LSCOG 2050 LRTP - Base Year SE Data Development

Pukar Bhandari

2025-07-05

Table of contents

# 1. Set up the environment

This section establishes the computational environment for processing socioeconomic data inputs for the Lower Savannah Council of Governments regional travel demand model using both R and Python platforms.

## 1.1 Install and load packages

The package installation process incorporates essential libraries for comprehensive geospatial data analysis.

The R environment includes {tidyverse} for data manipulation, {sf} for spatial data handling, {tidycensus} for Census Bureau data access, and {lehdr} for Longitudinal Employer-Household Dynamics data retrieval.

The Python environment focuses on core data science libraries including {pandas} for data manipulation, {geopandas} for spatial analysis, and {pygris} for Census data queries. These packages form the analytical backbone for processing demographic, employment, and geographic data required for travel demand modeling.

#### R

# # Install required packages  
# install.packages(c(  
# "tidyverse",  
# "vroom",  
# "sf",  
# "tidycensus",  
# "lehdr",  
# "arcgislayers",  
# "mapview",  
# "RColorBrewer",  
# "janitor"  
# ), dependencies = TRUE)  
  
# Load packages  
library(tidyverse) # Data manipulation and visualization  
library(vroom) # Read rectangular data  
library(sf) # Spatial analysis  
library(tidycensus) # Accessing US Census Data  
library(lehdr) # Access LODES data  
library(arcgislayers) # ArcGIS REST API access  
library(mapview) # Interactive mapping  
library(RColorBrewer) # Color palettes for maps  
library(janitor) # Data cleaning and preparation

#### Python

# Install required packages if not available  
# pip install numpy pandas geopandas shapely folium requests pygris  
  
# Load processing libraries & modules  
import os  
from pathlib import Path  
import zipfile  
import requests  
import urllib.parse  
import warnings  
warnings.filterwarnings('ignore')  
  
# Load data and visualization libraries & modules  
import numpy as np  
import pandas as pd  
import geopandas as gpd  
import folium  
from shapely.geometry import Point  
  
# Census data query libraries & modules  
from pygris import blocks, block\_groups  
from pygris.helpers import validate\_state, validate\_county  
from pygris.data import get\_census, get\_lodes

## 1.2 Set global options and parameters

Configuration settings optimize performance and establish spatial consistency. The tigris cache prevents redundant TIGER/Line shapefile downloads. The South Carolina State Plane coordinate system (EPSG:3361) serves as the standard projection for accurate GIS operations

#### R

# Set options  
options(tigris\_use\_cache = TRUE) # cache tiger/line shapefile for future use  
  
# set project CRS  
project\_crs <- "EPSG:3361"

#### Python

# Set project CRS  
project\_crs = "EPSG:3361"

## 1.3 Set census API key

API authentication enables access to detailed demographic and economic datasets from the Census Bureau. The key configuration supports both R and Python environments for automated data retrieval workflows.

💡 **Need a Census API key?** Get one for free at [census.gov/developers](https://www.census.gov/developers/)

#### R

# Set your API key into environment  
tidycensus::census\_api\_key("your\_api\_key\_here", install = TRUE)

#### Python

# Set your API key into environment  
os.environ['CENSUS\_API\_KEY'] = 'your\_api\_key\_here'

## 1.4 Project folder

The centralized directory structure organizes input data, processing files, and model outputs. The standardized root folder path ensures consistent file management across computing environments and team members.

#### R

# Set your main data folder  
root <- "M:/MA\_Project/SC\_LSCOG LRTP"

#### Python

# Set your main data folder  
root = "M:/MA\_Project/SC\_LSCOG LRTP"

# 2. Define study area

This section defines the geographic extent of the Lower Savannah Council of Governments region and loads the Traffic Analysis Zone (TAZ) geometry for spatial analysis.

## 2.1 Define state and counties

The study area encompasses six counties within South Carolina: Aiken, Allendale, Bamberg, Barnwell, Calhoun, and Orangeburg. These counties constitute the LSCOG planning region for travel demand modeling purposes.

#### R

# Define state abbreviation and county names  
state\_abb <- "SC"  
county\_names <- c(  
 "Aiken",  
 "Allendale",  
 "Bamberg",  
 "Barnwell",  
 "Calhoun",  
 "Orangeburg"  
)

#### Python

# Define state abbreviation and county names  
state\_abb = "SC"  
county\_names = [  
 "Aiken",  
 "Allendale",  
 "Bamberg",  
 "Barnwell",  
 "Calhoun",  
 "Orangeburg"  
]

FIPS code conversion translates state abbreviations and county names into standardized Federal Information Processing Standard codes. These codes enable consistent data retrieval across census datasets and ensure proper geographic matching with demographic and economic data sources.

#### R

# Converting state abbreviation code to FIPS code  
state\_fips <- tidycensus:::validate\_state(state = state\_abb)  
county\_fips <- vapply(  
 county\_names,  
 function(x) tidycensus:::validate\_county(state = state\_abb, county = x),  
 character(1)  
)  
  
# Converting County Names to FIPS code  
fips\_codes <- paste(state\_fips, county\_fips, sep = "")  
fips\_codes

[1] "45003" "45005" "45009" "45011" "45017" "45075"

#### Python

# Converting state abbreviation code to FIPS code  
state\_fips = validate\_state(state\_abb)

Using FIPS code '45' for input 'SC'

# Converting County Names to FIPS code  
county\_fips = [  
 validate\_county(state\_fips, county)  
 for county in county\_names  
]

Using FIPS code '003' for input 'Aiken'  
Using FIPS code '005' for input 'Allendale'  
Using FIPS code '009' for input 'Bamberg'  
Using FIPS code '011' for input 'Barnwell'  
Using FIPS code '017' for input 'Calhoun'  
Using FIPS code '075' for input 'Orangeburg'

# Converting County Names to FIPS code  
fips\_codes = [f"{state\_fips}{county}" for county in county\_fips]  
fips\_codes

['45003', '45005', '45009', '45011', '45017', '45075']

## 2.2 Load TAZ geometry

The TAZ shapefile provides the fundamental spatial framework for travel demand modeling. The geometry is loaded from the TDM exports geodatabase and filtered to include only zones within the six-county study area using FIPS code matching.

Coordinate transformation converts the TAZ geometry to the project’s standard coordinate reference system (EPSG:3361) for accurate spatial calculations. The attribute selection retains essential fields including TAZ identifiers, area measurements, area type classifications, and county assignments.

#### R

# Load TAZ Shapefile  
lscog\_taz <- sf::read\_sf(  
 file.path(root, "GIS/data\_temp/TDM Exports/TDM\_Exports.gdb"),  
 query = paste0(  
 "SELECT \* FROM \"SE\_2019\_AD\_10\_30\_2023\" WHERE countyID IN (",  
 paste0("'", fips\_codes, "'", collapse = ", "),  
 ")"  
 )  
) |>  
 sf::st\_transform(project\_crs) |>  
 dplyr::select(  
 ID,  
 Area,  
 Acres,  
 TAZ\_ID = TAZ\_IDs,  
 AREA\_TYPE,  
 COUNTY,  
 COUNTYID = countyID  
 )  
  
lscog\_taz

Simple feature collection with 585 features and 7 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691490 ymin: 331894 xmax: 2237471 ymax: 744679.9  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 585 × 8  
 ID Area Acres TAZ\_ID AREA\_TYPE COUNTY COUNTYID  
 <int> <dbl> <dbl> <int> <chr> <chr> <int>  
 1 9050130 13.3 8518. 9050130 RURAL Bamberg SC 45009  
 2 9050132 39.2 25084. 9050132 RURAL Bamberg SC 45009  
 3 75050131 9.03 5778. 75050131 RURAL Orangeburg S 45075  
 4 75050045 15.5 9934. 75050045 RURAL Orangeburg S 45075  
 5 75050182 17.5 11216. 75050182 SUBURBAN Orangeburg S 45075  
 6 75050183 12.8 8191. 75050183 RURAL Orangeburg S 45075  
 7 75050172 8.10 5181. 75050172 SUBURBAN Orangeburg S 45075  
 8 75050204 13.4 8568. 75050204 SUBURBAN Orangeburg S 45075  
 9 75050200 6.00 3843. 75050200 SUBURBAN Orangeburg S 45075  
10 75050201 5.06 3239. 75050201 SUBURBAN Orangeburg S 45075  
# ℹ 575 more rows  
# ℹ 1 more variable: SHAPE <MULTIPOLYGON [foot]>

#### Python

# Load TAZ Shapefile  
lscog\_taz = gpd.read\_file(  
 Path(root) / "GIS/data\_temp/TDM Exports/TDM\_Exports.gdb",  
 layer="SE\_2019\_AD\_10\_30\_2023",  
 where=f"countyID IN ({', '.join([f"'{fips}'" for fips in fips\_codes])})"  
)  
  
lscog\_taz = lscog\_taz.to\_crs(project\_crs)  
  
lscog\_taz = lscog\_taz.rename(  
 columns={  
 'TAZ\_IDs': 'TAZ\_ID',  
 'countyID': 'COUNTYID'  
})[['ID', 'Area', 'Acres', 'TAZ\_ID', 'AREA\_TYPE', 'COUNTY', 'COUNTYID', 'geometry']]  
  
lscog\_taz

ID ... geometry  
0 9050130 ... MULTIPOLYGON (((2046194.08 478862.054, 2046134...  
1 9050132 ... MULTIPOLYGON (((2012593.472 500179.47, 2013190...  
2 75050131 ... MULTIPOLYGON (((2056266.071 515975.986, 205617...  
3 75050045 ... MULTIPOLYGON (((2061826.693 488917.873, 206173...  
4 75050182 ... MULTIPOLYGON (((2154221.857 534929.221, 215441...  
.. ... ... ...  
580 5050049 ... MULTIPOLYGON (((1924248.136 433664.04, 1924004...  
581 5050055 ... MULTIPOLYGON (((1905461.23 427627.626, 1905456...  
582 5050066 ... MULTIPOLYGON (((1880812.362 414251.546, 188090...  
583 5050054 ... MULTIPOLYGON (((1871871.454 413403.458, 187167...  
584 5050065 ... MULTIPOLYGON (((1849074.015 398566.33, 1849218...  
  
[585 rows x 8 columns]

The interactive map visualization displays the TAZ structure colored by county, providing spatial context for the analysis area and enabling quality assurance of the geometric data loading process.

#### R

# Create interactive map  
mapview::mapview(lscog\_taz, zcol = "COUNTY", lwd = 1.6, map.types = "CartoDB.Positron", col.regions = RColorBrewer::brewer.pal(6, "Dark2"))

#### Python

# Create interactive map  
lscog\_taz.explore(column="COUNTY", categorical=True, legend=True, tiles="CartoDB positron", zoom\_start=8)

# 3. Fetch raw data

This section retrieves demographic, economic, and employment data from multiple Census Bureau sources at the appropriate geographic scales for travel demand modeling.

## 3.1 2020 Decennial census

The 2020 Decennial Census provides population and housing data at the census block level, offering the finest spatial resolution for demographic analysis. Population variables include total population, group quarters population, and household population derived by subtraction. Housing variables encompass total dwelling units and household counts by size categories.

Household size distributions are consolidated into four categories: 1-person, 2-person, 3-person, and 4-or-more-person households. The 4-or-more category aggregates larger household sizes to simplify model implementation while maintaining essential demographic stratification for trip generation analysis.

For more information on the Decennial Census data, refer to the [Decennial Census Technical Documentation](https://www.census.gov/programs-surveys/decennial-census/technical-documentation.html).

#### R

# Define variables to download  
dec\_variables <- c(  
 TOTPOP = "P1\_001N", # Total Population  
 GQPOP = "P18\_001N", # Population living in Group Quarters  
 DU = "H1\_001N", # Dwelling Units  
 HH\_1 = "H9\_002N", # 1-person household  
 HH\_2 = "H9\_003N", # 2-person household  
 HH\_3 = "H9\_004N", # 3-person household  
 # HH\_4 = "H9\_005N", # 4-person household  
 # HH\_5 = "H9\_006N", # 5-person household  
 # HH\_6 = "H9\_007N", # 6-person household  
 # HH\_7 = "H9\_008N", # 7-or-more-person household  
 HH = "H9\_001N" # Total Number of Households  
 )  
  
# Load Population and Household Data  
lscog\_dec <- tidycensus::get\_decennial(  
 year = 2020,  
 sumfile = "dhc",  
 geography = "block",  
 state = state\_fips,  
 county = county\_fips,  
 output = "wide",  
 cb = FALSE,  
 geometry = TRUE,  
 keep\_geo\_vars = TRUE,  
 # key = Sys.getenv('CENSUS\_API\_KEY'),  
 variables = dec\_variables  
) |>  
 sf::st\_transform(project\_crs) |>  
 dplyr::mutate(  
 HHPOP = TOTPOP - GQPOP,  
 HH\_4 = HH - (HH\_1 + HH\_2 + HH\_3)  
 ) |>  
 dplyr::select(GEOID, TOTPOP, GQPOP, HHPOP, HH, HH\_1, HH\_2, HH\_3, HH\_4, DU)  
  
lscog\_dec

Simple feature collection with 13961 features and 10 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 13,961 × 11  
 GEOID TOTPOP GQPOP HHPOP HH HH\_1 HH\_2 HH\_3 HH\_4 DU  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 450179504004051 0 0 0 0 0 0 0 0 0  
 2 450179504003011 0 0 0 0 0 0 0 0 0  
 3 450179502011045 54 4 50 16 3 1 3 9 18  
 4 450179504001020 0 0 0 0 0 0 0 0 0  
 5 450750105003029 13 0 13 4 0 3 0 1 4  
 6 450750117021009 10 0 10 3 3 0 0 0 8  
 7 450750117011051 0 0 0 0 0 0 0 0 0  
 8 450750118021087 6 0 6 4 0 1 2 1 4  
 9 450750120003020 0 0 0 0 0 0 0 0 0  
10 450179501003046 18 0 18 0 0 0 0 0 1  
# ℹ 13,951 more rows  
# ℹ 1 more variable: geometry <MULTIPOLYGON [foot]>

#### Python

# Define variables to download  
dec\_variables = {  
 'P1\_001N': 'TOTPOP', # Total Population  
 'P18\_001N': 'GQPOP', # Population living in Group Quarters  
 'H1\_001N': 'DU', # Dwelling Units  
 'H9\_002N': 'HH\_1', # 1-person household  
 'H9\_003N': 'HH\_2', # 2-person household  
 'H9\_004N': 'HH\_3', # 3-person household  
 # 'H9\_005N': 'HH\_4', # 4-person household  
 # 'H9\_006N': 'HH\_5', # 5-person household  
 # 'H9\_007N': 'HH\_6', # 6-person household  
 # 'H9\_008N': 'HH\_7', # 7-or-more-person household  
 'H9\_001N': 'HH' # Total Number of Households  
}  
  
# get census block geometries  
lscog\_cb = blocks(  
 state=state\_fips,  
 county=county\_fips,  
 year=2020,  
 cache=True  
)  
  
# Download decennial census data at block level  
lscog\_dec = get\_census(  
 dataset="dec/dhc",  
 year=2020,  
 variables=list(dec\_variables.keys()),  
 params={  
 "for": f"block:\*",  
 # "key": f"{os.getenv('CENSUS\_API\_KEY')}",  
 "in": f"state:{state\_fips} county:{','.join(county\_fips)}"  
 },  
 return\_geoid=True,  
 guess\_dtypes=True,  
)  
  
# join data to geometry  
lscog\_dec = lscog\_cb[['GEOID20', 'geometry']].merge(lscog\_dec, left\_on = "GEOID20", right\_on = "GEOID")  
  
# Rename columns  
lscog\_dec = lscog\_dec.rename(columns=dec\_variables)  
  
# Transform CRS  
lscog\_dec = lscog\_dec.to\_crs(project\_crs)  
  
# Calculate derived variables  
lscog\_dec['HHPOP'] = lscog\_dec['TOTPOP'] - lscog\_dec['GQPOP']  
lscog\_dec['HH\_4'] = lscog\_dec['HH'] - (  
 lscog\_dec['HH\_1'] + lscog\_dec['HH\_2'] + lscog\_dec['HH\_3']  
)  
  
# Select final columns  
lscog\_dec = lscog\_dec[['GEOID', 'TOTPOP', 'GQPOP', 'HHPOP',  
 'HH', 'HH\_1', 'HH\_2', 'HH\_3', 'HH\_4', 'DU', 'geometry']]  
  
lscog\_dec

GEOID ... geometry  
0 450179504004051 ... POLYGON ((2084300.974 622308.526, 2084460.74 6...  
1 450179504003011 ... POLYGON ((2112518.54 626782.208, 2112608.778 6...  
2 450179502011045 ... POLYGON ((2067775.167 662339.483, 2068091.491 ...  
3 450179504001020 ... POLYGON ((2087947.053 684345.612, 2087992.284 ...  
4 450750105003029 ... POLYGON ((2089072.743 552687.443, 2089248.831 ...  
... ... ... ...  
13956 450059705003050 ... POLYGON ((1926607.333 409005.466, 1926609.861 ...  
13957 450030203042000 ... POLYGON ((1778751.197 641782.585, 1778797.329 ...  
13958 450030218002115 ... POLYGON ((1919787.314 648288.439, 1919913.531 ...  
13959 450030206012008 ... POLYGON ((1723756.559 630234.507, 1723784.384 ...  
13960 450059705002005 ... POLYGON ((1926817.622 432583.896, 1930423.825 ...  
  
[13961 rows x 11 columns]

## 3.2 2020 ACS estimates

The American Community Survey 5-year estimates provide household income data at the block group level. Income categories are aggregated into three broad ranges: under $15,000, $15,000-$49,999, and $50,000 and above. This stratification aligns with travel behavior research indicating distinct mobility patterns across income levels.

The block group geography represents the finest spatial resolution available for ACS income data, providing sufficient detail for socioeconomic modeling while maintaining statistical reliability through the 5-year aggregation period.

For more information on the ACS data, refer to the [ACS Technical Documentation](https://www.census.gov/programs-surveys/acs/technical-documentation.html).

#### R

# Define variables to download  
acs\_variables <- c(  
 INC\_CAT\_02 = "B19001\_002", # Less than $10,000  
 INC\_CAT\_03 = "B19001\_003", # $10,000 to $14,999  
 INC\_CAT\_04 = "B19001\_004", # $15,000 to $19,999  
 INC\_CAT\_05 = "B19001\_005", # $20,000 to $24,999  
 INC\_CAT\_06 = "B19001\_006", # $25,000 to $29,999  
 INC\_CAT\_07 = "B19001\_007", # $30,000 to $34,999  
 INC\_CAT\_08 = "B19001\_008", # $35,000 to $39,999  
 INC\_CAT\_09 = "B19001\_009", # $40,000 to $44,999  
 INC\_CAT\_10 = "B19001\_010", # $45,000 to $49,999  
 # INC\_CAT\_11 = "B19001\_011", # $50,000 to $59,999  
 # INC\_CAT\_12 = "B19001\_012", # $60,000 to $74,999  
 # INC\_CAT\_13 = "B19001\_013", # $75,000 to $99,999  
 # INC\_CAT\_14 = "B19001\_014", # $100,000 to $124,999  
 # INC\_CAT\_15 = "B19001\_015", # $125,000 to $149,999  
 # INC\_CAT\_16 = "B19001\_016", # $150,000 to $199,999  
 # INC\_CAT\_17 = "B19001\_017", # $200,000 or more  
 INC\_CAT\_01 = "B19001\_001" # Total  
 )  
  
# Load Household Income Data  
lscog\_acs <- tidycensus::get\_acs(  
 year = 2020,  
 survey = "acs5",  
 geography = "block group",  
 state = state\_fips,  
 county = county\_fips,  
 output = "wide",  
 cb = FALSE,  
 geometry = TRUE,  
 # key = Sys.getenv('CENSUS\_API\_KEY'),  
 variables = acs\_variables  
) |>  
 sf::st\_transform(project\_crs) |>  
 dplyr::mutate(  
 INC\_14999 = INC\_CAT\_02E + INC\_CAT\_03E,  
 INC\_49999 = INC\_CAT\_04E +  
 INC\_CAT\_05E +  
 INC\_CAT\_06E +  
 INC\_CAT\_07E +  
 INC\_CAT\_08E +  
 INC\_CAT\_09E +  
 INC\_CAT\_10E,  
 INC\_50000 = INC\_CAT\_01E - (INC\_14999 + INC\_49999)  
 ) |>  
 dplyr::select(GEOID, INC\_TOTAL = INC\_CAT\_01E, INC\_14999, INC\_49999, INC\_50000)  
  
lscog\_acs

Simple feature collection with 262 features and 5 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
First 10 features:  
 GEOID INC\_TOTAL INC\_14999 INC\_49999 INC\_50000  
1 450030207011 467 82 171 214  
2 450030212052 949 0 256 693  
3 450030212051 857 21 250 586  
4 450030213001 612 103 129 380  
5 450030216031 1191 176 303 712  
6 450030215003 773 95 240 438  
7 450030205004 1081 56 302 723  
8 450030205003 937 37 158 742  
9 450030212041 758 37 344 377  
10 450030215004 973 77 247 649  
 geometry  
1 MULTIPOLYGON (((1701402 614...  
2 MULTIPOLYGON (((1776367 591...  
3 MULTIPOLYGON (((1768340 598...  
4 MULTIPOLYGON (((1768476 630...  
5 MULTIPOLYGON (((1785900 618...  
6 MULTIPOLYGON (((1782074 620...  
7 MULTIPOLYGON (((1707972 630...  
8 MULTIPOLYGON (((1708123 634...  
9 MULTIPOLYGON (((1775528 610...  
10 MULTIPOLYGON (((1779272 619...

#### Python

# Define variables to download  
acs\_variables = {  
 'B19001\_002E': 'INC\_CAT\_02', # Less than $10,000  
 'B19001\_003E': 'INC\_CAT\_03', # $10,000 to $14,999  
 'B19001\_004E': 'INC\_CAT\_04', # $15,000 to $19,999  
 'B19001\_005E': 'INC\_CAT\_05', # $20,000 to $24,999  
 'B19001\_006E': 'INC\_CAT\_06', # $25,000 to $29,999  
 'B19001\_007E': 'INC\_CAT\_07', # $30,000 to $34,999  
 'B19001\_008E': 'INC\_CAT\_08', # $35,000 to $39,999  
 'B19001\_009E': 'INC\_CAT\_09', # $40,000 to $44,999  
 'B19001\_010E': 'INC\_CAT\_10', # $45,000 to $49,999  
 # 'B19001\_011E': 'INC\_CAT\_11', # $50,000 to $59,999  
 # 'B19001\_012E': 'INC\_CAT\_12', # $60,000 to $74,999  
 # 'B19001\_013E': 'INC\_CAT\_13', # $75,000 to $99,999  
 # 'B19001\_014E': 'INC\_CAT\_14', # $100,000 to $124,999  
 # 'B19001\_015E': 'INC\_CAT\_15', # $125,000 to $149,999  
 # 'B19001\_016E': 'INC\_CAT\_16', # $150,000 to $199,999  
 # 'B19001\_017E': 'INC\_CAT\_17', # $200,000 or more  
 'B19001\_001E': 'INC\_CAT\_01' # Total  
}  
  
# get blockgroup geometries  
lscog\_bg = block\_groups(  
 state=state\_fips,  
 county=county\_fips,  
 year=2020,  
 cache=True  
)  
  
# Download household income data at block group level  
lscog\_acs = get\_census(  
 dataset="acs/acs5",  
 year=2020,  
 variables=list(acs\_variables.keys()),  
 params={  
 "for": f"block group:\*",  
 # "key": f"{os.getenv('CENSUS\_API\_KEY')}",  
 "in": f"state:{state\_fips} county:{','.join(county\_fips)}"  
 },  
 return\_geoid=True,  
 guess\_dtypes=True  
)  
  
# join data to geometry  
lscog\_acs = lscog\_bg[['GEOID', 'geometry']].merge(lscog\_acs, on = "GEOID")  
  
# Rename columns  
lscog\_acs = lscog\_acs.rename(columns=acs\_variables)  
  
# Transform CRS  
lscog\_acs = lscog\_acs.to\_crs(project\_crs)  
  
# Calculate derived variables  
lscog\_acs['INC\_14999'] = lscog\_acs['INC\_CAT\_02'] + lscog\_acs['INC\_CAT\_03']  
lscog\_acs['INC\_49999'] = (  
 lscog\_acs['INC\_CAT\_04'] +  
 lscog\_acs['INC\_CAT\_05'] +  
 lscog\_acs['INC\_CAT\_06'] +  
 lscog\_acs['INC\_CAT\_07'] +  
 lscog\_acs['INC\_CAT\_08'] +  
 lscog\_acs['INC\_CAT\_09'] +  
 lscog\_acs['INC\_CAT\_10']  
)  
lscog\_acs['INC\_50000'] = lscog\_acs['INC\_CAT\_01'] - (  
 lscog\_acs['INC\_14999'] + lscog\_acs['INC\_49999']  
)  
  
# Select final columns  
lscog\_acs = lscog\_acs.rename(columns={'INC\_CAT\_01': 'INC\_TOTAL'})  
lscog\_acs = lscog\_acs[['GEOID', 'INC\_TOTAL', 'INC\_14999', 'INC\_49999', 'INC\_50000', 'geometry'  
]]  
  
lscog\_acs

GEOID ... geometry  
0 450030207011 ... POLYGON ((1701402.37 614608.958, 1701441.07 61...  
1 450030212052 ... POLYGON ((1776366.684 591083.944, 1776445.37 5...  
2 450030212051 ... POLYGON ((1768340.248 598546.468, 1768482.141 ...  
3 450030213001 ... POLYGON ((1768475.931 630588.733, 1768610.076 ...  
4 450030216031 ... POLYGON ((1785900.107 618251.028, 1786047.488 ...  
.. ... ... ...  
257 450750119004 ... POLYGON ((1974249.527 620700.361, 1975105.197 ...  
258 450750103011 ... POLYGON ((2142864.631 581695.327, 2142874.033 ...  
259 450750109011 ... POLYGON ((2033500.16 608479.774, 2033509.304 6...  
260 450750109022 ... POLYGON ((2013478.807 622487.426, 2013488.515 ...  
261 450750109023 ... POLYGON ((2022268.105 617836.647, 2022656.655 ...  
  
[262 rows x 6 columns]

## 3.3 2019 LEHD data

The Longitudinal Employer-Household Dynamics Workplace Area Characteristics data provides employment counts by industry sector at the census block level. Employment categories follow the North American Industry Classification System and are aggregated into transportation-relevant sectors including retail, services, manufacturing, and public administration.

The 2019 reference year represents pre-pandemic employment patterns, providing a stable baseline for long-term transportation planning. Employment data at the block level enables precise spatial allocation of work destinations within the travel demand model framework.

For more information on the LEHD data, refer to the [LODES Technical Documentation](https://lehd.ces.census.gov/data/lodes/LODES8/LODESTechDoc8.3.pdf).

#### R

# Download LEHD WAC data at block level  
lscog\_emp <- lehdr::grab\_lodes(  
 version = "LODES8",  
 state = tolower(state\_abb),  
 lodes\_type = "wac",  
 segment = "S000",  
 job\_type = "JT00",  
 year = 2019,  
 state\_part = "",  
 agg\_geo = "block",  
 use\_cache = TRUE  
) |>  
 dplyr::filter(grepl(  
 paste("^(", paste(fips\_codes, collapse = "|"), ")", sep = ""),  
 w\_geocode  
 )) |>  
 # check the documentation at: https://lehd.ces.census.gov/data/lodes/LODES8/LODESTechDoc8.0.pdf  
 dplyr::mutate(  
 GEOID = as.character(w\_geocode),  
 TOTAL\_EMP = C000, # Total Employment  
 AGR\_FOR\_FI = CNS01, # Agricultural, forestry, and fishing employment  
 MINING = CNS02, # Mining employment  
 CONSTRUCTI = CNS04, # Construction employment  
 MANUFACTUR = CNS05, # Manufacturing employment  
 TRANSP\_COM = CNS08 + CNS09, # Transportation, communication employment  
 WHOLESALE = CNS06, # Wholesale employment  
 RETAIL = CNS07, # Retail employment  
 FIRE = CNS10 + CNS11, # Finance / Insurance / Real Estate employment  
 SERVICES = CNS03 +  
 CNS12 +  
 CNS13 +  
 CNS14 + # Service employment  
 CNS15 +  
 CNS16 +  
 CNS17 +  
 CNS18 +  
 CNS19,  
 PUBLIC\_ADM = CNS20 # Public Administration employment  
 ) |>  
 dplyr::select(  
 GEOID,  
 TOTAL\_EMP,  
 AGR\_FOR\_FI,  
 MINING,  
 CONSTRUCTI,  
 MANUFACTUR,  
 TRANSP\_COM,  
 WHOLESALE,  
 RETAIL,  
 FIRE,  
 SERVICES,  
 PUBLIC\_ADM  
 )  
  
lscog\_emp

# A tibble: 2,450 × 12  
 GEOID TOTAL\_EMP AGR\_FOR\_FI MINING CONSTRUCTI MANUFACTUR TRANSP\_COM WHOLESALE  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 45003… 6 6 0 0 0 0 0  
 2 45003… 4 0 0 0 0 4 0  
 3 45003… 12 0 0 12 0 0 0  
 4 45003… 1 0 0 0 0 0 0  
 5 45003… 11 11 0 0 0 0 0  
 6 45003… 1 0 0 0 0 0 0  
 7 45003… 9 0 0 0 0 0 0  
 8 45003… 18 0 0 18 0 0 0  
 9 45003… 1 0 0 1 0 0 0  
10 45003… 4 4 0 0 0 0 0  
# ℹ 2,440 more rows  
# ℹ 4 more variables: RETAIL <dbl>, FIRE <dbl>, SERVICES <dbl>,  
# PUBLIC\_ADM <dbl>

#### Python

# Download LEHD WAC data at block level  
lscog\_emp = get\_lodes(  
 state=state\_abb,  
 year=2019,  
 version="LODES8",  
 lodes\_type="wac",  
 part="main",  
 segment="S000",  
 job\_type="JT00",  
 agg\_level="block",  
 cache=True,  
 return\_geometry=False  
)  
  
# Filter for specific FIPS codes  
lscog\_emp = lscog\_emp[lscog\_emp['w\_geocode'].str.match(f"^({'|'.join(fips\_codes)})")]  
  
# Create new columns with employment categories  
# Check documentation at: https://lehd.ces.census.gov/data/lodes/LODES8/LODESTechDoc8.0.pdf  
lscog\_emp = lscog\_emp.assign(  
 GEOID=lscog\_emp['w\_geocode'].astype(str),  
 TOTAL\_EMP=lscog\_emp['C000'], # Total Employment  
 AGR\_FOR\_FI=lscog\_emp['CNS01'], # Agricultural, forestry, and fishing employment  
 MINING=lscog\_emp['CNS02'], # Mining employment  
 CONSTRUCTI=lscog\_emp['CNS04'], # Construction employment  
 MANUFACTUR=lscog\_emp['CNS05'], # Manufacturing employment  
 TRANSP\_COM=lscog\_emp['CNS08'] + lscog\_emp['CNS09'], # Transportation, communication employment  
 WHOLESALE=lscog\_emp['CNS06'], # Wholesale employment  
 RETAIL=lscog\_emp['CNS07'], # Retail employment  
 FIRE=lscog\_emp['CNS10'] + lscog\_emp['CNS11'], # Finance / Insurance / Real Estate employment  
 SERVICES=(lscog\_emp['CNS03'] +  
 lscog\_emp['CNS12'] +  
 lscog\_emp['CNS13'] +  
 lscog\_emp['CNS14'] +  
 lscog\_emp['CNS15'] +  
 lscog\_emp['CNS16'] +  
 lscog\_emp['CNS17'] +  
 lscog\_emp['CNS18'] +  
 lscog\_emp['CNS19']), # Service employment  
 PUBLIC\_ADM=lscog\_emp['CNS20'] # Public Administration employment  
)  
  
# Select only the desired columns  
lscog\_emp = lscog\_emp[['GEOID', 'TOTAL\_EMP', 'AGR\_FOR\_FI', 'MINING', 'CONSTRUCTI',  
 'MANUFACTUR', 'TRANSP\_COM', 'WHOLESALE', 'RETAIL', 'FIRE',  
 'SERVICES', 'PUBLIC\_ADM']]  
  
# Display structure/info about the dataframe  
lscog\_emp

GEOID TOTAL\_EMP AGR\_FOR\_FI ... FIRE SERVICES PUBLIC\_ADM  
191 450030201001001 6 6 ... 0 0 0  
192 450030201001017 4 0 ... 0 0 0  
193 450030201001037 12 0 ... 0 0 0  
194 450030201001049 1 0 ... 0 0 0  
195 450030201002006 11 11 ... 0 0 0  
... ... ... ... ... ... ... ...  
28115 450750120002099 25 0 ... 0 0 0  
28116 450750120002108 8 0 ... 0 0 0  
28117 450750120002118 3 0 ... 0 0 0  
28118 450750120004038 3 0 ... 0 0 0  
28119 450750120004071 6 0 ... 0 5 0  
  
[2450 rows x 12 columns]

## 3.4 2020 NCES school and college enrollment data

The National Center for Education Statistics provides comprehensive educational institution data including enrollment and staffing information for transportation planning analysis.

### Public schools

Public school data is made available by National Center for Education Statistics through their Common Core of Data (CCD) program. For ease of processing, we retrieve the CCD from the ArcGIS REST service for the 2019-2020 academic year. The dataset includes total student enrollment and full-time equivalent teacher counts for each institution within the six-county region. Public schools represent major trip generation sources for both student and employee travel, requiring precise spatial location data for accurate modeling.

For more information on the NCES Public School data, refer to [CCD Online Documentation](https://nces.ed.gov/ccd/).

To retrieve the data from ArcGIS REST service, we use the {arcgislayers} package in R and create a custom function in Python to read the ArcGIS FeatureLayer or Table. This has been implemented to function similarly to the {arcgislayers} package in R, allowing us to query the service with a SQL WHERE clause and return the data as a GeoDataFrame.

#### R

# Public School Location data 2019-2020  
lscog\_pub\_sch\_enroll <- arcgislayers::arc\_read(  
 url = "https://nces.ed.gov/opengis/rest/services/K12\_School\_Locations/EDGE\_ADMINDATA\_PUBLICSCH\_1920/MapServer/0",  
 where = paste0(  
 "LSTATE = '",  
 state\_abb,  
 "' AND NMCNTY IN (",  
 paste0("'", paste0(county\_names, " County"), "'", collapse = ", "),  
 ")"  
 ),  
 alias = "label",  
 crs = project\_crs  
) |>  
 dplyr::select(  
 INSTITUTION\_ID = NCESSCH,  
 NAME = SCH\_NAME,  
 STATE = LSTATE,  
 STUDENT\_COUNT\_PUB = TOTAL,  
 TEACHER\_COUNT\_PUB = FTE  
 )  
  
lscog\_pub\_sch\_enroll

Simple feature collection with 98 features and 5 fields  
Geometry type: POINT  
Dimension: XY  
Bounding box: xmin: 1700952 ymin: 406810.6 xmax: 2203406 ymax: 724699.4  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
First 10 features:  
 INSTITUTION\_ID NAME STATE STUDENT\_COUNT\_PUB  
1 450108001163 Barnwell Elementary SC 462  
2 450075000064 Allendale Fairfax High SC 283  
3 450075001184 Allendale Elementary SC 245  
4 450075001415 Allendale-Fairfax Middle SC 268  
5 450093000119 Bamberg-Ehrhardt High SC 381  
6 450093000120 Bamberg-Ehrhardt Middle SC 188  
7 450096000122 Denmark Olar High SC 162  
8 450096000123 Denmark-Olar Middle SC 149  
9 450096001426 Denmark-Olar Elementary SC 353  
10 450098000127 Barnwell County Career Center SC 0  
 TEACHER\_COUNT\_PUB geometry  
1 32.0 POINT (1891531 517378.5)  
2 28.9 POINT (1913416 420526.6)  
3 18.0 POINT (1916344 418212.2)  
4 18.0 POINT (1913416 420526.6)  
5 28.5 POINT (1991502 535259.1)  
6 15.0 POINT (1990622 534442.6)  
7 20.0 POINT (1962410 543779)  
8 10.0 POINT (1956236 534420.3)  
9 25.0 POINT (1960237 537023.1)  
10 12.0 POINT (1894891 540093)

#### Python

# Create function to read ArcGIS FeatureLayer or Table  
def arc\_read(url, where="1=1", outFields="\*", outSR=4326, \*\*kwargs):  
 """  
 Read an ArcGIS FeatureLayer or Table to a GeoDataFrame.  
  
 Parameters:  
 url (str): The ArcGIS REST service URL ending with /MapServer/0 or /FeatureServer/0  
 where (str): SQL WHERE clause for filtering. Default: "1=1" (all records)  
 outFields (str): Comma-separated field names or "\*" for all fields. Default: "\*"  
 outSR (int): Output spatial reference EPSG code. Default: 4326  
 \*\*kwargs: Additional query parameters passed to the ArcGIS REST API  
  
 Returns:  
 geopandas.GeoDataFrame: Spatial data from the service  
 """  
  
 # Ensure URL ends with /query  
 if not url.endswith('/query'):  
 url = url.rstrip('/') + '/query'  
  
 # Build query parameters  
 params = {  
 'where': where,  
 'outFields': outFields,  
 'returnGeometry': 'true',  
 # 'geometryType': 'esriGeometryPoint',  
 'outSR': outSR,  
 'f': 'geojson'  
 }  
  
 # Add any additional parameters  
 params.update(kwargs)  
  
 # Make request  
 response = requests.get(url, params=params)  
  
 # Read as GeoDataFrame  
 return gpd.read\_file(response.text)

# Public School Enrollment data 2019-2020  
lscog\_pub\_sch\_enroll = arc\_read(  
 url="https://nces.ed.gov/opengis/rest/services/K12\_School\_Locations/EDGE\_ADMINDATA\_PUBLICSCH\_1920/MapServer/0",  
 where=f"LSTATE = '{state\_abb}' AND NMCNTY IN ('{"', '".join([f"{name} County" for name in county\_names])}')",  
 outFields='NCESSCH,SCH\_NAME,LSTATE,TOTAL,FTE'  
)  
  
# Transform CRS  
lscog\_pub\_sch\_enroll = lscog\_pub\_sch\_enroll.to\_crs(project\_crs)  
  
# Select and rename columns  
lscog\_pub\_sch\_enroll = lscog\_pub\_sch\_enroll.rename(columns={  
 'NCESSCH': 'INSTITUTION\_ID',  
 'SCH\_NAME': 'NAME',  
 'LSTATE': 'STATE',  
 'TOTAL': 'STUDENT\_COUNT\_PUB',  
 'FTE': 'TEACHER\_COUNT\_PUB'  
})  
  
lscog\_pub\_sch\_enroll

INSTITUTION\_ID ... geometry  
0 450108001163 ... POINT (1891532.112 517376.183)  
1 450075000064 ... POINT (1913417.281 420524.266)  
2 450075001184 ... POINT (1916345.361 418209.84)  
3 450075001415 ... POINT (1913417.281 420524.266)  
4 450093000119 ... POINT (1991503.106 535256.782)  
.. ... ... ...  
93 450391001291 ... POINT (2048660.583 606357.033)  
94 450391001370 ... POINT (2040865.5 608975.982)  
95 450391001604 ... POINT (2050404.085 617575.512)  
96 450391001693 ... POINT (2030130.8 597632.129)  
97 450391001694 ... POINT (2043557.083 596562.932)  
  
[98 rows x 6 columns]

### Private schools

Private school enrollment data is accessed from the NCES Private School Universe Survey (PSS) archived dataset. The data is spatially enabled using latitude and longitude coordinates and filtered to include only institutions within the study area TAZ boundaries. Private schools contribute to the regional education trip matrix and must be incorporated alongside public institutions for comprehensive coverage.

For more information on the NCES Private Schools data, refer to [PSS Online Documentation](https://nces.ed.gov/surveys/pss/).

#### R

# Private School Enrollment data 2019-2020  
lscog\_pvt\_sch\_enroll <- vroom::vroom(  
 unz(  
 file.path(  
 root,  
 "GIS/data\_external/20250315 NCES/PSS - Private/2019-20/pss1920\_pu\_csv.zip"  
 ),  
 "pss1920\_pu.csv"  
 ),  
 col\_types = vroom::cols\_only(  
 PPIN = vroom::col\_character(),  
 PINST = vroom::col\_character(),  
 PL\_STABB = vroom::col\_character(),  
 PCNTNM = vroom::col\_character(),  
 SIZE = vroom::col\_double(),  
 NUMTEACH = vroom::col\_double(),  
 LATITUDE20 = vroom::col\_double(),  
 LONGITUDE20 = vroom::col\_double()  
 )  
) |>  
 sf::st\_as\_sf(coords = c("LONGITUDE20", "LATITUDE20"), crs = "EPSG:4326") |>  
 sf::st\_transform(project\_crs) |>  
 sf::st\_filter(lscog\_taz, .predicate = st\_intersects) |>  
 dplyr::select(  
 INSTITUTION\_ID = PPIN,  
 NAME = PINST,  
 STATE = PL\_STABB,  
 STUDENT\_COUNT\_PVT = SIZE,  
 TEACHER\_COUNT\_PVT = NUMTEACH  
 )  
  
lscog\_pvt\_sch\_enroll

Simple feature collection with 24 features and 5 fields  
Geometry type: POINT  
Dimension: XY  
Bounding box: xmin: 1704062 ymin: 492304.2 xmax: 2179510 ymax: 720184.2  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 24 × 6  
 INSTITUTION\_ID NAME STATE STUDENT\_COUNT\_PVT TEACHER\_COUNT\_PVT  
 <chr> <chr> <chr> <dbl> <dbl>  
 1 K9305823 AIKENS FBC PRESCHOOL SC 1 1.3  
 2 01264947 ANDREW JACKSON ACAD… SC 3 13.3  
 3 A9106158 BARNWELL CHRISTIAN … <NA> 2 5.8  
 4 01263568 CALHOUN ACADEMY SC 3 26.6  
 5 A1771477 FIRST BAPTIST CHURC… <NA> 1 3.1  
 6 A9703151 FIRST PRESBYTERIAN … <NA> 1 2.8  
 7 BB170334 FIRST SOUTHERN METH… SC 1 6   
 8 A1102039 FOUNDATION CHRISTIA… <NA> 1 5   
 9 A0307976 GRACE CHILD DEVELOP… <NA> 1 2.9  
10 A0307978 GREATER FAITH BAPTI… <NA> 1 1   
# ℹ 14 more rows  
# ℹ 1 more variable: geometry <POINT [foot]>

#### Python

# Private School Enrollment data 2019-2020  
zip\_path = Path(root) / "GIS/data\_external/20250315 NCES/PSS - Private/2019-20/pss1920\_pu\_csv.zip"  
  
with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:  
 with zip\_ref.open('pss1920\_pu.csv') as csv\_file:  
 lscog\_pvt\_sch\_enroll = pd.read\_csv(  
 csv\_file,  
 usecols=['PPIN', 'PINST', 'PL\_STABB', 'PCNTNM', 'SIZE', 'NUMTEACH', 'LATITUDE20', 'LONGITUDE20'],  
 dtype={'PPIN': 'str', 'PINST': 'str', 'PL\_STABB': 'str', 'PCNTNM': 'str', 'SIZE': 'float64', 'NUMTEACH': 'float64'}  
 )  
  
lscog\_pvt\_sch\_enroll = gpd.GeoDataFrame(  
 lscog\_pvt\_sch\_enroll,  
 geometry=gpd.points\_from\_xy(lscog\_pvt\_sch\_enroll['LONGITUDE20'], lscog\_pvt\_sch\_enroll['LATITUDE20']),  
 crs='EPSG:4326'  
).to\_crs(project\_crs)  
  
lscog\_pvt\_sch\_enroll = gpd.sjoin(lscog\_pvt\_sch\_enroll, lscog\_taz, how='inner', predicate='intersects')[  
 ['PPIN', 'PINST', 'PL\_STABB', 'SIZE', 'NUMTEACH', 'geometry']  
].rename(columns={  
 'PPIN': 'INSTITUTION\_ID',  
 'PINST': 'NAME',  
 'PL\_STABB': 'STATE',  
 'SIZE': 'STUDENT\_COUNT\_PVT',  
 'NUMTEACH': 'TEACHER\_COUNT\_PVT'  
})  
  
lscog\_pvt\_sch\_enroll

INSTITUTION\_ID ... geometry  
17469 K9305823 ... POINT (1781275.702 629440.997)  
17474 01264947 ... POINT (1993756.78 492304.247)  
17476 A9106158 ... POINT (1914982.59 524548.558)  
17484 01263568 ... POINT (2072197.427 660303.544)  
17533 A1771477 ... POINT (1704061.773 606321.234)  
17537 A9703151 ... POINT (1780623.642 630282.337)  
17538 BB170334 ... POINT (2037420.57 608741.769)  
17543 A1102039 ... POINT (2006658.982 720184.199)  
17547 A0307976 ... POINT (1704601.274 606316.17)  
17550 A0307978 ... POINT (2047302.618 604573.723)  
17566 A1904026 ... POINT (2179509.785 547981.083)  
17571 01263692 ... POINT (1918041.289 549840.623)  
17599 01264754 ... POINT (1779301.654 629488.248)  
17601 A0308015 ... POINT (1733987.726 611499.727)  
17623 A1303185 ... POINT (1853302.82 681159.161)  
17637 A9903938 ... POINT (2047246.79 597082.682)  
17651 A0109147 ... POINT (1780574.399 631607.375)  
17657 01264288 ... POINT (1778262.542 613244.891)  
17658 K9305825 ... POINT (1779510.666 617490.702)  
17663 K9305640 ... POINT (2041004.71 611841.216)  
17672 A9703170 ... POINT (1780823.53 629681.358)  
17676 01262826 ... POINT (1781344.96 628712.135)  
17722 01932407 ... POINT (2037497.91 608547.947)  
17733 A9903957 ... POINT (1875514.037 565892.953)  
  
[24 rows x 6 columns]

### Post-secondary institutions

Post-secondary institution locations are obtained from the NCES Integrated Postsecondary Education Data System (IPEDS), filtered by state and county FIPS codes. These institutions generate significant travel demand through student commuting, employee travel, and visitor trips, making them essential components of the regional transportation network analysis.

For more information on the NCES Post-Secondary Education data, refer to [IPEDS Documentation](https://nces.ed.gov/ipeds/).

#### R

# Post-Secondary Location data 2019-2020  
lscog\_college\_loc <- arcgislayers::arc\_read(  
 url = "https://nces.ed.gov/opengis/rest/services/Postsecondary\_School\_Locations/EDGE\_GEOCODE\_POSTSECONDARYSCH\_1920/MapServer/0",  
 where = paste0(  
 "STATE = '",  
 state\_abb,  
 "' AND CNTY IN (",  
 paste0("'", fips\_codes, "'", collapse = ", "),  
 ")"  
 ),  
 alias = "label",  
 crs = project\_crs  
)  
  
lscog\_college\_loc

Simple feature collection with 11 features and 24 fields  
Geometry type: POINT  
Dimension: XY  
Bounding box: xmin: 1708029 ymin: 429827.9 xmax: 2052011 ymax: 633710.3  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
First 10 features:  
 OBJECTID UNITID NAME  
1 3411 217615 Aiken Technical College  
2 3424 217873 Claflin University  
3 3431 217989 Denmark Technical College  
4 3439 218159 Kenneth Shuler School of Cosmetology-North Augusta  
5 3448 218487 Orangeburg Calhoun Technical College  
6 3451 218645 University of South Carolina Aiken  
7 3455 218681 University of South Carolina-Salkehatchie  
8 3459 218733 South Carolina State University  
9 3468 218919 Voorhees College  
10 5829 457998 Aiken School of Cosmetology and Barbering  
 STREET CITY STATE ZIP STFIP CNTY  
1 2276 Jefferson Davis Highway Graniteville SC 29829 45 45003  
2 400 Magnolia Street Orangeburg SC 29115-4498 45 45075  
3 1126 Solomon Blatt Blvd Denmark SC 29042 45 45009  
4 1113 Knox Avenue North Augusta SC 29841 45 45003  
5 3250 Saint Matthews Rd Orangeburg SC 29118-8299 45 45075  
6 471 University Pkwy Aiken SC 29801 45 45003  
7 465 James Brandt Blvd Allendale SC 29810-0617 45 45005  
8 300 College St NE Orangeburg SC 29117-0001 45 45075  
9 481 Porter Drive Denmark SC 29042 45 45009  
10 225 Richland Ave East Aiken SC 29801 45 45003  
 NMCNTY LOCALE LAT LON CBSA  
1 Aiken County 41 33.53383 -81.84167 12260  
2 Orangeburg County 32 33.49844 -80.85432 36700  
3 Bamberg County 41 33.31336 -81.12363 N  
4 Aiken County 21 33.49759 -81.95789 12260  
5 Orangeburg County 41 33.54485 -80.82927 36700  
6 Aiken County 21 33.57270 -81.76761 12260  
7 Allendale County 41 33.01431 -81.30184 N  
8 Orangeburg County 32 33.49797 -80.84872 36700  
9 Bamberg County 32 33.30720 -81.12786 N  
10 Aiken County 21 33.55954 -81.71728 12260  
 NMCBSA CBSATYPE CSA NMCSA  
1 Augusta-Richmond County, GA-SC 1 N N  
2 Orangeburg, SC 2 192 Columbia-Orangeburg-Newberry, SC  
3 N 0 N N  
4 Augusta-Richmond County, GA-SC 1 N N  
5 Orangeburg, SC 2 192 Columbia-Orangeburg-Newberry, SC  
6 Augusta-Richmond County, GA-SC 1 N N  
7 N 0 N N  
8 Orangeburg, SC 2 192 Columbia-Orangeburg-Newberry, SC  
9 N 0 N N  
10 Augusta-Richmond County, GA-SC 1 N N  
 NECTA NMNECTA CD SLDL SLDU SCHOOLYEAR geometry  
1 N N 4502 45084 45025 2019-2020 POINT (1743560 619744.3)  
2 N N 4506 45095 45039 2019-2020 POINT (2044404 605856.3)  
3 N N 4506 45090 45040 2019-2020 POINT (1962235 538509.9)  
4 N N 4502 45083 45024 2019-2020 POINT (1708029 606866.3)  
5 N N 4506 45095 45040 2019-2020 POINT (2052011 622751.3)  
6 N N 4502 45081 45025 2019-2020 POINT (1766228 633710.3)  
7 N N 4506 45091 45045 2019-2020 POINT (1907482 429827.9)  
8 N N 4506 45095 45039 2019-2020 POINT (2046110 605685.6)  
9 N N 4506 45090 45040 2019-2020 POINT (1960939 536271.9)  
10 N N 4502 45081 45026 2019-2020 POINT (1781522 628812.8)

#### Python

# Post-Secondary Location data 2019-2020  
lscog\_college\_loc = arc\_read(  
 url="https://nces.ed.gov/opengis/rest/services/Postsecondary\_School\_Locations/EDGE\_GEOCODE\_POSTSECONDARYSCH\_1920/MapServer/0",  
 where=f"STATE = '{state\_abb}' AND CNTY IN ('{"', '".join([f"{fip}" for fip in fips\_codes])}')",  
 outFields='\*',  
 outSR=project\_crs  
)  
  
lscog\_college\_loc = lscog\_college\_loc.to\_crs(project\_crs)  
  
lscog\_college\_loc

OBJECTID UNITID ... SCHOOLYEAR geometry  
0 3411 217615 ... 2019-2020 POINT (1743561.221 619741.983)  
1 3424 217873 ... 2019-2020 POINT (2044404.666 605853.958)  
2 3431 217989 ... 2019-2020 POINT (1962236.326 538507.569)  
3 3439 218159 ... 2019-2020 POINT (1708030.354 606863.953)  
4 3448 218487 ... 2019-2020 POINT (2052011.899 622748.89)  
5 3451 218645 ... 2019-2020 POINT (1766228.593 633707.888)  
6 3455 218681 ... 2019-2020 POINT (1907483.079 429825.628)  
7 3459 218733 ... 2019-2020 POINT (2046110.928 605683.233)  
8 3468 218919 ... 2019-2020 POINT (1960940.241 536269.53)  
9 5829 457998 ... 2019-2020 POINT (1781523.402 628810.398)  
10 6683 488022 ... 2019-2020 POINT (2043606.983 603651.758)  
  
[11 rows x 25 columns]

# 4. Clean data

The data cleaning process involves harmonizing multiple Census data sources to create a comprehensive socioeconomic dataset at the census block level. This requires careful interpolation and integration of American Community Survey (ACS) estimates with Decennial Census counts to maintain spatial consistency and statistical accuracy.

## 4.1 Household-weighted interpolation

The interpolation process transfers ACS block group data to individual census blocks using household counts as weights. This method ensures that socioeconomic characteristics are distributed proportionally based on residential density rather than simple geometric overlay. The {tidycensus} package provides robust interpolation functionality that preserves the extensive nature of count variables while maintaining spatial relationships. For Python, the interpolate\_pw() function is implemented to achieve similar functionality using population-weighted interpolation.

#### R

# Interpolate ACS data to Decennial Census blocks  
lscog\_acs\_cb <- tidycensus::interpolate\_pw(  
 from = lscog\_acs,  
 to = lscog\_dec,  
 to\_id = "GEOID",  
 extensive = TRUE,  
 weights = lscog\_dec,  
 crs = project\_crs,  
 weight\_column = "HH"  
)  
  
lscog\_acs\_cb

Simple feature collection with 13961 features and 5 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 13,961 × 6  
 GEOID geometry INC\_TOTAL INC\_14999 INC\_49999 INC\_50000  
 <chr> <MULTIPOLYGON [foot]> <dbl> <dbl> <dbl> <dbl>  
 1 4501795040… (((2084301 622308.5, 208… 0 0 0 0   
 2 4501795040… (((2112519 626782.2, 211… 0 0 0 0   
 3 4501795020… (((2067775 662339.5, 206… 20.6 4.27 7.56 8.72   
 4 4501795040… (((2087947 684345.6, 208… 0 0 0 0   
 5 4507501050… (((2089073 552687.4, 208… 5.78 1.94 0.924 2.92   
 6 4507501170… (((2046694 542273.9, 204… 2.48 0.393 1.30 0.780  
 7 4507501170… (((2056704 510850.5, 205… 0 0 0 0   
 8 4507501180… (((1960896 587381.7, 196… 3.19 0.694 1.42 1.08   
 9 4507501200… (((2000969 657869.3, 200… 0 0 0 0   
10 4501795010… (((2016178 708106.6, 201… 0 0 0 0   
# ℹ 13,951 more rows

#### Python

def interpolate\_pw(from\_gdf, to\_gdf, weights\_gdf, to\_id=None, extensive=True,  
 weight\_column=None, weight\_placement='surface', crs=None):  
 """  
 Population-weighted areal interpolation between geometries.  
  
 Transfers numeric data from source geometries to target geometries using  
 population-weighted interpolation based on point weights (e.g., census blocks).  
  
 Parameters:  
 -----------  
 from\_gdf : GeoDataFrame  
 Source geometries with numeric data to interpolate  
 to\_gdf : GeoDataFrame  
 Target geometries to interpolate data to  
 weights\_gdf : GeoDataFrame  
 Weight geometries (e.g., census blocks) used for interpolation.  
 If polygons, will be converted to points. Can be the same as to\_gdf.  
 to\_id : str, optional  
 Column name for unique identifier in target geometries.  
 If None, creates an 'id' column.  
 extensive : bool, default True  
 If True, return weighted sums (for counts).  
 If False, return weighted means (for rates/percentages).  
 weight\_column : str, optional  
 Column name in weights\_gdf for weighting (e.g., 'POP', 'HH').  
 If None, all weights are equal.  
 weight\_placement : str, default 'surface'  
 How to convert polygons to points: 'surface' or 'centroid'  
 crs : str or CRS object, optional  
 Coordinate reference system to project all datasets to  
  
 Returns:  
 --------  
 GeoDataFrame  
 Target geometries with interpolated numeric values  
 """  
  
 # Input validation  
 if not all(isinstance(gdf, gpd.GeoDataFrame) for gdf in [from\_gdf, to\_gdf, weights\_gdf]):  
 raise ValueError("All inputs must be GeoDataFrames")  
  
 # Make copies to avoid modifying originals  
 from\_gdf = from\_gdf.copy()  
 to\_gdf = to\_gdf.copy()  
 weights\_gdf = weights\_gdf.copy()  
  
 # Set CRS if provided  
 if crs:  
 from\_gdf = from\_gdf.to\_crs(crs)  
 to\_gdf = to\_gdf.to\_crs(crs)  
 weights\_gdf = weights\_gdf.to\_crs(crs)  
  
 # Check CRS consistency  
 if not (from\_gdf.crs == to\_gdf.crs == weights\_gdf.crs):  
 raise ValueError("All inputs must have the same CRS")  
  
 # Handle to\_id  
 if to\_id is None:  
 to\_id = 'id'  
 to\_gdf[to\_id] = to\_gdf.index.astype(str)  
  
 # Remove conflicting columns  
 if to\_id in from\_gdf.columns:  
 from\_gdf = from\_gdf.drop(columns=[to\_id])  
  
 # Create unique from\_id  
 from\_id = 'from\_id'  
 from\_gdf[from\_id] = from\_gdf.index.astype(str)  
  
 # Handle weight column  
 if weight\_column is None:  
 weight\_column = 'interpolation\_weight'  
 weights\_gdf[weight\_column] = 1.0  
 else:  
 # Rename to avoid conflicts  
 weights\_gdf['interpolation\_weight'] = weights\_gdf[weight\_column]  
 weight\_column = 'interpolation\_weight'  
  
 # Convert weights to points if needed  
 if weights\_gdf.geometry.geom\_type.iloc[0] in ['Polygon', 'MultiPolygon']:  
 if weight\_placement == 'surface':  
 weights\_gdf = weights\_gdf.copy()  
 weights\_gdf.geometry = weights\_gdf.geometry.representative\_point()  
 elif weight\_placement == 'centroid':  
 weights\_gdf = weights\_gdf.copy()  
 weights\_gdf.geometry = weights\_gdf.geometry.centroid  
 else:  
 raise ValueError("weight\_placement must be 'surface' or 'centroid'")  
  
 # Keep only weight column and geometry  
 weight\_points = weights\_gdf[[weight\_column, 'geometry']].copy()  
  
 # Calculate denominators (total weights per source geometry)  
 with warnings.catch\_warnings():  
 warnings.filterwarnings('ignore', category=UserWarning)  
 source\_weights = gpd.sjoin(from\_gdf, weight\_points, how='left', predicate='contains')  
  
 denominators = (source\_weights.groupby(from\_id)[weight\_column]  
 .sum()  
 .reset\_index()  
 .rename(columns={weight\_column: 'weight\_total'}))  
  
 # Calculate intersections between from and to  
 with warnings.catch\_warnings():  
 warnings.filterwarnings('ignore', category=UserWarning)  
 intersections = gpd.overlay(from\_gdf, to\_gdf, how='intersection')  
  
 # Filter to keep only polygon intersections  
 intersections = intersections[intersections.geometry.geom\_type.isin(['Polygon', 'MultiPolygon', 'GeometryCollection'])]  
  
 if len(intersections) == 0:  
 raise ValueError("No valid polygon intersections found between source and target geometries")  
  
 # Add intersection ID  
 intersections['intersection\_id'] = range(len(intersections))  
  
 # Spatial join intersections with weight points to get weights within each intersection  
 with warnings.catch\_warnings():  
 warnings.filterwarnings('ignore', category=UserWarning)  
 intersection\_weights = gpd.sjoin(intersections, weight\_points, how='left', predicate='contains')  
  
 # Calculate intersection values (sum of weights per intersection)  
 intersection\_values = (intersection\_weights.groupby('intersection\_id')[weight\_column]  
 .sum()  
 .reset\_index()  
 .rename(columns={weight\_column: 'intersection\_value'}))  
  
 # Merge back to intersections and keep only unique intersections  
 intersections = intersections.merge(intersection\_values, on='intersection\_id', how='left')  
 intersections['intersection\_value'] = intersections['intersection\_value'].fillna(0)  
  
 # Remove duplicates created by the spatial join  
 intersections = intersections.drop\_duplicates(subset='intersection\_id')  
  
 # Merge with denominators to calculate weight coefficients  
 intersections = intersections.merge(denominators, on=from\_id, how='left')  
 intersections['weight\_total'] = intersections['weight\_total'].fillna(1)  
  
 # Calculate weight coefficients (intersection weight / total weight in source)  
 intersections.loc[intersections['weight\_total'] > 0, 'weight\_coef'] = (  
 intersections['intersection\_value'] / intersections['weight\_total']  
 )  
 intersections['weight\_coef'] = intersections['weight\_coef'].fillna(0)  
  
 # Get numeric columns from source data  
 numeric\_cols = from\_gdf.select\_dtypes(include=[np.number]).columns  
 # Remove ID columns  
 numeric\_cols = [col for col in numeric\_cols if col not in [from\_id]]  
  
 # Prepare intersection data for interpolation  
 intersection\_data = intersections[[from\_id, to\_id, 'weight\_coef'] + numeric\_cols].copy()  
  
 if extensive:  
 # For extensive variables: multiply by weight coefficient, then sum by target  
 for col in numeric\_cols:  
 intersection\_data[col] = intersection\_data[col] \* intersection\_data['weight\_coef']  
  
 interpolated = (intersection\_data.groupby(to\_id)[numeric\_cols]  
 .sum()  
 .reset\_index())  
 else:  
 # For intensive variables: weighted average  
 interpolated\_data = []  
 for target\_id in intersection\_data[to\_id].unique():  
 target\_data = intersection\_data[intersection\_data[to\_id] == target\_id]  
 if len(target\_data) > 0 and target\_data['weight\_coef'].sum() > 0:  
 weighted\_vals = {}  
 for col in numeric\_cols:  
 weighted\_vals[col] = (target\_data[col] \* target\_data['weight\_coef']).sum() / target\_data['weight\_coef'].sum()  
 weighted\_vals[to\_id] = target\_id  
 interpolated\_data.append(weighted\_vals)  
  
 interpolated = pd.DataFrame(interpolated\_data)  
  
 # Merge with target geometries  
 result = to\_gdf[[to\_id, 'geometry']].merge(interpolated, on=to\_id, how='left')  
  
 # Fill NaN values with 0 for missing interpolations  
 for col in numeric\_cols:  
 if col in result.columns:  
 result[col] = result[col].fillna(0)  
  
 return result

# Interpolate ACS data to Decennial Census blocks  
lscog\_acs\_cb = interpolate\_pw(  
 from\_gdf=lscog\_acs,  
 to\_gdf=lscog\_dec,  
 weights\_gdf=lscog\_dec,  
 to\_id='GEOID',  
 extensive=True,  
 weight\_column='HH',  
 crs=project\_crs  
)  
  
lscog\_acs\_cb.head()

GEOID ... INC\_50000  
0 450179504004051 ... 0.000000  
1 450179504003011 ... 0.000000  
2 450179502011045 ... 8.723288  
3 450179504001020 ... 0.000000  
4 450750105003029 ... 2.916335  
  
[5 rows x 6 columns]

The comparative visualization reveals the increased spatial resolution achieved through interpolation. Block-level data provides more granular detail for transportation modeling applications, enabling better representation of local variations in income distribution across the study area.

#### R

# Compare before and after interpolation  
mapview::mapview(lscog\_acs\_cb, zcol = "INC\_49999", color = NA) |  
 mapview::mapview(lscog\_acs, zcol = "INC\_49999", color = NA)

#### Python

# Compare before and after interpolation  
lscog\_acs\_cb.explore(column="INC\_49999", color="blue", legend=True, tiles="CartoDB positron") |\  
 lscog\_acs.explore(column="INC\_49999", color="red", legend=True, tiles="CartoDB positron")

## 4.2 Combine population and households

The integration step merges the interpolated ACS socioeconomic data with the Decennial Census population and household counts. This join operation creates a unified dataset containing both demographic totals and detailed characteristics at the census block level. The left join ensures that all census blocks retain their geographic boundaries while incorporating available socioeconomic attributes.

#### R

## Combine ACS Data to Decennial data  
lscog\_pop\_hh <- lscog\_dec |>  
 dplyr::left\_join(  
 sf::st\_drop\_geometry(lscog\_acs\_cb),  
 by = dplyr::join\_by(GEOID)  
 )  
  
lscog\_pop\_hh

Simple feature collection with 13961 features and 14 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 13,961 × 15  
 GEOID TOTPOP GQPOP HHPOP HH HH\_1 HH\_2 HH\_3 HH\_4 DU  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 450179504004051 0 0 0 0 0 0 0 0 0  
 2 450179504003011 0 0 0 0 0 0 0 0 0  
 3 450179502011045 54 4 50 16 3 1 3 9 18  
 4 450179504001020 0 0 0 0 0 0 0 0 0  
 5 450750105003029 13 0 13 4 0 3 0 1 4  
 6 450750117021009 10 0 10 3 3 0 0 0 8  
 7 450750117011051 0 0 0 0 0 0 0 0 0  
 8 450750118021087 6 0 6 4 0 1 2 1 4  
 9 450750120003020 0 0 0 0 0 0 0 0 0  
10 450179501003046 18 0 18 0 0 0 0 0 1  
# ℹ 13,951 more rows  
# ℹ 5 more variables: geometry <MULTIPOLYGON [foot]>, INC\_TOTAL <dbl>,  
# INC\_14999 <dbl>, INC\_49999 <dbl>, INC\_50000 <dbl>

#### Python

## Combine ACS Data to Decennial data  
lscog\_pop\_hh = lscog\_dec.merge(  
 lscog\_acs\_cb.drop(columns=['geometry']),  
 on='GEOID',  
 how='left'  
)  
  
lscog\_pop\_hh

GEOID TOTPOP GQPOP ... INC\_14999 INC\_49999 INC\_50000  
0 450179504004051 0 0 ... 0.000000 0.000000 0.000000  
1 450179504003011 0 0 ... 0.000000 0.000000 0.000000  
2 450179502011045 54 4 ... 4.273973 7.561644 8.723288  
3 450179504001020 0 0 ... 0.000000 0.000000 0.000000  
4 450750105003029 13 0 ... 1.944223 0.924303 2.916335  
... ... ... ... ... ... ... ...  
13956 450059705003050 5 0 ... 0.000000 0.000000 0.000000  
13957 450030203042000 0 0 ... 0.000000 0.000000 0.000000  
13958 450030218002115 8 0 ... 0.101167 0.470817 0.295720  
13959 450030206012008 0 0 ... 0.000000 0.000000 0.000000  
13960 450059705002005 0 0 ... 0.000000 0.000000 0.000000  
  
[13961 rows x 15 columns]

Income category adjustments reconcile the interpolated ACS estimates with actual household counts from the Decennial Census. The proportional allocation method redistributes income categories based on the ratio of interpolated totals to observed household counts, maintaining consistency between data sources. The three-tier income classification (under $15,000, $15,000-$49,999, and $50,000 and above) provides sufficient granularity for travel demand modeling while ensuring statistical reliability.

#### R

## Combine adjusted HH income level to Decennial census instead of ACS  
lscog\_pop\_hh <- lscog\_pop\_hh |>  
 dplyr::mutate(  
 INC\_49999 = tidyr::replace\_na(round(INC\_49999 / INC\_TOTAL \* HH, 0), 0),  
 INC\_50000 = tidyr::replace\_na(round(INC\_50000 / INC\_TOTAL \* HH, 0), 0),  
 INC\_14999 = HH - (INC\_49999 + INC\_50000)  
 ) |>  
 dplyr::select(-INC\_TOTAL)  
  
lscog\_pop\_hh

Simple feature collection with 13961 features and 13 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 13,961 × 14  
 GEOID TOTPOP GQPOP HHPOP HH HH\_1 HH\_2 HH\_3 HH\_4 DU  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 450179504004051 0 0 0 0 0 0 0 0 0  
 2 450179504003011 0 0 0 0 0 0 0 0 0  
 3 450179502011045 54 4 50 16 3 1 3 9 18  
 4 450179504001020 0 0 0 0 0 0 0 0 0  
 5 450750105003029 13 0 13 4 0 3 0 1 4  
 6 450750117021009 10 0 10 3 3 0 0 0 8  
 7 450750117011051 0 0 0 0 0 0 0 0 0  
 8 450750118021087 6 0 6 4 0 1 2 1 4  
 9 450750120003020 0 0 0 0 0 0 0 0 0  
10 450179501003046 18 0 18 0 0 0 0 0 1  
# ℹ 13,951 more rows  
# ℹ 4 more variables: geometry <MULTIPOLYGON [foot]>, INC\_14999 <dbl>,  
# INC\_49999 <dbl>, INC\_50000 <dbl>

#### Python

## Combine adjusted HH income level to Decennial census instead of ACS  
lscog\_pop\_hh["INC\_49999"] = ((lscog\_pop\_hh["INC\_49999"] / lscog\_pop\_hh["INC\_TOTAL"]) \* lscog\_pop\_hh["HH"]).round().fillna(0)  
lscog\_pop\_hh["INC\_50000"] = ((lscog\_pop\_hh["INC\_50000"] / lscog\_pop\_hh["INC\_TOTAL"]) \* lscog\_pop\_hh["HH"]).round().fillna(0)  
lscog\_pop\_hh["INC\_14999"] = lscog\_pop\_hh["HH"] - (lscog\_pop\_hh["INC\_49999"] + lscog\_pop\_hh["INC\_50000"])  
  
lscog\_pop\_hh = lscog\_pop\_hh.drop(columns="INC\_TOTAL")  
lscog\_pop\_hh

GEOID TOTPOP GQPOP ... INC\_14999 INC\_49999 INC\_50000  
0 450179504004051 0 0 ... 0.0 0.0 0.0  
1 450179504003011 0 0 ... 0.0 0.0 0.0  
2 450179502011045 54 4 ... 3.0 6.0 7.0  
3 450179504001020 0 0 ... 0.0 0.0 0.0  
4 450750105003029 13 0 ... 1.0 1.0 2.0  
... ... ... ... ... ... ... ...  
13956 450059705003050 5 0 ... 0.0 0.0 0.0  
13957 450030203042000 0 0 ... 0.0 0.0 0.0  
13958 450030218002115 8 0 ... 0.0 1.0 0.0  
13959 450030206012008 0 0 ... 0.0 0.0 0.0  
13960 450059705002005 0 0 ... 0.0 0.0 0.0  
  
[13961 rows x 14 columns]

## 4.3 Employment data

The employment integration incorporates LEHD (Longitudinal Employer-Household Dynamics) workplace area characteristics into the combined dataset. This addition provides employment counts by census block, enabling the development of trip attraction models and work-based travel pattern analysis. The merge operation maintains the geographic integrity of census blocks while adding employment variables essential for comprehensive transportation planning.

#### R

# Join LEHD Data to the Decennial data  
lscog\_pop\_hh\_emp <- lscog\_pop\_hh |>  
 dplyr::left\_join(lscog\_emp, by = dplyr::join\_by(GEOID))  
  
lscog\_pop\_hh\_emp

Simple feature collection with 13961 features and 24 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691517 ymin: 331891.7 xmax: 2237472 ymax: 744669.5  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 13,961 × 25  
 GEOID TOTPOP GQPOP HHPOP HH HH\_1 HH\_2 HH\_3 HH\_4 DU  
 <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 450179504004051 0 0 0 0 0 0 0 0 0  
 2 450179504003011 0 0 0 0 0 0 0 0 0  
 3 450179502011045 54 4 50 16 3 1 3 9 18  
 4 450179504001020 0 0 0 0 0 0 0 0 0  
 5 450750105003029 13 0 13 4 0 3 0 1 4  
 6 450750117021009 10 0 10 3 3 0 0 0 8  
 7 450750117011051 0 0 0 0 0 0 0 0 0  
 8 450750118021087 6 0 6 4 0 1 2 1 4  
 9 450750120003020 0 0 0 0 0 0 0 0 0  
10 450179501003046 18 0 18 0 0 0 0 0 1  
# ℹ 13,951 more rows  
# ℹ 15 more variables: geometry <MULTIPOLYGON [foot]>, INC\_14999 <dbl>,  
# INC\_49999 <dbl>, INC\_50000 <dbl>, TOTAL\_EMP <dbl>, AGR\_FOR\_FI <dbl>,  
# MINING <dbl>, CONSTRUCTI <dbl>, MANUFACTUR <dbl>, TRANSP\_COM <dbl>,  
# WHOLESALE <dbl>, RETAIL <dbl>, FIRE <dbl>, SERVICES <dbl>, PUBLIC\_ADM <dbl>

#### Python

# Join LEHD Data to the Decennial data  
lscog\_pop\_hh\_emp = lscog\_pop\_hh.merge(  
 lscog\_emp,  
 on='GEOID',  
 how='left'  
)  
  
lscog\_pop\_hh\_emp

GEOID TOTPOP GQPOP ... FIRE SERVICES PUBLIC\_ADM  
0 450179504004051 0 0 ... NaN NaN NaN  
1 450179504003011 0 0 ... NaN NaN NaN  
2 450179502011045 54 4 ... NaN NaN NaN  
3 450179504001020 0 0 ... NaN NaN NaN  
4 450750105003029 13 0 ... NaN NaN NaN  
... ... ... ... ... ... ... ...  
13956 450059705003050 5 0 ... NaN NaN NaN  
13957 450030203042000 0 0 ... NaN NaN NaN  
13958 450030218002115 8 0 ... NaN NaN NaN  
13959 450030206012008 0 0 ... NaN NaN NaN  
13960 450059705002005 0 0 ... NaN NaN NaN  
  
[13961 rows x 25 columns]

# 5. Export raw data

The data export process creates standardized datasets for travel demand model development, maintaining both tabular and spatial formats to support various modeling applications. All exports follow consistent file naming conventions and directory structures to facilitate model integration and data management workflows.

## 5.1 TAZ data

The Traffic Analysis Zone (TAZ) boundary export provides the fundamental geographic framework for the regional travel demand model. The blank TAZ file serves as a template for subsequent socioeconomic data allocation, containing only zone identification fields and geometric boundaries without attribute data.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_taz |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_taz\_blank.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_taz.to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_taz\_blank.csv",  
 index=False  
)

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_taz |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_taz\_blank",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_taz.to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_taz\_blank',  
 driver='FileGDB'  
)

## 5.2 Census blocks to TAZ conversion

The block-to-TAZ conversion table establishes the critical linkage between fine-scale Census geography and the modeling zone system. This crosswalk file enables the aggregation of block-level socioeconomic data to TAZ boundaries while maintaining traceability to source geographies.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_cb |>  
 sf::st\_join(lscog\_taz) |>  
 dplyr::select(GEOID20, ID, TAZ\_ID) |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_block\_taz.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_cb.merge(lscog\_taz[['ID', 'TAZ\_ID']], left\_on='GEOID20', right\_on='ID')[['GEOID20', 'ID', 'TAZ\_ID']].to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_block\_taz.csv",  
 index=False  
)

## 5.3 Decennial census data at census block level

The decennial census block export captures the foundational demographic counts used throughout the modeling process. This dataset provides the most reliable population and household totals at the finest geographic resolution, serving as the base for all subsequent data integration and validation steps.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_dec |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_dec\_block.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_dec.to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_dec\_block.csv",  
 index=False  
)

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_dec |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_dec\_block",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_dec.to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_dec\_block',  
 driver='FileGDB'  
)

## 5.4 ACS estimates at census block level

The interpolated ACS data export delivers income distribution estimates at the census block level, providing the socioeconomic stratification necessary for trip generation modeling. This processed dataset represents the final product of the household-weighted interpolation methodology, ready for direct integration into the travel demand model framework.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_acs\_cb |>  
 dplyr::select(GEOID, INC\_14999, INC\_49999, INC\_50000) |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_acs\_block.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_acs\_cb[['GEOID', 'INC\_14999', 'INC\_49999', 'INC\_50000']].to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_acs\_block.csv",  
 index=False  
)

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_acs\_cb |>  
 dplyr::select(GEOID, INC\_14999, INC\_49999, INC\_50000) |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_acs\_block",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_acs\_cb[['GEOID', 'INC\_14999', 'INC\_49999', 'INC\_50000']].to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_acs\_block',  
 driver='FileGDB'  
)

## 5.5 LEHD data at census block level

The employment data export provides comprehensive workplace characteristics by industry sector at the census block level. This dataset captures the spatial distribution of employment opportunities across the study region, supporting both trip attraction modeling and economic impact analysis.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_pop\_hh\_emp |>  
 dplyr::select(GEOID, TOTAL\_EMP:PUBLIC\_ADM) |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_emp\_block.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_pop\_hh\_emp[['GEOID', 'TOTAL\_EMP', 'AGR\_FOR\_FI', 'MINING', 'CONSTRUCTI',  
 'MANUFACTUR', 'TRANSP\_COM', 'WHOLESALE', 'RETAIL', 'FIRE',  
 'SERVICES', 'PUBLIC\_ADM']].to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_emp\_block.csv",  
 index=False  
)

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_pop\_hh\_emp |>  
 dplyr::select(GEOID, TOTAL\_EMP:PUBLIC\_ADM) |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_emp\_block",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_pop\_hh\_emp[['GEOID', 'TOTAL\_EMP', 'AGR\_FOR\_FI', 'MINING', 'CONSTRUCTI',  
 'MANUFACTUR', 'TRANSP\_COM', 'WHOLESALE', 'RETAIL', 'FIRE',  
 'SERVICES', 'PUBLIC\_ADM']].to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_emp\_block',  
 driver='FileGDB'  
 )

## 5.6 Public schools to TAZ

The public school location export integrates educational facility data with the TAZ system, providing essential inputs for school-related trip modeling. Student and teacher counts by facility support the development of specialized trip generation rates for educational purposes.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_pub\_sch\_enroll |>  
 sf::st\_join(lscog\_taz |> dplyr::select(ID, TAZ\_ID)) |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_pubsch\_loc.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_pub\_sch\_enroll.sjoin(  
 lscog\_taz[['ID', 'TAZ\_ID']],  
 how='left'  
)[['INSTITUTION\_ID', 'NAME', 'STATE', 'STUDENT\_COUNT\_PUB', 'TEACHER\_COUNT\_PUB', 'geometry']] \  
 .to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_pubsch\_loc.csv",  
 index=False  
 )

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_pub\_sch\_enroll |>  
 sf::st\_join(lscog\_taz |> dplyr::select(ID, TAZ\_ID)) |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_pubsch\_loc",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_pub\_sch\_enroll.sjoin(  
 lscog\_taz[['ID', 'TAZ\_ID']],  
 how='left'  
).to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_pubsch\_loc',  
 driver='FileGDB'  
)

## 5.7 Private schools to TAZ

The private school dataset complements the public education data by capturing enrollment patterns in private educational institutions. This comprehensive coverage of educational facilities ensures that all school-related travel demand is properly represented in the regional model.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_pvt\_sch\_enroll |>  
 sf::st\_join(lscog\_taz |> dplyr::select(ID, TAZ\_ID)) |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_raw/lscog\_pvtsch\_loc.csv"  
 ),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_pvt\_sch\_enroll.sjoin(  
 lscog\_taz[['ID', 'TAZ\_ID']],  
 how='left'  
)[['INSTITUTION\_ID', 'NAME', 'STATE', 'STUDENT\_COUNT\_PVT', 'TEACHER\_COUNT\_PVT', 'geometry']] \  
 .to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_pvtsch\_loc.csv",  
 index=False  
 )

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_pvt\_sch\_enroll |>  
 sf::st\_join(lscog\_taz |> dplyr::select(ID, TAZ\_ID)) |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_pvtsch\_loc",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_pvt\_sch\_enroll.sjoin(  
 lscog\_taz[['ID', 'TAZ\_ID']],  
 how='left'  
).to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb",  
 layer='lscog\_pvtsch\_loc',  
 driver='FileGDB'  
)

# 6. Aggregate data to TAZ

The aggregation process transforms fine-scale census block data into TAZ-level inputs suitable for travel demand modeling. This spatial aggregation preserves the total counts while organizing data according to the modeling zone structure required for trip generation and distribution analysis.

## 6.1 Population, households, and employment

The spatial join operation aggregates all demographic, housing, and employment variables from census blocks to their corresponding TAZs using centroid-based assignment. This process ensures that each block’s socioeconomic characteristics are properly allocated to the appropriate modeling zone while maintaining data integrity through comprehensive summation of all relevant variables.

#### R

# Aggregate population, households, and employment to TAZ  
lscog\_taz\_pop <- lscog\_taz |>  
 sf::st\_join(  
 lscog\_pop\_hh\_emp |> sf::st\_centroid(of\_largest\_polygon = TRUE)  
 ) |>  
 dplyr::group\_by(  
 ID,  
 Area,  
 TAZ\_ID,  
 COUNTY,  
 AREA\_TYPE,  
 COUNTYID,  
 .drop = FALSE  
 ) |>  
 dplyr::summarize(  
 .groups = "drop",  
 # Population and Household Size  
 TOTPOP = sum(TOTPOP, na.rm = TRUE),  
 GQPOP = sum(GQPOP, na.rm = TRUE),  
 HHPOP = sum(HHPOP, na.rm = TRUE),  
 HH = sum(HH, na.rm = TRUE),  
 HH\_1 = sum(HH\_1, na.rm = TRUE),  
 HH\_2 = sum(HH\_2, na.rm = TRUE),  
 HH\_3 = sum(HH\_3, na.rm = TRUE),  
 HH\_4 = sum(HH\_4, na.rm = TRUE),  
 DU = sum(DU, na.rm = TRUE),  
 # Household Income  
 INC\_14999 = sum(INC\_14999, na.rm = TRUE),  
 INC\_49999 = sum(INC\_49999, na.rm = TRUE),  
 INC\_50000 = sum(INC\_50000, na.rm = TRUE),  
 # Employment  
 TOTAL\_EMP = sum(TOTAL\_EMP, na.rm = TRUE),  
 AGR\_FOR\_FI = sum(AGR\_FOR\_FI, na.rm = TRUE),  
 MINING = sum(MINING, na.rm = TRUE),  
 CONSTRUCTI = sum(CONSTRUCTI, na.rm = TRUE),  
 MANUFACTUR = sum(MANUFACTUR, na.rm = TRUE),  
 TRANSP\_COM = sum(TRANSP\_COM, na.rm = TRUE),  
 WHOLESALE = sum(WHOLESALE, na.rm = TRUE),  
 RETAIL = sum(RETAIL, na.rm = TRUE),  
 FIRE = sum(FIRE, na.rm = TRUE),  
 SERVICES = sum(SERVICES, na.rm = TRUE),  
 PUBLIC\_ADM = sum(PUBLIC\_ADM, na.rm = TRUE)  
 )  
  
lscog\_taz\_pop

Simple feature collection with 585 features and 29 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691490 ymin: 331894 xmax: 2237471 ymax: 744679.9  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 585 × 30  
 ID Area TAZ\_ID COUNTY AREA\_TYPE COUNTYID TOTPOP GQPOP HHPOP HH HH\_1  
 <int> <dbl> <int> <chr> <chr> <int> <dbl> <dbl> <dbl> <dbl> <dbl>  
 1 3.01e6 0.846 3.01e6 Aiken… SUBURBAN 45003 66 0 66 20 2  
 2 3.01e6 1.10 3.01e6 Aiken… URBAN 45003 1299 0 1299 482 84  
 3 3.01e6 0.395 3.01e6 Aiken… URBAN 45003 657 0 657 268 46  
 4 3.01e6 0.338 3.01e6 Aiken… URBAN 45003 593 1 592 237 67  
 5 3.01e6 0.381 3.01e6 Aiken… URBAN 45003 462 0 462 261 152  
 6 3.01e6 0.125 3.01e6 Aiken… URBAN 45003 383 0 383 189 64  
 7 3.01e6 0.0879 3.01e6 Aiken… URBAN 45003 160 0 160 62 5  
 8 3.01e6 0.103 3.01e6 Aiken… URBAN 45003 420 0 420 194 50  
 9 3.01e6 0.0659 3.01e6 Aiken… URBAN 45003 142 0 142 67 19  
10 3.01e6 0.0279 3.01e6 Aiken… URBAN 45003 100 0 100 71 30  
# ℹ 575 more rows  
# ℹ 19 more variables: HH\_2 <dbl>, HH\_3 <dbl>, HH\_4 <dbl>, DU <dbl>,  
# INC\_14999 <dbl>, INC\_49999 <dbl>, INC\_50000 <dbl>, TOTAL\_EMP <dbl>,  
# AGR\_FOR\_FI <dbl>, MINING <dbl>, CONSTRUCTI <dbl>, MANUFACTUR <dbl>,  
# TRANSP\_COM <dbl>, WHOLESALE <dbl>, RETAIL <dbl>, FIRE <dbl>,  
# SERVICES <dbl>, PUBLIC\_ADM <dbl>, SHAPE <MULTIPOLYGON [foot]>

#### Python

# Aggregate population, households, and employment to TAZ  
lscog\_taz\_pop = (lscog\_taz  
 .sjoin(  
 lscog\_pop\_hh\_emp.assign(geometry=lscog\_pop\_hh\_emp.geometry.centroid).to\_crs(lscog\_taz.crs),  
 how='left'  
 )  
 .groupby(['ID', 'Area', 'TAZ\_ID', 'COUNTY', 'AREA\_TYPE', 'COUNTYID'],  
 as\_index=False, dropna=False)  
 .agg({  
 # Population and Household Size  
 'TOTPOP': lambda x: x.sum(skipna=True),  
 'GQPOP': lambda x: x.sum(skipna=True),  
 'HHPOP': lambda x: x.sum(skipna=True),  
 'HH': lambda x: x.sum(skipna=True),  
 'HH\_1': lambda x: x.sum(skipna=True),  
 'HH\_2': lambda x: x.sum(skipna=True),  
 'HH\_3': lambda x: x.sum(skipna=True),  
 'HH\_4': lambda x: x.sum(skipna=True),  
 'DU': lambda x: x.sum(skipna=True),  
 # Household Income  
 'INC\_14999': lambda x: x.sum(skipna=True),  
 'INC\_49999': lambda x: x.sum(skipna=True),  
 'INC\_50000': lambda x: x.sum(skipna=True),  
 # Employment  
 'TOTAL\_EMP': lambda x: x.sum(skipna=True),  
 'AGR\_FOR\_FI': lambda x: x.sum(skipna=True),  
 'MINING': lambda x: x.sum(skipna=True),  
 'CONSTRUCTI': lambda x: x.sum(skipna=True),  
 'MANUFACTUR': lambda x: x.sum(skipna=True),  
 'TRANSP\_COM': lambda x: x.sum(skipna=True),  
 'WHOLESALE': lambda x: x.sum(skipna=True),  
 'RETAIL': lambda x: x.sum(skipna=True),  
 'FIRE': lambda x: x.sum(skipna=True),  
 'SERVICES': lambda x: x.sum(skipna=True),  
 'PUBLIC\_ADM': lambda x: x.sum(skipna=True),  
 'geometry': 'first'  
 })  
)  
  
lscog\_taz\_pop = gpd.GeoDataFrame(lscog\_taz\_pop, crs=lscog\_taz.crs)  
lscog\_taz\_pop

ID ... geometry  
0 3010700 ... MULTIPOLYGON (((1699181.31 620454.142, 1699123...  
1 3010701 ... MULTIPOLYGON (((1694583.352 615949.391, 169461...  
2 3010702 ... MULTIPOLYGON (((1700348.674 611719.751, 170010...  
3 3010703 ... MULTIPOLYGON (((1701031.299 609784.953, 170063...  
4 3010704 ... MULTIPOLYGON (((1698257.603 608476.138, 169825...  
.. ... ... ...  
580 75050360 ... MULTIPOLYGON (((1967942.928 639523.392, 196793...  
581 75050361 ... MULTIPOLYGON (((1942618.133 631129.68, 1942312...  
582 75050362 ... MULTIPOLYGON (((1985475.577 652643.249, 198567...  
583 75050366 ... MULTIPOLYGON (((2047618.073 635211.818, 204760...  
584 75050373 ... MULTIPOLYGON (((2053533.596 627788.863, 205352...  
  
[585 rows x 30 columns]

## 6.2 School and college enrollment

The school enrollment combination merges public and private educational institution data into a unified dataset for comprehensive coverage of student populations.

#### R

# Combine school enrollment data  
lscog\_sch\_enroll <- dplyr::bind\_rows(  
 lscog\_pub\_sch\_enroll,  
 lscog\_pvt\_sch\_enroll  
) |>  
 dplyr::mutate(  
 STUDENT\_COUNT = dplyr::coalesce(STUDENT\_COUNT\_PUB, 0) +  
 dplyr::coalesce(STUDENT\_COUNT\_PVT, 0),  
 TEACHER\_COUNT = dplyr::coalesce(TEACHER\_COUNT\_PUB, 0) +  
 dplyr::coalesce(TEACHER\_COUNT\_PVT, 0)  
 )  
  
lscog\_sch\_enroll

Simple feature collection with 122 features and 9 fields  
Geometry type: POINT  
Dimension: XY  
Bounding box: xmin: 1700952 ymin: 406810.6 xmax: 2203406 ymax: 724699.4  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
First 10 features:  
 INSTITUTION\_ID NAME STATE STUDENT\_COUNT\_PUB  
1 450108001163 Barnwell Elementary SC 462  
2 450075000064 Allendale Fairfax High SC 283  
3 450075001184 Allendale Elementary SC 245  
4 450075001415 Allendale-Fairfax Middle SC 268  
5 450093000119 Bamberg-Ehrhardt High SC 381  
6 450093000120 Bamberg-Ehrhardt Middle SC 188  
7 450096000122 Denmark Olar High SC 162  
8 450096000123 Denmark-Olar Middle SC 149  
9 450096001426 Denmark-Olar Elementary SC 353  
10 450098000127 Barnwell County Career Center SC 0  
 TEACHER\_COUNT\_PUB STUDENT\_COUNT\_PVT TEACHER\_COUNT\_PVT  
1 32.0 NA NA  
2 28.9 NA NA  
3 18.0 NA NA  
4 18.0 NA NA  
5 28.5 NA NA  
6 15.0 NA NA  
7 20.0 NA NA  
8 10.0 NA NA  
9 25.0 NA NA  
10 12.0 NA NA  
 geometry STUDENT\_COUNT TEACHER\_COUNT  
1 POINT (1891531 517378.5) 462 32.0  
2 POINT (1913416 420526.6) 283 28.9  
3 POINT (1916344 418212.2) 245 18.0  
4 POINT (1913416 420526.6) 268 18.0  
5 POINT (1991502 535259.1) 381 28.5  
6 POINT (1990622 534442.6) 188 15.0  
7 POINT (1962410 543779) 162 20.0  
8 POINT (1956236 534420.3) 149 10.0  
9 POINT (1960237 537023.1) 353 25.0  
10 POINT (1894891 540093) 0 12.0

#### Python

# Combine school enrollment data  
lscog\_sch\_enroll = pd.concat([  
 lscog\_pub\_sch\_enroll.assign(  
 STUDENT\_COUNT=lscog\_pub\_sch\_enroll['STUDENT\_COUNT\_PUB'],  
 TEACHER\_COUNT=lscog\_pub\_sch\_enroll['TEACHER\_COUNT\_PUB']  
 ),  
 lscog\_pvt\_sch\_enroll.assign(  
 STUDENT\_COUNT=lscog\_pvt\_sch\_enroll['STUDENT\_COUNT\_PVT'],  
 TEACHER\_COUNT=lscog\_pvt\_sch\_enroll['TEACHER\_COUNT\_PVT']  
 )  
])  
  
lscog\_sch\_enroll

INSTITUTION\_ID ... TEACHER\_COUNT\_PVT  
0 450108001163 ... NaN  
1 450075000064 ... NaN  
2 450075001184 ... NaN  
3 450075001415 ... NaN  
4 450093000119 ... NaN  
... ... ... ...  
17663 K9305640 ... 3.0  
17672 A9703170 ... 4.8  
17676 01262826 ... 23.0  
17722 01932407 ... 6.0  
17733 A9903957 ... 1.0  
  
[122 rows x 10 columns]

The subsequent TAZ aggregation counts total student enrollment within each zone, providing essential data for modeling education-related trip patterns and supporting specialized trip generation rates for school-based travel.

#### R

# count the number of school enrollment within each TAZ  
lscog\_taz\_enroll <- lscog\_taz |>  
 sf::st\_join(lscog\_sch\_enroll) |>  
 dplyr::group\_by(  
 ID,  
 Area,  
 TAZ\_ID,  
 COUNTY,  
 AREA\_TYPE,  
 COUNTYID,  
 .drop = FALSE  
 ) |>  
 dplyr::summarize(  
 .groups = "drop",  
 STUDENT\_COUNT = sum(STUDENT\_COUNT, na.rm = TRUE)  
 )  
  
lscog\_taz\_enroll

Simple feature collection with 585 features and 7 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691490 ymin: 331894 xmax: 2237471 ymax: 744679.9  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 585 × 8  
 ID Area TAZ\_ID COUNTY AREA\_TYPE COUNTYID STUDENT\_COUNT  
 <int> <dbl> <int> <chr> <chr> <int> <dbl>  
 1 3010700 0.846 3010700 Aiken SC SUBURBAN 45003 0  
 2 3010701 1.10 3010701 Aiken SC URBAN 45003 0  
 3 3010702 0.395 3010702 Aiken SC URBAN 45003 0  
 4 3010703 0.338 3010703 Aiken SC URBAN 45003 0  
 5 3010704 0.381 3010704 Aiken SC URBAN 45003 0  
 6 3010705 0.125 3010705 Aiken SC URBAN 45003 717  
 7 3010706 0.0879 3010706 Aiken SC URBAN 45003 0  
 8 3010707 0.103 3010707 Aiken SC URBAN 45003 0  
 9 3010708 0.0659 3010708 Aiken SC URBAN 45003 0  
10 3010709 0.0279 3010709 Aiken SC URBAN 45003 0  
# ℹ 575 more rows  
# ℹ 1 more variable: SHAPE <MULTIPOLYGON [foot]>

#### Python

# count the number of school enrollment within each TAZ  
lscog\_taz\_enroll = lscog\_taz.sjoin(lscog\_sch\_enroll, how='left') \  
 .groupby(['ID', 'Area', 'TAZ\_ID', 'COUNTY', 'AREA\_TYPE', 'COUNTYID'], as\_index=False) \  
 .agg({'STUDENT\_COUNT': 'sum'})  
  
lscog\_taz\_enroll

ID Area TAZ\_ID ... AREA\_TYPE COUNTYID STUDENT\_COUNT  
0 3010700 0.846321 3010700 ... SUBURBAN 45003 0.0  
1 3010701 1.098166 3010701 ... URBAN 45003 0.0  
2 3010702 0.395302 3010702 ... URBAN 45003 0.0  
3 3010703 0.338204 3010703 ... URBAN 45003 0.0  
4 3010704 0.381229 3010704 ... URBAN 45003 0.0  
.. ... ... ... ... ... ... ...  
580 75050360 9.593491 75050360 ... RURAL 45075 549.0  
581 75050361 17.433178 75050361 ... RURAL 45075 0.0  
582 75050362 13.319107 75050362 ... RURAL 45075 0.0  
583 75050366 4.323094 75050366 ... SUBURBAN 45075 0.0  
584 75050373 6.635425 75050373 ... RURAL 45075 0.0  
  
[585 rows x 7 columns]

# 7. Combine into a single data

This step integrates all TAZ-level socioeconomic datasets into a comprehensive base year file for travel demand modeling. This consolidated dataset contains all essential variables organized in a standardized format with geographic identifiers, demographic characteristics, employment data, and educational enrollment totals for each traffic analysis zone in the study region.

#### R

# Combine population, household, employments, and school enrollment  
lscog\_se\_base <- lscog\_taz\_pop |>  
 dplyr::left\_join(  
 lscog\_taz\_enroll |> sf::st\_drop\_geometry(),  
 by = dplyr::join\_by(ID, Area, TAZ\_ID, COUNTY, AREA\_TYPE, COUNTYID)  
 ) |>  
 dplyr::select(  
 ID,  
 Area,  
 TAZ\_ID,  
 COUNTY,  
 AREA\_TYPE,  
 COUNTYID,  
 INC\_14999,  
 INC\_49999,  
 INC\_50000,  
 TOTPOP,  
 GQPOP,  
 HHPOP,  
 HH,  
 HH\_1,  
 HH\_2,  
 HH\_3,  
 HH\_4,  
 DU,  
 dplyr::everything()  
 )  
  
lscog\_se\_base

Simple feature collection with 585 features and 30 fields  
Geometry type: MULTIPOLYGON  
Dimension: XY  
Bounding box: xmin: 1691490 ymin: 331894 xmax: 2237471 ymax: 744679.9  
Projected CRS: NAD83(HARN) / South Carolina (ft)  
# A tibble: 585 × 31  
 ID Area TAZ\_ID COUNTY AREA\_TYPE COUNTYID INC\_14999 INC\_49999 INC\_50000  
 <int> <dbl> <int> <chr> <chr> <int> <dbl> <dbl> <dbl>  
 1 3010700 0.846 3.01e6 Aiken… SUBURBAN 45003 0 4 16  
 2 3010701 1.10 3.01e6 Aiken… URBAN 45003 12 85 385  
 3 3010702 0.395 3.01e6 Aiken… URBAN 45003 5 88 175  
 4 3010703 0.338 3.01e6 Aiken… URBAN 45003 7 86 144  
 5 3010704 0.381 3.01e6 Aiken… URBAN 45003 49 48 164  
 6 3010705 0.125 3.01e6 Aiken… URBAN 45003 3 69 117  
 7 3010706 0.0879 3.01e6 Aiken… URBAN 45003 1 23 38  
 8 3010707 0.103 3.01e6 Aiken… URBAN 45003 0 99 95  
 9 3010708 0.0659 3.01e6 Aiken… URBAN 45003 0 35 32  
10 3010709 0.0279 3.01e6 Aiken… URBAN 45003 0 36 35  
# ℹ 575 more rows  
# ℹ 22 more variables: TOTPOP <dbl>, GQPOP <dbl>, HHPOP <dbl>, HH <dbl>,  
# HH\_1 <dbl>, HH\_2 <dbl>, HH\_3 <dbl>, HH\_4 <dbl>, DU <dbl>, TOTAL\_EMP <dbl>,  
# AGR\_FOR\_FI <dbl>, MINING <dbl>, CONSTRUCTI <dbl>, MANUFACTUR <dbl>,  
# TRANSP\_COM <dbl>, WHOLESALE <dbl>, RETAIL <dbl>, FIRE <dbl>,  
# SERVICES <dbl>, PUBLIC\_ADM <dbl>, SHAPE <MULTIPOLYGON [foot]>,  
# STUDENT\_COUNT <dbl>

#### Python

# Define the column order  
field\_order = [  
 'ID', 'Area', 'TAZ\_ID', 'COUNTY', 'AREA\_TYPE', 'COUNTYID',  
 'INC\_14999', 'INC\_49999', 'INC\_50000',  
 'TOTPOP', 'GQPOP', 'HHPOP',  
 'HH', 'HH\_1', 'HH\_2', 'HH\_3', 'HH\_4', 'DU'  
]  
  
# Combine population, household, employments, and school enrollment  
lscog\_se\_base = lscog\_taz\_pop.merge(  
 lscog\_taz\_enroll,  
 on=['ID', 'Area', 'TAZ\_ID', 'COUNTY', 'AREA\_TYPE', 'COUNTYID'],  
 how='left'  
)  
  
lscog\_se\_base = lscog\_se\_base[field\_order +  
 list(lscog\_se\_base.columns.difference(field\_order))]  
  
lscog\_se\_base

ID ... geometry  
0 3010700 ... MULTIPOLYGON (((1699181.31 620454.142, 1699123...  
1 3010701 ... MULTIPOLYGON (((1694583.352 615949.391, 169461...  
2 3010702 ... MULTIPOLYGON (((1700348.674 611719.751, 170010...  
3 3010703 ... MULTIPOLYGON (((1701031.299 609784.953, 170063...  
4 3010704 ... MULTIPOLYGON (((1698257.603 608476.138, 169825...  
.. ... ... ...  
580 75050360 ... MULTIPOLYGON (((1967942.928 639523.392, 196793...  
581 75050361 ... MULTIPOLYGON (((1942618.133 631129.68, 1942312...  
582 75050362 ... MULTIPOLYGON (((1985475.577 652643.249, 198567...  
583 75050366 ... MULTIPOLYGON (((2047618.073 635211.818, 204760...  
584 75050373 ... MULTIPOLYGON (((2053533.596 627788.863, 205352...  
  
[585 rows x 31 columns]

# 8. Data checks and validation

The validation process ensures data integrity and consistency across all socioeconomic variables through systematic checks of categorical totals. These validation steps identify any discrepancies between aggregate totals and component categories that may have occurred during the interpolation or aggregation processes.

## 8.1 Household size total

This validation confirms that the total household count matches the sum of all household size categories for each TAZ. Any discrepancies indicate potential issues in the household size distribution that require investigation and correction before model implementation.

#### R

# check the sum of household by household size  
lscog\_se\_base |>  
 dplyr::filter(HH != (HH\_1 + HH\_2 + HH\_3 + HH\_4)) |>  
 nrow()

[1] 0

#### Python

# check the sum of household by household size  
lscog\_se\_base[  
 lscog\_se\_base['HH'] != (lscog\_se\_base['HH\_1'] + lscog\_se\_base['HH\_2'] +  
 lscog\_se\_base['HH\_3'] + lscog\_se\_base['HH\_4'])  
].shape[0]

0

## 8.2 Household income total

The income category validation verifies that household totals equal the sum of all three income brackets across all zones. This check ensures the integrity of the income distribution data following the proportional allocation methodology applied during the ACS interpolation process.

#### R

# check the sum of household by income level  
lscog\_se\_base |>  
 dplyr::filter(HH != (INC\_14999 + INC\_49999 + INC\_50000)) |>  
 nrow()

[1] 0

#### Python

# check the sum of household by income level  
lscog\_se\_base[  
 lscog\_se\_base['HH'] != (lscog\_se\_base['INC\_14999'] + lscog\_se\_base['INC\_49999'] +  
 lscog\_se\_base['INC\_50000'])  
].shape[0]

0

## 8.3 Employment categories

The employment validation confirms that total employment equals the sum of all industry sector categories for each TAZ. This comprehensive check validates the LEHD data integration and ensures that no employment is lost or double-counted during the sectoral disaggregation process.RetryClaude can make mistakes. Please double-check responses.

#### R

# check the sum of employment by categories  
lscog\_se\_base |>  
 dplyr::filter(  
 TOTAL\_EMP !=  
 (AGR\_FOR\_FI +  
 MINING +  
 CONSTRUCTI +  
 MANUFACTUR +  
 TRANSP\_COM +  
 WHOLESALE +  
 RETAIL +  
 FIRE +  
 SERVICES +  
 PUBLIC\_ADM)  
 ) |>  
 nrow()

[1] 0

#### Python

# check the sum of employment by categories  
lscog\_se\_base[  
 lscog\_se\_base['TOTAL\_EMP'] != (  
 lscog\_se\_base['AGR\_FOR\_FI'] +  
 lscog\_se\_base['MINING'] +  
 lscog\_se\_base['CONSTRUCTI'] +  
 lscog\_se\_base['MANUFACTUR'] +  
 lscog\_se\_base['TRANSP\_COM'] +  
 lscog\_se\_base['WHOLESALE'] +  
 lscog\_se\_base['RETAIL'] +  
 lscog\_se\_base['FIRE'] +  
 lscog\_se\_base['SERVICES'] +  
 lscog\_se\_base['PUBLIC\_ADM']  
 )  
].shape[0]

0

# 9. Export the final data

The final export creates the complete TAZ-level socioeconomic dataset in both tabular and spatial formats for direct integration into the travel demand model. This comprehensive dataset serves as the primary input for trip generation, providing all necessary demographic, economic, and educational variables organized by traffic analysis zone for the LSCOG regional modeling system.

**Export as CSV flat file**

#### R

# Export as CSV  
lscog\_se\_base |>  
 sf::st\_drop\_geometry() |>  
 readr::write\_csv(  
 file.path(root, "Task 1 TDM Development/Base Year/\_raw/lscog\_se\_taz.csv"),  
 append = FALSE  
 )

#### Python

# Export as CSV  
lscog\_se\_base |>  
 gpd.GeoDataFrame.to\_csv(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_raw" / "lscog\_se\_taz.csv",  
 index=False  
 )

**Export as Geodatabase layer**

#### R

# Export as GDB  
lscog\_se\_base |>  
 sf::write\_sf(  
 file.path(  
 root,  
 "Task 1 TDM Development/Base Year/\_gis/LSCOG\_2020Base\_SE.gdb"  
 ),  
 layer = "lscog\_se\_base",  
 append = FALSE  
 )

#### Python

# Export as GDB  
lscog\_se\_base |>  
 gpd.GeoDataFrame.to\_file(  
 Path(root) / "Task 1 TDM Development" / "Base Year" / "\_gis" / "LSCOG\_2020Base\_SE.gdb"  
 layer='lscog\_se\_base',  
 driver='FileGDB'  
 )