Nodes

Node Label	Properties	Count
`Business` : Node to represent a single business.	`business_id` : Business unique identifier. `business_name` : Business name. `business_type` : Business type (grocery_store, fast_food, farmers_market, bakery). `location` : Business lat/lon. `address` : Business address. `rating` : avg. customer rating. `price_level` : rating for avg cost of goods.	
`BlockGroup` : Node that represents a census block group.	`blockgroup_id` : Block group id. Equal to the ctblockgroup in SANDAG. `census_tract` : Census tract the block group belongs to. `block_group` : Block group of census tract. `object_id` : Unique identifier for the shape. `geometry` : WKB polygon shape	2057
`Zipcode` : Node that represents a zip code.	`zip` : The 5-digit number identifying a zip code. `geometry` : WKB polygon shape	
`City` : Node that represents a city (provided by "Administrative Topology").	`city_id` : City unique identifier. `city_name` : City name. `state_name` : State city belongs to. `county` : County city belongs to. `is_unincorporated` : Boolean indicating the incorporation status of the city(or place).	
`Neighborhood` : Node that represents a neighborhood (provided by "Administrative Topology").	`neighborhood_id` : Unique identifier for each neighborhood. `neighborhood_name` : Name of the neighborhood.	
`TotalPopulation` : Node containing the total population level of the block group.	`level` : 2024 Total population level.	
`PopulationGrowth` : Node to categorize each block group based on its growth rate.	`growth_rate` : 2024-2029 Population: Compound Annual Growth Rate category	
`AgeGroup` : Node containing age group population representation of block groups.	`group` : An age group range. `representation` : The age groups population representation.	
`EducationLevel` : Node containing education level population representation of block groups.	`level` : The education level. `representation` : The education level population representation.	
`WealthIndex` : Node categorizing the wealth index of block groups.	`category` : The total wealth index category.	
`CrimeIndex` : Node categorizing crime index of block groups.	`category` : The total crime index category.	
`FastFoodSpendingIndex` : Node categorizing spending levels at fast food places, including take-out and delivery.	`category` : The fast food spending index category.	

Edges

Relationship Type	Properties	Source Node	Target Node
`HAS_NEIGHBOR`	`neighbor_type` : "City" | "Neighborhood".	`City` `Neighborhood`	→ `City` → `Neighborhood`

`HAS_NEARBY`	`nearby_type` : "City" | "Neighborhood".	`City`
		`Neighborhood`
		`Neighborhood`
			→ `City`
			→ `Neighborhood`
			→ `City`

`LOCATED_IN`		`Business` `Business`	→ `BlockGroup` → `Zipcode`
`HAS_NEIGHBORHOOD`		`City`	→ `Neighborhood`

`IS_WITHIN`	`containment_type` : "Full" | "Partial"	`City`
		`Neighborhood`
		`BlockGroup`
			→ `Zipcode`
			→ `Zipcode`
			→ `Zipcode`

`HAS_ENRICHMENT`		`BlockGroup`	→ `TotalPopulation` → `PopulationGrowth` → `AgeGroup` → `EducationLevel` → `WealthIndex` → `CrimeIndex` → `FastFoodSpendingIndex`

Geo-enrichment Mapping

Enrichments

Name	Alias	Category	Use	Nodes	Properties	Values
TOTPOP_CY	2024 Total Population	Demographic	categorize each block group based on its population	TotalPopulation	level	"low": fewer than 1,000 residents "Medium: between 1,000 and 2,000 residents "High": more than 2,000 residents
POPGRWCYFY	2024-2029 Population: Compound Annual Growth Rate	Demographic	categorize each block group based on its growth rate	PopulationGrowth	growth_rate	"Negative": less than 0% (negative rate) "Low": 0% to 1% annually "Moderate": 1% to 2% annually "High": 2% to 3% annually "Very High": greater than 3% annually
male0 - male 85, fem85 OPEN	2024 Male Population Age 0-85 & 2024 Female Population Age 0-85	Demographic	compute average age for each block group, categorize each block group based on its average age	AgeGroup	group representation	group : "0-4", "5-14", "15-24", "25-44", "45-64", "65+" representation : "Very Low": 0% to 5%. "Low": 5% to 10%. "Moderate": 10% to 20%. "High": 20% to 30%. "Dominant": Over 30%.
wlthindxcy	2023 Wealth Index	Socioecono...	categorize each block group into richest, upper middle, mid-class, lower middle, and poverty Normalized it from 0 to 1	WealthIndex	category	"Low": Index score between 0.0 and 0.2 "Lower-Middle": Index score between 0.2 and 0.4 Middle: Index score between 0.4 and 0.6 Upper-Middle: Index score between 0.6 and 0.8 "High": Index score between 0.8 and 1.0
NOHS_CY, SOMEHS_CY, HSGRAD_CY, GED_CY, SMCOLL_CY, ASSCDEG_CY, BACHDEG_CY, GRADDEG_CY, educbasecy	2024 Population Age 25+: Less than 9th Grade, 9-12th Grade/No Diploma , High School Diploma, GED/Alternative Credential, Some College/No Degree, Associate's Degree, Bachelor's Degree, Graduate/Professional Degree	Socioecono...	compute the percentage of each block group in terms of Basic Education, Secondary Education, and Higher Education. basic_education_pct: Includes NOHS_CY, SOMEHS_CY, secondary_education_pct: Includes HSGRAD_CY, GED_CY, SMCOLL_CY, Higher_education_pct: Includes ASSCDEG_CY, BACHDEG_CY, GRADDEG_CY	EducationLevel	level representation	level : "Basic": Less than 9th Grade, 9-12th Grade/No Diploma. "Secondary": High School Diploma, GED/Alternative Credential, Some College/No Degree. "Higher": Associate's Degree, Bachelor's Degree, Graduate/Professional Degree. representation : "Very Low": 0% to 5%. "Low": 5% to 15%. "Moderate": 15% to 30%. "High": 30% to 50%. "Very High": Over 50%.
CRMCYTOTC OPEN	2024 Total Crime Index	Socioecono...	categorize each block group from safest to most unsafe based on the Total Crime Index	CrimeIndex	category	"Safest": Index < 80 "Safe": Index 80–119 "Moderate": Index 120–199 "Unsafe": Index 200–499 "Most Unsafe": Index ≥ 500
x1133a, x1138a, x1148a	spending ($) on lunch, dinner, breakfast at fast food/take-out/deliv	Spending	categorize spending levels at fast food places, including take-out and delivery combine x1133a, x1138a, x1148a first, then Normalized it from 0 to 1	FastFoodSpendingIndex	category	"Occasional": Index score between 0.0 and 0.2 "Light spender": Index score between 0.2 and 0.4 "Regular": Index score between 0.4 and 0.6 "Enthusiast": Index score between 0.6 and 0.8 "Super Fan": Index score between 0.8 and 1.0

Data Transformations

Created	@November 3, 2024 8:50 PM
↗ ✅ Tasks	🏗️ Restructure schema and pseudocode to reflect recent decisions regarding interplay between block groups, cities, neighborhoods, and zipcodes., 🗺️ Map geo-enriched features to new nodes/edges/properties, 📝 Write pseudocode for Administrative Topology., 📝 Update pseudocode for SANDAG if using postgres now, 📝 Write pseudocode for Google Place API, 📝 Write pseudocode for schema transformations

Documentation for all data transformations required for constructing the knowledge graph (excluding any text processing and NLP operations).

▼ Pseudocode

1. Initialize Neo4j Environment

```
CONNECT to Graph
```

2. Extract and Transform Data from Sources

   a. **Block Groups**

```
CONNECT to Postgres server

LOAD sandag_layer_census_block_groups table

CREATE_NODES:
    # BlockGroup
    FOR EACH Block Group DO
        CREATE_NODE(BlockGroup(blockgroup_id, census_tract, block_group, object_id, geometry))
    END FOR
```

   b. **Neighborhoods & Cities**

```
CONNECT to Postgres server

LOAD city_neighborhoods table
LOAD community_neighborhoods table

CREATE_NODES:
    # City
    FOR EACH UNIQUE city DO
        CREATE_NODE(City(city_id, city_name, state_name, county))
    END FOR

    # Neighborhood
    FOR EACH UNIQUE neighborhood DO
        CREATE_NODE(Neighborhood(neighborhood_id, neighborhood_name))
    END FOR

CREATE_RELATIONSHIPS:
    FOR EACH city in graph DO
        neighboring_cities = GET_NEIGHBORING_CITIES(city)
        nearby_cities = GET_NEARBY_CITIES(city)

        # City -> City (HAS_NEIGHBOR)
        FOR EACH neighbor_city in neighboring_cities DO
```

```
            CREATE_EDGE(HAS_NEIGHBOR, city, neighbor_city)
        END FOR

        # City -> City (HAS_NEARBY)
        FOR EACH nearby_city in nearby_cities DO
            CREATE_EDGE(HAS_NEARBY, city, nearby_city)
        END FOR
    END FOR

    FOR EACH neighborhood in graph DO
        neighboring_neighborhoods = GET_NEIGHBORING_NEIGHBORHOODS(neighborhood)
        nearby_neighborhood = GET_NEARBY_NEIGHBORHOODS(neighborhood)
        nearby_cities = GET_NEARBY_CITIES(neighborhood)

        # Neighborhood -> Neighborhood (HAS_NEIGHBORHOOD)
        FOR EACH neighbor_neighborhood in neighboring_neighborhoods DO
            CREATE_EDGE(HAS_NEIGHBORHOOD, neighborhood, neighbor_neighborhood)
        END FOR

        # Neighborhood -> Neighborhood (HAS_NEARBY)
        FOR EACH nearby_neighborhood in nearby_neighborhoods DO
            CREATE_EDGE(HAS_NEARBY, neighborhood, nearby_neighborhood)
        END FOR

        # Neighborhood -> City (HAS_NEARBY)
        FOR EACH nearby_city in nearby_cities DO
            CREATE_EDGE(HAS_NEARBY, neighborhood, nearby_city)
        END FOR
    END FOR
```

c. **Zipcodes**

```
CONNECT to Postgres server

OPEN csv_file("zipcodes.csv") as CSV_FILE:
  FOR EACH row in CSV_FILE:
    zipcode = row['ZIP']
    geometry = row['the_geom'].wkb
      # Zip Code
      CREATE_NODE(ZipCode(zipcode_number, geometry))

    CREATE_RELATIONSHIPS:
        FOR EACH zipcode in GRAPH DO:
            # Decode WKB to geometry objects using Shapely library
          zipcode_polygon = wkb.loads(zipcode['geometry'])

          FOR EACH block_group in GRAPH DO:
              blockgroup_polygon = wkb.loads(block_group['geometry'])
              IF do_polygons_intersect(zipcode_polygon, blockgroup_polygon):
                  # Blockgroup -> ZipCode (IS_WITHIN)
                  CREATE_EDGE(IS_WITHIN, block_group, zipcode)

            zipcode_number = zipcode['zipcode_number']
            FOR EACH neighborhood in GRAPH DO:
                IF zipcode_number IN neighborhood['zipcodes']
                    # Neighborhood -> ZipCode (IS_WITHIN)
                    CREATE_EDGE(IS_WITHIN, neighborhood, zipcode)
```

d. **Enrichments**

```
# Connect to Postgres server and load the necessary table
CONNECT to Postgres server

# Load table containing GeoProfile data (assuming it's called geoprofiles_table)
LOAD bgs_sd_imp

# Create nodes for each attribute category in GeoProfile
FOR EACH ROW IN bgs_sd_imp RESULT DO

    # Total Population Node
    CREATE_NODE(
        type = "TotalPopulation",
        total_population = ROW["TOTPOP_CY"]
    )

    # Population Growth Category Node
    CREATE_NODE(
        type = "PopulationGrowthCategory",
        population_growth_category = CATEGORIZE(ROW["POPGRWCYFY"], {
            "Negative growth": ROW["POPGRWCYFY"] < 0,
            "Low growth": 0 <= ROW["POPGRWCYFY"] < 1,
            "Moderate growth": 1 <= ROW["POPGRWCYFY"] < 2,
            "High growth": 2 <= ROW["POPGRWCYFY"] < 3,
            "Very high growth": ROW["POPGRWCYFY"] >= 3
        })
    )

    # Average Age Node
    CREATE_NODE(
        type = "AverageAge",
        average_age = CALCULATE_AVERAGE_AGE(ROW["male0"]...ROW["male85"], ROW["fem0"]...ROW["fem
85"])
    )

    # Wealth Index Node
    CREATE_NODE(
        type = "WealthIndex",
        wealth_index = CATEGORIZE(ROW["wlthindxcy"], {
            "poverty": 0.0 <= ROW["wlthindxcy"] < 0.2,
            "lower-middle": 0.2 <= ROW["wlthindxcy"] < 0.4,
            "mid-class": 0.4 <= ROW["wlthindxcy"] < 0.6,
            "upper-middle": 0.6 <= ROW["wlthindxcy"] < 0.8,
            "richest": 0.8 <= ROW["wlthindxcy"] <= 1.0
        })
    )

    # Crime Index Category Node
    CREATE_NODE(
        type = "CrimeIndexCategory",
        crime_index_category = CATEGORIZE(ROW["CRMCYTOTC"], {
            "Safest": ROW["CRMCYTOTC"] < 80,
            "Safe": 80 <= ROW["CRMCYTOTC"] < 120,
            "Moderate": 120 <= ROW["CRMCYTOTC"] < 200,
            "Unsafe": 200 <= ROW["CRMCYTOTC"] < 500,
            "Most unsafe": ROW["CRMCYTOTC"] >= 500
        })
    )
```

```
    # Spending Category Node
    CREATE_NODE(
        type = "SpendingCategory",
        spending_category = CATEGORIZE(NORMALIZE(ROW["x1133a"] + ROW["x1138a"] + ROW["x1148a"]),
{
            "Occasional": 0.0 <= value < 0.2,
            "Light spender": 0.2 <= value < 0.4,
            "Regular": 0.4 <= value < 0.6,
            "Enthusiast": 0.6 <= value < 0.8,
            "Super Fan": 0.8 <= value <= 1.0
        })
    )

    # Education Level Node
    CREATE_NODE(
        type = "EducationLevel",
        basic_education_pct = CALCULATE_PERCENTAGE(ROW["NOHS_CY"], ROW["SOMEHS_CY"]),
        secondary_education_pct = CALCULATE_PERCENTAGE(ROW["HSGRAD_CY"], ROW["GED_CY"], ROW["SMC
OLL_CY"]),
        higher_education_pct = CALCULATE_PERCENTAGE(ROW["ASSCDEG_CY"], ROW["BACHDEG_CY"], ROW["G
RADDEG_CY"])
    )

END FOR

# Establish relationships between BlockGroup and each separate attribute node
FOR EACH blockgroup IN sandag_layer_census_block_groups DO
    FIND BlockGroup node based on blockgroup_id

    # For each attribute node type, establish relationships if both nodes exist
    FOR EACH attribute_node IN [TotalPopulation, PopulationGrowthCategory, AverageAge, WealthInd
ex, CrimeIndexCategory, SpendingCategory, EducationLevel] DO
        FIND attribute_node based on matching criteria

        IF BlockGroup node exists AND attribute_node exists THEN
            CREATE RELATIONSHIP from BlockGroup to attribute_node
            SET relationship attribute value to blockgroup's specific value (if applicable)
        END IF
    END FOR
END FOR
```

e. **Businesses**

```
DEFINE business_types = ["grocery_store", "fast_food", "farmers_market", "bakery"]

FOR EACH BlockGroup in Graph:
    vertices = EXTRACT_VERTICES(BlockGroup)
    center, radius = CALCULATE_MINIMUM_ENCLOSING_CIRCLE(vertices)

    api_results = GOOGLE_PLACES_QUERY(
        location=center, radius=radius, types=business_types
    )

    FOR EACH business in api_results:
        IF is_point_in_polygon(connection, business["location"], BlockGroup.object_id, sandag_laye
                # Business
            CREATE_NODE(Business(business_id, business_name, business_type, location, rating, pric
            # Business -> BlockGroup (LOCATED_IN)
```

```
            CREATE_EDGE(LOCATED_IN, Business(business_id=business["business_id"]), BlockGroup)
        END IF
    END FOR
END FOR
```