**ELECTION AUTOMATION ANALYSIS**

Mid-Term Project Report

Master’s in computer science

By

Spandana Atagalla

Aasritha Bhimisetty

Divya Mahesha

Sowmya Kollipara

Under the guidance of

**Shubham Sawant**



Seidenberg School of Computer Science

At

Pace University

April,2024

**ABSTRACT**

This report conducts a thorough analysis of a congressional election, focusing on critical elements such as total votes cast, county-wise voting distribution, identification of the county with the highest voter turnout, and candidate performance. It begins by summarizing the total votes cast, establishing a foundational understanding for further analysis. Subsequently, it delves into an intricate breakdown of votes and their percentages across precincts, facilitating a nuanced comprehension of regional voting patterns. Through this analysis, the report pinpoints the county with the highest voter turnout, highlighting its significance within the electoral landscape. Additionally, it presents a detailed examination of candidate performance, delineating the number of votes secured by each candidate and their corresponding percentages of the total votes. This analysis culminates in the identification of the winning candidate, providing insight into the electoral victor. Furthermore, the report proposes modifications to the existing election auditing script for broader applicability and presents examples showcasing their efficacy in different election scenarios. Through these efforts, the report aims to refine election analysis and auditing methodologies, ultimately contributing to the integrity and transparency of democratic processes.

**LIST OF FIGURES**

2.1.1 Data Set 8

2.2.1 Data Set without Duplicates 9

2.2.2 Missing values in dataset 11

2.2.3 Grouping data with candidate 12

2.3.1 Histogram for ballot ID’s 14

2.3.2 Bar graph for candidate 17

2.3.3 Results 25

**TABLE OF CONTENTS**

1 INTRODUCTION 5

1.1 Introduction 5

1.2 Aim of the project 5

1.3 Scope of the Project 6

2 Methodology 7

2.1 Data Sourcing 7

2.2 Data Cleaning 9

2.3 Data Visualization 13

3 Analysis 18

4 Result 25

5 Conclusion 26

**Chapter 1**

**INTRODUCTION**

* 1. Introduction

This project focuses on a thorough election audit conducted through Python programming to efficiently analyze and visualize election data. The primary aim is to deliver precise and reliable results for a recent local congressional election. The audit encompasses several key elements, such as total votes, voter turnout per county, percentage of votes per county, identification of the county with the highest turnout, and determining the winning candidate along with their vote count and percentage. By leveraging Python's capabilities, the project seeks to streamline the auditing process and provide actionable insights into the electoral landscape. Ultimately, it endeavors to enhance understanding of Python for data analysis while contributing to the integrity and transparency of the democratic process.

* 1. Aim of the Project

The project aims to audit a recent local congressional election using Python and data analysis techniques. It focuses on determining the total votes cast, analyzing votes by county, identifying the county with the highest turnout, and analyzing candidate performance. Additionally, it proposes modifications to the auditing script for broader use across elections and provides examples to demonstrate its effectiveness. Overall, the project aims to improve understanding of Python for data analysis and provide insights for election auditing.

* 1. Scope of the Project

The project's scope involves conducting a thorough election audit using Python. It includes tasks like determining total votes, analyzing county-wise voter turnout and percentages, identifying the county with the highest turnout, and determining the winning candidate. Additionally, the project proposes script modifications for wider applicability and provides examples to illustrate their effectiveness. However, the project does not involve running elections or influencing results. Its focus is solely on analyzing and auditing election data to ensure accuracy and credibility. Overall, the project aims to simplify Python data analysis and contribute to transparent electoral processes.

**CHAPTER 2**

**METHODOLOGY**

**2.1 Data Sourcing**

The project utilizes a dataset sourced from the Colorado Board of Elections, contained within a CSV file named "election\_results.csv." This dataset is rich with pertinent information, including total votes cast, candidate identities, and county affiliations. Its comprehensive nature makes it an ideal resource for conducting a meticulous election audit, enabling deep insights into critical aspects such as voter turnout, candidate popularity, and voting patterns across different counties. The project's primary focus centers on executing a thorough and transparent election audit, leveraging Python's robust data analysis and data visualization capabilities. The script navigates through the dataset, extracting invaluable insights while maintaining clarity and conciseness in presenting the results. The dataset's suitability is crucial for this endeavor, as its contents seamlessly align with the objectives of the election audit. By dissecting the data, the script facilitates a granular examination of voter behaviors and candidate performances, laying the groundwork for informed decision-making and policy formulation.

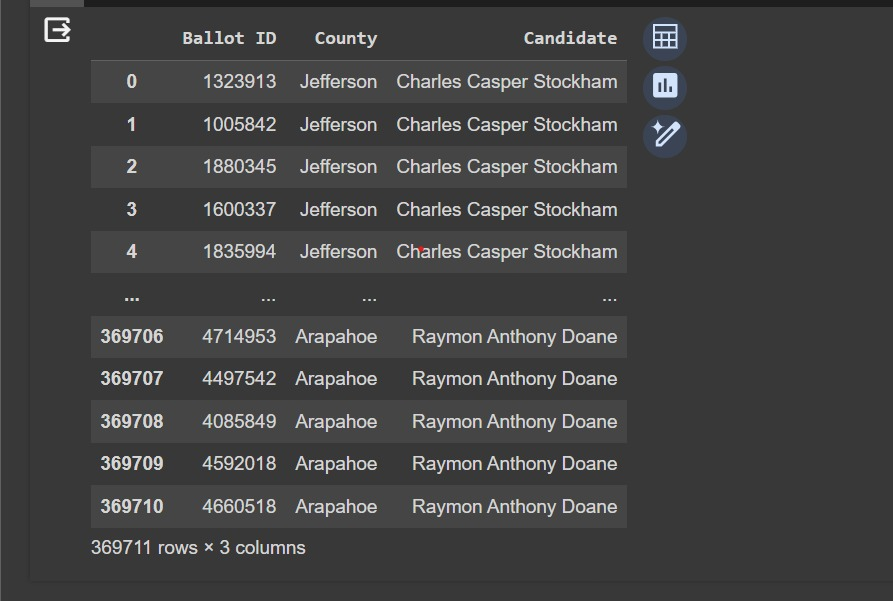


Figure 2.1.1: Data Set

**2.2 Data Processing**

Ensuring clean data is key for reliable analysis. Data cleaning tackles two frequent issues: duplicates (data points appearing multiple times) and missing values (empty data points). Duplicates can be removed entirely or merged if slightly different. Missing values require strategies like deletion (if minimal), imputation (estimating values), or adding a new category (for informative missing reasons).

**2.2.1 Source Code**

import pandas as pd

data = pd.read\_csv('/content/election\_results.csv')

# Remove duplicates based on all columns

data\_no\_duplicates = data.drop\_duplicates()

# Print the DataFrame without duplicates

print("DataFrame without duplicates:")

print(data\_no\_duplicates)

A screenshot of a computer

Description automatically generated

Figure 2.2.1: Data set without duplicates

**2.2.2 Source Code**

import numpy as np

# Checking for missing values

missing\_values = data.isnull().sum()

print("Missing values by columns:\n", missing\_values)

# Removing rows with missing values

data\_cleaned = data.dropna()

# Filling missing values with a constant

data\_filled\_constant = data.fillna(value=0)

# Filling missing values with the mean of the column

data\_filled\_mean = data.fillna(data.mean(numeric\_only=True))

# Filling missing values with the median of the column

data\_filled\_median = data.fillna(data.median(numeric\_only=True)

# Filling missing values with the mode of the column

data\_filled\_mode = data.fillna(data.mode().iloc[0])

# Using forward-fill method to fill missing values

data\_filled\_ffill = data.fillna(method='ffill')

# Using backward-fill method to fill missing values

data\_filled\_bfill = data.fillna(method='bfill')

# Interpolating missing values

data\_filled\_interpolate = data.interpolate()

# Choose the appropriate method to handle missing values based on your specific dataset and save it as 'data\_preprocessed'

data\_preprocessed = data\_filled\_mean

.A black screen with white text

Description automatically generated

Figure 2.2.2: checking for missing values in data set

Data grouping by candidate name is a technique used to organize information based on individual candidates. It essentially sorts data entries where the "candidate name" field acts as the category. This allows you to analyze and compare information specific to each candidate.

**2.2.2 Source Code**

# Group the DataFrame by candidate names

grouped\_candidates = data.groupby('Candidate')

# Iterate over the groups and perform operations if needed

for candidate, group in grouped\_candidates:

print(candidate)

print(group)

A black rectangular object with white text

Description automatically generated

Figure 2.2.3: Grouping data with candidate.

**2.3 Data Visualization**

Histogram for Ballot IDs and frequency shows how often each Ballot ID appears in a dataset. It uses bars to represent Ballot ID ranges on the x-axis and the frequency of each range on the y-axis. By looking at the histogram, you can see which Ballot IDs occur more frequently and get an overall idea of how they are distributed in the dataset. This helps in understanding the dataset's characteristics and identifying any patterns or outliers.

**2.3.1 Source Code**

import matplotlib.pyplot as plt

plt.hist(data['Ballot ID'], bins=20, color='skyblue', edgecolor='black')

plt.xlabel('Ballot ID')

plt.ylabel('Frequency')

plt.title('Distribution of Ballot IDs')

plt.show()

A graph of distribution of ballot ids

Description automatically generated

Figure 2.3.1: HISTOGRAM for Ballot Ids

Bar graph comparing counties and their frequencies simply shows how often each county appears in a dataset. Each county is represented by a bar, with the height of the bar indicating its frequency. This graph helps easily identify which counties are most common in the dataset and compare their frequencies.

**2.3.2 Source Code**

# Bar chart for County

county\_counts = data['County'].value\_counts()

plt.bar(county\_counts.index, county\_counts.values, color='lightgreen')

plt.xlabel('County')

plt.ylabel('Frequency')

plt.title('Frequency of Votes by County')

plt.xticks(rotation=45, ha='right')

plt.show()

A graph of a number of votes

Description automatically generated

Figure 2.3.2: BAR GRAPH for County

Bar graph showing candidates and their frequencies simply displays how often each candidate appears in a dataset. Each candidate is represented by a bar, with the height of the bar indicating their frequency. This graph makes it easy to see which candidates are most common and compare their frequencies.

**2.3.3 Source code**

# Bar chart for Candidate

candidate\_counts = data['Candidate'].value\_counts()

plt.bar(candidate\_counts.index, candidate\_counts.values, color='lightcoral')

plt.xlabel('Candidate')

plt.ylabel('Frequency')

plt.title('Frequency of Votes by Candidate')

plt.xticks(rotation=45, ha='right')

plt.show()

A graph with red rectangular bars

Description automatically generated with medium confidence

Figure 2.3.2: BAR GRAPH for Candidate

**Analysis**

**Source Code**

import csv

import os

file\_to\_load = os.path.join("/content/election\_results.csv")

file\_to\_save = os.path.join("/content/election\_analysis.txt")

# Initialize a total vote counter.

total\_votes = 0

# Candidate Options and candidate votes.

candidate\_options = []

candidate\_votes = {}

# Create a county list and county votes dictionary.

county\_list = []

county\_votes = {}

# Track the winning candidate, vote count and percentage

winning\_candidate = ""

winning\_count = 0

winning\_percentage = 0

# Track the largest county and county voter turnout.

largest\_county\_turnout = ""

largest\_county\_turnout\_count = 0

largest\_county\_percentage = 0

# Read the csv and convert it into a list of dictionaries

with open(file\_to\_load) as election\_data:

# Read the file object with the reader function.

reader = csv.reader(election\_data)

# Read the header

header = next(reader)

# For each row in the CSV file.

for row in reader:

# Add to the total vote count

total\_votes += 1

# Get the candidate name from each row.

candidate\_name = row[2]

# Extract the county name from each row.

county\_name = row[1]

# If the candidate does not match any existing candidate add it to

# the candidate list

if candidate\_name not in candidate\_options:

# Add the candidate name to the candidate list.

candidate\_options.append(candidate\_name)

# And begin tracking that candidate's voter count.

candidate\_votes[candidate\_name] = 0

# Add a vote to that candidate's count

candidate\_votes[candidate\_name] += 1

# Write a decision statement that checks that the

# county does not match any existing county in the county list.

if county\_name not in county\_list:

# Add the existing county to the list of counties.

county\_list.append(county\_name)

# Begin tracking the county's vote count.

county\_votes[county\_name] = 0

# Add a vote to that county's vote count.

county\_votes[county\_name] += 1

# Save the results to our text file.

with open(file\_to\_save, "w") as txt\_file:

# Print the final vote count (to terminal)

election\_results = (

f"\nElection Results\n"

f"-------------------------\n"

f"Total Votes: {total\_votes:,}\n"

f"-------------------------\n\n"

f"County Votes:\n"

)

print(election\_results, end="")

# Save the final vote count to the text file.

txt\_file.write(election\_results)

# Write a repetition statement to get the county from the county dictionary.

for county in county\_list:

# Initialize a variable to hold the county’s votes as they are retrieved from the county votes dictionary.

county\_vote = county\_votes.get(county)

# Calculate the percent of total votes for the county.

county\_vote\_percentage = float(county\_vote) / float(total\_votes) \* 100

# Print the county results to the terminal.

county\_results = f"{county}: {county\_vote\_percentage:.1f}% ({county\_vote:,})\n"

# Print the counties to test.

print(county\_results)

# Save the county votes to a text file.

txt\_file.write(county\_results)

# Write a decision statement to determine the winning county and get its vote count.

if (county\_vote > largest\_county\_turnout\_count) and (

county\_vote\_percentage > largest\_county\_percentage

):

# True

largest\_county\_turnout\_count = county\_vote

largest\_county\_percentage = county\_vote\_percentage

largest\_county\_turnout = county

# Print the county with the largest turnout to the terminal.

winning\_county\_print = (

f"-------------------------\n"

f"Largest County Turnout: {largest\_county\_turnout}\n"

f"-------------------------\n"

)

print(winning\_county\_print)

# Save the county with the largest turnout to a text file.

txt\_file.write(winning\_county\_print)

# Save the final candidate vote count to the text file.

for candidate\_name in candidate\_votes:

# Retrieve vote count and percentage

votes = candidate\_votes.get(candidate\_name)

vote\_percentage = float(votes) / float(total\_votes) \* 100

candidate\_results = f"{candidate\_name}: {vote\_percentage:.1f}% ({votes:,})\n"

# Print each candidate's voter count and percentage to the

# terminal.

print(candidate\_results)

# Save the candidate results to our text file.

txt\_file.write(candidate\_results)

# Determine winning vote count, winning percentage, and candidate.

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

winning\_count = votes

winning\_candidate = candidate\_name

winning\_percentage = vote\_percentage

# Print the winning candidate (to terminal)

winning\_candidate\_summary = (

f"-------------------------\n"

f"Winner: {winning\_candidate}\n"

f"Winning Vote Count: {winning\_count:,}\n"

f"Winning Percentage: {winning\_percentage:.1f}%\n"

f"-------------------------\n"

)

print(winning\_candidate\_summary)

# Save the winning candidate's name to the text file

txt\_file.write(winning\_candidate\_summary)

**Result**

The Election-Audit Results present a comprehensive overview of the recent election, with a total of 369,711 votes cast. Notably, Denver County exhibited the highest turnout, with 82.8% of the vote and 306,055 ballots counted. Diana DeGette emerged victorious, securing 73.8% of the total vote, amounting to 272,892 votes. The accuracy and credibility of these results are emphasized, inviting validation by the election commission and Colorado Board of Elections through the provided PyPoll\_Challenge.py script. This commitment to electoral integrity underscores the transparency and accountability of the electoral process.

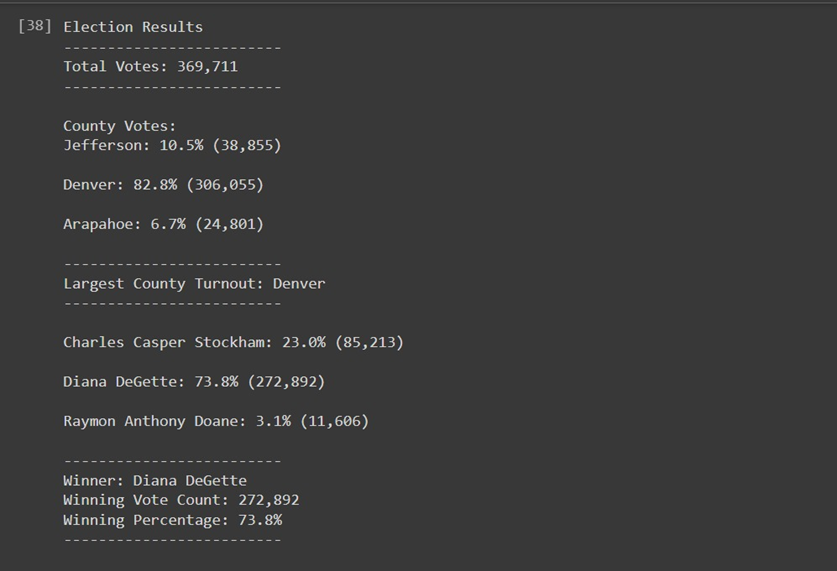


Figure 3 Results

**Conclusion**

**References**

**GITHUB Repository for Code**