# Libraries

```
import pandas as pd
import psycopg2
import folium
from IPython.display import display
```

# Extract

```
# Load the CSV file
def extract_data(file_path):
    try:
        data = pd.read_csv(file_path)
        return data
    except Exception as e:
        print(f"Error occurred while reading the file: {e}")
        return None

# Path to CSV
file_path = r'D:\Internship Task\Using Jupyter Notebook\
Data_Before_ETL\Address list - Sheet1.csv'
raw_data = extract_data(file_path)
raw_data.head()
```

```
   id                                          address       lat
lng
0   1       23685 Walton Ave, New Caney, TX 77357 USA  30.15307 -
95.2186
1   2  55 Maine Mall Rd, South Portland, ME 04106 USA  43.64398 -
70.3318
2   3       124 Ossipee Trl E, Standish, ME 04084 USA  43.73120 -
70.5415
3   4           701 N Main St, Harrison, AR 72601 USA  36.23694 -
93.1068
4   5             700 Minot Ave, Auburn, ME 04210 USA  44.08159 -
70.2569
```

# Transform

## Drop Null

```
raw_data.isnull().sum()
```

```
id         0
address    0
lat        0
lng        0
dtype: int64
```

```
raw_data = raw_data.dropna(subset=['address', 'lat', 'lng'])
```

```
raw_data.isnull().sum()
```

```
id         0
address    0
lat        0
lng        0
dtype: int64
```

## Split addresses

The split_address function is designed to break an address into separate parts like house number, city, state, and zip code. It assumes the address follows this format: "House No, City, State ZipCode." First, the house number is taken from the beginning of the address (before the first comma). Next, the city is taken from the part after the first comma. Then, the state and zip code are extracted from the part after the second comma, where the state comes first and the zip code follows. If the address doesn't fit this format or has missing parts, the function handles the error and returns None for each part so that the program doesn't stop working.

Once the split_address function is defined, it is applied to the entire dataset. This is done using the apply() method in pandas, which processes each address row by row. The lambda function helps turn the output of split_address into a pandas Series, which allows each part (house number, city, state, zip code) to be saved into new columns in the dataset.

```
def transform_data(data):

    #Function to split the address into components
    def split_address(address):
        try:
            parts = address.split(',')
            house_no = parts[0].split()[0]
            city = parts[1].strip()
            state_zip = parts[2].strip().split(' ')
            state_code = state_zip[0]
            zip_code = state_zip[1]
            return house_no, city, state_code, zip_code
        except Exception as e:
            print(f"Error splitting address: {e}")
            return None, None, None, None

    #Apply the address splitting function to the dataframe
    data[['house_no', 'city', 'state_code', 'zip_code']] = data.apply(
        lambda row: pd.Series(split_address(row['address'])), axis=1
```

```
    )

    #Sort by state_code and city
    data = data.sort_values(by=['state_code', 'city'])

    return data

#Transform the extracted data
cleaned_data = transform_data(raw_data)

cleaned_data.head()
```

```
        id                                          address
lat  \
375    376       401 W Evergreen Ave, Palmer, AK 99645 USA  61.59941

1900  1901        214 Kirkland St, Abbeville, AL 36310 USA  31.56762

643    644        2517 County Road 36, Akron, AL 35441 USA  32.87059

1418  1419  7959 US Highway 431, Albertville, AL 35950 USA  34.28000

36      37          118 Shelby St, Andalusia, AL 36421 USA  31.32004


           lng house_no         city state_code zip_code
375  -149.1140      401       Palmer         AK    99645
1900  -85.2512      214    Abbeville         AL    36310
643   -87.7213     2517        Akron         AL    35441
1418  -86.2170     7959  Albertville         AL    35950
36    -86.4623      118    Andalusia         AL    36421
```

```
cleaned_data.to_csv(r'D:\Internship Task\Using Jupyter Notebook\
Data_After_ETL\Transformed_data.csv', index=False)
```

# Load

## Connect to the PostgreSQL database

```
def connect_to_db():
    try:
        #my database props
        conn = psycopg2.connect(
            host="localhost",
            database="postgres",
            user="postgres",
            password="admin"
        )
        return conn
```

```
    except Exception as e:
        print(f"Error connecting to the database: {e}")
        return None
```

## Create PostgreSQL table 'Addresses'

```
# create table as given in the instructions
def create_table(conn):
    with conn.cursor() as cursor:
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS addresses (
                id SERIAL PRIMARY KEY,
                house_no VARCHAR(10),
                city VARCHAR(50),
                state_code VARCHAR(2),
                zip_code VARCHAR(10),
                lat DOUBLE PRECISION,
                lng DOUBLE PRECISION
            );
        ''')
        conn.commit()
```

## Insert data into the table

```
def insert_data(conn, data):
    with conn.cursor() as cursor:
        for _, row in data.iterrows():
            cursor.execute('''
                INSERT INTO addresses (house_no, city, state_code,
zip_code, lat, lng)
                VALUES (%s, %s, %s, %s, %s, %s)
            ''', (row['house_no'], row['city'], row['state_code'],
row['zip_code'], row['lat'], row['lng']))
        conn.commit()
```

## Function call

```
conn = connect_to_db()

if conn:
    create_table(conn)
    insert_data(conn, cleaned_data)
    print("Connection successful ")
    conn.close()
else:
    print("Failed to connect to Local Postgresql.")
```

```
Connection successful
```

# Bonus

## SQL query From Local PostgreSQL

```
# Function to execute the SQL query and fetch results
def get_state_address_counts():
    conn = connect_to_db()
    cursor = conn.cursor()

    # SQL query to count addresses for each state
    query = '''
        SELECT state_code, COUNT(*) AS address_count
        FROM addresses
        GROUP BY state_code
        ORDER BY state_code;
    '''

    # Execute the query
    cursor.execute(query)

    # Fetch the result
    results = cursor.fetchall()

    # Close the connection
    cursor.close()
    conn.close()

    return results

# Fetch the results
state_address_counts = get_state_address_counts()

# Convert the results into a pandas DataFrame
df = pd.DataFrame(state_address_counts, columns=['State Code',
'Address Count'])

# Save the DataFrame to a CSV file
csv_file_path = r'D:\Internship Task\Using Jupyter Notebook\Bonus\
state_address_counts(bonus query from local PostgreSQL).csv'
df.to_csv(csv_file_path, index=False)

df.head()
```

| | State Code | Address Count |
|---|---|---|
| 0 | AK | 6 |
| 1 | AL | 678 |
| 2 | AR | 372 |
| 3 | AZ | 36 |
| 4 | CA | 36 |

## Display the addresses From Local PostgreSQL

```
# Fetch address data (latitude and longitude) from the PostgreSQL
database
def fetch_address_data():
    conn = connect_to_db()
    cursor = conn.cursor()
    cursor.execute('SELECT lat, lng, house_no, city, state_code,
zip_code FROM addresses')
    rows = cursor.fetchall()  # Fetch all rows
    cursor.close()
    conn.close()
    return rows

# Generate the Folium map with the address data and display it in the
notebook
def display_map(address_data):

    folium_map = folium.Map(location=[40.7128, -74.0060],
zoom_start=5)

    # Loop through the address data and add each location to the map
    for address in address_data:
        lat, lng, house_no, city, state_code, zip_code = address
        # Create a popup description for each marker
        popup_text = f'{house_no}, {city}, {state_code} {zip_code}'
        folium.Marker(location=[lat, lng],
popup=popup_text).add_to(folium_map)

    # Display the map inline in the notebook
    display(folium_map)

# Fetch the data
address_data = fetch_address_data()

# Display the map
display_map(address_data)
```

```
<folium.folium.Map at 0x18e3372f7a0>
```

# AWS RDS

```
# Before this connections I create account on AWS and create RDS
postgresql
```

## Connect to the AWS RDS PostgreSQL database

```
import psycopg2
```

```
# Connect to the PostgreSQL database on AWS RDS
def connect_to_rds():
    conn = psycopg2.connect(
        # RDS credentioal
        host="postgres.czo0ge886dzc.eu-north-1.rds.amazonaws.com",
        database="postgres",
        user="postgres",
        password="waqasdost",
        port="5432"
    )
    return conn


conn = connect_to_rds()
if conn:
    print("Connection successful ")
    conn.close()
else:
    print("Failed to connect to RDS.")
```

```
Connection successful
```

## Create Table on RDS

```
# Function to create the PostgreSQL table
def create_table():
    conn = connect_to_rds()
    with conn.cursor() as cursor:
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS addresses (
                id SERIAL PRIMARY KEY,
                house_no VARCHAR(10),
                city VARCHAR(50),
                state_code VARCHAR(2),
                zip_code VARCHAR(10),
                lat DOUBLE PRECISION,
                lng DOUBLE PRECISION
            );
        ''')
        conn.commit()
    conn.close()
    print("Table 'addresses' created")

create_table()
```

```
Table 'addresses' created
```

## Load to RDS

```
def load_data_to_rds(data):
    conn = connect_to_rds()
    with conn.cursor() as cursor:
        for index, row in data.iterrows():
            cursor.execute('''
                INSERT INTO addresses (house_no, city, state_code,
zip_code, lat, lng)
                VALUES (%s, %s, %s, %s, %s, %s)
            ''', (row['house_no'], row['city'], row['state_code'],
row['zip_code'], row['lat'], row['lng']))
        conn.commit()
    conn.close()
    print("Data successfully loaded to AWS")

# Example usage with cleaned data
load_data_to_rds(cleaned_data)
```

```
Data successfully loaded to AWS
```

# Bonus

## SQL query From AWS RDS

```
# Function to execute the SQL query and fetch results
def get_state_address_counts():
    conn = connect_to_db()
    cursor = conn.cursor()

    # SQL query to count addresses for each state
    query = '''
        SELECT state_code, COUNT(*) AS address_count
        FROM addresses
        GROUP BY state_code
        ORDER BY state_code;
    '''

    # Execute the query
    cursor.execute(query)

    # Fetch the result
    results = cursor.fetchall()

    # Close the connection
    cursor.close()
    conn.close()
```

```
    return results

# Fetch the results
state_address_counts = get_state_address_counts()

# Convert the results into a pandas DataFrame
df = pd.DataFrame(state_address_counts, columns=['State Code',
'Address Count'])

# Save the DataFrame to a CSV file
csv_file_path = r'D:\Internship Task\Using Jupyter Notebook\Bonus\
state_address_counts(bonus query from AWS).csv'
df.to_csv(csv_file_path, index=False)

df.head()
```

| | State Code | Address Count |
|---|---|---|
| 0 | AK | 6 |
| 1 | AL | 678 |
| 2 | AR | 372 |
| 3 | AZ | 36 |
| 4 | CA | 36 |

## Display the addresses From AWS RDS

```
# Fetch address data (latitude and longitude) from the PostgreSQL
database
def fetch_address_data():
    conn = connect_to_db()
    cursor = conn.cursor()
    cursor.execute('SELECT lat, lng, house_no, city, state_code,
zip_code FROM addresses')
    rows = cursor.fetchall()  # Fetch all rows
    cursor.close()
    conn.close()
    return rows

# Generate the Folium map with the address data and display it in the
notebook
def display_map(address_data):

    folium_map = folium.Map(location=[40.7128, -74.0060],
zoom_start=5)

    # Loop through the address data and add each location to the map
    for address in address_data:
        lat, lng, house_no, city, state_code, zip_code = address
        # Create a popup description for each marker
        popup_text = f'{house_no}, {city}, {state_code} {zip_code}'
```

```
        folium.Marker(location=[lat, lng],
popup=popup_text).add_to(folium_map)

    # Display the map inline in the notebook
    display(folium_map)

# Fetch the data
address_data = fetch_address_data()

# Display the map
display_map(address_data)
```

```
<folium.folium.Map at 0x18e31237ef0>
```