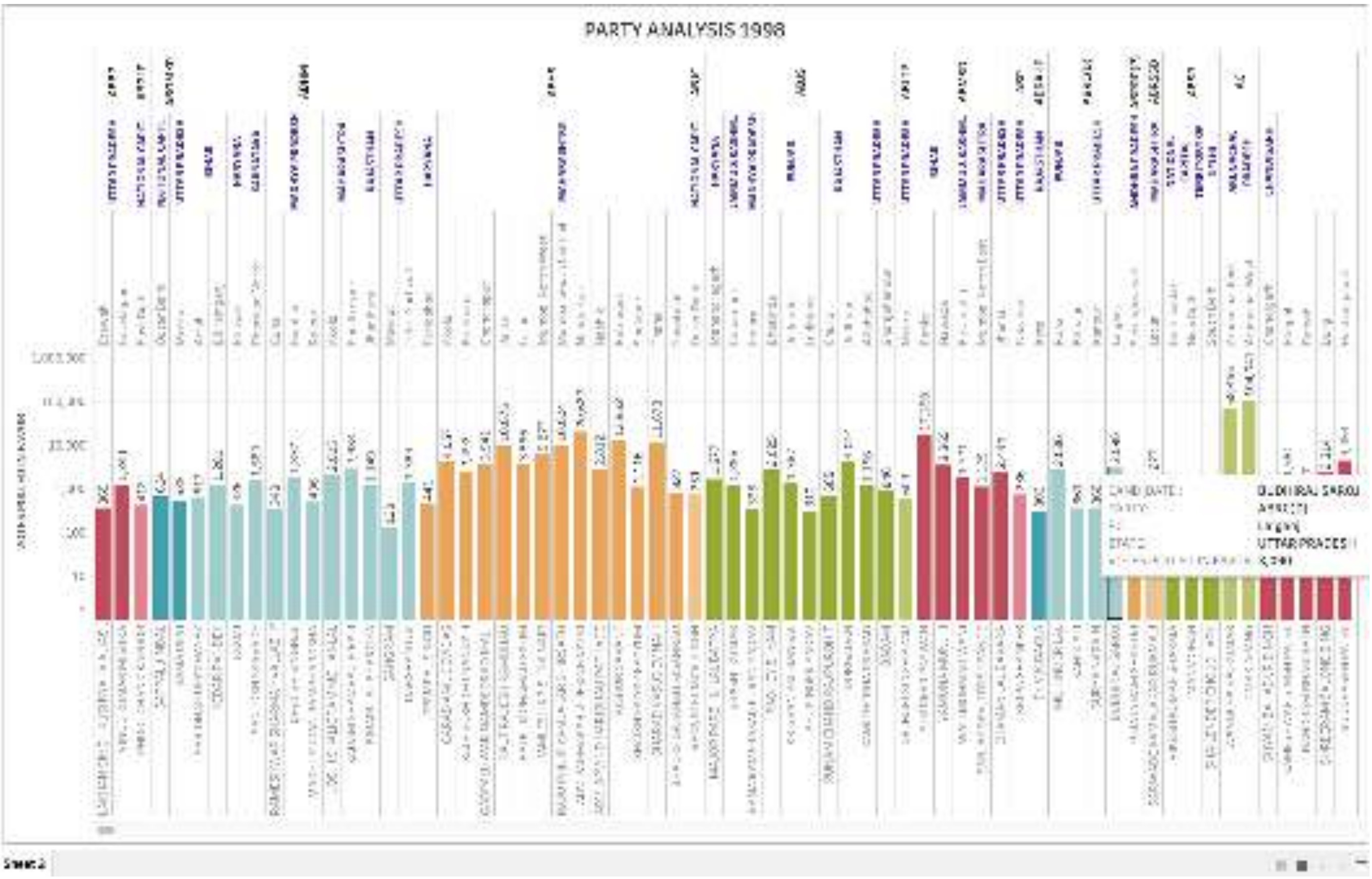
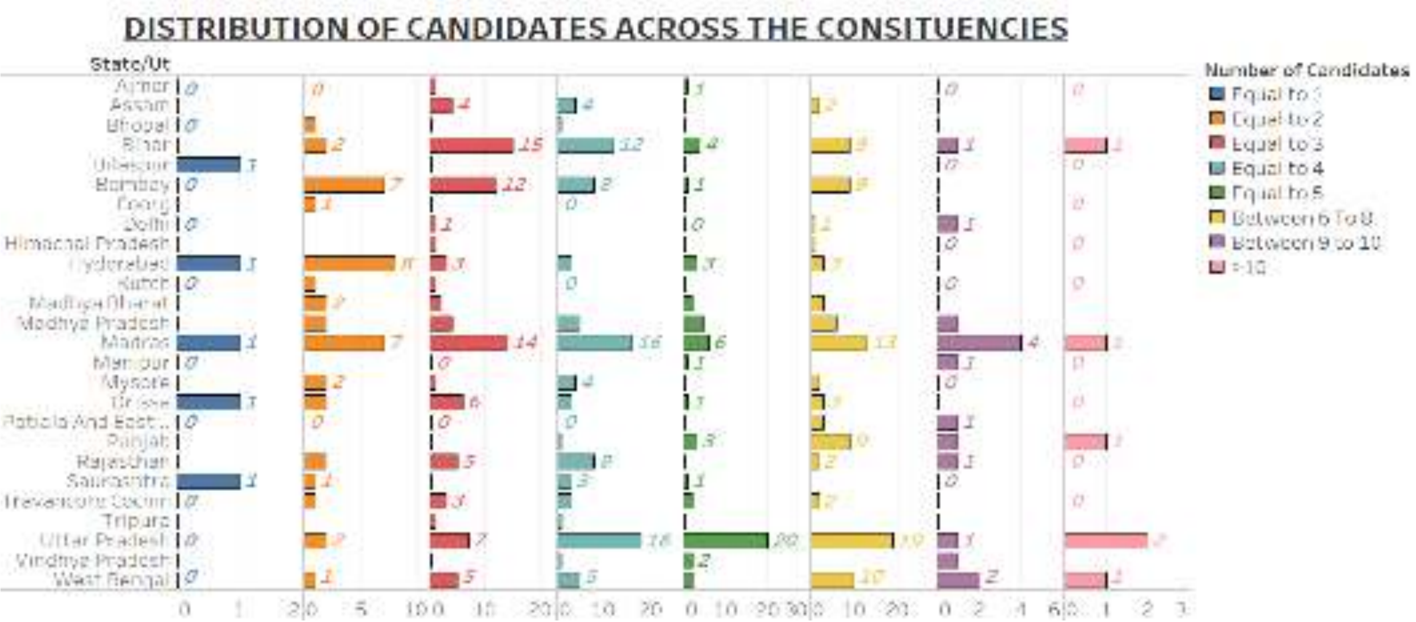


Figure 3.3: Result of detailed twitter extraction in Tableau

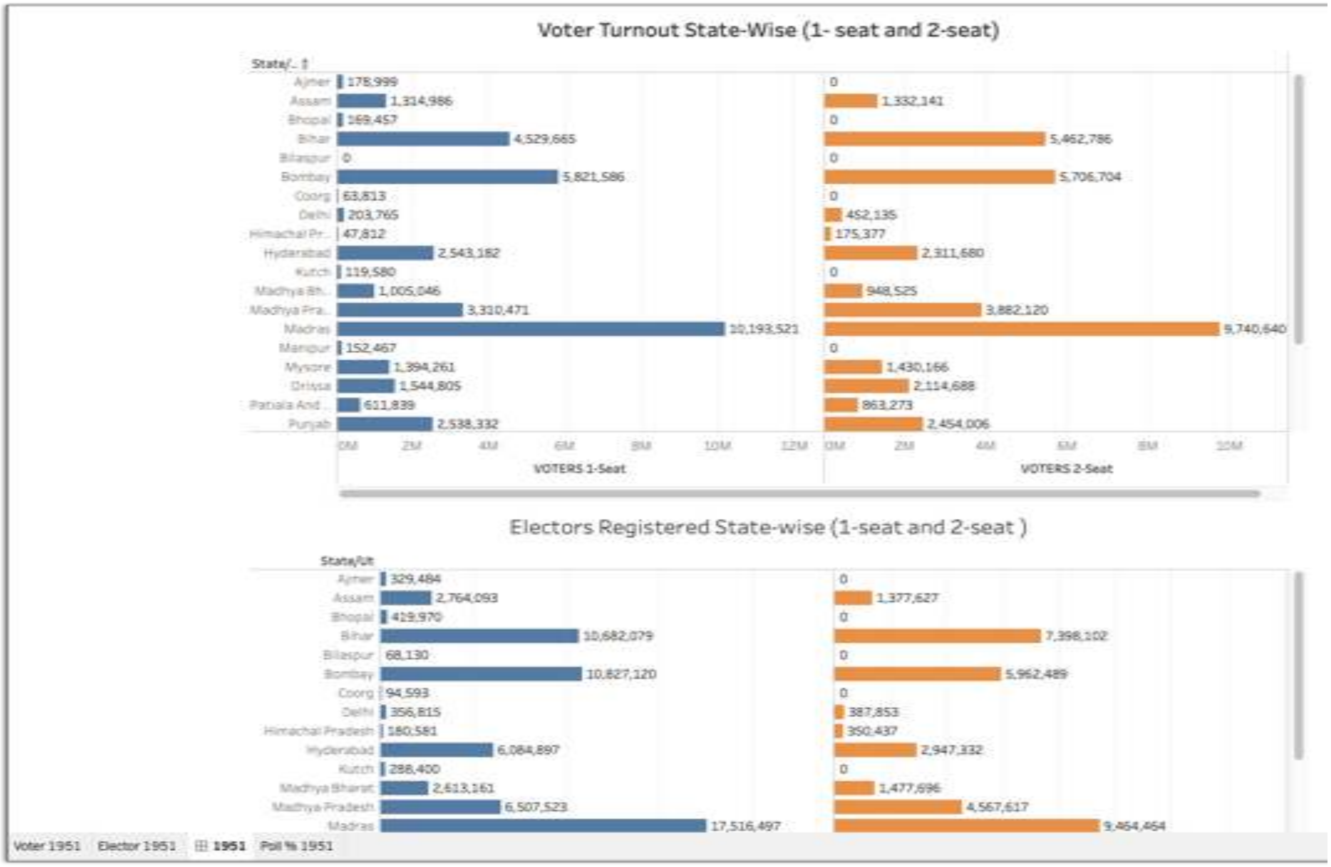
DATA VISUALIZATION



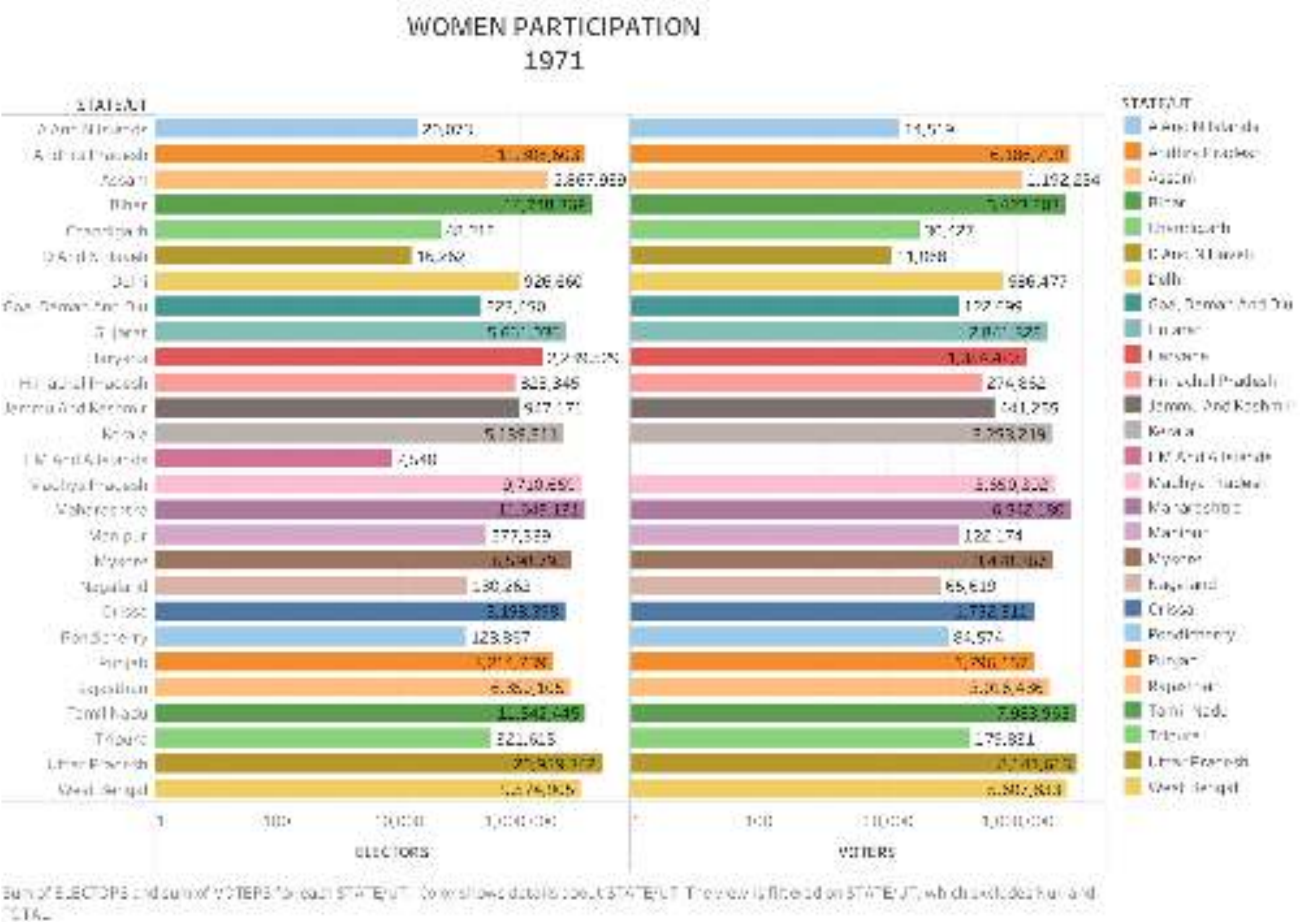
Graph 3.1: Detailed Graphical representation of PC-wise winning candidate along with the winning margin of votes



Graph 3.2: Detailed Graphical representation of State-wise candidates standing across seats in the year 1951

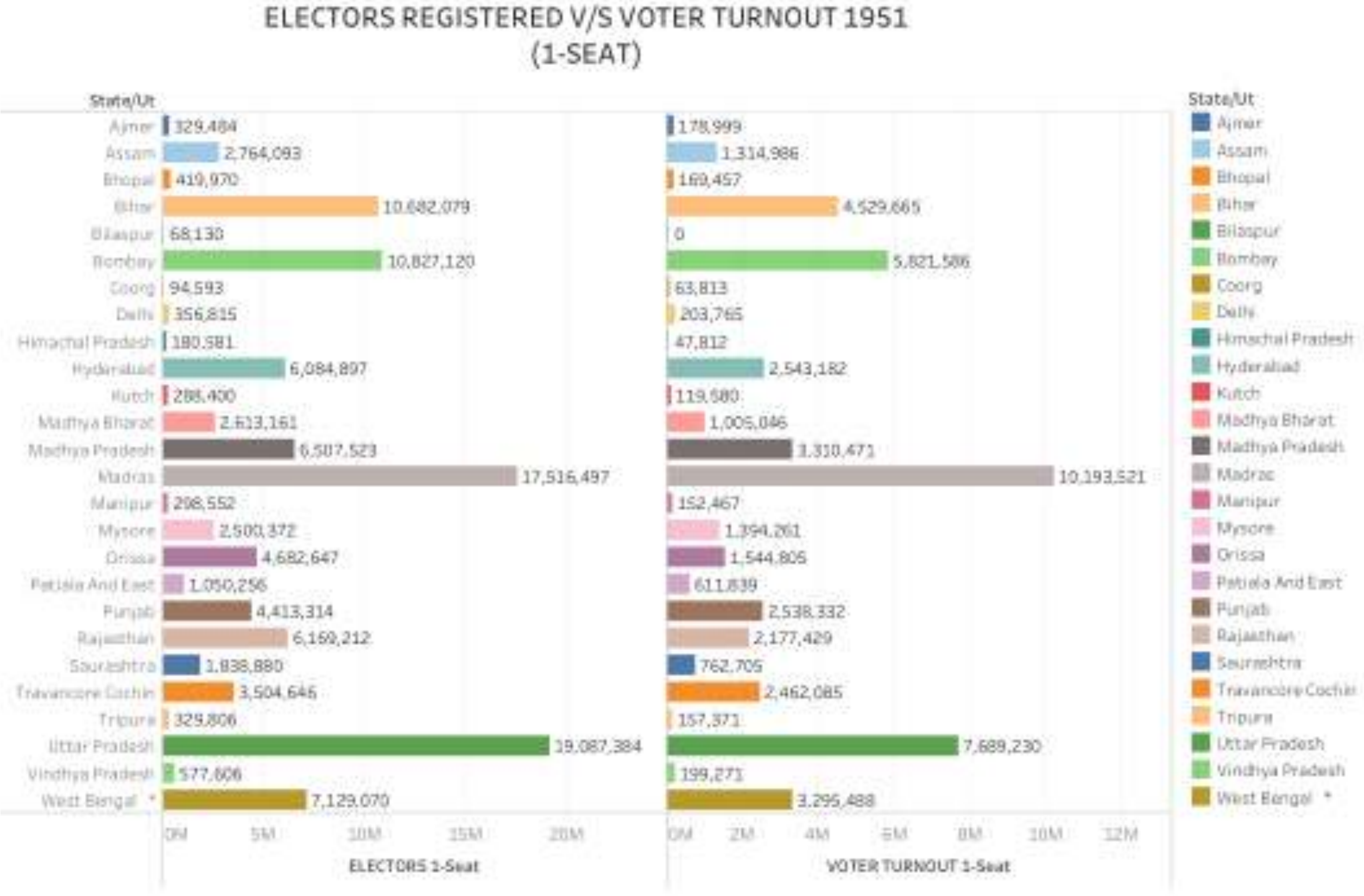


Graph 3.3: Detailed Graphical representation of State-wise electors registered across seats in the year 1951



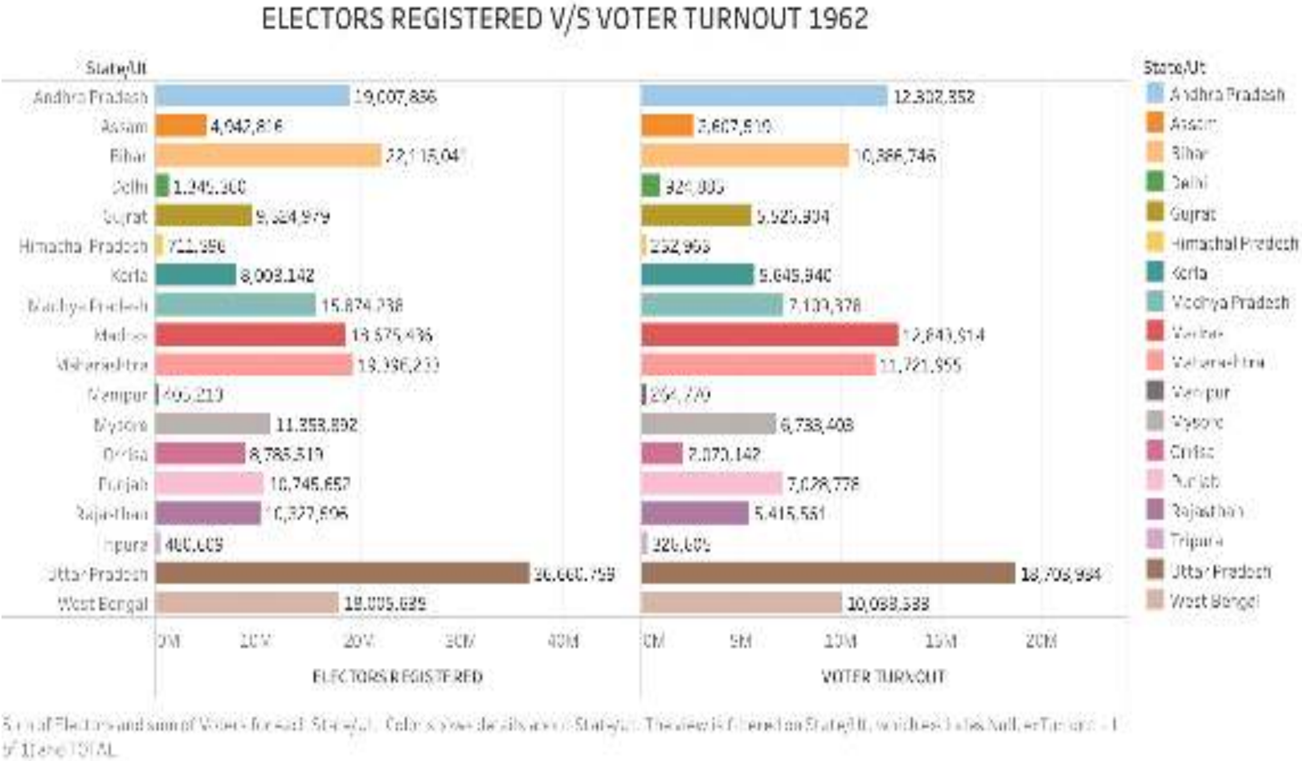
Graph 3.4: Detailed Graphical representation of State-wise women participation in electoral in the year 1971

ANALYSIS OF ELECTOR REGISTRATION AND VOTER TURNOUT WITH PYTHON / R

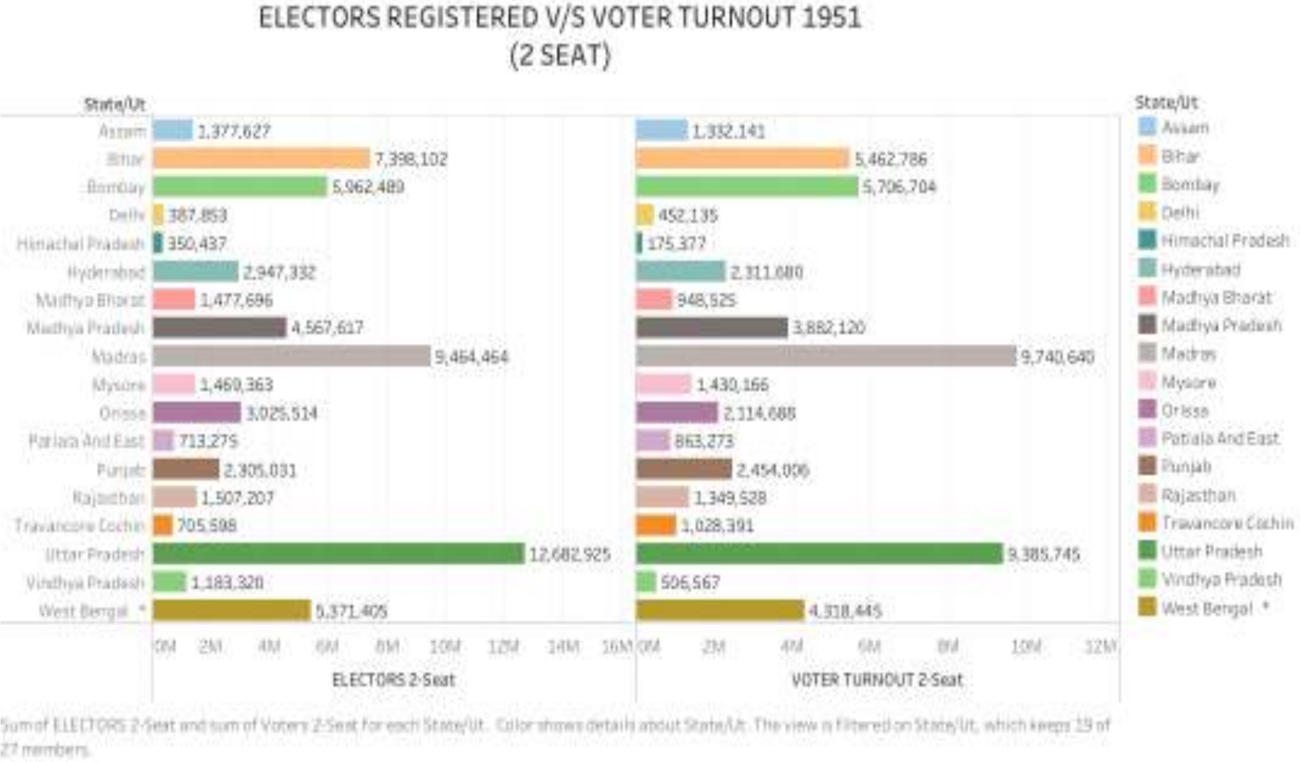


Sum of ELECTORS 1-Seat and sum of Voters 1-Seat for each State/Ut.. Color shows details about State/Ut.

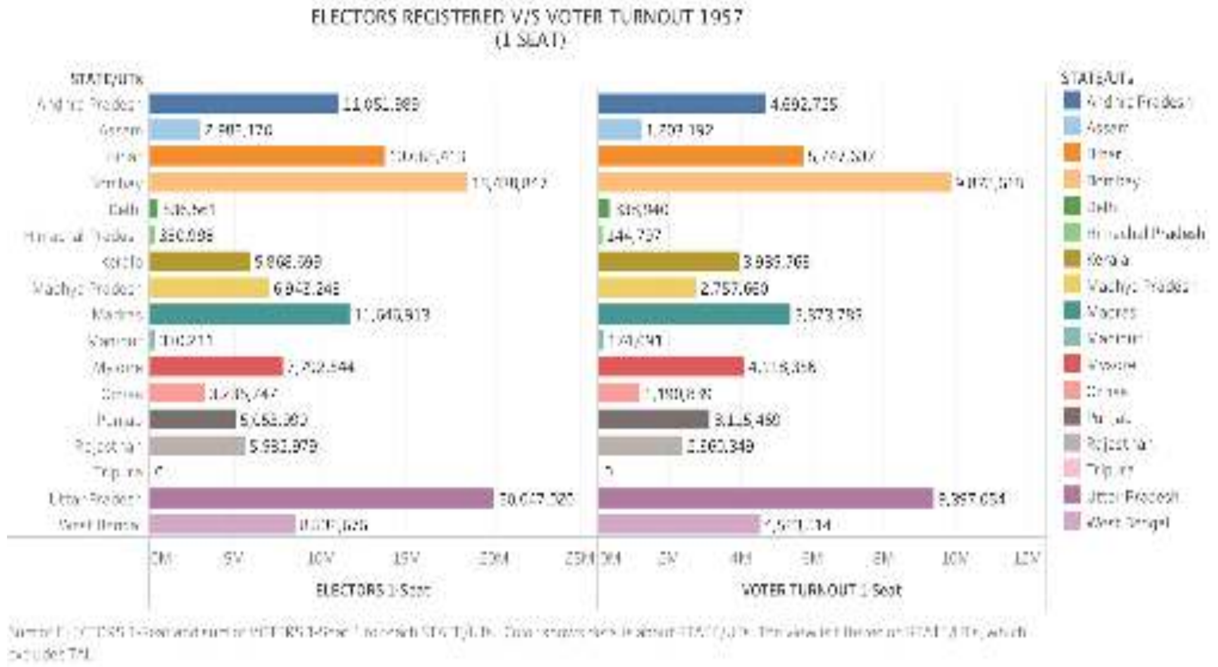
Graph 3.5: Detailed Graphical representation of State-wise elector registered and voter turnout (Seat 1) in the year 1951



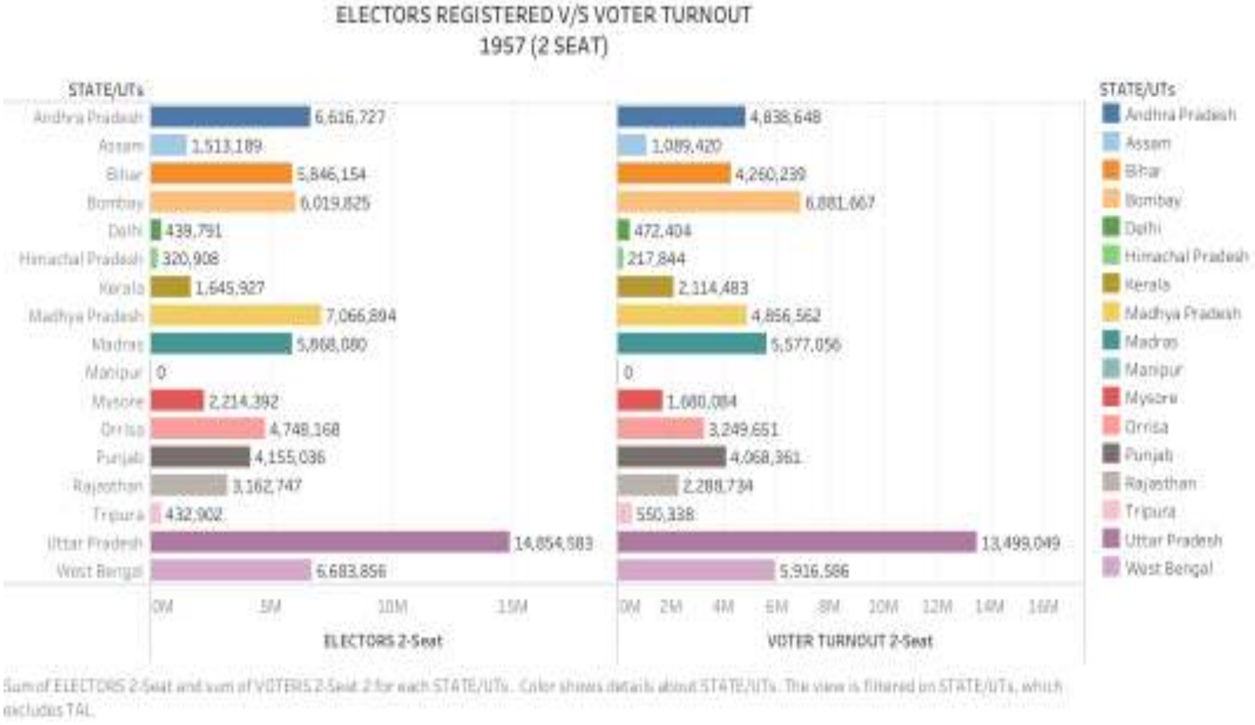
Graph 3.6: Detailed Graphical representation of State-wise elector registered and voter turnout in the year 1962



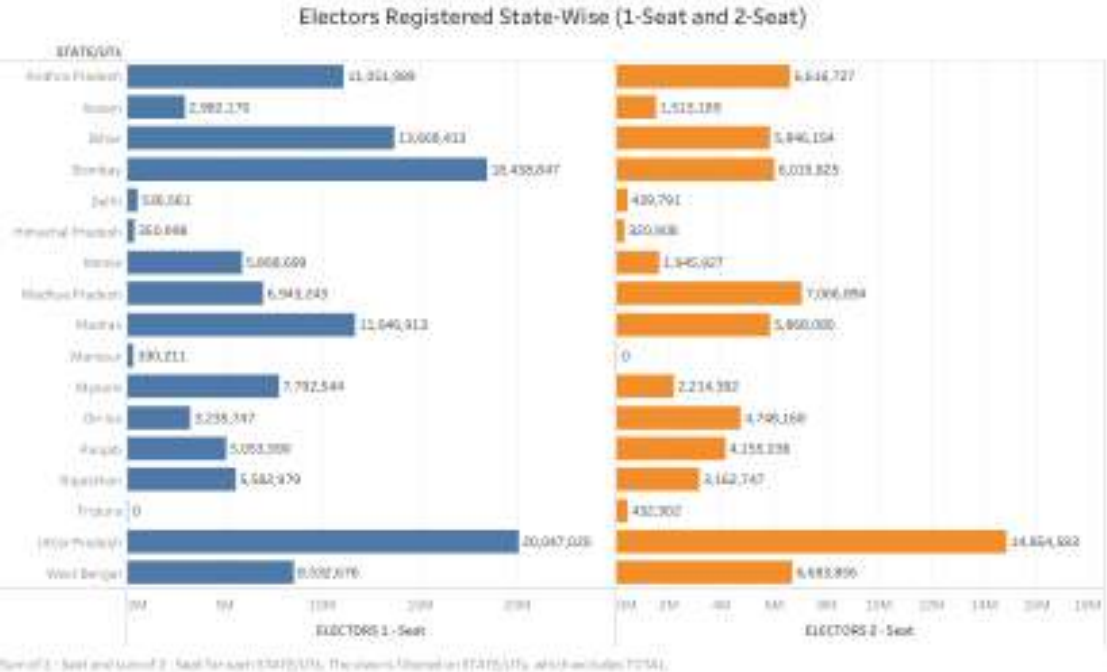
Graph 3.7: Detailed Graphical representation of State-wise elector registered and voter turnout (Seat 2) in the year 1951



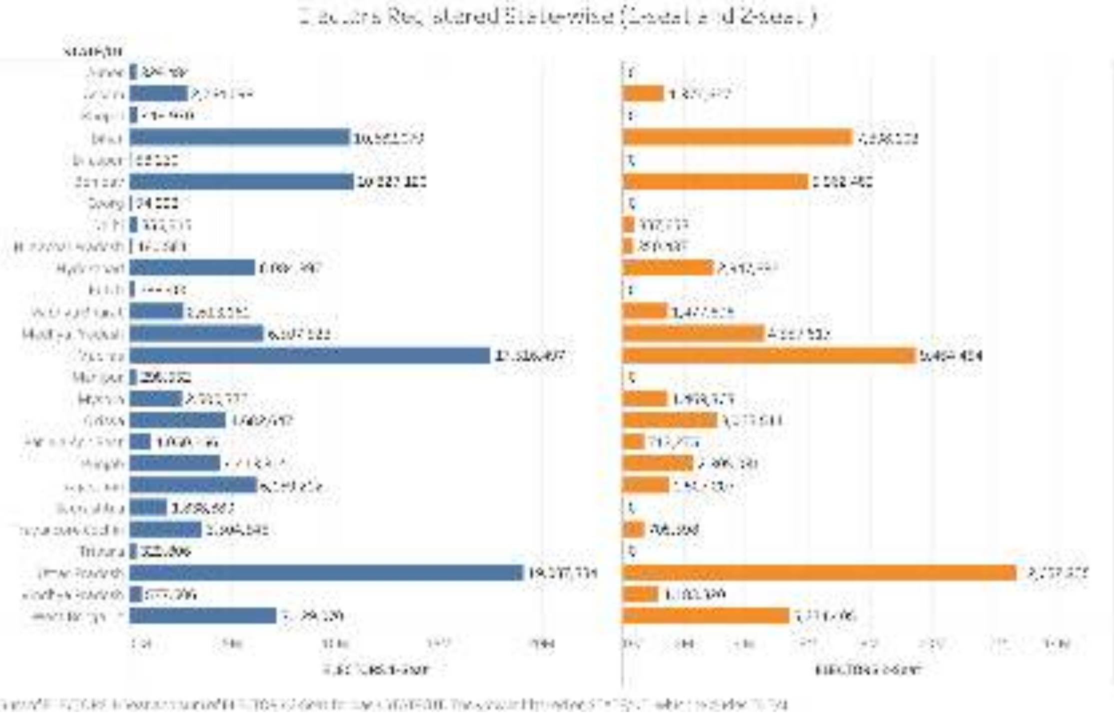
Graph 3.8: Detailed Graphical representation of State-wise elector registered and voter turnout (Seat 1) in the year 1957



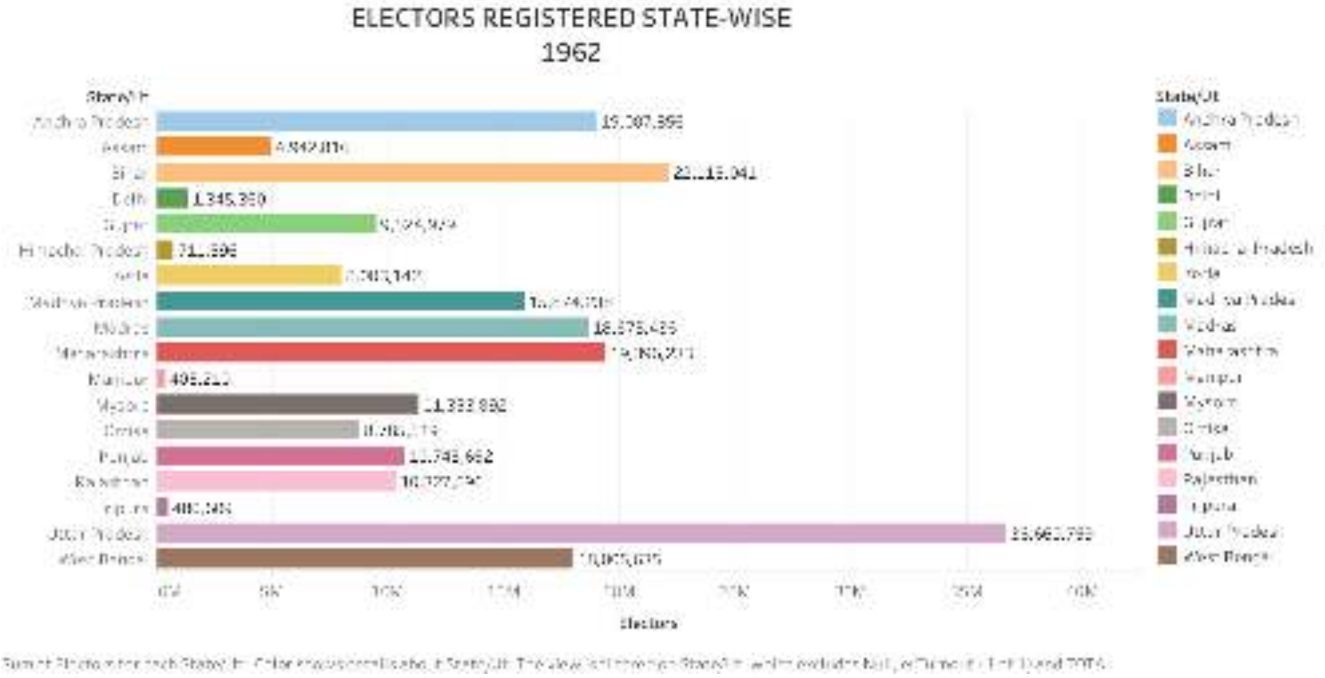
Graph 3.9: Detailed Graphical representation of State-wise elector registered and voter turnout (Seat 2) in the year 1957



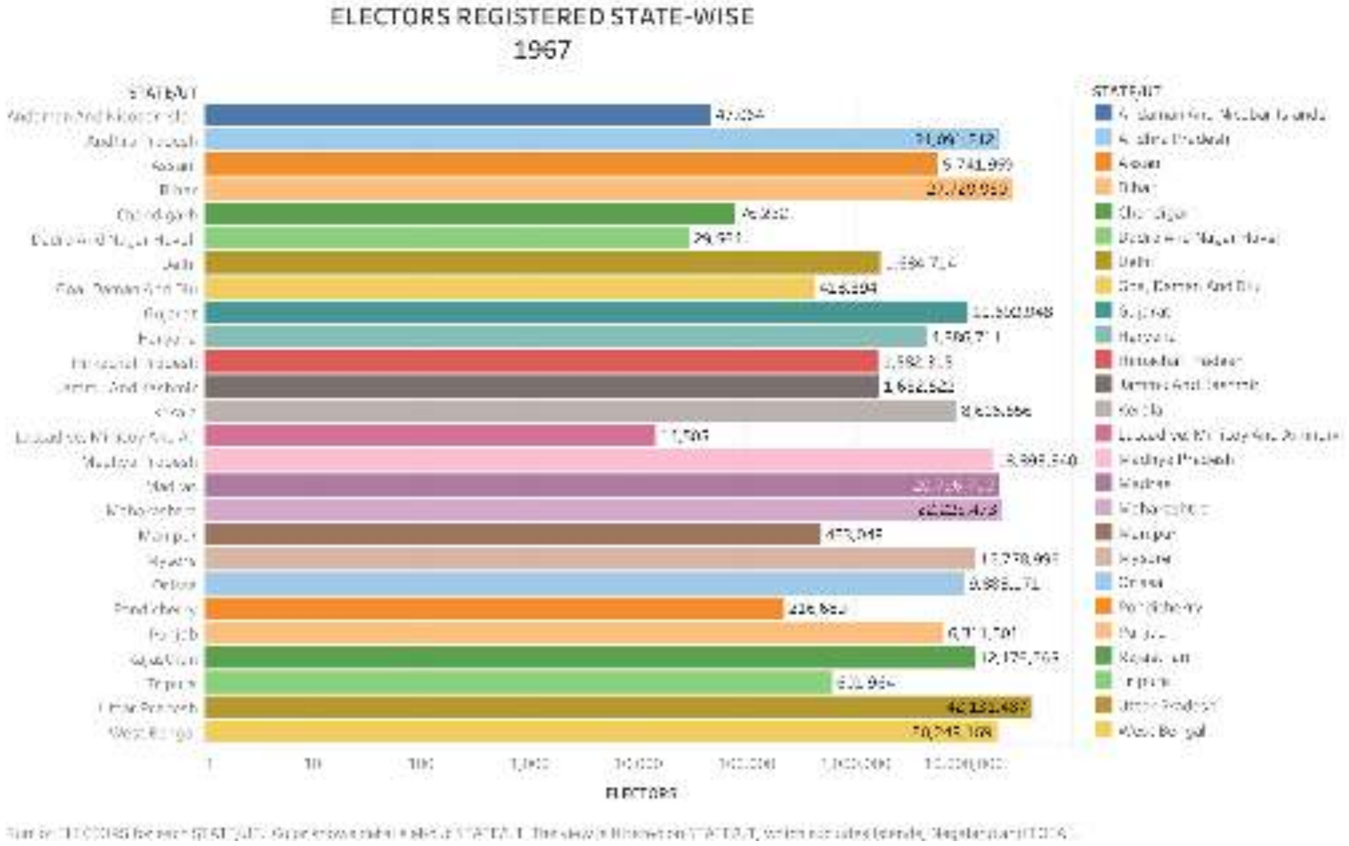
Graph 3.10: Graphical representation of State-wise elector registered on Seat 1 and Seat 2: 1951



Graph 3.11: Graphical representation of State-wise elector registered on Seat 1 and Seat 2: 1957

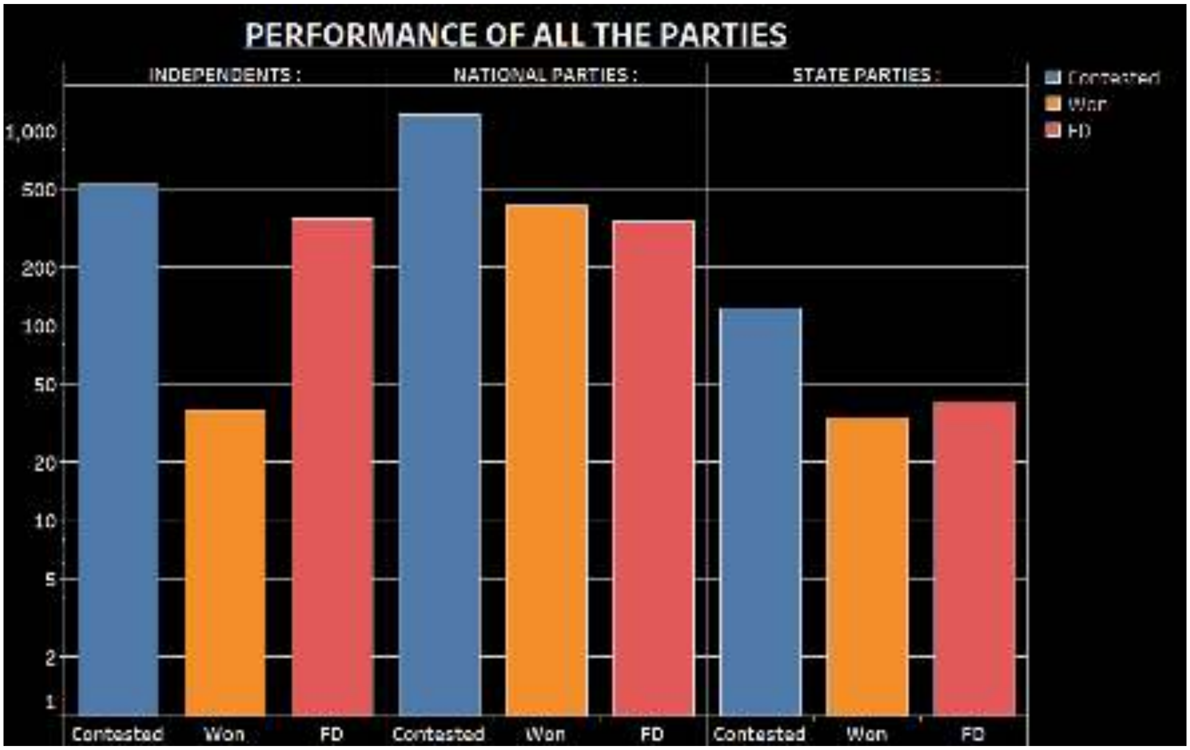


Graph 3.12: Graphical representation of State-wise elector registered 1962

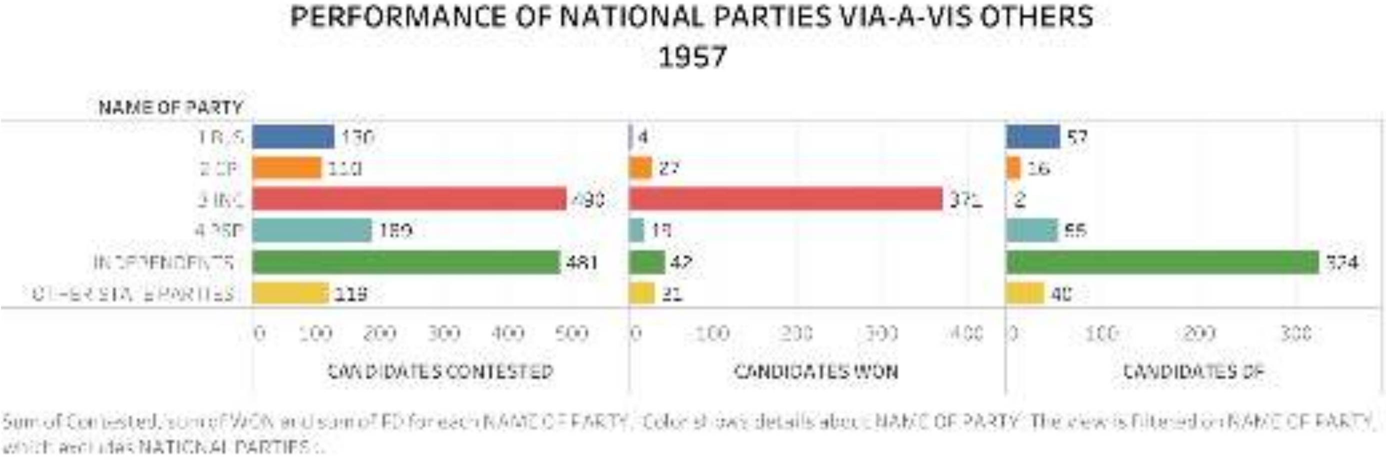


Graph 3.13: Graphical representation of State-wise elector registered 1967

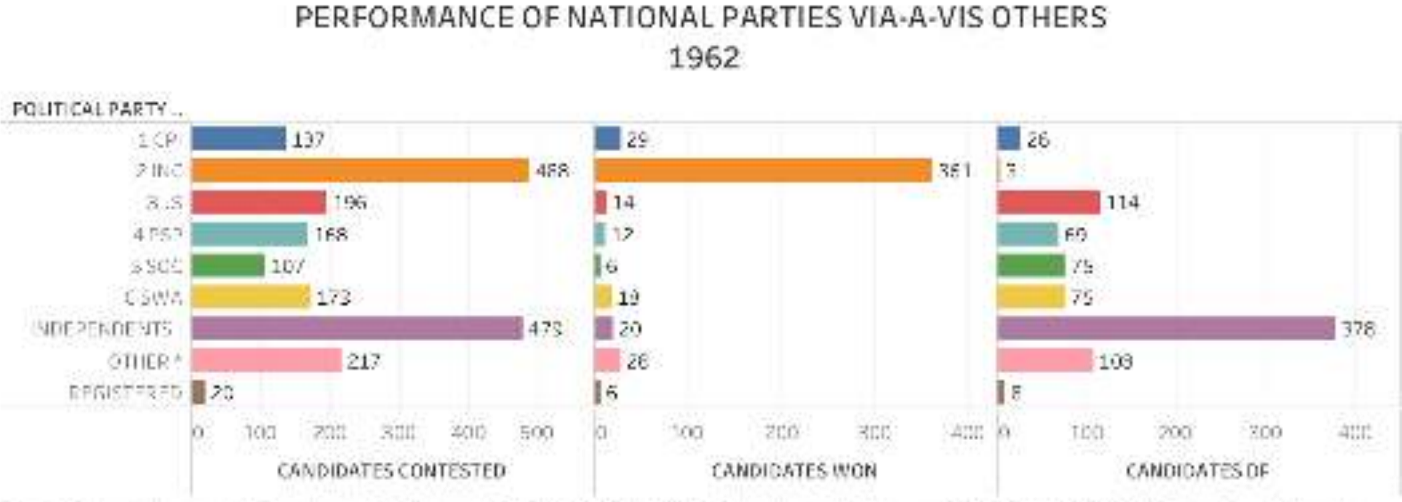
ANALYSIS OF POLITICAL PARTIES' PERFORMANCE OVER YEARS 1951-2019 USING TABLEAU



Graph 3.14: Graphical comparison of candidates 1951

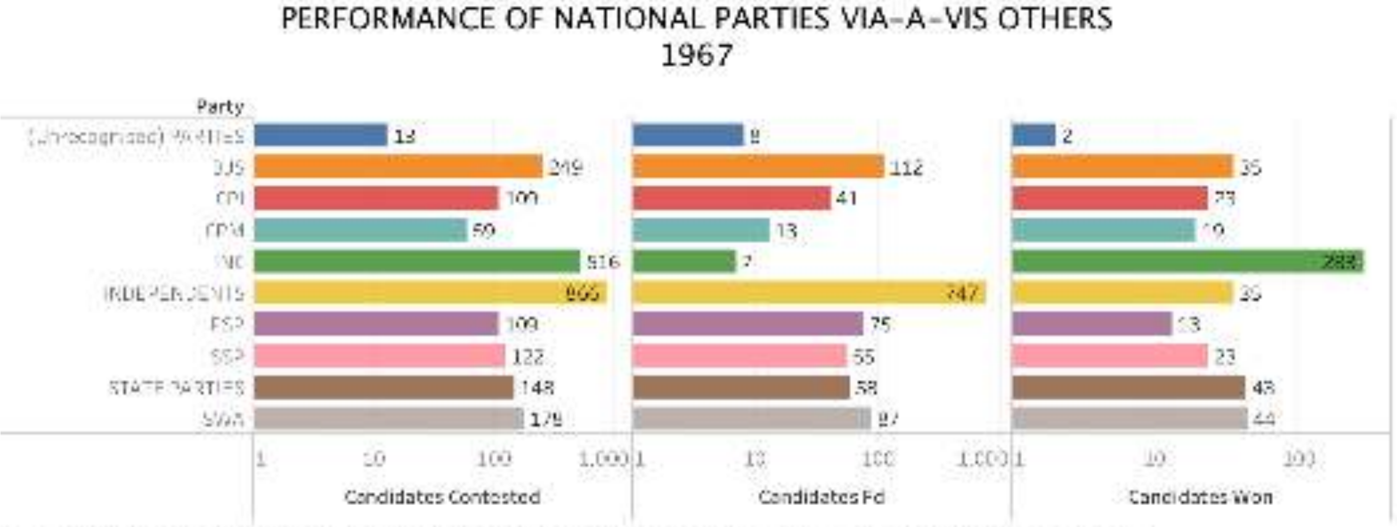


Graph 3.15: Graphical comparison of candidates 1957



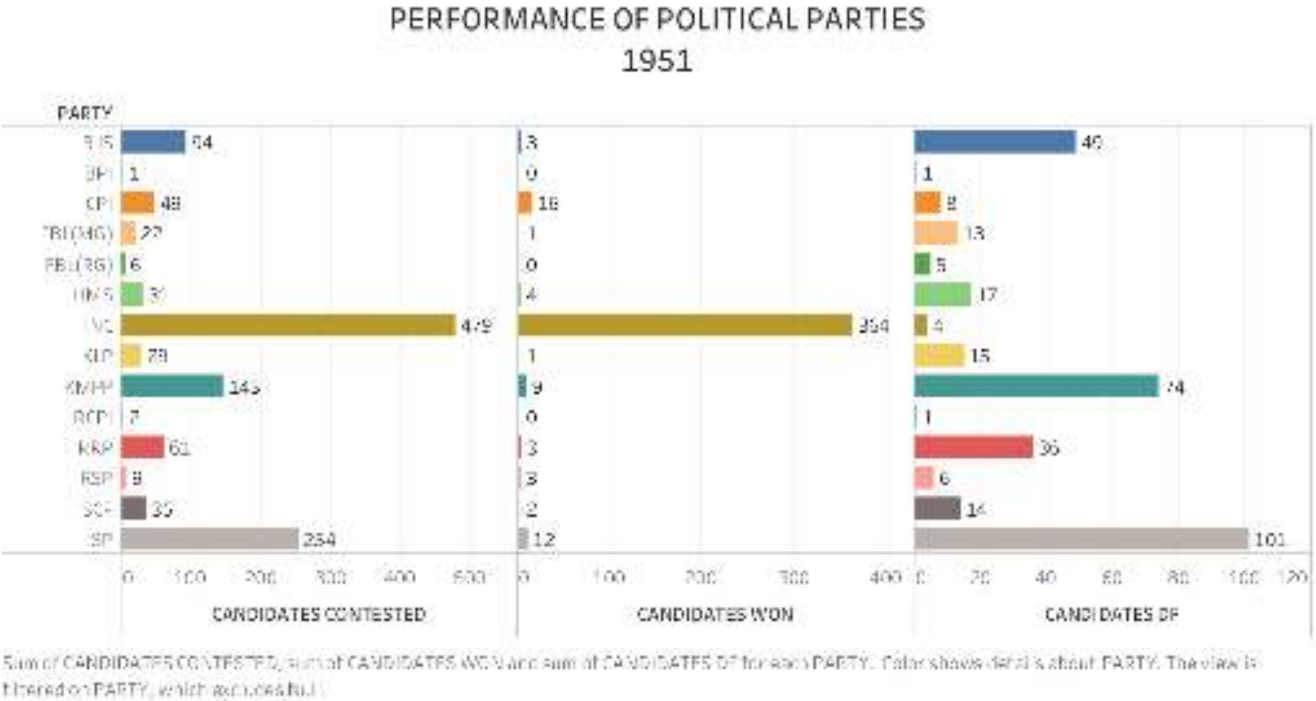
Sum of Contested, sum of WON and sum of FD for each POLITICAL PARTY NAME. Color shows details about POLITICAL PARTY NAME. The view is filtered on POLITICAL PARTY NAME, which excludes (Unrecognised) PARTIES and NATIONAL PARTIES.

Graph 3.16: Graphical comparison of candidates 1962

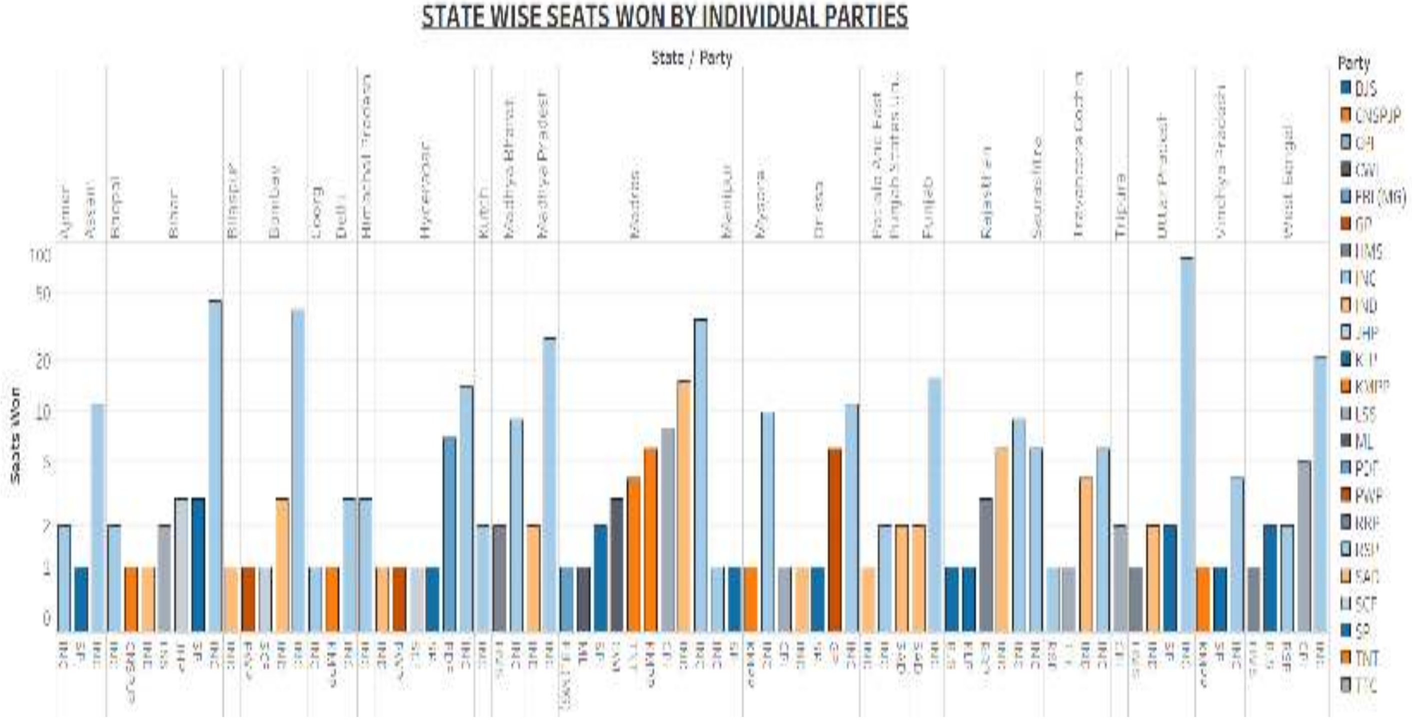


Sum of Candidates Contested, sum of Candidates Fd and sum of Candidates Won for each Party. Color shows details about Party.

Graph 3.17: Graphical comparison of candidates 1967

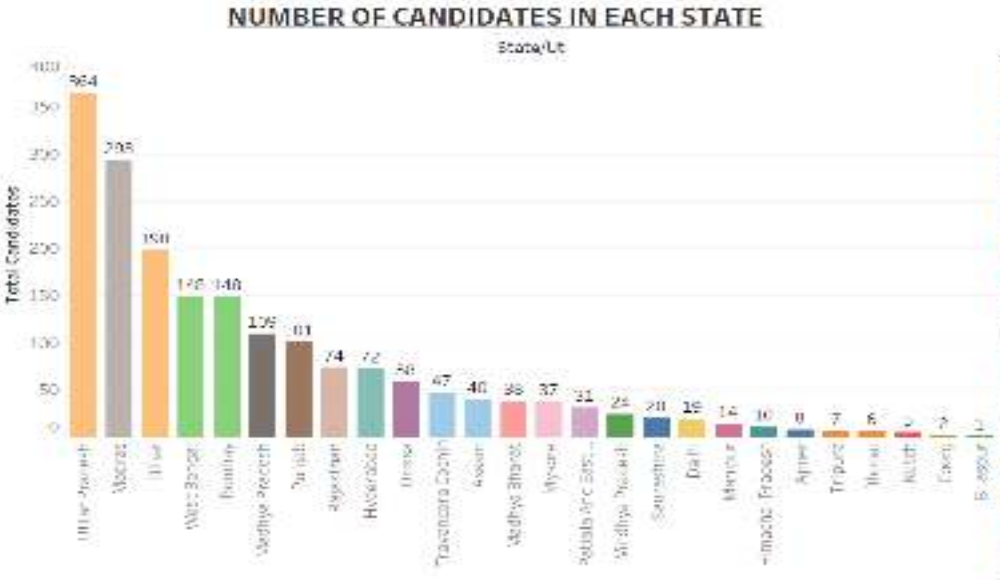


Graph 3.18: Detailed Graphical comparison of party-wise candidates 1951

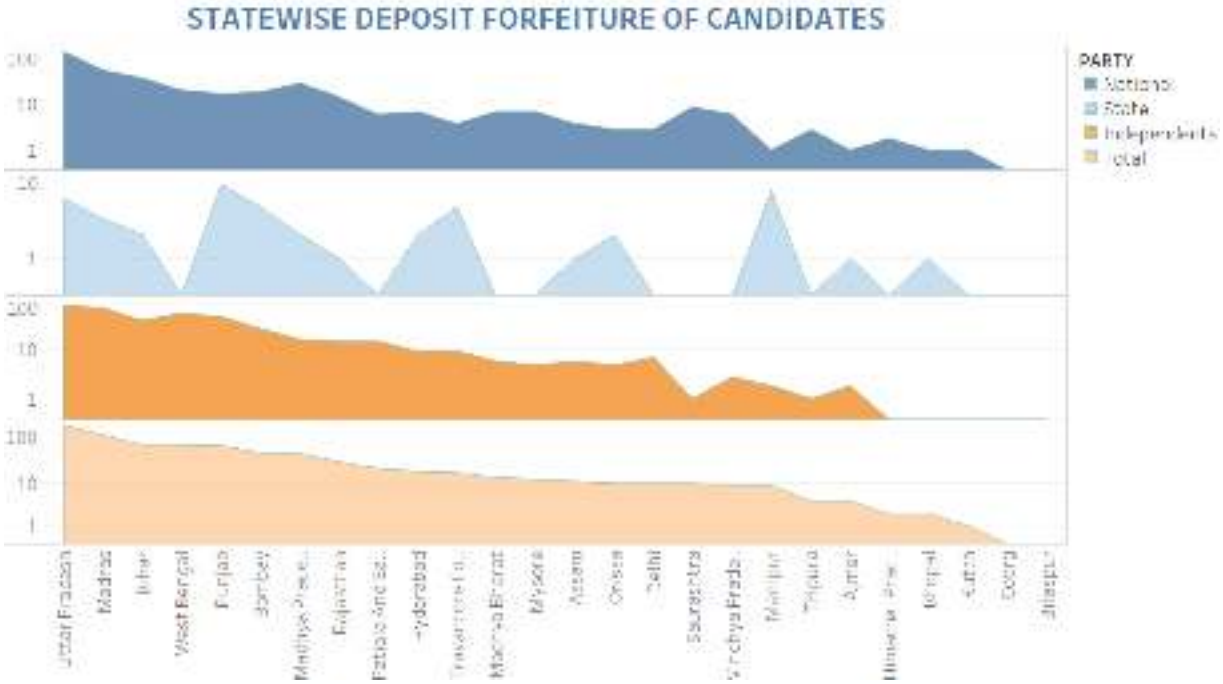


Graph 3.19: Detailed Graphical comparison of party-wise candidates 1962

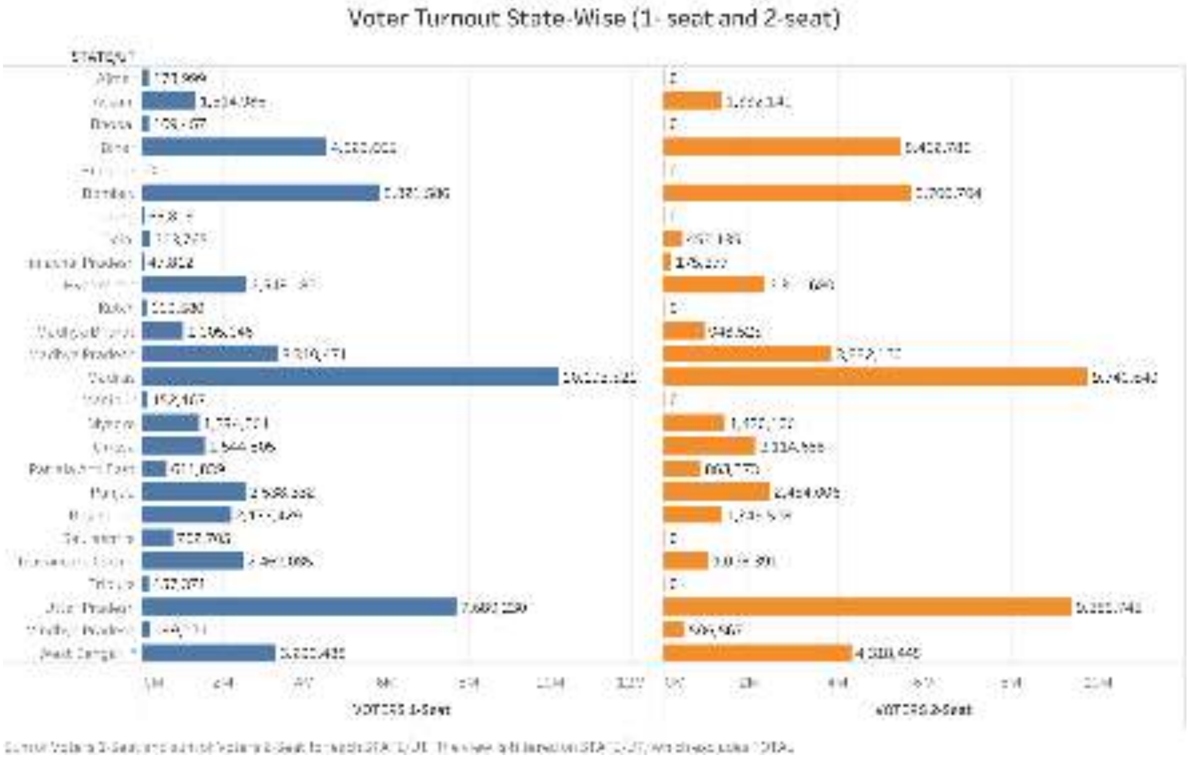
ANALYSIS OF VOTER TURNOUT OVER YEARS 1951-2019 USING TABLEAU



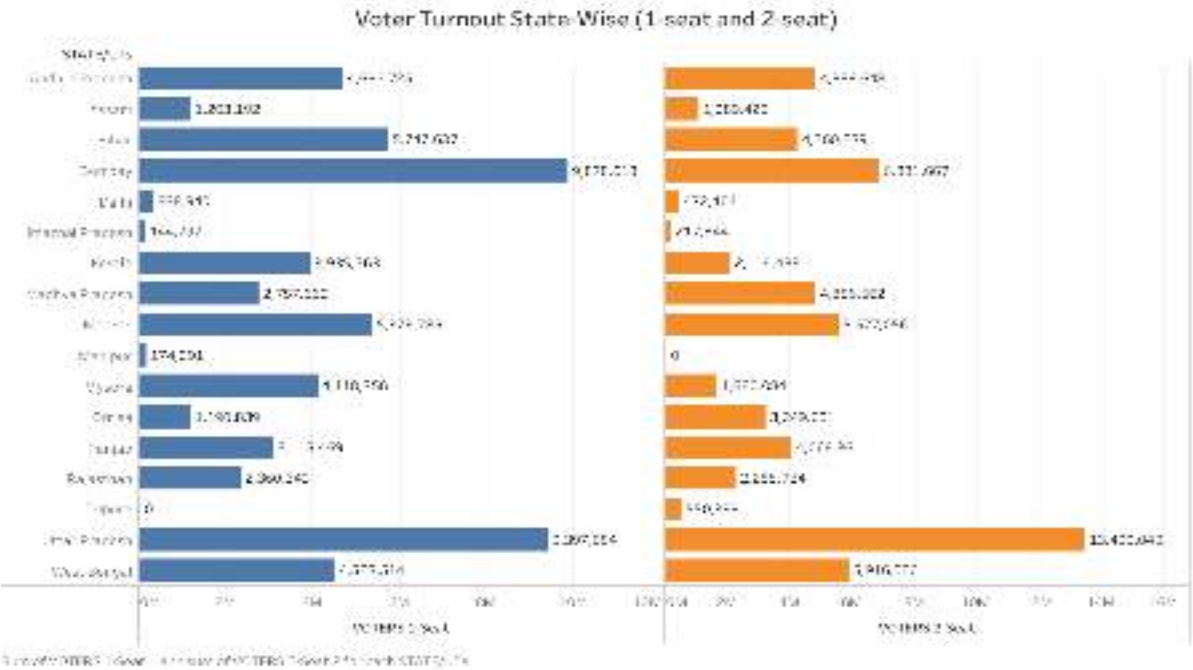
Graph 3.20: Detailed Graphical comparison of state-wise candidates 1951



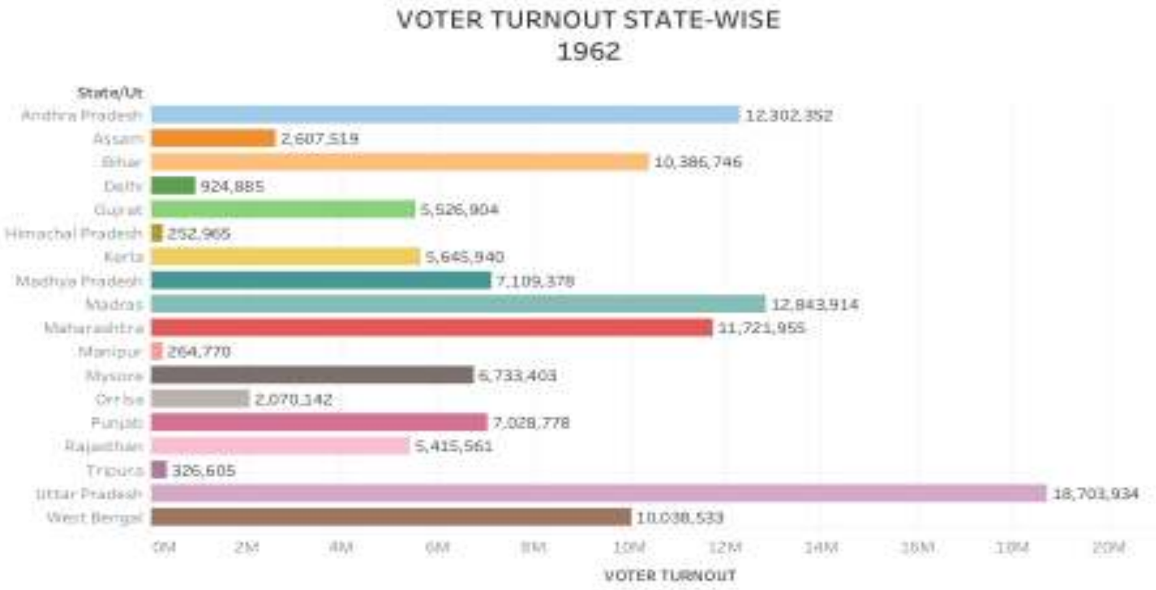
Graph 3.21: Detailed Graphical comparison of state-wise candidates 1962



Graph 3.22: Detailed Graphical comparison of voter turnout state-wise 1951

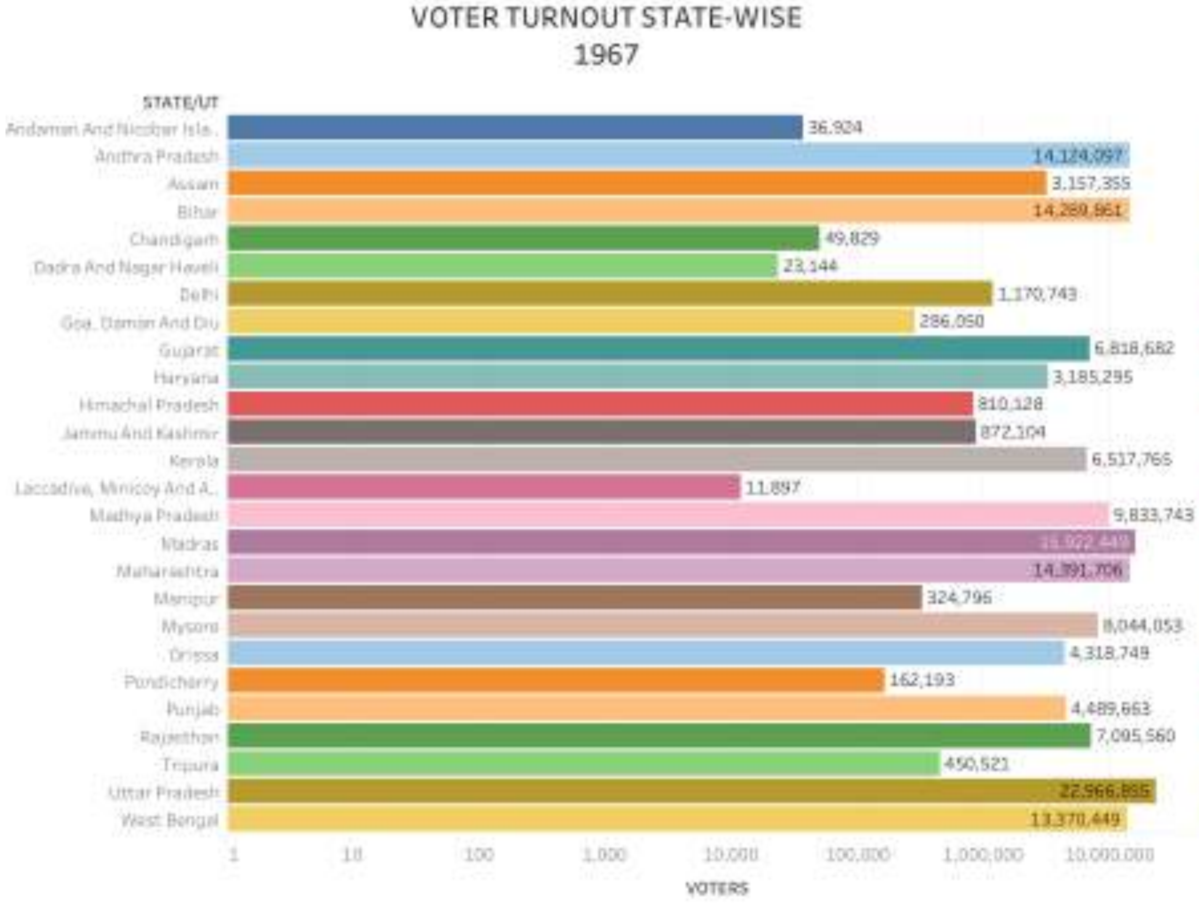


Graph 3.23: Detailed Graphical comparison of voter turnout state-wise 1957



Sum of Voters for each State/Ut. Color shows details about State/Ut. The view is filtered on State/Ut, which excludes Null, erTurnout - 1 of 1) and TOTAL.

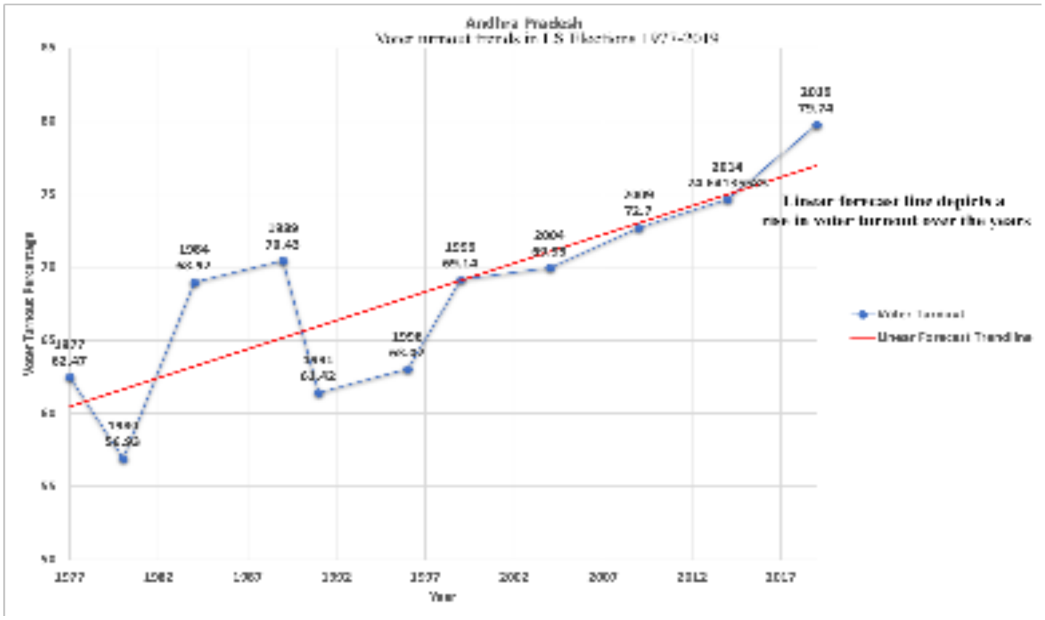
Graph 3.24: Detailed Graphical representation of voter turnout state-wise 1962



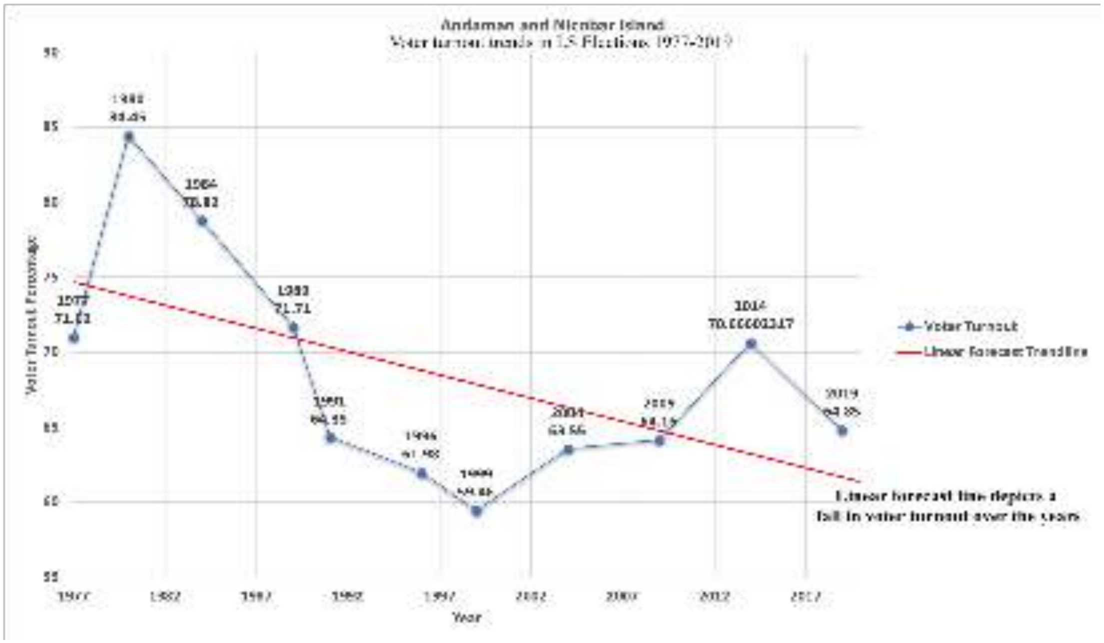
Sum of VOTERS for each STATE/UT. Color shows details about STATE/UT. The view is filtered on STATE/UT, which excludes Islands, Nagaland and TOTAL.

Graph 3.25: Detailed Graphical representation of voter turnout state-wise 1967

VOTER TURNOUT FORECAST PREDICTION V/S ACTUAL FIGURES



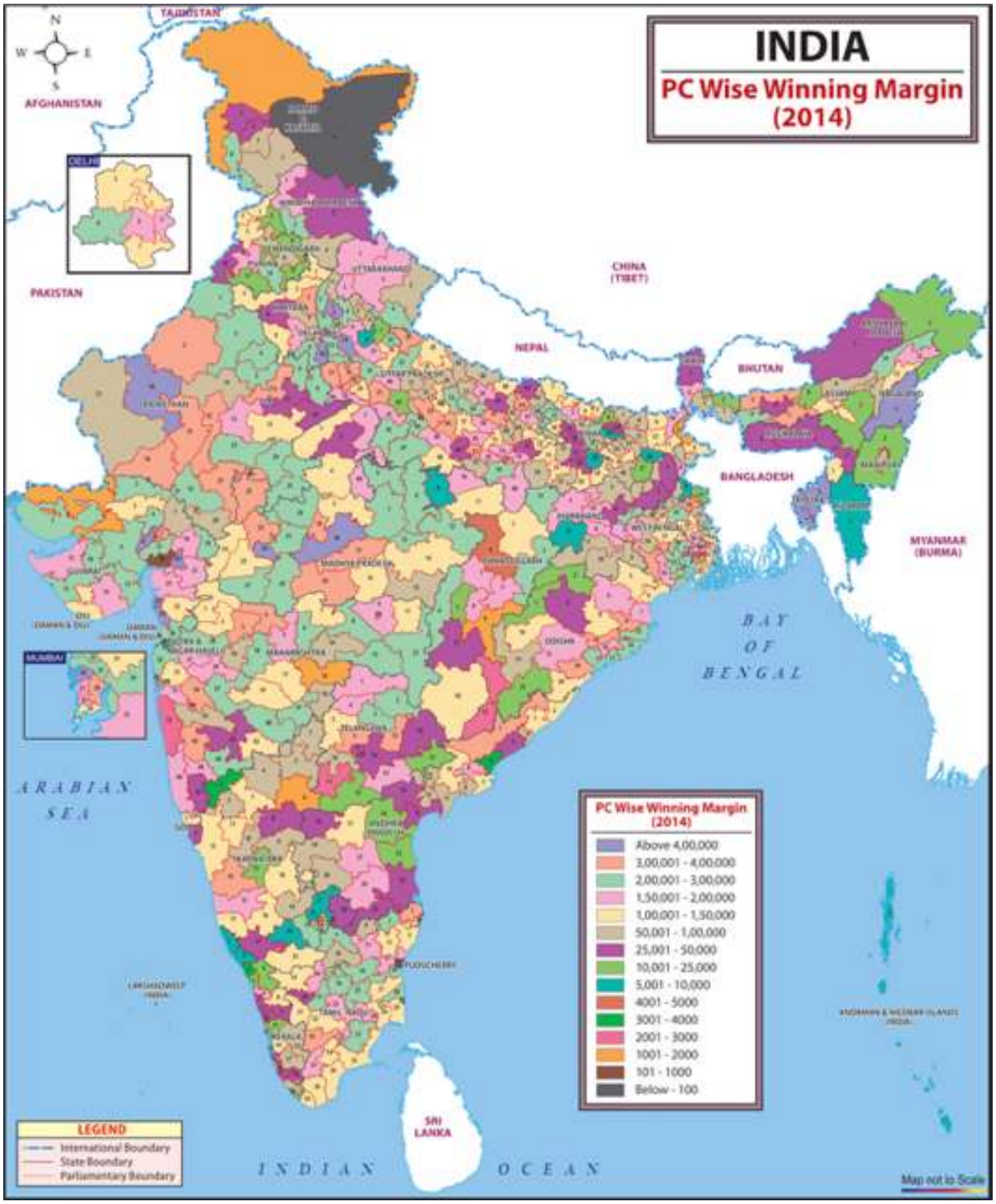
Graph 3.26: Detailed Graphical comparison of forecast line and actual figure of voter turn-out in 2019 in Andhra Pradesh (Positive trend justified)



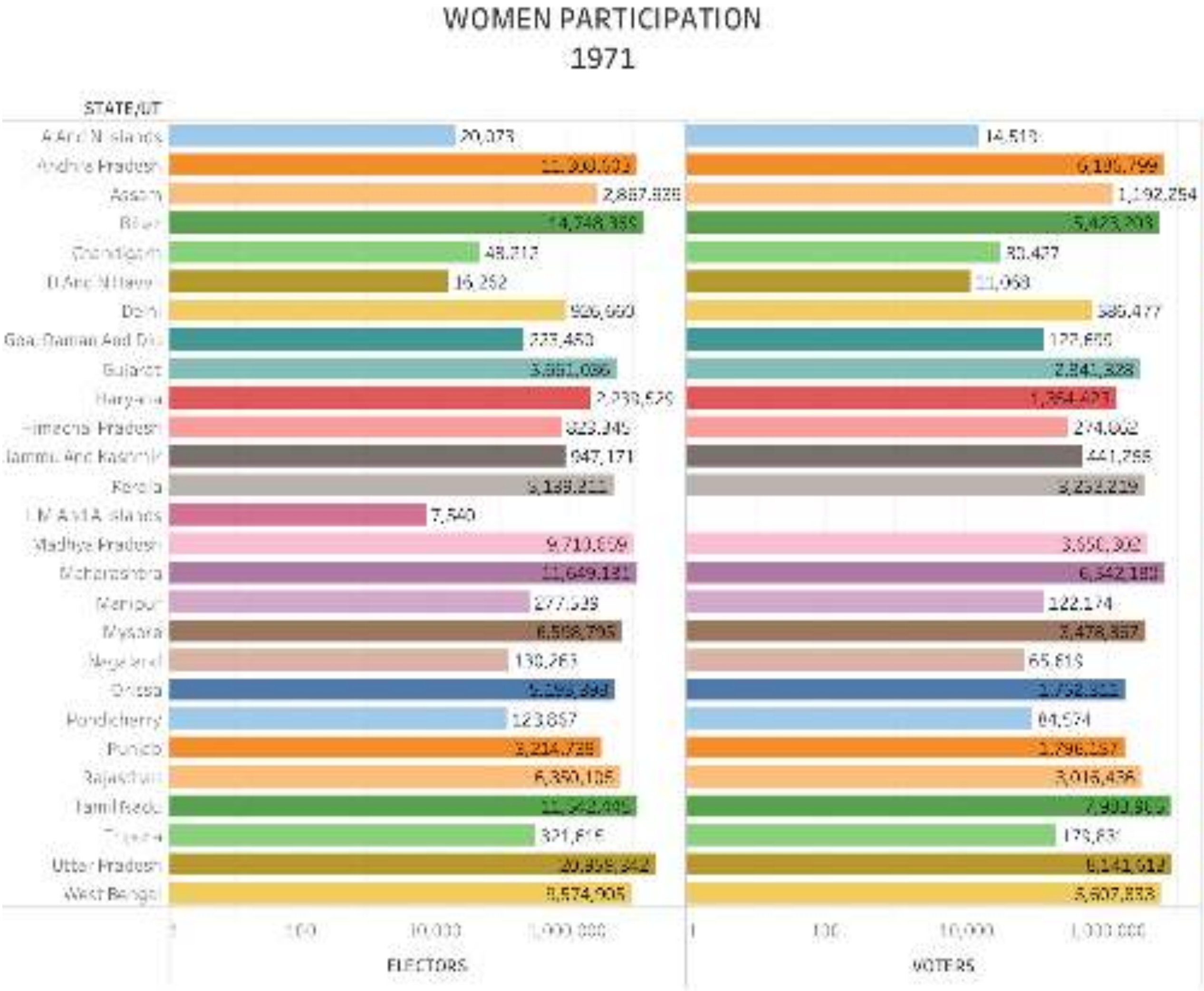
Graph 3.27: Detailed Graphical comparison of forecast line and actual figure of voter turn-out in 2019 in Andaman and Nicobar Island (Negative trend justified)

MAP GENERATED USING TABLEAU FOR WINNING MARGIN ANALYSIS ACROSS THE COUNTRY

YEAR: 2014

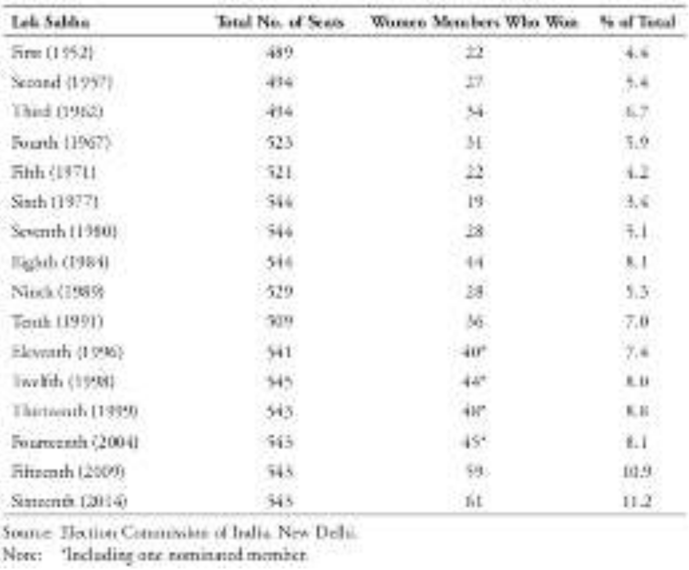


ANALYSIS OF WOMEN EMPOWERMENT OVER YEARS 1951-2019



Sum of ELECTORS and sum of VOTERS for each STATE/UT. Color shows details about STATE/UT. The view is filtered on STATE/UT, which excludes Null and TOTAL.

Graph 3.29: Detailed Graphical comparison of women in electoral roll v/s turnout of women for vote casting in the year 1971(The first-year women turnout was registered)

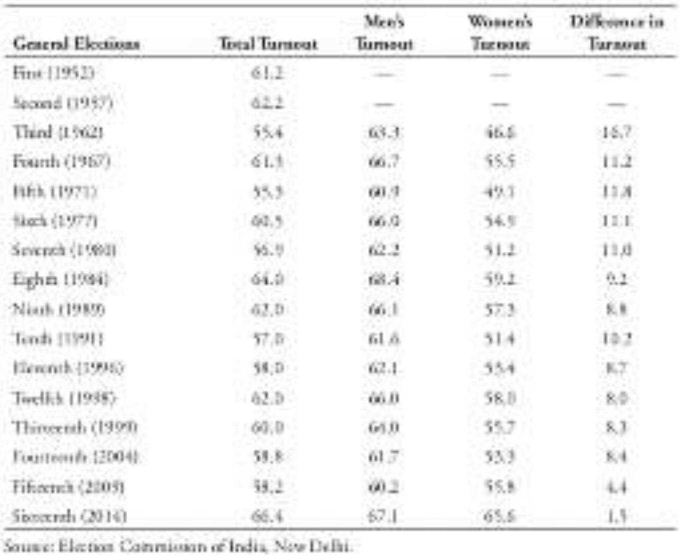


Lok Sabha	Total No. of Seats	Women Members Who Won	% of Total
First (1952)	489	22	4.4
Second (1957)	494	27	5.4
Third (1962)	494	34	6.7
Fourth (1967)	523	31	5.9
Fifth (1971)	521	22	4.2
Sixth (1977)	544	19	3.4
Seventh (1980)	544	28	5.1
Eighth (1984)	544	44	8.1
Ninth (1989)	529	28	5.3
Tenth (1991)	509	36	7.0
Eleventh (1996)	541	40*	7.4
Twelfth (1998)	545	44*	8.0
Thirteenth (1999)	543	48*	8.8
Fourteenth (2004)	543	45*	8.1
Fifteenth (2009)	543	59	10.9
Sixteenth (2014)	543	61	11.2

Source: Election Commission of India, New Delhi.

Note: *Including one nominated member.

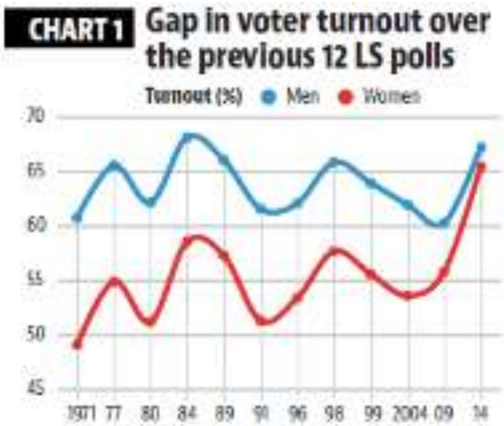
Table 3.1: Representation of women in the Lok Sabha Elections since 1952



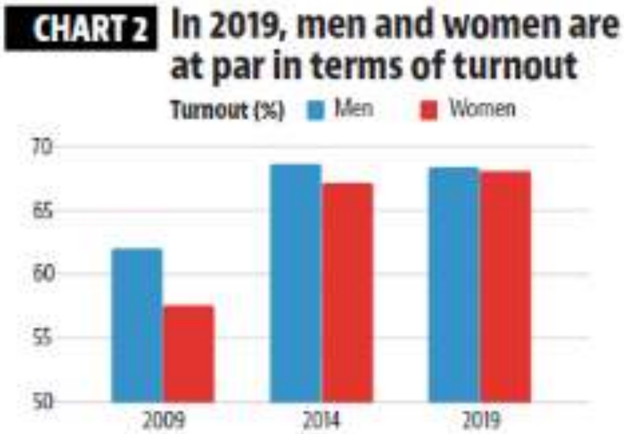
General Elections	Total Turnout	Men's Turnout	Women's Turnout	Difference in Turnout
First (1952)	61.2	—	—	—
Second (1957)	62.2	—	—	—
Third (1962)	55.4	63.3	46.6	16.7
Fourth (1967)	61.3	66.7	55.5	11.2
Fifth (1971)	55.3	60.9	49.1	11.8
Sixth (1977)	60.5	66.0	54.9	11.1
Seventh (1980)	56.9	62.2	51.2	11.0
Eighth (1984)	64.0	68.4	59.2	9.2
Ninth (1989)	62.0	66.1	57.3	8.8
Tenth (1991)	57.0	61.6	51.4	10.2
Eleventh (1996)	58.0	62.1	53.4	8.7
Twelfth (1998)	62.0	66.0	58.0	8.0
Thirteenth (1999)	60.0	64.0	55.7	8.3
Fourteenth (2004)	58.8	61.7	53.3	8.4
Fifteenth (2009)	58.2	60.2	55.8	4.4
Sixteenth (2014)	66.4	67.1	65.6	1.5

Source: Election Commission of India, New Delhi.

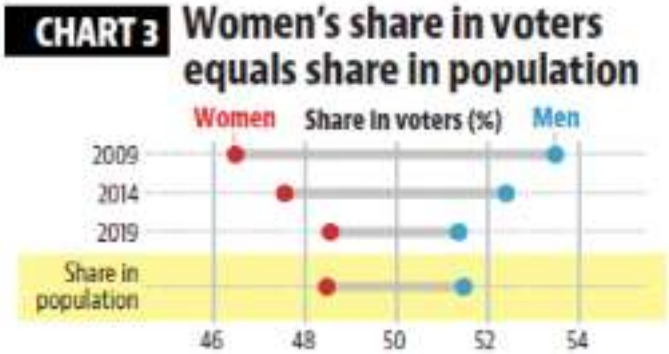
Table 3.2: Turnout of Women for voting in the Lok Sabha Elections across years



Graph 3.30: Graphical comparison of men and women for vote casting across the years

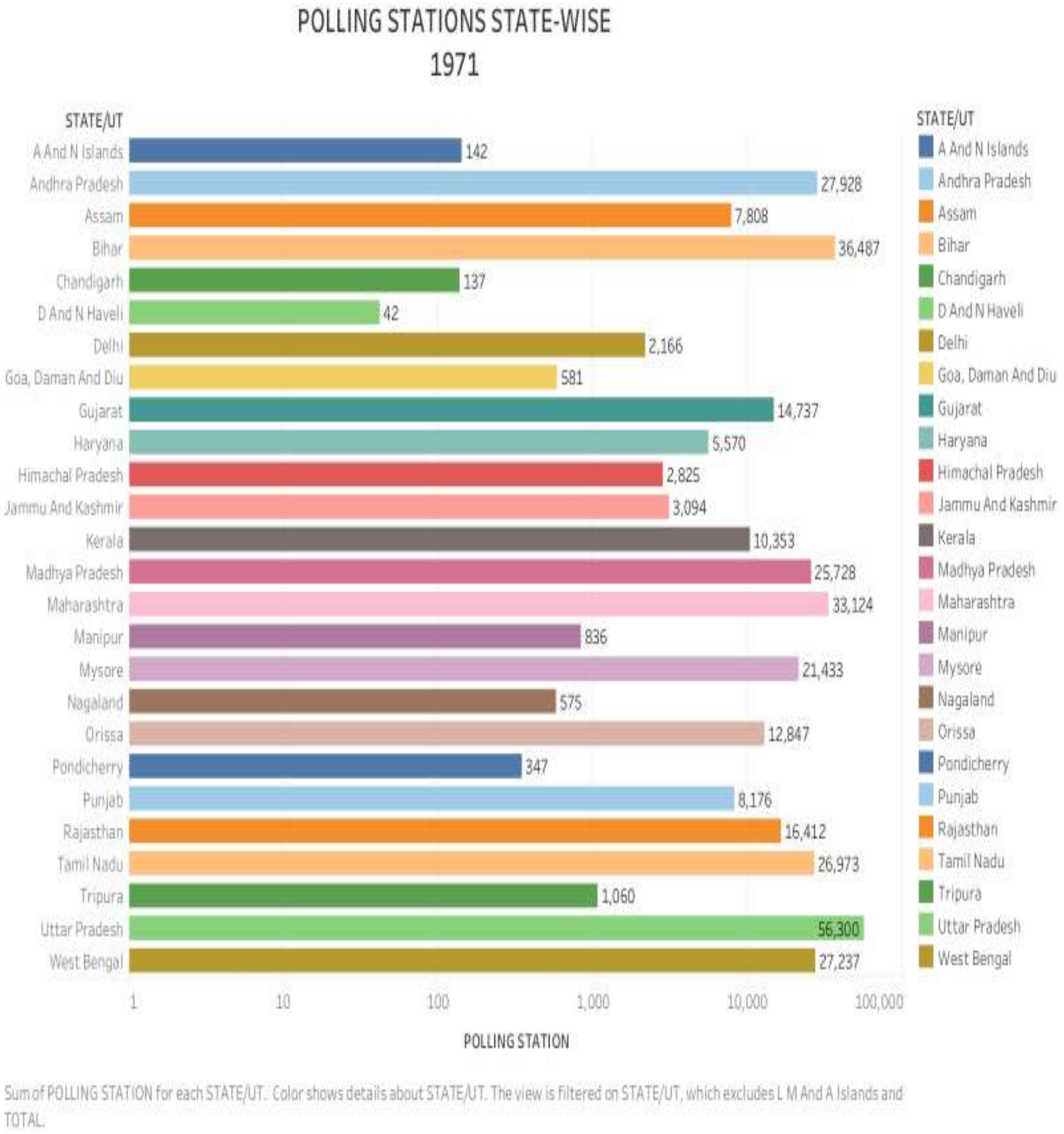


Graph 3.31: Depiction of women empowerment reflecting in the voting behaviour



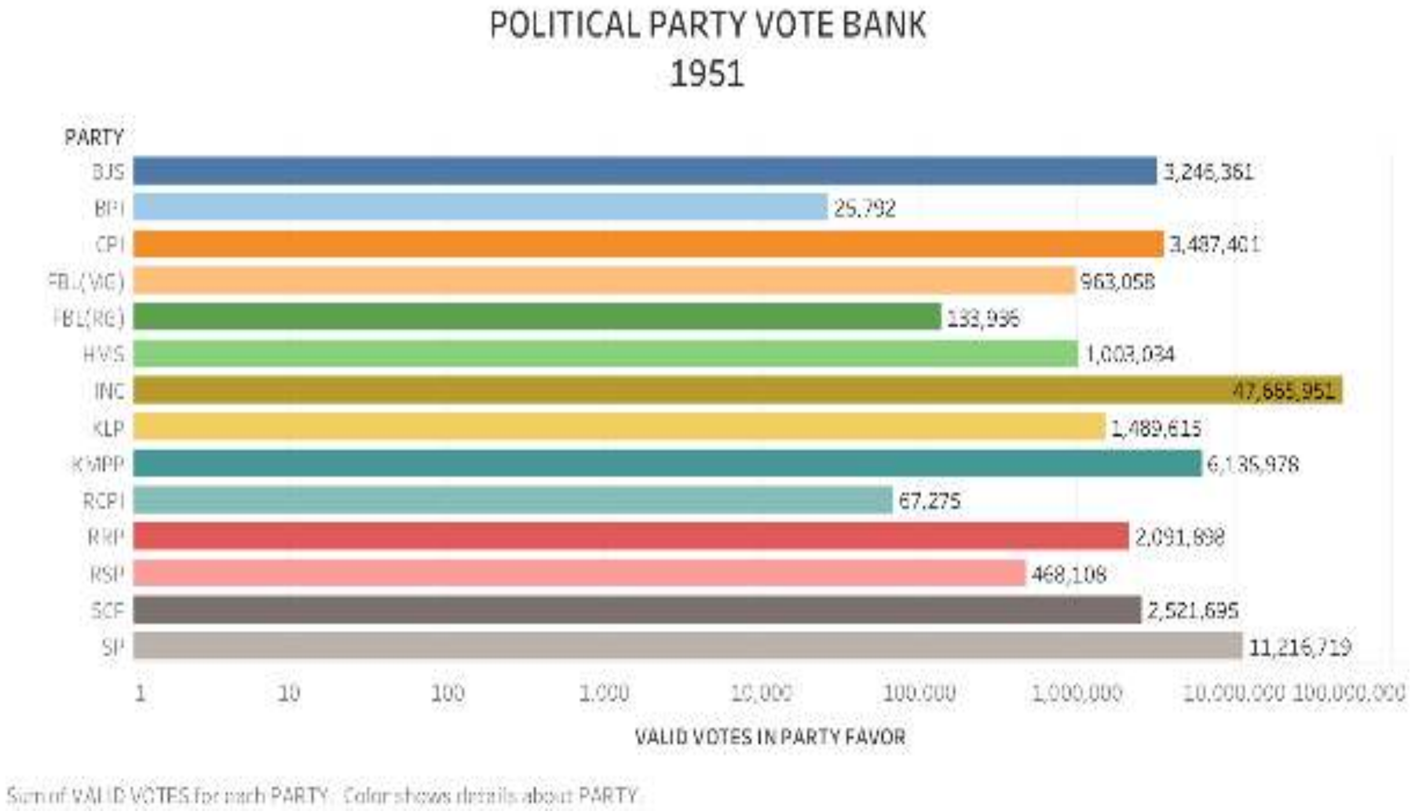
Graph 3.32: Graphical representation of share in voting percentage (Men v/s Women)

ANALYSIS OF POLLING STATIONS IN THE ELECTORAL BATTLE

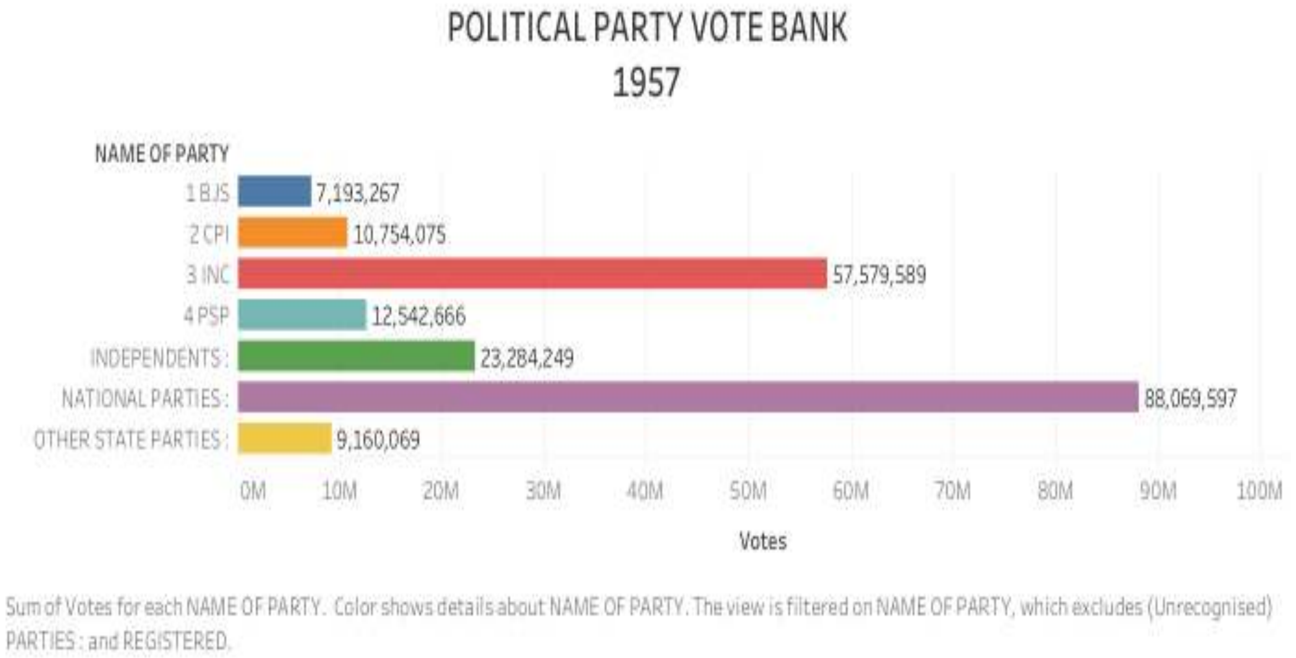


Sum of POLLING STATION for each STATE/UT. Color shows details about STATE/UT. The view is filtered on STATE/UT, which excludes L M And A Islands and TOTAL.

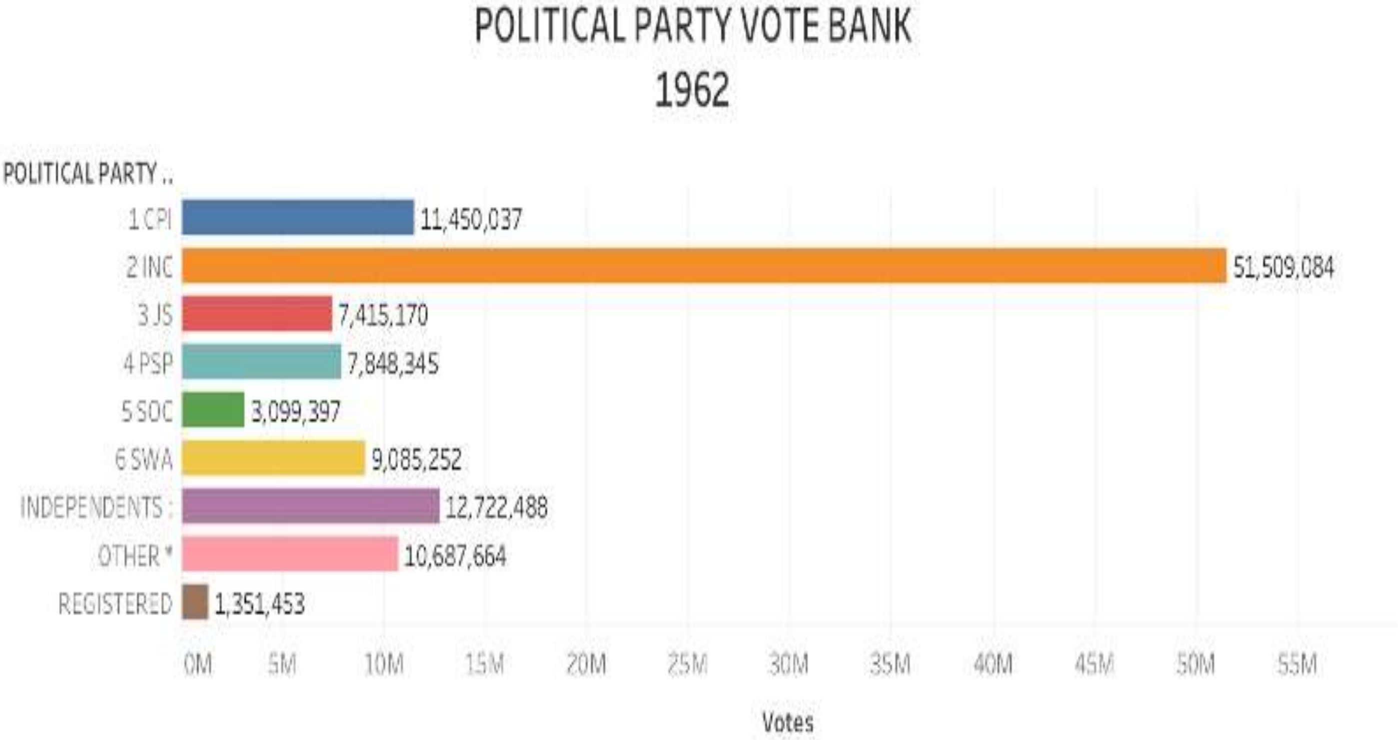
Graph 3.33: Depiction of polling stations across states 1971



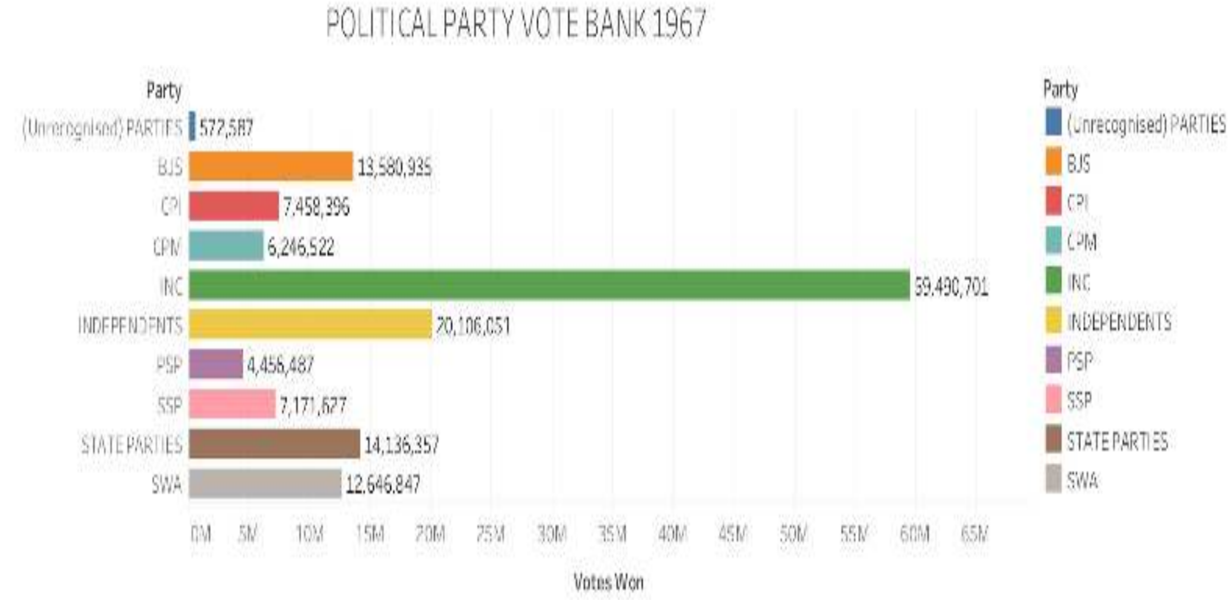
Graph 3.34: Comparison of party wise voter-bank prevailing since 1951



Graph 3.35: Comparison of party wise voter-bank 1957

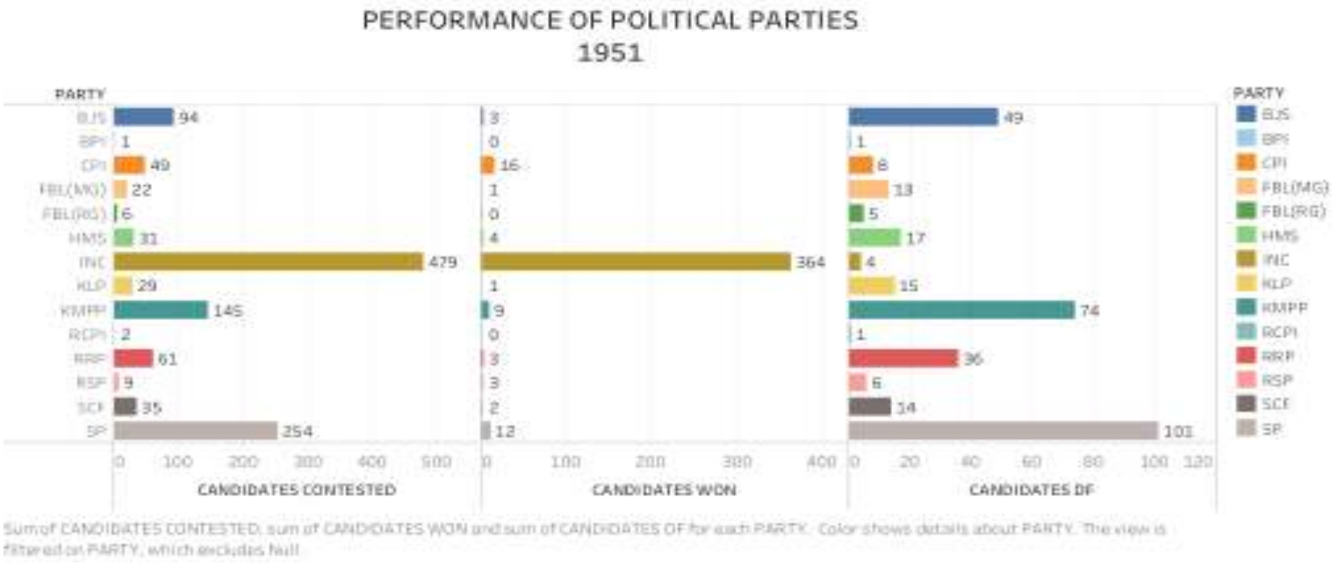


Graph 3.36: Comparison of party wise voter-bank clearly widening the gap as years change

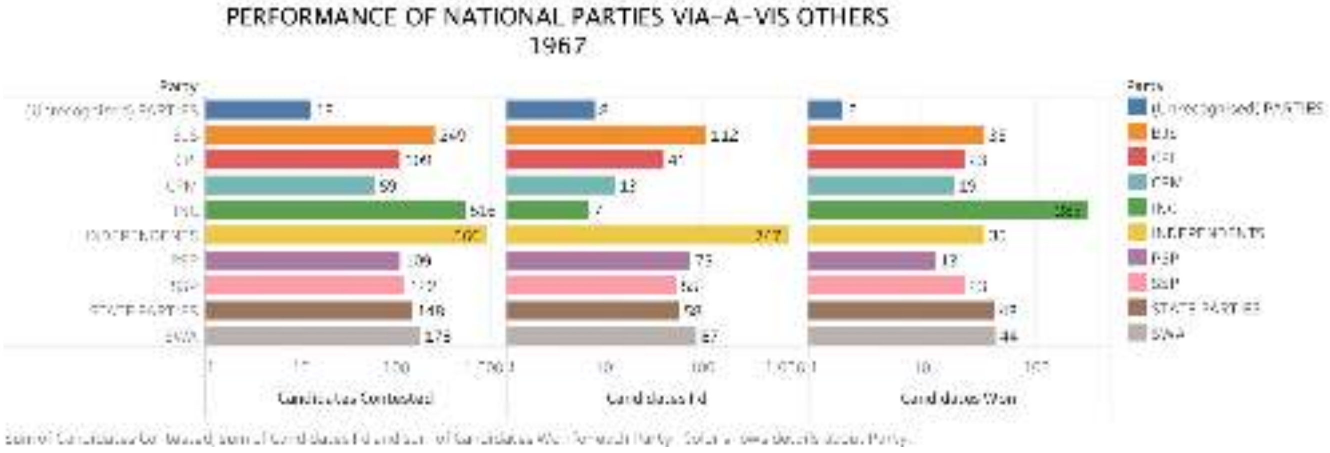


Sum of Votes Won for each Party. Color shows details about Party.

Graph 3.37: Comparison of party wise voter-bank 1967

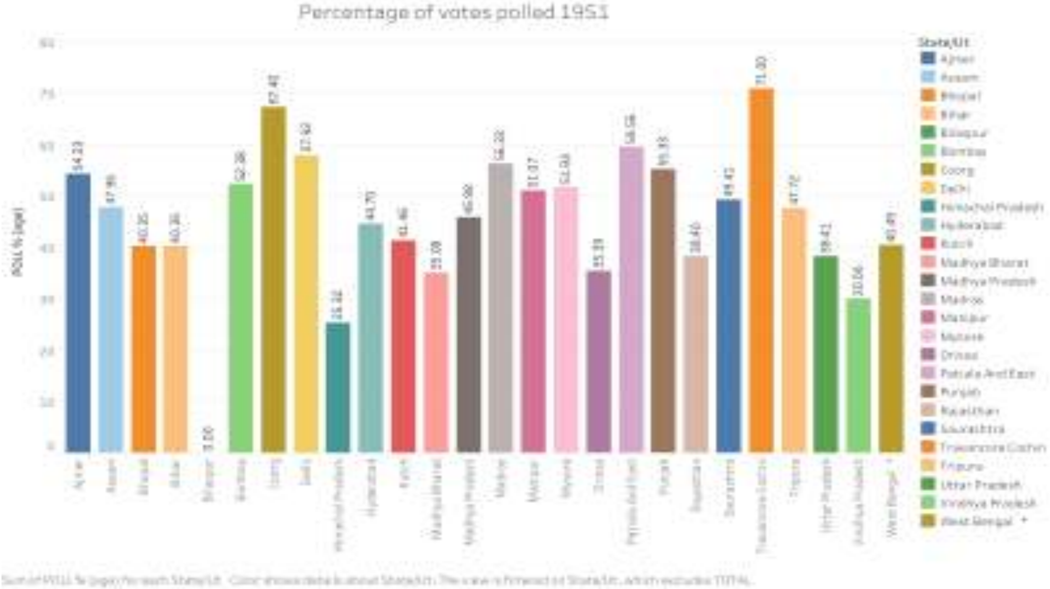


Graph 3.38: Detailed Graphical comparison of party-wise candidates 1951

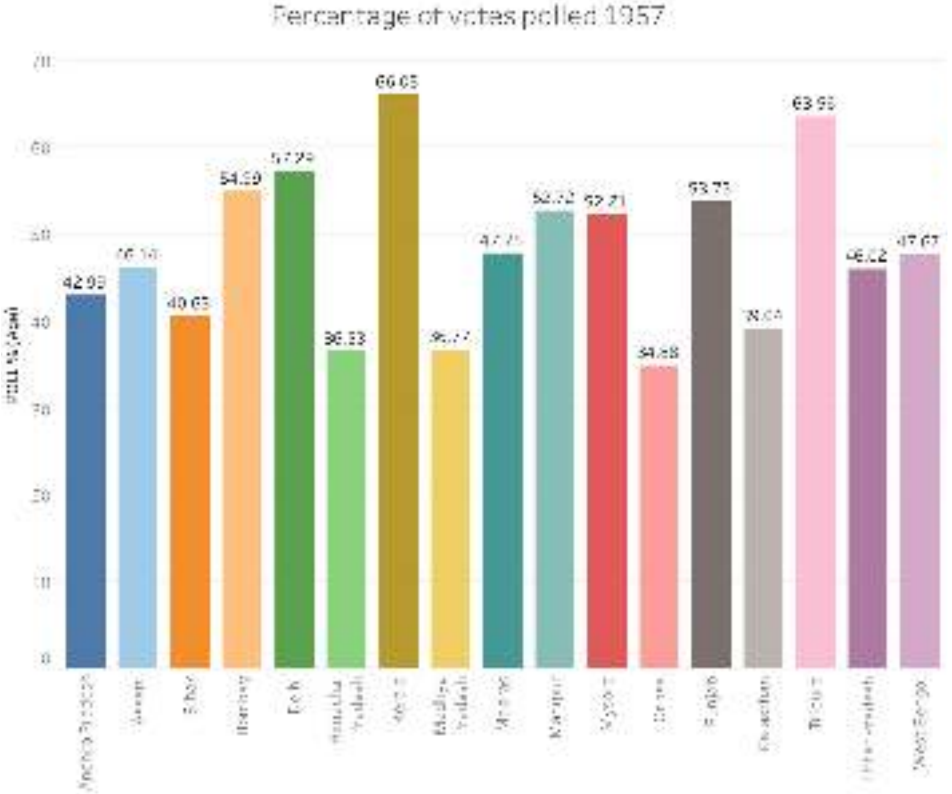


Graph 3.39: Detailed Graphical comparison of party-wise candidates 1967

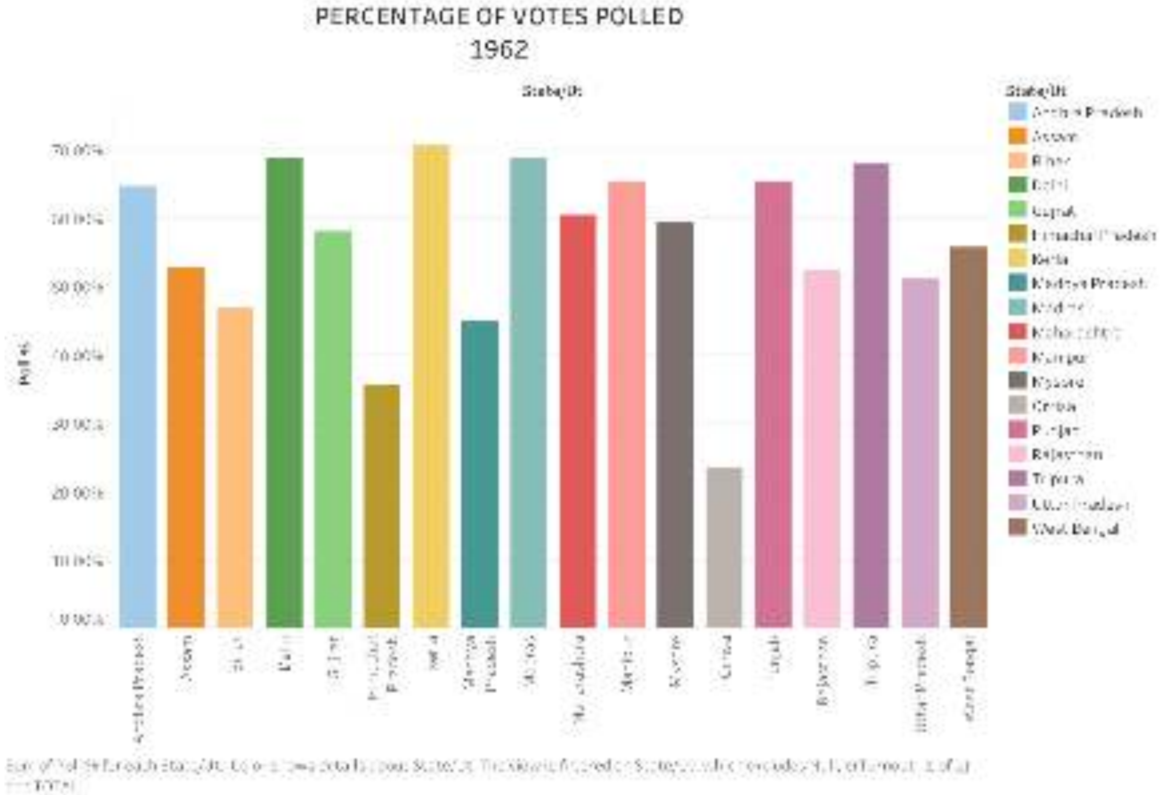
ANALYSIS OF VOTES PERCENTAGE ACROSS YEARS 1951-2019 USING TABLEAU



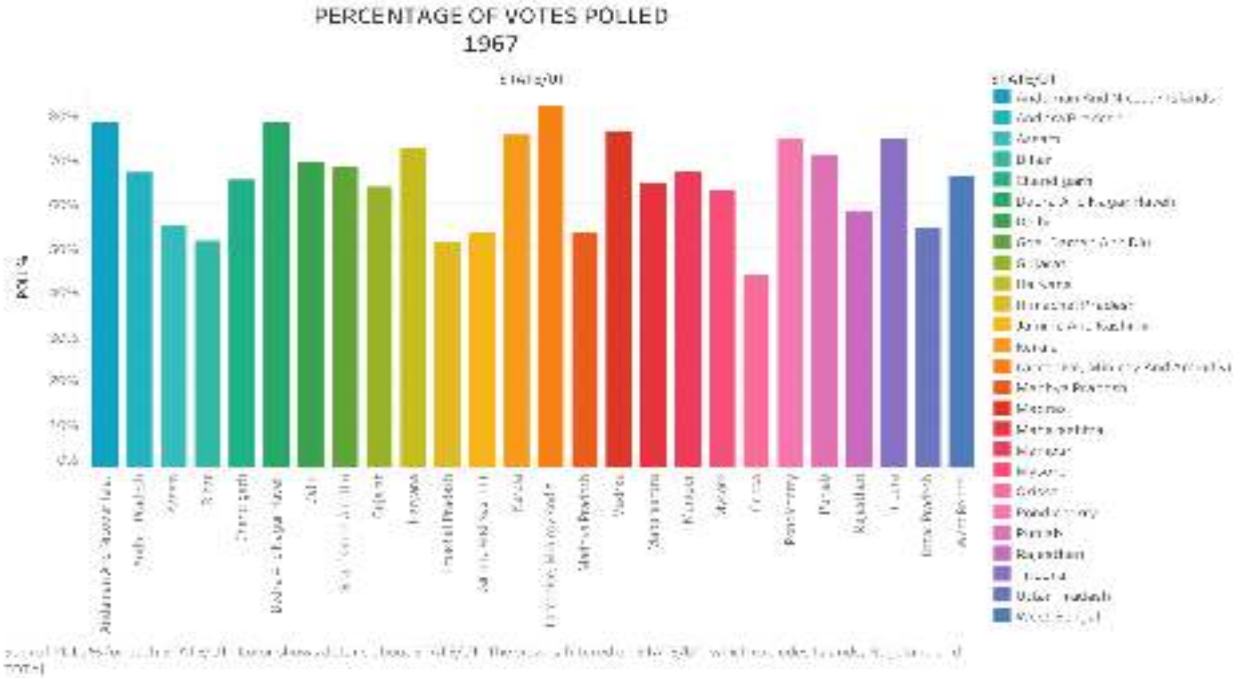
Graph 3.40: Detailed Graphical representation of percentage of votes polled across states in 1951



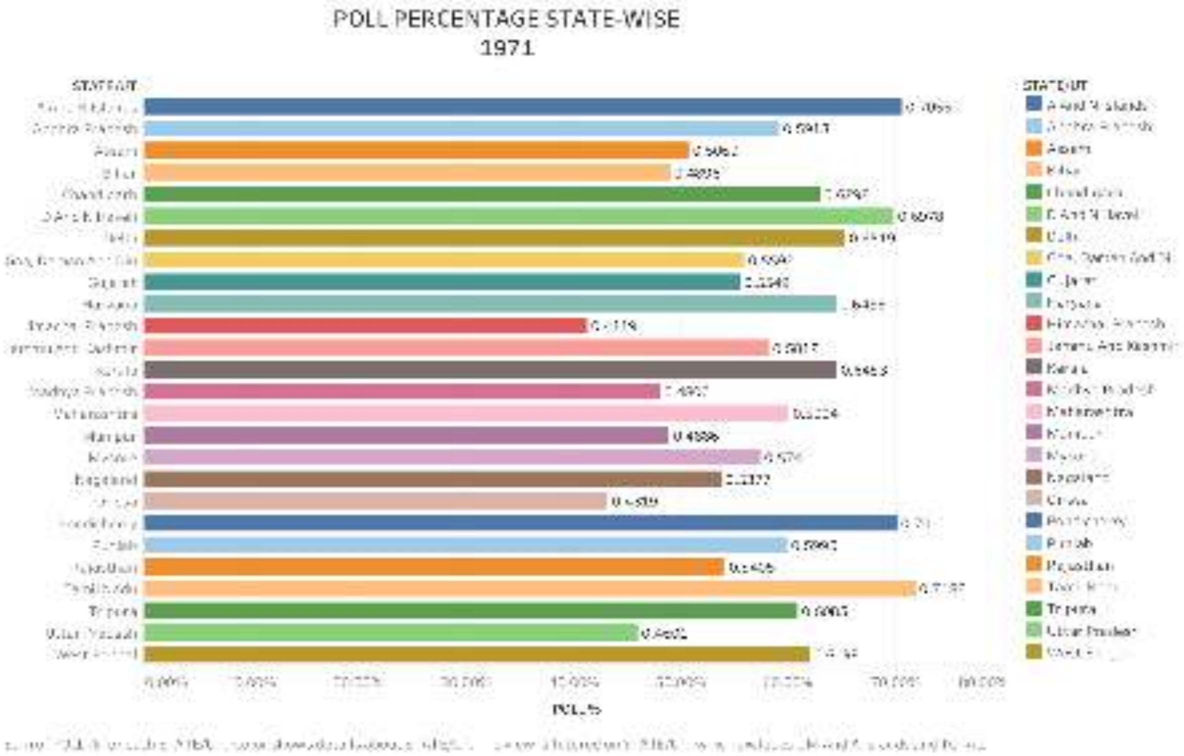
Graph 3.41: Detailed Graphical representation of percentage of votes polled across states in 1957



Graph 3.42: Detailed Graphical representation of percentage of votes polled across states in 1962



Graph 3.43: Detailed Graphical representation of percentage of votes polled across states in 1967



Graph 3.44: Detailed Graphical representation of percentage of votes polled across states in 1971

IMPLEMENTATION OF DEEP LEARNING ALGORITHMS

RANDOM FOREST ALGORITHM:

Random Forest is an adaptable, simple to utilize AI calculation that produces, even without hyper-parameter tuning, an incredible outcome more often than not. It is likewise a standout amongst the most utilized calculations, since it's straightforwardness and the way that it tends to be utilized for both order and relapse undertakings. In this post, you will realize, how the arbitrary backwoods calculation works and a few other significant things about it.

Random Forest is a directed learning calculation. Like you would already be able to see from its name, it makes backwoods and makes it some way or another irregular. The „forest" it manufactures, is a group of Decision Trees, more often than not prepared with the "sacking" strategy. The general thought of the packing technique is that a blend of learning models builds the general result. Random Forest has almost a similar hyperparameters as a choice tree or a stowing classifier. Luckily, you don't need to consolidate a choice tree with a packing classifier and can just effectively utilize the classifier-class of Random Forest. Like I previously stated, with Random Forest, you can likewise manage Regression assignments by utilizing the Random Forest regressor.

Random Forest adds extra arbitrariness to the model, while developing the trees. Rather than hunting down the most significant element while part a hub, it looks for the best element among an arbitrary subset of highlights. This outcomes in a wide decent variety that for the most part results in a superior model.

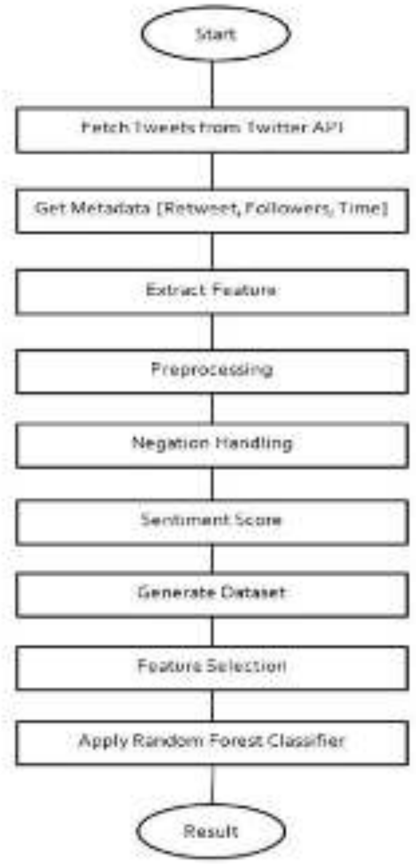


Figure 4.1: Flowchart to depict the working of random forest classifier in the electoral tweet analysis

Subsequently, in Random Forest, just any random subset of the highlights is mulled over by the calculation for part a hub. You can even make trees increasingly arbitrary, by moreover utilizing irregular edges for each component as opposed to hunting down the most ideal edges (like a typical choice tree does).

BAGGING OF DATA

Decision tree application

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier

import os
import pandas as pd
import utils as ul
from sklearn.pipeline import Pipeline
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as mplt

import inspect
fileDir =  os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe()))) # script directory
filePath = fileDir.rsplit('/', 1)[0]


import numpy as np
import matplotlib.pyplot as plt
from itertools import cycle

tfidf_vect, tfidf_vect_ngram, tfidf_vect_ngram_chars, vectorizer = ul.get_vectorize_ngrams()

def plot_precision_recall(precision, recall, f1_score, text, figname):
    num_classes = 8
    colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black', 'gray']
    full_label = []
    labels = ['anger', 'arousal', 'dominance', 'faith',  'fear', 'joy', 'neutral', 'sadness']
    plt.figure(figsize=(7, 8))
    lines = []
    count = 0
    for f_score in f1_score:
        x = np.linspace(0.01, 1)
        y = f_score * x / (2 * x - f_score)
        l, = plt.plot(x[y >= 0], y[y >= 0], color=colors[count], alpha=0.2)
        full_label.append(labels[count] + ',f1={0:0.1f},'.format(f_score) + 'p={0:0.1f},'.format(precision[count]) +
'r={0:0.1f}'.format(recall[count]),)
        count = count + 1
```

```
    for i in range(num_classes):
        l, = plt.plot(recall[i], precision[i], color=colors[i], lw=2)
        plt.legend([l], [full_label[i] ], loc=(0.6, .7), prop=dict(size=8))
        lines.append(l)

    fig = plt.gcf()
    fig.subplots_adjust(bottom=0.25)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve for' + text, fontsize =6)
    plt.legend(lines, full_label, loc=(0.6, .7), prop=dict(size=8))
    mplt.savefig(filePath + "/figs/congress/bagging/"+ figname + ".png")  #save the figs


def apply_rf_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = RandomForestClassifier(n_estimators=500,
                                 max_features=0.25,
                                 criterion="entropy",
                                 class_weight="balanced")
    clf.fit(X, training_set_y)

    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore_1 = metrics.accuracy_score(test_set_y, pred)
    print("RandomForest Accuracy" , pscore_1)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("RandomForest Precision =", precisions)
    print("RandomForest Recall =", recall)
    print("RandomForest F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector with RandomForestClassifier'
, 'rf-cv')

    clf = RandomForestClassifier(n_estimators=500,
                                 max_features=0.25,
                                 criterion="entropy",
                                 class_weight="balanced")
```

```
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    pscore_2 = metrics.accuracy_score(test_set_y, pred)
    print("RandomForest Accuracy" , pscore_2)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("RandomForest Precision =", precisions)
    print("RandomForest Recall =", recall)
    print("RandomForest F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :TfIdf with RandomForestClassifier', 'rf-
tdidf')



    text_clf = Pipeline([('vect', vectorizer), ('tfidf', TfidfTransformer()), ('rf', RandomForestClassifier(n_estimato
rs=500,
                                max_features=0.25,
                                criterion="entropy",
                                class_weight="balanced"))])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore_3 = metrics.accuracy_score(test_set_y, pred)
    print("RandomForest Accuracy" , pscore_3)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("RandomForest Precision =", precisions)
    print("RandomForest Recall =", recall)
    print("RandomForest F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector & TfIdf with RandomForestCla
ssifier', 'rf-pip')


def apply_extratree_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = ExtraTreesClassifier(criterion="gini",
                 max_depth=None,
                 min_samples_split=2,
                 min_samples_leaf=1,
                 max_features=0.25)
```

```
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    print("ExtraTrees-count vector" + str(pred))
    pscore_1 = metrics.accuracy_score(test_set_y, pred)
    print("ExtraTrees Accuracy" , pscore_1)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("ExtraTrees Precision =", precisions)
    print("ExtraTrees Recall =", recall)
    print("ExtraTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector with ExtraTreesClassifier', 'et-cv')



    clf = ExtraTreesClassifier(criterion="gini",
                 max_depth=None,
                 min_samples_split=2,
                 min_samples_leaf=1,
                 max_features=0.25)
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    print("ExtraTrees - tfidf" + str(pred))
    pscore_2 = metrics.accuracy_score(test_set_y, pred)
    print("ExtraTrees Accuracy" , pscore_2)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("ExtraTrees Precision =", precisions)
    print("ExtraTrees Recall =", recall)
    print("ExtraTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :TfIdf with ExtraTreesClassifier', 'et-tdif')

    text_clf = Pipeline([('vect', vectorizer), ('tfidf', TfidfTransformer()), ('eb', ExtraTreesClassifier(criterion="gini",
                 max_depth=None,
                 min_samples_split=2,
                 min_samples_leaf=1,
                 max_features=0.25))])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore_3 = metrics.accuracy_score(test_set_y, pred)
    print("ExtraTrees Accuracy" , pscore_3)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("ExtraTrees Precision =", precisions)
    print("ExtraTrees Recall =", recall)
    print("ExtraTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector & TfIdf with ExtraTreesClassifier', 'et-pip')
```

```
def apply_decisiontree_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = DecisionTreeClassifier()
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore_1 = metrics.accuracy_score(test_set_y, pred)
    print("DecisionTrees Accuracy" , pscore_1)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("DecisionTrees Precision =", precisions)
    print("DecisionTrees Recall =", recall)
    print("DecisionTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector with DecisionTreeClassifier'
, 'dt-cv')


    clf = DecisionTreeClassifier()
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    pscore_2 = metrics.accuracy_score(test_set_y, pred)
    print("DecisionTrees Accuracy" , pscore_2)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("DecisionTrees Precision =", precisions)
    print("DecisionTrees Recall =", recall)
    print("DecisionTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :TfIdf with DecisionTreeClassifier', 'dt-
tdif')


    text_clf = Pipeline([('vect', vectorizer), ('tfidf', TfidfTransformer()), ('eb', DecisionTreeClassifier())])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore_3 = metrics.accuracy_score(test_set_y, pred)
    print("DecisionTrees Accuracy" , pscore_3)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("DecisionTrees Precision =", precisions)
    print("DecisionTrees Recall =", recall)
    print("DecisionTrees F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector & TfIdf with DecisionTreeCla
ssifier', 'dt-pip')
```

```
def apply_clf_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = BaggingClassifier(n_estimators =25,
                max_features=0.25)
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore_1 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_1)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector with BaggingClassifier', 'bg
-cv')


    clf = BaggingClassifier(n_estimators =25,
                max_features=0.25)
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    pscore_2 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_2)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)


    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :TfIdf with BaggingClassifier', 'bg-tdif'
)

    text_clf = Pipeline([('vect', vectorizer), ('tfidf', TfidfTransformer()), ('pipbg', BaggingClassifier(n_estimators
=25,
                max_features=0.25))])
```

```
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore_3 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_3)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector & TfIdf with BaggingClassifier on RandomForest', 'bg-pip')


def apply_clf_rf_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    rf = RandomForestClassifier(n_estimators=500,
                                max_features=0.25,
                                criterion="entropy",
                                class_weight="balanced")

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = BaggingClassifier(base_estimator = rf, n_estimators =25,
                max_features=0.25)
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore_1 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_1)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :CountVector with BaggingClassifier on RandomForest', 'bg-rf-cv')


    clf = BaggingClassifier(base_estimator = rf, n_estimators =25,
                max_features=0.25)
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    pscore_2 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_2)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)
```

```
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :TfIdf with BaggingClassifier on RandomFo
rest', 'bg-rf-tdif')

    text_clf = Pipeline([('vect', vectorizer), ('tfidf', TfidfTransformer()), ('pipbg', BaggingClassifier(n_estimators
=25,
                 max_features=0.25))])


    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore_3 = metrics.accuracy_score(test_set_y, pred)
    print("Bagging Accuracy" , pscore_3)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Bagging Precision =", precisions)
    print("Bagging Recall =", recall)
    print("Bagging F1 score =", f1_score)
    plot_precision_recall(precisions, recall, f1_score, ' ' + party_name + ' :Count Vector & TfIdf with BaggingClassif
ier on RandomForest', 'bg-rf-pip')



if __name__ == '__main__':

    tain_path = filePath + "/train/sentiments/train-dataset/"
    test_path = filePath + "/train/sentiments/test-dataset/"
    train_file_names = os.listdir(tain_path)
    test_file_names = os.listdir(test_path)
    party_names = ['Bjp', 'Congress', 'Bjp-Congress', 'Neutral']
    for i in range(0, len(train_file_names)):
        train = pd.read_csv(tain_path+ train_file_names[i]).dropna()
        test = pd.read_csv(test_path+ test_file_names[i]).dropna()

        print("Processing file.." + train_file_names[i])

        training_set_x = train['tweet'].values
        training_set_y = ul.label_encode(train['mood'])
        test_set_x = test['tweet'].values
        test_set_y = ul.label_encode(test['mood'])
        apply_rf_bagging(training_set_x, training_set_y, test_set_x, test_set_y, party_names[i])
        apply_clf_bagging(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
        apply_extratree_bagging(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
        apply_decisiontree_bagging(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
        apply_clf_rf_bagging(training_set_x, training_set_y, test_set_x,
                                                                     test_set_y,  party_names[i])
```

Bagging Classifier : BJP Sentiment prediction

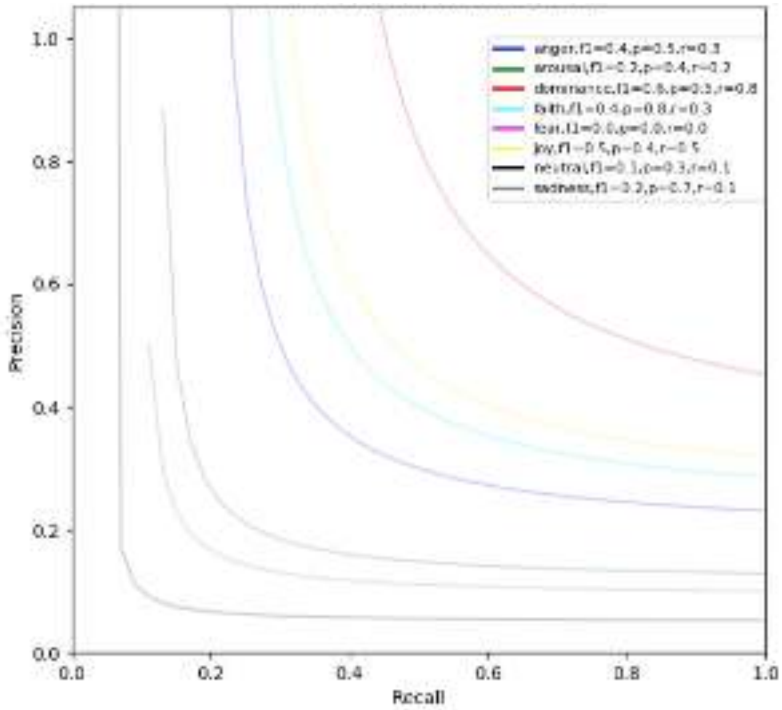


Figure 3.2: Precision recall curve for BJP: Count Vector with bagging classifier

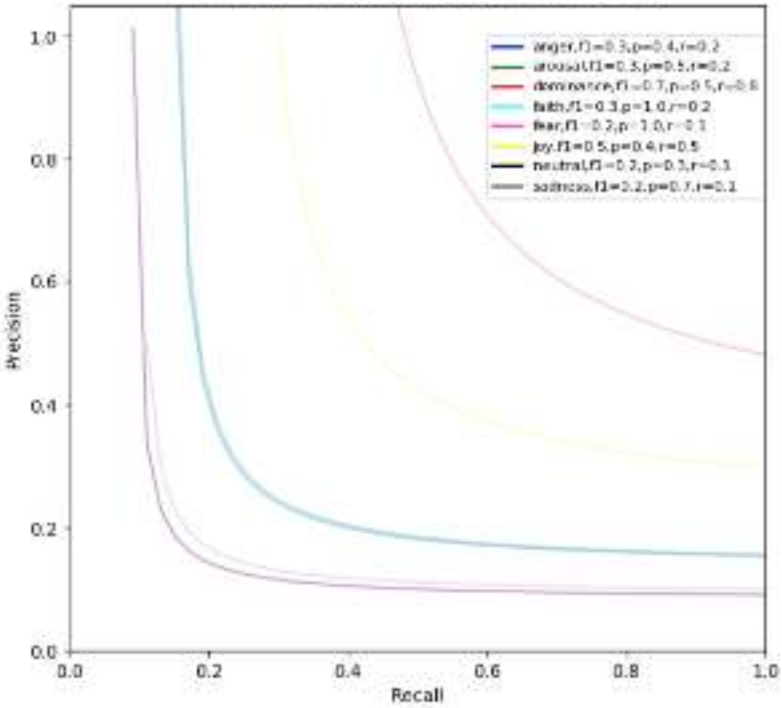


Figure 3.3: Precision recall curve for BJP: Random forest with bagging classifier

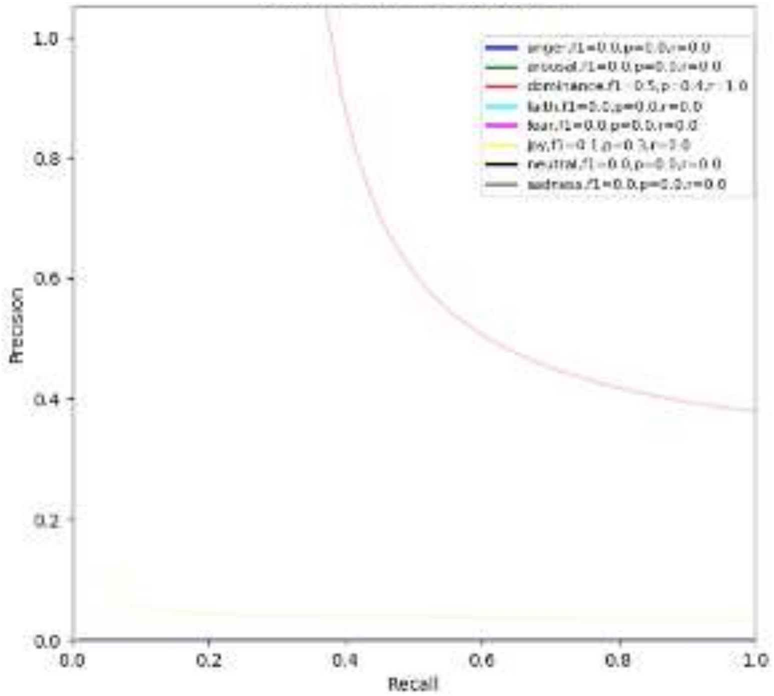


Figure 3.4: Precision recall curve for BJP: TfIdf with bagging classifier

Bagging Classifier : Congress Sentiment prediction

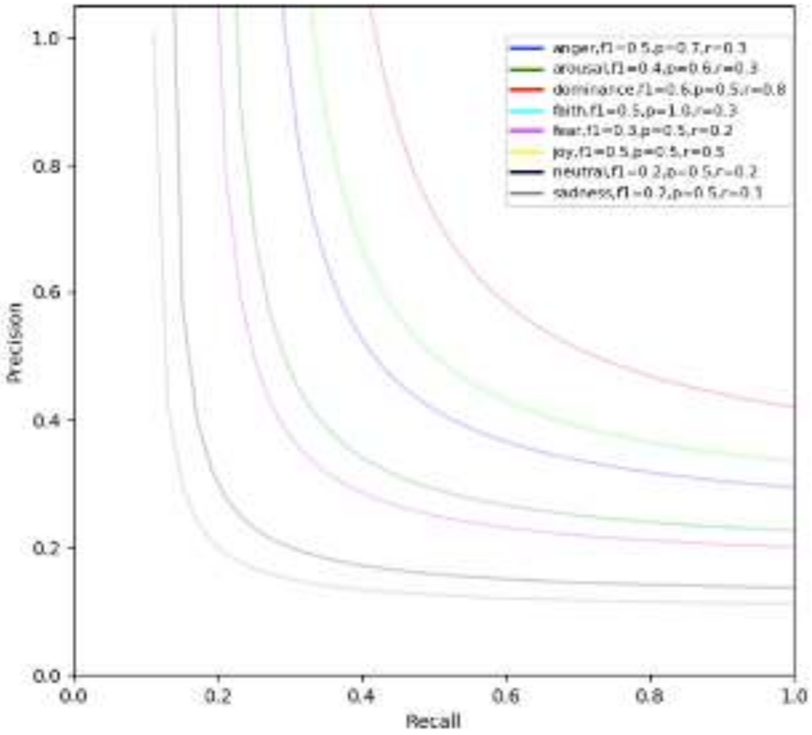


Figure 3.5: Precision recall curve for Congress: Count Vector with bagging classifier

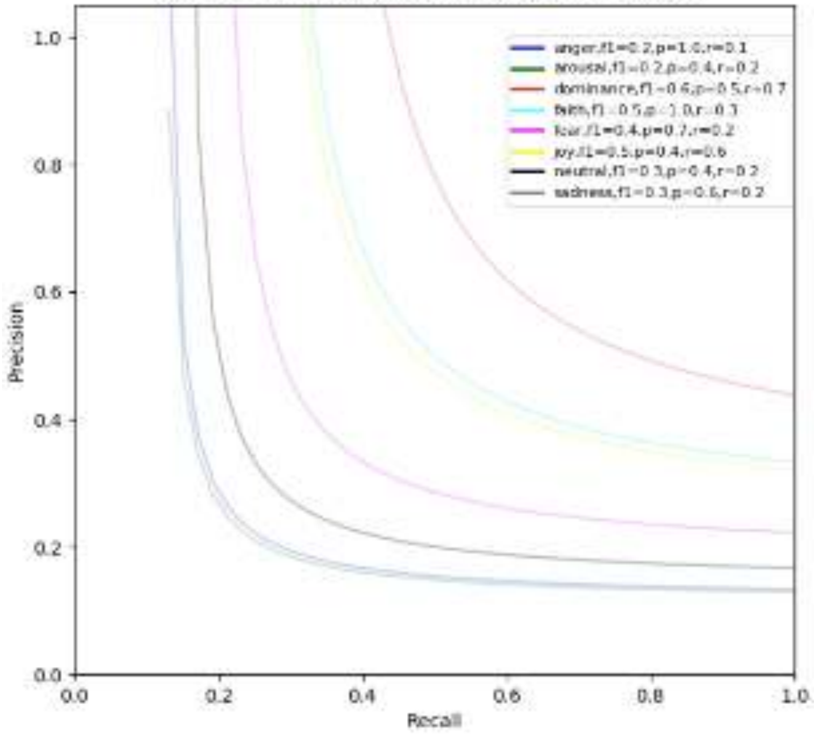


Figure 3.6: Precision recall curve for Congress: Random forest with bagging classifier

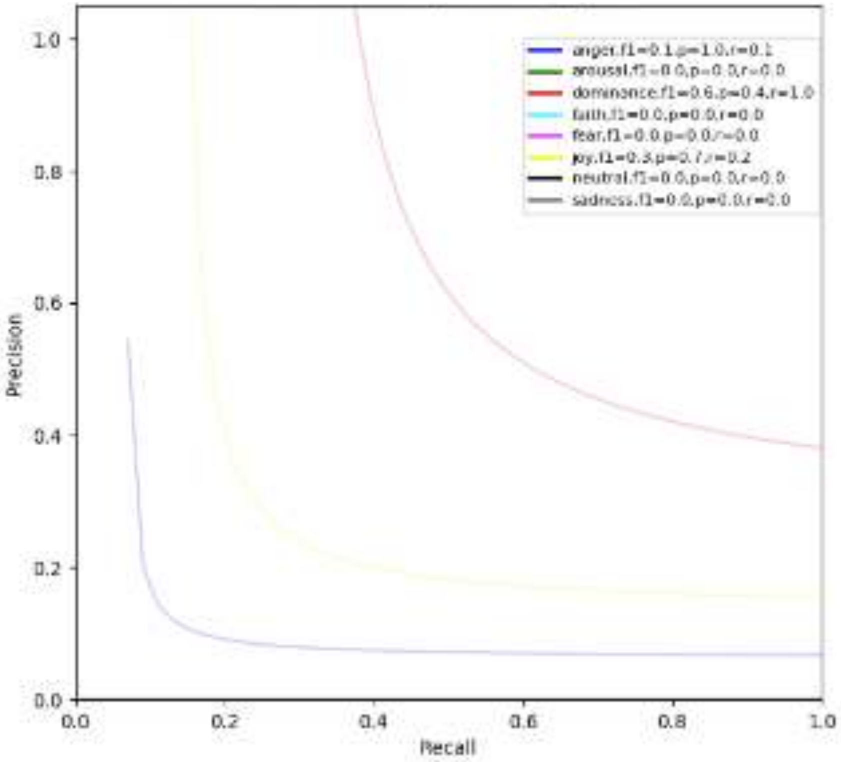


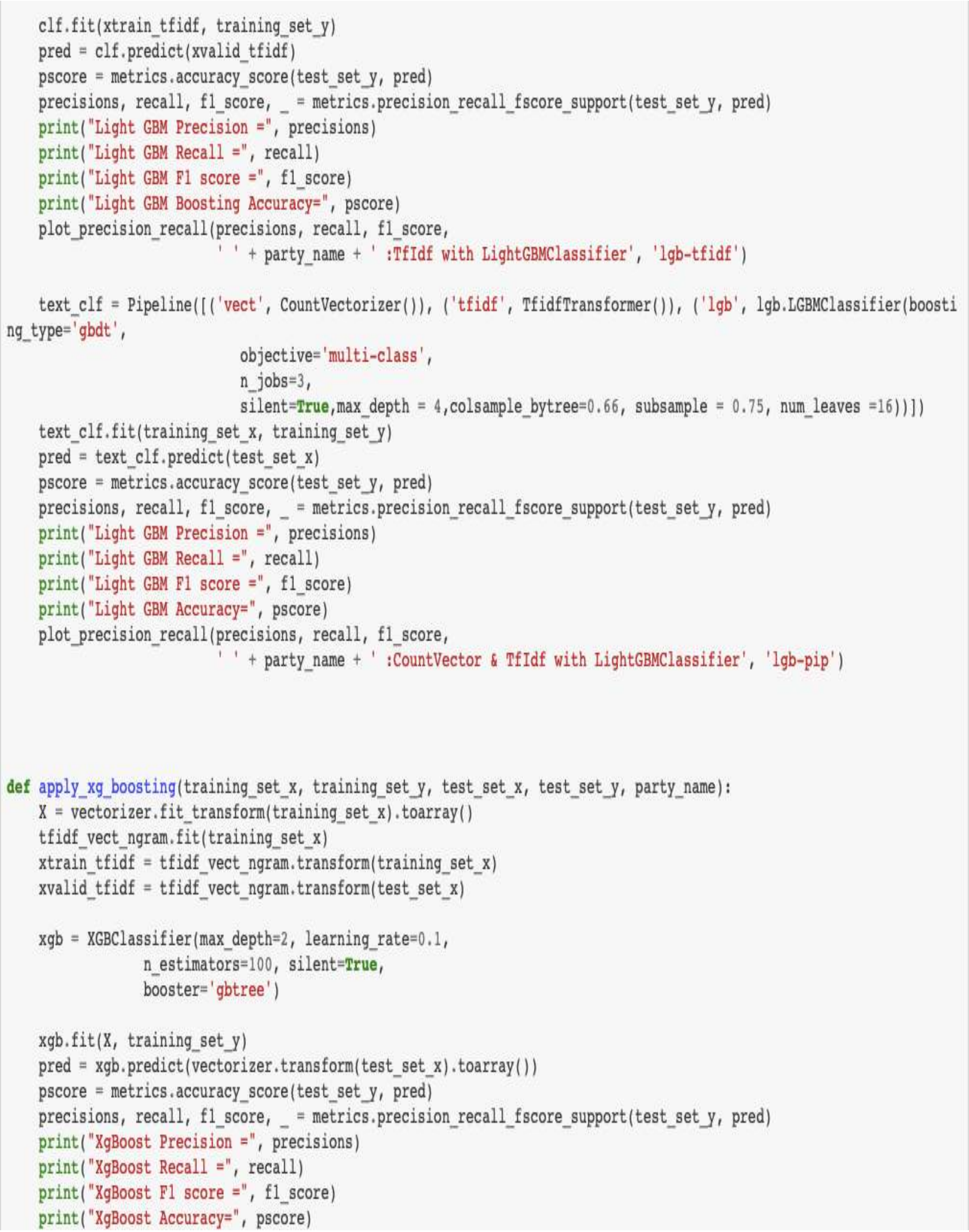
Figure 3.7: Precision recall curve for Congress: Tfldf with bagging classifier

BOOSTING OF DATA

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import metrics
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import  XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
import datetime
from sklearn.preprocessing import label_binarize
from catboost import CatBoostClassifier
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import pandas as pd
import utils as ul

import os
from sklearn.pipeline import Pipeline
import lightgbm as lgb

import inspect
fileDir =  os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe()))) # script directory
filePath = fileDir.rsplit('/', 1)[0]


tfidf_vect, tfidf_vect_ngram, tfidf_vect_ngram_chars, vectorizer = ul.get_vectorize_ngrams()


def plot_precision_recall(precision, recall, f1_score, text, figname):
    num_classes = 8
    colors = ['blue', 'green', 'red', 'cyan', 'magenta', 'yellow', 'black', 'gray']
    full_label = []
    labels = ['anger', 'arousal', 'dominance', 'faith',  'fear', 'joy', 'neutral', 'sadness']
    plt.figure(figsize=(7, 8))
    lines = []
    count = 0
    for f_score in f1_score:
        x = np.linspace(0.01, 1)
        y = f_score * x / (2 * x - f_score)
        l, = plt.plot(x[y >= 0], y[y >= 0], color=colors[count], alpha=0.2)
        full_label.append(labels[count] + ',f1={0:0.1f},'.format(f_score) + 'p={0:0.1f},'.format(precision[count]) +
'r={0:0.1f}'.format(recall[count]),)
        count = count + 1
```

```
    for i in range(num_classes):
        l, = plt.plot(recall[i], precision[i], color=colors[i], lw=2)
        plt.legend([l], [full_label[i] ], loc=(0.6, .7), prop=dict(size=8))
        lines.append(l)

    fig = plt.gcf()
    fig.subplots_adjust(bottom=0.25)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve for ' + text, fontsize =6)
    plt.legend(lines, full_label, loc=(0.6, .7), prop=dict(size=8))
    plt.savefig(filePath + "/figs/bjp/boosting/"+ figname + ".png")

def timer(start_time=None):
    if not start_time:
        start_time = datetime.datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.datetime.now() - start_time).total_seconds(), 3600)
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round(tsec, 2)))

def apply_gradient_boosting(training_set_x, training_set_y, test_set_x, test_set_y, party_name):


    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = GradientBoostingClassifier(n_estimators =100, learning_rate =0.1, max_depth=6, min_samples_leaf =1, max_feat
ures=1.0)
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("GradientBoosting Precision =", precisions)
    print("GradientBoosting Recall =", recall)
    print("GradientBoosting F1 score =", f1_score)
    print("Gradient Boosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector with GradientBoostClassifier', 'gb-count')

    clf = GradientBoostingClassifier(n_estimators =100, learning_rate =0.1, max_depth=6, min_samples_leaf =1, max_feat
ures=1.0)
    clf.fit(xtrain_tfidf, training_set_y)
```

```
    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('gb', GradientBoostingClassifier
(n_estimators =100, learning_rate =0.1, max_depth=6, min_samples_leaf =1, max_features=1.0))])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("GradientBoosting Precision =", precisions)
    print("GradientBoosting Recall =", recall)
    print("GradientBoosting F1 score =", f1_score)
    print("Gradient Boosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector & TfIdf with GradientBoostClassifier', 'gb-pip')


def apply_lightgbm_boosting(training_set_x, training_set_y, test_set_x, test_set_y, party_name):
    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = lgb.LGBMClassifier(boosting_type= 'gbdt',
          objective = 'multi-class',
          n_jobs = 3,
          silent = True,
          max_depth = 4,colsample_bytree=0.66, subsample = 0.75, num_leaves =16)

    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Light GBM Precision =", precisions)
    print("Light GBM Recall =", recall)
    print("Light GBM F1 score =", f1_score)
    print("Light GBM Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector with LightGBMClassifier', 'lgb-count')

    clf = lgb.LGBMClassifier(boosting_type='gbdt',
                             objective='multi-class',
                             n_jobs=3,
                             silent=True,
                             max_depth=4, colsample_bytree=0.66, subsample=0.75, num_leaves=16)
```



```
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Light GBM Precision =", precisions)
    print("Light GBM Recall =", recall)
    print("Light GBM F1 score =", f1_score)
    print("Light GBM Boosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :TfIdf with LightGBMClassifier', 'lgb-tfidf')

    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('lgb', lgb.LGBMClassifier(boosti
ng_type='gbdt',
                             objective='multi-class',
                             n_jobs=3,
                             silent=True,max_depth = 4,colsample_bytree=0.66, subsample = 0.75, num_leaves =16))])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("Light GBM Precision =", precisions)
    print("Light GBM Recall =", recall)
    print("Light GBM F1 score =", f1_score)
    print("Light GBM Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector & TfIdf with LightGBMClassifier', 'lgb-pip')




def apply_xg_boosting(training_set_x, training_set_y, test_set_x, test_set_y, party_name):
    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    xgb = XGBClassifier(max_depth=2, learning_rate=0.1,
                 n_estimators=100, silent=True,
                 booster='gbtree')

    xgb.fit(X, training_set_y)
    pred = xgb.predict(vectorizer.transform(test_set_x).toarray())
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("XgBoost Precision =", precisions)
    print("XgBoost Recall =", recall)
    print("XgBoost F1 score =", f1_score)
    print("XgBoost Accuracy=", pscore)
```

```
    xgb.fit(xtrain_tfidf, training_set_y)
    pred = xgb.predict(xvalid_tfidf)
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("XgBoost Precision =", precisions)
    print("XgBoost Recall =", recall)
    print("XgBoost F1 score =", f1_score)
    print("XgBoost Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :TfIdf with XgBoostClassifier', 'xgb-tfidf')

    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('xgb', XGBClassifier(max_depth=2
, learning_rate=0.1,
                        n_estimators=100, silent=True,
                        booster='gbtree'))])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("XgBoost Precision =", precisions)
    print("XgBoost Recall =", recall)
    print("XgBoost F1 score =", f1_score)
    print("XgBoost Boosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector & TfIdf with XgBoostClassifier', 'xgb-pip')


def apply_ada_boosting(training_set_x, training_set_y, test_set_x, test_set_y, party_name):

    X = vectorizer.fit_transform(training_set_x).toarray()
    tfidf_vect_ngram.fit(training_set_x)
    xtrain_tfidf = tfidf_vect_ngram.transform(training_set_x)
    xvalid_tfidf = tfidf_vect_ngram.transform(test_set_x)

    clf = AdaBoostClassifier()
    clf.fit(X, training_set_y)
    pred = clf.predict(vectorizer.transform(test_set_x).toarray())
    pscore = metrics.accuracy_score(test_set_y, pred)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("AdaBoosting Precision =", precisions)
    print("AdaBoosting Recall =", recall)
    print("AdaBoosting F1 score =", f1_score)
    print("AdaBoosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector with AdaBoostClassifier', 'ab-cv')

    clf = AdaBoostClassifier()
    clf.fit(xtrain_tfidf, training_set_y)
    pred = clf.predict(xvalid_tfidf)
```

```
    text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('ab', AdaBoostClassifier())])
    text_clf.fit(training_set_x, training_set_y)
    pred = text_clf.predict(test_set_x)
    pscore = metrics.accuracy_score(test_set_y, pred)
    print("AdaBoost - pipeline count vector + tdiff" + str(pred))
    print(pscore)
    precisions, recall, f1_score, _ = metrics.precision_recall_fscore_support(test_set_y, pred)
    print("AdaBoosting Precision =", precisions)
    print("AdaBoosting Recall =", recall)
    print("AdaBoosting F1 score =", f1_score)
    print("AdaBoosting Accuracy=", pscore)
    plot_precision_recall(precisions, recall, f1_score,
                          ' ' + party_name + ' :CountVector & TfIdf with AdaBoostClassifier', 'ab-pip')



if __name__ == '__main__':

    tain_path = filePath + "/train/sentiments/train-dataset/"
    test_path = filePath + "/train/sentiments/test-dataset/"
    train_file_names = os.listdir(tain_path)
    test_file_names = os.listdir(test_path)
    party_names = ['Bjp', 'Congress', 'Bjp-Congress', 'Neutral']
    for i in range(0, len(train_file_names)):
        train = pd.read_csv(tain_path+ train_file_names[i]).dropna()
        test = pd.read_csv(test_path+ test_file_names[i]).dropna()

        print("Processing file.." + train_file_names[i])

        training_set_x = train['tweet'].values
        training_set_y = ul.label_encode(train['mood'])
        test_set_x = test['tweet'].values
        test_set_y = ul.label_encode(test['mood'])
        apply_xg_boosting(training_set_x, training_set_y, test_set_x, test_set_y, party_names[i])
        apply_gradient_boosting(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
        apply_ada_boosting(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
        apply_lightgbm_boosting(training_set_x, training_set_y, test_set_x, test_set_y,  party_names[i])
```

Result for BJP

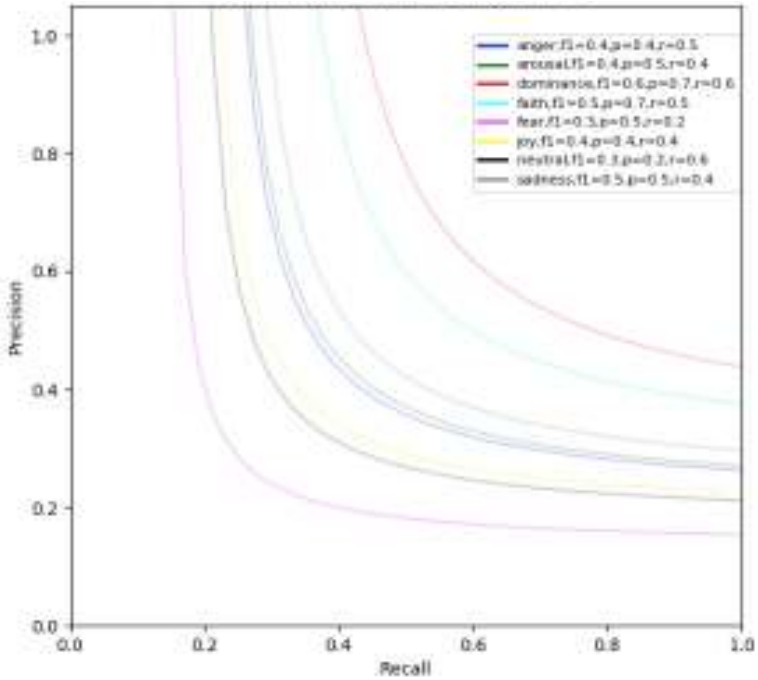


Figure 4.1: Precision recall curve for BJP: Decision tree classifier

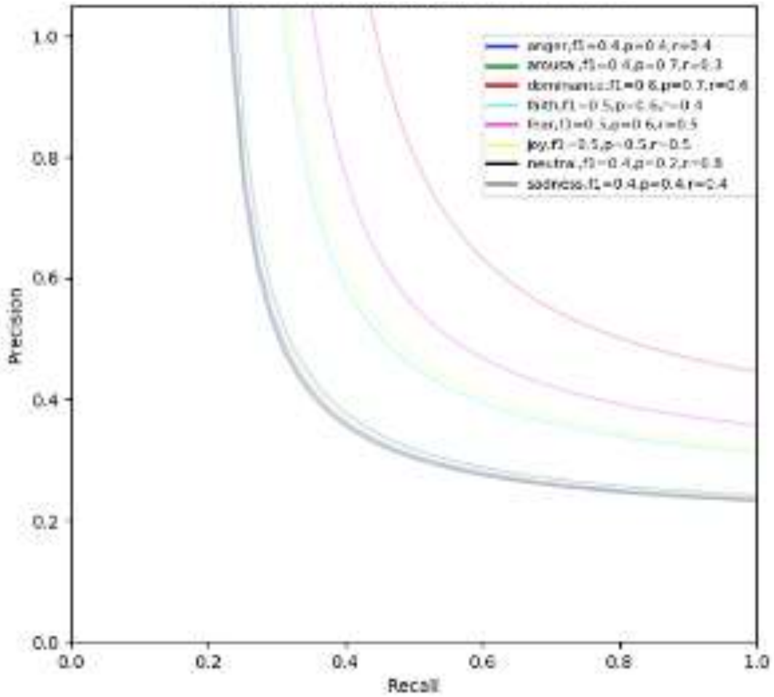


Figure 4.2: Precision recall curve for BJP: Tfldf with Decision tree classifier and countVector

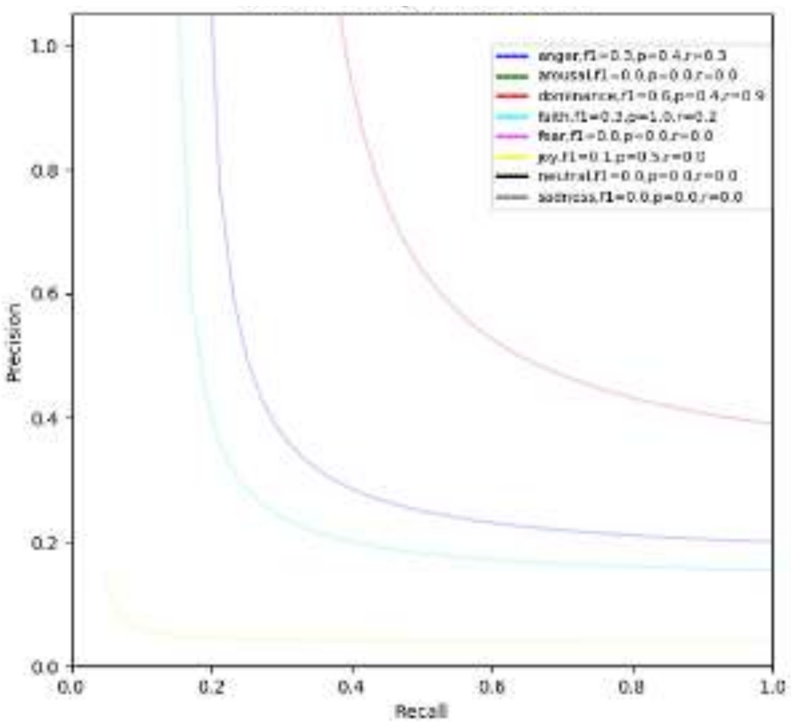


Figure 4.3: Precision recall curve for BJP: Tfldf with Decision tree classifier

Results for Congress

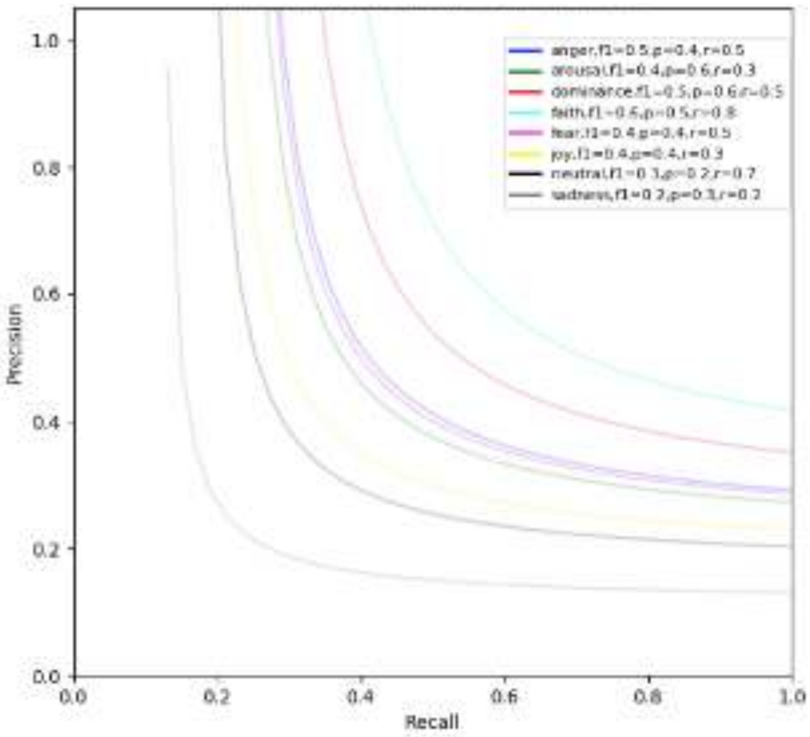


Figure 4.4: Precision recall curve for Congress: Decision tree classifier

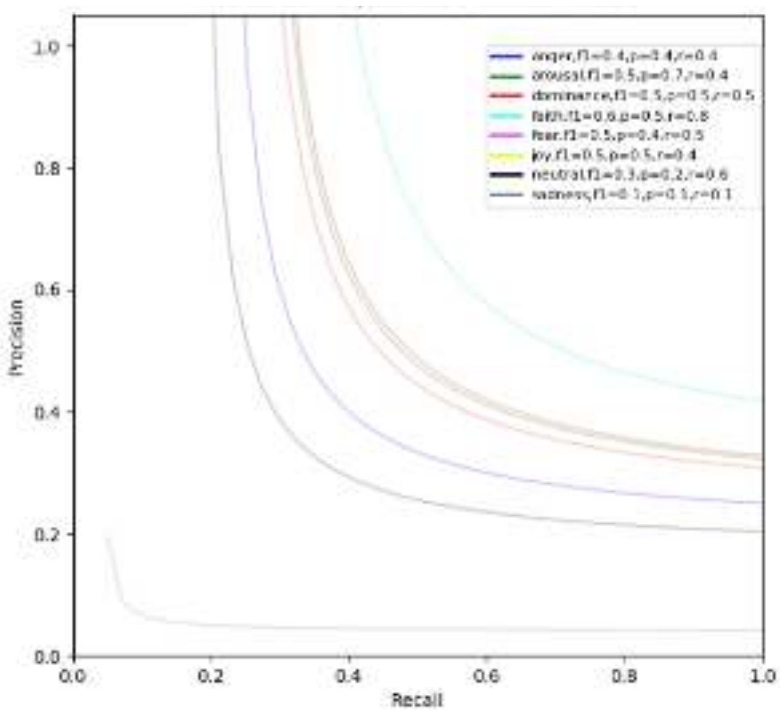


Figure 4.5: Precision recall curve for Congress: TfIdf with Decision tree classifier and countVector

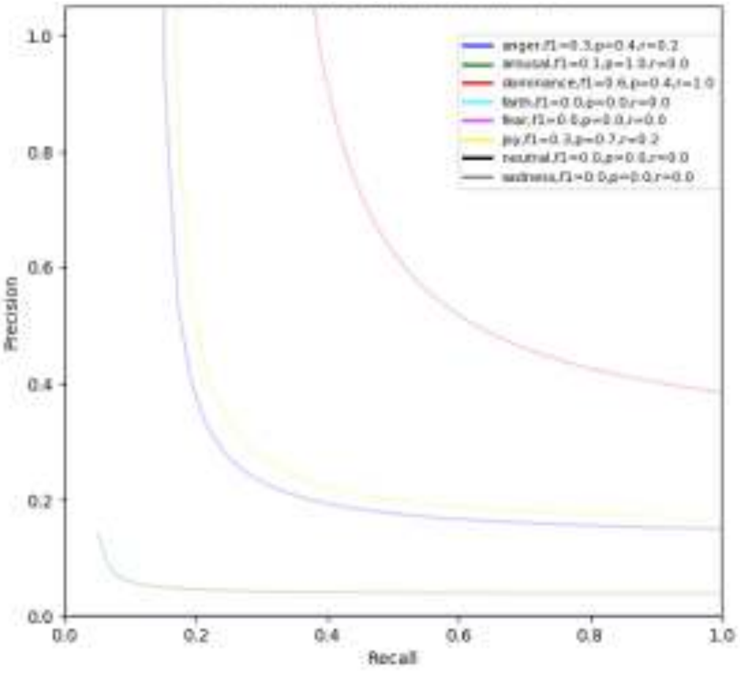
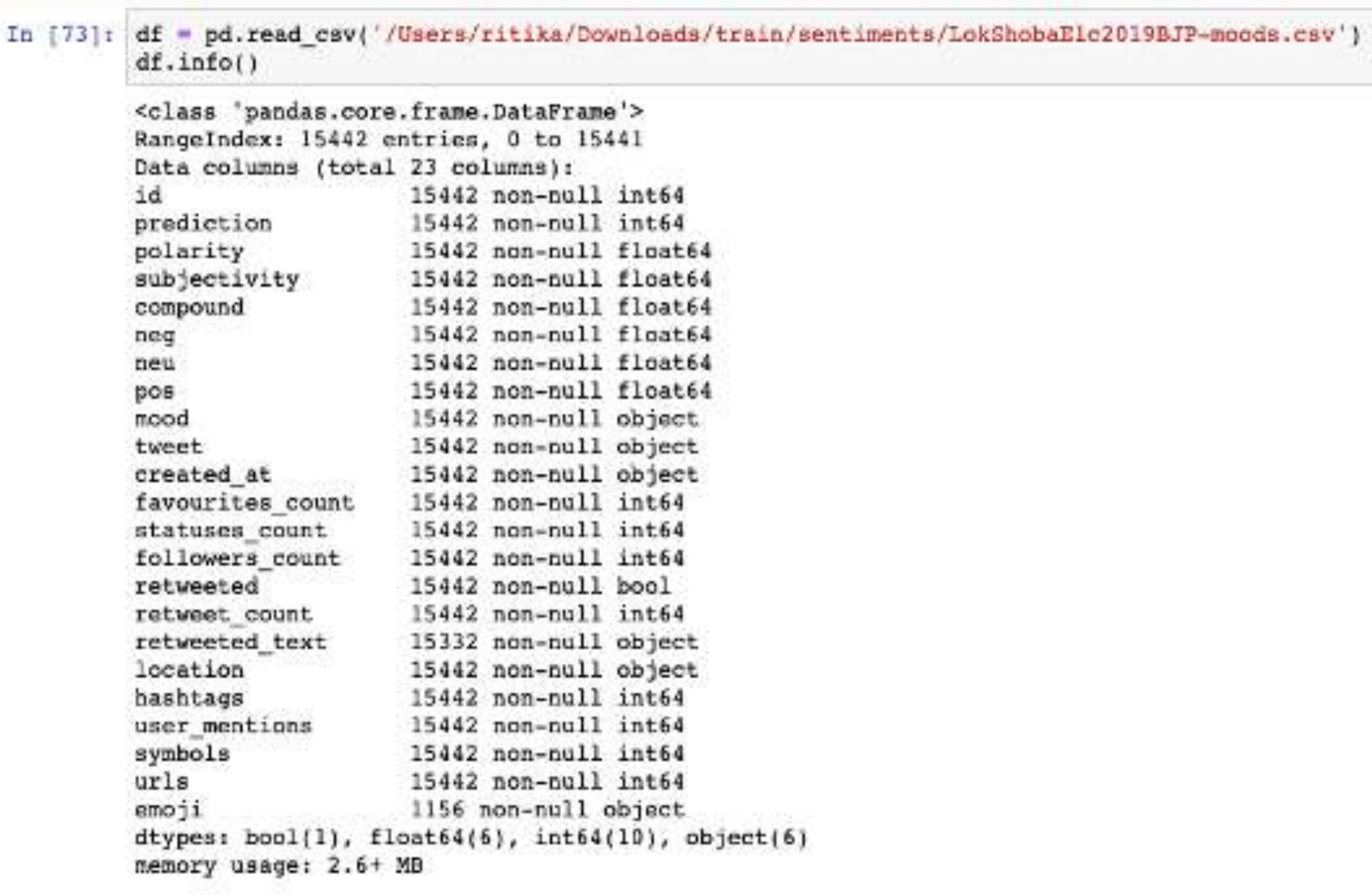


Figure 4.6: Precision recall curve for Congress: TfIdf with Decision tree classifier

MULTI-CLASS TEXT CLASSIFICATION WITH LSTM

Automatic text classification or report arrangement should be possible from multiple points of view in AI as we have seen previously.

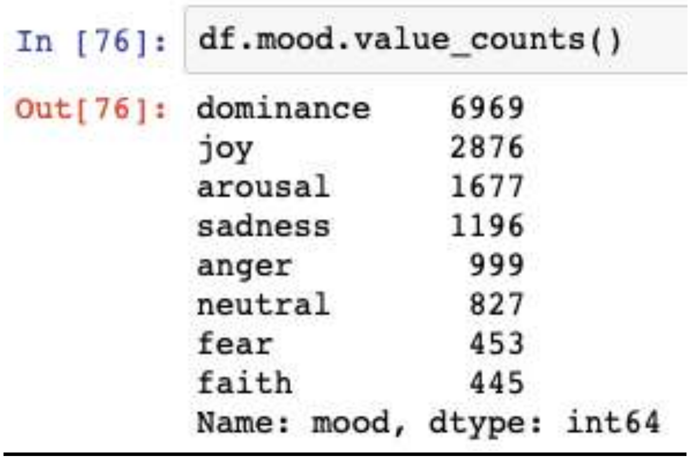
In this it is shown how a Recurrent Neural Network (RNN) utilizing the Long Short Term Memory (LSTM) design can be executed utilizing Keras.



```
In [73]: df = pd.read_csv('/Users/ritika/Downloads/train/sentiments/LokShobaElc2019BJP-moods.csv')
         df.info()
```

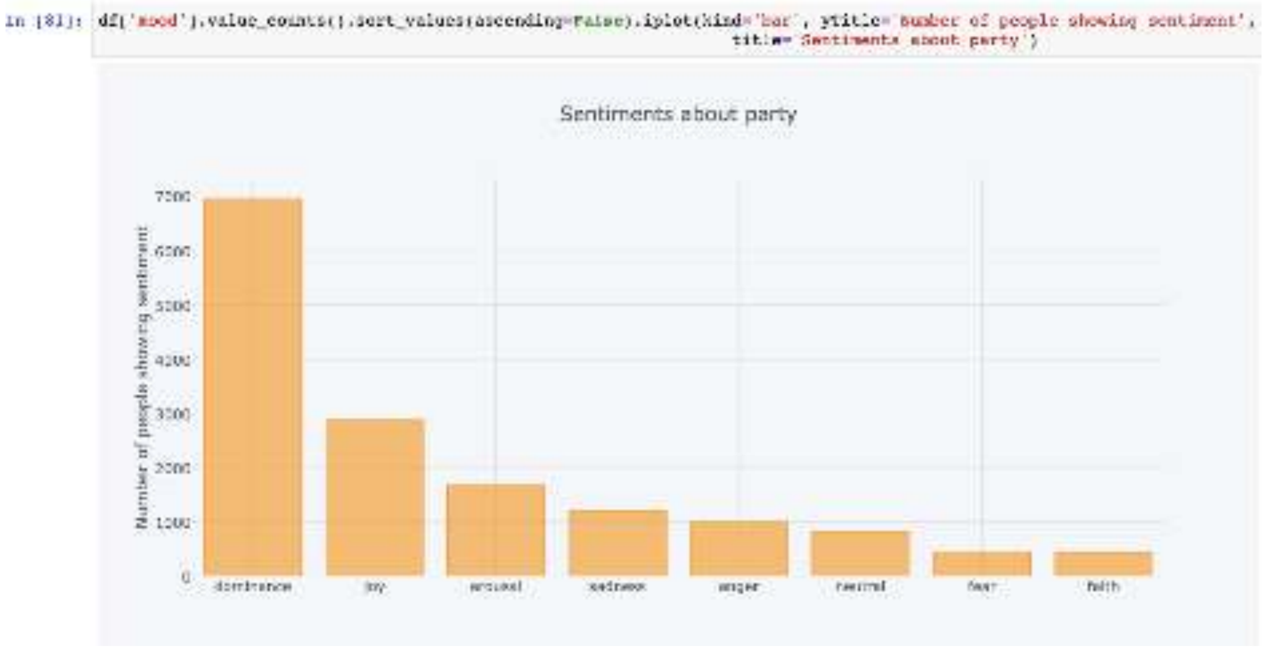
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15442 entries, 0 to 15441
Data columns (total 23 columns):
id                  15442 non-null int64
prediction          15442 non-null int64
polarity            15442 non-null float64
subjectivity        15442 non-null float64
compound            15442 non-null float64
neg                 15442 non-null float64
neu                 15442 non-null float64
pos                 15442 non-null float64
mood                15442 non-null object
tweet               15442 non-null object
created_at          15442 non-null object
favourites_count    15442 non-null int64
statuses_count      15442 non-null int64
followers_count     15442 non-null int64
retweeted           15442 non-null bool
retweet_count       15442 non-null int64
retweeted_text      15332 non-null object
location            15442 non-null object
hashtags            15442 non-null int64
user_mentions       15442 non-null int64
symbols             15442 non-null int64
urls                15442 non-null int64
emoji               1156 non-null object
dtypes: bool(1), float64(6), int64(10), object(6)
memory usage: 2.6+ MB
```

LABEL CONSOLIDATION



```
In [76]: df.mood.value_counts()

Out[76]: dominance    6969
         joy          2876
         arousal      1677
         sadness      1196
         anger         999
         neutral       827
         fear          453
         faith         445
         Name: mood, dtype: int64
```

Table 5.1: Value of occurrence of moods (BJP)



```
In [81]: df['mood'].value_counts().sort_values(ascending=False).iplot(kind='bar', ytitle='Number of people showing sentiment',
                                                                      title='Sentiments about party')
```

Graph 5.1: Sentiment of public towards BJP

LSTM MODELING

- Vectorize tweet content, by transforming every content into either an arrangement of numbers or into a vector.
- Point of confinement the informational collection to the best 5,0000 words.
- Set the maximum number of words in each tweet at 250.

```
In [91]: def print_plot(index):
             example = df[df.index == index][['tweet', 'mood']].values[0]
             if len(example) > 0:
                 print(example[0])
                 print('mood:', example[1])
         print_plot(1)

         opposition statehood delhi confession lied elections
         mood: anger
```

```
In [89]: print_plot(106)

         president amit shah party's election incharge piyush goyal visit today
         mood: dominance
```

```
In [92]: # The maximum number of words to be used. (most frequent)
         MAX_NB_WORDS = 50000
         # Max number of words in each complaint.
         MAX_SEQUENCE_LENGTH = 250
         # This is fixed.
         EMBEDDING_DIM = 100
         tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~', lower=True)
         tokenizer.fit_on_texts(df['tweet'].values)
         word_index = tokenizer.word_index
         print('Found %s unique tokens.' % len(word_index))

         Found 16272 unique tokens.
```

```
In [93]: X = tokenizer.texts_to_sequences(df['tweet'].values)
         X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
         print('Shape of data tensor:', X.shape)

         Shape of data tensor: (15442, 250)
```

```
In [95]: Y = pd.get_dummies(df['mood']).values
         print('Shape of label tensor:', Y.shape)

         Shape of label tensor: (15442, 8)
```

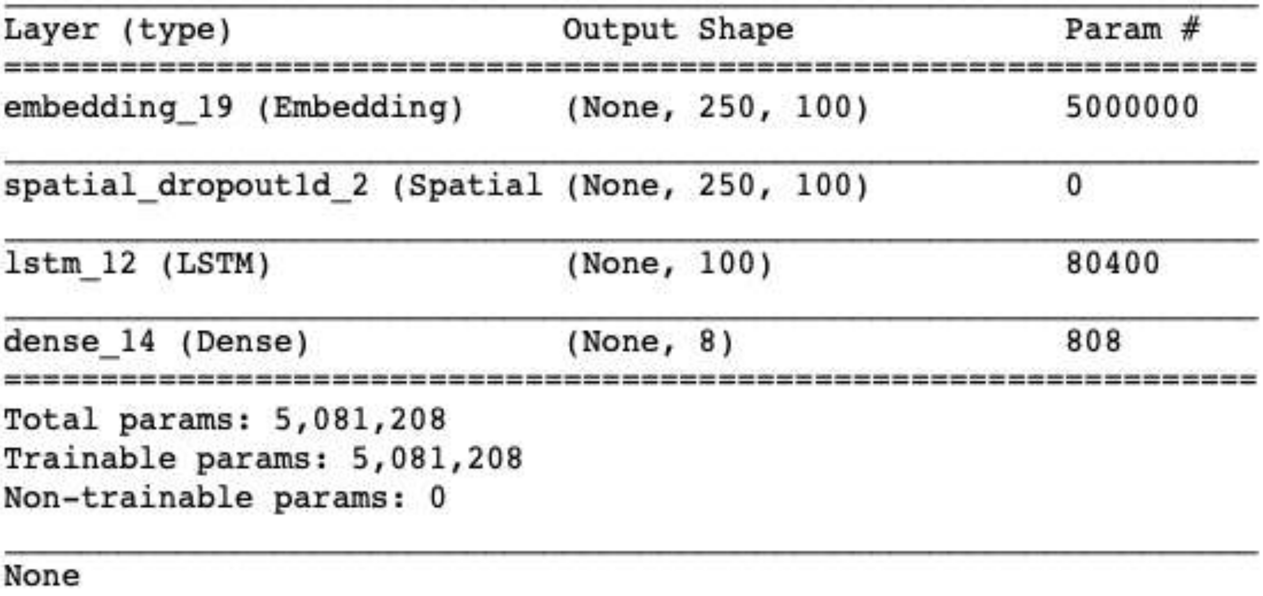
```
In [96]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10, random_state = 42)
         print(X_train.shape,Y_train.shape)
         print(X_test.shape,Y_test.shape)
```

TRAIN TEST SPLIT

```
In [96]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10, random_state = 42)
         print(X_train.shape,Y_train.shape)
         print(X_test.shape,Y_test.shape)

         (13897, 250) (13897, 8)
         (1545, 250) (1545, 8)
```

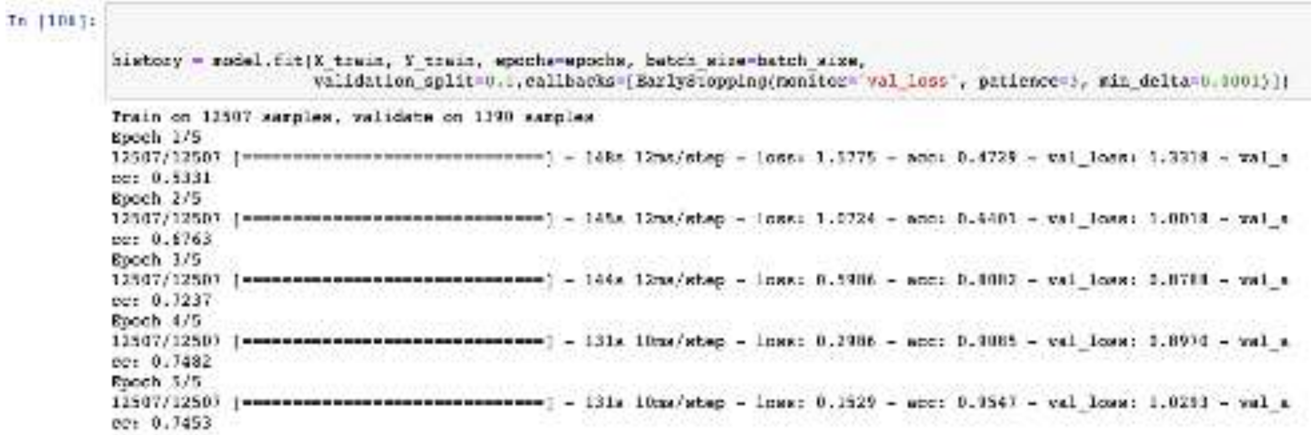
```
In [101]: import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          import seaborn as sns
          from keras.preprocessing.text import Tokenizer
          from keras.preprocessing.sequence import pad_sequences
          from keras.models import Sequential
          from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
          from sklearn.model_selection import train_test_split
          from keras.utils.np_utils import to_categorical
          from keras.callbacks import EarlyStopping
          from keras.layers import Dropout
          import re
          from nltk.corpus import stopwords
          from nltk import word_tokenize
          STOPWORDS = set(stopwords.words('english'))
          from bs4 import BeautifulSoup
          import plotly.graph_objs as go
          import plotly.plotly as py
          import cufflinks
          from IPython.core.interactiveshell import InteractiveShell
          import plotly.figure_factory as ff
          InteractiveShell.ast_node_interactivity = 'all'
          from plotly.offline import iplot
          cufflinks.go_offline()
          cufflinks.set_config_file(world_readable=True, theme='pearl')
```

```
In [105]: model = Sequential()
          model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
          model.add(SpatialDropout1D(0.2))
          model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
          model.add(Dense(8, activation='softmax'))
          model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
          print(model.summary())
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_19 (Embedding)     (None, 250, 100)          5000000
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 250, 100)          0
_________________________________________________________________
lstm_12 (LSTM)               (None, 100)               80400
_________________________________________________________________
dense_14 (Dense)             (None, 8)                 808
=================================================================
Total params: 5,081,208
Trainable params: 5,081,208
Non-trainable params: 0
_________________________________________________________________
None
```

- The primary layer is the inserted layer that utilizes 100 length vectors to speak to each word.
- SpatialDropout1D performs variational dropout in NLP models.
- The following layer is the LSTM layer with 100 memory units.
- The output layer must make 8 yield qualities, one for each class.
- Actuation work is SoftMax for multi-class characterization.
- Since it is a multi-class order issue, categorical_crossentropy is utilized as the misfortune work.



```
In [108]:

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,
                    validation_split=0.1,callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
Train on 12507 samples, validate on 1390 samples
Epoch 1/5
12507/12507 [==============================] - 148s 12ms/step - loss: 1.5775 - acc: 0.4729 - val_loss: 1.3318 - val_a
cc: 0.5331
Epoch 2/5
12507/12507 [==============================] - 145s 12ms/step - loss: 1.0724 - acc: 0.6401 - val_loss: 1.0018 - val_a
cc: 0.6763
Epoch 3/5
12507/12507 [==============================] - 144s 12ms/step - loss: 0.5986 - acc: 0.8082 - val_loss: 0.8788 - val_a
cc: 0.7237
Epoch 4/5
12507/12507 [==============================] - 131s 10ms/step - loss: 0.2986 - acc: 0.9085 - val_loss: 0.8930 - val_a
cc: 0.7482
Epoch 5/5
12507/12507 [==============================] - 131s 10ms/step - loss: 0.1529 - acc: 0.9547 - val_loss: 1.0251 - val_a
cc: 0.7453
```

Table 6.1: Training model for epochs value 6

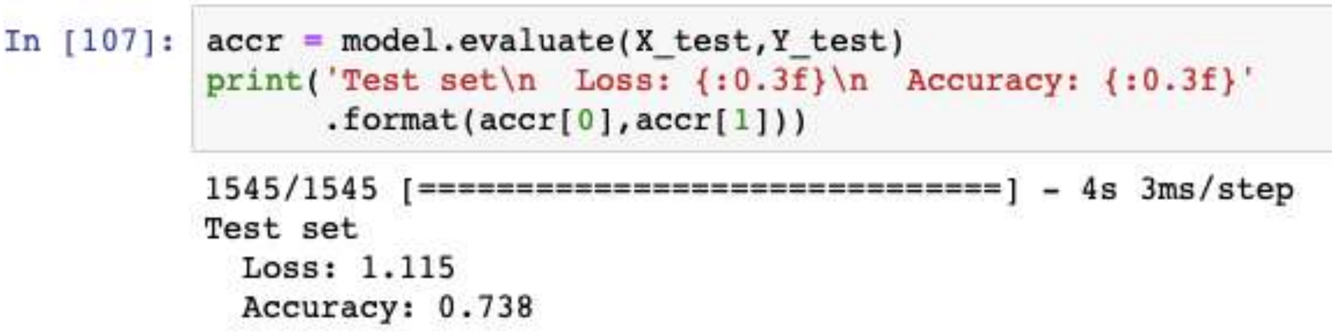
The primary layer is the Embedded layer that utilizes 32 length vectors to speak to each word. The following layer is the LSTM layer with 100 memory units (smart neurons). At long last, since this is a grouping issue we utilize a Dense yield layer with a solitary neuron and a sigmoid enactment capacity to make 0 or 1 expectations for the two classes (great and terrible) in the issue.

Since it is a binary classification issue, log loss is utilized as the loss function (binary_crossentropy in Keras). The proficient ADAM optimization algorithm enhancement calculation is utilized. The model rapidly overfits the issue. A huge cluster size of 64 surveys is utilized to space out weight refreshes.

We can see dropout having the ideal effect on preparing with a marginally slower pattern in combination and for this situation a lower last precision. The model could most likely utilize a couple of more ages of preparing and may accomplish a higher ability (attempt it a see).

On the other hand, dropout can be connected to the info and intermittent associations of the memory units with the LSTM definitely and independently.

Keras gives this capacity parameters on the LSTM layer, the dropout for designing the info dropout and recurrent_dropout for arranging the repetitive dropout. For instance, we can alter the primary guide to add dropout to the info and intermittent associations as pursues:



```
In [107]: accr = model.evaluate(X_test,Y_test)
          print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'
                .format(accr[0],accr[1]))

          1545/1545 [==============================] - 4s 3ms/step
          Test set
            Loss: 1.115
            Accuracy: 0.738
```

MAPPING OF ACCURACY AND DATA LOSS

```
In [108]: plt.title('Loss')
          plt.plot(history.history['loss'], label='train')
          plt.plot(history.history['val_loss'], label='test')
          plt.legend()
          plt.show();
```

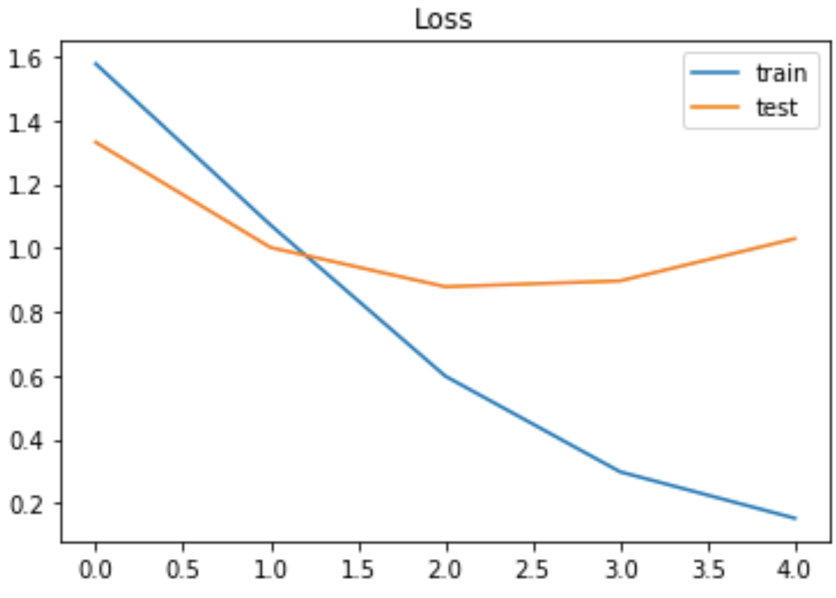


Figure 6.1: Loss depiction on training v/s testing model

```
In [109]: plt.title('Accuracy')
          plt.plot(history.history['acc'], label='train')
          plt.plot(history.history['val_acc'], label='test')
          plt.legend()
          plt.show();
```

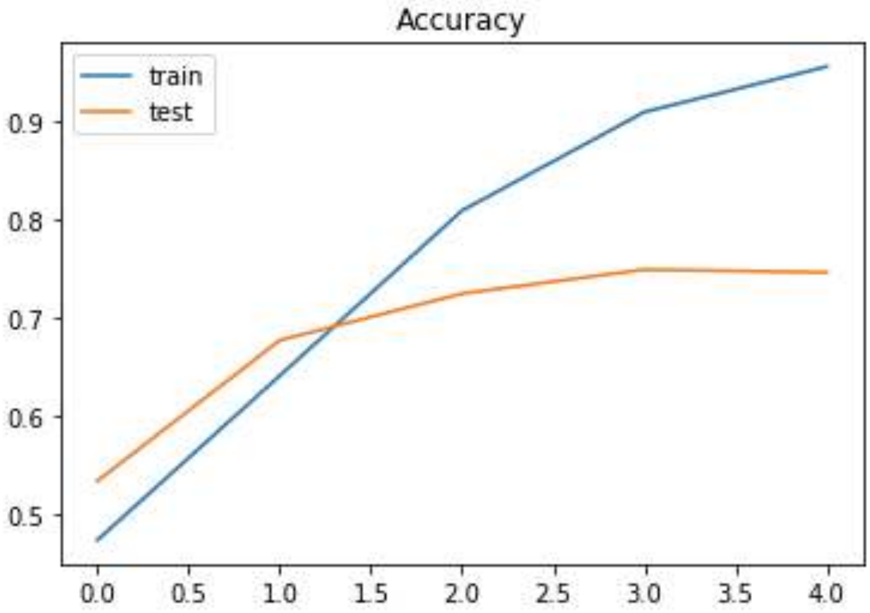
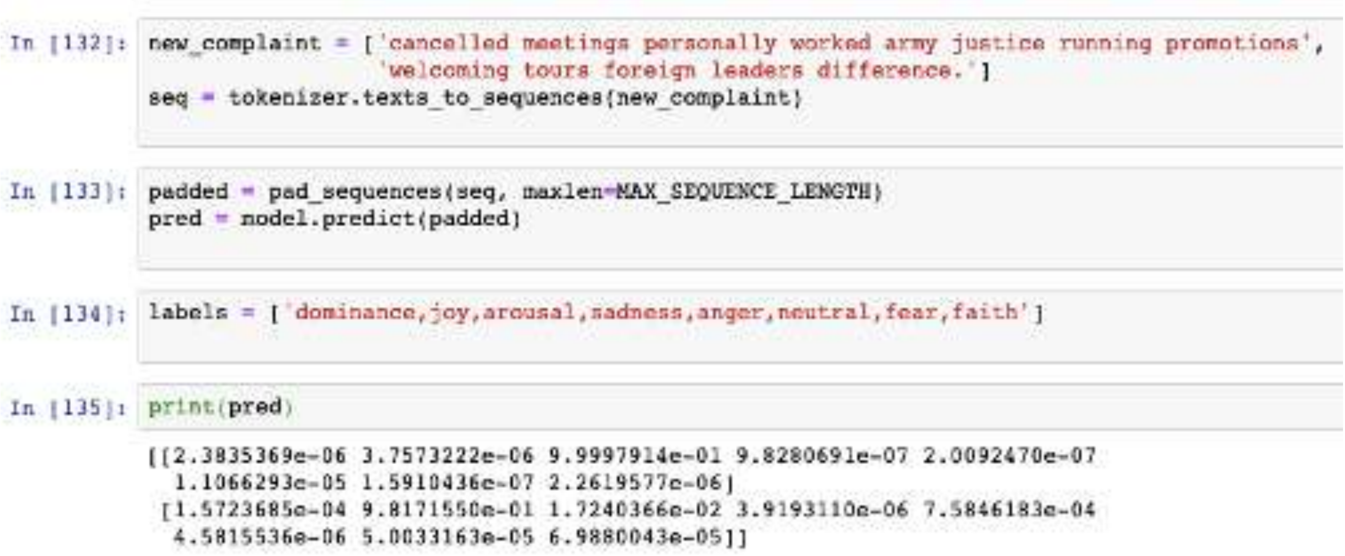


Figure 6.2: Accuracy depiction on training v/s testing model

The plots propose that the model has a little over fitting issue, more information may help, yet more epochs won't with the present data.

TEST WITH A NEW COMPLAINT



```
In [132]: new_complaint = ['cancelled meetings personally worked army justice running promotions',
                            'welcoming tours foreign leaders difference.']
          seq = tokenizer.texts_to_sequences(new_complaint)

In [133]: padded = pad_sequences(seq, maxlen=MAX_SEQUENCE_LENGTH)
          pred = model.predict(padded)

In [134]: labels = ['dominance,joy,arousal,sadness,anger,neutral,fear,faith']

In [135]: print(pred)

          [[2.3835369e-06 3.7573222e-06 9.9997914e-01 9.8280691e-07 2.0092470e-07
            1.1066293e-05 1.5910436e-07 2.2619577e-06]
           [1.5723685e-04 9.8171550e-01 1.7240366e-02 3.9193110e-06 7.5846183e-04
            4.5815536e-06 5.0033163e-05 6.9880043e-05]]
```

The model works fine for new tested results.

Prediction and keywords generated correctly.

```
In [120]: new_positive_tweets = pd.Series(["nda achieved sustained growth low inflation"
                                           ,"healthy india making"
                                           ,"president exudes confidence together win seats"])

          df_counts_pos = tc.transform(new_positive_tweets)
          df_clean_pos = ct.transform(new_positive_tweets)
          df_model_pos = df_counts_pos
          df_model_pos['clean_text'] = df_clean_pos

          best_model.predict(df_model_pos).tolist()

Out[120]: ['joy', 'joy', 'joy']
```

REGRESSION ALGORITHMS

RIDGE REGRESSION

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

# generate random data-set
#np.random.seed(0)
#x = np.random.rand(100,2)
#y = 2 + 3*x + np.random.rand(100,2)
#print(x)

df = pd.read_csv('C:\\Users\\ADMIN\\Desktop\\eci_data_1951_2019\\voter.csv')

#type(df['Electors'])
x = np.reshape(df['Electors'].values,(len(df), 1))
y = np.reshape(df['Voters'].values, (len(df),1))
# sckit-learn implementation
print(x)
# Model initialization
#regression_model = LinearRegression()

#regression_model = LinearRegression()

regression_model = Ridge(alpha=10.5)
#Fit the data(train the model)
regression_model.fit(x, y)
# Predict
y_predicted = regression_model.predict(x)

# model evaluation
rmse = mean_squared_error(y, y_predicted)
r2 = r2_score(y, y_predicted)

# printing values
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)

# plotting values

# data points
plt.scatter(x, y, s=10)
plt.xlabel('x')
plt.ylabel('y')

# predicted values
plt.plot(x, y_predicted, color='r')
plt.show()
```

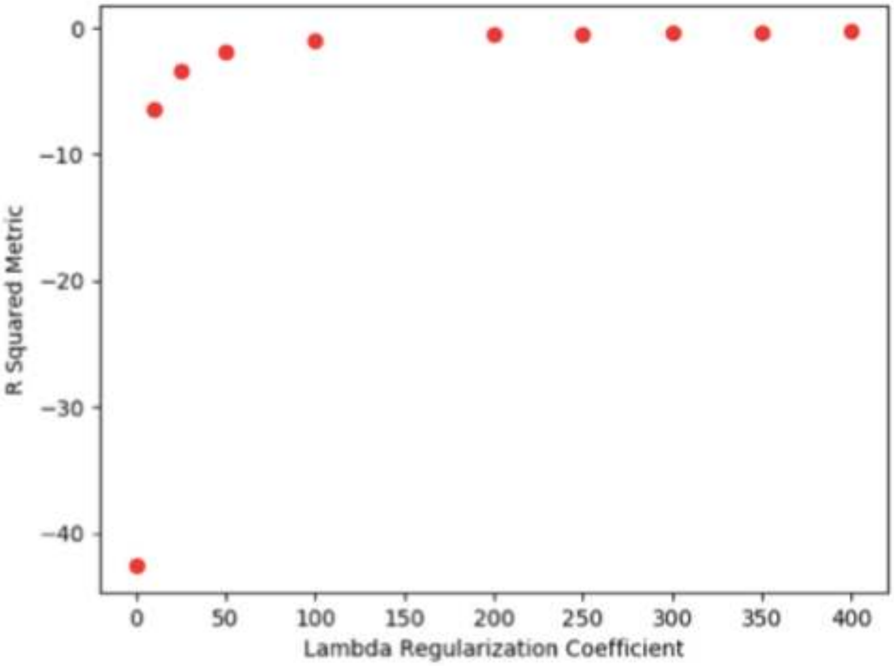


Figure 7.1: Ridge regression on unclean data

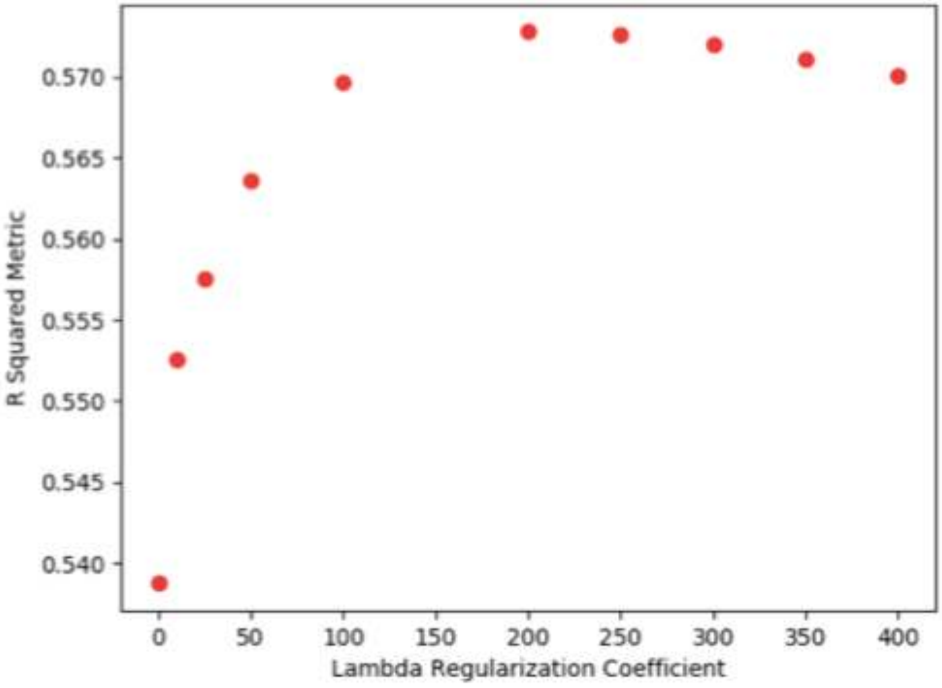


Figure 7.2: Ridge regression on clean data

SIMPLE LINEAR REGRESSION MODEL AND LASSO REGRESSION

We presently depict the procedure through which we directed regression on our data. We outline the regression issue as an endeavor to utilize the previously mentioned statistic information to anticipate the portion of voters who pick candidates by region. We perceive promptly that we have couple of precedents relative to the quantity of highlights in our model. For the 2014 race, we have 820 precedents and 897 highlights. Utilizing the information cleaning strategy portrayed in segment 2, we can decrease the quantity of highlights in our model from 897 to 188. We speculate that utilizing LASSO relapse would suit our concern well, the same number of the highlights are probably going to be futile or excess. LASSO regularization regularizes utilizing the L1 standard, and seeks to optimize the loss function.

$$L = \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \sum_{j=1}^{p} \|\beta_j\|^1$$

All in all, we see that LASSO relapse boundlessly beats Ridge relapse on the uncleaned information. After manually cleaning the data, we find that Ridge regression and LASSO regression perform similarly, with Ridge relapse giving more consistent outcomes which are correspondingly less touchy to changes in the esteem λ. This recommends a little measure of highlights from our unique informational collection are sufficient to describe our model to define decision expectations giving election predictions.

All the more by and large, our outcomes here likewise recommend that when the information is too intricate to even consider being physically cleaned and just restricted information are accessible, LASSO regression can possibly be a valuable apparatus for distinguishing superfluous or repetitive highlights.

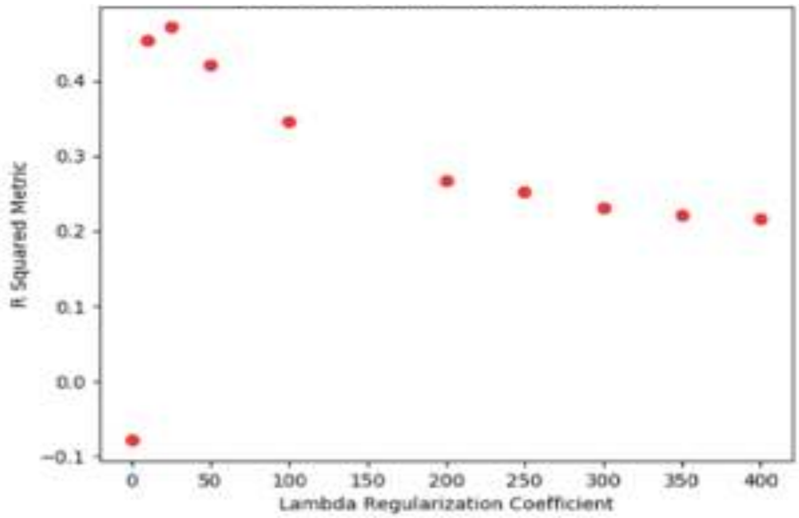


Figure 7.3: LASSO regression on unclean data

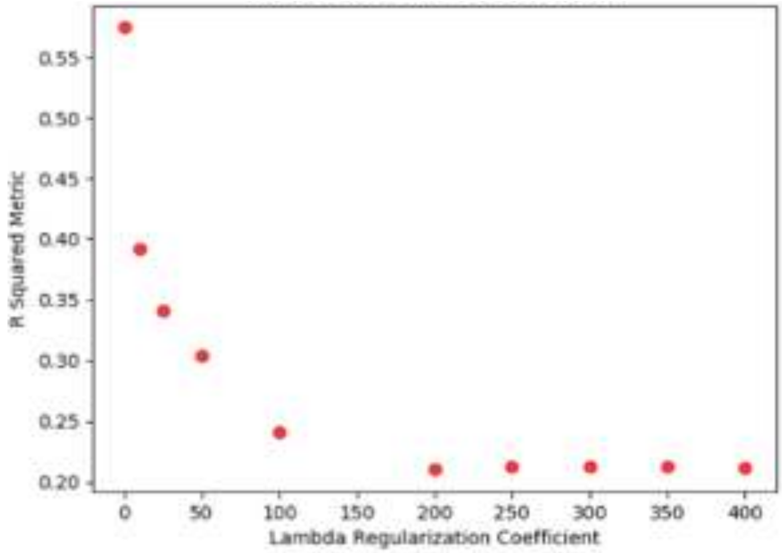
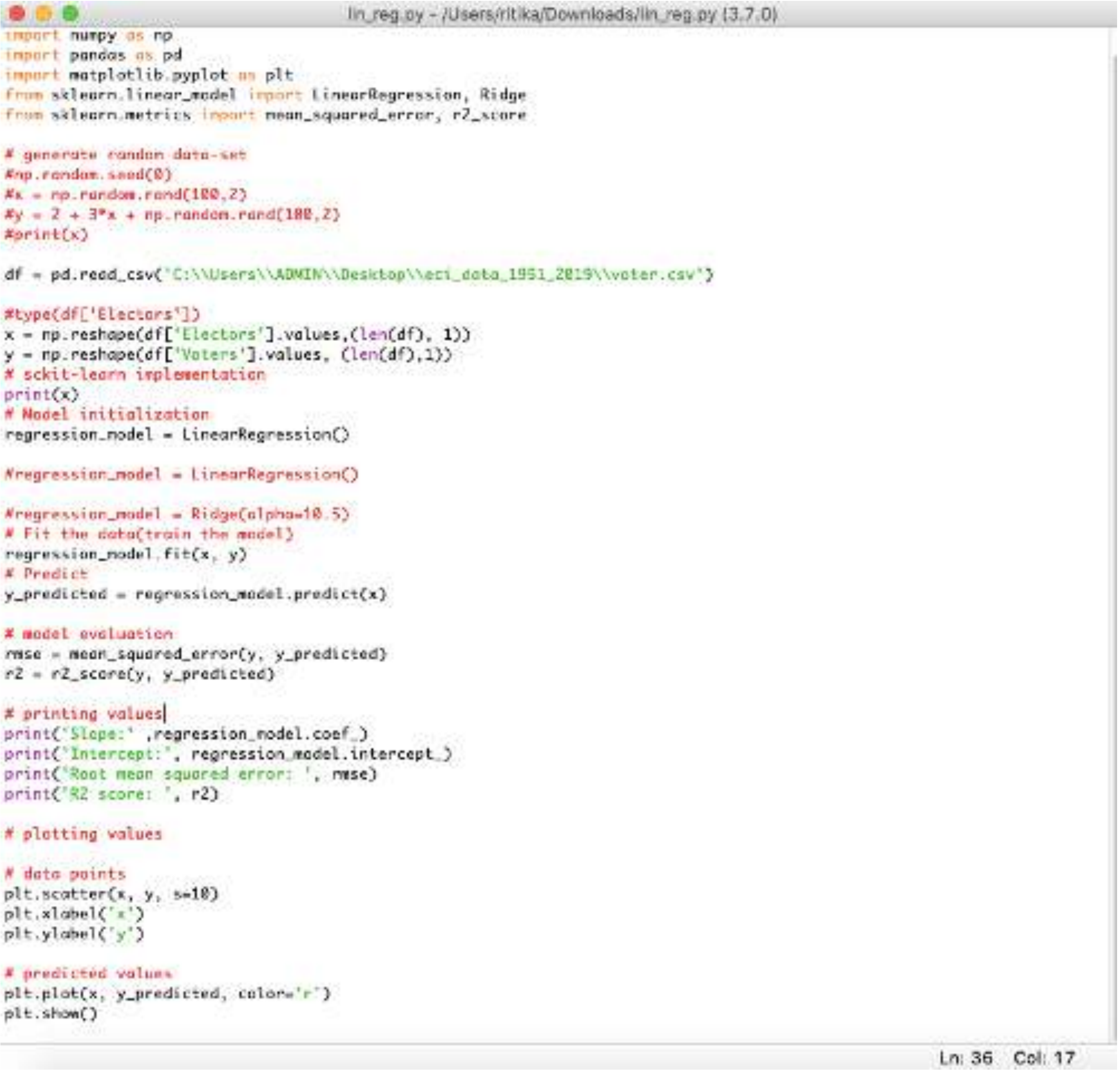


Figure 7.4: LASSO regression on clean data



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error, r2_score

# generate random data-set
#np.random.seed(0)
#x = np.random.rand(100,2)
#y = 2 + 3*x + np.random.rand(100,2)
#print(x)

df = pd.read_csv('C:\\Users\\ADMIN\\Desktop\\eci_data_1951_2019\\voter.csv')

#type(df['Electors'])
x = np.reshape(df['Electors'].values,(len(df), 1))
y = np.reshape(df['Voters'].values, (len(df),1))
# sckit-learn implementation
print(x)
# Model initialization
regression_model = LinearRegression()

#regression_model = LinearRegression()

#regression_model = Ridge(alpha=10.5)
# Fit the data(train the model)
regression_model.fit(x, y)
# Predict
y_predicted = regression_model.predict(x)

# model evaluation
rmse = mean_squared_error(y, y_predicted)
r2 = r2_score(y, y_predicted)

# printing values
print('Slope:' ,regression_model.coef_)
print('Intercept:', regression_model.intercept_)
print('Root mean squared error: ', rmse)
print('R2 score: ', r2)

# plotting values

# data points
plt.scatter(x, y, s=10)
plt.xlabel('x')
plt.ylabel('y')

# predicted values
plt.plot(x, y_predicted, color='r')
plt.show()
```

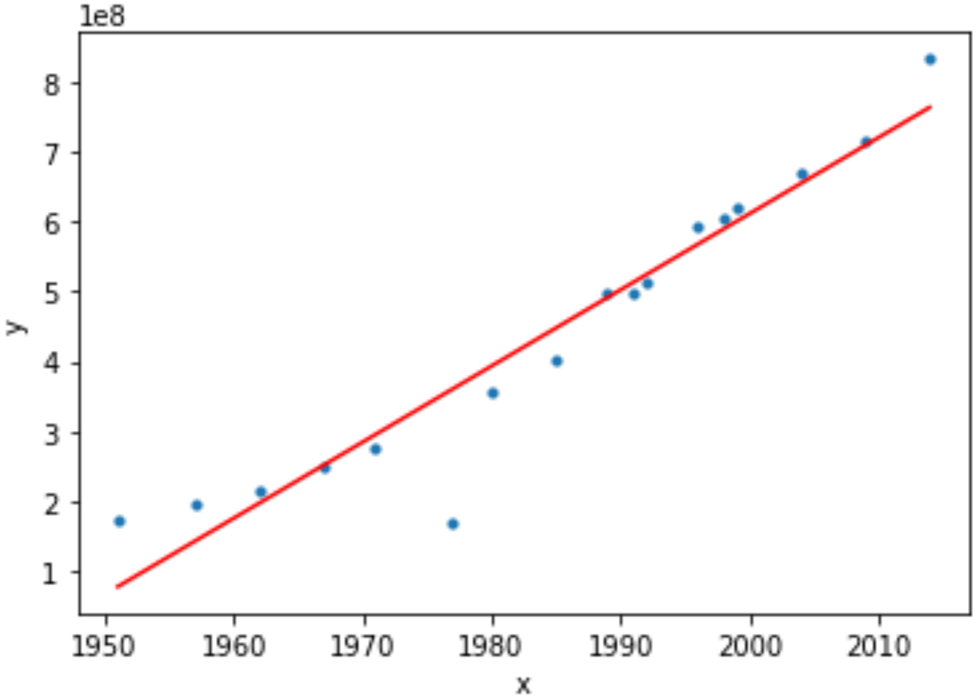


Figure 7.5: Linear regression on unclean data

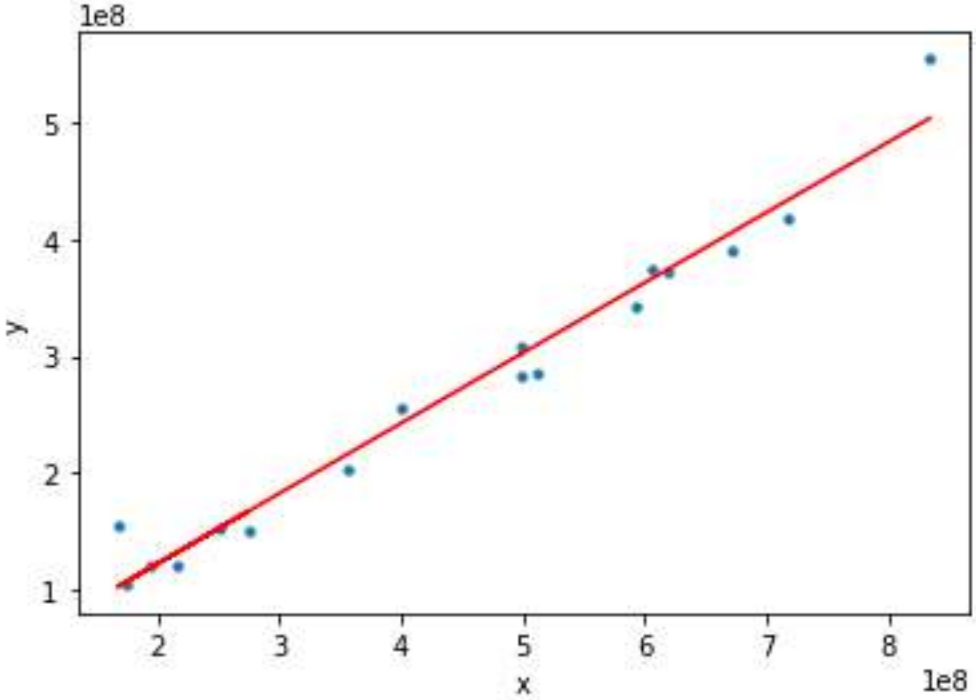
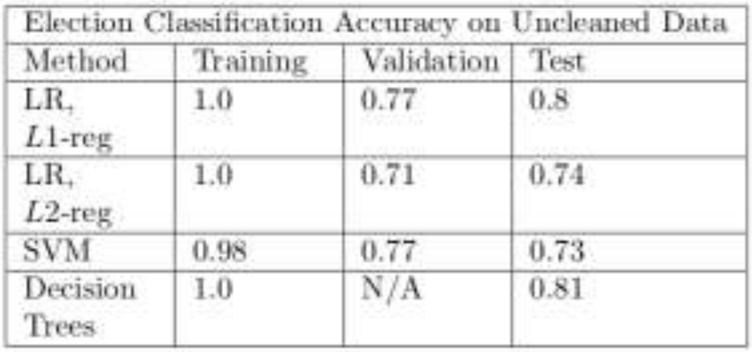


Figure 7.6: Linear regression on clean data

CLASSIFICATION ALGORITHMS

We currently outline our undertaking as classification problem. All the more explicitly, we utilize similar data to prepare and test our models, characterizing a positive guide to be one where more prominent than the greater part of the voters pick an INC candidate, and a negative guide to be one where not exactly 50% of the voters pick a BJP competitor. To tackle the characterization issue, we utilize calculated logistic regression (with both L1 and L2 regularization), a delicate edge SVM classifier, and decision trees. We play out these methods on both the cleaned and uncleaned variants of our information to determine the nearness of analysis like what we saw in segment 3.1. Here, we save 720 precedents for preparing and 100 case for testing. On account of calculated regression and the SVM, we utilize 120 of the 720 preparing precedents as an approval set, to pick λ (regularization coefficient) and C (punishment on blunder term) separately. To create our outcomes, we pick regularization quality for strategic relapse and SVM by tuning regularization parameters (λ and C respectively) through matrix look, measuring our exhibition on the approval set. In Tab. 1 and Tab. 2, we show the characterization exactnesses we accomplish on the uncleaned and cleaned information, separately.

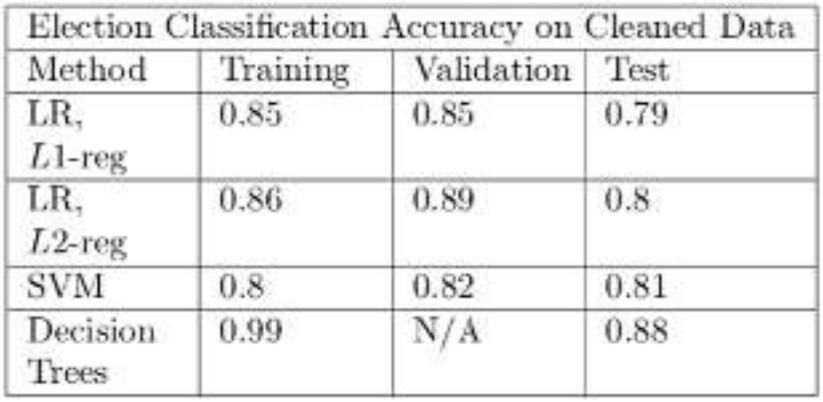


Election Classification Accuracy on Uncleaned Data

Method	Training	Validation	Test
LR, $L1$-reg	1.0	0.77	0.8
LR, $L2$-reg	1.0	0.71	0.74
SVM	0.98	0.77	0.73
Decision Trees	1.0	N/A	0.81

Table 8.1: Characterization exactnesses: uncleaned data

Table: Order exactness or Classification accuracy on uncleaned information of PCs casting a ballot GOP utilizing calculated logistic regression, SVM classifiers, and decision trees. We utilized a matrix inquiry and execution on a held-out approval set to discover ideal hyperparameters



Election Classification Accuracy on Cleaned Data			
Method	Training	Validation	Test
LR, $L1$-reg	0.85	0.85	0.79
LR, $L2$-reg	0.86	0.89	0.8
SVM	0.8	0.82	0.81
Decision Trees	0.99	N/A	0.88

Table 8.2: Characterization exactnesses: cleaned data

Table: Characterization precision of Classification accuracy on cleaned information of areas casting a ballot GOP utilizing strategic logistic regression, SVM classifiers, and choice trees. We utilized a matrix pursuit and execution on a held-out approval set to discover ideal hyperparameters.

Our work with classification arrangement yields a few interesting results. Outstandingly, we can acquire a 0.88 grouping exactness on inconspicuous test information with generally equivalent extents of positive and negative examples. Plainly, we can separate well between PCs that vote GOP, and regions that don't, founded exclusively on statistic data. Further, we see that while the two regularization procedures produce comparative results on the cleaned information, L1 regularization immeasurably beats L2 regularization on the un-cleaned informational indexes. This outcome again proposes that a large portion of the information we expelled in cleaning was immaterial, just prompting a pointlessly unpredictable model. As L1 regularization drives singular parameters to 0 not at all like L2 regularization, L1 regularization adequately pruned good for nothing highlights when preparing. Besides, we see that decision trees yield the best test precision of the order techniques tried. The decision trees result in an inadequate arrangement, which can be checked by just a couple of highlights having high "significant" in our model. The XGBoost (Chen and Guestrin, 2016) "significance score" is a proportion of the data increase of a specific component arrived at the midpoint of over the trees in a helped model. Rotating on just a couple of unmistakable highlights with high data gain allowed our supported decision trees to perceive the most significant highlights and sum up well.

CLUSTERING

To check whether our information contained any normal segmentation that could be learned, we test a Gaussian Mixture Model on our data. We utilize the sci-kit (Pedregosa et al., 2011) learn package and used the EM algorithm to fit k clusters to our data. This segment will talk about how these outcomes coordinated different strategies.

We first train exclusively on 2014 statistic data. Utilizing our cleaned dataset, for instance information that does not utilize numerous sections, and so forth, we can accomplish combination utilizing k = 2 to k = 8 groups. Since our information contains a couple of exceptions as far as populace, pay, or different variables, we regularly watch a couple of groups that catch these few points.

There are some significant insights for our bunches utilizing a couple of various number of groups. Outcomes are plotted along the two most significant highlights.

Presently we might want to break down the consequences of our simplest and most robust clustering, that is k = 3. It is exceptionally obvious to see that our clusters are for the most part isolated along populace lines. There are a lot of rustic regions that have exceptionally low populace and differing ages. These districts will in general vote BJP generally 70% of the time, which matches with regular instinct that country territories are republican inclining.

This plot likewise confirms our instinct that substantial urban focuses vote only BJP. The little cluster relating to these extensive provinces have no INC casting a ballot region. Our fair measured group, which relates to rural regions with marginally higher salary all things considered than the other two, likewise will in general vote democrat however not only. With generally 10% of these areas casting a ballot BJP, this matches with our instinct and our outcomes from different strategies.

Clustering could be utilized to produce a decision outcome. To do this, we could characterize a generative procedure that utilizes our blended Gaussian Model. Accepting delicate cluster assignments characterized by:

$$P(y = i|x) = \frac{p(x|y = i)p(y = i)}{\sum_i p(x|y = i)p(y = i)}$$
$$= \mathcal{N}(x; \mu_i, \sigma_i^2)\pi_i/\mathcal{Z},$$

where Z speaks to the normalization constant steady, πi describes the likelihood of being in each cluster. Presently given another PC, we might want to foresee whether a district cast a BJP or INC vote. First register the soft cluster assignments for another region, and after that utilization a binomial procedure for each cluster to appoint BJP or INC. We document this utilizing V to speak to the clear-cut categorical vote variable, with R for Independent candidates

$$P(V = R|x) = \sum_i p(V = R|y = i, x) * p(y = i|x)$$

This generative procedure could be utilized to anticipate a future election given current statistic information.

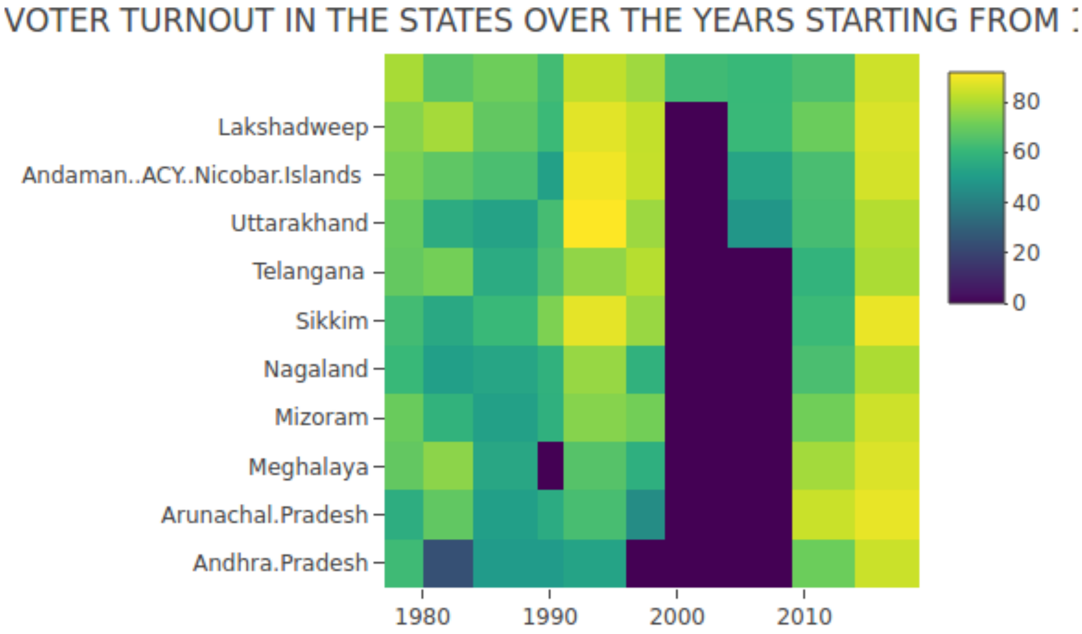
ADVANCEMENT OF AI IN POLITICS

The greatest choice the triumphant political party made was to move far from the customary way to deal with voters; which is to send only one communicated message to the entire nation, covering everybody, paying little mind to age, sexual orientation, area or some other key differentiator. Utilizing Big Data, the winning candidate had the option to draw nearer to the intended interest group, improving commitment and discussing the definite things that was imperative to the voter.

A smarter methodology is to send messages that is significant to individuals, instead of worn-out updates that do nothing as well as simply report the campaigners. Big Data analytics enables the ideological groups to become more acquainted with their voters on an individual dimension. Prevalent innovation and complex instruments help find and comprehend their necessities and the various ways the voters can be drawn nearer. This is a colossal arrangement for the universe of political issues.

ENGAGING VOTERS

Generally, ideological political parties' groups depended on that minority that bolstered them wholeheartedly and relied upon them to voice their adoration on social stages to construct more mindfulness. Presently, with Big Data, a noteworthy number of voters who truly 'don't have the foggiest idea' can be focused on. Give us a chance to call these don't-knows, gliding voters. Machine-based learning can help with more astute focusing of such coasting voters. These voters can likewise be isolated into numerous different classifications, for example, the segment that can be influenced into settling on a choice, another segment that is uncertain and numerous others. In this way, changing over gliding voters can get in votes that generally were never conceivable to get.



Graph 10.1: Heatmap for voting percentage across Phase 1 states: 1951 - 2019

Information investigation instruments can help comprehend which segments are bound to pick and bolster a gathering. Rather than utilizing area as a parameter, Big Data analytics can produce a ton of details, for example, paper inclinations, age, and instruction foundation and look into watchwords in Facebook and Twitter channels too.

VARIOUS CHANNELS TO REACH THE VOTERS

With Big Data, AI and information mining, political parties have an edge over other people who keep concentrating just on conventional models for their battles. This edge is basically interfacing with voters by means of web-based life, a significant channel. Information mining from web-based life stages, for example, Facebook and Twitter enables gatherings to comprehend the worries, issues and feelings voters host towards their get-together and how it could be various inside the various areas of the nation.

It was such data that was utilized by India's political parties to fabricate their system, successfully target voters, enroll volunteers and improve their various channels to achieve the voter at the doorstep to customizing messages via web-based networking media accounts. Before we go there, how about we comprehend what data mining is in any case.

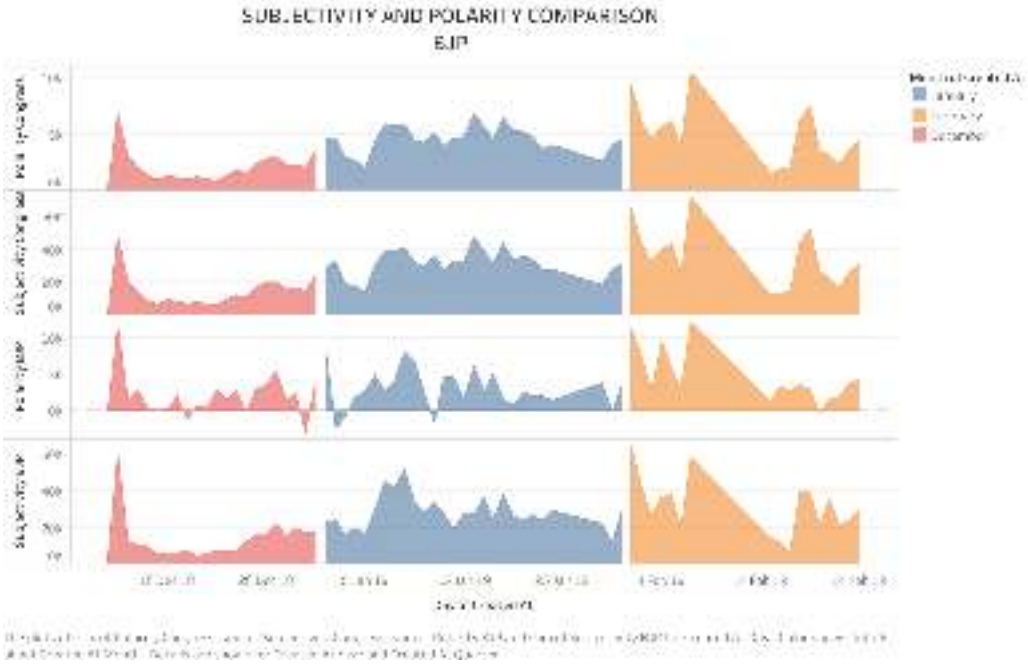
Data Mining is "Filtering through exceptionally a lot of information for helpful data. It uses artificial intelligence techniques, neural systems, and progressed factual apparatuses, (for example, bunch data) to uncover patterns, examples, and connections, which may somehow or another have stayed undetected. Rather than a specialist framework (which draws inferences from the given information based on a lot of principles) information mining endeavors to find concealed guidelines fundamental the information. Additionally, called data surfing, it empowers clients to find examples and bits of knowledge from enormous scale archives.

IDEOLOGICAL POLITICAL GROUPS MOVING AWAY FROM TRADITIONAL MODELS

Being enlivened by effective battles of Obama's Big Data in decisions, even ideological groups in the UK have gradually moved far from the customary models as of late. Both the work and the traditionalist gatherings have put intensely into their computerized mediums; they've likewise procured the equivalent advanced counselors who were a piece of the Obama group!

Data can be utilized research indeed, yet breaking down information from open responses to ideological groups, their crusades, strategies and even reactions in basic circumstances should be done in continuously. Information examination assumes a significant job in changing the final product of any crusade. It is no big surprise that India's own one of a kind Narendra Modi is viewed as a standout amongst the most innovation and online life sagacious government officials on the planet! PM Modi has no under 10k adherents on Twitter, 32 million likes on Facebook and 440 million perspectives on Google+

Modi has a place with the BJP party which won the 2014 races in India with the assistance of open-source computerized devices that put them legitimately in contact with their voters. The majority of the measurements to do that were accomplished utilizing information mining and information investigation to dive into the plenty of web-based life exercises. Indeed, even versatile clients were thought about. Be that as it may, there is no precluding a noteworthy number from claiming voters must be achieved the conventional way - direct human contact.

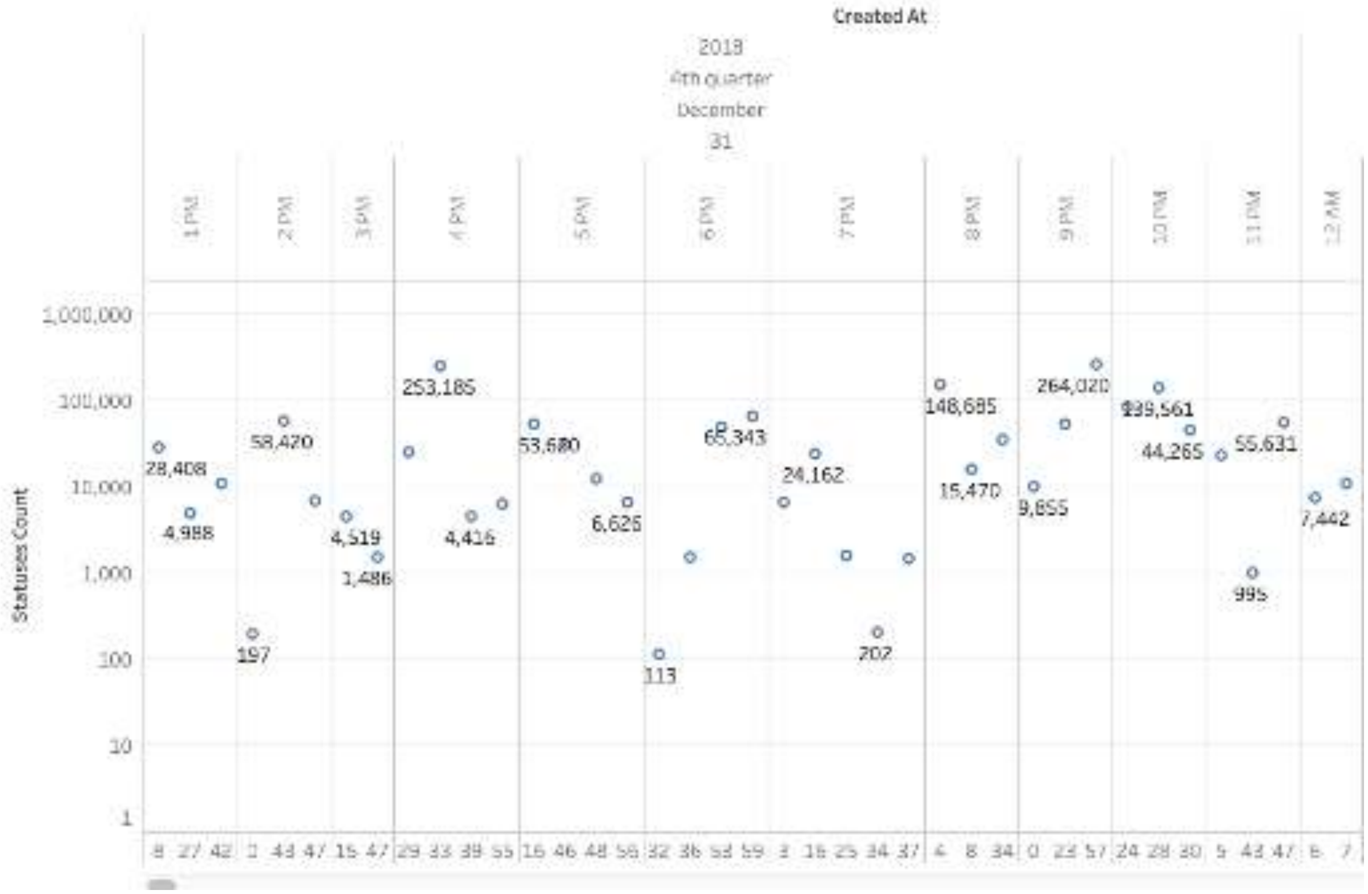


Graph 10.2: Area Chart for voting comparison of polarity and subjectivity between BJP and Congress public opinion

In any case, BJP made brilliant moves to connect with their potential voters utilizing a mix of advanced and conventional channels to enroll volunteers - both on the web and disconnected. Albeit broad communications were the committed channel for the voters certain about voting in favor of BJP, to achieve the gliding voters and even negative voters, correspondence connected at the smaller scale levels on the Internet, portable and web-based life - separated from the conventional road battles.

ELECTION DATA ANALYSIS

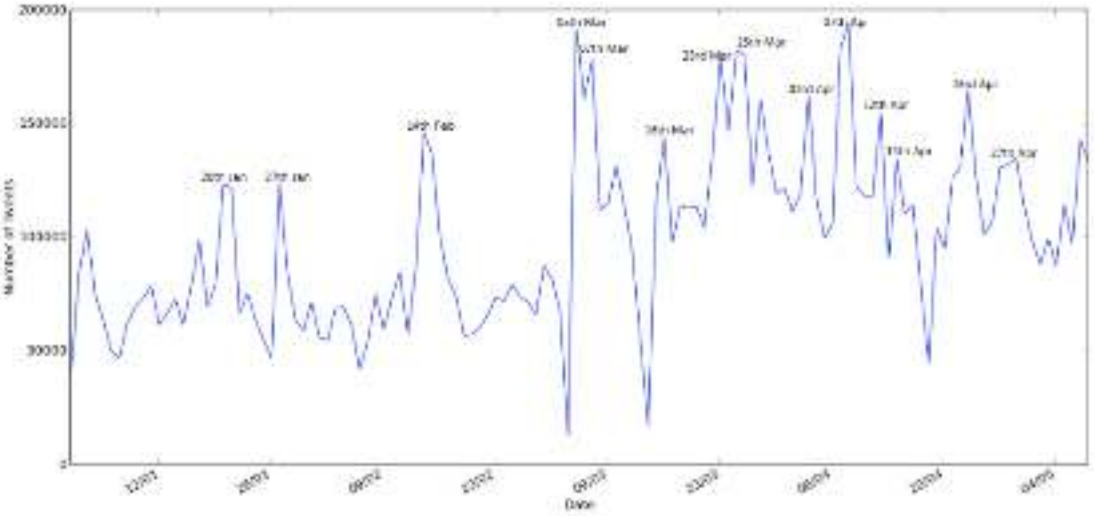
Skoric et. al. [25] in their paper considered the decisions information amid the Singapore General Elections. They discovered that however the prescient intensity of Twitter for decisions cannot be professed to be on a par with in the Germany races by Tumasjan et. al. [26], however it is still superior to risk. Their mean outright mistake was higher than that of past investigations and reasoned that Twitter is characteristic of the popular feeling. Gayo-Avello [11] in his paper considered the US Presidential Elections 2008 and indicates how examination of twitter information neglected to foresee Obama's success even in Texas. He guaranteed that the twitter information is one-sided and cannot be utilized as an agent test. He likewise tested the assumption examination utilized in the before papers.



Graph 11.1: Tweet count for one day broken down from yearly database

To the extent India General Elections 2019 were concerned, twitter information was likewise examined by Sim-plify360.1 They prepared both short summary as well as a detailed report on the elections data. They arranged a Simplify360 Social Index (SSI) to figure the ubiquity of the legislators. Mindfulness, spread, conspicuousness and positivity were 4 parameters they utilized for the figuring of their file. The investigation by the NExT Center at the National University of Singapore was a week by week analysis.2 They would discover the insights about the three noteworthy gatherings AAP, BJP and Congress from the week by week information and furthermore report the political occasions that occurred that week. Their last segment had some political audits about the three principle competitors. Kno.e.sis, an examination bunch at Wright State University additionally broke down India General Elections 2014 with the assistance of Twitris+, a semantic social web application.3 The entry demonstrated the cheerfulness for the three noteworthy gatherings. The confidence was determined taking the quantity of notices and slants of the tweets as parameters. Aside from this, there were a few entryways by news media houses that demonstrated some bit about the sort of exercises going on Online Social Media (OSM), for e.g., the pages by IBN and TOI

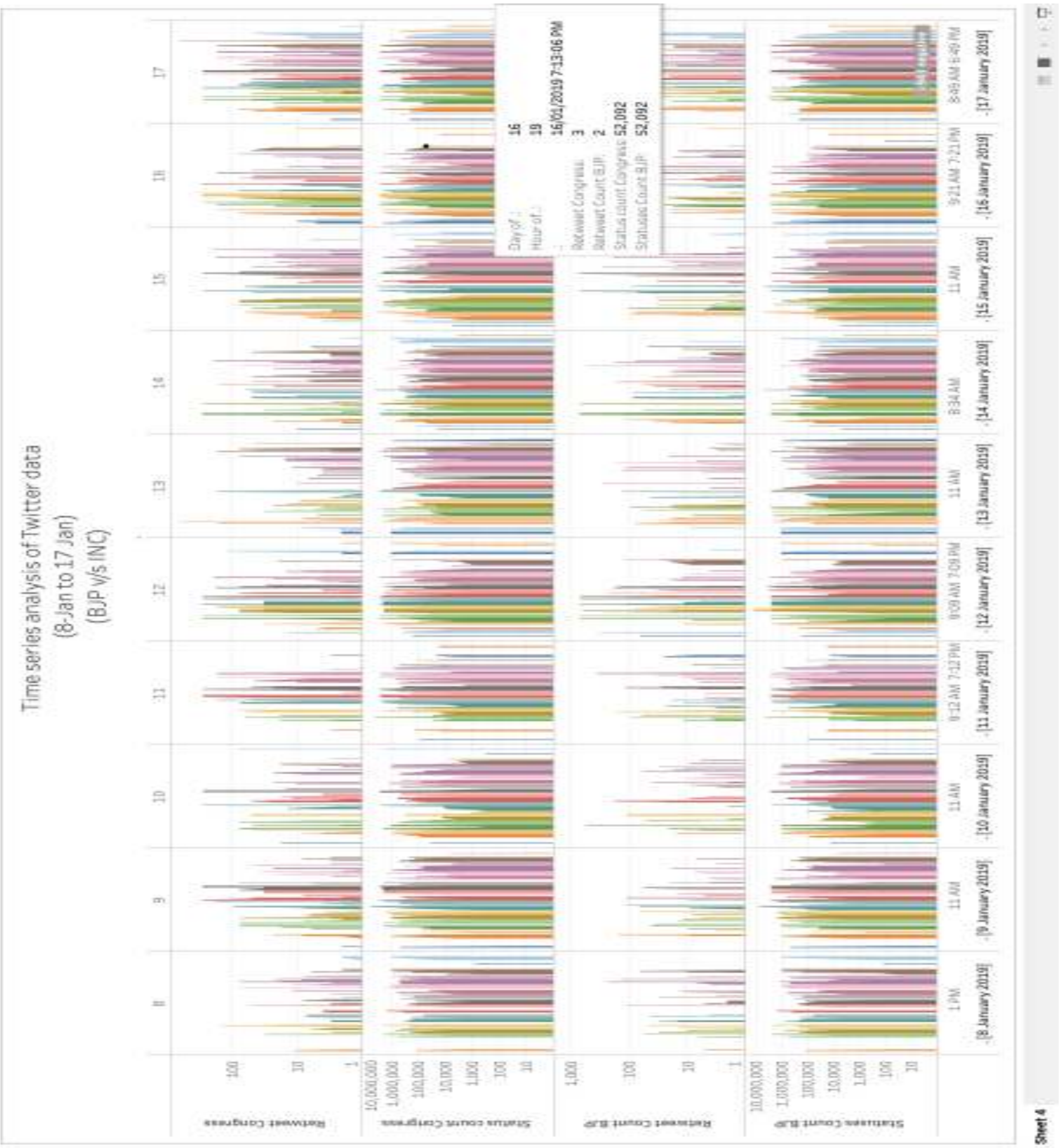
We broke down the total dataset to discover intriguing examples with regards to it and furthermore to check if the trifling things were likewise obvious in the information gathered. We found that the movement on Twitter topped amid significant occasions identified with races. It was clear from our information that the political conduct of the legislators influenced their adherents check and accordingly prominence on Twitter. One more point of our work was to locate a proficient method to arrange the political direction of the clients on Twitter. To achieve this undertaking, we utilized four distinct procedures: two depended on the substance of the tweets made by the client, one on the client-based highlights and another dependent on network recognition calculation on the retweet and client notice systems. We found that the network location calculation worked best with a proficiency of over 80%. It was likewise observed that the substance-based strategies did not admission well in the arrangement results.



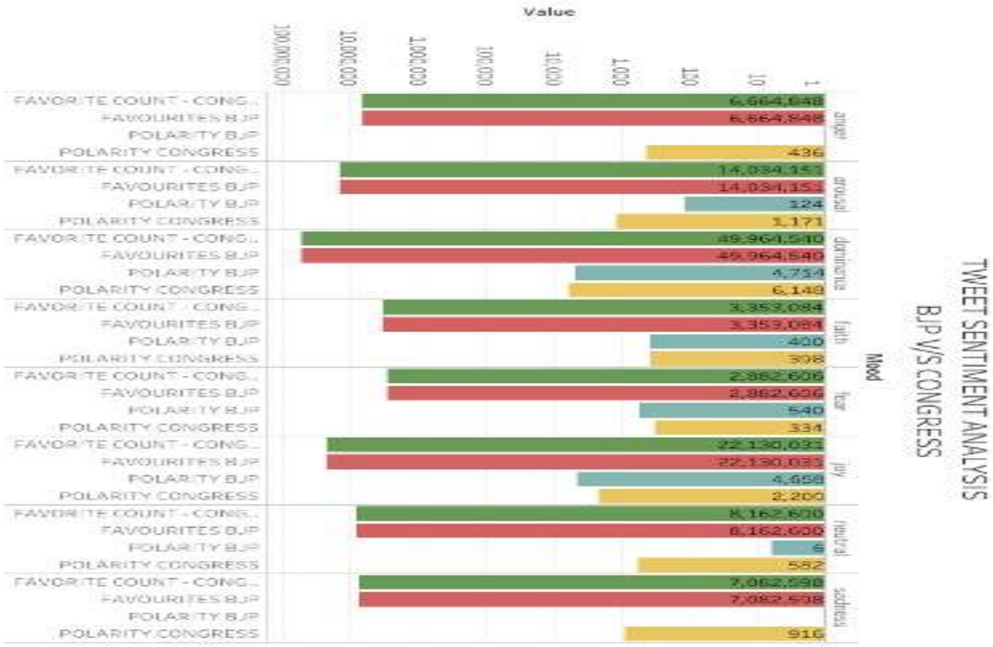
Graph 11.2: Time series analysis of tweet occurrences

With an expect to screen the day by day approaching information, we manufactured an entryway to demonstrate the examination of the tweets of the most recent 24 hours. This entrance broke down the tweets to locate the most drifting themes, hashtags, the sort of slants gotten by the gatherings, area of the tweets and furthermore observed the ubiquity of different political pioneers and their gatherings' records on Twitter. As far as we could possibly know, this is the primary scholarly interest to investigate the races information and arrange the clients in the India General Elections 2019.

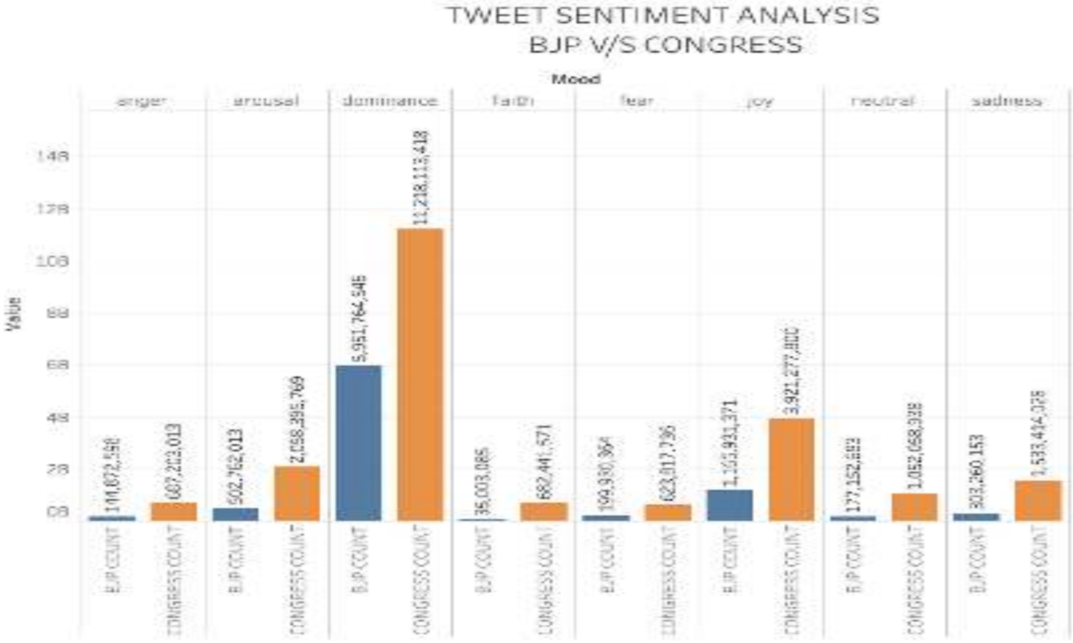
GRAPH DEPICTING THE COMPLEXITY AND VASTNESS OF DATA FROM TWITTER FOR MERELY 10 DAYS



Graph 11.3: Dashboard for Twitter data storage simultaneous comparison between BJP and Congress



Graph 11.4: Comparison between Favourite count of BJP and Congress



Graph 11.5: Comparison between Tweets count of BJP and Congress analysed with sentiment of the tweets

POLITICAL ORIENTATION PREDICTION

Despite the fact that forecast of decision results with Twitter has been a beaten issue, relatively few atentices have been made at the order of the political introduction of the clients. Conoveret. al. [9] examined the 2010 U.S. midterm races information and anticipated the introduction of the twitter clients with an effectiveness of 95%. They utilized the dormant semantic investigation for ordering the substance of the clients' tweets and discovered high connection esteems between the substance and the political arrangement of the clients.

Golbeck et.al. [12] attempted to locate the political inclinations of the devotees of famous news media's records on Twitter. They determined a political inclination score (P Score) for every one of the clients dependent on the records they were following. It was proposed that dependent on the count of the groups of onlookers' inclinations, the inclining of the particular media houses can likewise be discovered. Cohen et. al. [8] in their work tested crafted by Conover et. al. [9]. They said that the classifier created by them would not work on the off chance that it was utilized for a lot of clients who were not in all respects politically dynamic on Twitter. They partitioned the arrangement of clients into political figures, politically dynamic and moderate clients and contrasted it and the Conover et. al. informational index and found the effectiveness of the classifier to be lower for their situation.

In this segment we will look at the prevalence of the two heads, Arvind Kejriwal (AAP) and Narendra Modi (BJP). We are not taking Rahul Gandhi for the examination since he doesn't have a checked record and the record @BeWithRG is additionally not asserted by him. To think about the prominence, we have taken two estimates that are utilized in computation of the Klout score too, for example the quantity of supporters and the quantity of retweets of the tweets made by their record. We determined the Pearson's relationship factor of every one of these two parameters with the Klout score. We found the estimation of Pearson's connection between the Klout score and number of devotees to be 0.956, though it was 0.463 just between the Klout score and the normal number of retweets. So we can say that the quantity of devotees is a superior proportion of ubiquity.

We were following the records of these pioneers since long and endeavoured to see their number of adherents for every day. The record of Narendra Modi, screen name @narendramodi has been on Twitter since January 2009, while Arvind Kejriwal's record @ArvindKejriwal has been since November 2011. Diagrams appearing number of adherents for both these pioneers are appeared in Figure 4.8. We can see that the quantity of devotees for Kejriwal in the start of the chart is about 0.4 million, while it is more than 1.9 million for Modi. There have been steep ascent in the chart for Kejriwal around the dates eighth Dec (when he won Delhi gathering races), 29th Dec (when he made vow as the Delhi CM), twentieth Jan (when he organized dissent against Delhi police) and the lofty ascent to some degree standardized after 28th Jan. Anyway there was no fall in the bend around fourteenth Feb (when he quit as Delhi CM) similar to the hypothesis in the media. When we take a gander at Modi's diagram, we don't locate any precarious ascent or fall even after his presentation as PM applicant on thirteenth Sept, the bend has been always expanding with the exception of a couple of glitches on 23rd Nov (which could be because of ascend in prevalence of AAP).

To think about the two diagrams on an equivalent scale, we looked at the rate change in the quantity of supporters for every day for both these pioneers. The diagram in Figure 4.9 demonstrates the correlation. We can see that the adjustment in Kejriwal's supporters was constantly higher than Modi's, which never topped. If it's not too much trouble note that the drop in this diagram does not demonstrate a drop in number it just demonstrates that the ascent in the quantity of supporters was not as high as the earlier day. The change was never negative. The normal of progress in level of supporters for Kejriwal and Modi were observed to be 0.49% and 0.25% separately. The normal number of supporters every day for Kejriwal was 1,372, though it was 2,020 for Modi. So in the event that we take a gander at it with this angle, Modi has more devotees and along these lines progressively well known.

We would now take a gander at the retweet check of the tweets made by Kejriwal and Modi. For this we took last 1,000 tweets made by both these pioneers and recovered the retweet tally from the of followers, JSON object and plotted it on a diagram against the course of events. This chart can be found in the Figure 4.10. Despite the fact that Modi has the most extreme retweet tally of

5,000 on a tweet, the retweet mean Kejriwal's retweets are commonly higher. This can likewise be set up by looking at the normal retweet mean their last 1,000 tweets, which is 887 for Arvind Kejriwal, though 612 for Modi. So with this viewpoint, Kejriwal is more well-known than Modi. In any case, since we asserted that the quantity of supporters is a somewhat higher connected parameter to the Klout score, we can presume that Modi would be advised to prominence than Kejriwal.

Since we were following the records of different gatherings and government officials, one thing that grabbed our eye was the quantity of tweets made by the record of BJP, named @BJP4India. The tweet check of this record went from 2538 on Feb 01 to 96,462 on Apr 30. They made more than 20,000 on Apr 07 and over 100% ascent in tweets on a few events. This unmistakably demonstrates the way BJP was proactively advancing itself and its PM competitor on twitter, which could be a conceivable explanation behind their lead in practically all charts.

We additionally endeavoured to take a gander at the likelihood of the records being bot accounts. For this reason, we utilized an instrument by Indiana University.1 This apparatus took the screen names of the records as information and the dissected the record for being a bot or not. The end was made based on system investigation, the tweets made and some different variables. We utilized this instrument to break down somewhere in the range of 50 odd profiles from our dataset. What's more, we discovered that practically every one of the profiles were not bots. The conceivable explanation behind this could be that since we had hand-picked these profiles, the odds of them being bots was less. Aside from this we additionally checked for a couple of profiles which were tweeting about decisions and discovered that profiles like Tips4DayTrader were observed to be bots.

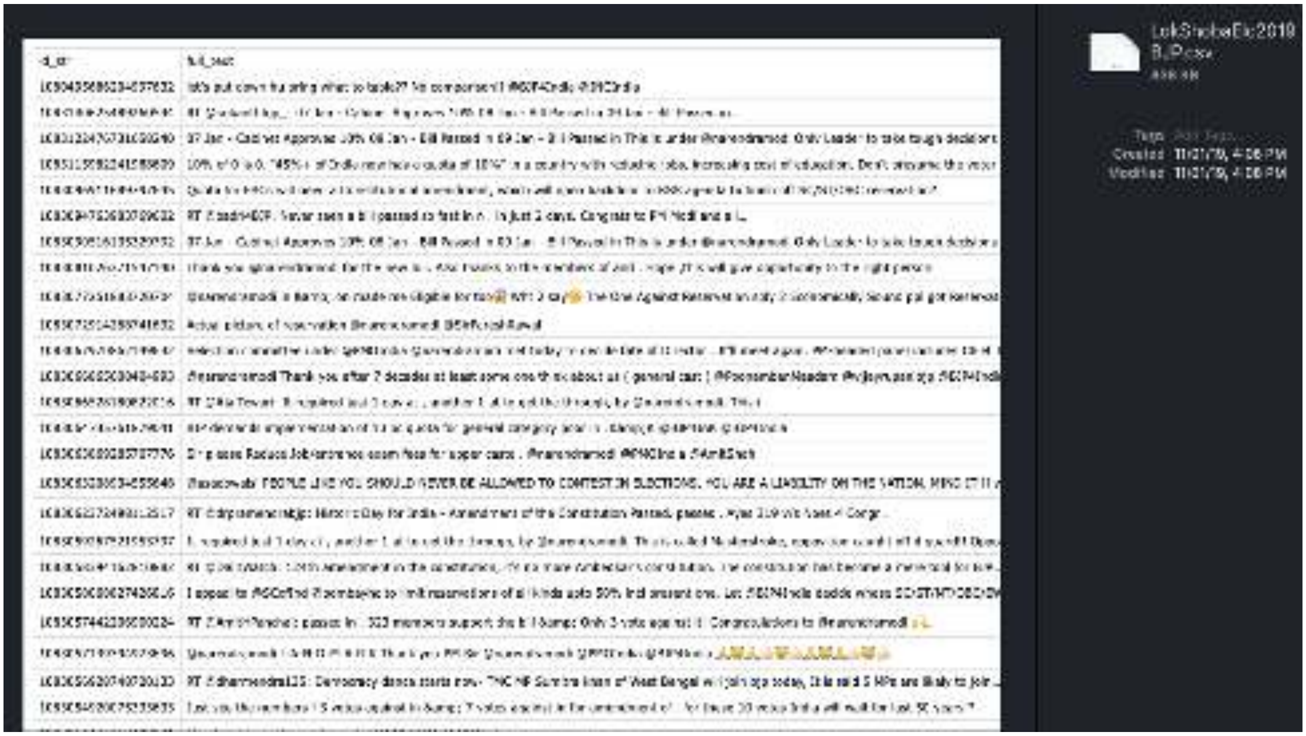
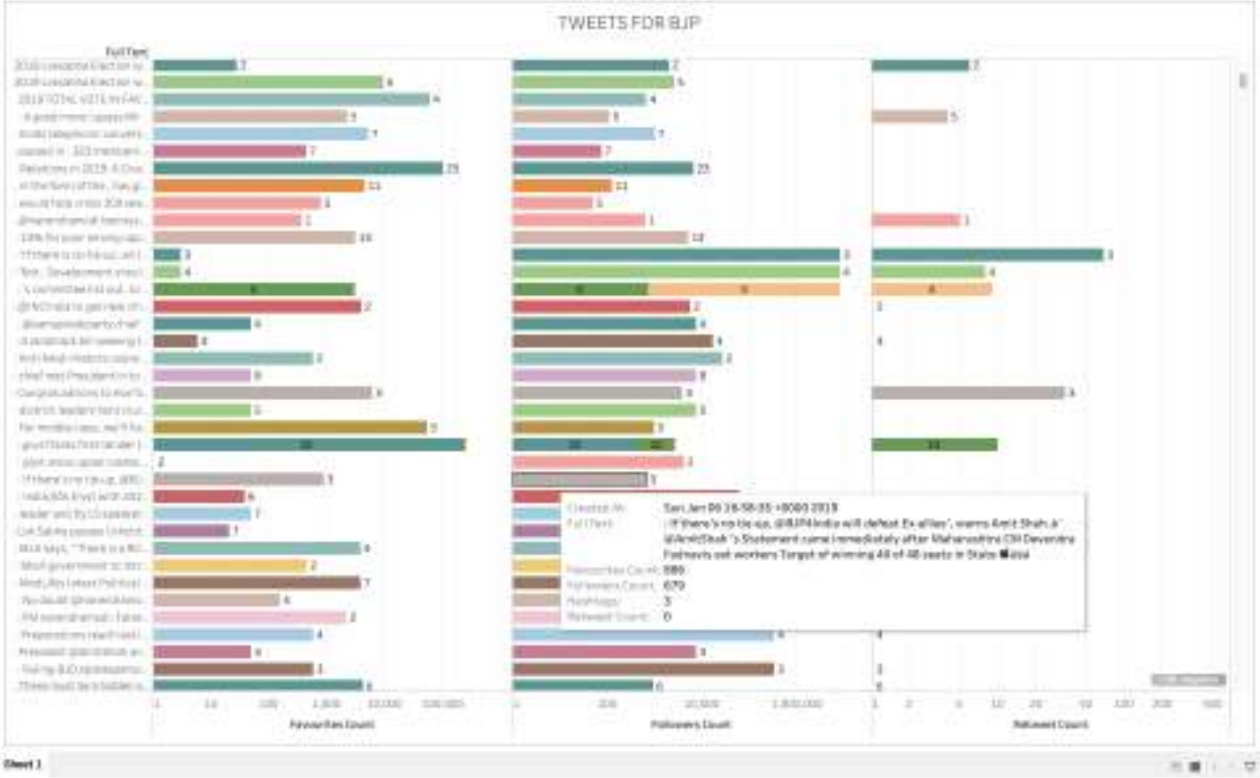
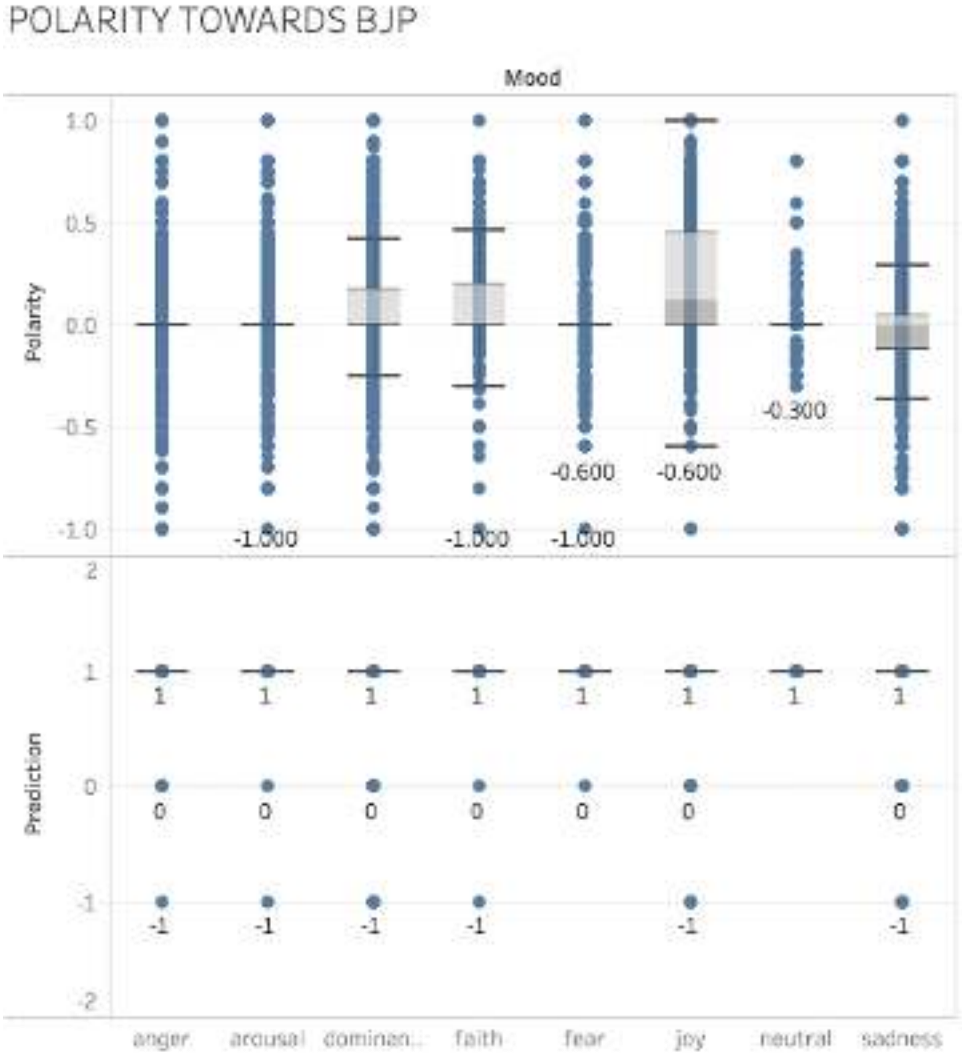


Table 12.1: Sample database retrieved from Twitter for BJP



Graph 12.1: Tweet Favourite count v/s Follower count v/s Retweet count for BJP database



Graph 12.2: Polarity and prediction for BJP tweets – sentiment wise

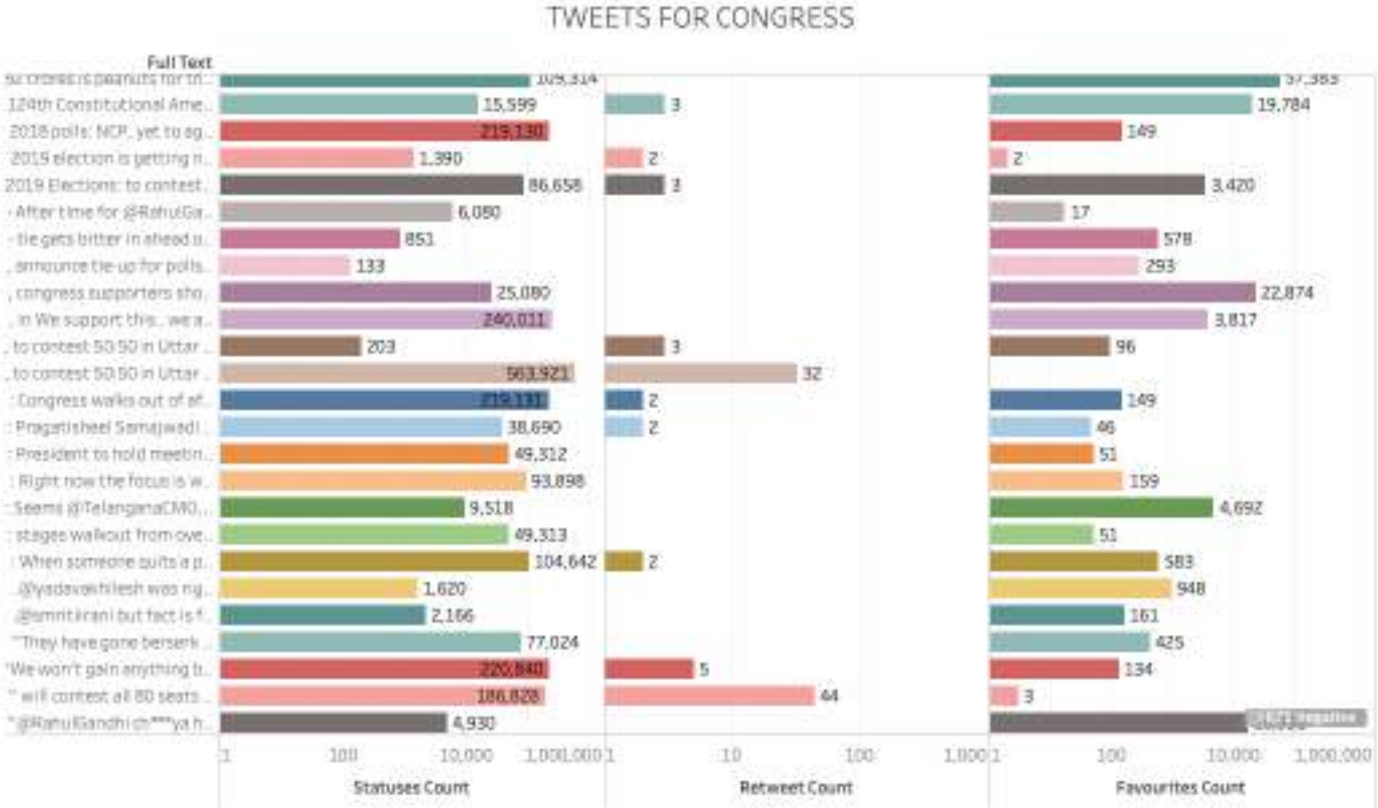
WORDCLOUD FOR BJP

Figure 12.1: BJP related tweet word cloud

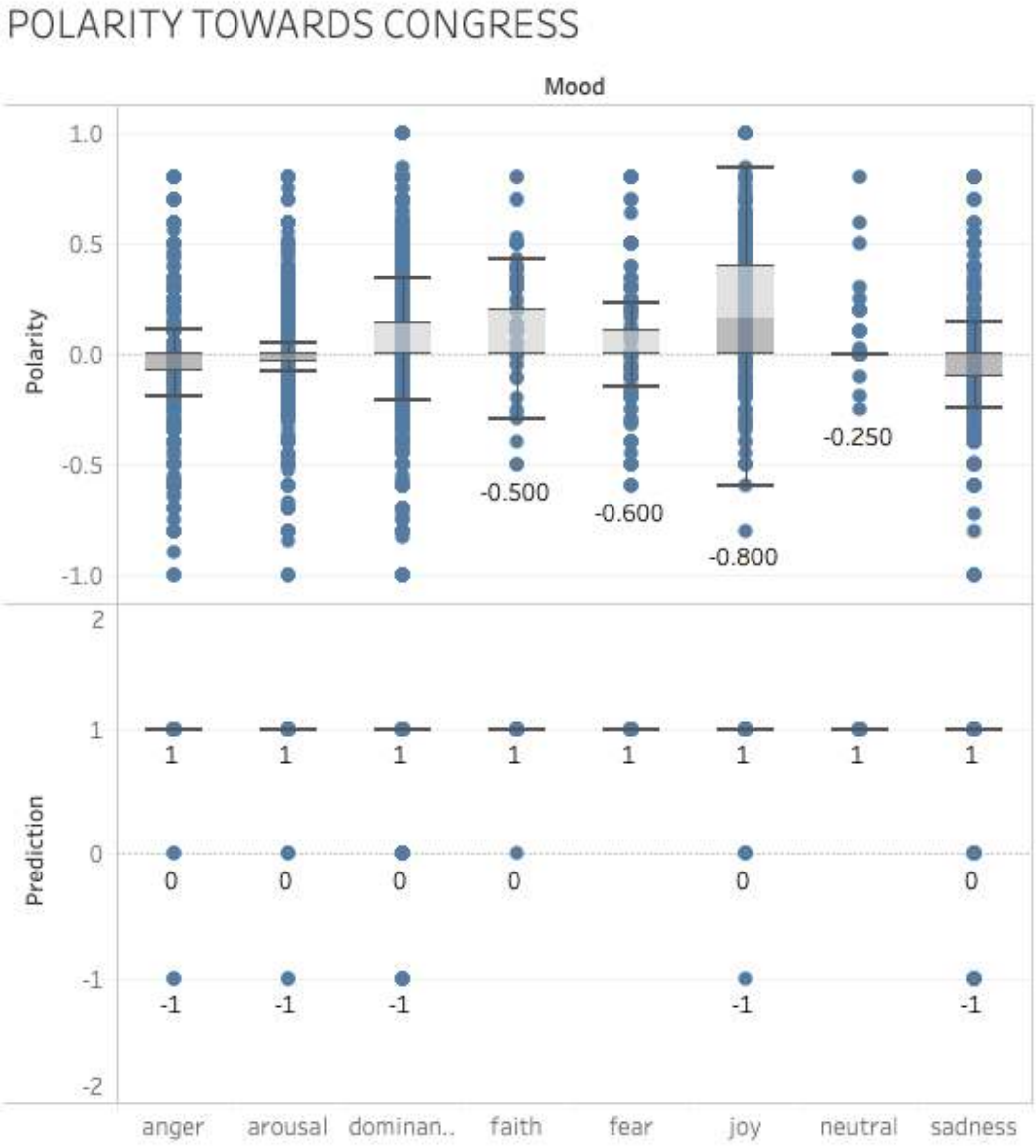
WORD CLOUD FOR CONGRESS

Figure 12.2: Congress related tweet word cloud

Table 12.2: Sample database retrieved from Twitter for Congress



Graph 12.3: Tweet Favourite count v/s Follower count v/s Retweet count for Congress database



Graph 12.4: Polarity and prediction for Congress tweets – sentiment wise

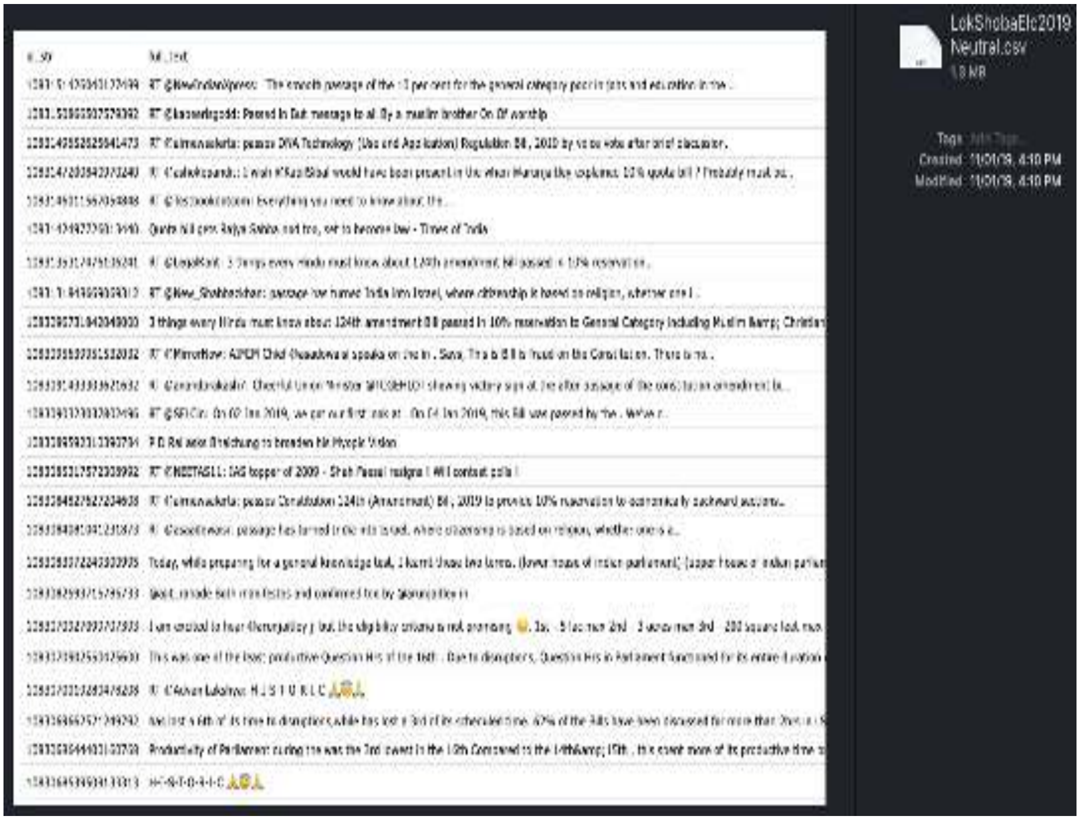


Table 12.3: Sample database retrieved from Twitter for general elections

WORDCLOUD FOR NEUTRAL TWEETS

Figure 12.5: Lok Sabha Election related tweet word cloud

SENTIMENT ANALYSIS

We broke down the total dataset to discover fascinating examples with regards to it and furthermore to confirm if the paltry things were likewise obvious in the data gathered. We found that the movement on Twitter crested amid significant occasions identified with races. It was apparent from our data that the political conduct of the lawmakers influenced their devotees check and in this manner ubiquity on Twitter. One more point of our work was to locate an effective method to arrange the political introduction of the clients on Twitter. To achieve this assignment, we utilized four distinct systems: two depended on the substance of the tweets made by the client, one on the client-based highlights and another dependent on network discovery calculation on the retweet and client notice systems. We found that the network discovery calculation worked best with a proficiency of over 80%. It was likewise observed that the substance-based techniques did not toll well in the arrangement results. With a mean to screen the everyday approaching information, we fabricated an entryway to demonstrate the investigation of the tweets of the most recent 24 hours. This gateway broke down the tweets to locate the most slanting points, hashtags, the sort of notions gotten by the gatherings, area of the tweets and furthermore checked the ubiquity of different political pioneers and their gatherings' records on Twitter. As far as we could possibly know, this is the primary scholastic interest to break down the races information and arrange the clients in the India General Elections 2019.

Sentiment analysis is the computational investigation of suppositions, assumptions, assessments, mentalities, perspectives and feelings communicated in content. It alludes to a characterization issue where the primary center is to foresee the extremity of words and after that order them into positive or negative slant. Assumption examination over Twitter offers individuals a quick and powerful approach to gauge the open's emotions towards their gathering and government officials. The essential issue in past conclusion examination strategies is the assurance of the most fitting classifier for a given order issue. On the off chance that one classifier is browsed the accessible classifiers, at that point there is no surety in the best execution on inconspicuous information. So, to decrease the danger of choosing an improper classifier, we are joining the yields of a lot of classifiers. In this way in this paper, we utilize a methodology that consequently arranges the conclusion of tweets by joining AI classifiers with vocabulary-based classifier. The new mix of classifiers are SentiWordNet classifier, innocent Bayes classifier and shrouded Markov model classifier. Here inspiration or antagonism of each tweet is controlled by utilizing the lion's share casting a ballot guideline on the aftereffect of these three classifiers. In this manner we were utilized this slant classifier for finding political opinion from constant tweets. In this way we have an improved exactness in conclusion examination utilizing classifier troupe Approach. Our strategy likewise utilizes refutation dealing with and word sense disambiguation to accomplish high precision.

ELECTIONS AND SOCIAL MEDIA

There was a noteworthy change in the General Elections 2014 from the General Elections 2009; this was the adjustment in the pretended by the web-based social networking amid the races. It has been watched worldwide that the majority rules systems have been taking part in discoursed with the general population over the online networking [16]. According to Internet and Mobile Association of India (IAMAI) the Internet clients in India are relied upon to be 243 million by June 2014, denoting a 28% development from June 2013.4 There are an assortment of online life stages available, e.g., Facebook, Twitter, Pinterest, Instagram, Tumblr, Flickr, Google+, LinkedIn to give some examples. The general population are progressively utilizing these stages to express their perspectives on various points. India has been at the cutting edge of the development stories in the quantity of clients of these online life stages. The Facebook being the most well-known has 114.8 million Indian users.5 Twitter positions second in the quantity of clients with 33 million clients from India.

Consider it an impact of the US Presidential Elections or not, yet the web-based life stages were utilized by the gatherings for their crusades. Practically all the significant gatherings made their essence felt on the internet-based life with the official records and checked pages of their pioneers and the gatherings. According to the media reports, proficient assistance was taken by the gatherings to improve the picture of their gathering and pioneers on the interpersonal interaction locales. According to an investigation of IAMAI, there were around 160 odd voting public out of 543 that were probably going to be impacted by these new media and furthermore influence 3 – 4% of votes.6 Some gatherings like the BJP and Congress utilized Google+ Hangout to lead gatherings with people.7 Candidates utilized the vehicle of broadcast meets on Facebook and WhatsApp informing administration to associate with a great many urban voters. Crusade recordings were transferred on YouTube by a few gatherings. A great deal of battling promotions was seen on Facebook amid the races, while every one of the photos of Rahul Gandhi's energizes were transferred on an Instagram account. Google did its bit by acquainting the Google Election center point with engage voters and by giving more data about the races and the candidates.

DATA GATHERED

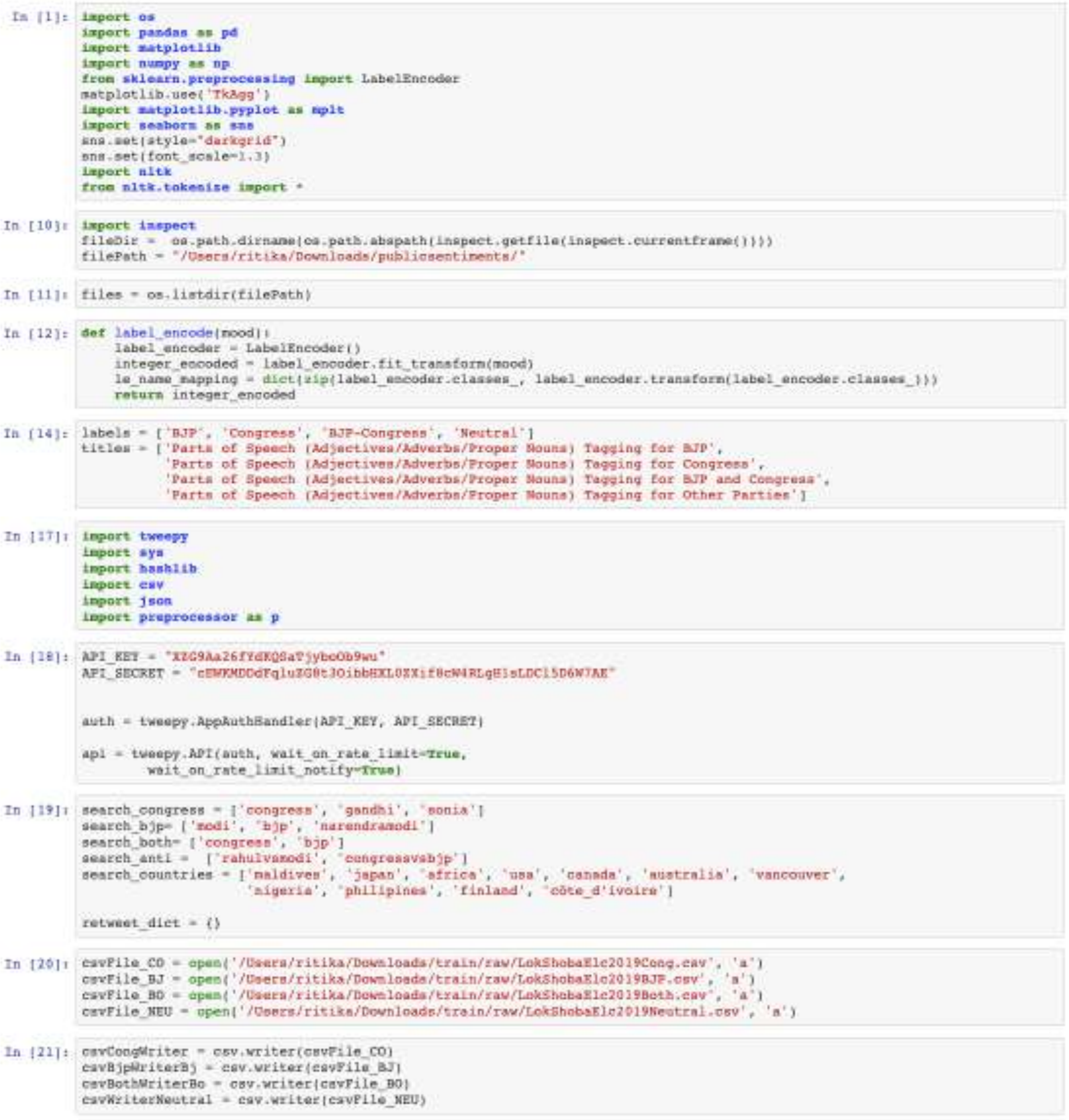
There are 50 million Twitter clients in the United States and utilize English as their medium to communicate on Twitter.10 But in India, the scenario is different with as many as 1,616 parties and almost 8,000 competitors. The populace in India utilizing Twitter is minutely little when contrasted with the genuine casting a ballot populace. With individuals discussing such a large number of gatherings and applicants and that also in transliterated messages, the issue of recognizing their political tendency ends up more extensive and intriguing. Every one of these variables alongside sheer measure of information inspired us to investigate the information and find intriguing examples with regards to it and on the off chance that it was in a state of harmony with the continuous occasions as they were going on.
So we can broadly specify the aims of our work as follows :

1. To dissect and draw important derivations from the gathering of tweets gathered over the whole term of races

2. To check the achievability of improvement of an arrangement model to distinguish the political introduction of the twitter clients dependent on the tweet content and other client based highlights.

3. To build up a framework to break down and screen the race related tweets on regular routine.

TRAINING THE DATASET USING MACHINE LEARNING



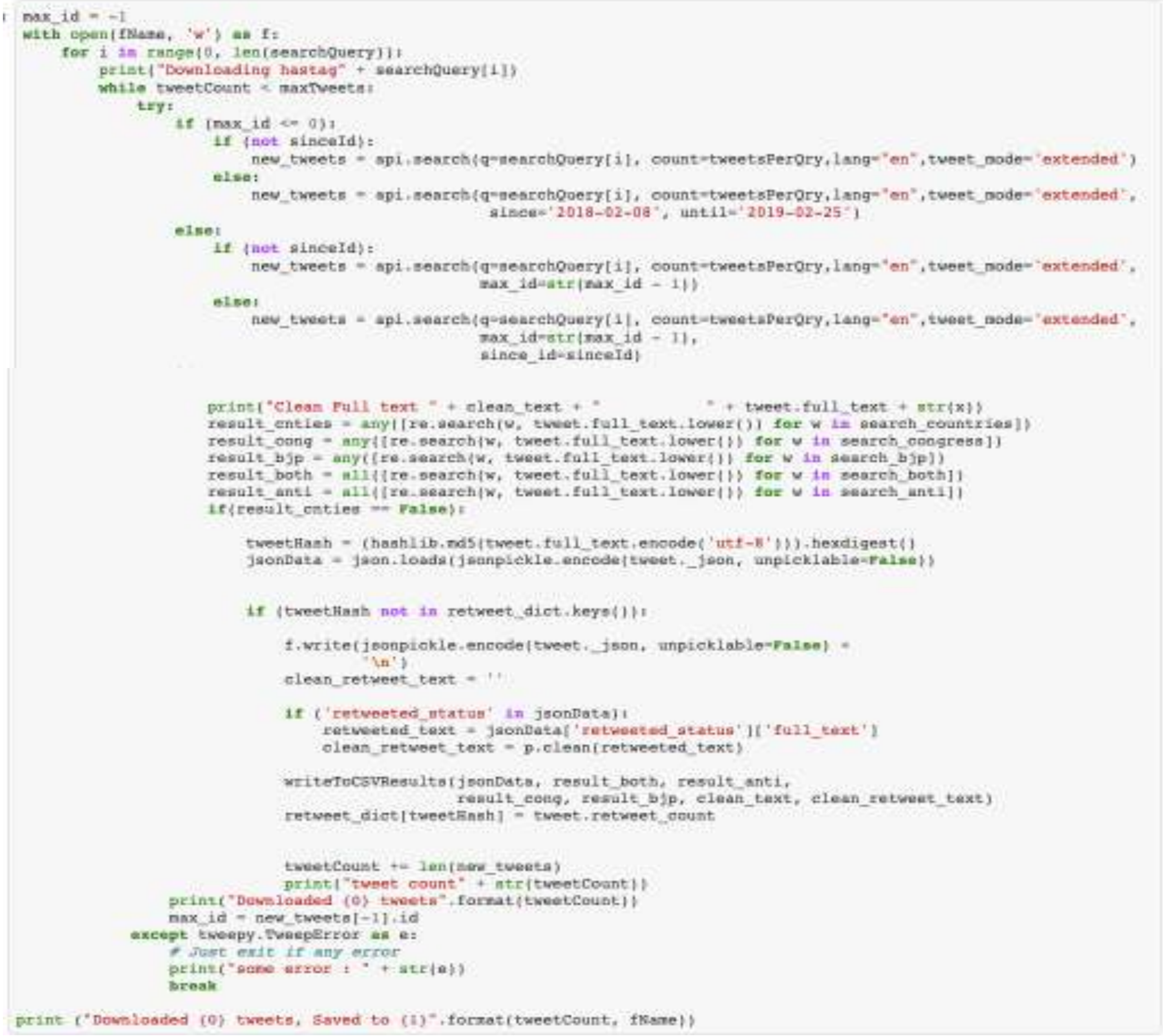
```
In [1]: import os
        import pandas as pd
        import matplotlib
        import numpy as np
        from sklearn.preprocessing import LabelEncoder
        matplotlib.use('TkAgg')
        import matplotlib.pyplot as mplt
        import seaborn as sns
        sns.set(style="darkgrid")
        sns.set(font_scale=1.3)
        import nltk
        from nltk.tokenize import *

In [10]: import inspect
         fileDir =  os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
         filePath = "/Users/ritika/Downloads/publicsentiments/"

In [11]: files = os.listdir(filePath)

In [12]: def label_encode(mood):
             label_encoder = LabelEncoder()
             integer_encoded = label_encoder.fit_transform(mood)
             le_name_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
             return integer_encoded

In [14]: labels = ['BJP', 'Congress', 'BJP-Congress', 'Neutral']
         titles = ['Parts of Speech (Adjectives/Adverbs/Proper Nouns) Tagging for BJP',
                   'Parts of Speech (Adjectives/Adverbs/Proper Nouns) Tagging for Congress',
                   'Parts of Speech (Adjectives/Adverbs/Proper Nouns) Tagging for BJP and Congress',
                   'Parts of Speech (Adjectives/Adverbs/Proper Nouns) Tagging for Other Parties']

In [17]: import tweepy
         import sys
         import hashlib
         import csv
         import json
         import preprocessor as p

In [18]: API_KEY = "XZG9Aa26fYdKQ8aTjybcOb9wu"
         API_SECRET = "cEWKMDDdFqluZG8tJOibbHXL0XXif8cW4RLgHisLDC15D6W7AE"


         auth = tweepy.AppAuthHandler(API_KEY, API_SECRET)

         api = tweepy.API(auth, wait_on_rate_limit=True,
                  wait_on_rate_limit_notify=True)

In [19]: search_congress = ['congress', 'gandhi', 'sonia']
         search_bjp= ['modi', 'bjp', 'narendramodi']
         search_both= ['congress', 'bjp']
         search_anti =  ['rahulvsmodi', 'congressvsbjp']
         search_countries = ['maldives', 'japan', 'africa', 'usa', 'canada', 'australia', 'vancouver',
                             'nigeria', 'philipines', 'finland', 'côte_d'ivoire']

         retweet_dict = {}

In [20]: csvFile_CO = open('/Users/ritika/Downloads/train/raw/LokShobaElc2019Cong.csv', 'a')
         csvFile_BJ = open('/Users/ritika/Downloads/train/raw/LokShobaElc2019BJP.csv', 'a')
         csvFile_BO = open('/Users/ritika/Downloads/train/raw/LokShobaElc2019Both.csv', 'a')
         csvFile_NEU = open('/Users/ritika/Downloads/train/raw/LokShobaElc2019Neutral.csv', 'a')

In [21]: csvCongWriter = csv.writer(csvFile_CO)
         csvBjpWriterBj = csv.writer(csvFile_BJ)
         csvBothWriterBo = csv.writer(csvFile_BO)
         csvWriterNeutral = csv.writer(csvFile_NEU)
```

```
In [22]: hastags = ['#congresswins']

         if (not api):
             print ("Can't Authenticate")
             sys.exit(-1)

In [23]: import sys
         import jsonpickle
         import os

In [35]: searchQuery = ['#ELECTION2019']

In [36]: maxTweets = 100000000 # Some arbitrary large number
         tweetsPerQry = 100  # this is the max the API permits
         fName = 'tweets.txt'

In [37]: sinceId = "2014-01-01"

In [38]: tweetCount = 0
         print("Downloading max {0} tweets".format(maxTweets))


         def writeToCSVResults(jsonData, result_both, result_anti,
                               result_cong, result_bjp, clean_text, clean_retweet_text):
             retweeted_text = clean_retweet_text
             if('location' in jsonData['user'] and len(jsonData['user']['location']) > 0):
                 location = jsonData['user']['location'].replace(',', '')
             else:
                 location = 'Unknown'

             if (result_both or result_anti):
```

```
        csvBothWriterBo.writerow([jsonData['id_str'], clean_text,
                                  jsonData['created_at'], jsonData['user']['favourites_count'],
                                  jsonData['user']['statuses_count'],
                                  jsonData['user']['followers_count'],
                                  jsonData['retweeted'], jsonData['retweet_count'],
                                  retweeted_text,
                                   location, len(jsonData['entities']['hashtags']), len(jsonData['entities']
                                                                                    ['user_mentions']),
                                   len(jsonData['entities']['symbols']), len(jsonData['entities']['urls'])])

     elif (result_cong):

        csvCongWriter.writerow([jsonData['id_str'], clean_text,
                                  jsonData['created_at'], jsonData['user']['favourites_count'],
                                  jsonData['user']['statuses_count'],
                                  jsonData['user']['followers_count'],
                                  jsonData['retweeted'], jsonData['retweet_count'],
                                  retweeted_text,
                                  location, len(jsonData['entities']['hashtags']),
                                 len(jsonData['entities']['user_mentions']),
                                 len(jsonData['entities']['symbols']), len(jsonData['entities']['urls'])])

     elif (result_bjp):

        csvBjpWriterBj.writerow([jsonData['id_str'],clean_text,
                                  jsonData['created_at'], jsonData['user']['favourites_count'],
                                  jsonData['user']['statuses_count'],
                                  jsonData['user']['followers_count'],
                                  jsonData['retweeted'], jsonData['retweet_count'],
                                  retweeted_text,
                                  location, len(jsonData['entities']
                                                ['hashtags']), len(jsonData['entities']['user_mentions']),
                                 len(jsonData['entities']['symbols']), len(jsonData['entities']['urls'])])
```

```
    else:
        csvWriterNeutral.writerow([jsonData['id_str'], clean_text,
                                   jsonData['created_at'], jsonData['user']['favourites_count'],
                                   jsonData['user']['statuses_count'],
                                   jsonData['user']['followers_count'],
                                   jsonData['retweeted'], jsonData['retweet_count'],
                                   retweeted_text,
                                   location, len(jsonData['entities']['hashtags']),
                                   len(jsonData['entities']['user_mentions']), len(jsonData['entities']['symbols']),
                                   len(jsonData['entities']['urls'])])

x = 0
```

```
Downloading max 100000000 tweets
```



```
max_id = -1
with open(fName, 'w') as f:
    for i in range(0, len(searchQuery)):
        print("Downloading hastag" + searchQuery[i])
        while tweetCount < maxTweets:
            try:
                if (max_id <= 0):
                    if (not sinceId):
                        new_tweets = api.search(q=searchQuery[i], count=tweetsPerQry,lang="en",tweet_mode='extended')
                    else:
                        new_tweets = api.search(q=searchQuery[i], count=tweetsPerQry,lang="en",tweet_mode='extended',
                                                 since='2018-02-08', until='2019-02-25')
                else:
                    if (not sinceId):
                        new_tweets = api.search(q=searchQuery[i], count=tweetsPerQry,lang="en",tweet_mode='extended',
                                                max_id=str(max_id - 1))
                    else:
                        new_tweets = api.search(q=searchQuery[i], count=tweetsPerQry,lang="en",tweet_mode='extended',
                                                max_id=str(max_id - 1),
                                                since_id=sinceId)


                    print("Clean Full text " + clean_text + "            " + tweet.full_text + str(x))
                    result_cnties = any([re.search(w, tweet.full_text.lower()) for w in search_countries])
                    result_cong = any([re.search(w, tweet.full_text.lower()) for w in search_congress])
                    result_bjp = any([re.search(w, tweet.full_text.lower()) for w in search_bjp])
                    result_both = all([re.search(w, tweet.full_text.lower()) for w in search_both])
                    result_anti = all([re.search(w, tweet.full_text.lower()) for w in search_anti])
                    if(result_cnties == False):

                        tweetHash = (hashlib.md5(tweet.full_text.encode('utf-8'))).hexdigest()
                        jsonData = json.loads(jsonpickle.encode(tweet._json, unpicklable=False))


                        if (tweetHash not in retweet_dict.keys()):

                            f.write(jsonpickle.encode(tweet._json, unpicklable=False) +
                                    '\n')
                            clean_retweet_text = ''

                            if ('retweeted_status' in jsonData):
                                retweeted_text = jsonData['retweeted_status']['full_text']
                                clean_retweet_text = p.clean(retweeted_text)

                            writeToCSVResults(jsonData, result_both, result_anti,
                                              result_cong, result_bjp, clean_text, clean_retweet_text)
                            retweet_dict[tweetHash] = tweet.retweet_count


                            tweetCount += len(new_tweets)
                            print("tweet count" + str(tweetCount))
                print("Downloaded {0} tweets".format(tweetCount))
                max_id = new_tweets[-1].id
            except tweepy.TweepError as e:
                # Just exit if any error
                print("some error : " + str(e))
                break

print ("Downloaded {0} tweets, Saved to {1}".format(tweetCount, fName))
```

OUTPUT

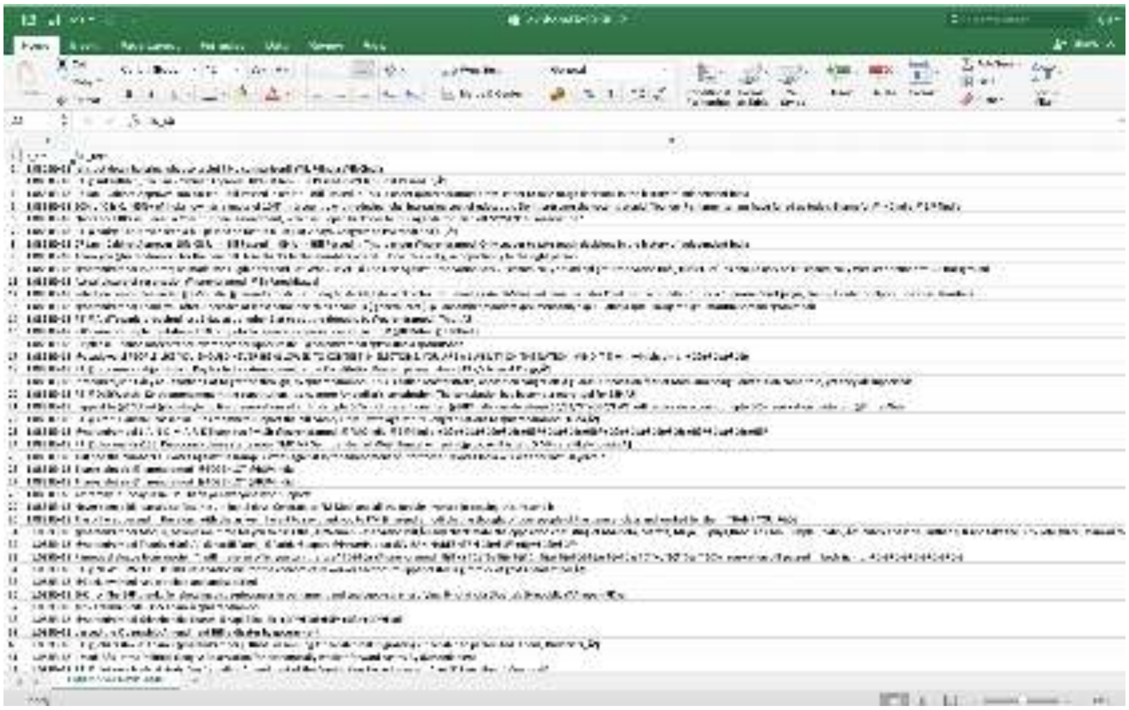


Table 12.1: Trained data sample 1

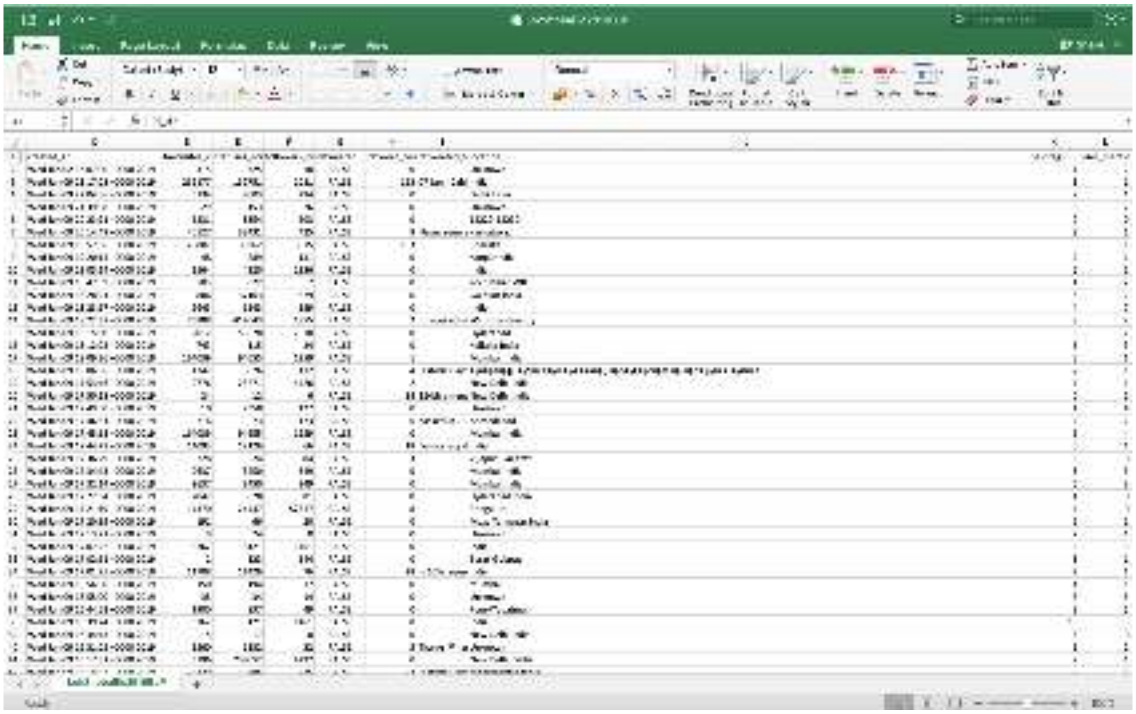
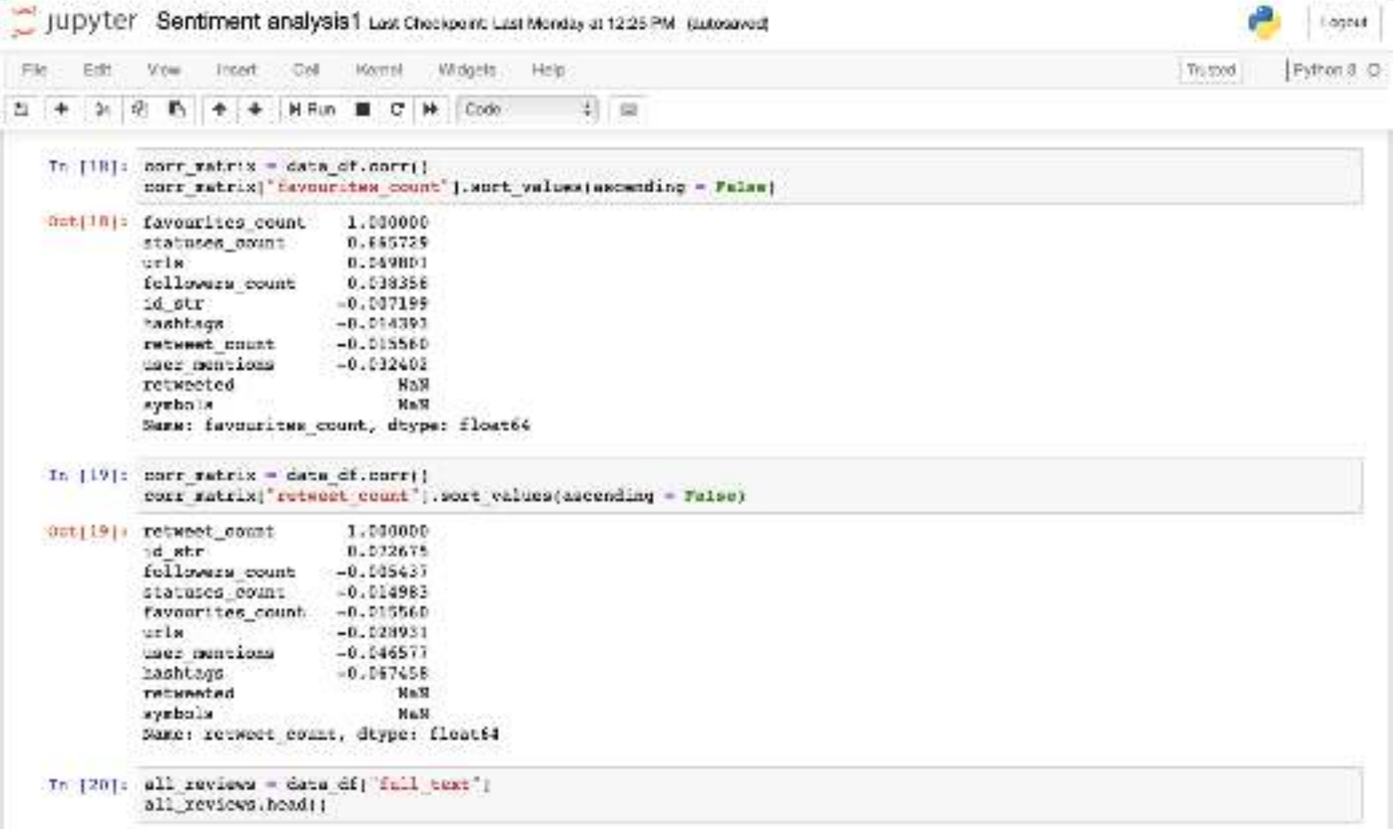
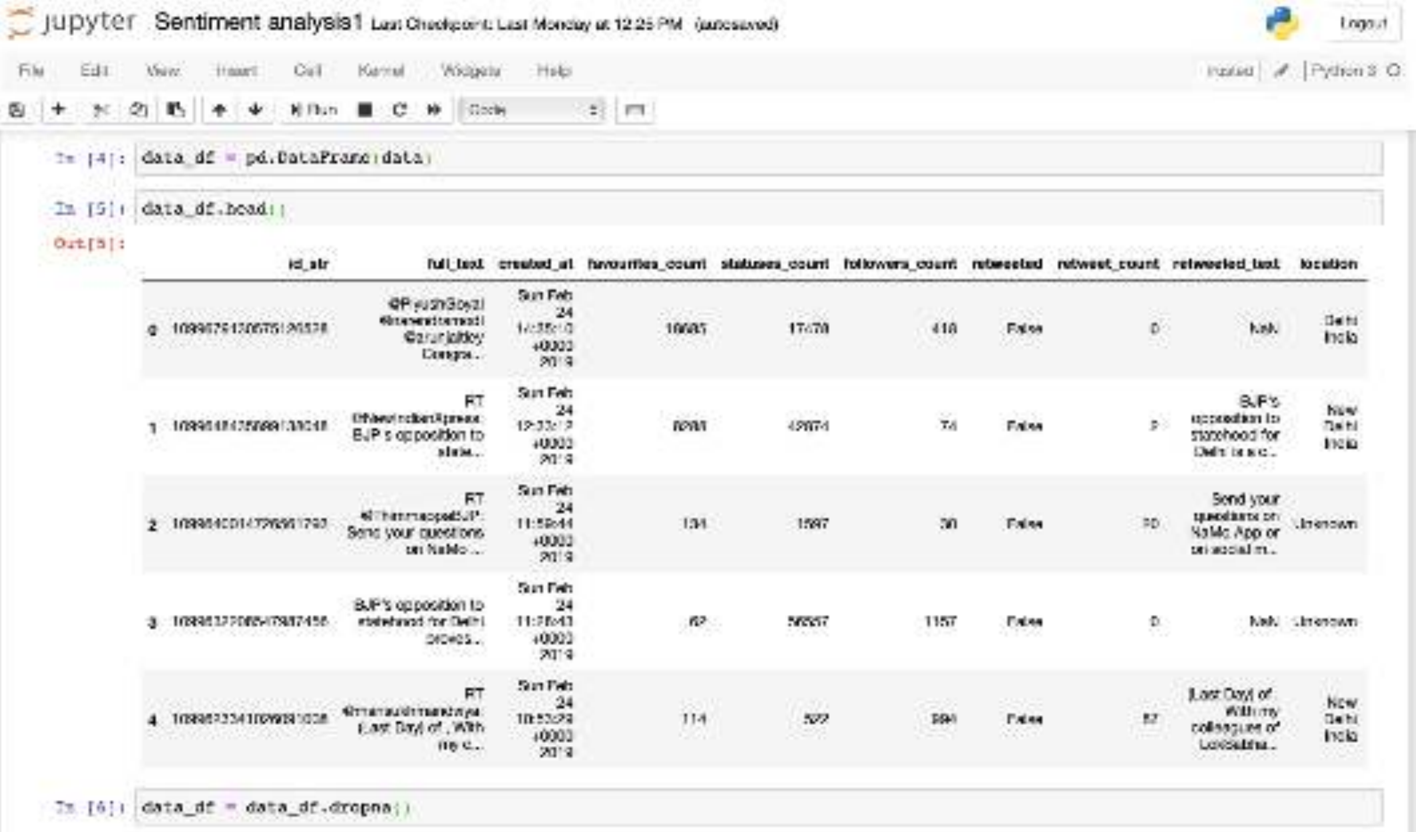


Table 12.2: Trained data sample 2

```
In [34]: sample = data_df['full_text'][1]
         print(sample)
         print('Sentiment: ')
         print(sentiment_value(sample))

         RT @NewIndianXpress: BJP's opposition to statehood for Delhi is a confession that lied during the elections in 2014…
         Sentiment:
         -0.4

In [35]: sample = data_df['full_text'][100]
         print(sample)
         print('Sentiment: ')
         print(sentiment_value(sample))

         RT @Hemants9781: @narendramodi PM Modi seeks full majority, says India stock up not because of him, but strong Govt
         Sentiment:
         0.7

In [37]: sample = data_df['full_text'][7]
         print(sample)
         print('Sentiment: ')
         print(sentiment_value(sample))

         RT @firstpost: 's final address saw a liberal dose of irony and metaphors as the prime minister recounted the govern…
         Sentiment:
         -0.1

In [47]: temp_data = data_df[0:20000]

In [49]: temp_data.shape

Out[49]: (755, 14)
```



```
In [18]: corr_matrix = data_df.corr()
         corr_matrix["favourites_count"].sort_values(ascending = False)

Out[18]: favourites_count    1.000000
         statuses_count      0.045725
         urls                0.049801
         followers_count     0.038356
         id_str             -0.007199
         hashtags           -0.014393
         retweet_count      -0.015560
         user_mentions      -0.032402
         retweeted                NaN
         symbols                  NaN
         Name: favourites_count, dtype: float64

In [19]: corr_matrix = data_df.corr()
         corr_matrix["retweet_count"].sort_values(ascending = False)

Out[19]: retweet_count       1.000000
         id_str              0.072675
         followers_count    -0.005437
         statuses_count     -0.014983
         favourites_count   -0.015560
         urls               -0.028931
         user_mentions      -0.046577
         hashtags           -0.067458
         retweeted                NaN
         symbols                  NaN
         Name: retweet_count, dtype: float64

In [20]: all_reviews = data_df['full_text']
         all_reviews.head()
```



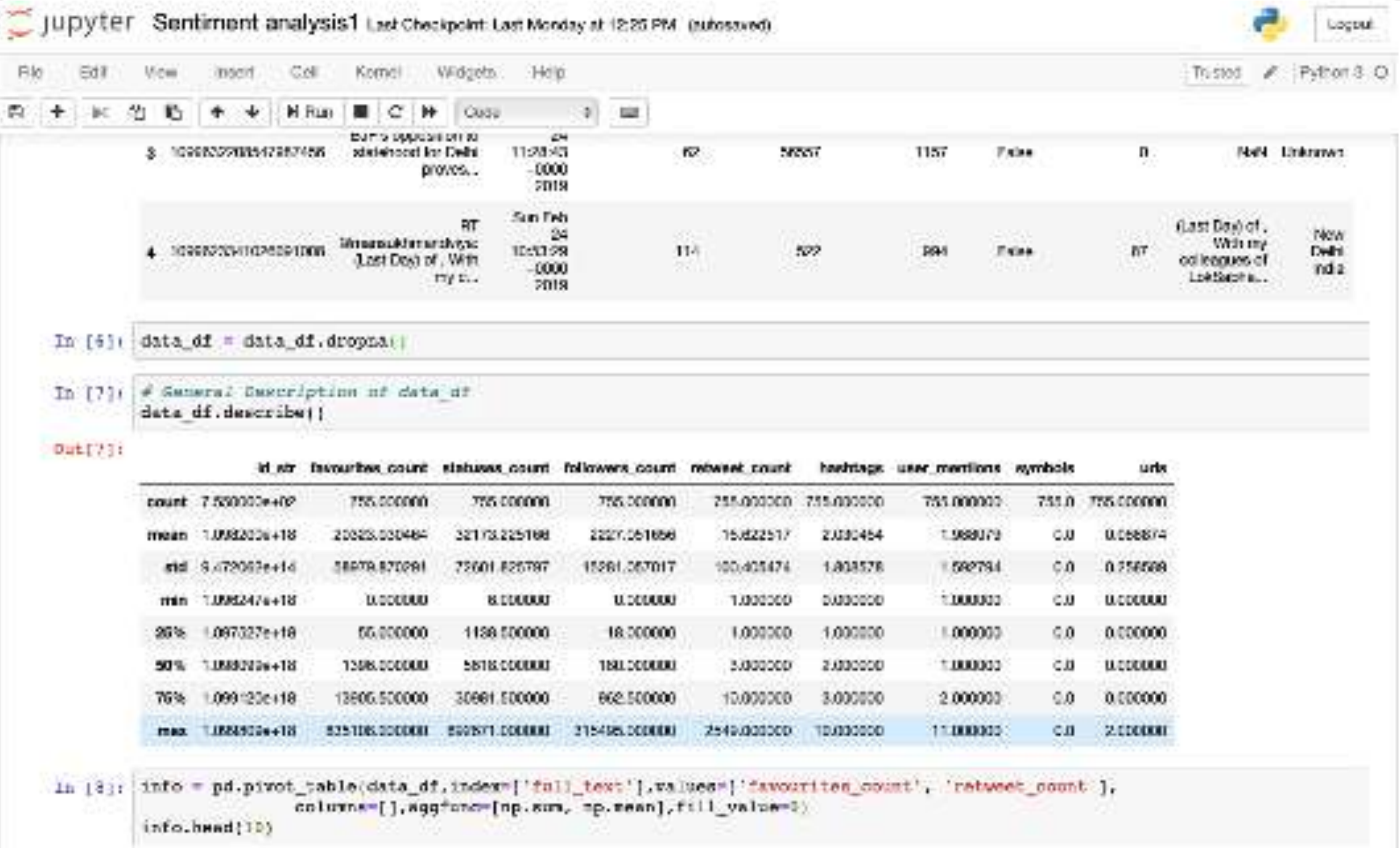
```
In [4]: data_df = pd.DataFrame(data)
```

```
In [5]: data_df.head()
```

Out[5]:

	id_str	full_text	created_at	favourites_count	statuses_count	followers_count	retweeted	retweet_count	retweeted_text	location
0	1099079413067512832	@PiyushGoyal #bharatbetrayed @arunjaitley Congre...	Sun Feb 24 12:35:10 +0000 2019	16665	17478	418	False	0	NaN	Delhi India
1	1099048475668133048	RT @NewIndianXpress: BJP's opposition to state...	Sun Feb 24 12:33:12 +0000 2019	6288	49874	74	False	2	BJP's opposition to statehood for Delhi is sic...	New Delhi India
2	1099040014726961793	RT @[illegible]: Send your questions on NaMo ...	Sun Feb 24 11:59:44 +0000 2019	134	1597	30	False	50	Send your questions on NaMo App or on social m...	Unknown
3	1099032709847867456	BJP's opposition to statehood for Delhi proves...	Sun Feb 24 11:28:43 +0000 2019	62	56657	1157	False	0	NaN	Unknown
4	1099023341029891008	RT @mansukhmandviya: (Last Day) of , With my c...	Sun Feb 24 10:53:28 +0000 2019	114	522	984	False	87	(Last Day) of , With my colleagues of LokSabha...	New Delhi India

```
In [6]: data_df = data_df.dropna()
```



3	1099032709847867456	BJP's opposition to statehood for Delhi proves...	11:28:43 +0000 2019	62	56657	1157	False	0	NaN	Unknown
4	1099023341029291008	RT @mansukhmandviya: (Last Day) of , With my c...	Sun Feb 24 10:53:28 +0000 2019	114	522	984	False	87	(Last Day) of , With my colleagues of LokSabha...	New Delhi India

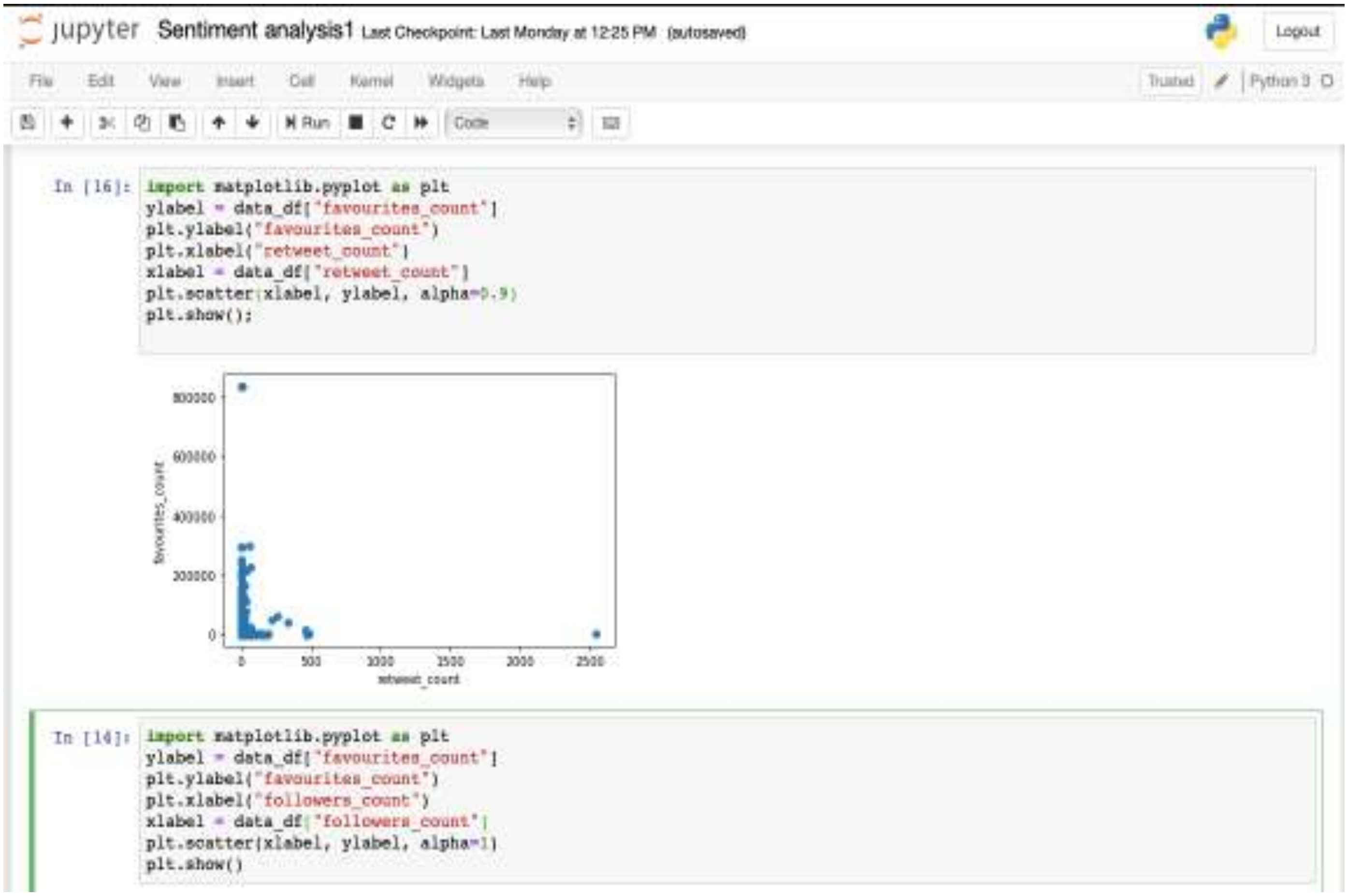
```
In [6]: data_df = data_df.dropna()
```

```
In [7]: # General Description of data_df
        data_df.describe()
```

Out[7]:

	id_str	favourites_count	statuses_count	followers_count	retweet_count	hashtags	user_mentions	symbols	urls
count	7.550000e+02	755.000000	755.000000	755.000000	755.000000	755.000000	755.000000	755.0	755.000000
mean	1.098202e+18	20023.030464	32173.225166	2227.061656	15.622517	2.030464	1.988079	0.0	0.066874
std	9.472067e+14	58979.870291	72601.825787	15261.267017	100.405474	1.808578	1.592794	0.0	0.256598
min	1.098247e+18	0.000000	8.000000	0.000000	1.000000	0.000000	1.000000	0.0	0.000000
25%	1.097327e+18	55.000000	1138.500000	18.000000	1.000000	1.000000	1.000000	0.0	0.000000
50%	1.098202e+18	1306.000000	5818.000000	180.000000	3.000000	2.000000	1.000000	0.0	0.000000
75%	1.099120e+18	13806.500000	30661.500000	662.500000	10.000000	3.000000	2.000000	0.0	0.000000
max	1.098902e+18	525106.000000	590571.000000	215496.000000	2549.000000	10.000000	11.000000	0.0	2.000000

```
In [8]: info = pd.pivot_table(data_df,index=['full_text'],values=['favourites_count', 'retweet_count'],
                    columns=[],aggfunc=[np.sum, np.mean],fill_value=0)
        info.head(10)
```

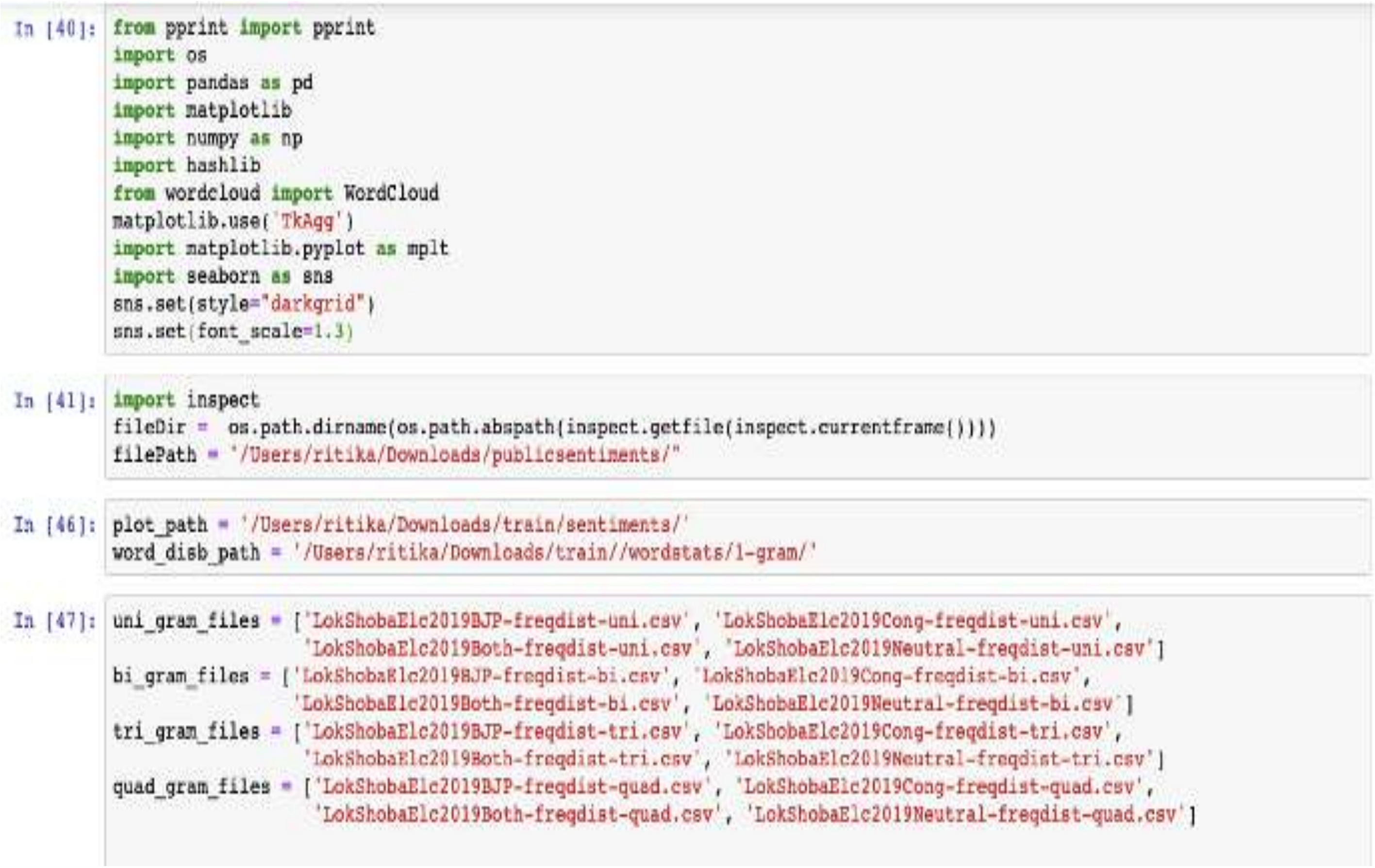


```
In [16]: import matplotlib.pyplot as plt
         ylabel = data_df["favourites_count"]
         plt.ylabel("favourites_count")
         plt.xlabel("retweet_count")
         xlabel = data_df["retweet_count"]
         plt.scatter(xlabel, ylabel, alpha=0.9)
         plt.show();
```

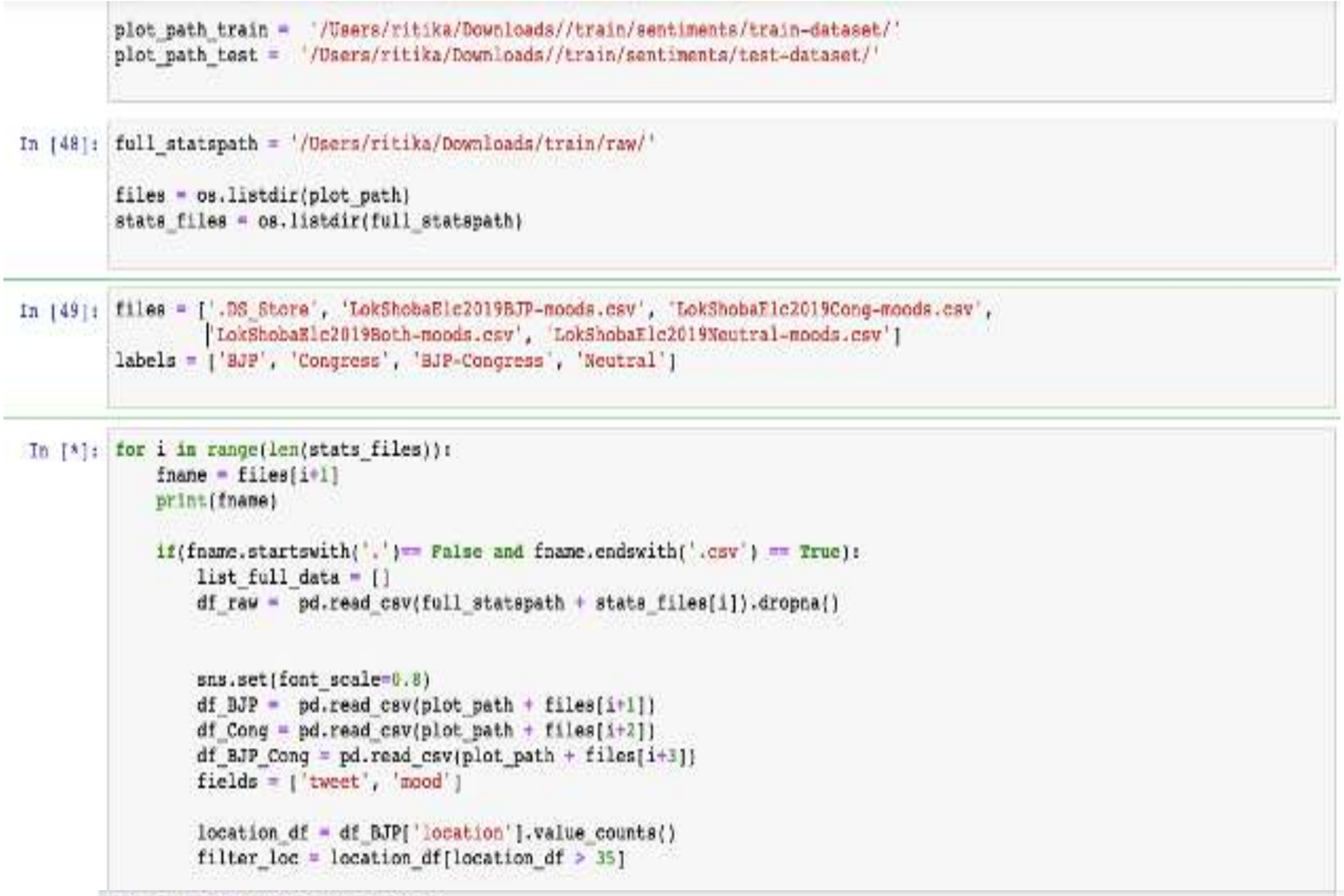
```
In [14]: import matplotlib.pyplot as plt
         ylabel = data_df["favourites_count"]
         plt.ylabel("favourites_count")
         plt.xlabel("followers_count")
         xlabel = data_df["followers_count"]
         plt.scatter(xlabel, ylabel, alpha=1)
         plt.show()
```

APPLICATION OF MACHINE LEARNING TO PRESENT SENTIMENT PLOTS OF THE LIVE STREAMING DATA

BJP MOOD ANALYSIS



```
In [40]: from pprint import pprint
         import os
         import pandas as pd
         import matplotlib
         import numpy as np
         import hashlib
         from wordcloud import WordCloud
         matplotlib.use('TkAgg')
         import matplotlib.pyplot as mplt
         import seaborn as sns
         sns.set(style="darkgrid")
         sns.set(font_scale=1.3)

In [41]: import inspect
         fileDir =  os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
         filePath = '/Users/ritika/Downloads/publicsentiments/"

In [46]: plot_path = '/Users/ritika/Downloads/train/sentiments/'
         word_disb_path = '/Users/ritika/Downloads/train//wordstats/1-gram/'

In [47]: uni_gram_files = ['LokShobaElc2019BJP-freqdist-uni.csv', 'LokShobaElc2019Cong-freqdist-uni.csv',
                           'LokShobaElc2019Both-freqdist-uni.csv', 'LokShobaElc2019Neutral-freqdist-uni.csv']
         bi_gram_files = ['LokShobaElc2019BJP-freqdist-bi.csv', 'LokShobaElc2019Cong-freqdist-bi.csv',
                          'LokShobaElc2019Both-freqdist-bi.csv', 'LokShobaElc2019Neutral-freqdist-bi.csv']
         tri_gram_files = ['LokShobaElc2019BJP-freqdist-tri.csv', 'LokShobaElc2019Cong-freqdist-tri.csv',
                           'LokShobaElc2019Both-freqdist-tri.csv', 'LokShobaElc2019Neutral-freqdist-tri.csv']
         quad_gram_files = ['LokShobaElc2019BJP-freqdist-quad.csv', 'LokShobaElc2019Cong-freqdist-quad.csv',
                            'LokShobaElc2019Both-freqdist-quad.csv', 'LokShobaElc2019Neutral-freqdist-quad.csv']
```



```
plot_path_train =  '/Users/ritika/Downloads//train/sentiments/train-dataset/'
plot_path_test =  '/Users/ritika/Downloads//train/sentiments/test-dataset/'
```

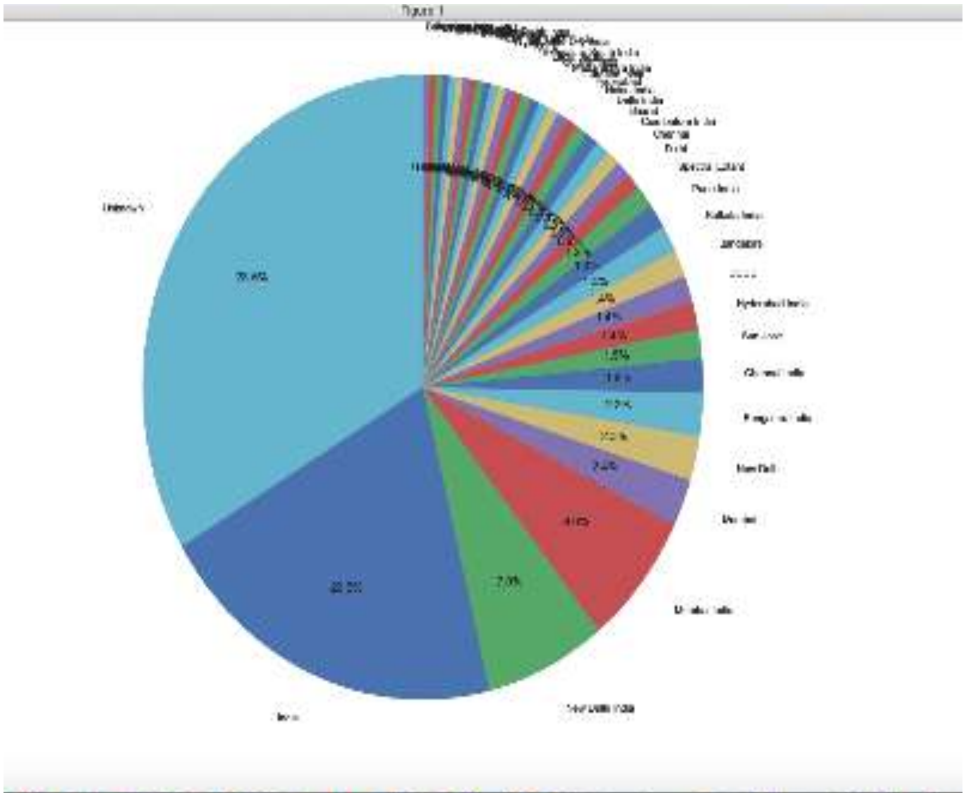
```
In [48]: full_statspath = '/Users/ritika/Downloads/train/raw/'

         files = os.listdir(plot_path)
         stats_files = os.listdir(full_statspath)
```

```
In [49]: files = ['.DS_Store', 'LokShobaElc2019BJP-moods.csv', 'LokShobaElc2019Cong-moods.csv',
                  'LokShobaElc2019Both-moods.csv', 'LokShobaElc2019Neutral-moods.csv']
         labels = ['BJP', 'Congress', 'BJP-Congress', 'Neutral']
```

```
In [*]: for i in range(len(stats_files)):
            fname = files[i+1]
            print(fname)

            if(fname.startswith('.')== False and fname.endswith('.csv') == True):
                list_full_data = []
                df_raw =  pd.read_csv(full_statspath + stats_files[i]).dropna()


                sns.set(font_scale=0.8)
                df_BJP =  pd.read_csv(plot_path + files[i+1])
                df_Cong = pd.read_csv(plot_path + files[i+2])
                df_BJP_Cong = pd.read_csv(plot_path + files[i+3])
                fields = ['tweet', 'mood']

                location_df = df_BJP['location'].value_counts()
                filter_loc = location_df[location_df > 35]
```

```
for i in range(len(stats_files)):
    fname = files[i+1]
    print(fname)

    if(fname.startswith('.')== False and fname.endswith('.csv') == True):
        list_full_data = []
        df_raw =  pd.read_csv(full_statspath + stats_files[i]).dropna()


        sns.set(font_scale=0.8)
        df_BJP =  pd.read_csv(plot_path + files[i+1])
        df_Cong = pd.read_csv(plot_path + files[i+2])
        df_BJP_Cong = pd.read_csv(plot_path + files[i+3])
        fields = ['tweet', 'mood']

        location_df = df_BJP['location'].value_counts()
        filter_loc = location_df[location_df > 35]

        location_df = df_BJP['location'].value_counts()
        filter_loc = location_df[location_df > 35]
        patches, texts, autotexts = mplt.pie(
            filter_loc,
            labels=filter_loc.index.values,
            shadow=False,
            startangle=90,
            pctdistance=0.7, labeldistance=1.15,
            # with the percent listed as a fraction
            autopct='%1.1f%%',
        )
        mplt.axis('equal')
        mplt.tight_layout()
        mplt.show();
```

```
LokShobaElc2019BJP-moods.csv
```



Graph 12.1: OUTPUT FOR BJP GEOGRAPHIC MAPPING

APPLICATION OF LOGISTIC REGRESSION AND MULTINOMIAL NAIVE BAYES

This included cleaning the content information, evacuating stop words and stemming. To deduce the tweets' feeling I utilized two classifiers: calculated regression and multinomial naive Bayes. I tuned the hyperparameters of the two classifiers with network search.

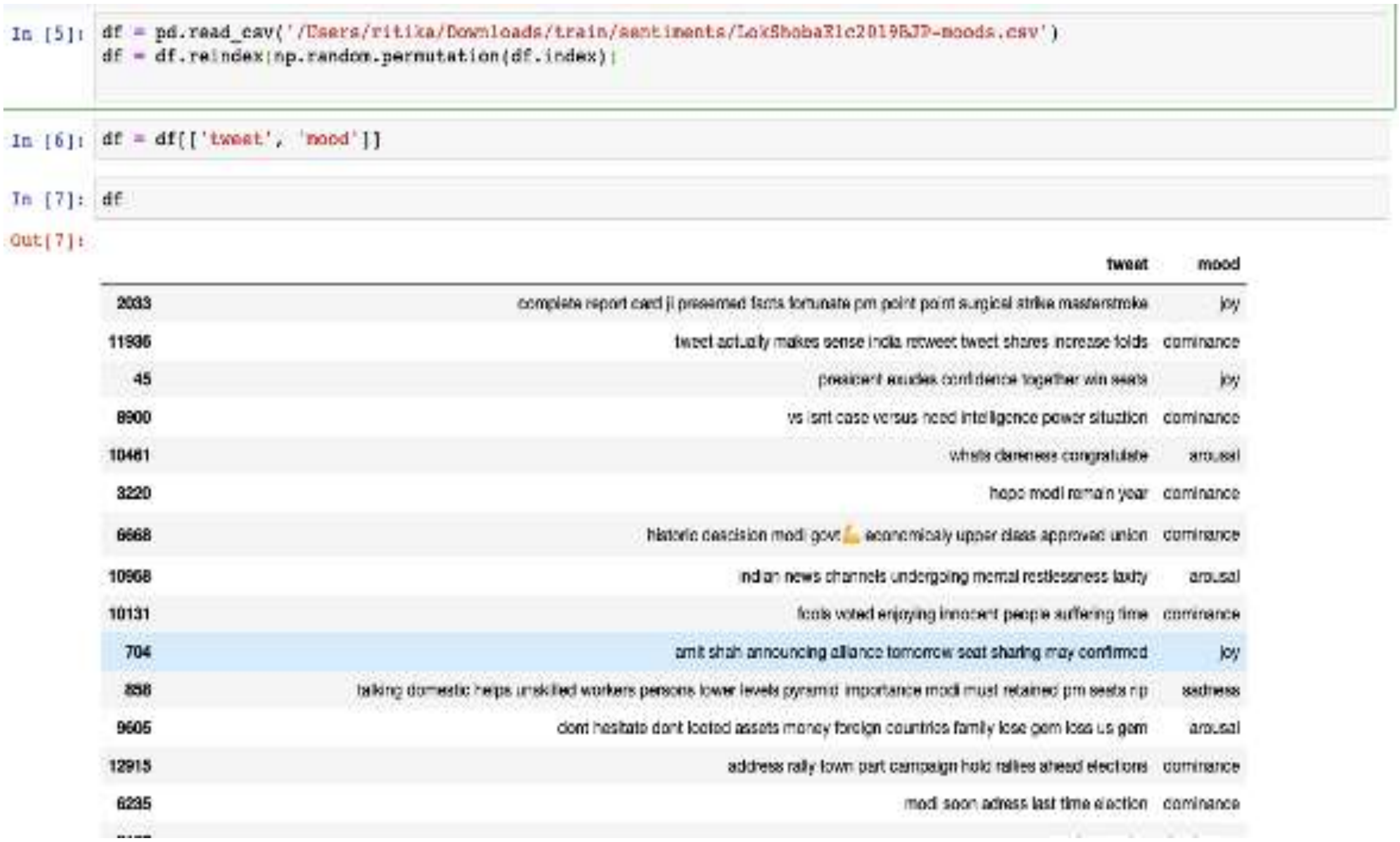
There are three class names that would be included in this section : negative, impartial or positive.

Code:

```
In [1]: import numpy as np
        import pandas as pd
        pd.set_option('display.max_colwidth', -1)
        from time import time
        import re
        import string
        import os
        import emoji
        from pprint import pprint
        import collections
```

```
In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
        sns.set(style="darkgrid")
        sns.set(font_scale=1.3)
        from sklearn.base import BaseEstimator, TransformerMixin
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.feature_extraction.text import TfidfVectorizer
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import train_test_split
        from sklearn.pipeline import Pipeline, FeatureUnion
        from sklearn.metrics import classification_report
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.linear_model import LogisticRegression
        from sklearn.externals import joblib
```

```
In [3]: import gensim
        from nltk.corpus import stopwords
        from nltk.stem import PorterStemmer
        from nltk.tokenize import word_tokenize
        import warnings
        warnings.filterwarnings('ignore')
        np.random.seed(37)
```



```
In [5]: df = pd.read_csv('/Users/ritika/Downloads/train/sentiments/LokShobaElc2019BJP-moods.csv')
        df = df.reindex(np.random.permutation(df.index))
```

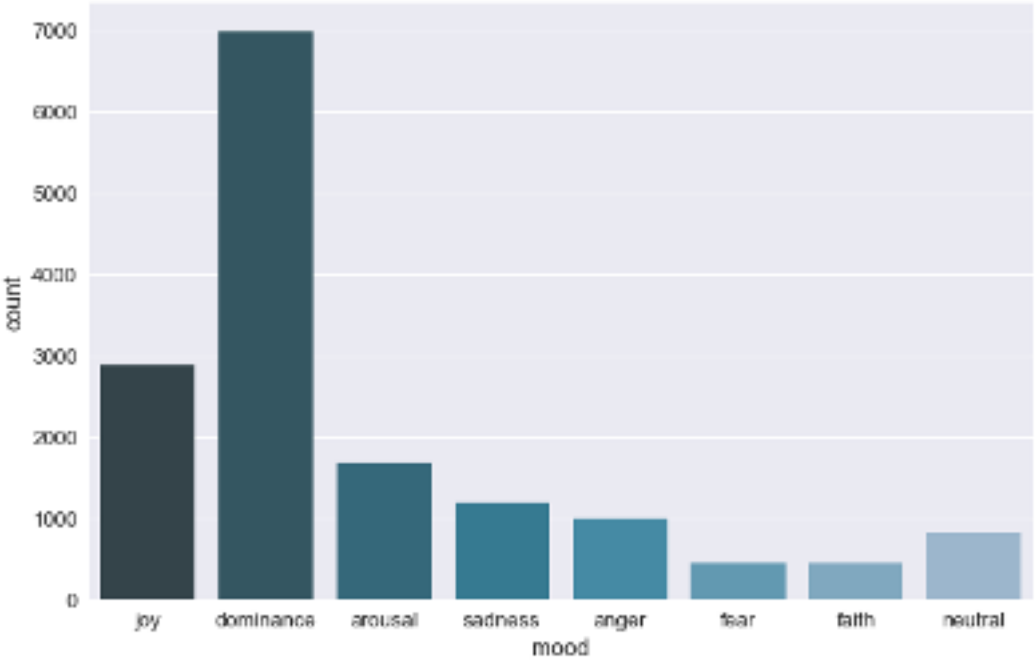
```
In [6]: df = df[['tweet', 'mood']]
```

```
In [7]: df
```

```
Out[7]:
```

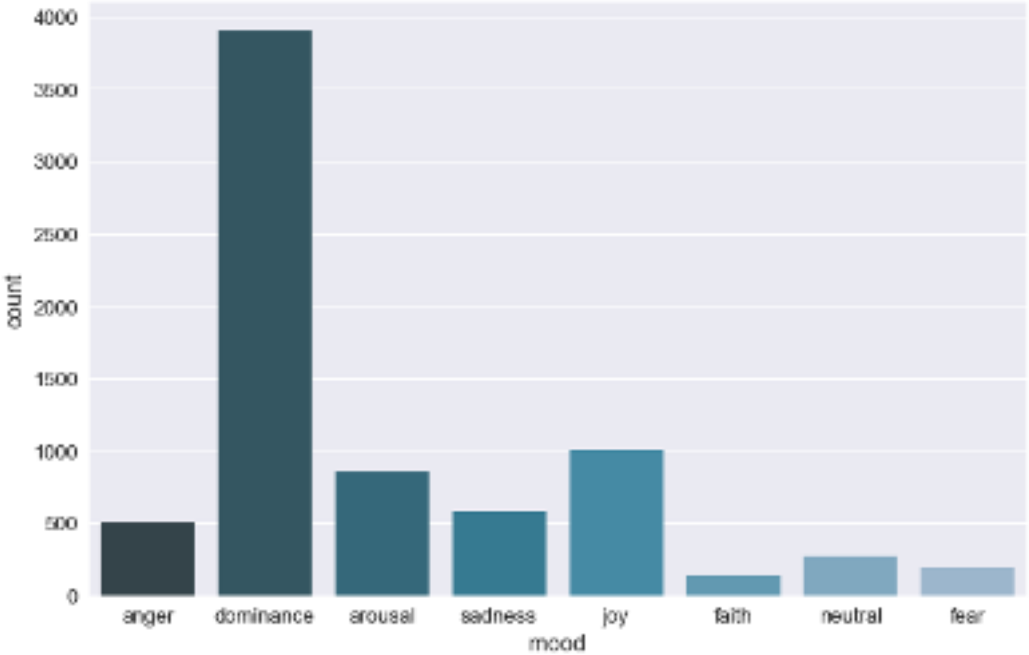
	tweet	mood
2033	complete report card ji presented facts fortunate pm point point surgical strike masterstroke	joy
11936	tweet actually makes sense india retweet tweet shares increase folds	dominance
45	president exudes confidence together win seats	joy
8900	vs isnt case versus need intelligence power situation	dominance
10461	whats dareness congratulate	arousal
3220	hope modi remain year	dominance
6668	historic descision modi govt💪 economicaly upper class approved union	dominance
10968	indian news channels undergoing mental restlessness laxity	arousal
10131	fools voted enjoying innocent people suffering time	dominance
704	amit shah announcing alliance tomorrow seat sharing may confirmed	joy
858	talking domestic helps unskilled workers persons lower levels pyramid importance modi must retained pm seats np	sadness
9605	dont hesitate dont looted assets money foreign countries family lose gem loss us gem	arousal
12915	address rally town part campaign hold rallies ahead elections	dominance
6235	modi soon adress last time election	dominance

RESULT FOR BJP



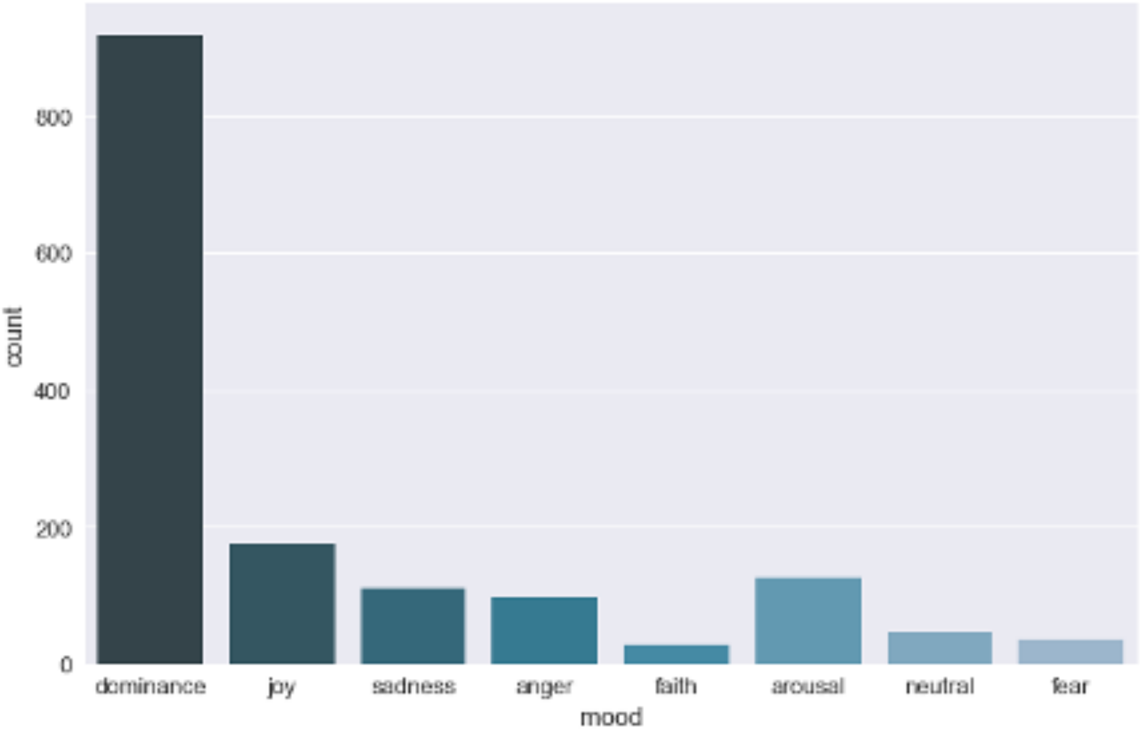
Graph 12.6: Graphical representation of moods for BJP

RESULT FOR CONGRESS



Graph 12.7: Graphical representation of moods for Congress

RESULT FOR BOTH PARTIES COMBINED



Graph 12.8: Graphical representation of moods for election data

PREDICTIVE ANALYTICS

The impulse to see Predictive Analytics as the magnet to draw voters can be very strong, given how various parties profit by it. Companies like Amazon utilize Big Data, information mining and AI to foresee what the client would purchase straightaway, and keep the stock inside delivery areas fully expecting the potential purchase. It can help basic leadership, expanding income and even procedure or item advancement.

Be that as it may, Big Data is anything but an enchantment issue solver for everything. For an organization to utilize analytics to foresee and envision patterns, must approach data in huge scale. This can be influenced by human conduct can be effectively influenced by the climate or by connections. A model that worked in the past may not work again on account of progress in human conduct. An intensive understanding, refined instruments and backing from upper dimension the executives is critical to the accomplishment of any system that chooses to utilize innovation for Big Data Analytics.

THE FUTURE OF DATA ANALYTICS IN ELECTIONS

The battle to get voters through innovation has the upside of two crucial patterns - ascent of the youthful populace and the advances in innovation. In India alone, over a 100 million new voters were added to the blend in 2014. A sizeable segment of this populace will connect with the web by and large and web based life intensely, making information that will be amazingly helpful for information examiners.

The youthful India is tied in with being urban, taught and hopeful and is equipped with the most recent advanced cells and rapid Internet network. This will prompt a monstrous information burst. It's solitary a matter of changing over such information into valuable and canny data to change the course of any business, or for this situation, constituent results.

Another sunrise has happened upon the manner in which races are battled. Amid the 2014 Indian races, the consumption on promoting efforts shot from the 2009 figure of $83 million to $300 million, with advanced showcasing being one reason. Experienced legislators who utilized their guts and senses to decide, will presently move towards innovation to settle on actuality based choices. Instructed specialists, software engineers and information researchers will enter constituent scenes, and gainfully in this way, what with the necessity for gifted experts talented in information examination!

Utilizing predictive analytics isn't just about exploiting to win the constituent fights. It's more than that. It's tied in with centering political endeavours to plan and manufacture their methodologies dependent on genuine open opinions. Legislators can now truly be a piece of individuals' lives each day. Advances in innovation can deliver the issues that truly matter to the general population. Along these lines Big Data and prescient examination can take races past political crusades to bring genuine change and win-win circumstances for entire countries.

PREDICTING SENTIMENT WITH TEXT FEATURES

• DATA LOADING

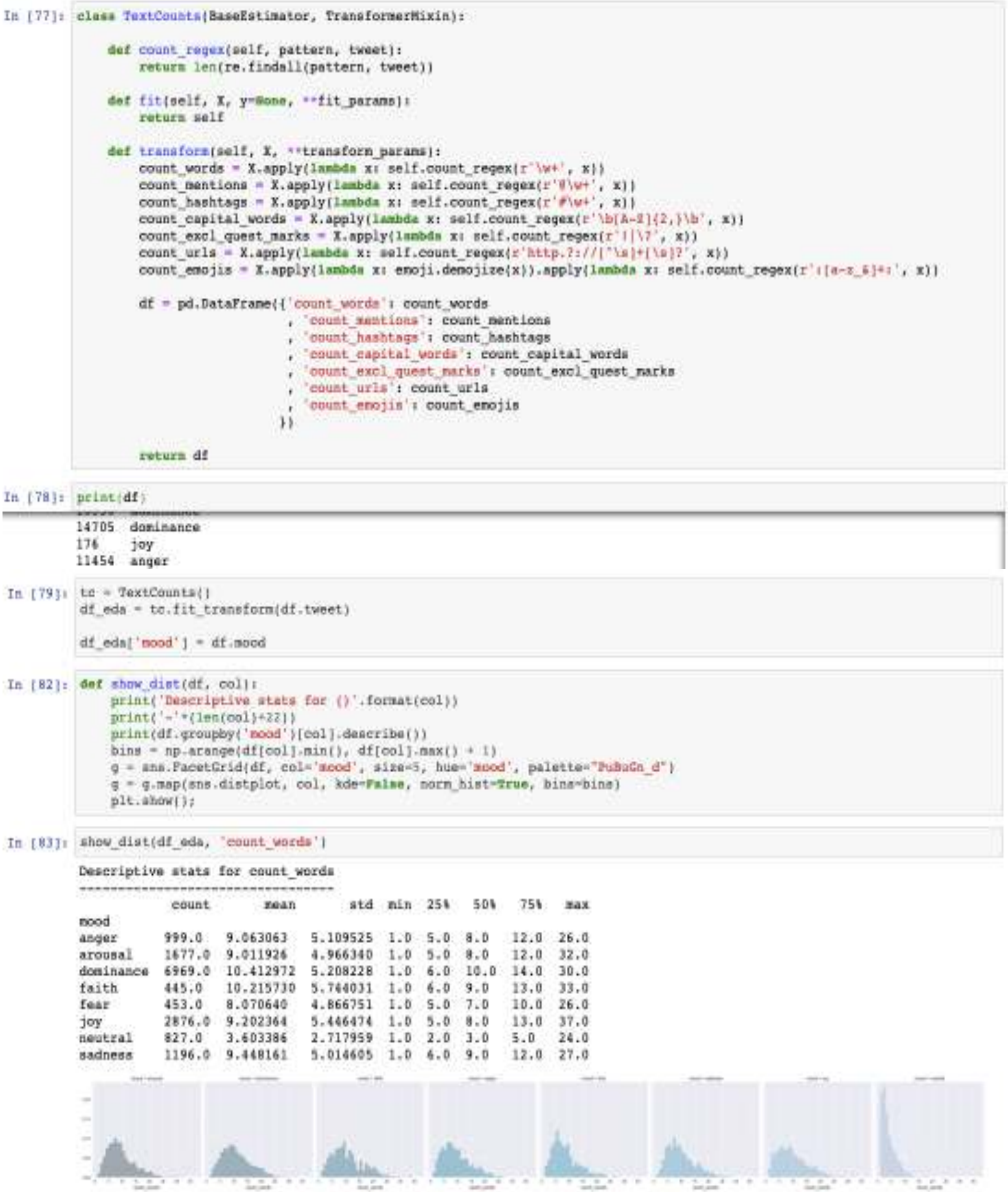
The reindexing method is applied to get a permutation of the original values

```
In [65]: df = pd.read_csv('/Users/ritika/Downloads/train/sentiments/LokShobaElc2019BJP-moods.csv')
         df = df.reindex(np.random.permutation(df.index))

In [67]: df = df[['tweet', 'mood']]
```

• ANALYSIS OF THE TEXTS IN THE DATA TO BE USED

To break down the content variable we make a class TextCounts. In this class we figure some fundamental measurements on the content variable. This class can be utilized later in a Pipeline, also.

1. **count_words** : number of words in the tweet
2. **count_mentions** : referrals to other Twitter accounts, which are preceded by a @
3. **count_hashtags** : number of tag words, preceded by a #
4. **count_capital_words** : number of uppercase words, could be used to "*shout*" and express (negative) emotions
5. **count_excl_quest_marks** : number of question or exclamation marks
6. **count_urls** : number of links in the tweet, preceded by http(s)
7. **count_emojis** : number of emoji, which might be a good indication of the sentiment



```
In [77]: class TextCounts(BaseEstimator, TransformerMixin):

             def count_regex(self, pattern, tweet):
                 return len(re.findall(pattern, tweet))

             def fit(self, X, y=None, **fit_params):
                 return self

             def transform(self, X, **transform_params):
                 count_words = X.apply(lambda x: self.count_regex(r'\w+', x))
                 count_mentions = X.apply(lambda x: self.count_regex(r'@\w+', x))
                 count_hashtags = X.apply(lambda x: self.count_regex(r'#\w+', x))
                 count_capital_words = X.apply(lambda x: self.count_regex(r'\b[A-Z]{2,}\b', x))
                 count_excl_quest_marks = X.apply(lambda x: self.count_regex(r'!|\?', x))
                 count_urls = X.apply(lambda x: self.count_regex(r'http.?://[^\s]+[\s]?', x))
                 count_emojis = X.apply(lambda x: emoji.demojize(x)).apply(lambda x: self.count_regex(r':[a-z_&]+:', x))

                 df = pd.DataFrame({'count_words': count_words
                                    , 'count_mentions': count_mentions
                                    , 'count_hashtags': count_hashtags
                                    , 'count_capital_words': count_capital_words
                                    , 'count_excl_quest_marks': count_excl_quest_marks
                                    , 'count_urls': count_urls
                                    , 'count_emojis': count_emojis
                                   })

                 return df
```

```
In [78]: print(df)
```

```
14705  dominance
176    joy
11454  anger
```

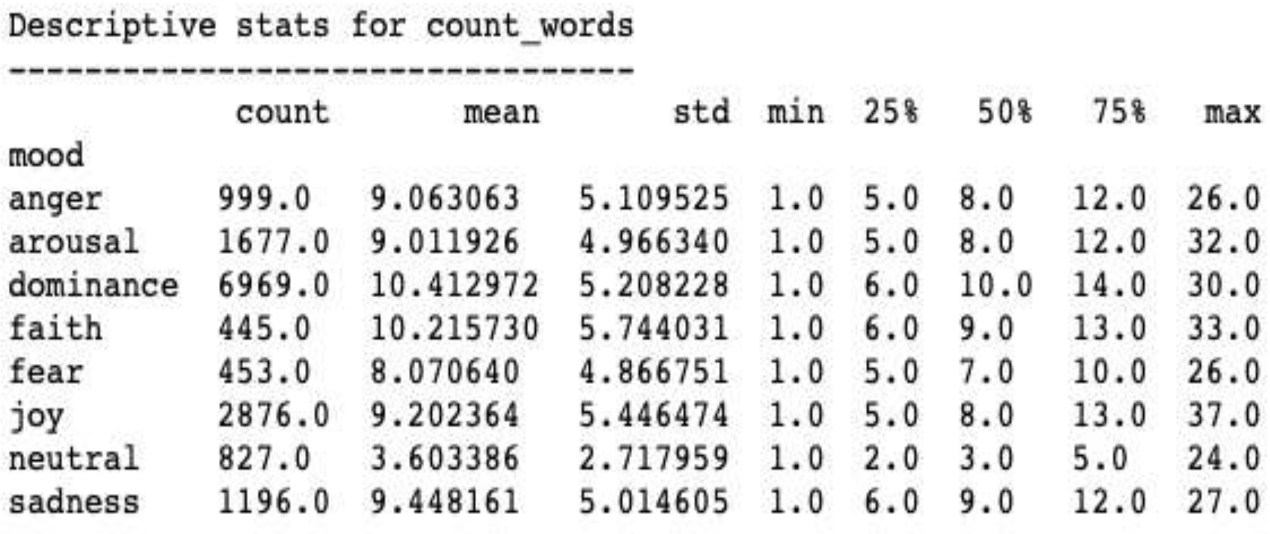
```
In [79]: tc = TextCounts()
         df_eda = tc.fit_transform(df.tweet)

         df_eda['mood'] = df.mood
```

```
In [82]: def show_dist(df, col):
             print('Descriptive stats for {}'.format(col))
             print('-'*(len(col)+22))
             print(df.groupby('mood')[col].describe())
             bins = np.arange(df[col].min(), df[col].max() + 1)
             g = sns.FacetGrid(df, col='mood', size=5, hue='mood', palette="PuBuGn_d")
             g = g.map(sns.distplot, col, kde=False, norm_hist=True, bins=bins)
             plt.show();
```

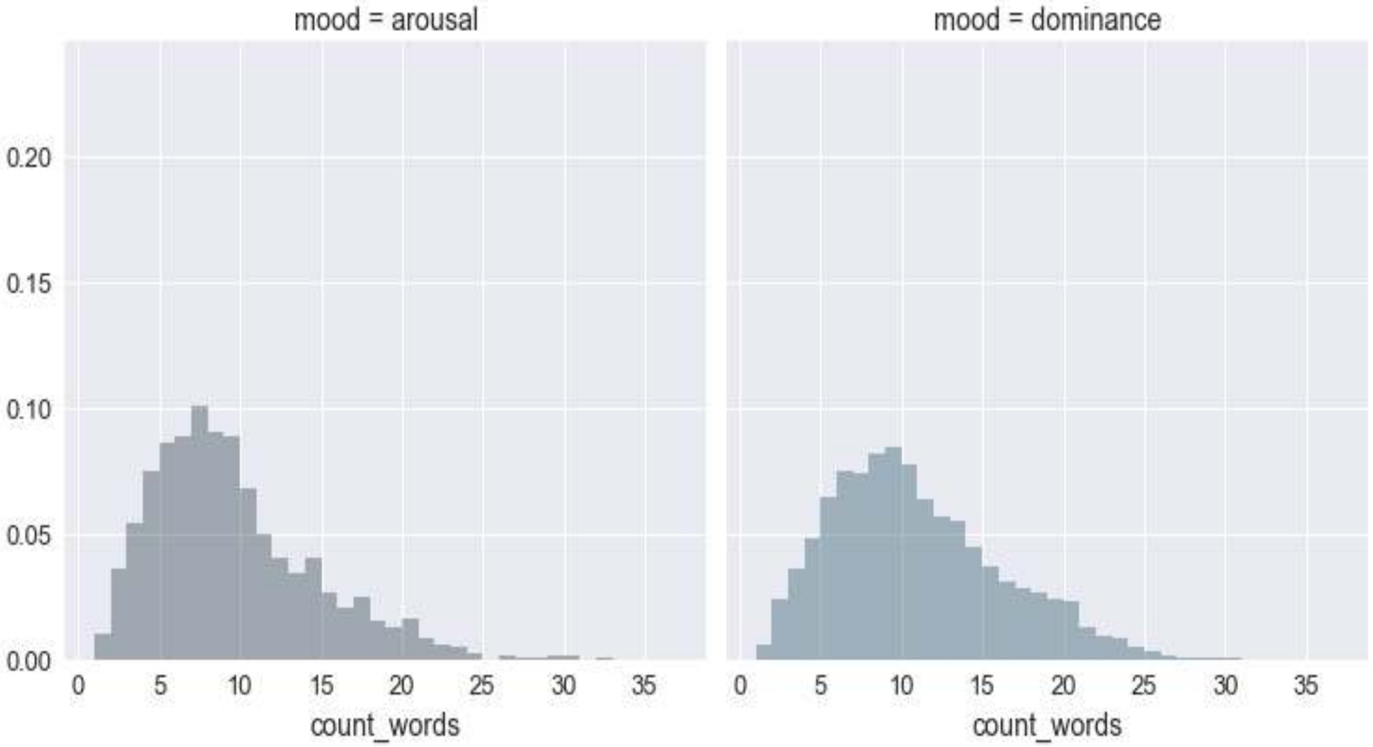
```
In [83]: show_dist(df_eda, 'count_words')
```

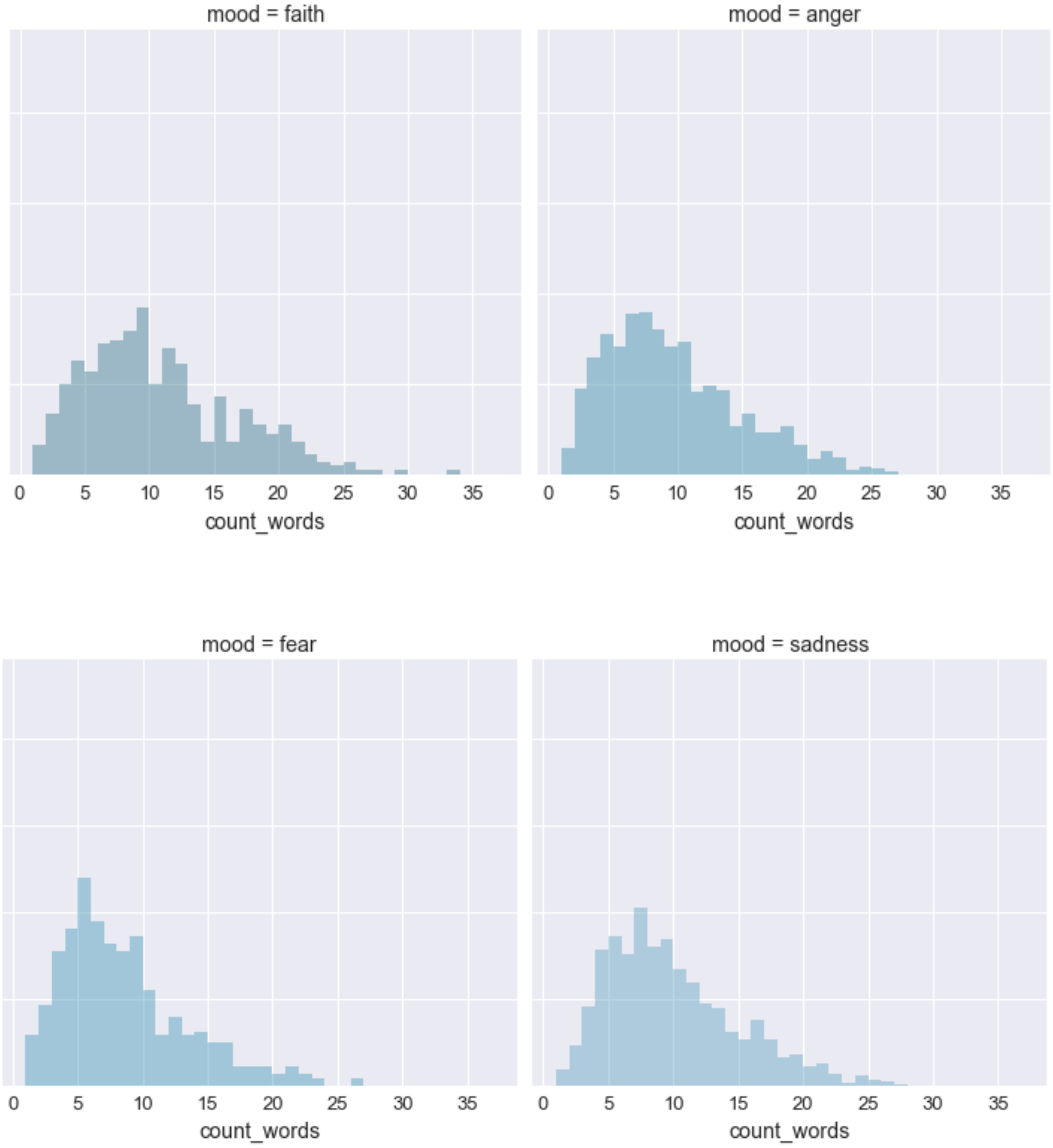
```
Descriptive stats for count_words
---------------------------------
            count       mean        std  min  25%   50%   75%   max
mood
anger       999.0   9.063063   5.109525  1.0  5.0   8.0  12.0  26.0
arousal    1677.0   9.011926   4.966340  1.0  5.0   8.0  12.0  32.0
dominance  6969.0  10.412972   5.208228  1.0  6.0  10.0  14.0  30.0
faith       445.0  10.215730   5.744031  1.0  6.0   9.0  13.0  33.0
fear        453.0   8.070640   4.866751  1.0  5.0   7.0  10.0  26.0
joy        2876.0   9.202364   5.446474  1.0  5.0   8.0  13.0  37.0
neutral     827.0   3.603386   2.717959  1.0  2.0   3.0   5.0  24.0
sadness    1196.0   9.448161   5.014605  1.0  6.0   9.0  12.0  27.0
```

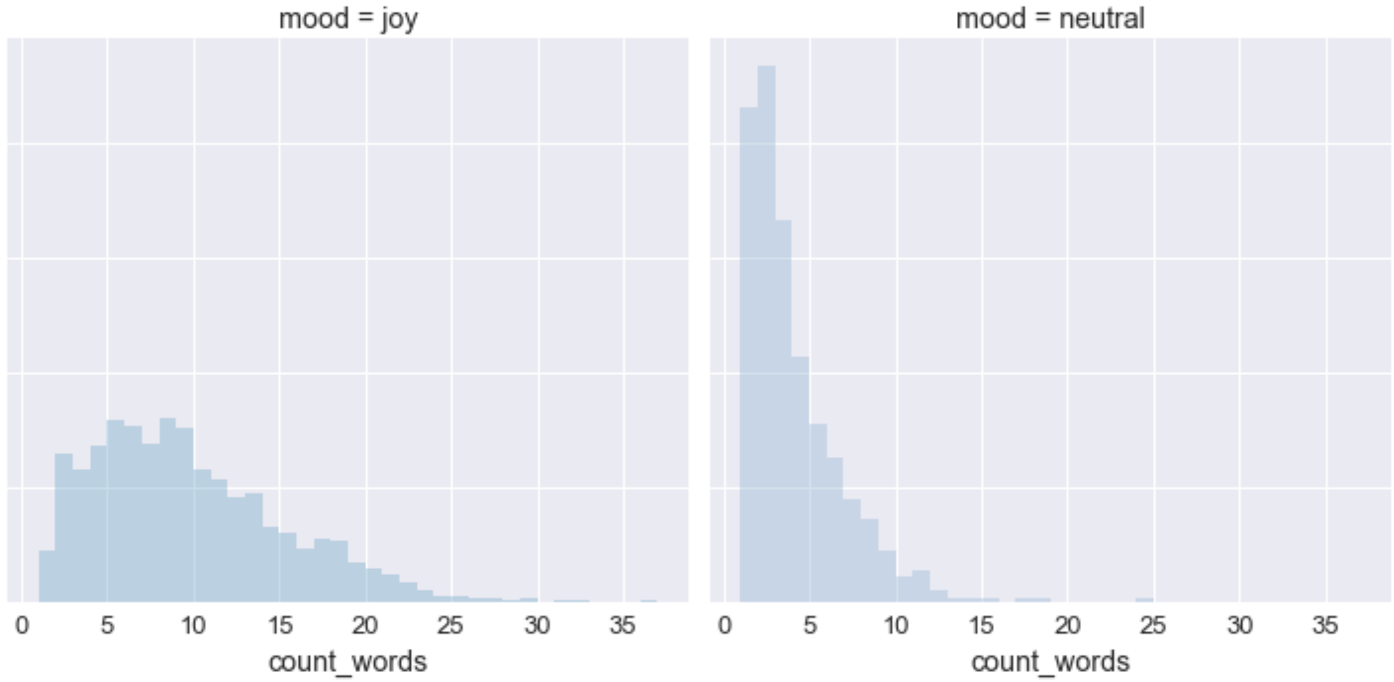
Output

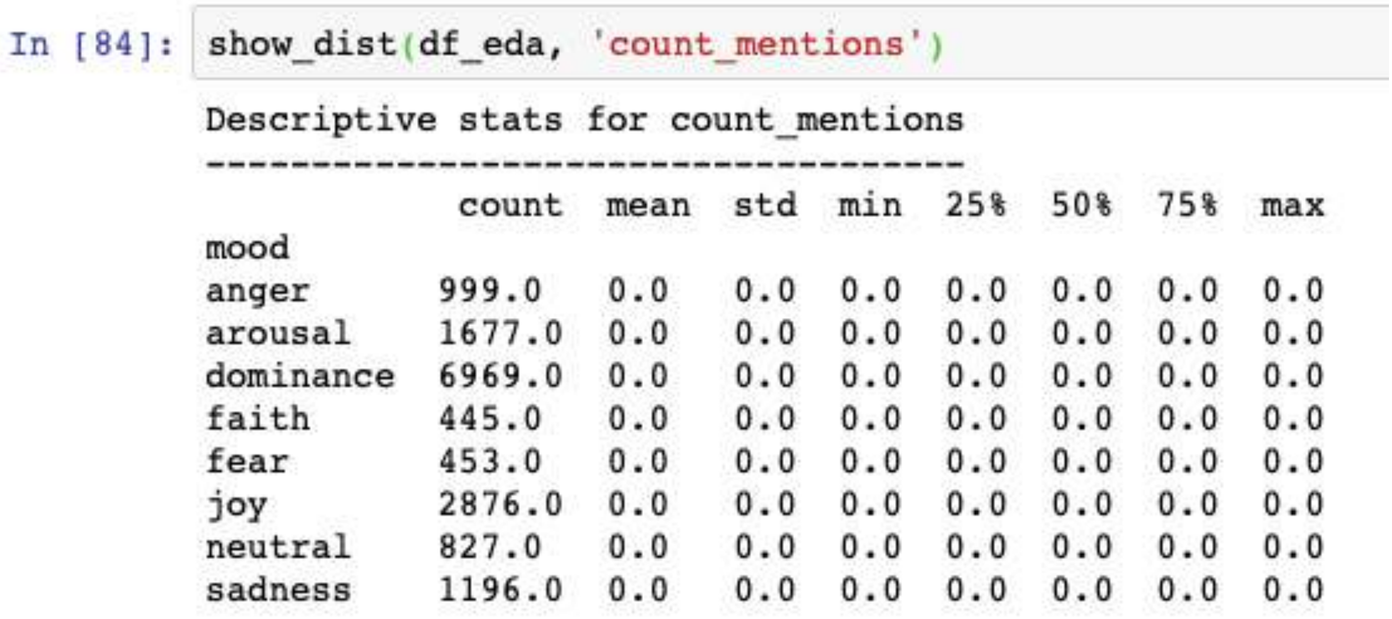


```
Descriptive stats for count_words
---------------------------------
           count       mean        std  min  25%   50%   75%   max
mood
anger      999.0   9.063063   5.109525  1.0  5.0  8.0   12.0  26.0
arousal    1677.0  9.011926   4.966340  1.0  5.0  8.0   12.0  32.0
dominance  6969.0  10.412972  5.208228  1.0  6.0  10.0  14.0  30.0
faith      445.0   10.215730  5.744031  1.0  6.0  9.0   13.0  33.0
fear       453.0   8.070640   4.866751  1.0  5.0  7.0   10.0  26.0
joy        2876.0  9.202364   5.446474  1.0  5.0  8.0   13.0  37.0
neutral    827.0   3.603386   2.717959  1.0  2.0  3.0   5.0   24.0
sadness    1196.0  9.448161   5.014605  1.0  6.0  9.0   12.0  27.0
```



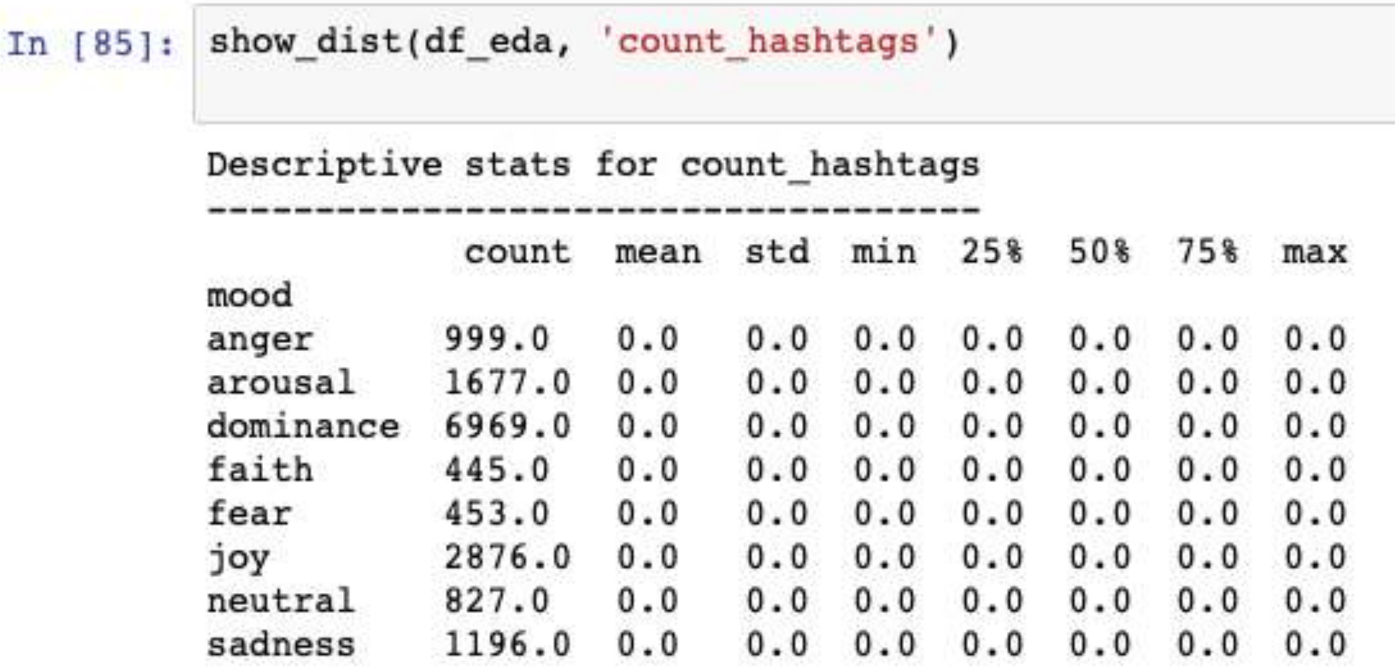

mood = faith
mood = anger
mood = fear
mood = sadness
0
5
10
15
20
25
30
35
count_words





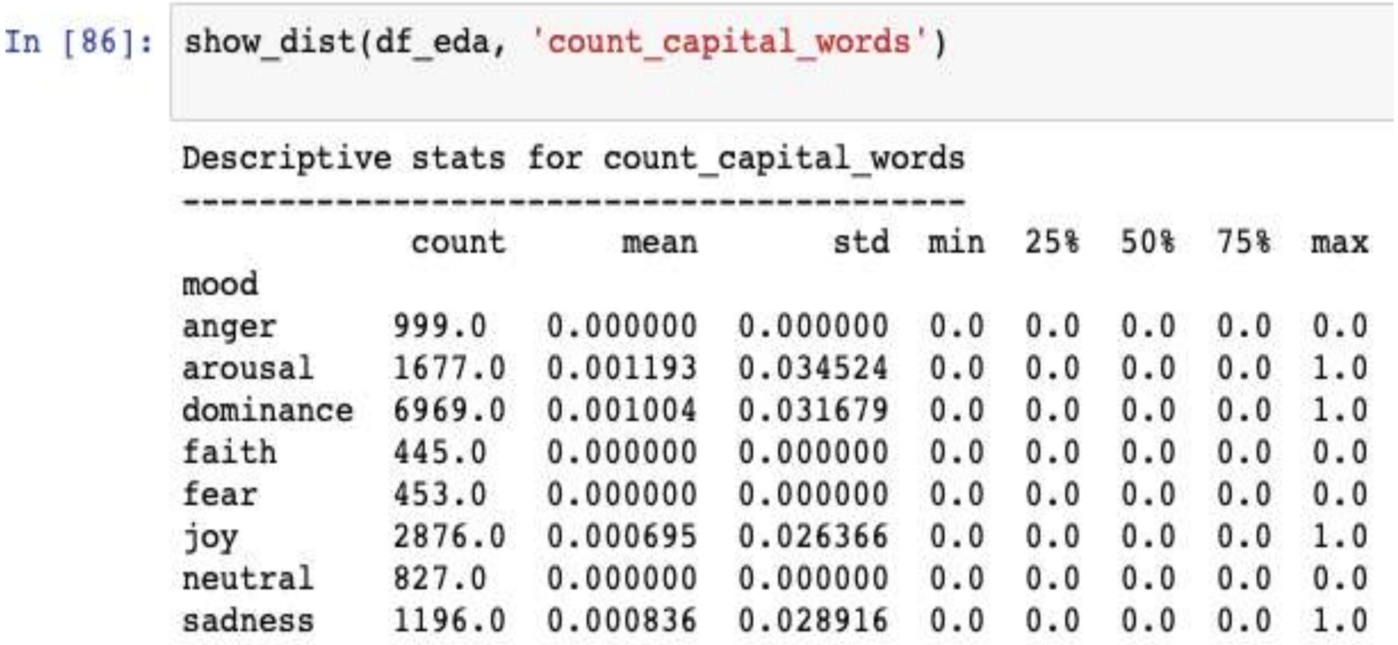
```
In [84]: show_dist(df_eda, 'count_mentions')
```

```
Descriptive stats for count_mentions
------------------------------------
            count   mean  std  min  25%  50%  75%  max
mood
anger       999.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
arousal     1677.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
dominance   6969.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
faith       445.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
fear        453.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
joy         2876.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
neutral     827.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
sadness     1196.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
```



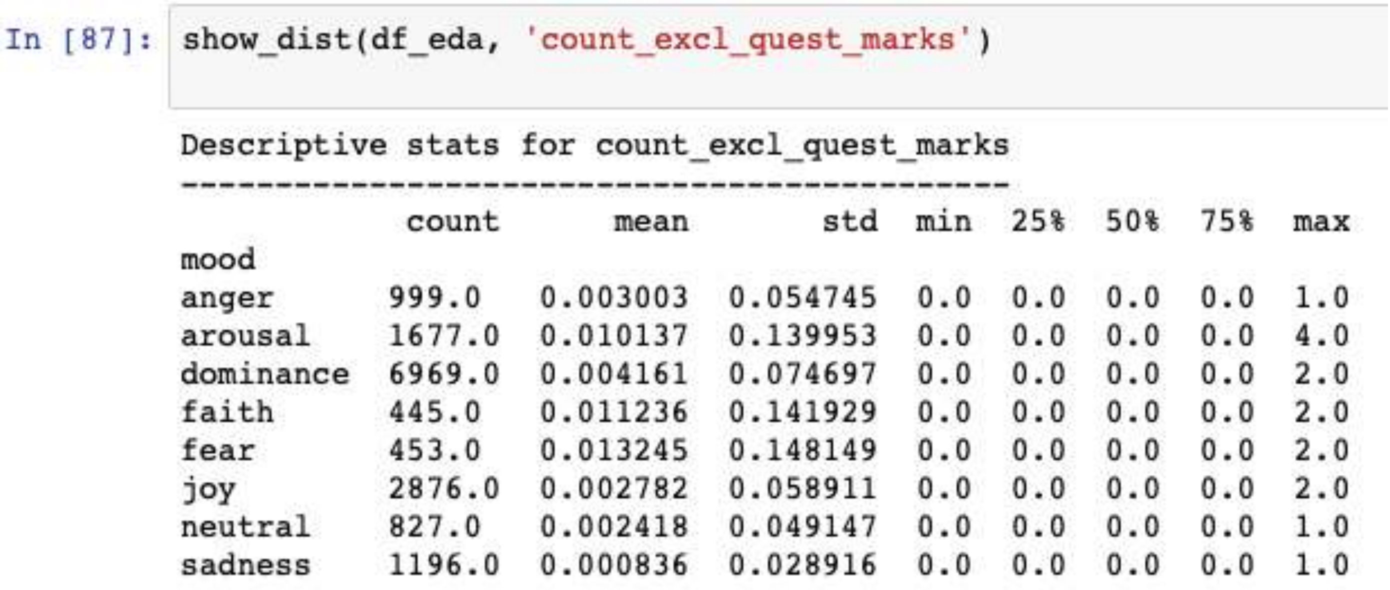
```
In [85]: show_dist(df_eda, 'count_hashtags')
```

```
Descriptive stats for count_hashtags
------------------------------------
            count   mean  std  min  25%  50%  75%  max
mood
anger       999.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
arousal     1677.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
dominance   6969.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
faith       445.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
fear        453.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
joy         2876.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
neutral     827.0   0.0   0.0  0.0  0.0  0.0  0.0  0.0
sadness     1196.0  0.0   0.0  0.0  0.0  0.0  0.0  0.0
```



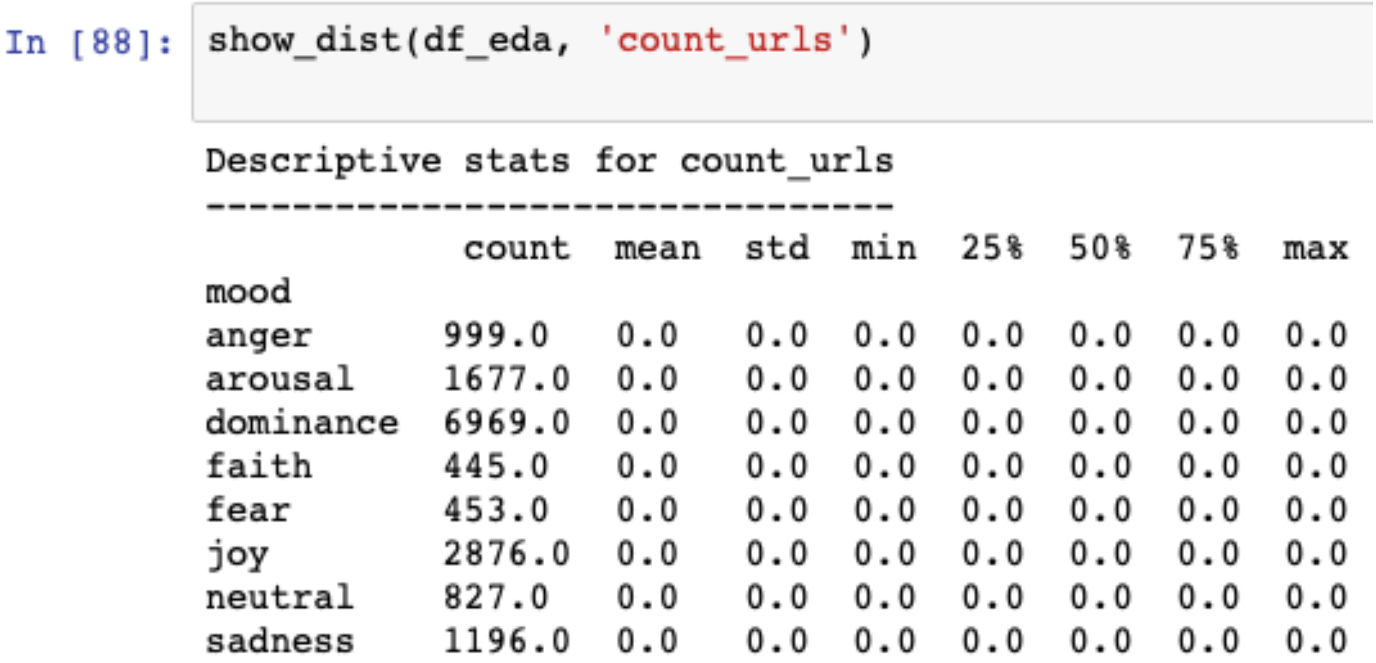
```
In [86]: show_dist(df_eda, 'count_capital_words')
```

```
Descriptive stats for count_capital_words
-----------------------------------------
            count      mean       std  min  25%  50%  75%  max
mood
anger       999.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
arousal    1677.0  0.001193  0.034524  0.0  0.0  0.0  0.0  1.0
dominance  6969.0  0.001004  0.031679  0.0  0.0  0.0  0.0  1.0
faith       445.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
fear        453.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
joy        2876.0  0.000695  0.026366  0.0  0.0  0.0  0.0  1.0
neutral     827.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
sadness    1196.0  0.000836  0.028916  0.0  0.0  0.0  0.0  1.0
```



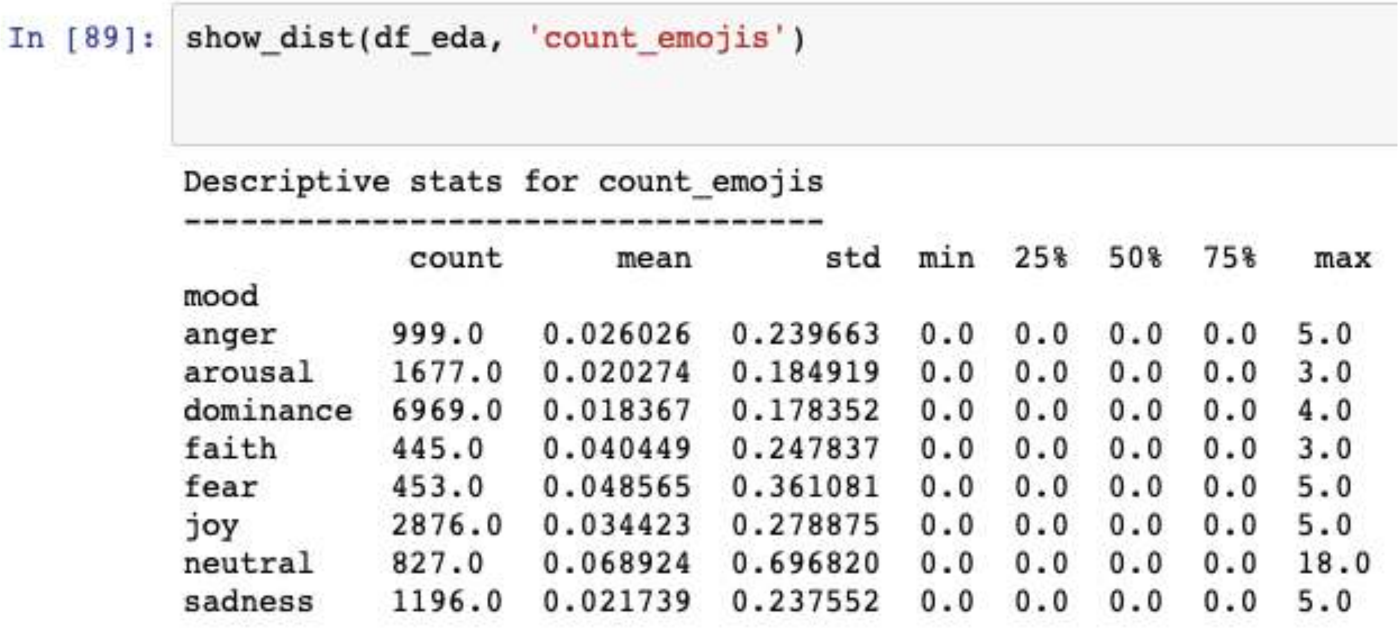
```
In [87]: show_dist(df_eda, 'count_excl_quest_marks')
```

```
Descriptive stats for count_excl_quest_marks
--------------------------------------------
            count      mean       std  min  25%  50%  75%  max
mood
anger       999.0  0.003003  0.054745  0.0  0.0  0.0  0.0  1.0
arousal    1677.0  0.010137  0.139953  0.0  0.0  0.0  0.0  4.0
dominance  6969.0  0.004161  0.074697  0.0  0.0  0.0  0.0  2.0
faith       445.0  0.011236  0.141929  0.0  0.0  0.0  0.0  2.0
fear        453.0  0.013245  0.148149  0.0  0.0  0.0  0.0  2.0
joy        2876.0  0.002782  0.058911  0.0  0.0  0.0  0.0  2.0
neutral     827.0  0.002418  0.049147  0.0  0.0  0.0  0.0  1.0
sadness    1196.0  0.000836  0.028916  0.0  0.0  0.0  0.0  1.0
```



```
In [88]: show_dist(df_eda, 'count_urls')
```

```
Descriptive stats for count_urls
--------------------------------
            count  mean  std  min  25%  50%  75%  max
mood
anger       999.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
arousal    1677.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
dominance  6969.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
faith       445.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
fear        453.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
joy        2876.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
neutral     827.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
sadness    1196.0   0.0  0.0  0.0  0.0  0.0  0.0  0.0
```



```
In [89]: show_dist(df_eda, 'count_emojis')
```

```
Descriptive stats for count_emojis
---------------------------------
            count      mean       std  min  25%  50%  75%   max
mood
anger       999.0  0.026026  0.239663  0.0  0.0  0.0  0.0   5.0
arousal    1677.0  0.020274  0.184919  0.0  0.0  0.0  0.0   3.0
dominance  6969.0  0.018367  0.178352  0.0  0.0  0.0  0.0   4.0
faith       445.0  0.040449  0.247837  0.0  0.0  0.0  0.0   3.0
fear        453.0  0.048565  0.361081  0.0  0.0  0.0  0.0   5.0
joy        2876.0  0.034423  0.278875  0.0  0.0  0.0  0.0   5.0
neutral     827.0  0.068924  0.696820  0.0  0.0  0.0  0.0  18.0
sadness    1196.0  0.021739  0.237552  0.0  0.0  0.0  0.0   5.0
```

CONCLUSION FROM THIS STAGE:

The quantity of words utilized in the tweets is rater low. Greatest number of words is 37 and there are even tweets with just 1 word. So, we'll must be cautious amid information cleaning not to evacuate such a large number of words. Then again, the content handling will be quicker. Negative tweets contain a bigger number of words than impartial or positive tweets.

All tweets have in any event one notice. Most likely this is the consequence of separating the tweets dependent on notices in the Twitter information. There is by all accounts no distinction in number of notices as to the opinion.

The majority of the tweets don't contain hash labels. So likely this variable won't be held amid model preparing. Once more, no distinction in number of hash labels as to the conclusion.

A large portion of the tweets don't contain uppercase words and we don't see a distinction in appropriation between the feelings. The positive tweets appear to utilize more outcry or question marks.

Most tweets don't contain a URL.

Most tweets don't utilize emoticons.

- **TEXT CLEANING**

Before we begin utilizing the tweets' content we clean it. We'll do the this in the class CleanText:

1. evacuate the mention, as we need to make the model generalisable to tweets of other parties for candidature as well.
2. expel the hash label sign (#) yet not the genuine tag as this may contain data
3. set all words to lowercase
4. evacuate all accentuations, including the inquiry and shout marks
5. evacuate the URLs as they don't contain valuable data and we didn't see a qualification in the quantity of URLs utilized between the assessment classes
6. ensure the changed over emoticons are kept as single word.
7. expel digits
8. expel stop words
9. apply the PorterStemmer to keep the stem of the words

```
In [94]: class CleanText(BaseEstimator, TransformerMixin):

             def remove_mentions(self, input_text):
                 return re.sub(r'@\w+', '', input_text)

             def remove_urls(self, input_text):
                 return re.sub(r'http.?://[^\s]+[\s]?', '', input_text)

             def emoji_oneword(self, input_text):
                 return input_text.replace('_','')

             def remove_punctuation(self, input_text):
                 punct = string.punctuation
                 trantab = str.maketrans(punct, len(punct)*' ')
                 return input_text.translate(trantab)

             def remove_digits(self, input_text):
                 return re.sub('\d+', '', input_text)

             def to_lower(self, input_text):
                 return input_text.lower()

             def remove_stopwords(self, input_text):
                 stopwords_list = stopwords.words('english')
                 whitelist = ["n't", "not", "no"]
                 words = input_text.split()
                 clean_words = [word for word in words if (word not in stopwords_list or word in whitelist) and len(word) > 1]
                 return " ".join(clean_words)

             def stemming(self, input_text):
                 porter = PorterStemmer()
                 words = input_text.split()
                 stemmed_words = [porter.stem(word) for word in words]
                 return " ".join(stemmed_words)

             def fit(self, X, y=None, **fit_params):
                 return self

             def transform(self, X, **transform_params):
                 clean_X = X.apply(self.remove_mentions).apply(self.remove_urls).apply(self.emoji_oneword).apply(self.remove_pu
         nctuation).apply(self.remove_digits).apply(self.to_lower).apply(self.remove_stopwords).apply(self.stemming)
                 return clean_X
```

```
In [98]: import nltk
         nltk.download('stopwords')
         from nltk.corpus import stopwords
         stopwords.words('english')

         [nltk_data] Downloading package stopwords to
         [nltk_data]     /Users/ritika/nltk_data...
         [nltk_data]   Unzipping corpora/stopwords.zip.

Out[98]: ['i',
          'me',
          'my',
          'myself',
          'we',
          'our',
          'ours',
          'ourselves',
          'you',
          "you're",
          "you've",
          "you'll",
          "you'd",
          'your',
          'yours',
```

```
In [99]: ct = CleanText()
         sr_clean = ct.fit_transform(df.tweet)
         sr_clean.sample(5)

Out[99]: 5748     get clearanc lesson applaud support esp hon word learn sth
         13626    lie wont work
         12952    wait..
         317      take holi dip
         58       agre share cm post allianc shiv sena minist
         Name: tweet, dtype: object
```

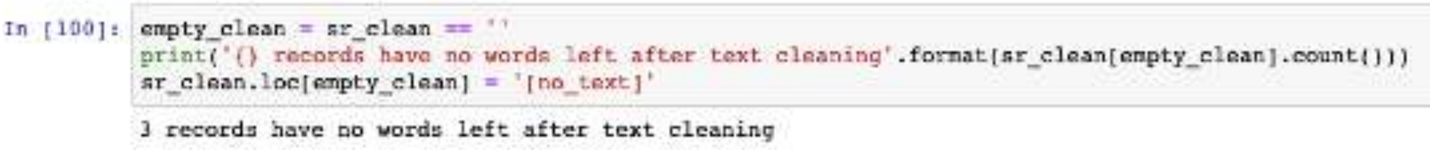
One negative impact of content cleaning is that a few lines don't have any words left in their content. For the CountTheVectorizer and TfIdfUniqueVectorizer this does not so much represent an issue. Be that as it may, for the Word2Vec calculation this causes a mistake. There are various procedures that you could apply to manage these missing qualities.

1. Evacuate the total line, however in a generation situation this isn't generally alluring.

2. Ascribe the missing an incentive with some placeholder content like [no text]

3. Word2Vec: utilize the normal all things considered

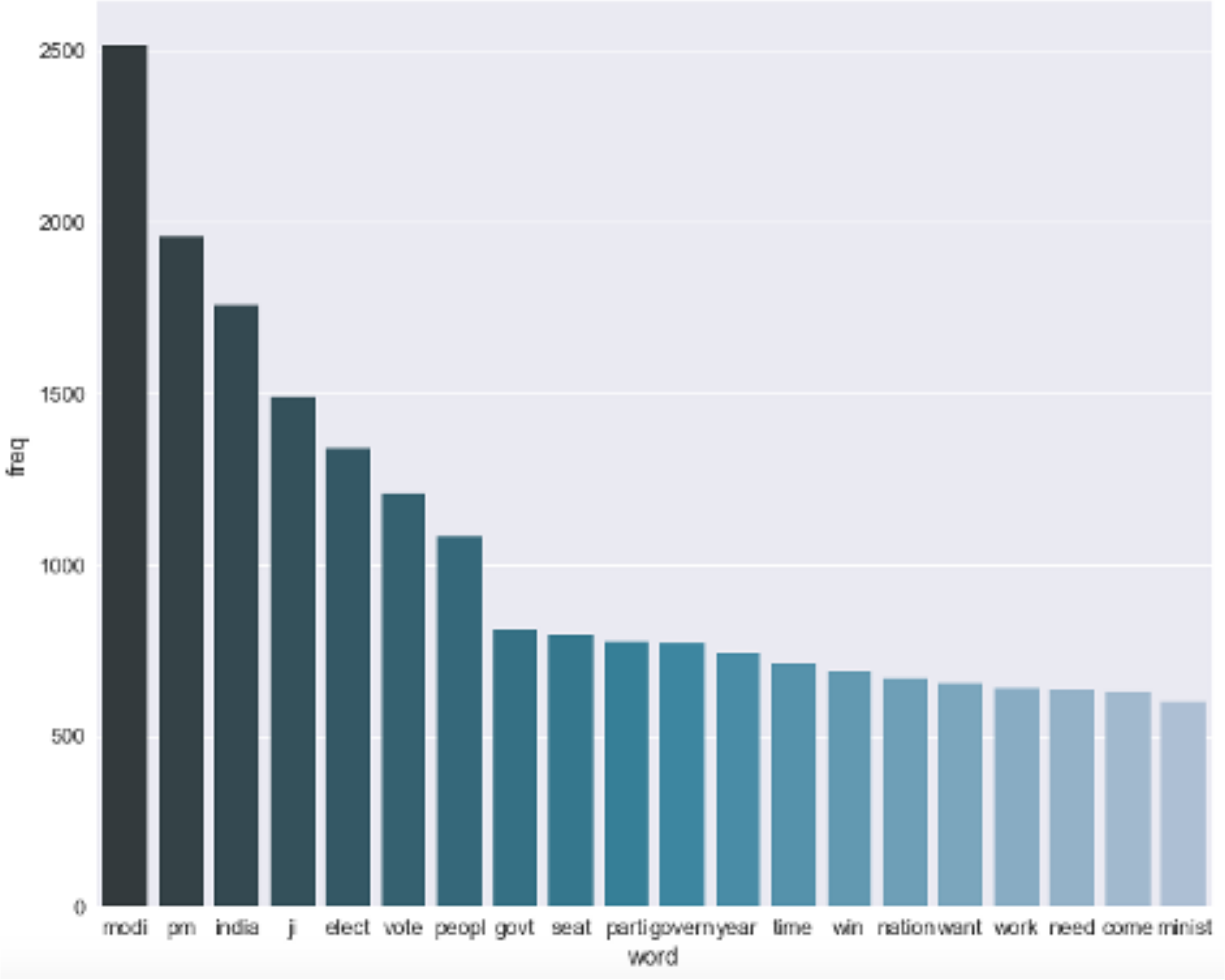
4. Here we will ascribe with a placeholder content.



```
In [100]: empty_clean = sr_clean == ''
          print('{} records have no words left after text cleaning'.format(sr_clean[empty_clean].count()))
          sr_clean.loc[empty_clean] = '[no_text]'

          3 records have no words left after text cleaning
```

- **FINDING THE FREQUENCY OF WORDS UPON DATA CLEANING**

```
In [101]: cv = CountVectorizer()
          bow = cv.fit_transform(sr_clean)
          word_freq = dict(zip(cv.get_feature_names(), np.asarray(bow.sum(axis=0)).ravel()))
          word_counter = collections.Counter(word_freq)
          word_counter_df = pd.DataFrame(word_counter.most_common(20), columns = ['word', 'freq'])

          fig, ax = plt.subplots(figsize=(12, 10))
          bar_freq_word = sns.barplot(x="word", y='freq', data=word_counter_df, palette="PuBuGn_d", ax=ax)
          plt.show();
```



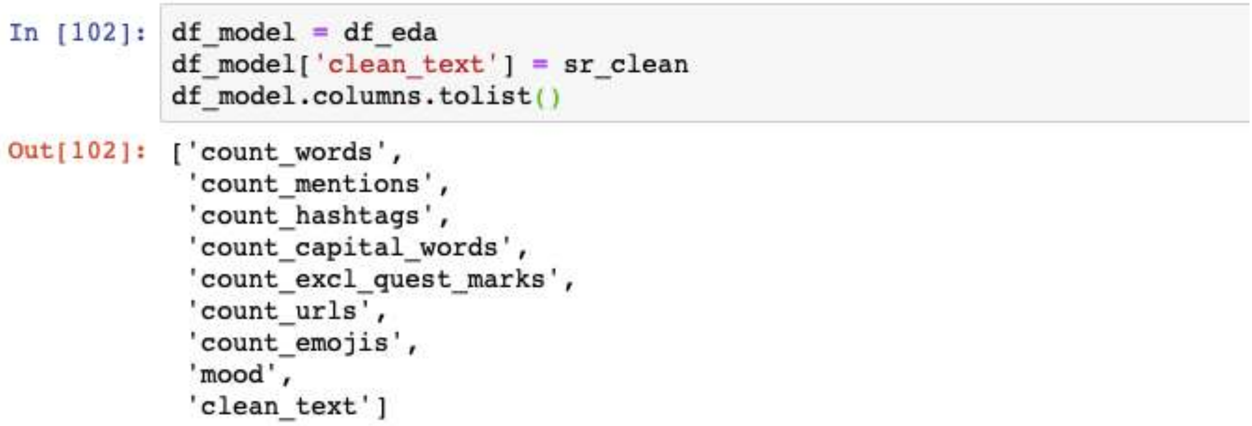
Graph 12.9: Graphical representation of frequency of keyword occurrence after cleaning

- **CREATING TEST DATA**

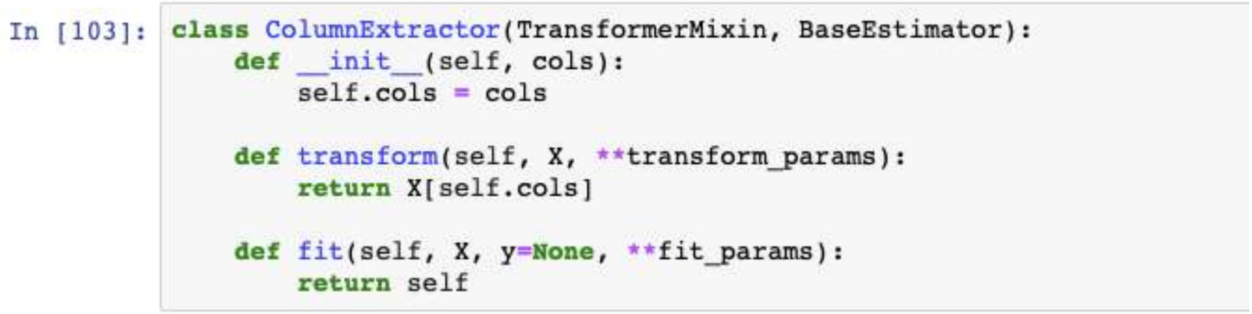
To assess the prepared models, it is required a test set. Assessing on the train information would not be right on the grounds that the models are prepared to limit their cost capacity.

Firs it is necessary to consolidate the TextCounts factors with the CleanText variable.

NOTE: Initially, I committed the error to do execute TextCounts and CleanText in the GridSearchCV underneath. This took excessively long as it applies these capacities each keep running of the GridSearch. It gets the job done to run them just once.

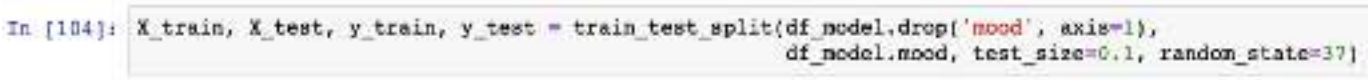


```
In [102]: df_model = df_eda
          df_model['clean_text'] = sr_clean
          df_model.columns.tolist()

Out[102]: ['count_words',
            'count_mentions',
            'count_hashtags',
            'count_capital_words',
            'count_excl_quest_marks',
            'count_urls',
            'count_emojis',
            'mood',
            'clean_text']
```



```
In [103]: class ColumnExtractor(TransformerMixin, BaseEstimator):
              def __init__(self, cols):
                  self.cols = cols

              def transform(self, X, **transform_params):
                  return X[self.cols]

              def fit(self, X, y=None, **fit_params):
                  return self
```

So df_model now contains a few factors. Be that as it may, our vectorizers will just need the clean_text variable. The TextCounts factors can be included all things considered. To explicitly choose sections, I composed the class ColumnExtractor underneath. This can be utilized in the Pipeline subsequently.



```
In [104]: X_train, X_test, y_train, y_test = train_test_split(df_model.drop('mood', axis=1),
                                                              df_model.mood, test_size=0.1, random_state=37)
```

- **HYPERPARAMETER TUNING AND CROSS-VALIDATION**

The vectorizers and classifiers all have configurable parameters. So as to picked the best parameters, we have to assess on a different approval set that was not utilized amid the preparation. Nonetheless, utilizing just a single approval set may not deliver solid approval results. Because of chance you may have a decent model presentation on the approval set. In the event that you would part the information else, you may finish up with different outcomes. To get a progressively precise estimation, we perform cross-approval.

With cross-approval the information is part into a train and approval set on numerous occasions. The assessment metric is then arrived at the midpoint of over the various folds. Fortunately, GridSearchCV applies cross-approval out-of-the-crate.

To locate the best parameters for both a vectorizer and classifier, we make a Pipeline. This is put into a capacity for usability. As a matter of course GridSearchCV utilizes the default scorer to figure the best score. For both the MultiNomialNb and LogisticRegression this default scoring metric is the exactness.

In our capacity grid_vect we moreover create the classification_report on the test information. This gives some fascinating measurements per target class, which may be increasingly suitable here. These measurements are the accuracy, recal and F1 score.

Accuracy: Of all columns we anticipated to be a sure class, what number of did we accurately foresee?
Review: Of all lines of a specific class, what number of did we effectively anticipate?

F1 score: Harmonic mean of Precision and Recall

```
In [105]: def grid_vect(clf, parameters_clf, X_train, X_test, parameters_text=None, vect=None, is_w2v=False):

    textcountscols = ['count_capital_words','count_emojis','count_excl_quest_marks','count_hashtags'
                      ,'count_mentions','count_urls','count_words']

    if is_w2v:
        w2vcols = []
        for i in range(SIZE):
            w2vcols.append(i)
        features = FeatureUnion([('textcounts', ColumnExtractor(cols=textcountscols))
                                , ('w2v', ColumnExtractor(cols=w2vcols))]
                               , n_jobs=-1)
    else:
        features = FeatureUnion([('textcounts', ColumnExtractor(cols=textcountscols))
                                , ('pipe', Pipeline([('cleantext', ColumnExtractor(cols='clean_text')), ('vect', vect
)]))]
                               , n_jobs=-1)


    pipeline = Pipeline([
        ('features', features)
        , ('clf', clf)
    ])

    # Join the parameters dictionaries together
    parameters = dict()
    if parameters_text:
        parameters.update(parameters_text)
    parameters.update(parameters_clf)

    # Make sure you have scikit-learn version 0.19 or higher to use multiple scoring metrics
    grid_search = GridSearchCV(pipeline, parameters, n_jobs=-1, verbose=1, cv=5)

    print("Performing grid search...")
    print("pipeline:", [name for name, _ in pipeline.steps])
    print("parameters:")
    pprint(parameters)

    t0 = time()
    grid_search.fit(X_train, y_train)
    print("done in %0.3fs" % (time() - t0))
    print()

    print("Best CV score: %0.3f" % grid_search.best_score_)
    print("Best parameters set:")
    best_parameters = grid_search.best_estimator_.get_params()
    for param_name in sorted(parameters.keys()):
        print("\t%s: %r" % (param_name, best_parameters[param_name]))
```

```
    print("Performing grid search...")
    print("pipeline:", [name for name, _ in pipeline.steps])
    print("parameters:")
    pprint(parameters)

    t0 = time()
    grid_search.fit(X_train, y_train)
    print("done in %0.3fs" % (time() - t0))
    print()

    print("Best CV score: %0.3f" % grid_search.best_score_)
    print("Best parameters set:")
    best_parameters = grid_search.best_estimator_.get_params()
    for param_name in sorted(parameters.keys()):
        print("\t%s: %r" % (param_name, best_parameters[param_name]))

    print("Test score with best_estimator_: %0.3f" % grid_search.best_estimator_.score(X_test, y_test))
    print("\n")
    print("Classification Report Test Data")
    print(classification_report(y_test, grid_search.best_estimator_.predict(X_test)))

    return grid_search
```

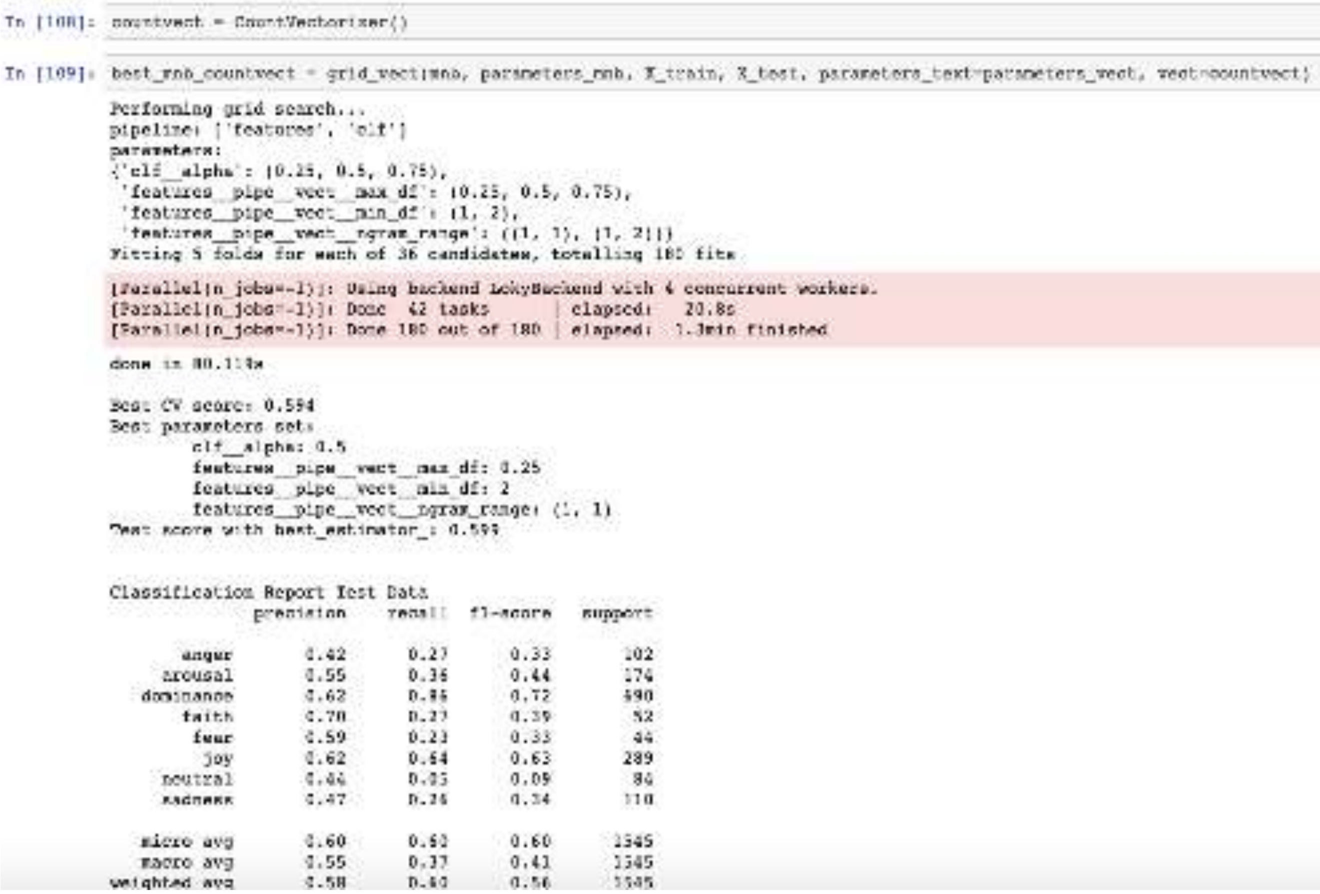
```
In [106]: parameters_vect = {
              'features__pipe__vect__max_df': (0.25, 0.5, 0.75),
              'features__pipe__vect__ngram_range': ((1, 1), (1, 2)),
              'features__pipe__vect__min_df': (1,2)
          }


          # Parameter grid settings for MultinomialNB
          parameters_mnb = {
              'clf__alpha': (0.25, 0.5, 0.75)
          }


          # Parameter grid settings for LogisticRegression
          parameters_logreg = {
              'clf__C': (0.25, 0.5, 1.0),
              'clf__penalty': ('l1', 'l2')
          }
```

- **CLASSIFIERS**

This is in order to derive a comparison of performance of a MultiNomial NB and Logistic Regression.

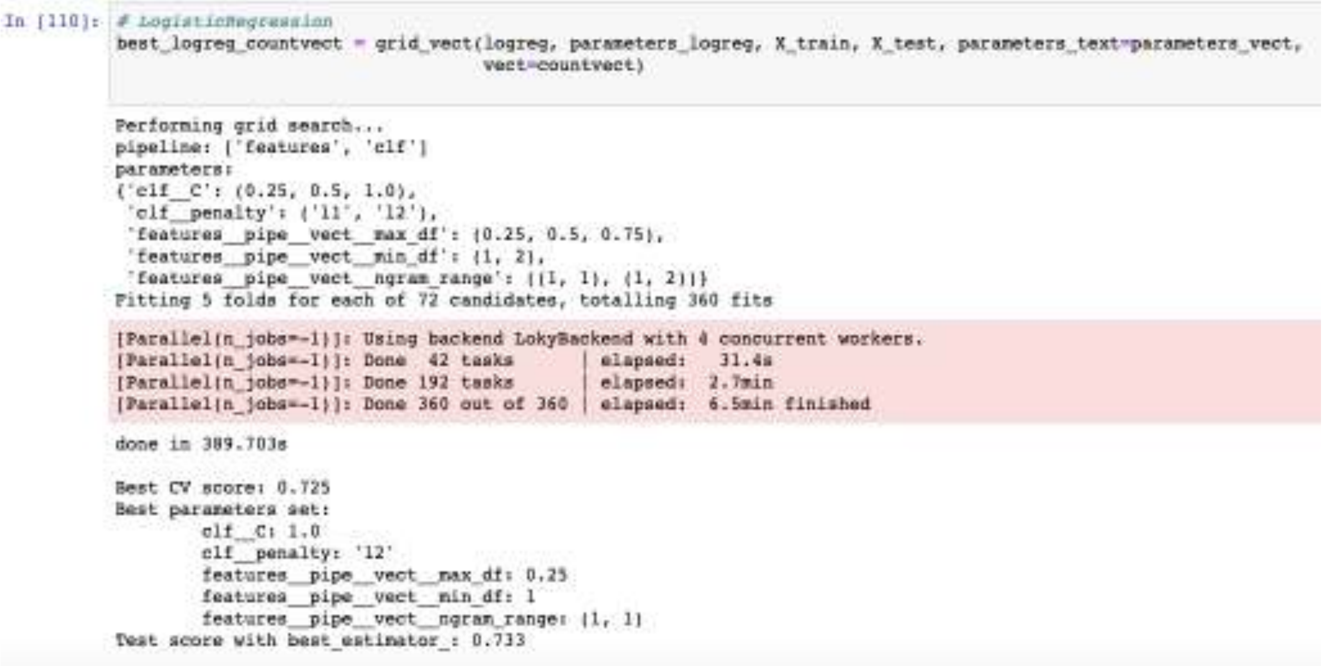
- **COUNT VECTORIZER**

To utilize words in a classifier, we have to change over the words to numbers. This should be possible with a CountVectorizer. Sklearn's CountVectorizer takes all words in all tweets, doles out an ID and tallies the recurrence of the word per tweet. This pack of words would then be able to be utilized as contribution for a classifier. It is what is known as a meager informational index, implying that each record will have a large number for the words not happening in the tweet.



```
In [108]: countvect = CountVectorizer()

In [109]: best_mnb_countvect = grid_vect(mnb, parameters_mnb, X_train, X_test, parameters_text=parameters_vect, vect=countvect)
          Performing grid search...
          pipeline: ['features', 'clf']
          parameters:
          {'clf__alpha': (0.25, 0.5, 0.75),
           'features__pipe__vect__max_df': (0.25, 0.5, 0.75),
           'features__pipe__vect__min_df': (1, 2),
           'features__pipe__vect__ngram_range': ((1, 1), (1, 2))}
          Fitting 5 folds for each of 36 candidates, totalling 180 fits

          [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
          [Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   20.8s
          [Parallel(n_jobs=-1)]: Done 180 out of 180 | elapsed:  1.3min finished

          done in 80.113s

          Best CV score: 0.594
          Best parameters set:
                  clf__alpha: 0.5
                  features__pipe__vect__max_df: 0.25
                  features__pipe__vect__min_df: 2
                  features__pipe__vect__ngram_range: (1, 1)
          Test score with best_estimator_: 0.593


          Classification Report Test Data
                        precision    recall  f1-score   support

                 anger       0.42      0.27      0.33       102
               arousal       0.55      0.36      0.44       174
             dominance       0.62      0.86      0.72       690
                 faith       0.70      0.27      0.39        52
                  fear       0.59      0.23      0.33        44
                   joy       0.62      0.64      0.63       289
               neutral       0.44      0.05      0.09        84
               sadness       0.47      0.26      0.34       110

             micro avg       0.60      0.60      0.60      1545
             macro avg       0.55      0.37      0.41      1545
          weighted avg       0.58      0.60      0.56      1545
```

- **LOGISTIC REGRESSION**



```
In [110]: # LogisticRegression
          best_logreg_countvect = grid_vect(logreg, parameters_logreg, X_train, X_test, parameters_text=parameters_vect,
                                            vect=countvect)
```

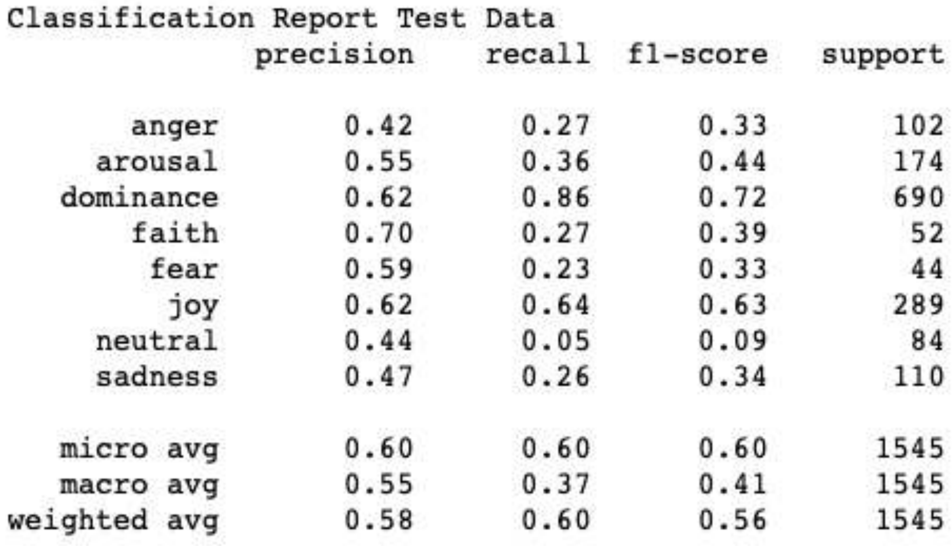
```
Performing grid search...
pipeline: ['features', 'clf']
parameters:
{'clf__C': (0.25, 0.5, 1.0),
 'clf__penalty': ('l1', 'l2'),
 'features__pipe__vect__max_df': (0.25, 0.5, 0.75),
 'features__pipe__vect__min_df': (1, 2),
 'features__pipe__vect__ngram_range': ((1, 1), (1, 2))}
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   31.4s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 360 out of 360 | elapsed:  6.5min finished
```

```
done in 389.703s

Best CV score: 0.725
Best parameters set:
	clf__C: 1.0
	clf__penalty: 'l2'
	features__pipe__vect__max_df: 0.25
	features__pipe__vect__min_df: 1
	features__pipe__vect__ngram_range: (1, 1)
Test score with best_estimator_: 0.733
```

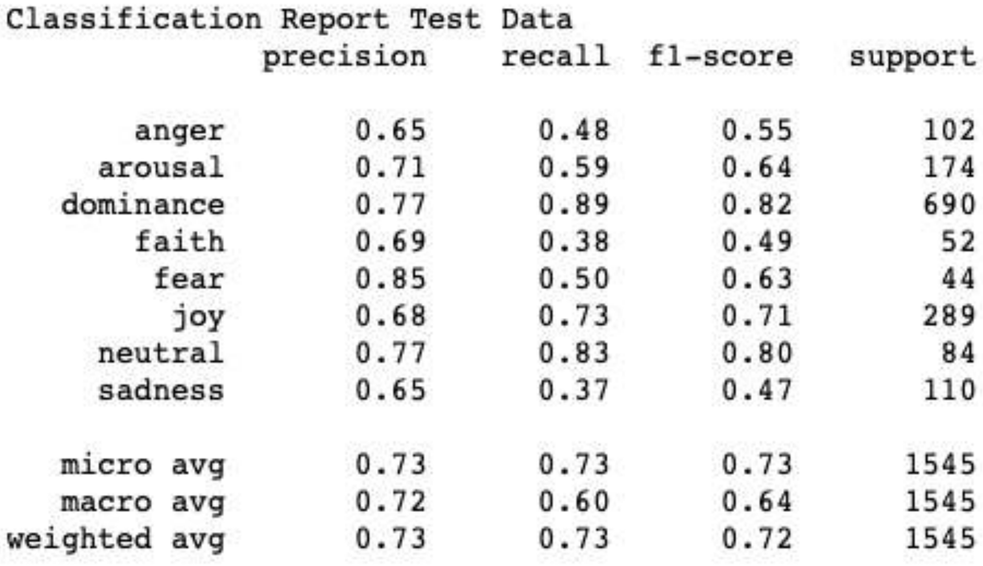
COMPARISON OF MULTINOMIAL NB AND LOGISTIC REGRESSION

MultiNomial NB



```
Classification Report Test Data
              precision    recall  f1-score   support

       anger       0.42      0.27      0.33       102
     arousal       0.55      0.36      0.44       174
   dominance       0.62      0.86      0.72       690
       faith       0.70      0.27      0.39        52
        fear       0.59      0.23      0.33        44
         joy       0.62      0.64      0.63       289
     neutral       0.44      0.05      0.09        84
     sadness       0.47      0.26      0.34       110

   micro avg       0.60      0.60      0.60      1545
   macro avg       0.55      0.37      0.41      1545
weighted avg       0.58      0.60      0.56      1545
```

Logistic Regression



```
Classification Report Test Data
              precision    recall  f1-score   support

       anger       0.65      0.48      0.55       102
     arousal       0.71      0.59      0.64       174
   dominance       0.77      0.89      0.82       690
       faith       0.69      0.38      0.49        52
        fear       0.85      0.50      0.63        44
         joy       0.68      0.73      0.71       289
     neutral       0.77      0.83      0.80        84
     sadness       0.65      0.37      0.47       110

   micro avg       0.73      0.73      0.73      1545
   macro avg       0.72      0.60      0.64      1545
weighted avg       0.73      0.73      0.72      1545
```

- **Word2Vec**

Another method for changing over the words in the tweets to numerical qualities can be accomplished with Word2Vec. Word2Vec maps each word in a multi-dimensional space. It does this by considering the setting wherein a word shows up in the tweets. Therefore, words that are semantically comparative are additionally near one another in the multi-dimensional space.

The Word2Vec calculation is actualized in the genism bundle.

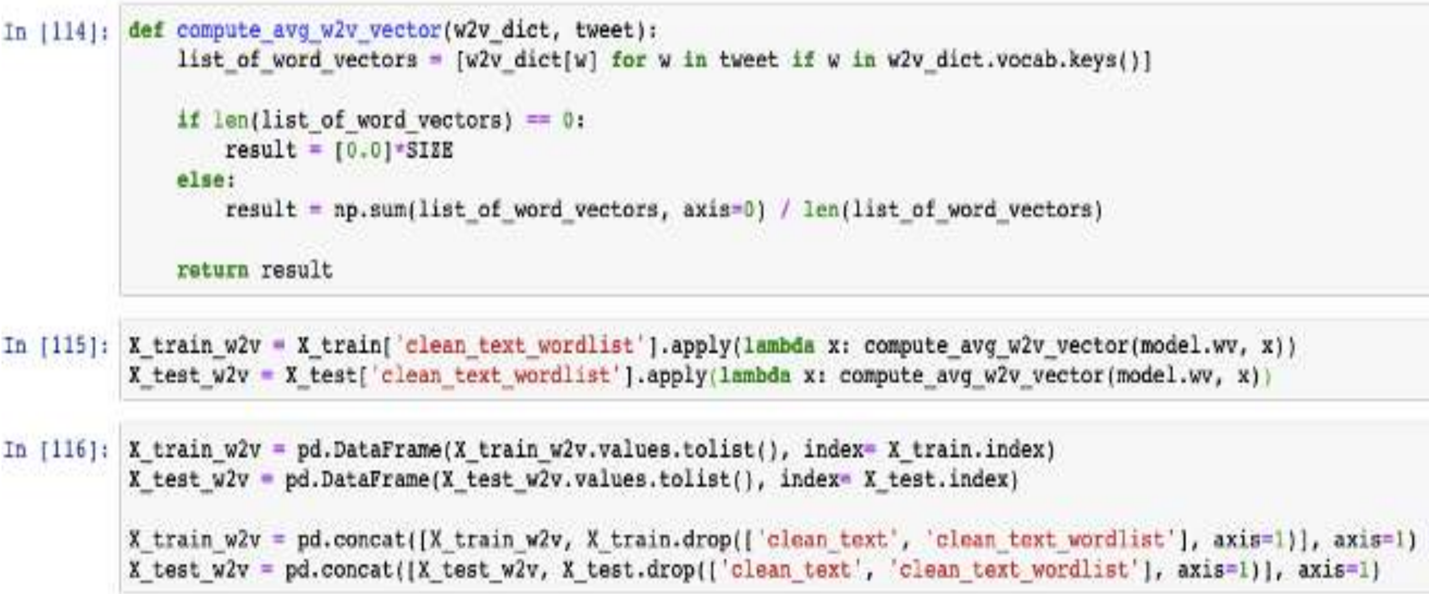
The Word2Vec calculation utilizes arrangements of words as information. For that reason, we utilize the word_tokenize strategy for the nltk bundle.

The Word2Vec model gives a vocabulary of the words in the corpus together with their vector esteems. The quantity of vector esteems is equivalent to the picked size. These are the measurements on which each word is mapped in the multi-dimensional space.

```
In [112]: nltk.download('punkt')
          SIZE = 25

          X_train['clean_text_wordlist'] = X_train.clean_text.apply(lambda x : word_tokenize(x))
          X_test['clean_text_wordlist'] = X_test.clean_text.apply(lambda x : word_tokenize(x))

          model = gensim.models.Word2Vec(X_train.clean_text_wordlist
                           , min_count=1
                           , size=SIZE
                           , window=3
                           , workers=4)

          [nltk_data] Downloading package punkt to /Users/ritika/nltk_data...
          [nltk_data]   Unzipping tokenizers/punkt.zip.

In [113]: model.most_similar('plane', topn=3)

Out[113]: [('manufactur', 0.9952282309532166),
           ('field', 0.9949126243591309),
           ('latest', 0.9949066042900085)]
```

Words with an event not exactly min_count are not kept in the vocabulary.

NOTE: A symptom of the min_count parameter is that a few tweets could have no vector esteems. This is would be the situation when the word(s) in the tweet happen in under min_count tweets. Because of the little corpus of tweets, there is a danger of this incident for our situation. In this manner we set the min_count esteem equivalent to 1.

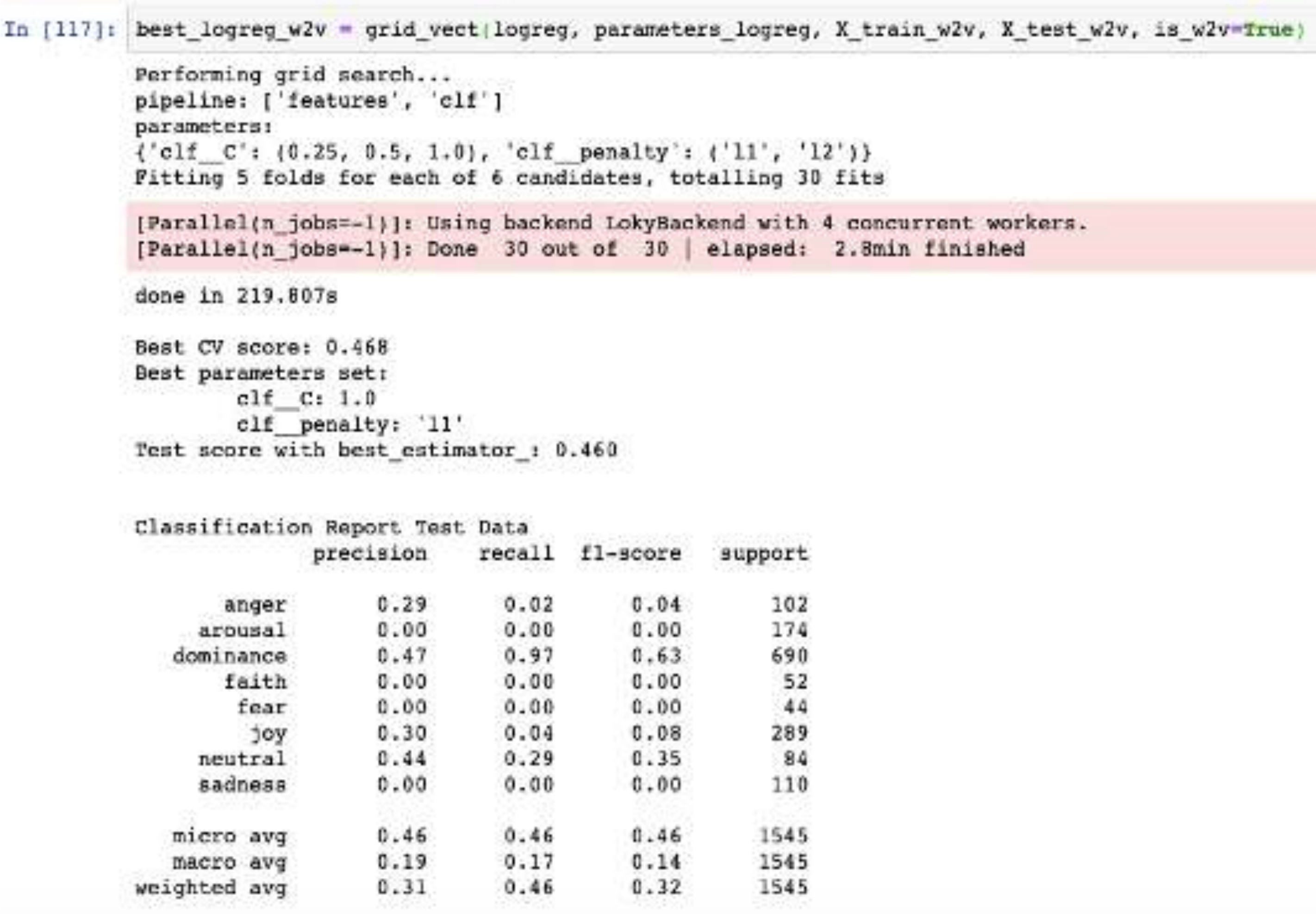


```
In [114]: def compute_avg_w2v_vector(w2v_dict, tweet):
              list_of_word_vectors = [w2v_dict[w] for w in tweet if w in w2v_dict.vocab.keys()]

              if len(list_of_word_vectors) == 0:
                  result = [0.0]*SIZE
              else:
                  result = np.sum(list_of_word_vectors, axis=0) / len(list_of_word_vectors)

              return result
```

```
In [115]: X_train_w2v = X_train['clean_text_wordlist'].apply(lambda x: compute_avg_w2v_vector(model.wv, x))
          X_test_w2v = X_test['clean_text_wordlist'].apply(lambda x: compute_avg_w2v_vector(model.wv, x))
```

```
In [116]: X_train_w2v = pd.DataFrame(X_train_w2v.values.tolist(), index= X_train.index)
          X_test_w2v = pd.DataFrame(X_test_w2v.values.tolist(), index= X_test.index)

          X_train_w2v = pd.concat([X_train_w2v, X_train.drop(['clean_text', 'clean_text_wordlist'], axis=1)], axis=1)
          X_test_w2v = pd.concat([X_test_w2v, X_test.drop(['clean_text', 'clean_text_wordlist'], axis=1)], axis=1)
```

The tweets can have an alternate number of vectors, contingent upon the quantity of words it contains. To utilize this yield for demonstrating we will total the vectors per tweet to have a similar number (for example estimate) of information factors per tweet. Consequently, we will take the normal of all vectors per tweet. We do this with the capacity compute_avg_w2v_vector. In this capacity we additionally check whether the words in the tweet happen in the vocabulary of the word2vec model. If not, a rundown loaded up with 0.0 is returned. Else the normal of the word vectors.



```
In [117]: best_logreg_w2v = grid_vect(logreg, parameters_logreg, X_train_w2v, X_test_w2v, is_w2v=True)

          Performing grid search...
          pipeline: ['features', 'clf']
          parameters:
          {'clf__C': (0.25, 0.5, 1.0), 'clf__penalty': ('l1', 'l2')}
          Fitting 5 folds for each of 6 candidates, totalling 30 fits

          [Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
          [Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.8min finished

          done in 219.807s

          Best CV score: 0.468
          Best parameters set:
                  clf__C: 1.0
                  clf__penalty: 'l1'
          Test score with best_estimator_: 0.460


          Classification Report Test Data
                        precision    recall  f1-score   support

                 anger       0.29      0.02      0.04       102
               arousal       0.00      0.00      0.00       174
             dominance       0.47      0.97      0.63       690
                 faith       0.00      0.00      0.00        52
                  fear       0.00      0.00      0.00        44
                   joy       0.30      0.04      0.08       289
               neutral       0.44      0.29      0.35        84
               sadness       0.00      0.00      0.00       110

             micro avg       0.46      0.46      0.46      1545
             macro avg       0.19      0.17      0.14      1545
          weighted avg       0.31      0.46      0.32      1545
```

The two classifiers accomplish the best outcomes when utilizing the highlights of the CountVectorizer By and large, Logistic Regression outflanks the Multinomial Naive Bayes classifier. The best execution on the test set originates from the LogisticRegression with highlights from CountVectorizer.

Best parameters:

C estimation of 1

L2 regularization

max_df: 0.5 or greatest report recurrence of half.

min_df: 1 or the words need to show up in at any rate 2 tweets

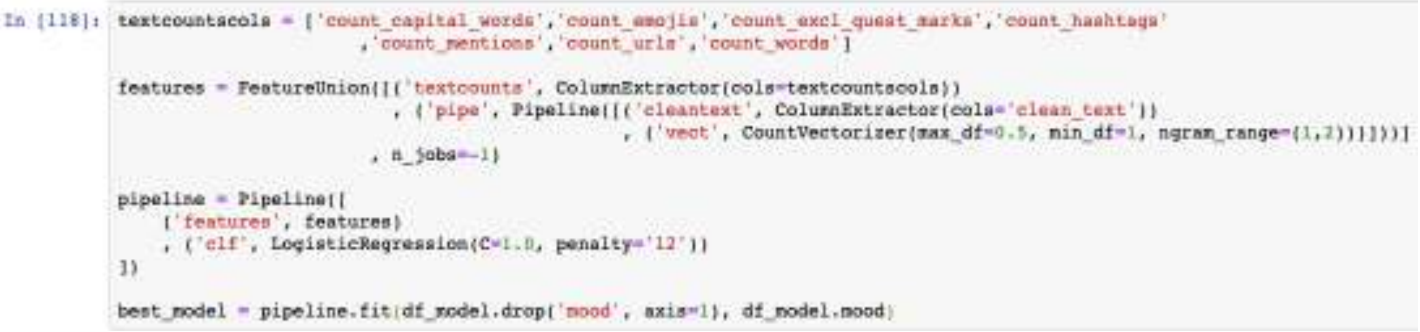
ngram_range: (1, 2), both single words as bi-grams are utilized

ASSESSMENT MEASUREMENTS:

A test precision of 57.3%, which is superior to anything what we would accomplish by setting the expectation for all perceptions to the greater part class (negative which would give 63% exactness).

The Precision is fairly high for each of the three classes. For example, of all cases that we foresee as negative, 60% is to be sure negative.

The Recall for the unbiased class is low. Of every single impartial case in our test information, we just foresee 58% as being unbiased.



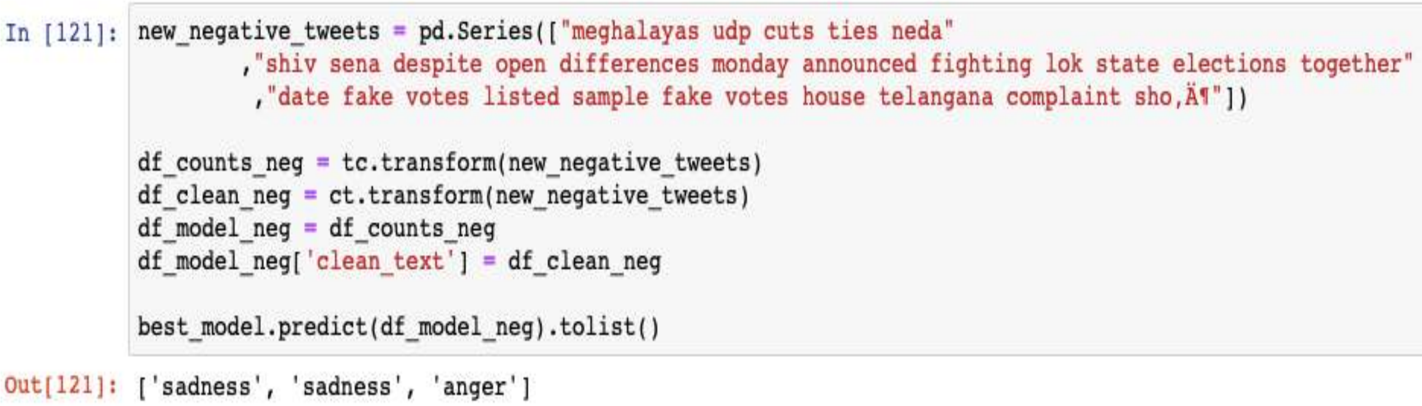
```
In [118]: textcountscols = ['count_capital_words','count_emojis','count_excl_quest_marks','count_hashtags'
                                ,'count_mentions','count_urls','count_words']

          features = FeatureUnion([('textcounts', ColumnExtractor(cols=textcountscols))
                                  , ('pipe', Pipeline([('cleantext', ColumnExtractor(cols='clean_text'))
                                                     , ('vect', CountVectorizer(max_df=0.5, min_df=1, ngram_range=(1,2)))]))]
                                 , n_jobs=-1)

          pipeline = Pipeline([
              ('features', features)
              , ('clf', LogisticRegression(C=1.0, penalty='l2'))
          ])

          best_model = pipeline.fit(df_model.drop('mood', axis=1), df_model.mood)
```

- **TESTING OF THE NEW MODEL DEVELOPED**

Test for positive tweets

```
In [120]: new_positive_tweets = pd.Series(["nda achieved sustained growth low inflation"
                                ,"healthy india making"
                                ,"president exudes confidence together win seats"])

          df_counts_pos = tc.transform(new_positive_tweets)
          df_clean_pos = ct.transform(new_positive_tweets)
          df_model_pos = df_counts_pos
          df_model_pos['clean_text'] = df_clean_pos

          best_model.predict(df_model_pos).tolist()

Out[120]: ['joy', 'joy', 'joy']
```

Test for negative tweets



```
In [121]: new_negative_tweets = pd.Series(["meghalayas udp cuts ties neda"
                ,"shiv sena despite open differences monday announced fighting lok state elections together"
                 ,"date fake votes listed sample fake votes house telangana complaint sho,Ä¶"])

          df_counts_neg = tc.transform(new_negative_tweets)
          df_clean_neg = ct.transform(new_negative_tweets)
          df_model_neg = df_counts_neg
          df_model_neg['clean_text'] = df_clean_neg

          best_model.predict(df_model_neg).tolist()

Out[121]: ['sadness', 'sadness', 'anger']
```

CONCLUSION

The model classifies all of the tweets correctly.

We will delineate motion election data survey into a genuine vector space, a well-known method when working with content called word embedding. This is where words are encoded as genuine esteemed vectors in a high dimensional space, where the similitude between words as far as importance means closeness in the vector space.

Keras gives an advantageous method to change over positive number portrayals of words into a word installing by an Embedding layer.

We will delineate word onto a 32 length genuine esteemed vector. We will likewise restrain the all-out number of words that we are keen on displaying to the 5000 most incessant words, and zero out the rest. At last, the arrangement length (number of words) in each audit shifts, so we will oblige each survey to be 500 words, truncating long surveys and cushion the shorter audits with zero qualities.

Since we have characterized our concern and how the information will be arranged and displayed, we are prepared to build up a LSTM model to group the supposition of motion picture surveys.

Given a chance to feature the principle commitments of this work.

- We endeavoured to arrange the political direction of the Twitter clients from India. Because of the assortment of ideological groups and pioneers and utilization of various dialects, this work turned out to be significantly all the more testing.

- We broke down the political direction of Twitter clients for the 'Ace' class, yet in addition for the 'Counter' classification.

• We demonstrated that the system dependent on retweets and client makes reference to works the best for order in the Indian situation.

• After investigating the information more than 5 months, we demonstrated that the weekdays saw the greatest movement on Twitter identified with legislative issues.

• We demonstrated that BJP and its Prime Ministerial competitor Narendra Modi were the most noteworthy gainers in the fields of notices and ubiquity on Twitter.

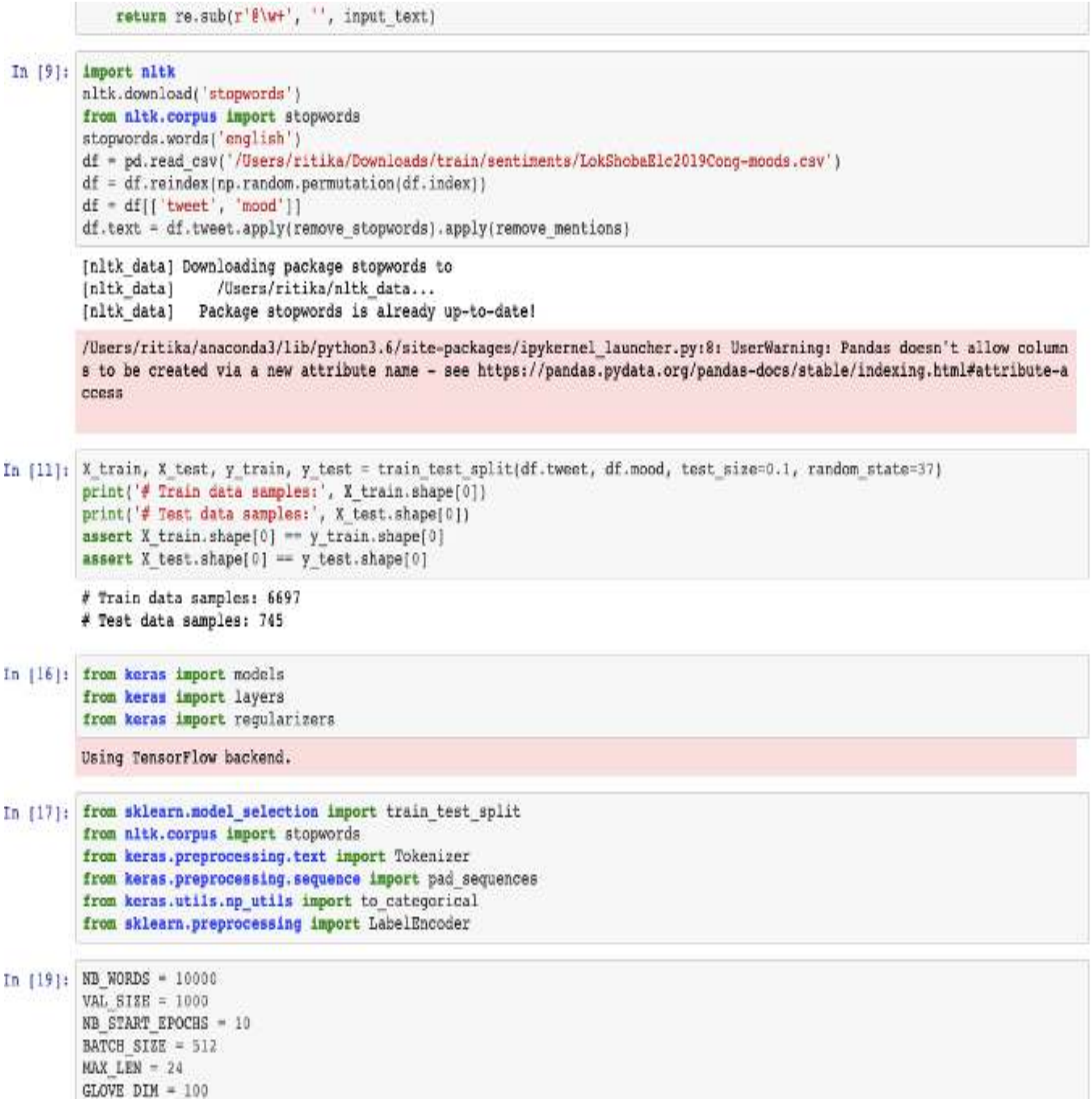
USE THE EMBEDDING LAYER OF KERAS TO CREATE WORD EMBEDDINGS FROM THE TRAINING DATA

While applying one-hot encoding to the words in the tweets, we end up with merger vectors of high dimensionality (here it is by increasing the quantity of words). On bigger informational collections this could cause execution issues. Moreover, one-hot encoding does not consider the semantics of the words. For example, plane and flying machine are various words yet have a comparative significance.

Word embeddings lessen these two issues. Word embeddings are thick vectors with a much lower dimensionality. Also, the semantic connections between words are reflected out there and bearing of the vector.

NETWORK BASED CLASSIFICATION

```
In [8]: def deep_model(model, X_train, y_train, X_valid, y_valid):

            model.compile(optimizer='rmsprop'
                          , loss='categorical_crossentropy'
                          , metrics=['accuracy'])

            history = model.fit(X_train
                               , y_train
                               , epochs=NB_START_EPOCHS
                               , batch_size=BATCH_SIZE
                               , validation_data=(X_valid, y_valid)
                               , verbose=0)
            return history


        def eval_metric(history, metric_name):

            metric = history.history[metric_name]
            val_metric = history.history['val_' + metric_name]

            e = range(1, NB_START_EPOCHS + 1)

            plt.plot(e, metric, 'bo', label='Train ' + metric_name)
            plt.plot(e, val_metric, 'b', label='Validation ' + metric_name)
            plt.legend()
            plt.show()

        def test_model(model, X_train, y_train, X_test, y_test, epoch_stop):

            model.fit(X_train
                     , y_train
                     , epochs=epoch_stop
                     , batch_size=BATCH_SIZE
                     , verbose=0)
            results = model.evaluate(X_test, y_test)

            return results

        def remove_stopwords(input_text):

            stopwords_list = stopwords.words('english')
            # Some words which might indicate a certain sentiment are kept via a whitelist
            whitelist = ["n't", "not", "no"]
            words = input_text.split()
            clean_words = [word for word in words if (word not in stopwords_list or word in whitelist) and len(word) > 1]
            return " ".join(clean_words)
```



```
    return re.sub(r'@\w+', '', input_text)
```

```
In [9]: import nltk
        nltk.download('stopwords')
        from nltk.corpus import stopwords
        stopwords.words('english')
        df = pd.read_csv('/Users/ritika/Downloads/train/sentiments/LokShobaElc2019Cong-moods.csv')
        df = df.reindex(np.random.permutation(df.index))
        df = df[['tweet', 'mood']]
        df.text = df.tweet.apply(remove_stopwords).apply(remove_mentions)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/ritika/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
/Users/ritika/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: UserWarning: Pandas doesn't allow column
s to be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-a
ccess
```

```
In [11]: X_train, X_test, y_train, y_test = train_test_split(df.tweet, df.mood, test_size=0.1, random_state=37)
         print('# Train data samples:', X_train.shape[0])
         print('# Test data samples:', X_test.shape[0])
         assert X_train.shape[0] == y_train.shape[0]
         assert X_test.shape[0] == y_test.shape[0]
```

```
# Train data samples: 6697
# Test data samples: 745
```

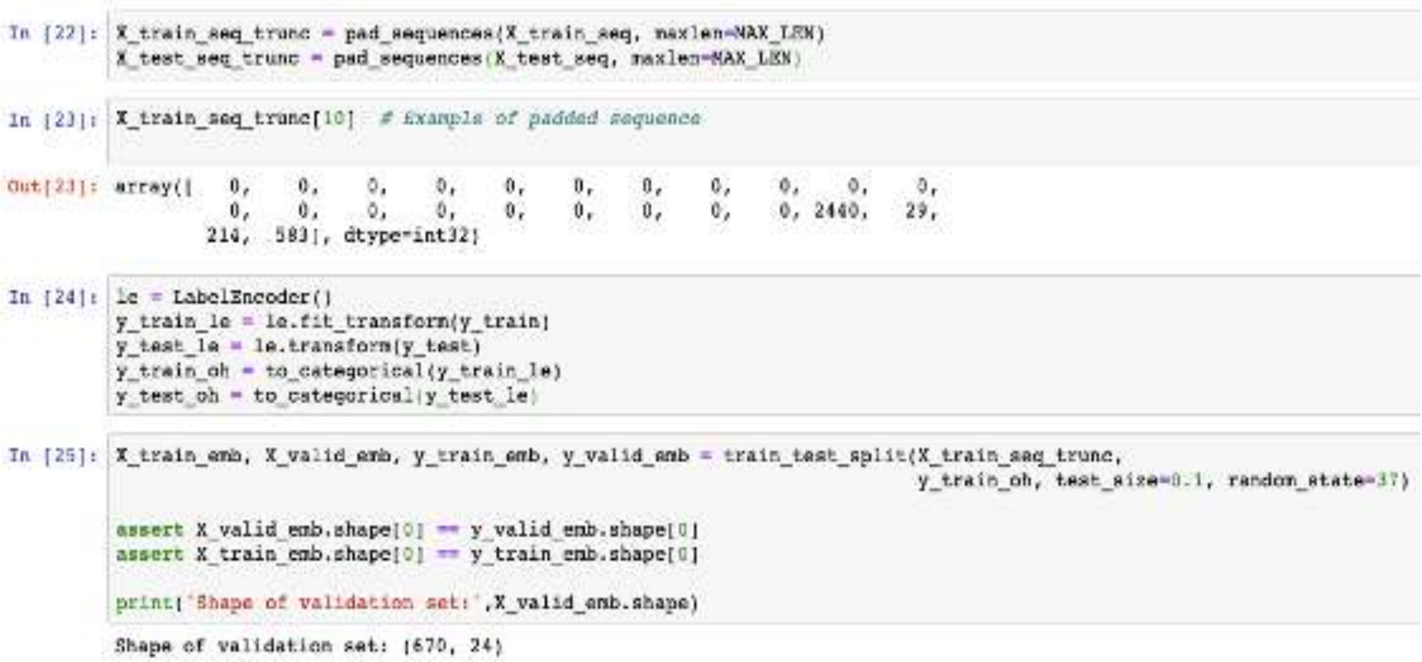
```
In [16]: from keras import models
         from keras import layers
         from keras import regularizers
```

```
Using TensorFlow backend.
```

```
In [17]: from sklearn.model_selection import train_test_split
         from nltk.corpus import stopwords
         from keras.preprocessing.text import Tokenizer
         from keras.preprocessing.sequence import pad_sequences
         from keras.utils.np_utils import to_categorical
         from sklearn.preprocessing import LabelEncoder
```

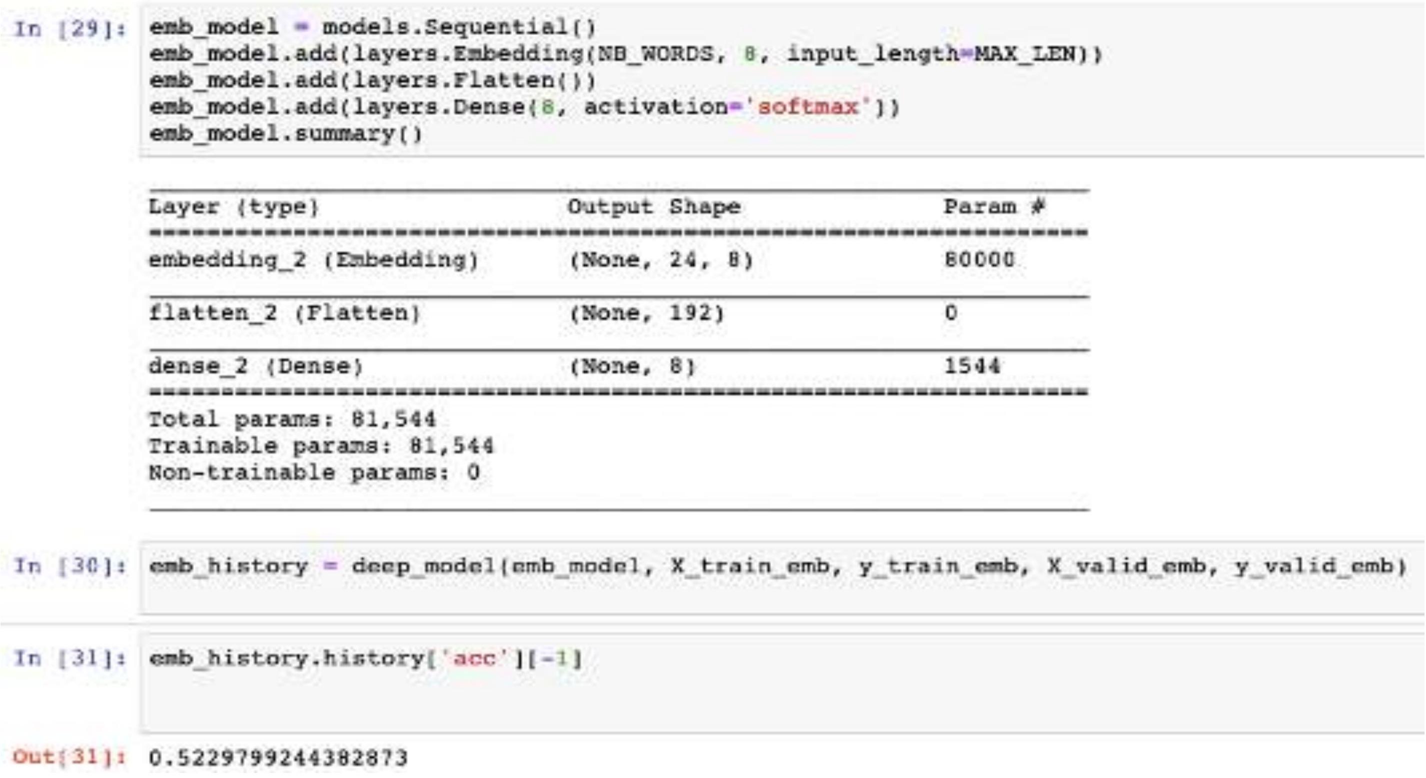
```
In [19]: NB_WORDS = 10000
         VAL_SIZE = 1000
         NB_START_EPOCHS = 10
         BATCH_SIZE = 512
         MAX_LEN = 24
         GLOVE_DIM = 100
```

- **CONVERTING THE TARGET CLASSES TO NUMBERS AND SPLITTING OFF VALIDATION DATA**

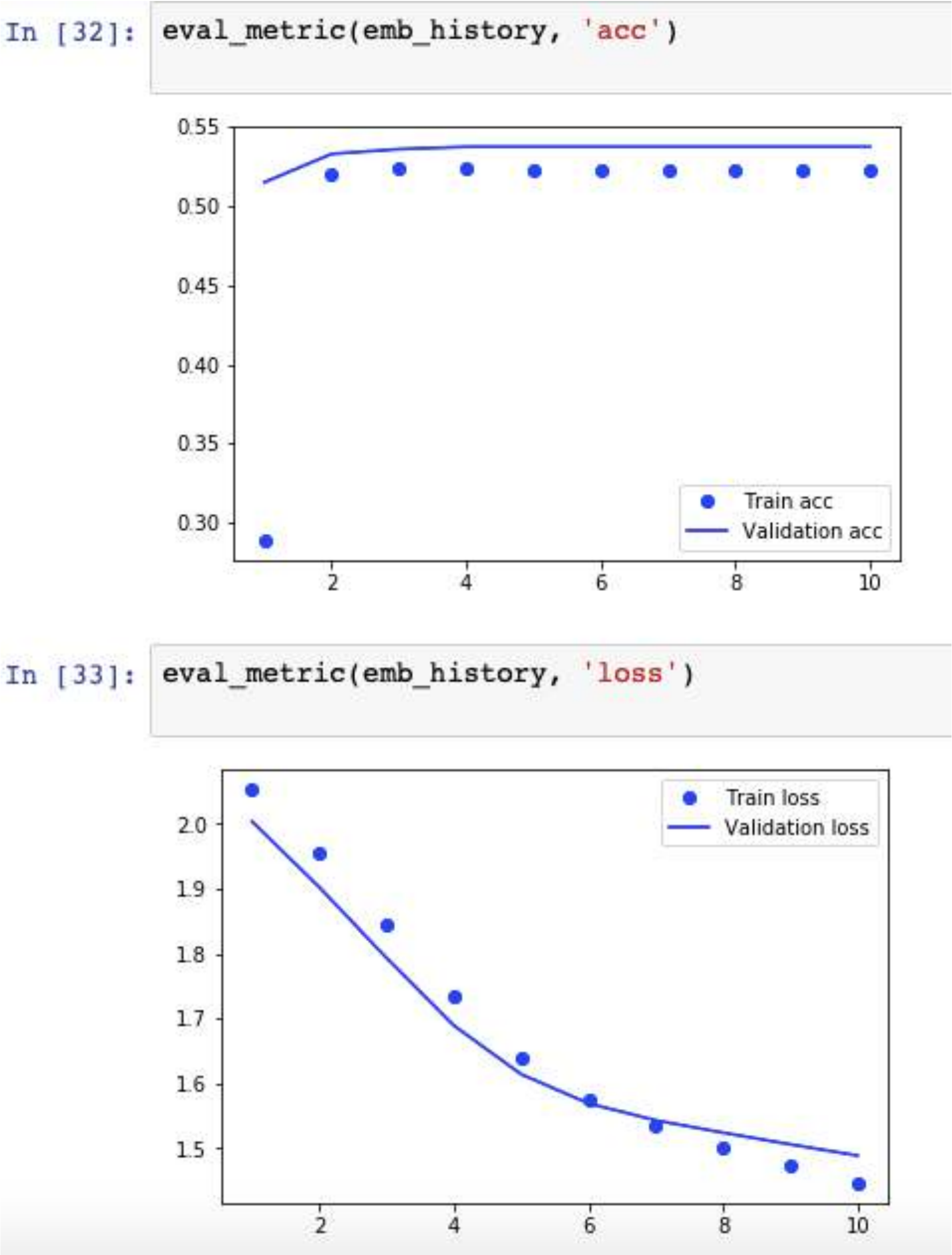


```
In [22]: X_train_seq_trunc = pad_sequences(X_train_seq, maxlen=MAX_LEN)
         X_test_seq_trunc = pad_sequences(X_test_seq, maxlen=MAX_LEN)

In [23]: X_train_seq_trunc[10]  # Example of padded sequence

Out[23]: array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
                   0,    0,    0,    0,    0,    0,    0,    0,    0, 2440,   29,
                 214,  583], dtype=int32)

In [24]: le = LabelEncoder()
         y_train_le = le.fit_transform(y_train)
         y_test_le = le.transform(y_test)
         y_train_oh = to_categorical(y_train_le)
         y_test_oh = to_categorical(y_test_le)

In [25]: X_train_emb, X_valid_emb, y_train_emb, y_valid_emb = train_test_split(X_train_seq_trunc,
                                                                               y_train_oh, test_size=0.1, random_state=37)

         assert X_valid_emb.shape[0] == y_valid_emb.shape[0]
         assert X_train_emb.shape[0] == y_train_emb.shape[0]

         print('Shape of validation set:',X_valid_emb.shape)

         Shape of validation set: (670, 24)
```

- **MODELING**



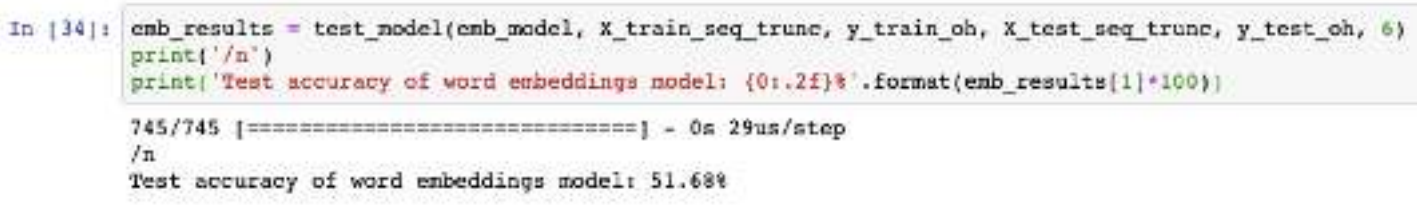
```
In [29]: emb_model = models.Sequential()
         emb_model.add(layers.Embedding(NB_WORDS, 8, input_length=MAX_LEN))
         emb_model.add(layers.Flatten())
         emb_model.add(layers.Dense(8, activation='softmax'))
         emb_model.summary()

         _________________________________________________________________
         Layer (type)                 Output Shape              Param #
         =================================================================
         embedding_2 (Embedding)      (None, 24, 8)             80000
         _________________________________________________________________
         flatten_2 (Flatten)          (None, 192)               0
         _________________________________________________________________
         dense_2 (Dense)              (None, 8)                 1544
         =================================================================
         Total params: 81,544
         Trainable params: 81,544
         Non-trainable params: 0
         _________________________________________________________________

In [30]: emb_history = deep_model(emb_model, X_train_emb, y_train_emb, X_valid_emb, y_valid_emb)

In [31]: emb_history.history['acc'][-1]

Out[31]: 0.5229799244382873
```



```
In [32]: eval_metric(emb_history, 'acc')
```

```
In [33]: eval_metric(emb_history, 'loss')
```

ACCURACY OF MODEL FOR WORD EMBEDDINGS



```
In [34]: emb_results = test_model(emb_model, X_train_seq_trunc, y_train_oh, X_test_seq_trunc, y_test_oh, 6)
         print('/n')
         print('Test accuracy of word embeddings model: {0:.2f}%'.format(emb_results[1]*100))

         745/745 [==============================] - 0s 29us/step
         /n
         Test accuracy of word embeddings model: 51.68%
```

The model has an approval exactness of about 52%. The quantity of words in the tweets is fairly low, so this outcome is somewhat great.

By looking at the preparation and approval precision and misfortune, it is seen that the model begins overfitting from epoch 6.

Keras gives an Embedding layer which causes us to prepare explicit word embeddings dependent on our preparation information. It will change over the words in our vocabulary to multi-dimensional vectors.

CONCLUSION

In the initial segment of this work, tweets were analyzed in our dataset. We broke down the volume of the tweets approaching each day and advocated the crests in the information by giving the course of events of the major political exercises amid India General Elections 2019. It was accounted for that Tuesdays and Wednesdays saw the most elevated number of Tweets as greater part of the exercises were amid weekdays and the action crested especially in the second 50% of the day. It was additionally discovered that we had 815,425 exceptional clients in our dataset. The quantity of one of a kind client for consistently were likewise detailed. It was appeared according to the prevalent view the volume of tweets expanded as races came nearer. The quantity of notices over recent months were appeared for AAP, BJP and Congress, however 8 different gatherings that were dynamic in different states. We additionally investigated the fame of Arvind Kejriwal and Narendra Modi on two distinct parameters and detailed how their political conduct influenced their prevalence on Twitter.

The second piece of the work was concerning the political direction of clients. We utilized 4 approaches for the creating 4 sorts of classifiers. Subsequent to getting 1000 profiles explained we could set up a genuine positive dataset with 613 cases of Pro and 425 occasion of Anti class. The principal approach utilized a client vector that had the TFIDF score for each term. The effectiveness with this strategy for equivalent number of examples was 42.42% for Pro and 37.25% for the Anti classification. We at that point proceeded onward to the second strategy which utilized the hashtags utilized in tweets as highlights for order. This technique improved the effectiveness somewhat yet not as far as possible. To check if the strategies were working right, we attempted 2-class arrangement also, however without much of any result.

The third technique was the utilization of client-based highlights, for example, number of companions and devotees, number of events of AAP, BJP and Congress related words and hashtags. While BJP led the pack in utilizing data analytics and predictive analysis for political purposes, Congress, alongside other national parties was viewed as a late participant in cementing a huge digital database. The 2019 Lok Sabha Election was a prime case of utilizing information driven strategies to make a viable advanced interface and communication crosswise over India and streaming data to be utilized.

The fourth part of the work was the advancement of the entry that showed the investigation of the tweets of the most recent 24 hours and successfully classified the sentiments of the live streaming Tweet dataset. Elections are a complex, multi-dimensional social and political occasion which can be caught just through an assortment of strategies: this thesis underlines how the various methodologies complete one another and are thusly similarly important.

While anyone thinks about the Indian electoral race, at the national and state levels, they have been ruled, since the 1990s, by study scholars or examiners. The Lokniti based venture of 'Comparative Electoral Ethnography' ought to add to reestablishing some harmony between different kinds of studies. Additionally, scholarly discussions around the logical and political ramifications and impediments of decision considers appear to prompt an assembly: while poll-based studies advance towards a better worry of the sentiments and dispositions of Indian voters, anthropological examinations endeavor to defeat the constraints of hands on work dependent on a solitary, restricted territory.

REFERENCES

1. N. D. Valakunde, M. S. Patwardhan, "Multi-Aspect and Multi-Class Based Document Sentiment Analysis of Educational Data Catering Accreditation Process", *IEEE International Conference on Cloud and Ubiquitous Computing and Emerging Technologies*, 2013.

2. Liu Bing, Sentiment Analysis and Opinion Mining, Morgan and Claypool Publishers, May 2012.

3. A. Bifet, G. Holmes, B. Pfahringer, "MOA-TweetReader: real-time analysis in twitter streaming data" in LNCS 6926, Springer-Verlag Berlin Heidelberg, pp. 4660, 2011.

4. Vivek Narayanan, Ishan Arora, Arjun Bhatia, *Fast and accurate sentiment classification using an enhanced Naive Bayes model*, 2012.

5. Pulkit Kathuria, Kiyoaki Shirai, Example Based Word Sense Disambiguation towards Reading Assistant System The Association for Natural Language Processing, 2012.

6. Farhan Hassan Khan, Saba Bashir, Usman Qamar, "TOM: Twitter opinion mining framework using hybrid classification scheme", *Decision Support Systems*, vol. 57, 2014.

7. Xingtong Ge, Xiaofang Jin, Bo Miao, Chenming Liu, Xinyi Wu, "Research on the Key Technology of Chinese Text Sentiment Analysis", *Software Engineering and Service Science (ICSESS) 2018 IEEE 9th International Conference on*, pp. 395-398, 2018.

8. Shubham Kumar Jain, Pardeep Singh, "Systematic Survey on Sentiment Analysis", *Secure Cyber Computing and Communication (ICSCCC) 2018 First International Conference on*, pp. 561-565, 2018.

9. Sneh Paliwal, Sunil Kumar Khatri, Mayank Sharma, "Sentiment Analysis and Prediction Using Neural Networks", *Inventive Research in Computing Applications (ICIRCA) 2018 International Conference on*, pp. 1035-1042, 2018.

10. Chacón D, (2013). “Why Big Data Is Important to You and Your Organization “

11. Nasser T and Tariq RS “Big Data Challenges” (2015). Journal of Computer Engineering and Information Technology.

12. Salla-Maaria Laaksonen, et al., Working the fields of big data: Using big-data-augmented online ethnography to study candidate- candidate interaction at election time. Journal of information technology & politics, 2017.

13. Eethany Anee Conway et al.,The Rise of Twitter in the Political Campaign: Searching for Intermedia Agenda-Setting Effects in the Presidential Primary. Article in Journal of Computer-Mediated Communication · May 2015.

14. Dubey Gaurav, et al., Social media opinion analysis for Indian political diplomats. IEEE 2017.

15. Daniel José Silva Oliveira, et al., Can social media reveal the preferences of voters? A comparison between sentiment analysis and traditional opinion polls, Journal of Information Technology & Politics, 2016.

16. Shubham Goyal. , Review Paper on Sentiment Analysis of Twitter Data Using Text Mining and Hybrid Classification Approach. In- ternational Journal of Engineering Development and Research (www.ijedr.org) 2017.

17. Miss. Payal Rajkumar Rathi et al., Big Data Analytics for Social Network--the Base Study, http://www.ijettjournal.org.2016.

18. Gagandeep Jagdev and Amandeep Kaur; Analyzing and Scripting Indian Election strategies using Big Data via Apache Hadoop framework, IEEE 2016.

19. Andy Januar Wicaksono et al. , A Proposed Method for Predicting US Presidential Election by Analyzing Sentiment in Social Media, 2nd International Conference on Science in Information Technology , IEEE 2016.

20. Suarez Hernandez A, et al., predicting political mood tendencies based on twitter data.

21. Tsakalidis Adam, et al., predicting elections for multiple countries using twitter and polls. IEEE 2015.

22. Kangan vadim, et al., using twitter sentiment to forecast the 2013 pakistani election and the 2014 indian election. IEEE 2015.

23. E. Sang and J. Bos, "Predicting the 2011 Dutch Senate Election Results with Twitter," Proc. Workshop Semantic Analysis in Social Media, 2012, pp. 53–60.

24. B. O"Connor et al., "From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series," Proc. 4th Int"l AAAI Conf. Web- logs and Social Media

25. Twitter Statisitics, http://www.statisticbrain.com/twitter-statistics/, Last Updated: 5.7.2013

26. C. Spengler, W. Werth, and R. Sigrist, "360o Touchpoint Man- agement -How important is Twitter for our brand", Marketing Re- view, St. Gallen, 2010.

27. M. Ghiassi, J. Skinner, and D. Zimbra, "Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic arti- ficial neural network", Expert Systems with Applications, Vol. 40, Issue 16, pp 6266-6282, November 2013.

28. Y. Bae and H. Lee, "Sentiment analysis of twitter audiences: Measuring the positive or negative influence of popular twitterers", Journal of the American Society for Information Science and Technology, Vol. 63, Issue 12, pp. 2521-2535, December 2012.

29. Ahuja, Amit; Chhibber, Pradeep (nd) 'Civic Duty, Empowerment and Patronage: Patterns of Political Participation in India', http://www.sscnet.ucla.edu/polisci/cpworkshop/papers/Chhibber.pdf (accessed on 21 November, 2009).

30. Auyero, Javier (2006) 'Introductory Note to Politics under the Microscope: Special Issue on Political Ethnography I', *Qualitative Sociology*, 2, pp. 257-9.

31. Banerjee, Mukulika (2007) 'Sacred Elections', *Economic and Political Weekly*, 28 April, pp. 1556-62. Brass, Paul (1985) *Caste, Faction and Party in Indian Politics. Volume Two: Election Studies*, Delhi: Chanakya Publications.

32. Butler, David; Lahiri, Ashok; Roy, Prannoy (1995) *India Decides. Elections 1952-1995*, Delhi: Books & Things. Chandra, Kanchan (2004) 'Elections as Auctions', *Seminar*, 539. Chandra, Kanchan (2008) 'Why voters in patronage democracies split their tickets: Strategic voting for ethnic parties', *Electoral Studies*, 28, pp. 21-32.

33. Chhibber, Pradeep; Petrocik, John R.K. (1989) 'The Puzzle of Indian Politics: Social Cleavages and the Indian Party System', *British Journal of Political Science*, 19(2), pp. 191-210.

34. Dikshit, S.K. (1993) *Electoral Geography of India, With Special Reference to Sixth and Seventh Lok Sabha Elections*, Varanasi: Vishwavidyalaya Prakashan.

35. Eldersveld, Samuel; Ahmed, Bashiruddin (1978) *Citizens and Politics: Mass Political Behaviour in India*, Chicago: University of Chicago Press.

36. Fauvelle-Aymar, Christine (2008) 'Electoral turnout in Johannesburg: Socio-economic and Political Determinants', *Transformation, Critical Perspectives on Southern Africa*, 66/67, pp. 142-67.

37. Hauser, Walter; Singer, Wendy (1986) 'The Democratic Rite: Celebration and Participation in the Indian Elections', *Asian Survey*, 26(9), pp. 941-58.

38. Hermet, Guy; Badie, Bertrand; Birnbaum, Pierre; Braud, Philippe (2001) *Dictionnaire de la science politique et des institutions politiques*, Paris: Armand Colin.

39. Jaffrelot, Christophe (2008) ''Why Should We Vote?' The Indian Middle Class and the Functioning of the World's Largest Democracy', in Christophe Jaffrelot & Peter Van der Veer (eds.), *Patterns of Middle Class Consumption in India and China*, Delhi: Sage.

40. Jayal, Niraja Gopal (2001) 'Introduction', in Niraja Gopal Jayal (ed.), *Democracy in India*, Delhi: Oxford University Press, pp. 1-49.

41. Lyngdoh, James Michael (2004) *Chronicle of an Impossible Election: The Election Commission and the 2002 Jammu and Kashmir Assembly Elections*, Delhi: Penguin/Viking.

42. Narain, Iqbal; Pande, K.C.; Sharma, M.L.; Rajpal, Hansa (1978) *Election Studies in India: An Evaluation,* New Delhi: Allied Publishers.

43. Palshikar, Suhas (2007) 'The Imagined Debate between Pollsters and Ethnographers', *Economic and Political Weekly*, 27 October, pp. 24-8.

44. Singer, Wendy (2007) *'A Constituency Suitable for Ladies' And Other Social Histories of Indian Elections*,

New Delhi: Oxford University Press.

45. Sundar, Nandini; Deshpande, Satish; Uberoi, Patricia (2000) 'Indian Anthropology and Sociology: Towards a History', *Economic and Political Weekly*, 10 June, pp. 1998-2002.

46. Yadav, Yogendra (2000) 'Understanding the Second Democratic Upsurge: Trends of Bahujan Participation in Electoral Politics in the 1990s', in Francine R. Frankel; Zoya Hasan; Rajeev Bhargava; Balveer Arora (eds.), *Transforming India: Social and Political Dynamics of Democracy*, Delhi: Oxford University Press.

SYNOPSIS

The essential goal of the analysis is to get a precise estimating model for the Lok Sabha Election 2019. To distinguish a dependable model, artificial neural networks (ANN) and support vector regression (SVR) models are thought about dependent on some predetermined exhibition measures. Besides, six autonomous factors, for example, GDP, job rate, the hopeful's endorsement rate, and others are considered in a stepwise relapse to distinguish noteworthy factors. The candidate's endorsement rate is distinguished as the most critical variable, in light of which eight different factors are recognized and considered in the model improvement.

Preprocessing techniques are connected to set up the information for the learning calculations. The proposed methodology essentially expands the exactness of the model up to 50%. The learning calculations (ANN and SVR) demonstrated to be better than direct relapse dependent on every technique's determined exhibition measures. The proposed methodology essentially expands the precision of the gauge for casting a ballot conduct and gathering execution.

Expanding the exactness of the achieved conjectures is another exploration objective. In addition, expository models are desired to give figures and distinctive used learning algorithms are additionally analyzed dependent on some predetermined exhibition measures.

The contrasts between artificial neural systems (ANN) and support vector regression (SVR) are researched additionally dependent on two proportions of errors: mean absolute prediction error (MAPE), and root-mean-squared error (RMSE). Exploring the effects of data mining procedures in expanding determining exactness is another goal of this experimental analysis, where two major datasets are compared utilizing every attribute.

To begin with, the characterization of the data, by a procedure of unsupervised learning as the gathering, clustering, brings the find of groups that are extraordinary yet the individual is equivalent among themselves as noted (Vega, 2012) In the philosophy of data mining.

The Clustering methods, is alluded in English as gathering, are strategies which are utilized to aggregate a progression of things. Clustering is utilized in insights and science. The strategies to manage are: the progressive technique since it is an exploratory instrument intended to uncover the normal gatherings inside a lot of information that would somehow isn't clear. It is helpful when you need to bunch few items, might be cases or factors, in the event that you need to arrange cases or look at connections between the factors. The target of this thesis was to determine an anticipating model for the Lok Sabha elections. Deep Learning Algorithms and data mining strategies were used towards this target. In addition, free factors, for example, GDP, employment ratio, individual salary, changes in the votes of the occupant party in the last electoral battle, and the candidate's activity endorsement were considered. The essentialness of every variable was dictated by applying stepwise regression. Thus, all factors relating from the party's activity endorsement rate were discarded.

The principle hypothesis was that the electoral decision reflects the public sentiment towards a party candidate. After the stepwise regression is performed, 7 factors identified with sentiment of public were considered to build up the anticipating model. By applying two preprocessing techniques, data transformation and clustering, the data was set up for the learning algorithm. The algorithm of bagging and boosting were applied on the dataset to reduce the variance of the data while significantly keeping the bias untouched. The primary standard behind the model is that a gathering of feeble learners meet up to frame a solid learned model. Bagging (Bootstrap Aggregation) is utilized when we will likely lessen the change of a decision tree. Here thought is to make a few subsets of data from training test picked arbitrarily with substitution.

Boosting was applied on the data set to generate the variation of prediction rule. Upon achieving a desirable strength of the prediction rule we declared the model fit for testing. Other algorithms like Classification, 3 types of regression algorithms and LSTM Time Series Forecasting was taken into account. A model was successfully trained and tested for sentiment analysis on live streaming data to justify both negative and positive tweets.

An audit of clustering algorithms calculations, can be referenced by (Xu, 2005) and examine distinctive calculation. The progressive technique is worked by a group of trees or various leveled hierarchical clusters. Arranged in agglomerative and disruptive. The initially starts with a group and after at least two comparative clusters. The second starts with a cluster containing every one of the data points and recursively partitions the gathering generally proper. The procedure proceeds and stops until the rule is improved

The election results are not so much unplanned, completely eased by past events and happenings, and that quite a bit of what occurs in neighborhood enables us to think about the potential situations of the nearby election result. The essential goal of this experiment is to demonstrate and figure the Lok Sabha 2019 decision by means of the utilization of learning calculations. Political and financial factors are used in the model, and noteworthy factors are distinguished through further examination and factual systems. The reliant variable is characterized as the discretionary votes of the occupant party. The candidate party is considered as the important variable, since it shows additionally related factors such the party's endorsement rate.

It is imperative to examine the connection between the verifiable pattern of the vote and the discretionary aftereffects of a particular election; it is significant in light of the fact that it enables us to make expectations, which can, in great measure, sharpening political candidates and voters about the potential consequences of the constituent voting pattern.

In the initial segment of our work, we investigated the tweets in our dataset. We dissected the volume of the tweets approaching each day and defended the peaks in the information by giving the course of events of the major political exercises amid India General Elections 2019. It was accounted for that Tuesdays and Wednesdays saw the most astounding number of Tweets as larger part of the exercises were amid weekdays and the action topped especially in the second 50% of the day. The quantity of unique account was additionally detailed. It was appeared according to the prevalent view the volume of tweets expanded as decisions came nearer. The quantity of notices over recent months were appeared for AAP, BJP and Congress, yet 8 different gatherings that were dynamic in different states. We additionally dissected the prevalence of Congress and BJP on several distinct parameters and revealed how their political conduct influenced their notoriety on Twitter.

The second piece of the work was concerning the political direction of clients. We utilized 4 approaches for the creating 2 sorts of classifiers. In the wake of getting 1000 profiles commented on we could set up a genuine positive dataset Pro of Anti classification. The principal approach utilized a client vector that had the TFIDF score for each term. The last strategy we attempted was the network detection algorithm. So, we can close our data analysis on conclusion that BJP and its Prime Ministerial competitor Narendra Modi were the most noteworthy gainers in the fields of notices and ubiquity on Twitter.

Elections are a complex, multi-dimensional social and political occasion which can be caught just through an assortment of strategies: this thesis underlines how the various methodologies complete one another and are thusly similarly important. While Indian electoral race thinks about, at any rate at the national and state levels, have been ruled, since the 1990s, by study examine, the Lokniti based venture of 'Comparative Electoral Ethnography' ought to add to reestablishing some harmony between different kinds of studies. Additionally, scholarly discussions around the logical and political ramifications and impediments of decision considers appear to prompt an assembly: while poll-based studies advance towards a better worry of the sentiments and dispositions of Indian voters, anthropological examinations endeavor to defeat the constraints of hands on work dependent on a solitary, restricted territory.