# Geospatial Data Engineering Associate

**What you will learn** This notebook will teach you to:

- Set up your environment and create a **WherobotsDB context**
- Load vector and raster datasets directly from **AWS S3** into **Apache Sedona DataFrames**
- Inspect and validate geometries for quality and consistency
- Standardize and transform **Coordinate Reference Systems (CRS)**
- Apply the **Bronze → Silver → Gold** data architecture pattern for spatial workflows
- Save and manage your first **Iceberg table** to prepare for scalable analysis

In [1]:
```
from sedona.spark import *
from pyspark.sql.functions import col, when, expr
from sedona.sql.st_functions import ST_IsValid, ST_IsValidReason, ST_MakeVal
from pyspark.sql import DataFrame
import urllib.request
import json
```

```
/tmp/ipykernel_436/3932548362.py:3: DeprecationWarning: Importing from 'sedon
  from sedona.sql.st_functions import ST_IsValid, ST_IsValidReason, ST_MakeVa
/tmp/ipykernel_436/3932548362.py:3: DeprecationWarning: Importing from 'sedon
  from sedona.sql.st_functions import ST_IsValid, ST_IsValidReason, ST_MakeVa
```

In [2]:
```
config = SedonaContext.builder().getOrCreate()
sedona = SedonaContext.create(config)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogL
Setting Spark log level to "WARN".
```

In [3]:
```
geoparquet_path = "s3://wherobots-examples/data/nyc_buildings.parquet"

df = sedona.read.format("geoparquet").load(geoparquet_path)
df.show(5)
```

```
[Stage 4:>                                                          (0 + 1) /
+--------+----------+--------------------+-------+---------+---------+-------
|BUILD_ID|   OCC_CLS|            PRIM_OCC|SEC_OCC|PROP_ADDR|PROP_CITY| PROP_S
+--------+----------+--------------------+-------+---------+---------+-------
| 2080161|Commercial|Entertainment and...|   NULL|     NULL|     NULL|New Yor
| 2080159|Commercial|Entertainment and...|   NULL|     NULL|     NULL|New Yor
| 2080158|Commercial|Entertainment and...|   NULL|     NULL|     NULL|New Yor
| 7776189|Commercial|Entertainment and...|   NULL|     NULL|     NULL|New Yor
| 7776214|Commercial|Entertainment and...|   NULL|     NULL|     NULL|New Yor
+--------+----------+--------------------+-------+---------+---------+-------
only showing top 5 rows
```

In [4]:
```
geojson_multi_path = "s3://wherobots-examples/data/noaa/storms/"

df = sedona.read.format("geojson")\
        .option("multiLine", "true")\
        .load(geojson_multi_path)\
        .selectExpr("explode(features) as features")\
        .select("features.*")

df.show(5)
```

```
[Stage 6:>                                                          (0 + 1) /
+--------------------+---+--------------------+-------+
|            geometry| id|          properties|   type|
+--------------------+---+--------------------+-------+
|POINT (-166.54001...|  1|{Snow, 1713549894...|Feature|
|POINT (-74.860001...|  2|{Rain, 1713549894...|Feature|
|POINT (-101.50000...|  3|{Non-Tstm Wnd Gst...|Feature|
|POINT (-101.93000...|  4|{Non-Tstm Wnd Gst...|Feature|
|POINT (-84.630005...|  5|{Non-Tstm Wnd Gst...|Feature|
+--------------------+---+--------------------+-------+
only showing top 5 rows
```

In [5]:
```
shp_path = "s3://wherobots-examples/data/austin_boundaries/"

df = sedona.read.format("shapefile").load(shp_path)
df.show(5)
```

```
[Stage 8:>                                                          (0 + 1) /
+--------------------+---------+--------------+-------+-------+--------------
|            geometry|zcta5ce10|    affgeoid10|geoid10|aland10|            aw
+--------------------+---------+--------------+-------+-------+--------------
|MULTIPOLYGON (((-...|    78612|8600000US78612|  78612|   NULL|
|POLYGON ((-97.844...|    78745|8600000US78745|  78745|   NULL|0.000000000000
|POLYGON ((-97.737...|    78751|8600000US78751|  78751|   NULL|0.000000000000
|POLYGON ((-97.737...|    78702|8600000US78702|  78702|   NULL|
|POLYGON ((-97.749...|    78741|8600000US78741|  78741|   NULL|
+--------------------+---------+--------------+-------+-------+--------------
only showing top 5 rows
```

In [6]:
```
df = sedona.read.format("shapefile") \
    .option("recursiveFileLookup", "true") \
    .load("s3://wherobots-examples/data/examples/Global_Landslide_Catalog/")

df.show(5)
```

```
[Stage 10:>                                                         (0 + 1) /
```

```
+--------------------+---+--------+--------------------+--------------------+
|            geometry|FID|OBJECTID|          source_nam|          source_lin|
+--------------------+---+--------+--------------------+--------------------+
|POINT (91.6157721...|  1| 1537380|  The Times of India|https://timesofin...|
|POINT (98.5310531...|  2| 1538980|NASA SERVIR Scien...|https://servir.ad...|
|POINT (7.12972136...|  3| 1537381|         Le Parisien|http://www.lepari...|
|POINT (170.551639...|  4| 1537382|   Otago Daily Times|https://www.odt.c...|
|POINT (75.9434000...|  5| 1537383|  The Times Of India|https://timesofin...|
+--------------------+---+--------+--------------------+--------------------+
only showing top 5 rows
```

In [7]:
```
geotiff_path = "s3://wherobots-examples/data/ghs_population/GHS_POP_E1975_GL

df = sedona.read.format("raster").load(geotiff_path)
df.show(5)
```

```
25/10/23 10:31:17 WARN RasterScan: Loading one large unsplittable file s3://w
[Stage 11:>                                                         (0 + 1) /
+--------------------+----+---+--------------------+
|                rast|   x|  y|                name|
+--------------------+----+---+--------------------+
|OutDbGridCoverage...| 654| 24|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|1576| 56|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|1286|254|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|1567|116|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...| 624|791|GHS_POP_E1975_GLO...|
+--------------------+----+---+--------------------+
only showing top 5 rows
```

In [8]:
```
df = sedona.read.format("raster") \
    .option("tileWidth", "512") \
    .option("tileHeight", "512") \
    .option("retile", "true") \
    .load(geotiff_path)

df.show(5)
```

```
25/10/23 10:31:44 WARN RasterScan: Loading one large unsplittable file s3://w
[Stage 14:>                                                         (0 + 1) /
+--------------------+---+---+--------------------+
|                rast|  x|  y|                name|
+--------------------+---+---+--------------------+
|OutDbGridCoverage...|274| 90|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|471|  8|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|476|403|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|339|353|GHS_POP_E1975_GLO...|
|OutDbGridCoverage...|523| 88|GHS_POP_E1975_GLO...|
+--------------------+---+---+--------------------+
only showing top 5 rows
```

In [9]:
```
stac_df = sedona.read.format("stac").load(
    "https://earth-search.aws.element84.com/v1/collections/sentinel-2-pre-c1
)

stac_df.printSchema()
stac_df.select("id", "datetime", "geometry", "collection").show(5, truncate=
```

```
root
 |-- stac_version: string (nullable = false)
 |-- stac_extensions: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- type: string (nullable = false)
 |-- id: string (nullable = false)
 |-- bbox: array (nullable = true)
 |    |-- element: double (containsNull = true)
 |-- geometry: geometry (nullable = true)
 |-- title: string (nullable = true)
 |-- description: string (nullable = true)
 |-- datetime: timestamp (nullable = true)
 |-- start_datetime: timestamp (nullable = true)
 |-- end_datetime: timestamp (nullable = true)
 |-- created: timestamp (nullable = true)
 |-- updated: timestamp (nullable = true)
 |-- platform: string (nullable = true)
 |-- instruments: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- constellation: string (nullable = true)
 |-- mission: string (nullable = true)
 |-- gsd: double (nullable = true)
 |-- grid:code: string (nullable = true)
 |-- eo:cloud_cover: double (nullable = true)
 |-- eo:snow_cover: double (nullable = true)
 |-- collection: string (nullable = true)
 |-- links: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- rel: string (nullable = true)
 |    |    |-- href: string (nullable = true)
 |    |    |-- type: string (nullable = true)
 |    |    |-- title: string (nullable = true)
 |-- assets: map (nullable = true)
 |    |-- key: string
 |    |-- value: struct (valueContainsNull = true)
 |    |    |-- href: string (nullable = true)
 |    |    |-- type: string (nullable = true)
 |    |    |-- title: string (nullable = true)
 |    |    |-- roles: array (nullable = true)
 |    |    |    |-- element: string (containsNull = true)
 |    |    |-- rast: raster (nullable = true)
```

```
+----------------------------+----------------------+--------------------
|id                          |datetime              |geometry
+----------------------------+----------------------+--------------------
|S2B_T21NYC_20221205T140704_L2A|2022-12-05 14:11:53.454|POLYGON ((-55.2012945
|S2B_T21NZC_20221205T140704_L2A|2022-12-05 14:11:50.088|POLYGON ((-54.3021455
|S2B_T22NBH_20221205T140704_L2A|2022-12-05 14:11:40.896|POLYGON ((-53.6981959
|S2B_T21NYD_20221205T140704_L2A|2022-12-05 14:11:38.955|POLYGON ((-55.1997337
|S2B_T21NZD_20221205T140704_L2A|2022-12-05 14:11:35.344|POLYGON ((-54.2998064
+----------------------------+----------------------+--------------------
only showing top 5 rows
```

In [10]:
```
df = sedona.read.format("stac") \
            .option("itemsLimitMax", "1000")\
```

```
            .option("itemsLimitPerRequest", "50")\
            .option("itemsLoadProcessReportThreshold", "500000")\
            .load("https://earth-search.aws.element84.com/v1/collections/ser
```

In [11]:
```
points_df = sedona.sql("""

SELECT ST_MakePoint(-122.349277, 47.620504) as space_needle, ST_MakePoint(-1

""")

points_df.show(1, False)
```

```
+---------------------------+---------------------------+---------------
|space_needle               |glass_museum               |pop_culture_muse
+---------------------------+---------------------------+---------------
|POINT (-122.349277 47.620504)|POINT (-122.350446 47.620556)|POINT (-122.3482
+---------------------------+---------------------------+---------------
```

In [12]:
```
map_config_url = "https://raw.githubusercontent.com/wherobots/geospatial-dat

with urllib.request.urlopen(map_config_url) as response:
    map_config = json.load(response)

map = SedonaKepler.create_map(points_df, "Tourist spots", map_config)
map
```

Out[12]:
```
KeplerGl(config={'version': 'v1', 'config': {'visState': {'layers': [{'id':
```

In [13]:
```
line_df = sedona.sql("""

SELECT ST_LineStringFromText('-122.349277, 47.620504, -122.350446, 47.620556

""")
line_df.show(1, False)
```

```
+-----------------------------------------------------------------------------
|order_to_visit
+-----------------------------------------------------------------------------
|LINESTRING (-122.349277 47.620504, -122.350446 47.620556, -122.348258 47.621
+-----------------------------------------------------------------------------
```

In [14]:
```
# PS. this allows us to access the dataframes in a SQL environment
points_df.createOrReplaceTempView("points")
line_df.createOrReplaceTempView("line")

collection_df = sedona.sql("""

SELECT
    ST_Collect(Array(space_needle, glass_museum, pop_culture_museum, order_t
FROM points, line

""")

collection_df.show(1, False)
```

```
+--------------------------------------------------------------------------------
|st_collect(array(space_needle, glass_museum, pop_culture_museum, order_to_vi
+--------------------------------------------------------------------------------
|GEOMETRYCOLLECTION (POINT (-122.349277 47.620504), POINT (-122.350446 47.620
+--------------------------------------------------------------------------------
```

In [15]:
```
map_collection = SedonaKepler.create_map(collection_df, "Things to do", map_
map_collection
```

Out[15]:
```
KeplerGl(config={'version': 'v1', 'config': {'visState': {'layers': [{'id':
```

In [16]:
```
envelope_df = sedona.sql("""

SELECT ST_MakeEnvelope(-122.352848,47.619674,-122.346539,47.622451) AS touri

""")

envelope_df.show(1, False)
```

```
+--------------------------------------------------------------------------------
|tourist_location_bbox
+--------------------------------------------------------------------------------
|POLYGON ((-122.352848 47.619674, -122.352848 47.622451, -122.346539 47.62245
+--------------------------------------------------------------------------------
```

In [17]:
```
places_df = (
    sedona.sql("""

    WITH seattle_downtown AS (
        SELECT ST_GeomFromWKT('POLYGON ((-122.360916 47.590189, -122.299461
    ),

    -- This is to leverage spatial predicate pushdown
    places AS (
      SELECT *
      FROM
        wherobots_open_data.overture_maps_foundation.places_place places, se
      WHERE
        ST_Intersects(places.geometry, seattle_downtown.geom)
    ),

    -- This is to leverage spatial predicate pushdown
    buildings AS (
      SELECT *
      FROM
        wherobots_open_data.overture_maps_foundation.buildings_building buil
      WHERE
        ST_Intersects(buildings.geometry, seattle_downtown.geom)
    )

    SELECT
      places.names.primary as place_name,
      places.categories.primary as place_type,
      element_at(places.addresses, 1) as place_address,
```

```
      ROUND(places.confidence * 100, 2) AS `place_confidence (%)`,
      places.geometry as place_geometry,
      buildings.geometry as building_geometry
    FROM
      places
    JOIN
      buildings
    ON
      ST_Intersects(places.geometry, buildings.geometry);

    """)
    # We are caching the result, as we will reuse it to visualize the data
        .cache()
)

places_df.show(10, False)
```

```
[Stage 27:=====================================>                   (3 + 2) /
+--------------------------+------------------------------+------------------
|place_name                |place_type                    |place_address
+--------------------------+------------------------------+------------------
|Epic Seats                |arts_and_entertainment        |{900 1st Ave S, Se
|Square Tomato             |advertising_agency            |{900 1st Ave S, Se
|Seattle Gold Grills       |jewelry_store                 |{901 Occidental Av
|April Lane's Home Cleaning|home_cleaning                 |{900 1st Ave S, Se
|Y-Designs                 |graphic_designer              |{900 1st Ave S, Se
|3 Point Productions       |event_planning                |{3411 Thorndyke Av
|Max Technologies          |information_technology_company|{900 1st Ave S, Se
|OnTheField                |sports_wear                   |{901 Occidental Av
|Pioneer Grill Hot Dogs    |american_restaurant           |{820 Occidental Av
|Helix Sport and Spine     |chiropractor                  |{902 1st Ave S, Se
+--------------------------+------------------------------+------------------
only showing top 10 rows
```

In [18]:
```
map_config_join_url = "https://raw.githubusercontent.com/wherobots/geospatia

with urllib.request.urlopen(map_config_join_url) as response:
    map_join_config = json.load(response)

map_places = SedonaKepler.create_map(places_df, "Places in buildings", map_j
map_places
```

Out[18]:
```
KeplerGl(config={'version': 'v1', 'config': {'visState': {'filters': [], 'la
```

In [19]:
```
# Create a new Havasu (Iceberg) database

database = 'gde_bronze'

sedona.sql(f'CREATE DATABASE IF NOT EXISTS org_catalog.{database}')
```

Out[19]:
```
DataFrame[]
```

In [27]:
```
geotiff_path = "s3://wherobots-examples/data/ghs_population/GHS_POP_E1975_GL

df = sedona.read.format("raster") \
    .option("tileWidth", "512") \
    .option("tileHeight", "512") \
    .option("retile", "true") \
    .load(geotiff_path)

df.writeTo(f"org_catalog.{database}.ghs_population_tiles").createOrReplace()
```

```
25/10/23 10:38:22 WARN RasterScan: Loading one large unsplittable file s3://w
```

In [23]:
```
prefix = 's3://wherobots-examples/gdea-course-data/raw-data/'

def check_invalid_geometries(df: DataFrame, geom_col: str = "geom", reason_c
    df_with_reason = df.withColumn(reason_col, ST_IsValidReason(col(geom_col
    # cache to avoid recomputation if you inspect reasons later
    df_with_reason.cache()
    invalid_count = df_with_reason.filter(~ST_IsValid(col(geom_col))).count(
    print(f"✅ Checked geometries — found {invalid_count} invalid geometries
    return invalid_count

def fix_invalid_geometries(df: DataFrame, invalid_count: int, geom_col: str
    if invalid_count > 1:
        print(f"🔧 Attempting to fix {invalid_count} invalid geometries...")
        return df.withColumn(
            geom_col,
            when(~ST_IsValid(col(geom_col)), ST_MakeValid(col(geom_col))).ot
        )
    else:
        print("⚡ Only one invalid geometry (or none). Skipping automated fix
        return df

# --- driver program ---
def process_geometries(
    df: DataFrame,
    geom_col: str = "geom",
    attempt_fix: bool = True,
    split_on_fail: bool = True
):
    """
    Runs validity check -> optional repair -> optional split.
    Returns either:
      - {"df": corrected_df}  when all geometries valid after repair (or nor
      - {"valid_df": ..., "invalid_df": ...} when some invalid remain and sp
    """
    # 1) Initial check
    invalid_count = check_invalid_geometries(df, geom_col=geom_col)

    if invalid_count == 0:
        print("✅ All geometries are valid.")
        return {"df": df}  # nothing to do

    # 2) Attempt repair (only changes rows that are invalid per your earlier
```

```
    if attempt_fix:
        df_fixed = fix_invalid_geometries(df, invalid_count, geom_col=geom_c
        remaining_invalid_count = df_fixed.filter(~ST_IsValid(col(geom_col))
        print(f"🔎 After fixing, {remaining_invalid_count} invalid geometrie

        if remaining_invalid_count == 0:
            print("✅ All geometries are valid after fixing.")
            return {"df": df_fixed}
        elif split_on_fail:
            print("⚠️ Some invalid geometries remain — splitting dataset.")
            valid_df = df_fixed.filter(ST_IsValid(col(geom_col)))
            invalid_df = df_fixed.filter(~ST_IsValid(col(geom_col)))
            print(f"✅ Split complete: {valid_df.count()} valid / {invalid_c
            return {"valid_df": valid_df, "invalid_df": invalid_df}
        else:
            print("⚠️ Some invalid geometries remain, returning best-effort
            return {"df": df_fixed}

    # If no fix attempt, just split if requested
    if split_on_fail:
        print("⚠️ Skipping fix — splitting into valid and invalid.")
        valid_df = df.filter(ST_IsValid(col(geom_col)))
        invalid_df = df.filter(~ST_IsValid(col(geom_col)))
        print(f"✅ Split complete: {valid_df.count()} valid / {invalid_df.co
        return {"valid_df": valid_df, "invalid_df": invalid_df}

    print("⚠️ Invalid geometries found but no fix or split requested. Returr
    return {"df": df}
```

In [24]:
```
# FEMA Flood Hazard Areas
fld_hazard_area = sedona.read.format('shapefile').load(f'{prefix}' + '53033C
```

In [25]:
```
result = process_geometries(fld_hazard_area, geom_col="geometry", attempt_fi

if "df" in result:
    df_final = result["df"]  # all valid (either already valid or successful
else:
    valid_df = result["valid_df"]
    invalid_df = result["invalid_df"]
    # handle invalids (e.g., export for manual review)(fld_hazard_area, 'geo
```

```
✅ Checked geometries — found 15 invalid geometries.
🔧 Attempting to fix 15 invalid geometries...
[Stage 39:>                                                         (0 + 1) /
🔎 After fixing, 0 invalid geometries remain.
✅ All geometries are valid after fixing.
```

In [26]:
```
fld_hazard_area.writeTo(f"org_catalog.{database}.fema_flood_zones_bronze").c
```

In [28]:
```
sedona.sql(f'''
select st_srid(geometry) as srid from org_catalog.{database}.fema_flood_zone
```

```
''').show()
```

```
+----+
|srid|
+----+
|4269|
+----+
```

In [29]:
```
sedona.sql(f'''
select
st_srid(st_transform(geometry, 'EPSG:4326')) as srid from org_catalog.{datab
''').show()
```

```
+----+
|srid|
+----+
|4326|
+----+
```

In [30]:
```
sedona.sql(f'''
select
st_srid(st_transform(geometry, 'EPSG:4269', 'EPSG:4326')) as srid from org_c
''').show()
```

```
+----+
|srid|
+----+
|4326|
+----+
```

In [31]:
```
# King County Generalized Land Use Data
gen_land_use = sedona.read.format('shapefile').load(f'{prefix}' + 'General_L
```

In [32]:
```
gen_land_use.writeTo(f"org_catalog.{database}.gen_land_use_bronze").createOr
```

In [33]:
```
# King County Sherrif Patrol Districts
sedona.read.format('shapefile').load(f'{prefix}' + 'King_County_Sheriff_Patr
    .writeTo(f"org_catalog.{database}.sherrif_districts_bronze").createOrRep
```

In [34]:
```
# King County Offense Reports
offense_reports = sedona.read.format('csv').load(f'{prefix}' + 'KCSO_Offense
```

In [35]:
```
offense_reports.writeTo(f"org_catalog.{database}.offense_reports_bronze").cr
```

In [36]:
```
# King County Bike Lanes
bike_lanes = sedona.read.format('shapefile').load(f'{prefix}' + 'Metro_Trans
```

In [37]:
```
bike_lanes.writeTo(f"org_catalog.{database}.bike_lanes_bronze").createOrRepl
```

In [38]:
```
# FEMA National Risk Index
fema_nri = sedona.read.format('shapefile').load(f'{prefix}' + 'NRI_Shapefile
```

In [39]:
```
fema_nri.writeTo(f"org_catalog.{database}.fema_nri_bronze").createOrReplace(
```

```
25/10/23 10:42:40 WARN SparkStringUtils: Truncated the string representation
```

In [40]:
```
# King County School Sites
school_sites = sedona.read.format('shapefile').load(f'{prefix}' + 'School_Si
```

In [41]:
```
school_sites.writeTo(f"org_catalog.{database}.school_sites_bronze").createOr
```

In [42]:
```
# Schools Report Card
report_card = sedona.read. \
    format('csv'). \
    load(f'{prefix}' + 'Report_Card_Growth_for_2024-25_20250923.csv')
```

In [43]:
```
report_card.writeTo(f"org_catalog.{database}.report_card_bronze").createOrRe
```

In [44]:
```
# Seismic Hazards
seismic_hazards = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Seismic_Hazards___seism_area/Seismic_Hazards___seism
```

In [45]:
```
# Seismic Hazards
seismic_hazards = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Seismic_Hazards___seism_area/Seismic_Hazards___seism
```

In [46]:
```
seismic_hazards.writeTo(f"org_catalog.{database}.seismic_hazards_bronze").cr
```

In [47]:
```
# Census Block Groups
block_groups = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'tl_2024_53_bg/tl_2024_53_bg.shp')
```

In [48]:
```
block_groups.writeTo(f"org_catalog.{database}.block_groups_bronze").createOr
```

In [49]:
```
# Census CSVs
median_age = sedona.read. \
    format('csv'). \
```

```
    load(f'{prefix}' + 'ACSDT5Y2023.B01002_2025-09-19T105233/ACSDT5Y2023.B01

median_age.writeTo(f"org_catalog.{database}.median_age_bronze").createOrRepl

total_pop = sedona.read. \
    format('csv'). \
    load(f'{prefix}' + 'ACSDT5Y2023.B01003_2025-09-19T105050/ACSDT5Y2023.B01

total_pop.writeTo(f"org_catalog.{database}.total_pop_bronze").createOrReplac

median_income = sedona.read. \
    format('csv'). \
    load(f'{prefix}' + 'ACSDT5Y2023.B19013_2025-09-19T105253/ACSDT5Y2023.B19

total_pop.writeTo(f"org_catalog.{database}.median_income_bronze").createOrRe
```

In [50]:
```
# Tranist Routes
transit_routes = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Transit_Routes_for_King_County_Metro___transitroute_
```

In [51]:
```
transit_routes.writeTo(f"org_catalog.{database}.transit_routes_bronze").crea
```

In [52]:
```
# Transit Stops
transit_stops = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Transit_Stops_for_King_County_Metro___transitstop_pc
```

In [53]:
```
transit_stops.writeTo(f"org_catalog.{database}.transit_stops_bronze").create
```

In [54]:
```
# Water Bodies
water_bodies = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Waterbodies_with_History_and_Jurisdictional_detail__
```

In [55]:
```
water_bodies.writeTo(f"org_catalog.{database}.water_bodies_bronze").createOr
```

In [56]:
```
# Wildfire Polygons
wildfires = sedona.read. \
    format('shapefile'). \
    load(f'{prefix}' + 'Wildfires_1878_2019_Polygon_Data/Shapefile/US_Wildfi
```

In [57]:
```
wildfires.writeTo(f"org_catalog.{database}.wildfires_bronze").createOrReplac
```

In [58]:
```
# Elevation

url = 's3://copernicus-dem-30m/*/*.tif'
```

```
elev_df = sedona.read.format("raster").option("retile", "true").load(url) \
.where(
    "RS_Intersects(rast, ST_GeomFromText('POLYGON((-125.0572 48.9964, -120.2
)

# Use the below function to load the global DEM

# elev_df = sedona.read.format("raster").option("retile", "true").load(url)
```

```
25/10/23 11:09:08 WARN SharedInMemoryCache: Evicting cached table partition m
```

In [59]:
```
elev_df.writeTo(f"org_catalog.{database}.elevation_bronze").createOrReplace(
```

In [60]:
```
# Geocoded Schools
schools = sedona.read. \
    format('geojson'). \
    option('mode', 'DROPMALFORMED'). \
    load(f'{prefix}' + 'Washington_State_Public_Schools_GeoCoded.geojson')
```

In [61]:
```
schools = schools \
    .withColumn("geometry", expr("geometry")) \
    .withColumn("AYPCode", expr("properties['AYPCode']")) \
    .withColumn("CongressionalDistrict", expr("properties['CongressionalDist
    .withColumn("County", expr("properties['County']")) \
    .withColumn("ESDCode", expr("properties['ESDCode']")) \
    .withColumn("ESDName", expr("properties['ESDName']")) \
    .withColumn("Email", expr("properties['Email']")) \
    .withColumn("GeoCoded_X", expr("properties['GeoCoded_X']")) \
    .withColumn("GeoCoded_Y", expr("properties['GeoCoded_Y']")) \
    .withColumn("GradeCategory", expr("properties['GradeCategory']")) \
    .withColumn("HighestGrade", expr("properties['HighestGrade']")) \
    .withColumn("LEACode", expr("properties['LEACode']")) \
    .withColumn("LEAName", expr("properties['LEAName']")) \
    .withColumn("LegislativeDistrict", expr("properties['LegislativeDistrict
    .withColumn("LowestGrade", expr("properties['LowestGrade']")) \
    .withColumn("MailingAddress", expr("properties['MailingAddress']")) \
    .withColumn("NCES_X", expr("properties['NCES_X']")) \
    .withColumn("NCES_Y", expr("properties['NCES_Y']")) \
    .withColumn("Phone", expr("properties['Phone']")) \
    .withColumn("Principal", expr("properties['Principal']")) \
    .withColumn("School", expr("properties['School']")) \
    .withColumn("SchoolCategory", expr("properties['SchoolCategory']")) \
    .withColumn("SchoolCode", expr("properties['SchoolCode']")) \
    .withColumn("SingleAddress", expr("properties['SingleAddress']")) \
    .drop("properties").drop("type") \
    .drop("_corrupt_record").drop("type") \
    .drop("type").drop("type")
```

In [62]:
```
schools.printSchema()
```

```
root
 |-- geometry: geometry (nullable = true)
 |-- AYPCode: string (nullable = true)
 |-- CongressionalDistrict: string (nullable = true)
 |-- County: string (nullable = true)
 |-- ESDCode: long (nullable = true)
 |-- ESDName: string (nullable = true)
 |-- Email: string (nullable = true)
 |-- GeoCoded_X: double (nullable = true)
 |-- GeoCoded_Y: double (nullable = true)
 |-- GradeCategory: string (nullable = true)
 |-- HighestGrade: string (nullable = true)
 |-- LEACode: long (nullable = true)
 |-- LEAName: string (nullable = true)
 |-- LegislativeDistrict: string (nullable = true)
 |-- LowestGrade: string (nullable = true)
 |-- MailingAddress: string (nullable = true)
 |-- NCES_X: double (nullable = true)
 |-- NCES_Y: double (nullable = true)
 |-- Phone: string (nullable = true)
 |-- Principal: string (nullable = true)
 |-- School: string (nullable = true)
 |-- SchoolCategory: string (nullable = true)
 |-- SchoolCode: long (nullable = true)
 |-- SingleAddress: string (nullable = true)
```

In [63]:
```
schools.writeTo(f"org_catalog.{database}.schools_bronze").createOrReplace()
```

In [ ]:
```

```