ETL Report Guide

for

Illidans-Ingenious-Infrastructure

Prateek Bardhan, William Crocker, Nicholas Fenech

ETL Report for Cohort-50 Capstone Project

Introduction

This ETL Report will act as a guide to replicate the process of extracting, transforming, and loading the data used for the group Illidans-Ingenious-Infrastructure in Cohort-50's capstone project. This report will focus on two datasets pulled from separate sources. We will introduce the sources below, as well as the first steps to begin the process of acquiring the data and provide an understanding for the entire flow of data from beginning to end.

Data Sources

As stated in the introduction, the data required to conduct our project was pulled from two different sources. The first data set comes from NYC Open Data. Be aware, this data set will require an extensive amount of CSV downloading, we will touch more on this later. For the other set, the data was acquired by an Application Programming Interface (API) pull from the United States Census Bureau. The actual data was retrieved from the American Community Survey 5-Year Data (2009-2021). This survey is conducted every 60 months, and for the needs we have will only need to pull from 2021 and 2016 to cover the last decade aggregated. This will also require a method for obtaining the data that may seem complicated, again we will touch on this during extraction.

Both data sites were reviewed from early May of 2023. Below, the APA cited references for both data sources can be seen.

References

- Bureau, U. C. (2022, November 30). *American Community survey 5-year data (2009-2021)*. Census.gov. https://www.census.gov/data/developers/data-sets/acs-5year.html

- Annualized sales update. (n.d.). https://www.nyc.gov/site/finance/taxes/property-annualized-sales-update.page

Extraction

This extraction will be split into two parts, part one will cover NYC Open Data and part 2 will cover United States Census Bureau.

Part 1:

All NYC Open Data extraction will end with it being stored in an Azure Blob Storage Container. Please see the steps outline below to complete this:

1) First, navigate to the following page to extract the xlsx. documents ranging from 2003 to 2022
2) Once on this page, one will need to download need to scroll down to the Detailed Annual Sales Reports by Borough. You will see on the left column titled *MS Excel* with blue Download buttons accompanied by each Borough in New York.
3) From here, download every Borough from each year ranging from 2003 to 2023.
4) Once these all have been downloaded to your local machine (we recommend making a folder for each year to keep track) you have completed the Extraction steps needed to quire the data from the NYC Open Data source.

We will touch on the transformation in a later section.

Part 2:

All Census Bureau Data extraction is not as simple as downloading a CSV, for this extraction one will need to conduct it in Python. We will be using VS Code Editor with a Python Environment. Please see these links to download if needed:

VS Code Editor

Anaconda

You are also free to use any environment and IDE you are comfortable with.

With this API pull, we will provide screenshots of the code as well as directions. If you would like to adjust the type of data being pulled, feel free to access the complete list of variables that can be called from the ACS-5.

1) Once you have your IDE opened, you will first need to import the following libraries:
    a. import requests
    b. import pandas as pd
2) From here, we can begin building the for loop to iterate over the API, this will allow us to pull the two data sets we need.
3) Below is a screen shot of the code:

```
import requests
import pandas as pd

base_url = "https://api.census.gov/data/"
variables = "NAME,B25106_002E,B25106_036E,B25087_001E,B25087_002E,B25087_019E,B25087_018E,B25087_017E,B25087_016E,B25087_015E,
geo_location = "&for=county:005,047,061,081,085&in=state:36"
years = ["2016", "2021"]

for i in years:

    url = base_url + i + "/acs/acs5?get=" + variables + geo_location
    response = requests.get(url)
    df = pd.DataFrame(response.json()[1:], columns=response.json()[0])
    df.to_csv(f"{i}_data.csv", index=False)
```

4) As one can see above, after importing the libraries, we then create variables splitting up the API URL, from there can we iterate over the years needed. After, we then parse the content as JSON, and accept all the data from it. Finally, it is saved as a CSV (use whichever file path works best for your case).
5) It is very important to note that the variables variable above contains specific information to call columns. The geo location is also important, representing the county codes for each Borough of New York. Below is a copy of the variables and geo location as one line of text.
    a. Columns: variables = "NAME,B25106_002E,B25106_036E,B25087_001E,B25087_002E,B25087_019E,B25087_018E,B25087_017E,B25087_016E,B25087_015E,B25087_014E,B25087_013E,B25087_012E,B25105_001E,B25079_005E,B25079_004E,B25079_003E,B25063_002E,B25063_019E,B25063_020E,B25063_021E,B25063_022E,B25063_023E,B25063_024E,B25063_025E,B25063_026E,B01003_001E,B02001_002E,B02001_003E,B02001_004E,B02001_005E,B02001_006E,B02001_007E,B02001_008E,B01002_001E,B01002A_001E,B01002B_001E,B01002C_001E,B01002D_001E,B01002E_001E,B01002F_001E,B01002G_001E"
    b. Geo location: "&for=county:005,047,061,081,085&in=state:36"
6) Be aware, this will create two CSV files for each year, this will be addressed in the transformation step to concatenate them together into one table.
7) Save the CSV files to a path that works best for you.

This concludes the required steps needed to extract all the data needed, feel free to include any extra data if needed. If any questions arise, please reach out to a team member for assistance.

Transformation

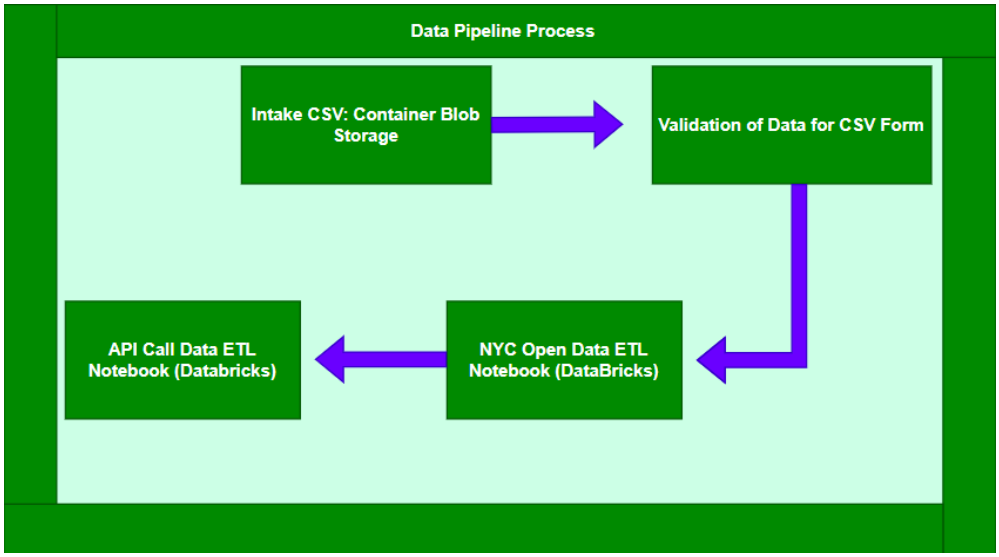
Once that data has been acquired from both sets of data, we can begin the process of transformation. At this point, the data should exist locally within your machine, however, through the transformations we will see the set up of the pipeline, uploading to the cloud, and eventually loading to a SQL database. Transformation will occur almost completely in Databricks, as well as some pre-upload.

We will break up the transformation steps into two parts again.

Part 1:

1) First, go to each downloaded CSV file from the NYC Open Data, and remove the first four rows from each spreadsheet as they were descriptive of the dataset and did not contain any actual data
2) Double check all column names are consistent each file as well.
3) Convert and save all xlsx. files to CSV files with the same standardized name i.e., *2003_bronx.csv*
4) Each year of files will need to be added to their own year folder, the total folders should range from 2003-2023 with all 5 borough CSVs in each year.
5) Once this has been completed, we will upload all folders into an Azure Data Container.
6) Now you will log in and navigate to your Microsoft Azure container area, create or use an existing blob, and upload all folders in the newly created blob.
7) While here, also create a secondary blob, this will be used to store long term data.

We will pause here to show an image of the data pipeline, see below:



It is important to understand the flow of the data from intake to outtake. See the steps below:

- The pipeline contains a trigger that will automatically run the pipeline once a new file has been uploaded to the first blob storage.
- From there, the data is then dumped out into another blob storage container that is connected to Databricks, as well as for long term storage.
- This will then run the pipeline to the notebook NYC Open Data, where the transformation, and final load process will occur in Databricks.
- Finally, the pipeline will hit the notebook for the API, where the data will be called and then consumed by Kafka, once in the confluent, the data will be produced back to the notebook, converted to a data frame, then finally dumped into our SQL database.

This is a high-level overview of the pipeline used, it is important to understand since the transformation steps occur as part of the pipeline. We will continue with the transformation below:

8) Now, you will need to sign into Databricks, and create a token to connect to the Azure blob, documentation for this can be found here.
9) Once your data bricks notebook is connected, we can begin the ETL process.
10) Create a mount point to the Azure blob to access the year folders, below is an example:

```
client_id = "5c0c63bf-b70d-461a-b3cb-fe0f9aeebe4a"
client_secret = "TvR8Q~Fn4FAsZTFIZO.-umdb61Kd2mfQDQdZBaQW"
tenant_id = "d4e104e3-ae7d-4371-a239-745aa8960cc9"

storage_account = "cohort50storage"
storage_container = "illidans-ingenious-infrastructure"

mount_point = "/mnt/illidans-ingenious-infrastructure"

configs = {"fs.azure.account.auth.type": "OAuth",
       "fs.azure.account.oauth.provider.type": "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
       "fs.azure.account.oauth2.client.id": client_id,
       "fs.azure.account.oauth2.client.secret": client_secret,
       "fs.azure.account.oauth2.client.endpoint": f"https://login.microsoftonline.com/{tenant_id}/oauth2/token",
       "fs.azure.createRemoteFileSystemDuringInitialization": "true"}

try:
    dbutils.fs.unmount(mount_point)
except:
    pass

dbutils.fs.mount(
source = f"abfss://{storage_container}@{storage_account}.dfs.core.windows.net/",
mount_point = mount_point,
extra_configs = configs)
```

11) Once the mount point has been created please check to make sure it was made, then import the libraries Pandas and Numpy. Below is a code snippet of the transformation process:

```
years = []
for i in range(2003,2023):
    years.append(str(i))
for i in years:
    globals()['manhattan'+i] = spark.read.csv('/mnt/illidans-ingenious-infrastructure/'+i+'/'+i+'_manhattan.csv', header = True).toPandas()
    globals()['bronx'+i] = spark.read.csv('/mnt/illidans-ingenious-infrastructure/'+i+'/'+i+'_bronx.csv', header = True).toPandas()
    globals()['brooklyn'+i] = spark.read.csv('/mnt/illidans-ingenious-infrastructure/'+i+'/'+i+'_brooklyn.csv', header = True).toPandas()
    globals()['queens'+i] = spark.read.csv('/mnt/illidans-ingenious-infrastructure/'+i+'/'+i+'_queens.csv', header = True).toPandas()
    globals()['staten_island'+i] = spark.read.csv('/mnt/illidans-ingenious-infrastructure/'+i+'/'+i+'_staten_island.csv', header = True).toPandas()

    globals()['boroughs'+i] = pd.concat([globals()['manhattan'+i],globals()['bronx'+i],globals()['brooklyn'+i],globals()['queens'+i],globals()['staten_island'+i]])
    globals()['boroughs'+i].dropna(how='all',inplace=True)

    globals()['boroughs'+i].drop(['EASE-MENT','APARTMENT NUMBER','BUILDING CLASS AT PRESENT','BUILDING CLASS AT TIME OF SALE','ADDRESS'],axis=1,inplace=True);

    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['SALE PRICE'].str.contains('-',regex=False) == False]
    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['LAND SQUARE FEET'].str.contains('-',regex=False) == False]
    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['GROSS SQUARE FEET'].str.contains('-',regex=False) == False]
    globals()['boroughs'+i]['SALE PRICE']=globals()['boroughs'+i]['SALE PRICE'].str.replace('$','', regex=False)
    globals()['boroughs'+i]['SALE PRICE']=globals()['boroughs'+i]['SALE PRICE'].str.replace(',','', regex=False)
    globals()['boroughs'+i]['LAND SQUARE FEET']=globals()['boroughs'+i]['LAND SQUARE FEET'].str.replace(',','', regex=False)
    globals()['boroughs'+i]['GROSS SQUARE FEET']=globals()['boroughs'+i]['GROSS SQUARE FEET'].str.replace(',','', regex=False)
    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['LAND SQUARE FEET'] != '']
    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['GROSS SQUARE FEET'] != '']
    globals()['boroughs'+i]['LAND SQUARE FEET'] = globals()['boroughs'+i]['LAND SQUARE FEET'].replace("0",np.nan)
    globals()['boroughs'+i]['GROSS SQUARE FEET'] = globals()['boroughs'+i]['GROSS SQUARE FEET'].replace("0",np.nan)
    globals()['boroughs'+i] = globals()['boroughs'+i].astype({'SALE PRICE': float})
    globals()['boroughs'+i] = globals()['boroughs'+i].astype({'LAND SQUARE FEET': float})
    globals()['boroughs'+i] = globals()['boroughs'+i].astype({'GROSS SQUARE FEET': float})
    globals()['boroughs'+i] = globals()['boroughs'+i].astype({'YEAR BUILT': str})
    globals()['boroughs'+i]['YEAR BUILT'] = globals()['boroughs'+i]['YEAR BUILT'].replace("0",np.nan)
    globals()['boroughs'+i]['YEAR BUILT'] = globals()['boroughs'+i]['YEAR BUILT'].replace("0.0",np.nan)
    globals()['boroughs'+i] = globals()['boroughs'+i][globals()['boroughs'+i]['SALE PRICE'] > 1000]
    globals()['boroughs'+i]['SALE DATE'] = pd.to_datetime(globals()['boroughs'+i]['SALE DATE'])
```

As you can see above, run a for loop to read in each file from the container by year, we then conduct all transformation steps, cleaning the data to make it into a suitable form to be used. (Recommend zooming in to see the code clearly)

12) Here are the steps for conducting the entire transformation process:

    a. In a Databricks notebook, used a for loop to iterate through all years from 2003 to 2022, so the following steps apply to all 20 years:
    b. Read CSVs from each borough into a Spark DataFrame and converted them all to pandas DataFrames
    c. Concatenated all five boroughs into one DataFrame
    d. Dropped any rows that were completely null
    e. Dropped these columns as they were deemed unnecessary:
        i. EASE-MENT
        ii. APARTMENT NUMBER
        iii. BUILDING CLASS AT PRESENT
        iv. BUILDING CLASS AT TIME OF SALE
        v. ADDRESS
    f. Dropped rows containing ‘-’ using str.contains() in these columns:

i. SALE PRICE
ii. LAND SQUARE FEET
iii. GROSS SQUARE FEET

g. Replaced values containing '$' with an empty string in the SALE PRICE column
h. Replaced values containing ',' with an empty string in these columns:
    i. SALE PRICE
    ii. LAND SQUARE FEET
    iii. GROSS SQUARE FEET
i. Dropped rows with only empty strings in these columns:
    i. LAND SQUARE FEET
    ii. GROSS SQUARE FEET
j. Replaced "0" with np.nan in these columns:
    i. LAND SQUARE FEET
    ii. GROSS SQUARE FEET
k. Changed data types of these columns:
    i. SALE PRICE: float
    ii. LAND SQUARE FEET: float
    iii. GROSS SQAURE FEET: float
    iv. YEAR BUILT: str
l. Replaced "0" and "0.0" with np.nan in the YEAR BUILT column

a. Dropped values where the value was less than or equal to 1000 in the SALE PRICE column
b. Changed the SALE DATE column to a datetime object using to_datetime
c. Concatenated all 20 years worth of data into one DataFrame
d. Used reset_index with drop=True and inplace=True
e. Selected all columns with the object type and used a lambda function with str.strip() to remove any leading and trailing spaces
f. Dropped the four rows with the following indices because the YEAR BUILT was before New York City ever existed:
    i. 1010446
    ii. 821591
    iii. 1019439
    iv. 764666

13) After completing all the steps above, the data should be completed and contained within a singular data frame. We will touch on loading this to a SQL database later in the report.

Below we will continue to part 2.

Part 2:

The transformation steps involved for the Census Bureau data is a little more straightforward.

However, it is worth noting that the transformation process for this data takes place at the end of the pipeline, please reference this information if needed above.

See all transform steps below:

1) Just as with the NYC Open Data, all transform steps will need to be coded within a Databricks notebook.
2) Please log in to your Databricks cluster
3) Create a mount point again for the long-term storage cluster
4) Now, the same extraction code/steps used below will be used within Databricks, however we will transform the data prior to saving it to our long-term blob storage container.

Please see the code snippet below:

```
import requests
import pandas as pd
from pyspark.sql.functions import col,lit

base_url = "https://api.census.gov/data/"
variables = "NAME,B25106_002E,B25106_036E,B25087_001E,B25087_002E,B25087_019E,B25087_018E,B25087_017E,B25087_016E,B25087_015E,B25087_014E,B25087_013E,B25087_012E,B25105_001E,B25079_005E,B25079_004E,B25079_003E,B25063_002E,
B25063_019E,B25063_020E,B25063_021E,B25063_022E,B25063_023E,B25063_024E,B25063_025E,B25063_026E,B01003_001E,B02001_002E,B02001_003E,B02001_004E,B02001_005E,B02001_006E,B02001_007E,B02001_008E,B01002_001E,B01002A_001E,
B01002B_001E,B01002C_001E,B01002D_001E,B01002E_001E,B01002F_001E,B01002G_001E"
geo_location = "&for=county:005,047,061,081,085&in=state:36"
years = ["2016", "2021"]

for i in years:

    url = base_url + i + "/acs/acs5?get=" + variables + geo_location
    response = requests.get(url)
    df = pd.DataFrame(response.json()[1:], columns=response.json()[0])
    new_columns = {'B25106_002E': 'Housing Cost: Total Survey Population', 'B25106_036E': 'Housing Cost 30 percent or more of income', 'B25087_001E' : 'Mortgage Status: Total Survey Population',
                'B25087_002E': 'Housing units with a Mortgage', 'B25087_019E': 'Mortgage Cost over $4,000 per Month', 'B25105_001E': 'Median Monthly Housing Cost', 'B25079_005E': 'Aggregate Value of Home: Owner Age 65 annd
                Over',
                'B25079_004E': 'Aggregate Value of Home: Owner Age 35 to 64 Years Old', 'B25079_003E' : 'Aggregate Value of Home: Owner Age 25 to 34 Years Old', 'B25087_012E': 'Mortgage Cost between $1,000 to $1,249',
                'B25087_018E': 'Mortgage Cost between $3,500 to $3,999','B25087_017E': 'Mortgage Cost between $3,000 to $3,499','B25087_016E': 'Mortgage Cost between $2,999 to $2,500','B25087_015E': 'Mortgage Cost between $2,
                000 to $2,499',
                'B25087_014E': 'Mortgage Cost between $1,500 to $1,999','B25087_013E': 'Mortgage Cost between $1,250 to $1,499','B25063_002E' : 'Gross Rent: Total Survey Population', 'B25063_019E': 'Rent Cost between $900 to
                $999',
                'B25063_020E': 'Rent Cost between $1,000 to $1,249', 'B25063_021E':' Rent Cost between $1,250 to $1,499' , 'B25063_022E': 'Rent Cost between $1,500 to $1,999','B25063_023E': 'Rent Cost between $2,000 to $2,
                499',
                'B25063_024E': 'Rent Cost between $2,500 to $2,999', 'B25063_025E': 'Rent Cost between $3,000 to $3,499', 'B25063_026E': 'Rent Cost between $3,500 or More','B01003_001E': 'Total Population',
                'B02001_002E': 'Race: White','B02001_003E': 'Race: Black or African American ','B02001_004E': 'Race: American Indian and Alaska Native','B02001_005E': 'Race: Asian','B02001_006E': 'Race: Native Hawaiian and
                Other Pacific Islander',
                'B02001_007E': 'Race: Other Race Alone', 'B02001_008E': 'Race: Two or More Races', 'B01002_001E': 'Median Age of Total Population','B01002A_001E': 'Median Age: White', 'B01002B_001E': 'Median Age: Black or
                African American',
                'B01002C_001E': 'Median Age: American Indian and Alaska Native', 'B01002D_001E': 'Median Age: Asian', 'B01002E_001E': 'Median Age: Native Hawaiian and Other Pacific Islander', 'B01002F_001E': 'Median Age:
                Other Race Alone',
                'B01002G_001E': 'Median Age: Two or More Races', 'NAME': 'Borough'}
    df = df.rename(columns=new_columns)
    df = spark.createDataFrame(df)
    df = df.withColumn("Year Survey", lit(i))
    df.write.mode("overwrite").option("header", "true").csv(f"/mnt/illidans-ingenious-infrastructure/census_data/data_{i}.csv")
```

This code is the initial transformation of the data, we will save this to our long-term blob storage as separate files, next we will read them back to concatenate the files into one database.

Please see the code below:

```
df3 = spark.read.csv("/mnt/illidans-ingenious-infrastructure/census_data/data_2021.csv", header=True, inferSchema=True)
df4 = spark.read.csv("/mnt/illidans-ingenious-infrastructure/census_data/data_2016.csv", header=True, inferSchema=True)

concatenated_df = df3.union(df4)

columns_to_drop = ["state", "county"]
census_bureau_dataframe = concatenated_df.drop(*columns_to_drop)
```

```
census_bureau_dataframe.write.mode("overwrite").option("header", "true").csv(f"/mnt/illidans-ingenious-infrastructure/census_data/census_bureau_dataframe.csv")
```

5) Here are the steps for conduction the entire transformation process:
    a. After importing appropriate libraries, begin by replacing all column names with appropriate header names, see all below:
        i. Borough, Housing Cost: Total Survey Population, Housing Cost 30 percent or more of income,
        ii. Mortgage Status: Total Survey Population,
        iii. Housing units with a Mortgage,
        iv. Mortgage Cost over $4,000 per Month,
        v. Mortgage Cost between $3,500 to $3,999,
        vi. Mortgage Cost between $3,000 to $3,499,
        vii. Mortgage Cost between $2,999 to $2,500,
        viii. Mortgage Cost between $2,000 to $2,499,
        ix. Mortgage Cost between $1,500 to $1,999,
        x. Mortgage Cost between $1,250 to $1,499,
        xi. Mortgage Cost between $1,000 to $1,249,
        xii. Median Monthly Housing Cost,
        xiii. Aggregate Value of Home: Owner Age 65 and Over,
        xiv. Aggregate Value of Home: Owner Age 35 to 64 Years Old,
        xv. Aggregate Value of Home: Owner Age 25 to 34 Years Old,
        xvi. Gross Rent: Total Survey Population,
        xvii. Rent Cost between $900 to $999,
        xviii. Rent Cost between $1,000 to $1,249,
        xix. Rent Cost between $1,250 to $1,499,
        xx. Rent Cost between $1,500 to $1,999,

xxi. Rent Cost between $2,000 to $2,499,
xxii. Rent Cost between $2,500 to $2,999,
xxiii. Rent Cost between $3,000 to $3,499,
xxiv. Rent Cost between $3,500 or More,
xxv. Total Population,
xxvi. Race: White,
xxvii. Race: Black or African American ,
xxviii. Race: American Indian and Alaska Native,
xxix. Race: Asian,
xxx. Race: Native Hawaiian and Other Pacific Islander,
xxxi. Race: Other Race Alone,
xxxii. Race: Two or More Races,
xxxiii. Median Age of Total Population,
xxxiv. Median Age: White,
xxxv. Median Age: Black or African American,
xxxvi. Median Age: American Indian and Alaska Native,
xxxvii. Median Age: Asian,
xxxviii. Median Age: Native Hawaiian and Other Pacific Islander,
xxxix. Median Age: Other Race Alone,
xl. Median Age: Two or More Races,
xli. Survey Range (Years),

b. Please make sure to align the appropriate variable name with the column header, this can be found in the documentation provided above
c. We then add a ‘Year Survey” column to keep track of which survey we are using.
d. From there, the two different surveys are saved to the long-term blob storage
e. For our purposes, we will then read back in both CSVs
f. Finally, using union to concatenate the data sets together as well as dropping the ‘county’ and ‘state’ columns, we have a final data frame to then be loaded

This concludes all the transformation steps needed for our datasets, we will now look at loading them to an appropriate database.

Load

Now that both datasets have been extracted and transformed, we can move on to loading them into our SQL database. Below are the steps used to load both datasets using the same code in a Databricks notebook.

1) First, one will need to make sure they have created an SQL database within Microsoft Azure Data Studio, documentation for this can be found here
2) We first loaded our datasets above as denormalized tables, then normalized with DDL and DML creation, we will include some of this information later
3) Now, please have Azure Data Studio open, and make sure you are connected to your database
4) For the NYC Open Data, since the data set is so large, chunks of the data frame needed to be sent to SQL, please see the code below: (Also be aware, both data frames will need to exist as Pandas Data frames, not PySpark)

```
a. chunks = [boroughs_housing[i:i+10000] for i in range(0,
   len(boroughs_housing), 10000)]
```

   b. This code uses list comprehension to send chunks of 10,000 rows of data
   c. After conducting this, we can then send all the data to the SQL database
   d. The code below first establishes connection credentials in the form of variables, then, we can see the use of a for loop to send chunks of data, and finally additional code to implement the SQL dump.
   e. Again, be aware this is only required for the NYC Open Data, we will show father below how to send the Census Bureau data

```
database = 'Illidans_Ingenious_Infrastructure'
table = 'dbo.boroughs_housing'
user = III_db_login
password = III_db_password
server = "cohort50sqlserver.database.windows.net"

for i in range(len(chunks)):
    boroughs_housing_chunk = spark.createDataFrame(chunks[i])
    boroughs_housing_chunk.write.format("jdbc").option(
        "url", f"jdbc:sqlserver://{server}:1433;databaseName={database};"
        ) \
        .mode("append") \
        .option("dbtable", table) \
        .option("user", user) \
        .option("password", password) \
        .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
        .save()
```

a. After executing this code, you should see a table within your database, the total amount of rows should be 1,113,589 (to verify)

8) Now, for the Census Bureau data, the process is only one block of code

a. Double check that the census data is contained within a Pandas data frame

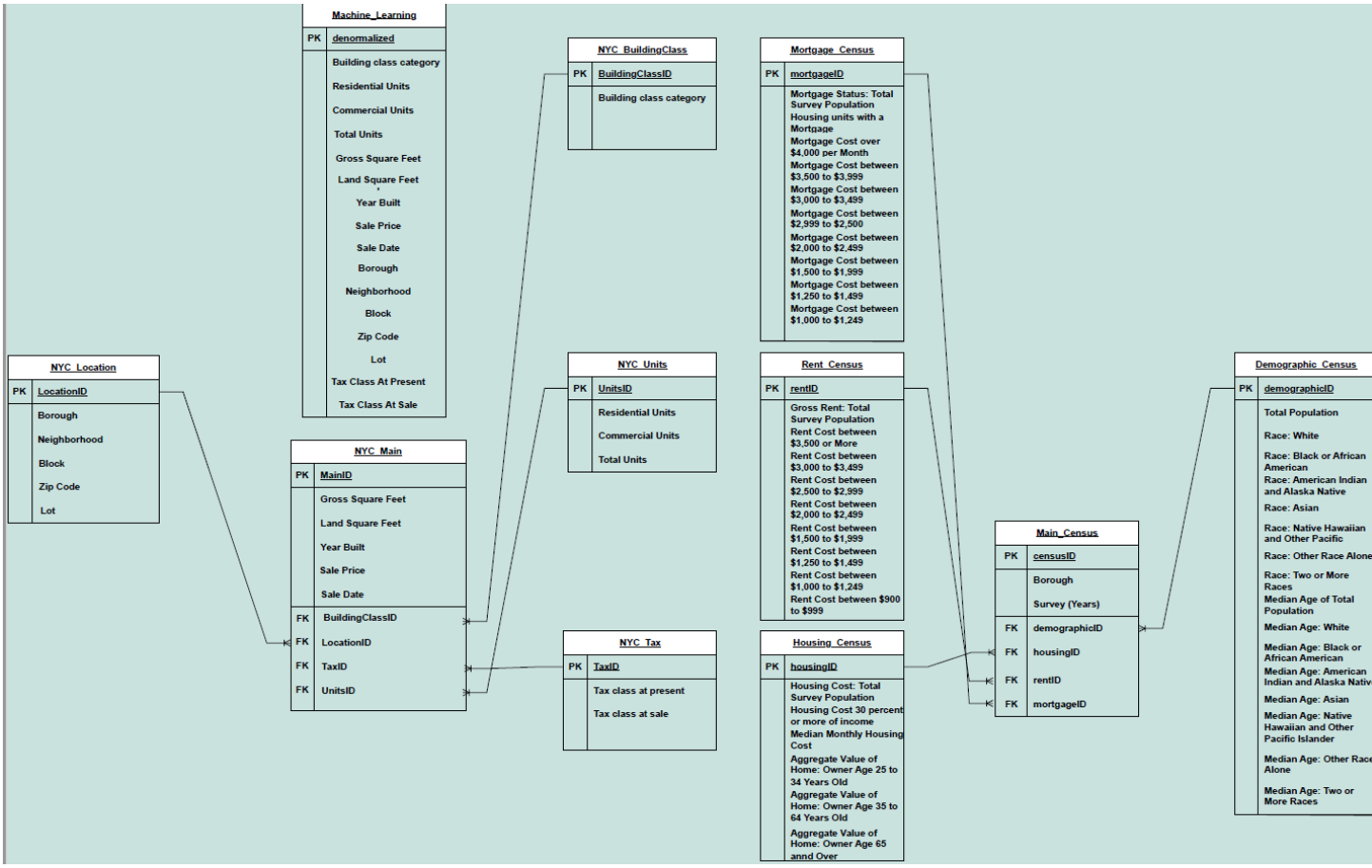
b. We then use the same model as above without chunking, establish credentials, format data frame, then dump into SQL database denormalized

c. To verify, the shape of the data frame should be 43 by 10, see the code below:

```
database = 'Illidans_Ingenious_Infrastructure'
table = 'dbo.census_data'
user = III_db_login
password = III_db_password
server = "cohort50sqlserver.database.windows.net"

df_from_list_of_dictionaries.write.format("jdbc").option(
    "url", f"jdbc:sqlserver://{server}:1433;databaseName={database};"
    ) \
    .mode("append") \
    .option("dbtable", table) \
    .option("user", user) \
    .option("password", password) \
    .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
    .save()
```

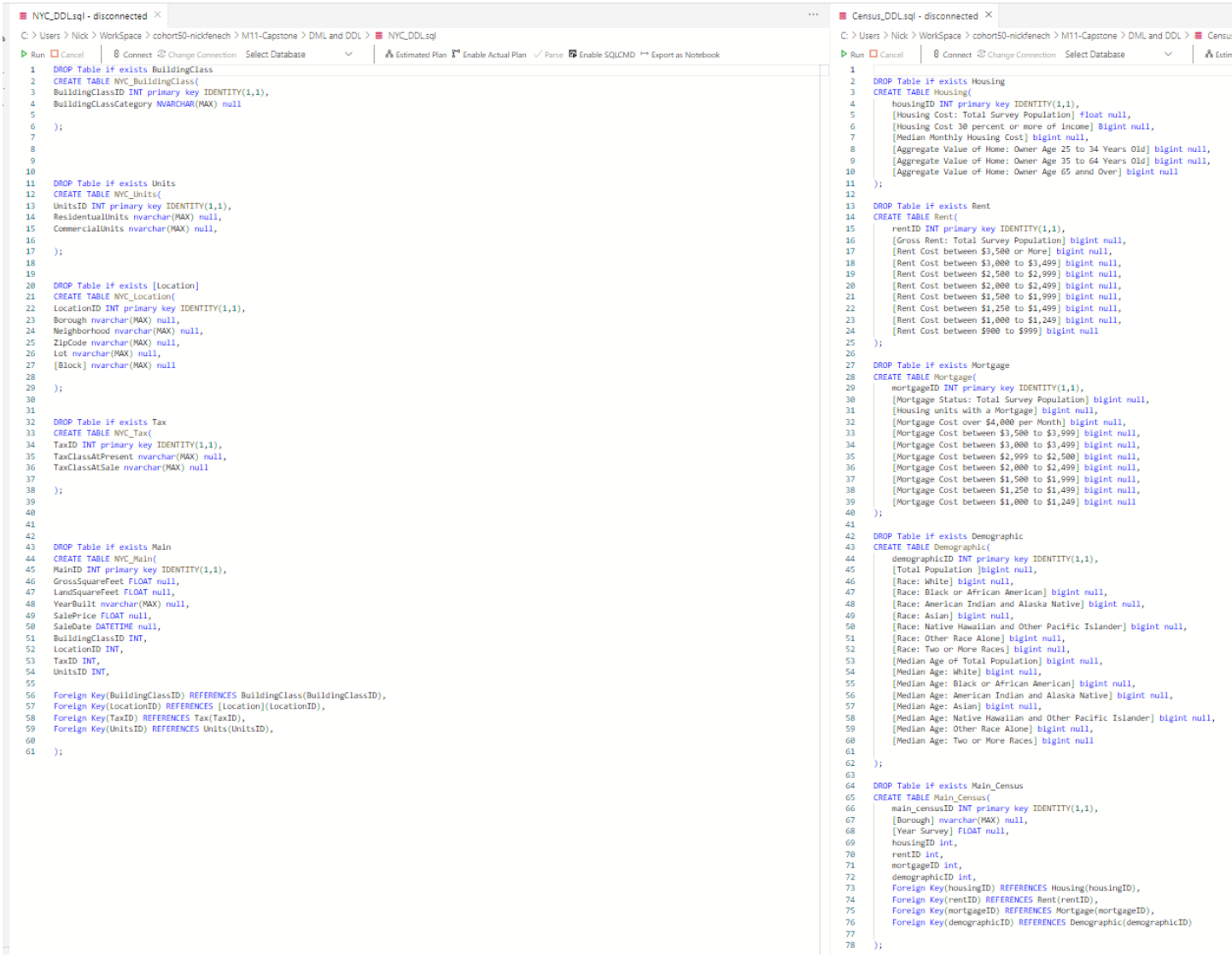
After conducting all steps above, one should be left with two denormalized tables in their SQL database and verified for size and shape above. We decided to conduct our normalization all with SQL, below is a snapshot of the ERD as well as all DDL and DML code:

Entity Relationship Diagram:



(Zooming in is recommended by holding CLTR + scrolling forward with your mouse)

DDL:

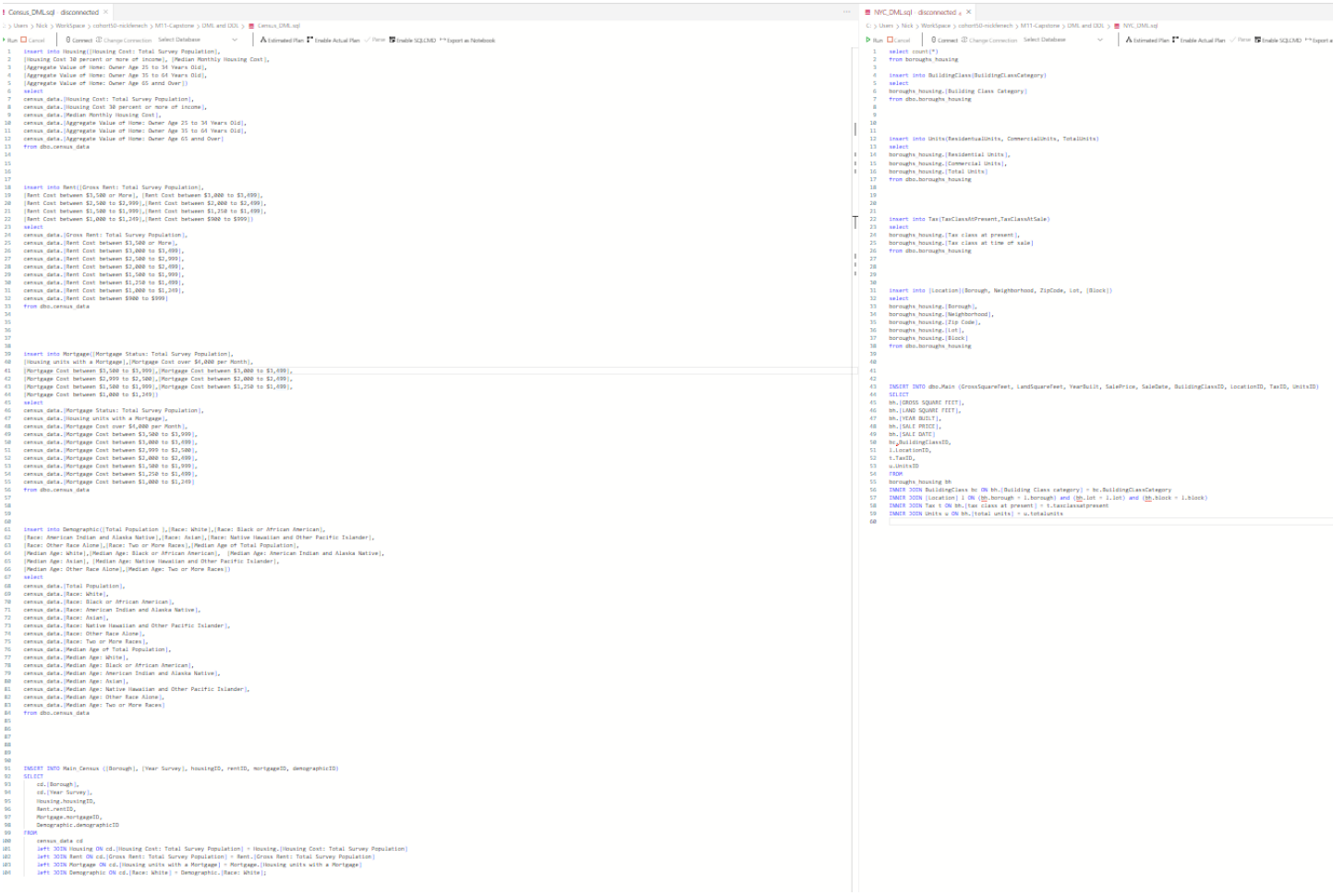


```
DROP Table if exists BuildingClass
CREATE TABLE NYC_BuildingClass(
BuildingClassID INT primary key IDENTITY(1,1),
BuildingClassCategory NVARCHAR(MAX) null

);




DROP Table if exists Units
CREATE TABLE NYC_Units(
UnitsID INT primary key IDENTITY(1,1),
ResidentualUnits nvarchar(MAX) null,
CommercialUnits nvarchar(MAX) null,

);


DROP Table if exists [Location]
CREATE TABLE NYC_Location(
LocationID INT primary key IDENTITY(1,1),
Borough nvarchar(MAX) null,
Neighborhood nvarchar(MAX) null,
ZipCode nvarchar(MAX) null,
Lot nvarchar(MAX) null,
[Block] nvarchar(MAX) null

);


DROP Table if exists Tax
CREATE TABLE NYC_Tax(
TaxID INT primary key IDENTITY(1,1),
TaxClassAtPresent nvarchar(MAX) null,
TaxClassAtSale nvarchar(MAX) null

);




DROP Table if exists Main
CREATE TABLE NYC_Main(
MainID INT primary key IDENTITY(1,1),
GrossSquareFeet FLOAT null,
LandSquareFeet FLOAT null,
YearBuilt nvarchar(MAX) null,
SalePrice FLOAT null,
SaleDate DATETIME null,
BuildingClassID INT,
LocationID INT,
TaxID INT,
UnitsID INT,

Foreign Key(BuildingClassID) REFERENCES BuildingClass(BuildingClassID),
Foreign Key(LocationID) REFERENCES [Location](LocationID),
Foreign Key(TaxID) REFERENCES Tax(TaxID),
Foreign Key(UnitsID) REFERENCES Units(UnitsID),

);
```

```
DROP Table if exists Housing
CREATE TABLE Housing(
    housingID INT primary key IDENTITY(1,1),
    [Housing Cost: Total Survey Population] float null,
    [Housing Cost 30 percent or more of income] Bigint null,
    [Median Monthly Housing Cost] bigint null,
    [Aggregate Value of Home: Owner Age 25 to 34 Years Old] bigint null,
    [Aggregate Value of Home: Owner Age 35 to 64 Years Old] bigint null,
    [Aggregate Value of Home: Owner Age 65 annd Over] bigint null
);

DROP Table if exists Rent
CREATE TABLE Rent(
    rentID INT primary key IDENTITY(1,1),
    [Gross Rent: Total Survey Population] bigint null,
    [Rent Cost between $3,500 or More] bigint null,
    [Rent Cost between $3,000 to $3,499] bigint null,
    [Rent Cost between $2,500 to $2,999] bigint null,
    [Rent Cost between $2,000 to $2,499] bigint null,
    [Rent Cost between $1,500 to $1,999] bigint null,
    [Rent Cost between $1,250 to $1,499] bigint null,
    [Rent Cost between $1,000 to $1,249] bigint null,
    [Rent Cost between $900 to $999] bigint null
);

DROP Table if exists Mortgage
CREATE TABLE Mortgage(
    mortgageID INT primary key IDENTITY(1,1),
    [Mortgage Status: Total Survey Population] bigint null,
    [Housing units with a Mortgage] bigint null,
    [Mortgage Cost over $4,000 per Month] bigint null,
    [Mortgage Cost between $3,500 to $3,999] bigint null,
    [Mortgage Cost between $3,000 to $3,499] bigint null,
    [Mortgage Cost between $2,999 to $2,500] bigint null,
    [Mortgage Cost between $2,000 to $2,499] bigint null,
    [Mortgage Cost between $1,500 to $1,999] bigint null,
    [Mortgage Cost between $1,250 to $1,499] bigint null,
    [Mortgage Cost between $1,000 to $1,249] bigint null
);

DROP Table if exists Demographic
CREATE TABLE Demographic(
    demographicID INT primary key IDENTITY(1,1),
    [Total Population ]bigint null,
    [Race: White] bigint null,
    [Race: Black or African American] bigint null,
    [Race: American Indian and Alaska Native] bigint null,
    [Race: Asian] bigint null,
    [Race: Native Hawaiian and Other Pacific Islander] bigint null,
    [Race: Other Race Alone] bigint null,
    [Race: Two or More Races] bigint null,
    [Median Age of Total Population] bigint null,
    [Median Age: White] bigint null,
    [Median Age: Black or African American] bigint null,
    [Median Age: American Indian and Alaska Native] bigint null,
    [Median Age: Asian] bigint null,
    [Median Age: Native Hawaiian and Other Pacific Islander] bigint null,
    [Median Age: Other Race Alone] bigint null,
    [Median Age: Two or More Races] bigint null

);

DROP Table if exists Main_Census
CREATE TABLE Main_Census(
    main_censusID INT primary key IDENTITY(1,1),
    [Borough] nvarchar(MAX) null,
    [Year Survey] FLOAT null,
    housingID int,
    rentID int,
    mortgageID int,
    demographicID int,
    Foreign Key(housingID) REFERENCES Housing(housingID),
    Foreign Key(rentID) REFERENCES Rent(rentID),
    Foreign Key(mortgageID) REFERENCES Mortgage(mortgageID),
    Foreign Key(demographicID) REFERENCES Demographic(demographicID)

);
```

(Once again, zooming is recommended, please reach out to gain access to any of the code in its original file)

DML:



Conclusion

The ETL process above has guided one to extract, transform, and load both datasets used within the scope of current needs. Pulling both data sets, conducting data cleaning, and loading them both into a SQL database will then allow a user to access the data for analysis and visualization. This ETL process also uses Kafka for the API Census Bureau data, however it is not included here due to its use of a proof of concept. Feel free to review this documentation at any point in the future for reference. Any questions can be directed to Illidans-Ingenious-Infrastructure group.