Methodology

Read the file and visualize the input file structure(different features used to predict the model) and different different value taken by feature Education

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler,
PolynomialFeatures
import pandas as pd
import re
```

Loading the Data

```
df = pd.read_csv('train.csv')

test_data = pd.read_csv('train.csv')

df.head()
```

```
   ID              Candidate         Constituency ∇ Party  Criminal Case
\
0   0            M.K. Mohan              ANNA NAGAR   DMK               4

1   1  Khatik Ramesh Prasad            KARERA (SC)   BJP               0

2   2      Dr. Mantar Gowda               MADIKERI   INC               0

3   3          Kundan Kumar              BEGUSARAI   BJP               0

4   4       Swapan Majumder  BANGAON DAKSHIN (SC)   BJP               2


  Total Assets Liabilities           state      Education
0   211 Crore+    2 Crore+      TAMIL NADU       8th Pass
1     1 Crore+           0  MADHYA PRADESH      12th Pass
2     7 Crore+     22 Lac+       KARNATAKA  Post Graduate
3     9 Crore+     24 Lac+           BIHAR  Post Graduate
4     2 Crore+     61 Lac+     WEST BENGAL       8th Pass
```

Visualizing Different values taken by features "Education". It will help in multiclass classificaton .

```
for column in df.columns:
    unique_values = df[column].unique()
    if column == 'Education':  # Add a colon here
        print(f"Unique values in column '{column}':")
        print(unique_values)
```

```
Unique values in column 'Education':
['8th Pass' '12th Pass' 'Post Graduate' 'Graduate Professional'
'Graduate'
 '10th Pass' 'Others' 'Doctorate' 'Literate' '5th Pass']
```

Printing info for getting more information about data

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2059 entries, 0 to 2058
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ID              2059 non-null   int64
 1   Candidate       2059 non-null   object
 2   Constituency ∇  2059 non-null   object
 3   Party           2059 non-null   object
 4   Criminal Case   2059 non-null   int64
 5   Total Assets    2059 non-null   object
 6   Liabilities     2059 non-null   object
 7   state           2059 non-null   object
 8   Education       2059 non-null   object
dtypes: int64(2), object(7)
memory usage: 144.9+ KB
```

Processing the data (converting string to float)

```
# Preprocess the test data

df['Total Assets'] = df['Total Assets'].apply(lambda x:
float(re.sub('[^0-9.]', '', str(x))))
df['Liabilities'] = df['Liabilities'].apply(lambda x:
float(re.sub('[^0-9.]', '', str(x))))
```

Encoding categorical variables:

Encoding the 'Party' column: The code uses LabelEncoder to encode the 'Party' column, converting categorical values into numeric labels. Encoding the 'state' column: A custom dictionary is created to map each unique state to an integer. The 'state' column is then replaced with the integer labels.

```
label_encoder_party = LabelEncoder()
df['Party'] = label_encoder_party.fit_transform(df['Party'])

# Create a dictionary to map states to integers
state_to_int = {state: idx for idx, state in
enumerate(df['state'].unique())}
```

```
# Replace states with integers
df['State_Label'] = df['state'].map(state_to_int)

print(df['State_Label'])
```

```
0         0
1         1
2         2
3         3
4         4
        ..
2054    11
2055    16
2056     5
2057     7
2058    18
Name: State_Label, Length: 2059, dtype: int64
```

```
# Define features and target variable
X = df[['Criminal Case', 'Total Assets', 'Liabilities', 'Party',
'State_Label']]
y = df['Education']

print(X)
```

```
      Criminal Case  Total Assets  Liabilities  Party  State_Label
0                 4         211.0          2.0      7            0
1                 0           1.0          0.0      4            1
2                 0           7.0         22.0      8            2
3                 0           9.0         24.0      4            3
4                 2           2.0         61.0      4            4
...             ...           ...          ...    ...          ...
2054              1          61.0         10.0      5           11
2055              0           2.0          8.0      8           16
2056              0          13.0         85.0      4            5
2057              1          25.0         94.0     13            7
2058              0          11.0          0.0      4           18

[2059 rows x 5 columns]
```

Scaling features:

MinMaxScaler is used to scale the features ('Criminal Case', 'Total Assets', 'Liabilities', 'Party', 'State_Label') to a range between 0 and 1. Creating interaction termract.

```
# Scale the features between 0 and 1
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

PolynomialFeatures is used to create interaction terms (combinations of features) up to the second degree (interaction_only=True). The interaction terms are stored in the variable X_interact.

```
# Create interaction terms up to the second degree
poly = PolynomialFeatures(2, interaction_only=True)
X_interact = poly.fit_transform(X_scaled)
```

Experiment Details

Data Insights

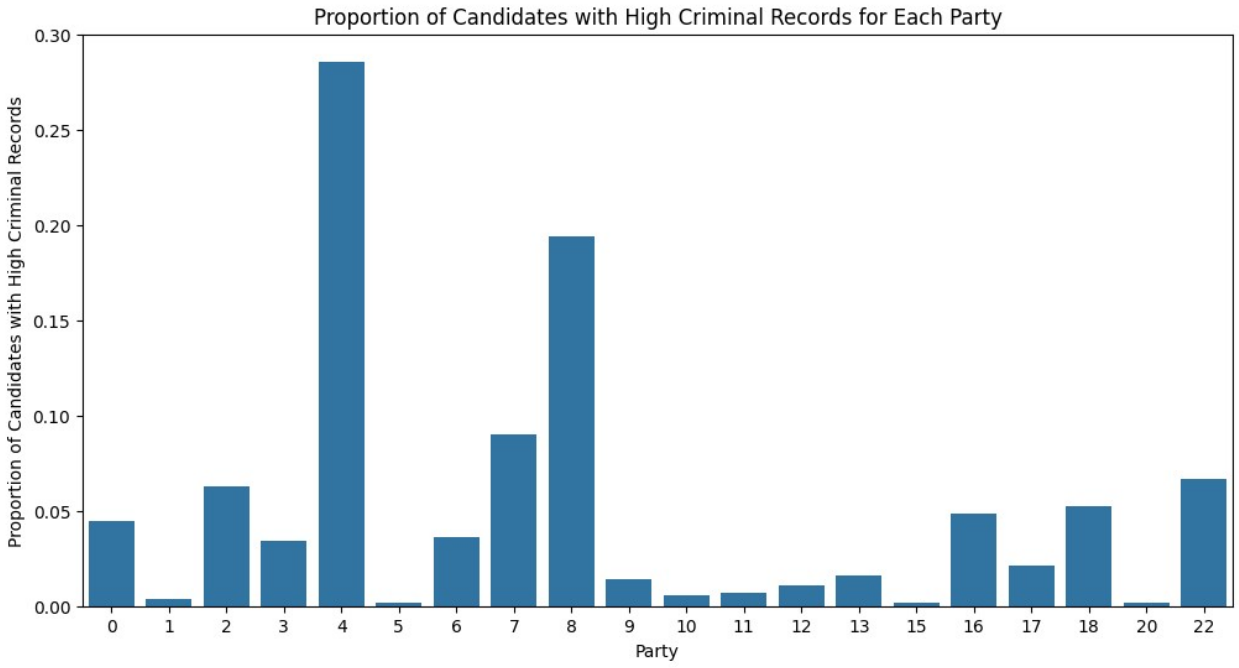
The percentage distribution of parties with candidates having the most criminal records.

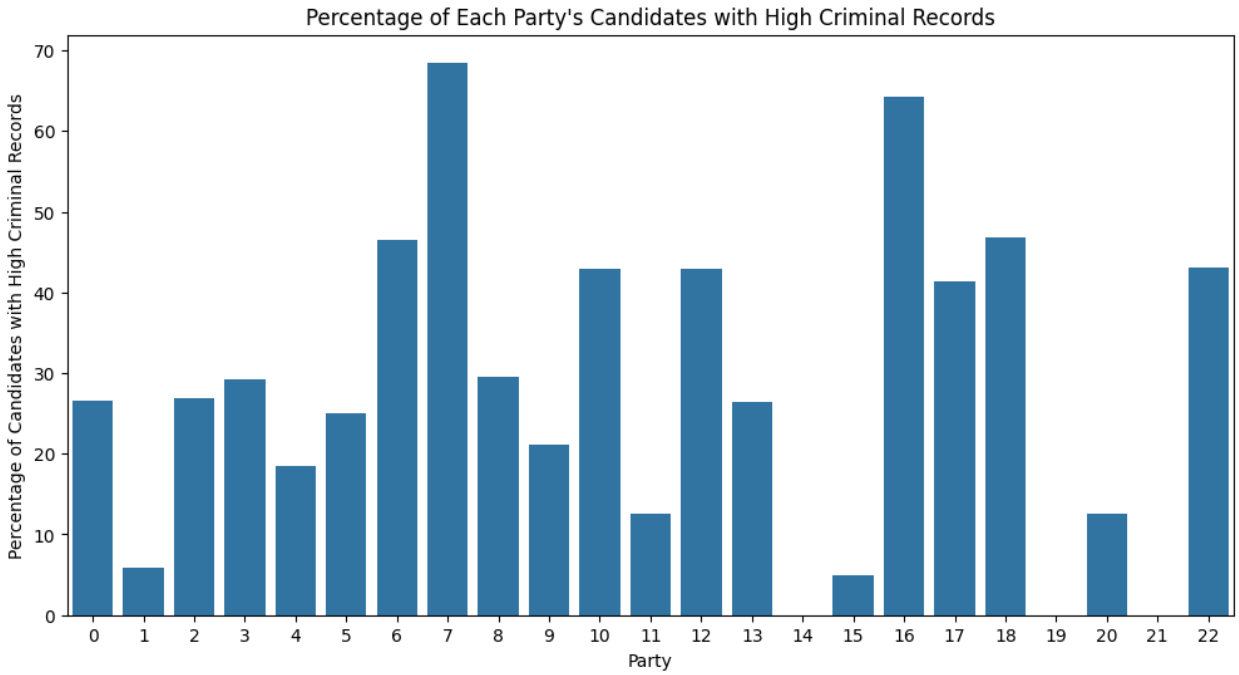
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

mean_criminal_cases = df['Criminal Case'].mean()

# Print the mean of criminal cases
print(f"Mean of Criminal Cases: {mean_criminal_cases}")
```

```
Mean of Criminal Cases: 1.7775619232637203
```

```
high_criminal_threshold = df['Criminal Case'].mean() # using mean of
criminal cases for high_criminal_threshold

# Filter data to include candidates with high criminal records
high_criminal_df = df[df['Criminal Case'] > high_criminal_threshold]

# Calculate total number of candidates with high criminal records
total_high_criminal = high_criminal_df.shape[0]

# Calculate the count of candidates with high criminal records for
each party
party_high_criminal_counts = high_criminal_df['Party'].value_counts()

# Calculate the proportion of candidates with high criminal records
for each party
party_high_criminal_proportions = party_high_criminal_counts /
total_high_criminal

# Calculate the total count of candidates for each party
party_total_counts = df['Party'].value_counts()

# Calculate the percentage of each party's candidates who have high
criminal records
party_high_criminal_percentages = (party_high_criminal_counts /
party_total_counts) * 100
```

```
# Plot the proportion of candidates with high criminal records for
each party
plt.figure(figsize=(12, 6))
sns.barplot(x=party_high_criminal_proportions.index,
y=party_high_criminal_proportions)
plt.title('Proportion of Candidates with High Criminal Records for
Each Party')
plt.xlabel('Party')
plt.ylabel('Proportion of Candidates with High Criminal Records')
plt.show()

# Plot the percentage of each party's candidates who have high
criminal records
plt.figure(figsize=(12, 6))
sns.barplot(x=party_high_criminal_percentages.index,
y=party_high_criminal_percentages)
plt.title('Percentage of Each Party\'s Candidates with High Criminal
Records')
plt.xlabel('Party')
plt.ylabel('Percentage of Candidates with High Criminal Records')
plt.show()
```

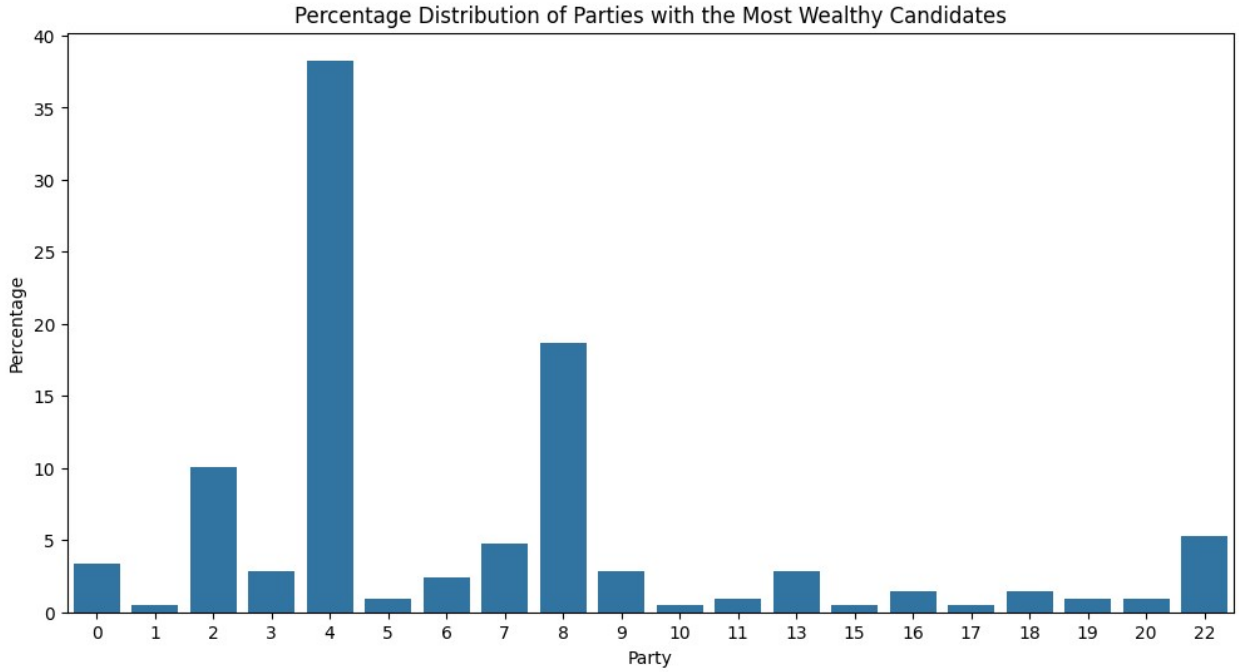




The percentage distribution of parties with the most wealthy candidates.

```
# Identify the column representing wealth (e.g., 'Total Assets')
wealth_column = 'Total Assets'

# Convert the 'Total Assets' column to numeric if it's not already
df[wealth_column] = df[wealth_column].replace('[^0-9.]', '',
regex=True).astype(float)

# Sort the data by wealth (descending order)
df_sorted_by_wealth = df.sort_values(by=wealth_column,
ascending=False)

# Define the threshold for the most wealthy candidates
# For example, using the top 10% of candidates based on wealth
top_10_percent_threshold =
df_sorted_by_wealth[wealth_column].quantile(0.9)

# Filter the data to include only the most wealthy candidates
most_wealthy_df =
df_sorted_by_wealth[df_sorted_by_wealth[wealth_column] >=
top_10_percent_threshold]

# Calculate the count of most wealthy candidates for each party
party_most_wealthy_counts = most_wealthy_df['Party'].value_counts()

# Calculate the total count of most wealthy candidates
total_most_wealthy = most_wealthy_df.shape[0]

# Calculate the percentage distribution of parties with the most
```

```
wealthy candidates
party_most_wealthy_percentages = (party_most_wealthy_counts /
total_most_wealthy) * 100

# Plot the percentage distribution of parties with the most wealthy
candidates
plt.figure(figsize=(12, 6))
sns.barplot(x=party_most_wealthy_percentages.index,
y=party_most_wealthy_percentages)
plt.title('Percentage Distribution of Parties with the Most Wealthy
Candidates')
plt.xlabel('Party')
plt.ylabel('Percentage')
plt.show()
```



I am splitting the training set to .2 to train the model and calculating f_1 score so that i can expect the accurracy of my model and model using is random forest to train the model

```
# Split the data into training and testing sets
X_train_interact, X_test_interact, y_train, y_test =
train_test_split(X_interact, y, test_size=0.2, random_state=48)

# Initialize the Random Forest Classifier with balanced class weights
rf_classifier_interact = RandomForestClassifier(n_estimators=500,
max_depth=10, class_weight='balanced', random_state=44)

# Train the Random Forest Classifier on data with interaction terms
rf_classifier_interact.fit(X_train_interact, y_train)
```

```
# Predict on the test set with interaction terms
y_pred_interact = rf_classifier_interact.predict(X_test_interact)

# Calculate F1 score on data with interaction terms
f1_interact = f1_score(y_test, y_pred_interact, average='weighted')
print("F1 Score with interaction terms:", f1_interact)
```

```
F1 Score with interaction terms: 0.23497446386597046
```

Now calculating for test data and write its prediction to csv file

```
# Load the test data
test_data = pd.read_csv('test.csv')

# Preprocess the test data
test_data['Total Assets'] = test_data['Total Assets'].apply(lambda x:
float(re.sub('[^0-9.]', '', str(x))))
test_data['Liabilities'] = test_data['Liabilities'].apply(lambda x:
float(re.sub('[^0-9.]', '', str(x))))

# Encode 'Party' column in test data
test_data['Party'] = label_encoder_party.transform(test_data['Party'])

# Replace states in test data with integers using the same dictionary
test_data['State_Label'] = test_data['state'].map(state_to_int)

# Scale the test data
test_data_scaled = scaler.transform(test_data[['Criminal Case', 'Total
Assets', 'Liabilities', 'Party', 'State_Label']])

# Create interaction terms for the test data
test_data_interact = poly.transform(test_data_scaled)

# Make predictions on the test data with interaction terms
predicted_education_levels_interact =
rf_classifier_interact.predict(test_data_interact)

# Create a new DataFrame for the predictions
predicted_df_interact = pd.DataFrame({'ID': test_data['ID'],
'Predicted_Education': predicted_education_levels_interact})
```

Result

- Final F_1 Score = 0.216000
- Public Leaderboard Rank - 177
- Private Leaderboard Rank - 173
- Github link of Source Code : Assignment-3

References

- For ploting MatplotLib {Used in Section 2}

- Model Used Random Forest { Used in Section 2}
- Pandas Learn from Pandas Used in Section 1
- Data Pre - processing youtube Used in Section 1
- Debugging - selt + ChatGpt ChatGpt Used in almost all section