Data Cleanup Strategy

1. Handling Missing Data:

- Empty strings are replaced with NaN values.
- Rows where 'VOTES' is "None" are dropped. No votes, no party, right?

```
# Replace empty strings with NaN
df.replace('', np.nan, inplace=True)

# Drop rows where 'VOTES' is "None"
df = df.loc[df['VOTES'] != "None"]
```

2. Cleaning Assembly Constituency (AC) Names:

- Leading numbers and spaces are stripped off. We don't need '01 Constituency', just 'Constituency'.
- Non-alphabetic characters (except spaces and periods) are removed. Keeping it clean and simple.

```
# Remove leading numbers and spaces
df['AC'] = df['AC'].str.replace(r'^\d+\s+', '', regex=True)

# Remove non-alphabetic characters (except spaces and periods)
df['AC'] = df['AC'].str.replace(r'[^a-zA-Z\s\.]', '', regex=True)
```

3. Cleaning Candidate Names:

- Like AC names, leading numbers and spaces are removed.
- Again, non-alphabetic characters (except spaces and periods) are stripped out. We want names, not noise.

```
# Remove leading numbers and spaces
df['CANDIDATE'] = df['CANDIDATE'].str.replace(r'^\d+\s+', '', regex=True)

# Remove non-alphabetic characters (except spaces and periods)
```

df['CANDIDATE'] = df['CANDIDATE'].str.replace(r'[^a-zA-Z\s\.]', '', regex=True)

4. Whitespace Cleanup:

- All string values get a good trim. No hanging spaces allowed.

 # Strip whitespace from all string columns

df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)

5. Duplicate Removal:

- First, a blanket removal of all duplicate rows. No double-counting here.

- Then, a more specific de-duplication based on key fields (ST_NAME, YEAR, AC, CANDIDATE, SEX, AGE, CATEGORY, PARTY, VOTES). We're keeping unique election entries only.

 # Remove all duplicate rows

df = df.drop_duplicates()

 # Remove duplicates based on specific columns

df = df.drop_duplicates(subset=['ST_NAME', 'YEAR', 'AC', 'CANDIDATE', 'SEX', 'AGE', 'CATEGORY', 'PARTY', 'VOTES'])

6. Data Type Consistency:

- While not explicitly converted, all data is treated as strings during the cleaning process. This ensures consistent handling across all fields.

 # Ensure all columns are treated as strings (if needed)

df = df.astype(str)

 # Or, for specific data type conversions:

df['YEAR'] = df['YEAR'].astype(int)

df['VOTES'] = df['VOTES'].astype(float)

7. Selective Saving:

- The full state data is saved to a CSV.

- Then, data is filtered for a specific Assembly Constituency (AC_NAME) and saved separately. Focused analysis, anyone?

```
# Save full state data
df.to_csv(f"{ST_NAME}.csv", index=False)

# Filter and save data for specific AC
condition = (df["AC"] == AC_NAME)
df_ac = df[condition]
df_ac.to_csv(f"{ST_NAME}_{AC_NAME}.csv", index=False)
```

8. Proactive Error Handling:

- The code checks if state data exists before proceeding with scraping and cleaning. No data, no problem - it'll let you know.

```
if state_data:
    election_data = scrape_data(start_url, state_data)
    save(ST_NAME, AC_NAME, election_data)
else:
    print(f"No data found for state: {ST_NAME}")
```