

Base Year SE Data Development

Lower Savannah Council of Governments (LSCOG) 2050 Long Range Transportation Plan

Pukar Bhandari
Metro Analytics
pbhandari@metroanalytics.com

2025-07-05

Table of contents

1	Set up the environment	2
1.1	Install and load packages	2
1.2	Set global options and parameters	3
1.3	Set census API key	4
1.4	Project folder	4
2	Define study area	4
2.1	Define state and counties	5
2.2	Load TAZ geometry	6
3	Fetch raw data	9
3.1	2020 Decennial census	9
3.2	2020 ACS estimates	12
3.3	2019 LEHD data	16
3.4	2020 NCES school and college enrollment data	20
	Public schools	20
	Private schools	23
	Post-secondary institutions	26
4	Clean data	29
4.1	Household-weighted interpolation	29
4.2	Combine population and households	36
4.3	Employment data	39
5	Export raw data	40
5.1	TAZ data	40
5.2	Census blocks to TAZ conversion	41
5.3	Decennial census data at census block level	42
5.4	ACS estimates at census block level	43
5.5	LEHD data at census block level	44
5.6	Public schools to TAZ	46
5.7	Private schools to TAZ	47

6	Aggregate data to TAZ	48
6.1	Population, households, and employment	48
6.2	School and college enrollment	52
7	Combine into a single data	55
8	Data checks and validation	58
8.1	Household size total	58
8.2	Household income total	58
8.3	Employment categories	59
9	Export the final data	60

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 `{pygrids}` for Census data queries. These packages form the analytical backbone for processing demographic, employment, and geographic data required for travel demand modeling.

1.1.-a 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
```

1.1.-b 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

1.2.-a R

```
# Set options
options(tigris_use_cache = TRUE) # cache tiger/line shapefile for future use

# set project CRS
project_crs <- "EPSG:3361"
```

1.2.-.b 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

1.3.-.a R

```
# Set your API key into environment
tidycensus::census_api_key("your_api_key_here", install = TRUE)
```

1.3.-.b 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.

1.4.-.a R

```
# Set your main data folder
root <- "M:/MA_Project/SC_LSCOG LRTP"
```

1.4.-.b 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.

2.1.-a R

```
# Define state abbreviation and county names
state_abb <- "SC"
county_names <- c(
  "Aiken",
  "Allendale",
  "Bamberg",
  "Barnwell",
  "Calhoun",
  "Orangeburg"
)
```

2.1.-b 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.

2.1.-c 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"
```

2.1.-d 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.

2.2.-a R

```
# Load TAZ Shapefile
lscog_taz <- sf::read_sf(
```

```

"data/SE_2019_AD_10_30_2023.gpkg",
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
  <dbl> <dbl> <dbl> <chr>   <chr>    <chr>  <chr>
1  9050... 13.3   8518. 90501... RURAL    Bambe... 45009  (((2046194 478862.1,
204...
2  9050... 39.2   25084. 90501... RURAL    Bambe... 45009  (((2012593 500179.5,
201...
3  7505... 9.03   5778. 75050... RURAL    Orang... 45075  (((2056266 515976,
20561...
4  7505... 15.5   9934. 75050... RURAL    Orang... 45075  (((2061827 488917.9,
206...
5  7505... 17.5   11216. 75050... SUBURBAN Orang... 45075  (((2154222 534929.2,
215...
6  7505... 12.8   8191. 75050... RURAL    Orang... 45075  (((2195053 518745.9,
219...
7  7505... 8.10   5181. 75050... SUBURBAN Orang... 45075  (((2179620 542034.9,
217...
8  7505... 13.4   8568. 75050... SUBURBAN Orang... 45075  (((2179131 542424,
21770...
9  7505... 6.00   3843. 75050... SUBURBAN Orang... 45075  (((2184440 568297.6,

```

```

218...
10 7505... 5.06 3239. 75050... SUBURBAN Orang... 45075 (((2204464 570787.3,
220...
# i 575 more rows

```

2.2.-b Python

```

# Load TAZ Shapefile
lscog_taz = gpd.read_file(
    "data/SE_2019_AD_10_30_2023.gpkg",
    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.

2.2.-c R


```
# Create interactive map
mapview::mapview(lscog_taz, zcol = "COUNTY", lwd = 1.6, map.types =
"CartoDB.Voyager", col.regions = RColorBrewer::brewer.pal(6, "Dark2"))
```

2.2.-.d Python

```
# Create interactive map
lscog_taz.explore(column="COUNTY", categorical=True, legend=True,
tiles="CartoDB.Voyager", 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.

3.1.-.a 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]>

3.1.-b 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[['GE0ID20', 'geometry']].merge(lscog_dec, left_on =
"GE0ID20", right_on = "GE0ID")

# 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[['GE0ID', 'TOTPOP', 'GQPOP', 'HHPOP',
                      'HH', 'HH_1', 'HH_2', 'HH_3', 'HH_4', 'DU', 'geometry']]

lscog_dec

```

```

      GE0ID  ...                               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.

3.2.-a 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...

3.2.-b 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[['GE0ID', 'geometry']].merge(lscog_acs, on = "GE0ID")

# 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.

3.3.-a 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>
```

3.3.-b 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.

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.

3.4.-a 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)

3.4.-b 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/ipeds/data/colleges/universities/area5082019/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.

3.4.-c 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]>

```

3.4.-d 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']]
lscog_pvt_sch_enroll.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.

3.4.-e 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)

3.4.-f 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.

4.1.-a 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				

4.1.-b 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.

4.1.-c 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)
```

4.1.-d 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.

4.2.-a 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>
```

4.2.-b 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.

4.2.-c 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>

```

4.2.-d 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.

4.3.-a 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>
```

4.3.-b 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

5.1.-a 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
  )
```


5.1.-b 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

5.1.-c 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
  )
```

5.1.-d 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

5.2.-a R

```
# Export as CSV
lscog_cb |>
  sf::st_join(lscog_taz) |>
  dplyr::select(GE0ID20, 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
)

```

5.2.-b Python

```

# Export as CSV
lscog_cb.merge(lscog_taz[['ID', 'TAZ_ID']], left_on='GE0ID20', right_on='ID')
[['GE0ID20', '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

5.3.-a 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
  )

```

5.3.-b 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

5.3.-c 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
  )
```

5.3.-d 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

5.4.-a 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
  )
```

5.4.-b Python

```
# Export as CSV
lscog_acs_cb[['GE0ID', '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

5.4.-c R

```
# Export as GDB
lscog_acs_cb |>
  dplyr::select(GE0ID, 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
  )
```

5.4.-d Python

```
# Export as GDB
lscog_acs_cb[['GE0ID', '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

5.5.-a R

```
# Export as CSV
lscog_pop_hh_emp |>
  dplyr::select(GE0ID, 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
)

```

5.5.-b 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

5.5.-c 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
)

```

5.5.-d 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

5.6.-a 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
  )
```

5.6.-b 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

5.6.-c 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
)

```

5.6.-.d 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

5.7.-.a 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
  )

```

5.7.-.b 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

5.7.-c 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
  )
```

5.7.-d 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.

6.1.-a 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
  <chr>   <dbl> <chr>  <chr>  <chr>      <chr>      <dbl> <dbl> <dbl>  <dbl>
<dbl>
1 11050... 37.7 11050... Barnw... RURAL      45011      802    0    802   329
95
2 11050... 21.5 11050... Barnw... RURAL      45011      715    0    715   320
118
3 11050... 24.4 11050... Barnw... RURAL      45011     1406   79   1327   555
194
4 11050... 17.4 11050... Barnw... RURAL      45011      712    0    712   309
109
5 11050... 12.6 11050... Barnw... RURAL      45011      941    0    941   395
99
6 11050... 191. 11050... Barnw... RURAL      45011        7    0        7    8
2
7 11050...  9.69 11050... Barnw... SUBURBAN 45011      185    0    185   69
15
8 11050... 16.5 11050... Barnw... RURAL      45011      680    0    680   274
87
9 11050... 11.4 11050... Barnw... SUBURBAN 45011      490    0    490   221
69
10 11050...  9.02 11050... Barnw... SUBURBAN 45011      711    0    711   276
76
# 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>, geom <MULTIPOLYGON [foot]>

```

6.1.-b 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	11050058	...	MULTIPOLYGON (((1940883.78 467711.359, 1940683...
1	11050059	...	MULTIPOLYGON (((1909744.245 514791.296, 190990...
2	11050060	...	MULTIPOLYGON (((1933978.652 551400.913, 193349...
3	11050061	...	MULTIPOLYGON (((1901544.254 537296.043, 190161...
4	11050062	...	MULTIPOLYGON (((1932677.078 512746.246, 193263...
..
580	9050125	...	MULTIPOLYGON (((1968427.616 539621.663, 196834...
581	9050126	...	MULTIPOLYGON (((1961524.044 543029.131, 196150...
582	9050128	...	MULTIPOLYGON (((1994408.254 547077.578, 199440...
583	9050130	...	MULTIPOLYGON (((2046194.08 478862.054, 2046134...
584	9050132	...	MULTIPOLYGON (((2012593.472 500179.47, 2013190...

[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.

6.2.-a 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

6.2.-b 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.

6.2.-c 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
  <chr>   <dbl> <chr>   <chr>   <chr>   <chr>      <dbl>
1 11050058 37.7 11050058 Barnwell SC RURAL    45011      0
2 11050059 21.5 11050059 Barnwell SC RURAL    45011      0
3 11050060 24.4 11050060 Barnwell SC RURAL    45011    598
4 11050061 17.4 11050061 Barnwell SC RURAL    45011      3
5 11050062 12.6 11050062 Barnwell SC RURAL    45011      2
6 11050072 191. 11050072 Barnwell SC RURAL    45011      0
7 11050073 9.69 11050073 Barnwell SC SUBURBAN 45011      0
8 11050074 16.5 11050074 Barnwell SC RURAL    45011      0
9 11050075 11.4 11050075 Barnwell SC SUBURBAN 45011      0
10 11050076 9.02 11050076 Barnwell SC SUBURBAN 45011      0
# i 575 more rows
# i 1 more variable: geom <MULTIPOLYGON [foot]>
```

6.2.-d 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	11050058	37.707611	11050058	...	RURAL	45011	0.0
1	11050059	21.450575	11050059	...	RURAL	45011	0.0
2	11050060	24.411972	11050060	...	RURAL	45011	598.0
3	11050061	17.351381	11050061	...	RURAL	45011	3.0
4	11050062	12.587147	11050062	...	RURAL	45011	2.0
...
580	9050125	12.789308	9050125	...	RURAL	45009	162.0
581	9050126	0.633810	9050126	...	SUBURBAN	45009	0.0
582	9050128	12.890918	9050128	...	RURAL	45009	693.0
583	9050130	13.308991	9050130	...	RURAL	45009	0.0
584	9050132	39.194309	9050132	...	RURAL	45009	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.

7.0.-.a 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								
	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>	<dbl>
<dbl>								
1	110500...	37.7	11050...	Barnw...	RURAL	45011	71	198
60								
2	110500...	21.5	11050...	Barnw...	RURAL	45011	32	137
151								
3	110500...	24.4	11050...	Barnw...	RURAL	45011	149	179
227								
4	110500...	17.4	11050...	Barnw...	RURAL	45011	81	111
117								
5	110500...	12.6	11050...	Barnw...	RURAL	45011	44	132
219								
6	110500...	191.	11050...	Barnw...	RURAL	45011	8	0
0								
7	110500...	9.69	11050...	Barnw...	SUBURBAN	45011	23	25
21								
8	110500...	16.5	11050...	Barnw...	RURAL	45011	25	92
157								
9	110500...	11.4	11050...	Barnw...	SUBURBAN	45011	76	80
65								
10	110500...	9.02	11050...	Barnw...	SUBURBAN	45011	12	77
187								
# 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>, geom <MULTIPOLYGON [foot]>,
# STUDENT_COUNT <dbl>
```

7.0.-b 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  11050058  ...  MULTIPOLYGON (((1940883.78 467711.359, 1940683...
1  11050059  ...  MULTIPOLYGON (((1909744.245 514791.296, 190990...
2  11050060  ...  MULTIPOLYGON (((1933978.652 551400.913, 193349...
3  11050061  ...  MULTIPOLYGON (((1901544.254 537296.043, 190161...
4  11050062  ...  MULTIPOLYGON (((1932677.078 512746.246, 193263...
..      ...      ...
580  9050125  ...  MULTIPOLYGON (((1968427.616 539621.663, 196834...
581  9050126  ...  MULTIPOLYGON (((1961524.044 543029.131, 196150...
582  9050128  ...  MULTIPOLYGON (((1994408.254 547077.578, 199440...
583  9050130  ...  MULTIPOLYGON (((2046194.08 478862.054, 2046134...
584  9050132  ...  MULTIPOLYGON (((2012593.472 500179.47, 2013190...

[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.

8.1.-a 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
```

8.1.-b 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.

8.2.-a R

```
# check the sum of household by income level
lscog_se_base |>
  dplyr::filter(HH != (INC_14999 + INC_49999 + INC_50000)) |>
  nrow()
```

```
[1] 0
```

8.2.-b 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.

8.3.-a 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
```

8.3.-b 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

9.0.-a 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
  )

```

9.0.-b 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

9.0.-c 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
)

```

9.0.-d 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'
  )

```