# Import required libraries  
from sqlalchemy import create\_engine  
import pandas as pd  
import configparser  
from urllib.parse import quote\_plus  
import os

# Define configuration reader - create a definition for read the config.ini file and look for 'postgresql' section in file  
def get\_db\_config():  
 config = configparser.ConfigParser()  
 config.read('config.ini')  
 return config['postgresql-sqlda2']  
  
db\_config = get\_db\_config()

#### 1. Copy the data from the public transportation dataset to the ZoomZoom customer database by importing this data into a new table in the ZoomZoom database.

# Read CSV File  
current\_path = os.getcwd() # get the current working directory  
file\_name = "public\_transportation\_statistics\_by\_zip\_code.csv"  
csv\_full\_path = os.path.join(current\_path, file\_name) # use os join function to create csv full path  
  
csv\_data = pd.read\_csv(  
 csv\_full\_path, dtype={'zip\_code': str} # make sure the zip\_code column is read as a string   
)

# take a look at data using head() for first 5 rows  
csv\_data.head()

zip\_code public\_transportation\_pct public\_transportation\_population  
0 01379 3.3 13  
1 01440 0.4 34  
2 01505 0.9 23  
3 01524 0.5 20  
4 01529 1.8 32

# Set up database connection  
cnxn\_string = (   
 "postgresql+psycopg2://{username}:{pswd}@{host}:{port}/{database}"   
)  
engine = create\_engine(   
 cnxn\_string.format(   
 username=db\_config['user'],   
 pswd=quote\_plus(db\_config['password']), #use quote\_plus for complex passwords  
 host=db\_config['host'],   
 port=db\_config['port'],   
 database=db\_config['database']   
 )   
)

# load data to sql  
csv\_data.to\_sql (  
 name='public\_transportation\_by\_zip',  
 con=engine,  
 if\_exists='replace',  
 index=False,  
 method='multi',  
 chunksize=10000  
)

33120

#### 2. Find the maximum and minimum values of public\_transportation\_pct in this data. Values less than 0 will most likely be missing data.

# max from csv data  
max\_pct = csv\_data.public\_transportation\_pct.max()  
print("CSV max\_pct: ", max\_pct)

CSV max\_pct: 100.0

# min from csv data  
min\_pct = csv\_data.public\_transportation\_pct.min()  
print("CSV min\_pct: ", min\_pct)

CSV min\_pct: -666666666.0

# compare with data in postgresSql  
query = """  
 select  
 max(public\_transportation\_pct) as max\_pct,  
 min(public\_transportation\_pct) as min\_pct  
 from public\_transportation\_by\_zip   
"""  
  
sql\_data = pd.read\_sql\_query(query, engine)  
  
sql\_data

max\_pct min\_pct  
0 100.0 -666666666.0

the result from the min/max of the csv are the same as the the data from SQL, we can also deduce that the min value is wrong since percentage should be 0 and not negative values this would show that some of the data is potentially missing or invalid.

#### 3. Calculate the average sales amounts for customers that live in high public transportation usage regions (over 10%) as well as low public transportation usage regions (less than, or equal to, 10%).

from sqlalchemy import text # was having issues executing CTE without this  
  
avg\_sales\_query = text("""  
 WITH customer\_transport AS (  
 SELECT  
 c.customer\_id,  
 pt.zip\_code,  
 CASE  
 WHEN pt.public\_transportation\_pct > 10 THEN 'High Usage (>10%)'  
 WHEN pt.public\_transportation\_pct <= 10 AND pt.public\_transportation\_pct >= 0 THEN 'Low Usage (≤10%)'  
 ELSE 'Invalid Data'  
 END AS transport\_usage\_category  
 FROM customers c  
 LEFT JOIN public\_transportation\_by\_zip pt ON c.postal\_code = pt.zip\_code  
 ),  
 sales\_summary AS (  
 SELECT  
 ct.transport\_usage\_category,  
 COUNT(DISTINCT ct.customer\_id) as number\_customers,  
 AVG(s.sales\_amount) as avg\_sale\_amount  
 FROM customer\_transport ct  
 LEFT JOIN sales s ON ct.customer\_id = s.customer\_id  
 WHERE ct.transport\_usage\_category != 'Invalid Data'  
 GROUP BY ct.transport\_usage\_category  
 )  
 SELECT  
 transport\_usage\_category,  
 number\_customers,  
 ROUND(avg\_sale\_amount::numeric, 2) as avg\_sale\_amount  
 FROM sales\_summary  
 ORDER BY transport\_usage\_category;  
""")  
  
avg\_sales\_data = pd.read\_sql\_query(avg\_sales\_query, engine)  
  
avg\_sales\_data

transport\_usage\_category number\_customers avg\_sale\_amount  
0 High Usage (>10%) 3545 6501.98  
1 Low Usage (≤10%) 16067 7209.68

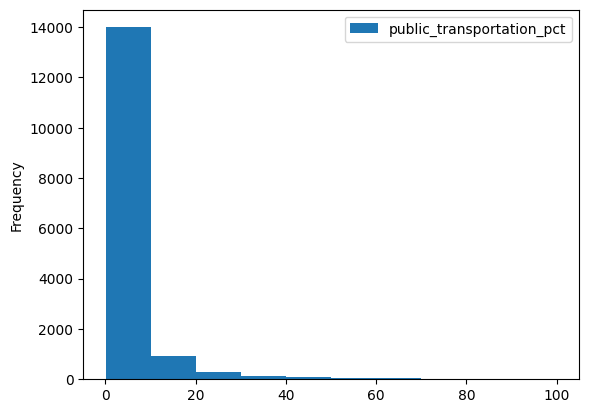
#### 4. Read the data into pandas and plot a histogram of the distribution (Hint: you can use my\_data.plot.hist(y='public\_transportation\_pct') to plot a histogram if you read the data into a my\_data pandas DataFrame).

# get data for histogram, remove null is any, get only data greater than 0  
query = """  
 select  
 \*  
 from public\_transportation\_by\_zip  
 where  
 coalesce(public\_transportation\_pct, 0) > 0   
"""  
  
my\_data = pd.read\_sql\_query(query, engine)  
my\_data

zip\_code public\_transportation\_pct public\_transportation\_population  
0 01379 3.3 13  
1 01440 0.4 34  
2 01505 0.9 23  
3 01524 0.5 20  
4 01529 1.8 32  
... ... ... ...  
15533 61604 3.9 528  
15534 61614 1.0 142  
15535 61721 0.6 2  
15536 61732 0.3 3  
15537 61275 1.0 19  
  
[15538 rows x 3 columns]

my\_data.plot.hist(y='public\_transportation\_pct')

<Axes: ylabel='Frequency'>



#### 5. Using pandas, test using the to\_sql function with and without the method=psql\_insert\_COPY parameter. How do the speeds compare? (Hint: in a Jupyter notebook, you can add %time in front of your command to see how long it takes to execute the code.)

%time   
my\_data.to\_sql (  
 name='public\_transportation\_by\_zip\_test1',  
 con=engine,  
 if\_exists='replace',  
 index=False,  
 chunksize=10000  
)

CPU times: total: 0 ns  
Wall time: 0 ns

1538

%time   
my\_data.to\_sql (  
 name='public\_transportation\_by\_zip\_test2',  
 con=engine,  
 if\_exists='replace',  
 index=False,  
 method='multi',  
 chunksize=10000  
)

CPU times: total: 0 ns  
Wall time: 1 ms

15538

The multi method is significantly faster and actually imported all the data. Not using multi resulted in a partial import that while it looked faster on a per-record basis would be 4 times faster. Trying to use psql\_insert\_copy result in invalid parameter, this seems to have been replaced by multi based on Google search.

#### 6. Group customers based on their zip code public transportation usage rounded to the nearest 10% and look at the average number of transactions per customer. Export this data to Excel and create a scatterplot to better understand the relationship between public transportation usage and sales.

from sqlalchemy import text # was having issues executing CTE without this  
  
research\_query = text("""  
 WITH customer\_transactions AS (  
 SELECT  
 c.customer\_id,  
 c.postal\_code,  
 COUNT(s.sales\_transaction\_date) as num\_transactions  
 FROM customers c  
 LEFT JOIN sales s ON c.customer\_id = s.customer\_id  
 GROUP BY c.customer\_id, c.postal\_code  
),  
 grouped\_data AS (  
 SELECT  
 ROUND(pt.public\_transportation\_pct / 10.0) \* 10 as transport\_usage\_pct,  
 AVG(ct.num\_transactions) as avg\_transactions\_per\_customer,  
 COUNT(DISTINCT ct.customer\_id) \* 1.0 as number\_of\_customers  
 FROM customer\_transactions ct  
 JOIN public\_transportation\_by\_zip pt ON ct.postal\_code = pt.zip\_code  
 GROUP BY ROUND(pt.public\_transportation\_pct / 10.0) \* 10  
 )  
SELECT  
 transport\_usage\_pct,  
 ROUND(avg\_transactions\_per\_customer, 2) as avg\_transactions\_per\_customer,  
 number\_of\_customers  
FROM grouped\_data  
where transport\_usage\_pct > 0  
ORDER BY transport\_usage\_pct;  
""")  
  
research\_data = pd.read\_sql\_query(research\_query, engine)  
  
research\_data

transport\_usage\_pct avg\_transactions\_per\_customer number\_of\_customers  
0 10.0 0.73 3430.0  
1 20.0 0.81 1171.0  
2 30.0 0.88 497.0  
3 40.0 0.76 274.0  
4 50.0 1.05 136.0  
5 60.0 0.99 167.0  
6 70.0 1.15 164.0  
7 80.0 0.76 38.0

# output data to folder  
current\_path = os.getcwd() # get the current working directory  
out\_file\_name = "sales\_vs\_public\_transport\_pct.csv"  
out\_csv\_full\_path = os.path.join(current\_path, out\_file\_name) # use os join function to create csv full path  
  
research\_data.to\_csv(out\_csv\_full\_path, index=False, mode='w')