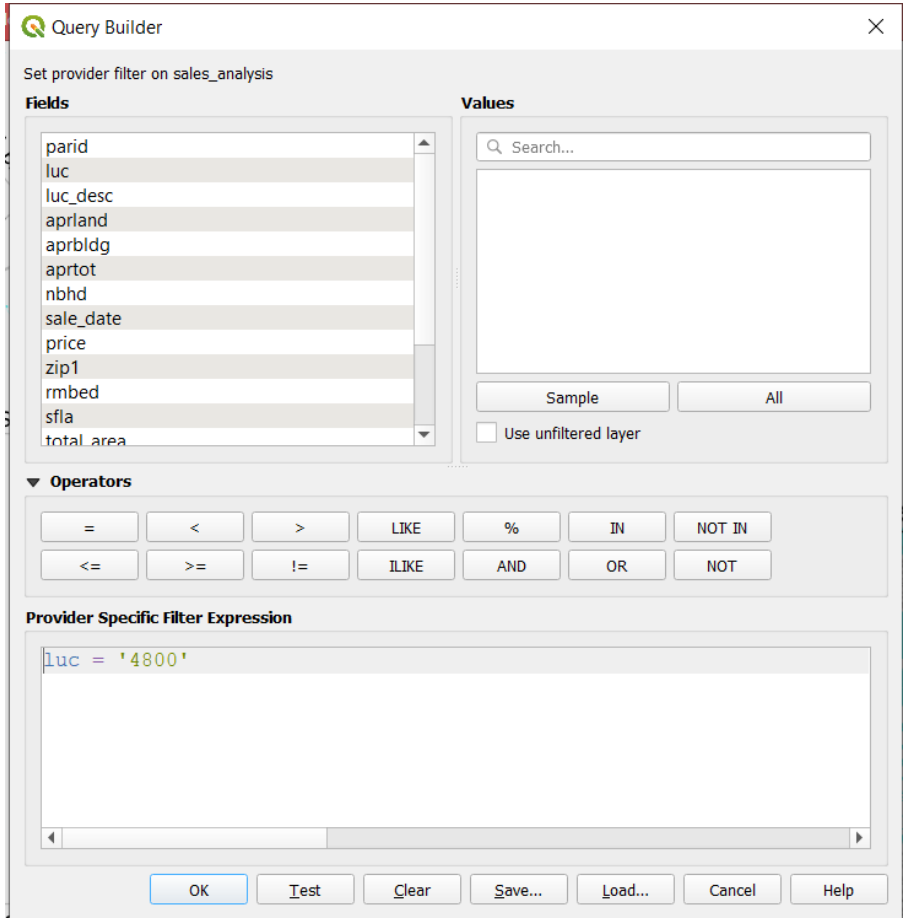
Add new column with machine learning price prediction

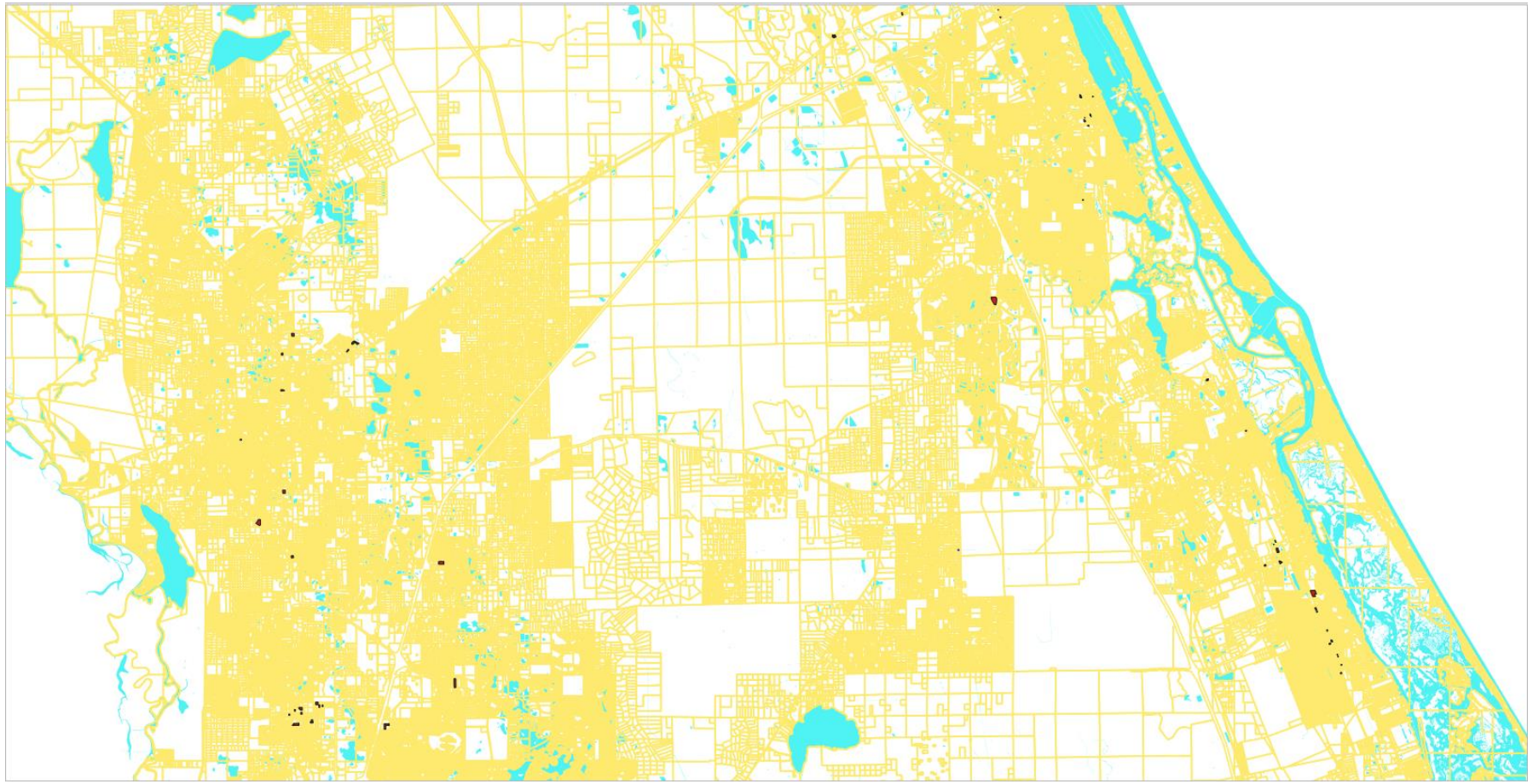
Column of parcel distance to the nearest warehouse/distribution center and how the distance would affect the price

CS540 Mike Luo

Make sure the parcel filter to only show warehouse



The dark spots are where warehouses/distributions located



Now we know warehouse/distribution center code is 4800
-- all the sql codes can be found in my git as new_column_sql

```
1  select luc, luc_desc from volusia.sales_analysis where luc = '4800' limit 1;
```

Data Output

	luc text	luc_desc text
1	4800	Warehousing, Distribution

Find 5 nearest warehouse/distribution center given parcel# 3565215

```
1  select p.parid, p.luc, p.luc_desc, ST_Distance(p.geom, (select p2.geom from volusia.parcel p2 where parid=3565215))/5280
2  from volusia.parcel p
3  where p.luc='4800'
4  order by p.geom <-> (select p2.geom from volusia.parcel p2 where parid=3565215)
5  limit 5;
```

Data Output

	parid double precision	luc text	luc_desc text	?column? double precision
1	3559801	4800	Warehousing...	0.14848992147650675
2	4842471	4800	Warehousing...	0.42011915991322035
3	3559223	4800	Warehousing...	0.42121781444972284
4	3558766	4800	Warehousing...	0.460797737941563
5	3560213	4800	Warehousing...	0.49700593428036044

Find 1 nearest warehouse/distribution center given parcel# 2004291

```
1  select p.parid, p.geom, ST_Distance(p.geom, (select p2.geom from volusia.parcel p2 where p2.parid=2004291))/5280
2  from volusia.parcel p
3  where p.luc='4800'
4  order by p.geom <-> (select p2.geom from volusia.parcel p2 where p2.parid=2004291) limit 1;
```

Data Output

	parid double precision	geom geometry	?column? double precision
1	2009471	0106000020BC080...	1.8422664980065455

Add a column to the parcel table, and update to find distance from each parcel to the nearest distribution center

```
alter table volusia.parcel add column gcdistance double precision;
```

Add geometry column

```
SELECT AddGeometryColumn ('volusia','parcel','geom',2236,'MULTIPOLYGON',2);
update volusia.parcel a set geom = p.geom from volusia.gis_parcels p where a.parid=p.altkey;
```

Distance Testing

```
1  update volusia.parcel p1 set gcdistance = ST_Distance(p1.geom, p2.geom)/5280
2  from volusia.parcel p2 where p1.parid=2004291 and p2.parid=5685331;
```

Messages

```
UPDATE 1

Query returned successfully in 77 msec.
```

Create index for the records which will be used for loop through the distance between each parcel and the nearest warehouse

```
create index idx_parcel_luc on volusia.parcel (luc);
create index idx_parcel on volusia.parcel (parid);

CREATE INDEX parcel_geom_idx
  ON volusia.parcel
  USING GIST (geom);
```

Loop through the records to calculate the distance between each parcel and the nearest warehouse/distribution center
loop code can be found in my git as new_column_loop.ipynb

In [1]:
```
import psycopg2
import re
import matplotlib.pyplot as plt
import os
import pandas as pd
```

In [2]:
```
try:
    conn = psycopg2.connect("dbname='spatial' user='postgres' host='localhost' password='111111'")
except:
    print("cant connect to the database")
```

In [3]:
```
cur = conn.cursor()
cur2 = conn.cursor()
cur3 = conn.cursor()
input_altkey = 3565215
```

In [4]:
```
sql = "select parid::integer from volusia.parcel p where geom is not null" # limit 10"
print('SQL: ', sql)
cur.execute(sql)
```

```
SQL:  select parid::integer from volusia.parcel p where geom is not null
```

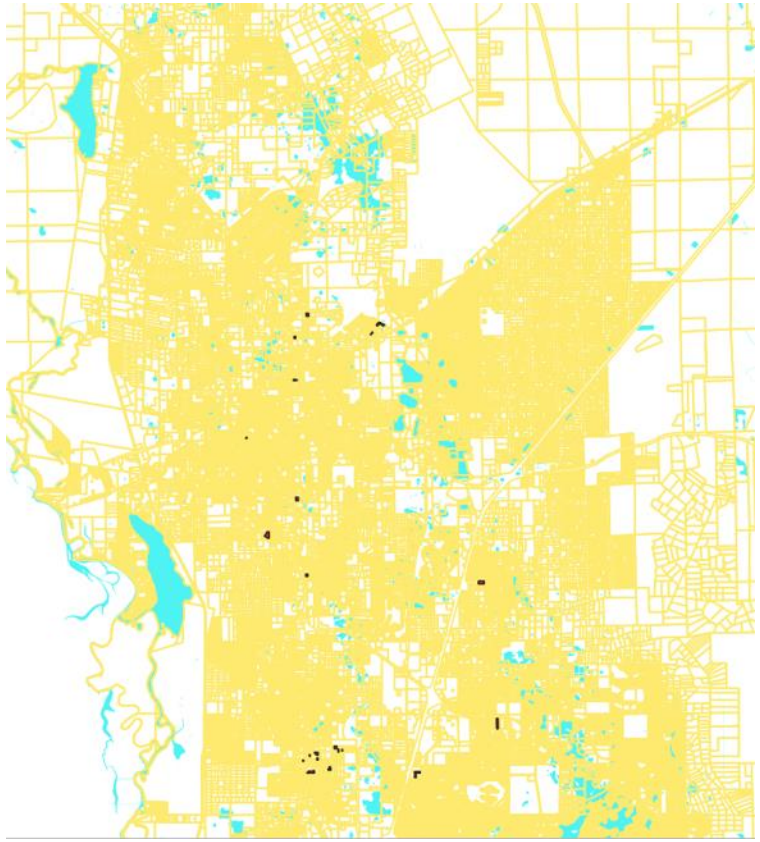
In [5]:
```
i=0
row = cur.fetchone()
while row is not None:
    i = i + 1
    parid = str(row[0])
    sql2 = "select p.parid::integer, p.geom, ST_Distance(p.geom, (select p2.geom from volusia.parcel p2 where p2.parid=" + parid + "))/528(
    ###sql2 = "select p.parid::integer, p.geom, ST_Distance(p.geom, (select p2.geom from volusia.parcel p2 where p2.parid="
    ### + parid + "))/5280  from volusia.parcel p where p.luc='4800' order by p.geom <-> (select p2.geom
    ### from volusia.parcel p2 where p2.parid=" + parid + ") limit 1;"
    cur2.execute(sql2)
    row2 = cur2.fetchone()
    parid2 = str(row2[0])
    distance = row2[2]
    sql3 = "update volusia.parcel p1 set gcdistance = " + str(distance) + " where p1.parid=" + parid + ";"
    cur3.execute(sql3)
    # print(sql3)
    if i%10000 == 0:
        print(i)
        conn.commit()
    row = cur.fetchone()
```

Gcdistance column created

geom geometry	gcdistance double precision
0106000020BC080...	1.7569501948564528
0106000020BC080...	1.7709894709927194
0106000020BC080...	1.7847247850418178
0106000020BC080...	1.8037865003509823
0106000020BC080...	1.8213035621088727
0106000020BC080...	1.8144891114292105
0106000020BC080...	1.8056495592419344
0106000020BC080...	1.7968369949839174

Now move to Python, and use linear regression to predict if the house price is actually affected by the gcdistance from the nearest warehouse/distribution center
regression code can be found in my git as regression.ipynb

As mentioned in README, my prediction specifically focused on the greater Deland area, as I could identify a price linear trend between the parcel and the nearest warehouse/distribution center while other beachside cities were not identified this trend



In [215]:
```
##the packages we needed
import pandas as pd
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import statsmodels.api as sm
import numpy as np
import seaborn as sns

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, train_test_split
```

In [216]:
```
##create SQL connection
engine = create_engine('postgresql://postgres@localhost:5432/spatial')
df = pd.read_sql_query('select parcel.gcdistance, sales_analysis.price from volusia.parcel, volusia.sales_analysis where sales_analysis.par
##location based on (541524,1695784,575191,1743390), which is the great Deland area
##or simply use deland.csv in my git
## df = pd.read_csv('your path||deland.csv')
```

In [217]:
```
df.head()
##connection success
```

Out[217]:

	gcdistance	price
0	1.441824	84000.0
1	1.118934	112000.0
2	0.615752	225000.0
3	0.332137	204600.0
4	0.332137	180000.0

In [218]:
```
df['gcdistance']
```

Out[218]:
```
0       1.441824
1       1.118934
2       0.615752
3       0.332137
4       0.332137
          ...
4897    0.266116
4898    0.272944
4899    0.281834
4900    0.281834
4901    0.444969
Name: gcdistance, Length: 4902, dtype: float64
```

In [170]:
```
##we want to remove the null values for the modeling purpose
df = df[df['gcdistance'].notna()]
```

In [171]:
```
##all good
df['gcdistance'].isnull().values.any()
```

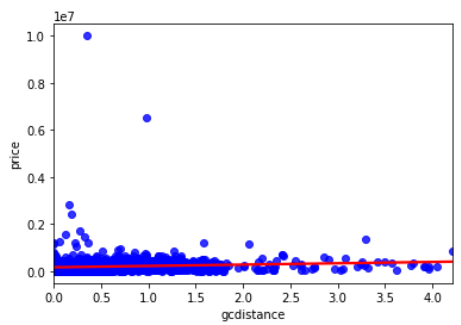
Out[171]:
```
False
```

In [172]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4771 entries, 0 to 4901
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   gcdistance  4771 non-null   float64
 1   price       4771 non-null   float64
dtypes: float64(2)
memory usage: 111.8 KB
```

In [173]:
```
##intial distribution, outliers identified
dfx = df['gcdistance']
dfy = df['price']
sns.regplot(x=dfx,y=dfy,data=df, scatter_kws={"color": "blue"}, line_kws={"color": "red"}, fit_reg=True)
```

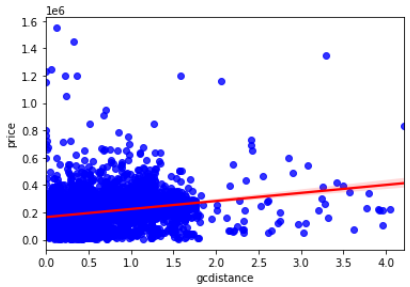
Out[173]:
```
<AxesSubplot:xlabel='gcdistance', ylabel='price'>
```



In [174]:
```
##remove them
q_hi = df['price'].quantile(0.999)
q_low = df['price'].quantile(0.001)
df = df[(df['price']>q_low) & (df['price']<q_hi)]
```

In [175]:
```
##a bit better, all we can see the trend is indicating further the distance higher the price
dfx = df['gcdistance']
dfy = df['price']
sns.regplot(x=dfx,y=dfy,data=df, scatter_kws={"color": "blue"}, line_kws={"color": "red"}, fit_reg=True)
```

Out[175]: <AxesSubplot:xlabel='gcdistance', ylabel='price'>



In [176]:
```
X = df[['gcdistance']]
```

In [177]:
```
y = df[['price']]
```

In [178]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                    test_size = 0.3, random_state = 666)
```

In [179]:
```
OLSmodel = sm.OLS(y_train, X_train).fit()
OLS_y_pred = OLSmodel.predict(X_test)

print_OLSmodel = OLSmodel.summary()
print(print_OLSmodel)

msq_pred = np.sqrt(mean_squared_error(y_test,OLS_y_pred))
r2_pred = r2_score(y_test, OLS_y_pred)

print('msq: ', msq_pred)
print('r2: ' , r2_pred)
```

OLS Regression Results

Dep. Variable:	price	R-squared (uncentered):	0.573
Model:	OLS	Adj. R-squared (uncentered):	0.573
Method:	Least Squares	F-statistic:	4468.
Date:	Fri, 30 Apr 2021	Prob (F-statistic):	0.00
Time:	15:37:39	Log-Likelihood:	-44517.
No. Observations:	3332	AIC:	8.904e+04
Df Residuals:	3331	BIC:	8.904e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
gcdistance	2.236e+05	3344.542	66.842	0.000	2.17e+05	2.3e+05

Omnibus:	1041.058	Durbin-Watson:	1.644
Prob(Omnibus):	0.000	Jarque-Bera (JB):	24199.671
Skew:	0.938	Prob(JB):	0.00
Kurtosis:	16.068	Cond. No.	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
msq: 154753.95026228455
r2: -0.7426821779109691

Conclusion: as the outliers of the dataset is being reduced, the trend is that in Deland area, whenever the distance between a parcel and the nearest warehouse/distribution center increase by 4 units, the price would increase by 0.2 unit, while the model accuracy is concerning since the error is larger than $154,753 and the residual of the model turned out negative.

For future analyses, the introduction of complex machine learning model is necessary since the variance of the simple linear model is large which the model result is not satisfying, while considering overfitting and noise as a possible issue in the future analyses, it is also necessary to tune the hyperparameters of the future models.