Python Assignment Report

Ayushmaan Jay Singh (220276)

12 April 2024

1 Methodology

1.1 Data Preprocessing Steps

1.1.1 Loading Datasets

We load the training and test datasets using the Pandas library, allowing us to work with the data in a structured format within Python.

```
import pandas as pd

train_df = pd.read_csv('/content/train.csv')
test_df = pd.read_csv('/content/test.csv')
```

1.1.2 Handling Missing Values

```
train_df = train_df.dropna(axis=0)
test_df = test_df.dropna(axis=0)
```

To ensure the integrity of our data, we address any missing values by removing rows containing missing values from both the training and test datasets. This ensures that our datasets are complete and ready for subsequent analysis and modeling.

1.1.3 Feature Selection

```
features = ['Party', 'Criminal Case', 'Total Assets', 'Liabilities', 'state']
```

We begin by selecting the features for our analysis. The selected features include:

- Party
- Criminal Case
- Total Assets
- Liabilities
- State

Almost all ID, Candidate (names) and Consistuency in train.csv are unique, there's no need to include them in predicting education.

Explanation of why these features were selected:

1. **Party:** Reflects political affiliation, which can influence candidate qualifications and selection processes.
2. **Criminal Case:** Indicates legal history and potential societal biases, impacting candidate opportunities.
3. **Total Assets**: Reflects socioeconomic status and financial resources, influencing access to education.
4. **Liabilities:** Reflects financial stability and potential barriers to education and political participation.
5. **State:** Captures regional variations in education, politics, and socioeconomic conditions, providing context for candidate backgrounds and opportunities.

1.1.4 Encoding Categorical Columns

```
columns_to_encode = ['Party', 'state']
```

By encoding the categorical columns before using them in the modeling process, we ensure that the data is properly prepared for analysis, thereby improving the effectiveness of our predictive model.
To facilitate modeling, we encode categorical columns ('Party' and 'State') into numeric values. We utilize LabelEncoder from the scikit-learn library to transform categorical variables into numeric representations.
This step ensures consistency in encoding and prevents data leakage.

1.1.5 Extracting Numerical Values

```python
def extract_numerical_value(asset_str):
    if 'Crore' in asset_str:
        return int(float(asset_str.replace('Crore+', '')) * 10000000)
    elif 'Lac' in asset_str:
        return int(float(asset_str.replace('Lac+', '')) * 100000)
    elif 'Thou' in asset_str:
        return int(float(asset_str.replace('Thou+', '')) * 1000)
    elif 'Hund' in asset_str:
        return int(float(asset_str.replace('Hund+', '')) * 100)
    else:
        return 0
```

We define a custom function, extract_numerical_value , to convert string representations of asset values into numerical format. This function identifies the unit of measurement ($'Crore', 'Lac', 'Thou', 'Hund'$) and converts the values accordingly, taking into account the appropriate multiplier for each unit.

1.1.6 Processing Train Data

```python
X['Total Assets'] = X['Total Assets'].apply(extract_numerical_value)
X['Liabilities']=X['Liabilities'].apply(extract_numerical_value)
```

We update the feature matrix (X) with encoded columns and replace the $'TotalAssets'$ and $'Liabilities'$ columns with their corresponding numerical values using the extract_numerical_value function.

1.1.7 Encoding Target Variable

```python
Y = label_encoder.fit_transform(train_df['Education'])
```

We encode the target variable, 'Education', using LabelEncoder() to transform it into numerical labels (Y). This step is essential for classification tasks.

1.2 Processing Test Data

Similarly, we process the test data using the same encoding techniques applied to the training data. We encode the categorical columns and replace 'Total Assets' and 'Liabilities' with numerical values.

1.3 Transformation

In summary, the following transformations are encoding categorical variables into numeric labels using label encoding and then decoding them back into their original categorical values when needed.

```
train_df[col + '_encoded_into_numeric'] = label_encoders[col].fit_transform(train_df[col])
```

```
Y= label_encoder.fit_transform(train_df['Education'])
```

```
test_df[col + '_encoded_into_numeric']= label_encoder_test[col].fit_transform(test_df[col])
```

```
answers = label_encoder.inverse_transform(guess)
```

2 Experiment Details

Table 1: Model Performance **(Random Forest)**

Model	**Parameters Used**	**Approx. Accuracy and F1 Score**
Random Forest	Number of trees: 100 Max depth: 10 Minimum samples split: 5	Training Accuracy : 0.625242718446602 Testing Accuracy : 0.6709844559585493 Testing f1 Score: 0.3951556580466389
Random Forest	Number of trees: 100 Max depth: 1 Minimum samples split: 2	Training Accuracy : 0.2757281553398058 Testing Accuracy : 0.25194300518134716 Testing f1 Score: 0.040248318675633735
Random Forest	Number of trees: 300 Max depth: 15 Minimum samples split: 5	Training Accuracy : 0.8660194174757282 Testing Accuracy : 0.883419689119171 Testing f1 Score: 0.7501981215968693
Random Forest	Number of trees: 350 Max depth: 15 Minimum samples split: 5	Training Accuracy : 0.8718446601941747 Testing Accuracy : 0.883419689119171 Testing f1 Score: 0.7869642123359104
Random Forest	Number of trees: 400 Max depth: 17 Minimum samples split: 3	Training Accuracy : 0.9611650485436893 Testing Accuracy : 0.9585492227979274 Testing f1 Score: 0.9466228065515278
Random Forest	Number of trees: 400 Max depth: 17 Minimum samples split: 3	Training Accuracy : 0.9669902912621359 Testing Accuracy : 0.9682642487046632 Testing f1 Score: 0.9702557897265336
Random Forest	Number of trees: 200 Max depth: 11 Minimum samples split: 2	Training Accuracy : 0.8252427184466019 Testing Accuracy : 0.8018134715025906 Testing f1 Score: 0.6263394706649843
Random Forest	Number of trees: 250 Max depth: 15 Minimum samples split: 3	Training Accuracy : 0.9223300970873787 Testing Accuracy : 0.9397668393782384 Testing f1 Score: 0.9016612991081242
Random Forest	Number of trees: 550 Max depth: 21 Minimum samples split: 4	Training Accuracy : 0.9514563106796117 Testing Accuracy : 0.9527202072538861 Testing f1 Score: 0.9346638911884202
Random Forest	Number of trees: 450 Max depth: 19 Minimum samples split: 5	Training Accuracy : 0.9281553398058252 Testing Accuracy : 0.9119170984455959 Testing f1 Score: 0.841324263838386

Table 2: Model Performance **(KNeighborsClassifier)**

Model	**Parameters Used**	**Approx. Accuracy and F1 Score**
KNeighborsClassifier	Number of Neighbors: 41	Training Accuracy : 0.2854368932038835 Testing Accuracy : 0.28950777202072536 Testing f1 Score: 0.12229399331626234
KNeighborsClassifier	Number of Neighbors: 81	Training Accuracy : 0.287378640776699 Testing Accuracy : 0.2713730569948187 Testing f1 Score: 0.10069567548809724
KNeighborsClassifier	Number of Neighbors: 3	Training Accuracy : 0.5300970873786408 Testing Accuracy : 0.5194300518134715 Testing f1 Score: 0.38352707509420725
KNeighborsClassifier	Number of Neighbors: 31	Training Accuracy : 0.3048543689320388 Testing Accuracy : 0.29468911917098445 Testing f1 Score: 0.13021575021834703
KNeighborsClassifier	Number of Neighbors: 21	Training Accuracy : 0.3262135922330097 Testing Accuracy : 0.32124352331606215 Testing f1 Score: 0.1495914844091728
KNeighborsClassifier	Number of Neighbors: 15	Training Accuracy : 0.36893203883495146 Testing Accuracy : 0.32966321243523317 Testing f1 Score: 0.16492239065885245
KNeighborsClassifier	Number of Neighbors: 10	Training Accuracy : 0.4174757281553398 Testing Accuracy : 0.36139896373056996 Testing f1 Score: 0.197966505003024
KNeighborsClassifier	Number of Neighbors: 7 (In Keggle, this actually showed lesser score)	Training Accuracy : 0.38058252427184464 Testing Accuracy : 0.3957253886010363 Testing f1 Score: 0.2373491457950884
KNeighborsClassifier	Number of Neighbors: 12	Training Accuracy : 0.3572815533980582 Testing Accuracy : 0.35751295336787564 Testing f1 Score: 0.19969301626592967
KNeighborsClassifier	Number of Neighbors: 11	Training Accuracy : 0.37087378640776697 Testing Accuracy : 0.3639896373056995 Testing f1 Score: 0.19512077256174779

Table 3: Model Performance **(SVM)**

Model	**Parameters Used**	**Approx. Accuracy and F1 Score**
SVM	C=1.0	Training Accuracy : 0.27184466019417475 Testing Accuracy : 0.25582901554404147 Testing f1 Score: 0.04325454573513015
SVM	kernel=rbf C=1.0	Training Accuracy : 0.287378640776699 Testing Accuracy : 0.2506476683937824 Testing f1 Score: 0.04201508181100017
SVM	kernel=poly C=1.0 degree=2	Training Accuracy : 0.23495145631067962 Testing Accuracy : 0.2661917098445596 Testing f1 Score: 0.04260018408049281
SVM	kernel=sigmoid C=1.0 coef0=0.0	Training Accuracy : 0.22330097087378642 Testing Accuracy : 0.2260362694300518 Testing f1 Score: 0.07321408612065834
SVM	kernel=rbf C=1.0	Training Accuracy : 0.26796116504854367 Testing Accuracy : 0.2603626943005181 Testing f1 Score: 0.05293342558683114

Based on the experiments conducted, it is evident that Random Forest outperformed both KNN and SVM classifiers in terms of classification performance. The highest F1 score was achieved by Random Forest with a configuration of 400 trees, a max depth of 17, and a minimum samples split of 3, resulting in a testing F1 score of approximately 0.970.

KNN performed relatively well but was slightly inferior to Random Forest, while SVM exhibited the poorest performance with an average F1 score around 0.05. This indicates that the decision boundary learned by SVM might not have been able to effectively separate the classes in the dataset.

The highest accuracy achieved on the testing set by Random Forest was approximately 96.83 percent, which demonstrates its robustness and effectiveness in classifying the data. Overall, Random Forest proved to be the most suitable model for the given dataset, providing superior performance compared to KNN and SVM.

2.1 Data Insights

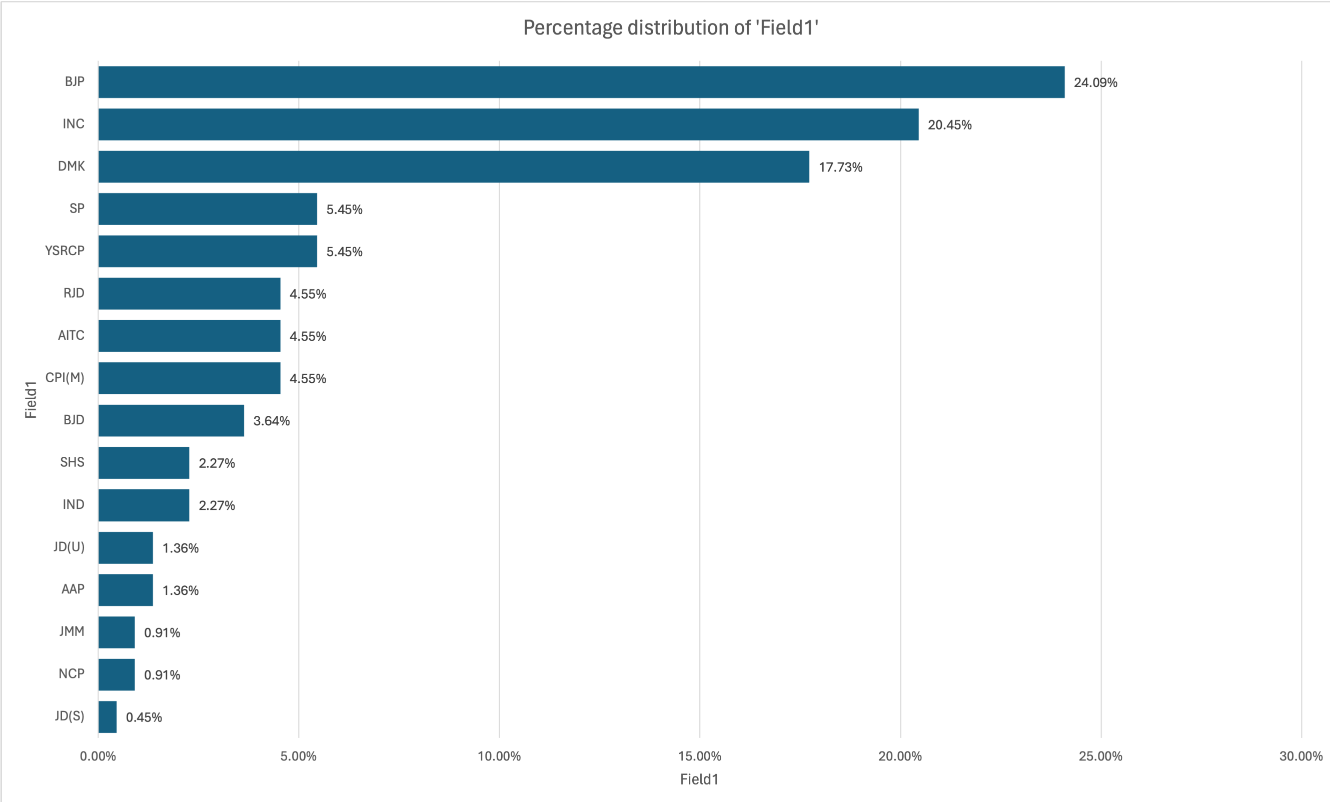
2.1.1 Graphs and Plots

- The percentage distribution of parties with candidates having the most criminal records.

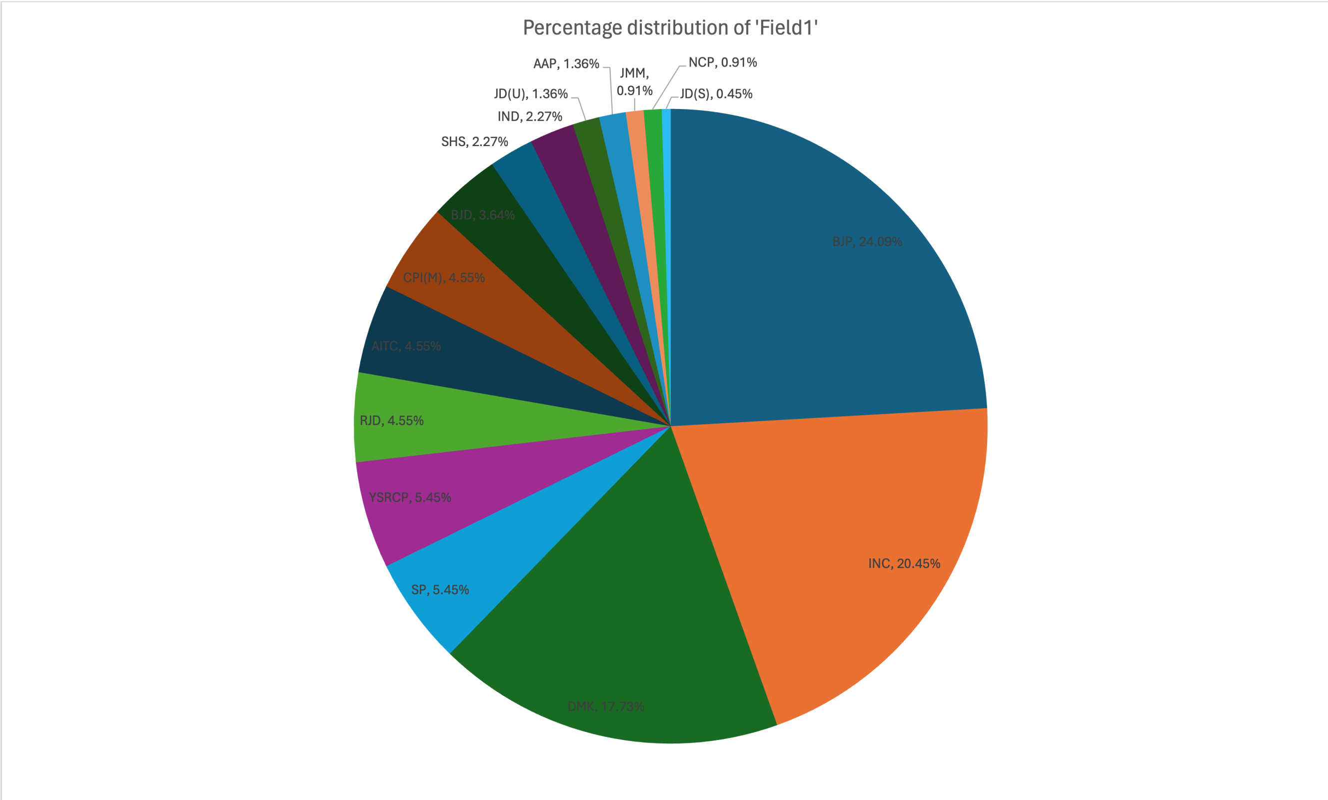
  1. For this plot , in the excel I sorted the list by the number of criminal cases.
  2. Then I made the criteria for most criminal record as $\geq$5.
  3. Then there were 221 such candidates.
  4. Then , I just simply plotted the data chart for the data , party-wise.

Table 4: Count of Criminal Candidates ($\geq$ 5 Cases)

Party	**Count (%)**	**Total Count**
BJP	24.09	53
INC	20.45	45
DMK	17.73	39
SP	5.45	12
YSRCP	5.45	12
RJD	4.55	10
AITC	4.55	10
CPI(M)	4.55	10
BJD	3.64	8
SHS	2.27	5
IND	2.27	5
JD(U)	1.36	3
AAP	1.36	3
JMM	0.91	2
NCP	0.91	2
JD(S)	0.45	1
Grand Total	100.00	220



Percentage distribution of 'Field1'
BJP
INC
DMK
SP
YSRCP
RJD
AITC
CPI(M)
BJD
SHS
IND
JD(U)
AAP
JMM
NCP
JD(S)
24.09%
20.45%
17.73%
5.45%
5.45%
4.55%
4.55%
4.55%
3.64%
2.27%
2.27%
1.36%
1.36%
0.91%
0.91%
0.45%
0.00%
5.00%
10.00%
15.00%
20.00%
25.00%
30.00%
Field1
Field1



The data provided reveals a concerning reality regarding the prevalence of serious criminal charges among political candidates. Among the major political parties, the BJP leads with 53 candidates facing serious criminal allegations, closely followed by the INC with 45 candidates. Other prominent parties like the DMK, SP, and YSRCP also have significant numbers of candidates with serious criminal charges against them, ranging from 12 to 39.

Additionally, smaller parties such as RJD, AITC, and CPI(M) each have 10 candidates with serious criminal cases. This data underscores the widespread issue across the political spectrum, where candidates from various parties have serious criminal allegations against them. Despite efforts to address this issue, the presence of candidates with criminal backgrounds remains a significant concern in the electoral landscape.

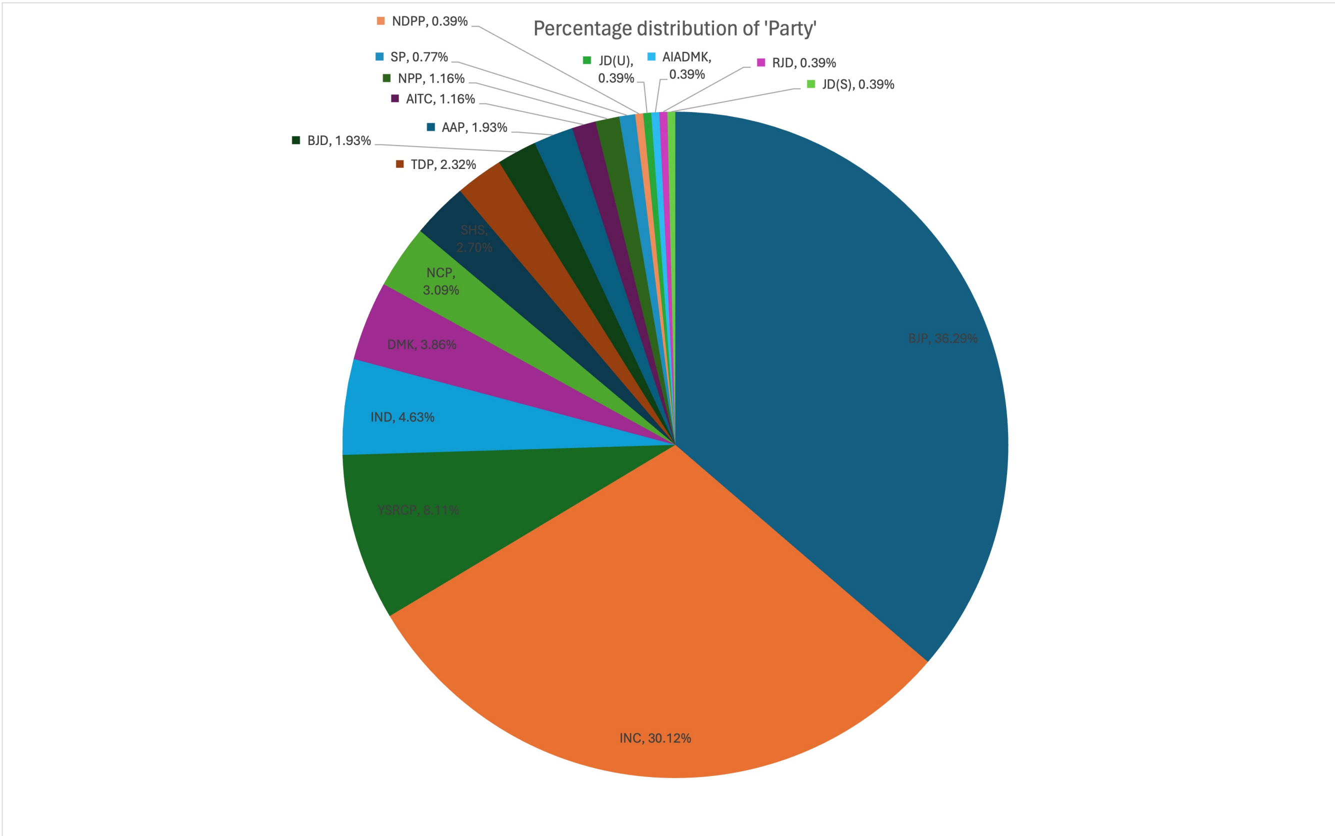
The total count of candidates with serious criminal charges stands at 220, highlighting the scale of the issue. It emphasizes the need for stricter scrutiny and measures to ensure the integrity of the electoral process and the accountability of those seeking public office. Addressing this challenge is crucial for upholding the principles of democracy and fostering public trust in the political system.

Here, the Field 1 is Count of Criminal Candidates (5 Cases)

- The percentage distribution of parties with the most wealthy candidates.
  - The percentage distribution of parties with candidates having the most wealth (= Assets- Liabilities).
    1. For this plot , in the excel I sorted the list by the Wealth Column.
    2. Then I made the criteria for Wealthy Candidate as Wealth $\geq$ 15 Crores.
    3. Then there were 259 such candidates.
    4. Then , I just simply plotted the data chart for the data , party-wise.

Table 5: Count of Parties

Party	**Count**	**Percentage**
BJP	94	36.29%
INC	78	30.12%
YSRCP	21	8.11%
IND	12	4.63%
DMK	10	3.86%
NCP	8	3.09%
SHS	7	2.70%
TDP	6	2.32%
BJD	5	1.93%
AAP	5	1.93%
AITC	3	1.16%
NPP	3	1.16%
SP	2	0.77%
NDPP	1	0.39%
JD(U)	1	0.39%
AIADMK	1	0.39%
RJD	1	0.39%
JD(S)	1	0.39%
Grand Total	**259**	**100.00%**



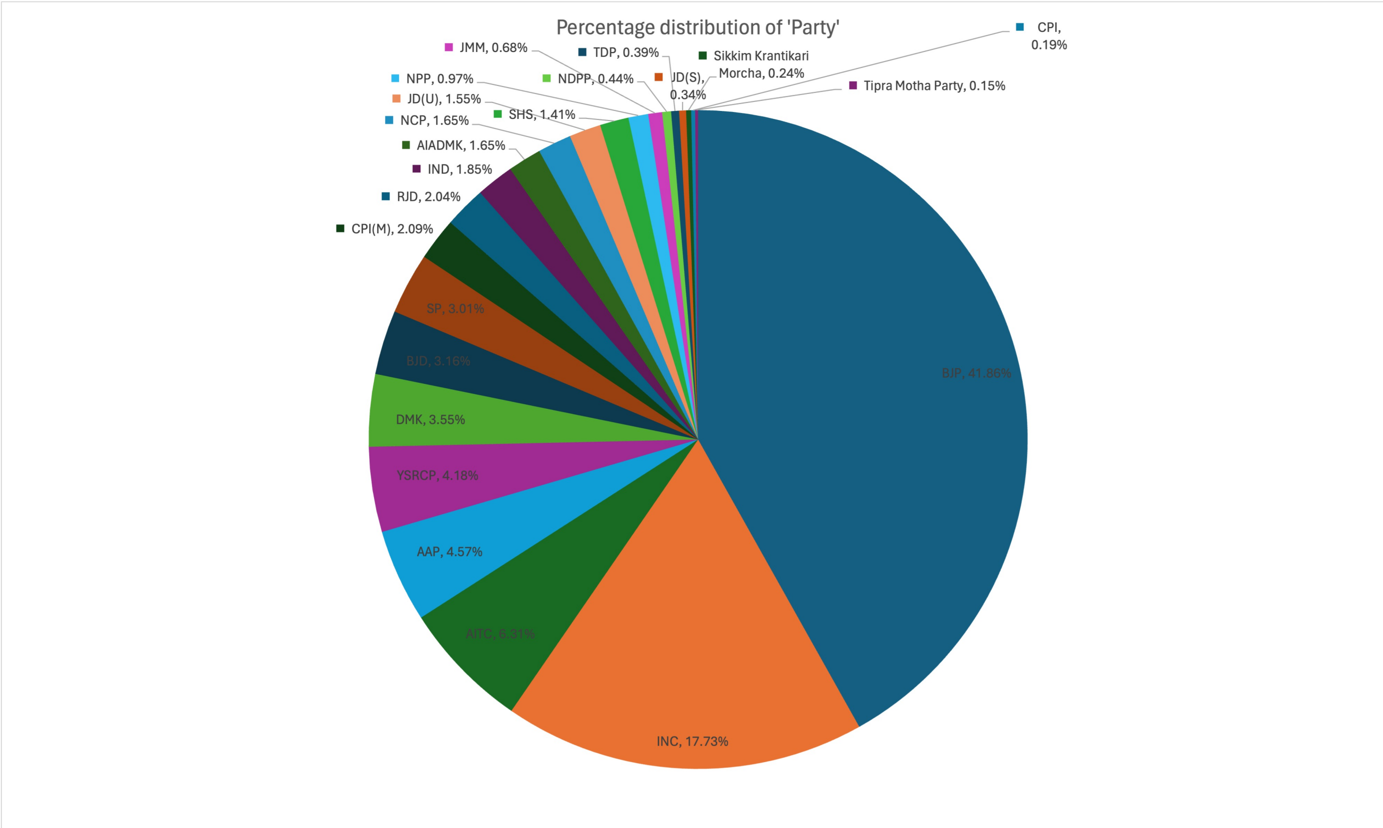
This table represents the distribution of the most wealthy candidates by party. It displays the count and percentage of wealthy candidates for each party. The data shows that the BJP and INC have the highest number of wealthy candidates, with 36.29 percent and 30.12 percent respectively, followed by YSRCP, IND, and DMK.

2.1.2 Insights

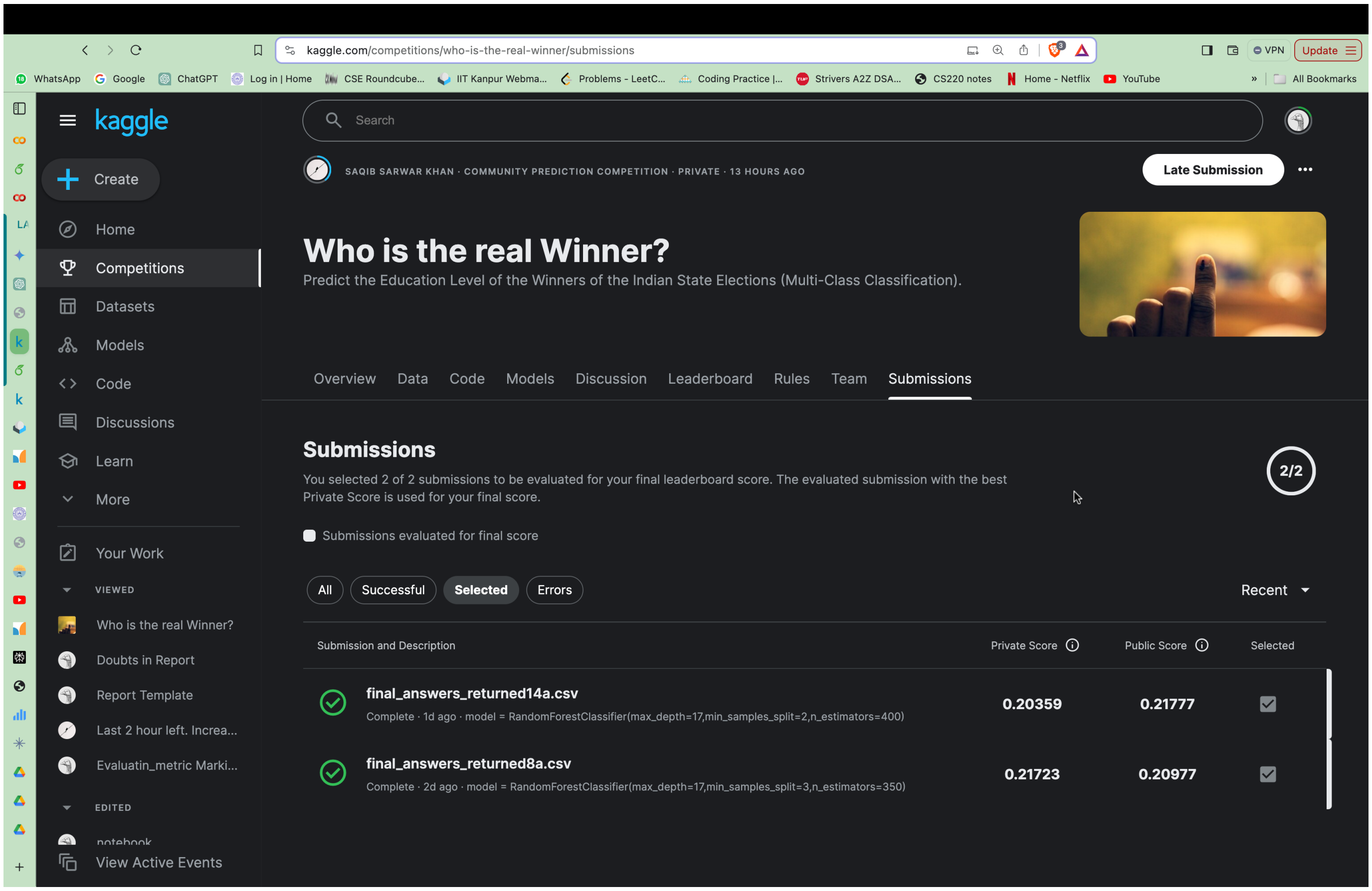
I have also analysed the Candidate Distribution Across Parties

Table 6: Count of Parties

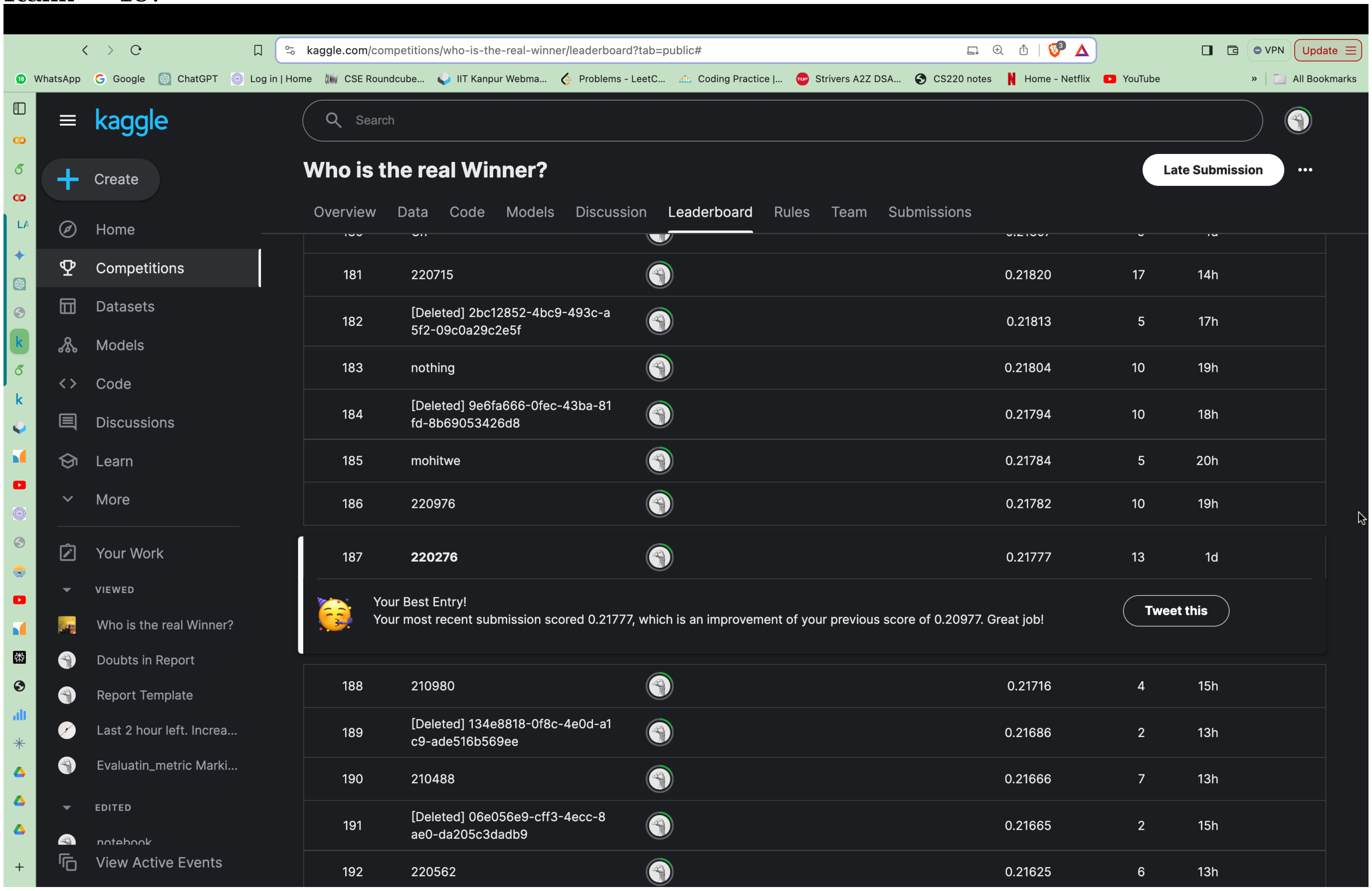
Party	**Percentage**
BJP	41.86%
INC	17.73%
AITC	6.31%
AAP	4.57%
YSRCP	4.18%
DMK	3.55%
BJD	3.16%
SP	3.01%
CPI(M)	2.09%
RJD	2.04%
IND	1.85%
AIADMK	1.65%
NCP	1.65%
JD(U)	1.55%
SHS	1.41%
NPP	0.97%
JMM	0.68%
NDPP	0.44%
TDP	0.39%
JD(S)	0.34%
Sikkim Krantikari Morcha	0.24%
CPI	0.19%
Tipra Motha Party	0.15%
Grand Total	**100.00%**


Percentage distribution of 'Party'
CPI, 0.19%
JMM, 0.68%
TDP, 0.39%
Sikkim Krantikari Morcha, 0.24%
NPP, 0.97%
NDPP, 0.44%
JD(S), 0.34%
Tipra Motha Party, 0.15%
JD(U), 1.55%
SHS, 1.41%
NCP, 1.65%
AIADMK, 1.65%
IND, 1.85%
RJD, 2.04%
CPI(M), 2.09%
SP, 3.01%
BJD, 3.16%
BJP, 41.86%
DMK, 3.55%
YSRCP, 4.18%
AAP, 4.57%
AITC, 6.31%
INC, 17.73%

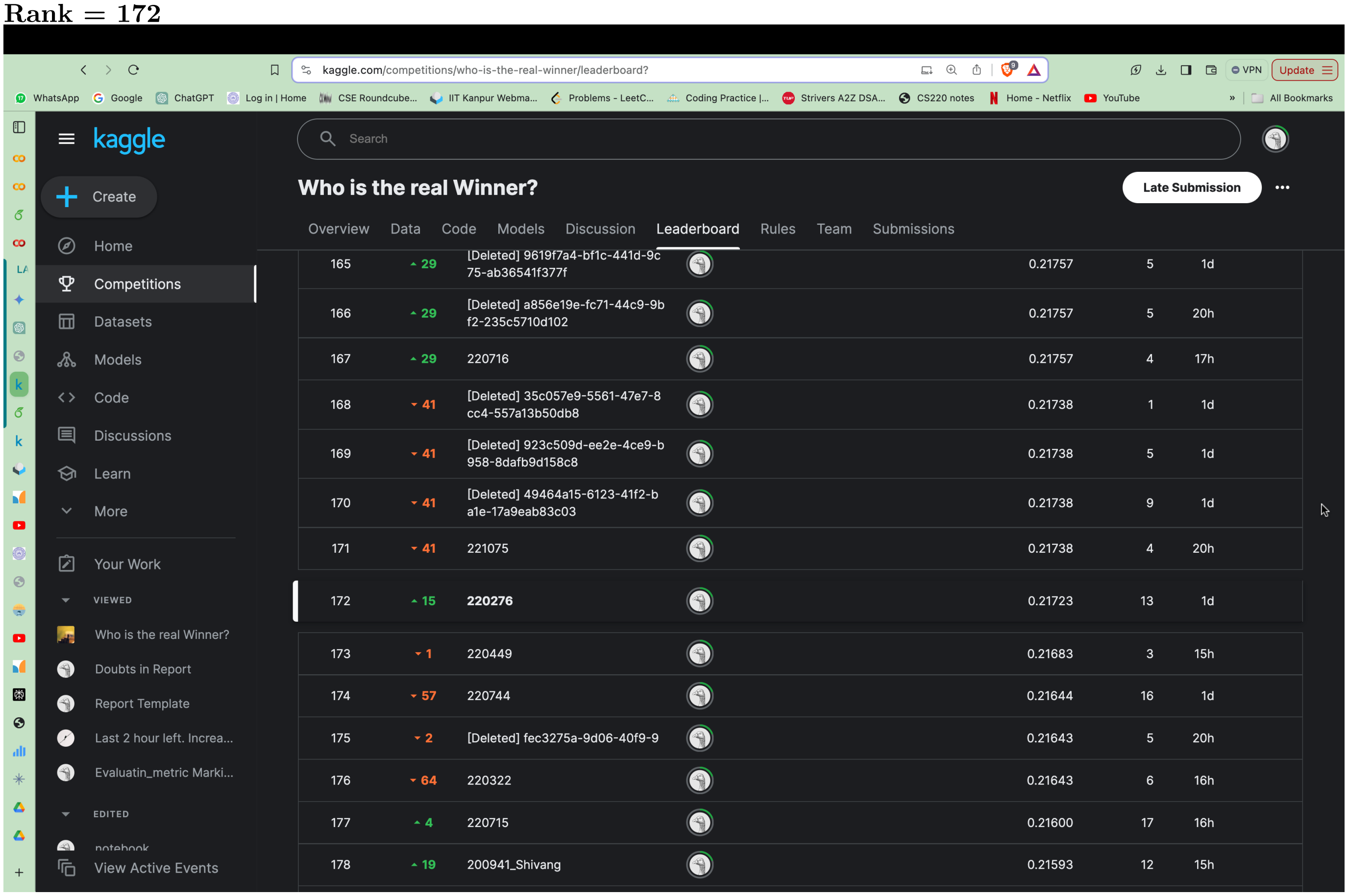
3 Results and F1 Score



Public LeaderBoard
Rank = 187



Private LeaderBoard
Rank = 172



4 References

Getting started tutorials — pandas 2.2.1 documentation
Citing
A Gentle Visual Intro to Data Analysis in Python Using Pandas – Jay Alammar
Citing
DataFrame — pandas 2.2.1 documentation
Citing
NumPy
Citing
A Visual Intro to NumPy and Data Representation – Jay Alammar
Citing
Scikit-learn
Citing
Matplotlib
Citing
Seaborn
Citing

5 GitHub Link

Link for Repository

https://github.com/aruj1207/Python-Assignment-CS253