In [160…

```
import pandas as pd
from gurobipy import Model, GRB, quicksum

# Assuming the CSV file is on your desktop
file_path = "Alabama.csv"

# Read the data into a Pandas DataFrame
census_df = pd.read_csv(file_path)


print(census_df.columns)
```

```
Index(['Label (Grouping)', 'Alabama', 'Alaska', 'Arizona', 'Arkansas',
       'California', 'Colorado', 'Connecticut', 'Delaware',
       'District of Columbia', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Rhode Island', 'South Carolina', 'South Dakota',
       'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming', 'Puerto Rico'],
      dtype='object')
```

In [143…

```
import pandas as pd
from gurobipy import Model, GRB, quicksum

# Assuming the CSV file is on your desktop
file_path = "Alabama.csv"

# Read the data into a Pandas DataFrame
census_df = pd.read_csv(file_path)


print(census_df.columns)d

# Assuming the column name for total population is 'Alabama'
total_population_str = census_df.loc[census_df['Label (Grouping)'].str.strip() == 'Total:', 'Alabama'].values[0]

# Convert total population to an integer
total_population = int(total_population_str.replace(',', ''))
```

```
racial_composition = 'White alone'


white_alone_row = census_df[census_df['Label (Grouping)'].str.strip() == racial_composition]

if not white_alone_row.empty:
    # Use the corresponding row as the racial composition data
    racial_data = white_alone_row['Alabama'].values[0]
else:
    raise ValueError(f"Row related to '{racial_composition}' not found.")

# Create a new model
model = Model("congressional_redistricting")

# Decision variables
num_precincts = len(census_df)
districts = range(1, 8)  # Assuming 7 districts
precincts = range(1, num_precincts + 1)

x = {}  # Binary variable: precinct i is assigned to district j
for i in precincts:
    for j in districts:
        x[i, j] = model.addVar(vtype=GRB.BINARY, name=f"x_{i}_{j}")

# Objective function: Minimize population divergence among districts
model.setObjective(
    quicksum((x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', ''))) for i in precincts for j in districts),
    GRB.MINIMIZE
)

# Constraint 1: Each precinct must be assigned to exactly one district
for i in precincts:
    model.addConstr(sum(x[i, j] for j in districts) == 1, f"Assign_Precinct_{i}_to_One_District")

# Constraint 2: Ensure that the total population in each district is approximately equal
target_population_per_district = total_population / len(districts)
for j in districts:
    model.addConstr(
        quicksum(x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', '')) for i in precincts) >= 0.0  # Relaxing the lower
    )
    model.addConstr(
        quicksum(x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', '')) for i in precincts) <= total_population  # Relax
```

```
    )

# Solve the model
model.optimize()

# Check the optimization status
if model.status == GRB.OPTIMAL:
    print("Optimal solution found")
else:
    print(f"Optimization terminated with status {model.status}")

# Extract the solution and visualize the results
for i in precincts:
    for j in districts:
        if x[i, j].x > 0.5:  # Check if the variable is assigned
            print(f"Precinct {i} is assigned to District {j}")


district_populations = {}
for j in districts:
    population = sum(x[i, j].x * int(census_df.loc[i - 1, 'Alabama'].replace(',', '')) for i in precincts)
    district_populations[f"District {j}"] = population

print("District Populations:")
print(district_populations)


print(f"Total Population: {total_population}")
```

```
Index(['Label (Grouping)', 'Alabama', 'Alaska', 'Arizona', 'Arkansas',
       'California', 'Colorado', 'Connecticut', 'Delaware',
       'District of Columbia', 'Florida', 'Georgia', 'Hawaii', 'Idaho',
       'Illinois', 'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana',
       'Maine', 'Maryland', 'Massachusetts', 'Michigan', 'Minnesota',
       'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'Nevada',
       'New Hampshire', 'New Jersey', 'New Mexico', 'New York',
       'North Carolina', 'North Dakota', 'Ohio', 'Oklahoma', 'Oregon',
       'Pennsylvania', 'Rhode Island', 'South Carolina', 'South Dakota',
       'Tennessee', 'Texas', 'Utah', 'Vermont', 'Virginia', 'Washington',
       'West Virginia', 'Wisconsin', 'Wyoming', 'Puerto Rico'],
      dtype='object')
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (win64)

CPU model: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 85 rows, 497 columns and 1477 nonzeros
Model fingerprint: 0xc46308e7
Variable types: 0 continuous, 497 integer (497 binary)
Coefficient statistics:
  Matrix range     [1e+00, 5e+06]
  Objective range  [2e+00, 5e+06]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 5e+06]
Found heuristic solution: objective 1.532979e+07
Presolve removed 8 rows and 7 columns
Presolve time: 0.01s
Presolved: 77 rows, 490 columns, 980 nonzeros
Variable types: 0 continuous, 490 integer (490 binary)

Explored 0 nodes (0 simplex iterations) in 0.04 seconds (0.00 work units)
Thread count was 12 (of 12 available processors)

Solution count 1: 1.53298e+07

Optimal solution found (tolerance 1.00e-04)
Best objective 1.532979000000e+07, best bound 1.532979000000e+07, gap 0.0000%
Optimal solution found
Precinct 1 is assigned to District 1
Precinct 2 is assigned to District 3
Precinct 3 is assigned to District 7
Precinct 4 is assigned to District 5
Precinct 5 is assigned to District 4
```

```
Precinct 6 is assigned to District 6
Precinct 7 is assigned to District 5
Precinct 8 is assigned to District 7
Precinct 9 is assigned to District 4
Precinct 10 is assigned to District 6
Precinct 11 is assigned to District 4
Precinct 12 is assigned to District 6
Precinct 13 is assigned to District 5
Precinct 14 is assigned to District 5
Precinct 15 is assigned to District 2
Precinct 16 is assigned to District 2
Precinct 17 is assigned to District 4
Precinct 18 is assigned to District 3
Precinct 19 is assigned to District 7
Precinct 20 is assigned to District 6
Precinct 21 is assigned to District 3
Precinct 22 is assigned to District 6
Precinct 23 is assigned to District 7
Precinct 24 is assigned to District 5
Precinct 25 is assigned to District 4
Precinct 26 is assigned to District 6
Precinct 27 is assigned to District 5
Precinct 28 is assigned to District 2
Precinct 29 is assigned to District 4
Precinct 30 is assigned to District 2
Precinct 31 is assigned to District 5
Precinct 32 is assigned to District 3
Precinct 33 is assigned to District 7
Precinct 34 is assigned to District 3
Precinct 35 is assigned to District 7
Precinct 36 is assigned to District 6
Precinct 37 is assigned to District 7
Precinct 38 is assigned to District 6
Precinct 39 is assigned to District 5
Precinct 40 is assigned to District 4
Precinct 41 is assigned to District 7
Precinct 42 is assigned to District 3
Precinct 43 is assigned to District 4
Precinct 44 is assigned to District 2
Precinct 45 is assigned to District 3
Precinct 46 is assigned to District 6
Precinct 47 is assigned to District 7
Precinct 48 is assigned to District 4
Precinct 49 is assigned to District 5
```

```
Precinct 50 is assigned to District 3
Precinct 51 is assigned to District 2
Precinct 52 is assigned to District 7
Precinct 53 is assigned to District 3
Precinct 54 is assigned to District 2
Precinct 55 is assigned to District 5
Precinct 56 is assigned to District 3
Precinct 57 is assigned to District 7
Precinct 58 is assigned to District 4
Precinct 59 is assigned to District 6
Precinct 60 is assigned to District 7
Precinct 61 is assigned to District 3
Precinct 62 is assigned to District 5
Precinct 63 is assigned to District 7
Precinct 64 is assigned to District 5
Precinct 65 is assigned to District 5
Precinct 66 is assigned to District 3
Precinct 67 is assigned to District 1
Precinct 68 is assigned to District 4
Precinct 69 is assigned to District 4
Precinct 70 is assigned to District 4
Precinct 71 is assigned to District 7
District Populations:
{'District 1': 5024279.0, 'District 2': 89452.0, 'District 3': 4769303.0, 'District 4': 338969.0, 'District 5': 1325501.0, 'Dist
rict 6': 414107.0, 'District 7': 3368179.0}
Total Population: 5024279
```

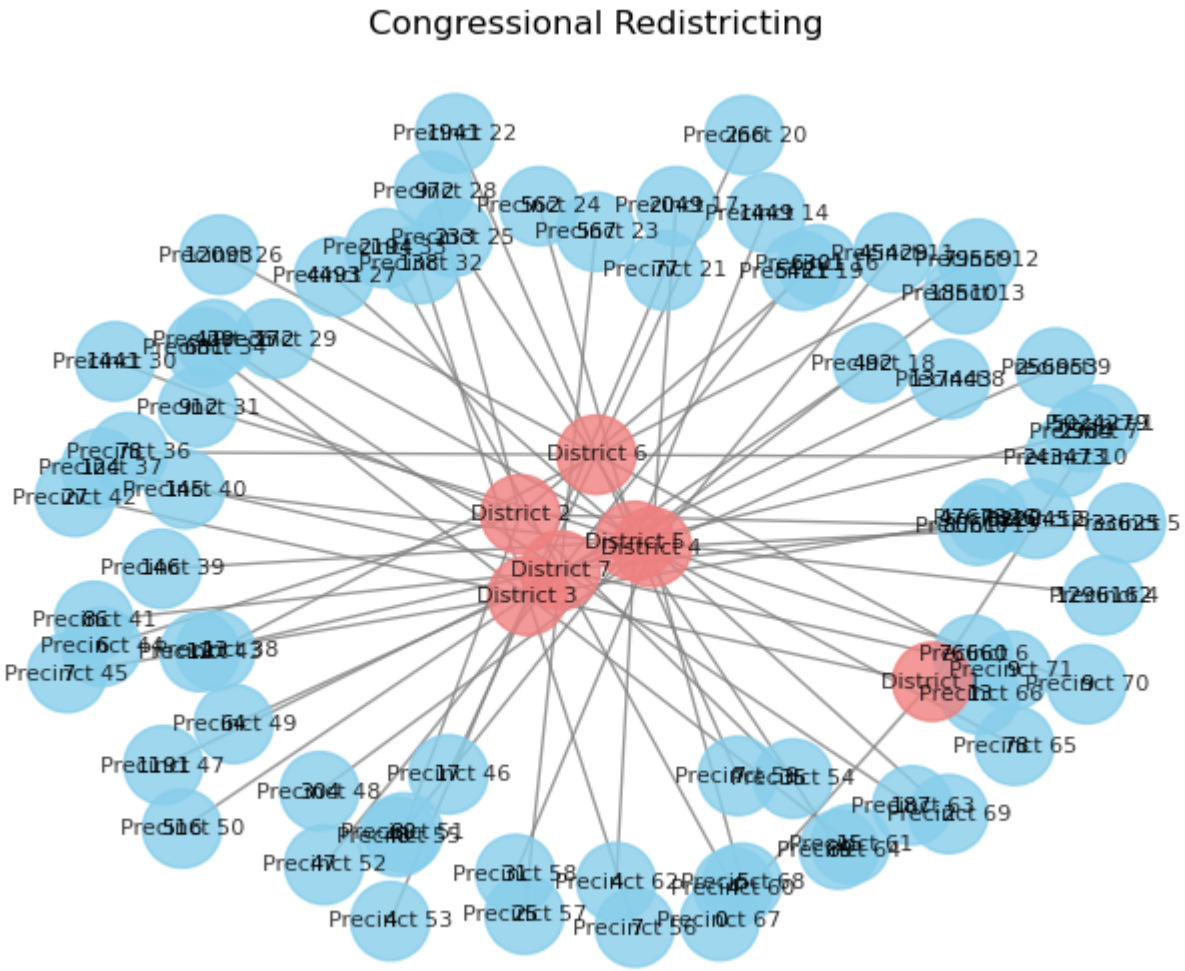
In [150…

```python
import pandas as pd
from gurobipy import Model, GRB, quicksum
import networkx as nx
import matplotlib.pyplot as plt

# Assuming the CSV file is on your desktop
file_path = "Alabama.csv"

# Read the data into a Pandas DataFrame
census_df = pd.read_csv(file_path)

# Assuming the column name for total population is 'Alabama'
total_population_str = census_df.loc[census_df['Label (Grouping)'].str.strip() == 'Total:', 'Alabama'].values[0]

# Convert total population to an integer
total_population = int(total_population_str.replace(',', ''))
```

```
# Assuming 'White alone' is a value in the 'Label (Grouping)' column
racial_composition = 'White alone'

# Find the row index where 'White alone' appears in the 'Label (Grouping)' column
white_alone_row = census_df[census_df['Label (Grouping)'].str.strip() == racial_composition]

if not white_alone_row.empty:
    # Use the corresponding row as the racial composition data
    racial_data = white_alone_row['Alabama'].values[0]
else:
    raise ValueError(f"Row related to '{racial_composition}' not found.")

# Create a new model
model = Model("congressional_redistricting")

# Decision variables
num_precincts = len(census_df)
districts = range(1, 8)  # Assuming 7 districts
precincts = range(1, num_precincts + 1)

x = {}  # Binary variable: precinct i is assigned to district j
for i in precincts:
    for j in districts:
        x[i, j] = model.addVar(vtype=GRB.BINARY, name=f"x_{i}_{j}")

# Objective function: Minimize population divergence among districts
model.setObjective(
    quicksum((x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', ''))) for i in precincts for j in districts),
    GRB.MINIMIZE
)

# Constraints: Add constraints based on federal and state criteria
# Constraint 1: Each precinct must be assigned to exactly one district
for i in precincts:
    model.addConstr(sum(x[i, j] for j in districts) == 1, f"Assign_Precinct_{i}_to_One_District")

# Constraint 2: Ensure that the total population in each district is approximately equal
target_population_per_district = total_population / len(districts)
for j in districts:
    model.addConstr(
        quicksum(x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', '')) for i in precincts) >= 0.0  # Relaxing the lower
    )
    model.addConstr(
        quicksum(x[i, j] * int(census_df.loc[i - 1, 'Alabama'].replace(',', '')) for i in precincts) <= total_population  # Relax
```

```
    )

# Solve the model
model.optimize()

# Check the optimization status
if model.status == GRB.OPTIMAL:
    print("Optimal solution found")
else:
    print(f"Optimization terminated with status {model.status}")

# Create a graph to visualize the districts and precincts
G = nx.Graph()

# Add nodes for precincts
for i in precincts:
    G.add_node(f"Precinct {i}", color='skyblue', population=int(census_df.loc[i - 1, 'Alabama'].replace(',', '')))

# Add nodes for districts
for j in districts:
    G.add_node(f"District {j}", color='lightcoral')

# Add edges for assigned districts
for i in precincts:
    for j in districts:
        if x[i, j].x > 0.5:  # Check if the variable is assigned
            G.add_edge(f"Precinct {i}", f"District {j}")

# Set node colors based on assignment
node_colors = [G.nodes[node]['color'] for node in G.nodes]

# Draw the graph with Kamada-Kawai layout
pos = nx.kamada_kawai_layout(G)
nx.draw(G, pos, with_labels=True, node_color=node_colors, font_color='black', font_size=8, node_size=800, cmap=plt.cm.Blues, alph

# Display population size as node labels
labels = {node: G.nodes[node].get('population', '') for node in G.nodes}
nx.draw_networkx_labels(G, pos, labels, font_size=8)

# Display the plot
plt.title("Congressional Redistricting")
plt.show()
```

```
Gurobi Optimizer version 10.0.2 build v10.0.2rc0 (win64)

CPU model: Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz, instruction set [SSE2|AVX|AVX2]
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 85 rows, 497 columns and 1477 nonzeros
Model fingerprint: 0xc46308e7
Variable types: 0 continuous, 497 integer (497 binary)
Coefficient statistics:
  Matrix range     [1e+00, 5e+06]
  Objective range  [2e+00, 5e+06]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 5e+06]
Found heuristic solution: objective 1.532979e+07
Presolve removed 8 rows and 7 columns
Presolve time: 0.00s
Presolved: 77 rows, 490 columns, 980 nonzeros
Variable types: 0 continuous, 490 integer (490 binary)

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 12 (of 12 available processors)

Solution count 1: 1.53298e+07

Optimal solution found (tolerance 1.00e-04)
Best objective 1.532979000000e+07, best bound 1.532979000000e+07, gap 0.0000%
Optimal solution found
C:\Users\nirmi\anaconda3\Lib\site-packages\networkx\drawing\nx_pylab.py:433: UserWarning: No data for colormapping provided via
'c'. Parameters 'cmap' will be ignored
  node_collection = ax.scatter(
```

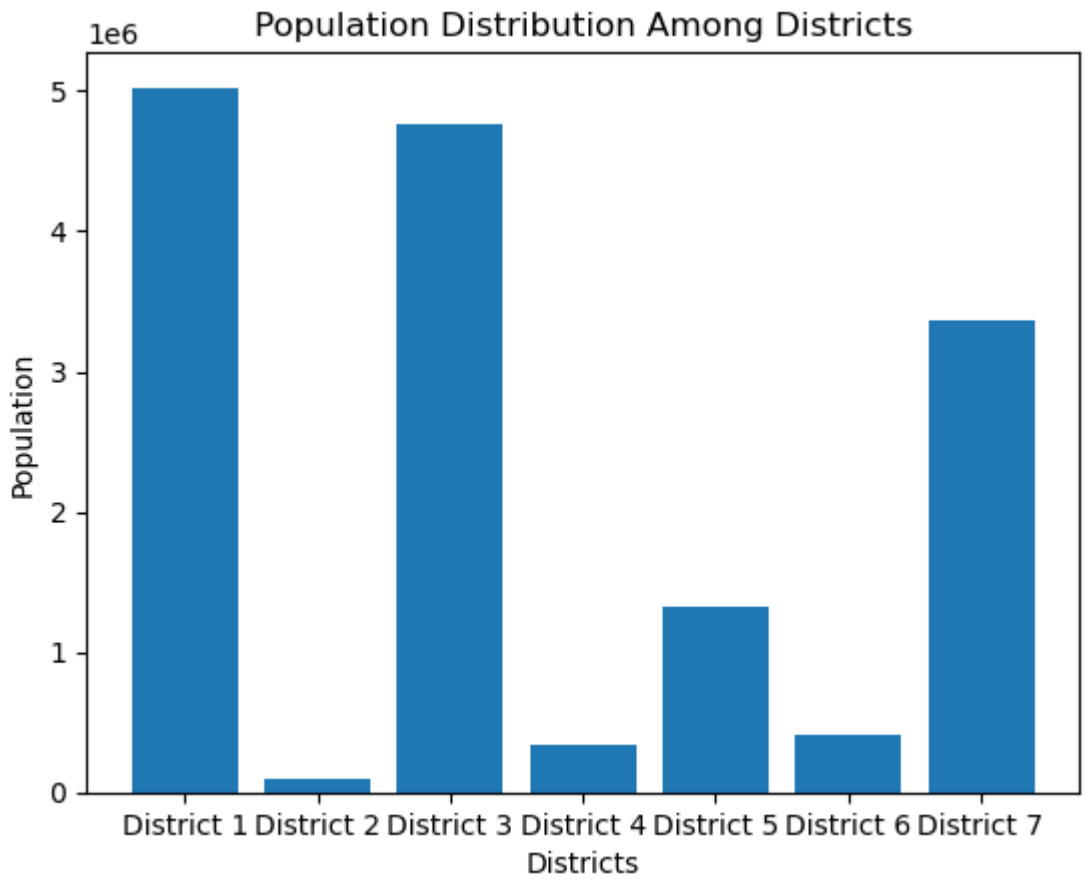


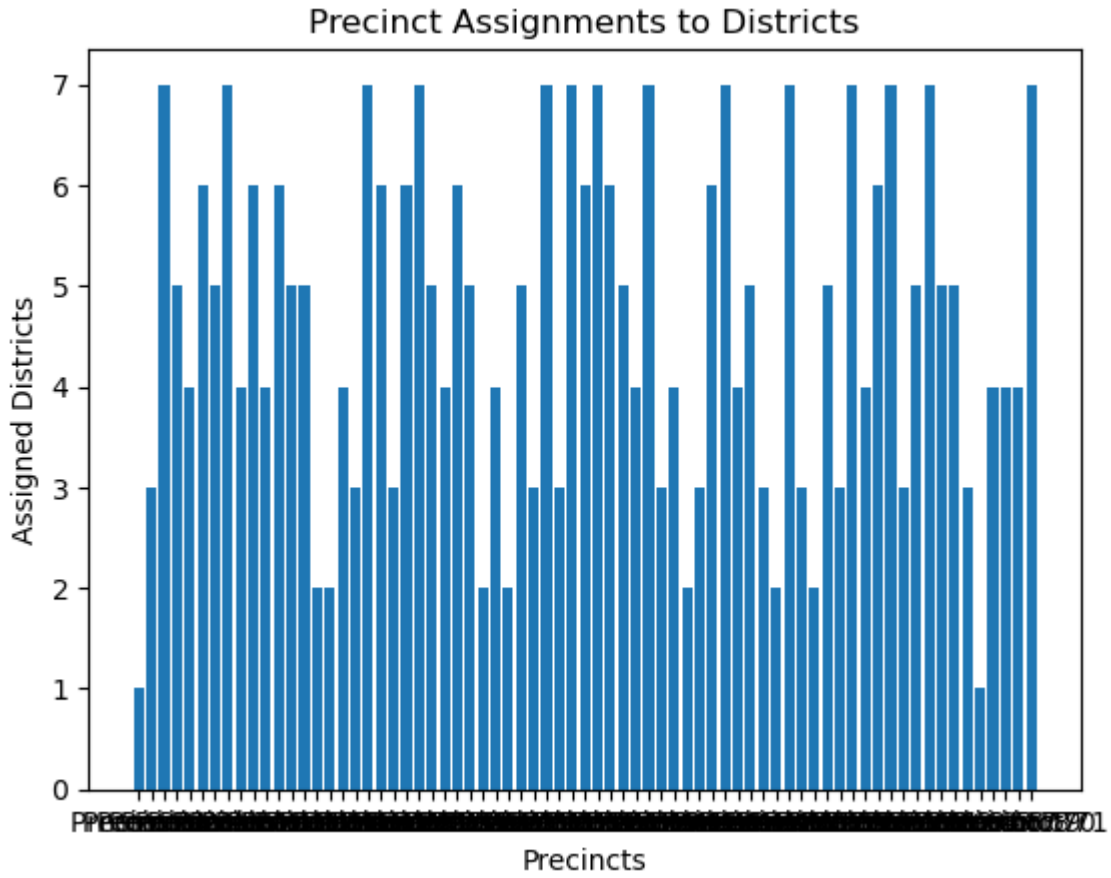
In [167…

```
import matplotlib.pyplot as plt

# Check the optimization status
if model.status == GRB.OPTIMAL:
    print("Optimal solution found")

    # Create a bar chart for district populations
    district_names = [f"District {j}" for j in districts]
    district_populations = [district_populations[d] for d in district_names]

    plt.bar(district_names, district_populations)
```

```
    plt.xlabel('Districts')
    plt.ylabel('Population')
    plt.title('Population Distribution Among Districts')
    plt.show()

    # Create a bar chart for precinct assignments
    precinct_assignments = {i: None for i in precincts}
    for i in precincts:
        for j in districts:
            if x[i, j].x > 0.5:
                precinct_assignments[i] = j

    precinct_names = [f"Precinct {i}" for i in precincts]
    precinct_districts = [precinct_assignments[i] for i in precincts]

    plt.bar(precinct_names, precinct_districts)
    plt.xlabel('Precincts')
    plt.ylabel('Assigned Districts')
    plt.title('Precinct Assignments to Districts')
    plt.show()

else:
    print(f"Optimization terminated with status {model.status}")
```

```
Optimal solution found
```


Population Distribution Among Districts
1e6
5
4
3
2
1
0
Population
District 1
District 2
District 3
District 4
District 5
District 6
District 7
Districts


Precinct Assignments to Districts
Assigned Districts
0
1
2
3
4
5
6
7
Precincts

In [ ]: