ETL Processes

**Fist iteration of ETL: Take a part of your design and implement ETL. The scope at this point is limited and should include just two of your dimensions and one fact table. Using Python, SQL document and walk through the initial process of loading your data into staging (SQL staging or a data frame) doing some transformation and loading into your dimensions and a fact table.**

Import datasets/ Data exploration:

*#!/usr/bin/env python*

*# coding: utf-8*

*# In[32]:*

**import** pandas **as** pd

**import** numpy **as** np

**import** pyodbc **as** odbc

*# In[69]:*

Airbnb\_df **=** pd.read\_csv('AB\_NYC\_2019.csv')

Airbnb\_df.head()

*# In[3]:*

Taxi\_df **=** pd.read\_csv('train.csv')

Taxi\_df.head()

*# In[4]:*

Taxi\_df2 **=** pd.read\_csv('test.csv')

Taxi\_df2.head()

*# In[6]:*

*# Combine the train and test df as Taxi\_df*

*# Add a new column to identify the source*

Taxi\_df['source'] **=** 'train'

Taxi\_df2['source'] **=** 'test'

*# Reset Index*

Taxi\_df2 **=** Taxi\_df2.reindex(columns**=**Taxi\_df.columns)

*# Concatenate train and test dataframe*

Taxi\_df **=** pd.concat([Taxi\_df, Taxi\_df2], ignore\_index**=True**)

*# Fill any missing value with NaN*

Taxi\_df['trip\_duration'] **=** Taxi\_df['trip\_duration'].fillna(**-**1)

*# Save the new df as CSV file*

Taxi\_df.to\_csv('NYC\_Taxi.csv', index**=False**)

*# In[7]:*

*# Check the new df*

Taxi\_df.head()

*# In[8]:*

*# Total rows in our target dfs*

Taxi\_total\_rows **=** Taxi\_df.shape[0]

Airbnb\_total\_rows **=** Airbnb\_df.shape[0]

print('Taxi total rows: ', Taxi\_total\_rows, '\nAirbnb total rows: ', Airbnb\_total\_rows)

*# In[11]:*

*# Count the total columns*

Taxi\_total\_cols **=** Taxi\_df.shape[1]

Airbnb\_total\_cols **=** Airbnb\_df.shape[1]

print('Taxi total columns: ', Taxi\_total\_cols, '\nAirbnb total columns: ', Airbnb\_total\_cols)

*# In[14]:*

*# Dtypes*

Airbnb\_df.dtypes

*# In[15]:*

Taxi\_df.dtypes

*# In[19]:*

*# PART 2 - Creating SCD Type 2 and 3 Dim and Key Maintenance*

*# dfs Cleaning*

Airbnb\_df.isnull().sum()

*# Duplication check*

duplicate\_counts **=** Airbnb\_df.groupby(['id','host\_id','host\_name']).size().reset\_index(name**=**'Count')

*# Filter the rows where the count is greater than 1 to find the combinations*

Airbnb\_duplicate **=** duplicate\_counts[duplicate\_counts['Count'] **>** 1]

print("Duplicate combinations of Columns in Airbnb:\n")

print(Airbnb\_duplicate['Count'].sum())

print('\n',Airbnb\_duplicate)

*# In[43]:*

*# Data normalization*

Airbnb\_df['host\_name'] **=** Airbnb\_df['host\_name'].str.slice(stop**=**255)

Airbnb\_df['host\_name'] **=** Airbnb\_df['host\_name'].str.encode('ascii', errors**=**'replace').str.decode('ascii')

Airbnb\_df['name'] **=** Airbnb\_df['name'].str.slice(stop**=**255)

Airbnb\_df['name'] **=** Airbnb\_df['name'].str.encode('ascii', errors**=**'replace').str.decode('ascii')

Airbnb\_df['last\_review'] **=** pd.to\_datetime(Airbnb\_df['last\_review'])

Airbnb\_df['reviews\_per\_month'] **=** Airbnb\_df['reviews\_per\_month'].fillna(0)

*# In[45]:*

*# Filter and load data*

Airbnb\_filtered **=** Airbnb\_df[['id','name','host\_id','host\_name','neighbourhood\_group','neighbourhood',

                            'room\_type','longitude','latitude','price','calculated\_host\_listings\_count',

                             'availability\_365','last\_review','reviews\_per\_month']]

Airbnb\_filtered.head()

*#Airbnb\_filtered.dtypes*

*# In[66]:*

*# Estiblish the connection*

conn **=** odbc.connect(Trusted\_Connection **=** 'YES',

                    Driver **=** '{ODBC Driver 17 for SQL Server}',

                    Server **=** 'DESKTOP-IAN\JLM\_SQLSERVER',

                    Database **=** 'CS689\_fINALpROJ')

print(conn)

*# In[67]:*

*# Create table*

cursor **=** conn.cursor()

cursor.execute("""

CREATE TABLE Airbnb\_staging (

    airbnb\_id INT PRIMARY KEY,

    name NVARCHAR(255),

    host\_id INT,

    host\_name NVARCHAR(255),

    neighborhood\_group NVARCHAR(255),

    neighborhood NVARCHAR(255),

    room\_type NVARCHAR(255),

    longitude FLOAT,

    latitude FLOAT,

    price INT,

    calculated\_host\_listings\_count INT,

    availability\_365 INT,

    last\_review DATETIME,

    reviews\_per\_month FLOAT

)

""")

conn.commit()

conn.close()

*# In[68]:*

*# Load data*

conn **=** odbc.connect(Trusted\_Connection **=** 'YES',

                    Driver **=** '{ODBC Driver 17 for SQL Server}',

                    Server **=** 'DESKTOP-IAN\JLM\_SQLSERVER',

                    Database **=** 'CS689\_fINALpROJ')

cursor **=** conn.cursor()

**for** row **in** Airbnb\_filtered.itertuples():

**try**:

        cursor.execute("""

            INSERT INTO Airbnb\_staging (

                airbnb\_id,

                name,

                host\_id,

                host\_name,

                neighborhood\_group,

                neighborhood,

                room\_type,

                longitude,

                latitude,

                price,

                calculated\_host\_listings\_count,

                availability\_365,

                last\_review,

                reviews\_per\_month

            )

            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

            """,

            row.id,

            row.name **if** **not** pd.isna(row.name) **else** 'Unknown',

            row.host\_id,

            row.host\_name **if** **not** pd.isna(row.host\_name) **else** 'Unknown',

            row.neighbourhood\_group,

            row.neighbourhood,

            row.room\_type,

            float(row.longitude),

            float(row.latitude),

            row.price,

            row.calculated\_host\_listings\_count,

            row.availability\_365,

            row.last\_review **if** **not** pd.isna(row.last\_review) **else** **None**,

            row.reviews\_per\_month **if** **not** pd.isna(row.reviews\_per\_month) **else** **None**

        )

**except** odbc.ProgrammingError **as** e:

        print(**f**"Error occurred for row: {row}")

        print(**f**"Error message: {e}")

**break**

conn.commit()

conn.close()

# load Taxi\_df to Taxi\_staging

# dfs Cleaning

Taxi\_df.isnull().sum()

Taxi\_df\_clean = Taxi\_df[~Taxi\_df.isnull().any(axis=1)]

Taxi\_df\_clean.isnull().sum()

duplicate\_counts = Taxi\_df\_clean.groupby(['id','pickup\_longitude','pickup\_latitude','dropoff\_longitude','dropoff\_latitude']).size().reset\_index(name='Count')

Taxi\_duplicate = duplicate\_counts[duplicate\_counts['Count'] > 1]

print("Duplicate combinations of Columns in Taxi:\n")

print(Taxi\_duplicate['Count'].sum())

print('\n',Taxi\_duplicate)

# Data normalization

Taxi\_df\_clean = Taxi\_df\_clean.copy()

# Transform into datetime data type

Taxi\_df\_clean.loc[:, 'pickup\_datetime'] = pd.to\_datetime(Taxi\_df\_clean['pickup\_datetime'])

Taxi\_df\_clean.loc[:, 'dropoff\_datetime'] = pd.to\_datetime(Taxi\_df\_clean['dropoff\_datetime'])

Taxi\_df\_clean.dtypes

# Filter data

Taxi\_filtered = Taxi\_df\_clean[['id','vendor\_id','pickup\_datetime','dropoff\_datetime',

'passenger\_count','pickup\_longitude','pickup\_latitude',

'dropoff\_longitude','dropoff\_latitude','store\_and\_fwd\_flag',

'trip\_duration']]

Taxi\_filtered.head()

# Transform the latitude and Longitude into Borough and Neighborhood format

!pip install geopy

from geopy.geocoders import Nominatim

from geopy.exc import GeocoderTimedOut

import time

def get\_borough\_neighborhood(latitude, longitude):

geolocator = Nominatim()

location = geolocator.reverse(f"{latitude}, {longitude}")

address = location.raw['address']

borough = address.get('county')

neighborhood = address.get('neighbourhood')

return borough, neighborhood

# set start index and retry limit

start\_index = 0

retry\_limit = 3

# loop through all records

for i in range(start\_index, len(Taxi\_filtered)):

row = Taxi\_filtered.iloc[i]

# retry up to retry\_limit times if timed out

for j in range(retry\_limit):

try:

# get pickup and dropoff borough and neighborhood

pickup\_borough, pickup\_neighborhood = get\_borough\_neighborhood(row['pickup\_latitude'], row['pickup\_longitude'])

dropoff\_borough, dropoff\_neighborhood = get\_borough\_neighborhood(row['dropoff\_latitude'], row['dropoff\_longitude'])

# assign values to dataframe

Taxi\_filtered.at[i, 'pickup\_borough'] = pickup\_borough

Taxi\_filtered.at[i, 'pickup\_neighborhood'] = pickup\_neighborhood

Taxi\_filtered.at[i, 'dropoff\_borough'] = dropoff\_borough

Taxi\_filtered.at[i, 'dropoff\_neighborhood'] = dropoff\_neighborhood

# print progress every 1000 records

if i % 1000 == 0:

print(f"Processed {i} records")

# break out of retry loop if successful

break

# handle timed out exception

except geopy.exc.GeocoderTimedOut:

print(f"Timed out, retrying {j+1}/{retry\_limit}")

time.sleep(1)

# save last processed index to a file or a database

with open('last\_processed\_index.txt', 'w') as f:

f.write(str(i))

# Estiblish the connection

conn = odbc.connect(Trusted\_Connection = 'YES',

Driver = '{ODBC Driver 17 for SQL Server}',

Server = 'DESKTOP-IAN\JLM\_SQLSERVER',

Database = 'CS689\_FinalProj')

print(conn)

# Create Taxi\_staging table

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE Taxi\_staging (

taxi\_id NVARCHAR(255) PRIMARY KEY,

vendor\_id INT,

pickup\_datetime DATETIME,

dropoff\_datetime DATETIME,

passenger\_count FLOAT,

pickup\_longitude FLOAT,

pickup\_latitude FLOAT,

dropoff\_longitude FLOAT,

dropoff\_latitude FLOAT,

store\_and\_fwd\_flag NVARCHAR(255),

trip\_duration FLOAT,

)

""")

conn.commit()

conn.close()

# load data

conn = odbc.connect(Trusted\_Connection = 'YES',

Driver = '{ODBC Driver 17 for SQL Server}',

Server = 'DESKTOP-IAN\JLM\_SQLSERVER',

Database = 'CS689\_FinalProj')

cursor = conn.cursor()

for row in Taxi\_filtered.itertuples():

try:

cursor.execute("""

INSERT INTO Taxi\_staging (taxi\_id, vendor\_id, pickup\_datetime, dropoff\_datetime,

passenger\_count, pickup\_longitude, pickup\_latitude,

dropoff\_longitude, dropoff\_latitude, store\_and\_fwd\_flag, trip\_duration

)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""",

row.id,

row.vendor\_id,

row.pickup\_datetime,

row.dropoff\_datetime,

row.passenger\_count,

row.pickup\_longitude,

row.pickup\_latitude,

row.dropoff\_longitude,

row.dropoff\_latitude,

row.store\_and\_fwd\_flag,

row.trip\_duration,

)

except odbc.ProgrammingError as e:

print(f"Error occurred for row: {row}")

print(f"Error message: {e}")

break

conn.commit()

conn.close()

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

SELECT \* FROM Airbnb\_staging;

A screenshot of a computer

Description automatically generated

In the Python part, we already have filtered some of the data in Airbnb\_df, such as NA value check and columns’ normalization, such as transform the data type of last\_review into dattime format and so on.

A screenshot of a computer code

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with medium confidence

DELETE FROM Airbnb\_staging

WHERE last\_review IS NULL;

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Since last\_review will become an important parameter for us to create the measures between DimDate and FactAirbnb, so we dropped all the NULL values in last\_review.

**Before dropped: 48895 rows.**

**After dropped: 38843 rows.**

A screenshot of a computer

Description automatically generated with medium confidence

Dropping NULL values in dropoff\_datetime

A screenshot of a computer program

Description automatically generated with low confidence

Checking the duplications.

# Filter data

Taxi\_filtered = Taxi\_df\_clean[['id','vendor\_id','pickup\_datetime','dropoff\_datetime',

'passenger\_count','pickup\_longitude','pickup\_latitude',

'dropoff\_longitude','dropoff\_latitude','store\_and\_fwd\_flag',

'trip\_duration']]

Taxi\_filtered.head()

# Estiblish the connection

conn = odbc.connect(Trusted\_Connection = 'YES',

Driver = '{ODBC Driver 17 for SQL Server}',

Server = 'DESKTOP-IAN\JLM\_SQLSERVER',

Database = 'CS689\_FinalProj')

print(conn)

# Create Taxi\_staging table

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE Taxi\_staging (

taxi\_id NVARCHAR(255) PRIMARY KEY,

vendor\_id INT,

pickup\_datetime DATETIME,

dropoff\_datetime DATETIME,

passenger\_count FLOAT,

pickup\_longitude FLOAT,

pickup\_latitude FLOAT,

dropoff\_longitude FLOAT,

dropoff\_latitude FLOAT,

store\_and\_fwd\_flag NVARCHAR(255),

trip\_duration FLOAT,

)

""")

conn.commit()

conn.close()

# load data

conn = odbc.connect(Trusted\_Connection = 'YES',

Driver = '{ODBC Driver 17 for SQL Server}',

Server = 'DESKTOP-IAN\JLM\_SQLSERVER',

Database = 'CS689\_FinalProj')

cursor = conn.cursor()

for row in Taxi\_filtered.itertuples():

try:

cursor.execute("""

INSERT INTO Taxi\_staging (taxi\_id, vendor\_id, pickup\_datetime, dropoff\_datetime,

passenger\_count, pickup\_longitude, pickup\_latitude,

dropoff\_longitude, dropoff\_latitude, store\_and\_fwd\_flag, trip\_duration

)

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

""",

row.id,

row.vendor\_id,

row.pickup\_datetime,

row.dropoff\_datetime,

row.passenger\_count,

row.pickup\_longitude,

row.pickup\_latitude,

row.dropoff\_longitude,

row.dropoff\_latitude,

row.store\_and\_fwd\_flag,

row.trip\_duration,

)

except odbc.ProgrammingError as e:

print(f"Error occurred for row: {row}")

print(f"Error message: {e}")

break

conn.commit()

conn.close()