Unsupervised anomaly detection (fraud) algorithm

We first do some data cleaning (exclusions, imputation, don't remove outliers - that's what we're looking for), then build variables that are designed to look for the kinds of anomalies we are interested in, in this case, unusual property valuations.

After we build the variables we know we have lots of correlations and too high dimensionality so we need to remove correlations and reduce dimensionality. Since we don't have a dependent variable the easiest useful thing to do is PCA. We z scale (always z scale before a PCA), do PCA, keep the top few PCs, then z scale again in order to make each retained PC equally important (optional step; only do this if you keep just a few PCs.).

We use two different anomaly detection (fraud) algorithms. The first just looks for outliers in the final scaled PC space using a Minkowski distance from the origin. Since we did zscale, PCA, zscale, this Minkowski distance is essentially a Mahalanobis distance for power 2 in the Minkowski formula. The second anomaly detection method trains a simple autoencoder, and the fraud score is then the reproduction error. It's important to note that each/either of these two methods would be a fine fraud score by itself.

Since we have two scores and we don't really know which one is better we just average the two scores. To do this we replace the score with its rank order and then average the rank-ordered scores for our final score.

Lastly we sort all the records by this final score and explore the top n records. To help the investigation we show which of the variables are driving these top scoring records with a heat map of the zscores of the variables, which can point the investigators to what's making the score high for these top scoring records.

This problem is an invented problem to demonstrate the process of building unsupervised fraud models. The data set is real and the invented problem is realistic. What's lacking the most is the ability to interact with domain experts in order to do proper exclusions and design good/appropriate variables.

The data can be found here: https://data.cityofnewyork.us/Housing-Development/Property-Valuation-and-Assessment-Data/rgy2-tti8

In [1]:
```
from datetime import datetime
from sklearn.neural_network import MLPRegressor
import pandas as pd
import numpy as np
import scipy.stats as sps
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
%matplotlib inline
start_time = datetime.now()
```

In [2]:
```
%%time
data = pd.read_csv('NY property data.csv')
NY_data_orig = data.copy()
data.shape
```

```
CPU times: user 1.35 s, sys: 143 ms, total: 1.5 s
Wall time: 1.53 s
```

Out[2]:
```
(1070994, 32)
```

In [3]:
```
data.dtypes
```

Out[3]:
```
RECORD        int64
BBLE         object
BORO          int64
BLOCK         int64
LOT           int64
EASEMENT     object
OWNER        object
BLDGCL       object
TAXCLASS     object
LTFRONT       int64
LTDEPTH       int64
EXT          object
STORIES     float64
FULLVAL     float64
AVLAND      float64
AVTOT       float64
EXLAND      float64
EXTOT       float64
EXCD1       float64
STADDR       object
ZIP         float64
EXMPTCL      object
BLDFRONT      int64
BLDDEPTH      int64
AVLAND2     float64
AVTOT2      float64
EXLAND2     float64
EXTOT2      float64
EXCD2       float64
PERIOD       object
YEAR         object
VALTYPE      object
dtype: object
```

In [4]:
```
data.head()
```

Out[4]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	BLDFRONT	BLDDEPTH	AVLAND2	AV
0	1	1000010101	1	1	101	NaN	U S GOVT LAND & BLDGS	P7	4	500	...	0	0	3775500.0	8613
1	2	1000010201	1	1	201	NaN	U S GOVT LAND & BLDGS	Z9	4	27	...	0	0	11111400.0	80690
2	3	1000020001	1	2	1	NaN	DEPT OF GENERAL SERVI	Y7	4	709	...	709	564	32321790.0	40179
3	4	1000020023	1	2	23	NaN	DEPARTMENT OF BUSINES	T2	4	793	...	85	551	13644000.0	15750
4	5	1000030001	1	3	1	NaN	PARKS AND RECREATION	Q1	4	323	...	89	57	106348680.0	107758

5 rows × 32 columns

Remove some properties that we aren't interested in

In [5]:
```
numrecords_orig = len(data)
numrecords = numrecords_orig
numrecords
```

Out[5]: 1070994

In [6]:
```
# remove the records with easement type as goverment
data = data[data["EASEMENT"] != "U"].reset_index(drop=True)
numremoved = numrecords - len(data)
print('# records removed:', numremoved)
```

```
# records removed: 1
```

In [7]:
```
# create some words for the owner name that might be goverment or a cemetery
gov_list = ['DEPT ', 'DEPARTMENT', 'UNITED STATES','GOVERNMENT',' GOVT ', 'CEMETERY']

# owner = list(set(data['OWNER'].to_list()))
# owner.pop(0) #remove the nan

owner1 = list(set(data['OWNER'].to_list()))
owner = [item for item in owner1 if str(item) != 'nan'] # remove any nan's

remove_list = []
print("Total owner number before removing is ", len(owner))

for i in owner:
   for g in gov_list:
    if g in i and 'STORES' not in i:
        remove_list.append(i)
```

```
Total owner number before removing is  863347
```

In [8]:
```
remove_list # check all the name here and edit if it is not a goverment name
```

```
Out[8]: ['POLICE DEPARTMENT',
 'NYS DEPT OF ENVIRONME',
 'DEPARTMENT OF CULTURA',
 'DEPARTMENT FOR THE AG',
 'WASHINGTON CEMETERY C',
 'GOVERNMENT OF BARBADO',
 'GOVERNMENT/MALAYSIA',
 'GOVERNMENT REP FRANCE',
 'NYS DEPT TRANSPORTATI',
 'FEDERAL GOVERNMENT (G',
 'GOVERNMENT OF BRUNEID',
 'N Y C DEPT OF HIGHWAY',
 'MT ZION CEMETERY',
 'GOVERNMENT OF THE UNI',
 'CEMETERY OF THE EVERG',
 'STATE OF NY DEPT P W',
 'DEPARTMENT OF BUSINES',
 'ST JOHNS CEMETERY',
 'U S GOVT POST OFFIC',
 'GOVERNMENT/FEDERAL RE',
 'ST MARYS CEMETERY',
 'THE WOODLAWN CEMETERY',
 'NYC DEPT OF W S G E',
 'NYC MARBLE CEMETERY',
 'NYC DEPT OF PUB WORKS',
 'DEPARTMENT OF HPD',
 'DEPARTMENT OF HEALTH',
 'GOVERNMENT REP KOREA',
 'UNITED STATES A VA',
 'NYS DEPT PUB WORKS',
 'U S GOVT NAVY',
 'DEPT OF TRANSPORTATIO',
 'US DEPT OF HUD-MF/REO',
 'PARKS DEPARTMENT',
 'EVERGREEN CEMETERY',
 'GOVERNMENT OF TURKEY',
 'NYC DEPT OF R E',
 'U.S. DEPARTMENT OF HU',
 'LAWRENCE CEMETERY',
 'NYC DEPT OF PUB WKS',
 'NYC DEPARTMENT OF HOM',
 'US GOVT POST OFFICE',
 'STATE OF N Y DEPT TRA',
 "GOVERNMENT OF PEOPLE'",
 'MOUNT CARMEL CEMETERY',
 'CALVARY CEMETERY',
 'UNITED STATES POSTAL',
 'DEPARTMENT OF HOUSING',
 'NY STATE DEPT TRANSP',
 'NYC DEPT HIGHWAYS',
 'ASBURY CEMETERY ASSOC',
 'DEPT OF ENVIRONMENTAL',
 'FLUSHING CEMETERY ASS',
 'CENTURY 21 DEPARTMENT',
 'ELMWIER CEMETERY ASSO',
 'DEPT OF VETERAN AFFAI',
 'GOVERNMENT OF REP ZAM',
 'GOVERNMENT OF THE PEO',
 'SILVER MNT CEMETERY',
 'DEPARTMENT OF CORRECT',
 'SPRINGFLD LI CEMETERY',
 'U S GOVT COAST GUARD',
 'DEPT OF PARKS',
 'PELHAM CEMETERY ASSOC',
 'THE GOVERNMENT OF ANT',
 'DEPT OF HOUSING PRESE',
 'DEPT OF WATER RESOURC',
 'DEPT OF PUBLIC WORKS',
 'NYC DEPT ENVIR PROT',
 'DEPARTMENT INC.',
 'GOVERNMENT/REPBLC/NGR',
 'N Y C DEPT H)WAY',
 'THE GOVERNMENT OF MON',
 'UNITED STATES/AMERICA',
 'ST RAYMONDS CEMETERY',
 'GOVERNMENT/THE ETC',
 'DEPT OF WATER RESOUR',
 'NYC DEPT WATER RESOUR',
 'N Y C DEPT HYWAY',
 'UNITED STATES POSTLSR',
 'UNITED STATES A-V A',
 'GOVERNMENT TUNISIA',
 'BARON HIRSCH CEMETERY',
 'DEPARTMENT OF JUVENIL',
 'NYC DEPT TRANSP',
 'DEPT OF PUB WKS',
 'N Y C DEPT HIGHWAY',
 'UNITED STATES OF MEXI',
 'DEPT  WATER RESOURCE',
 'DEPT OF GENERAL SERVI',
 'GOVERNMENT KNGDM LESO',
 'MOUNT HOPE CEMETERY A',
 'NYS DEPT PUB WKS',
```

```
'UNITED STATES POSTALS',
'NYC DEPT HWAY',
'UNITED STATES/AMER/A/',
'GOVERNMENT REP/SINGAP',
'HUNGARIAN GOVERNMENT',
'NYC DEPARTMENT OF FIN',
'DEPT OF HIGHWAYS NYC',
"SAINT JOHN'S CEMETERY",
'U.S. DEPARTMENT OF H.',
'DEPARTMENT OF MENTAL',
'GOVERNMENT KINGDOM LE',
'U S GOVT POST OFFICE',
'UNITED STATES LUGGAGE',
'GOVERNMENT OF ISRAEL',
'THE UNITED STATES POS',
'GOVERNMENT/REPUBLICET',
'WOODLAWN CEMETERY',
'N Y C DEPT HIGHWAYS',
'LEBANON CEMETERY ASSN',
'UNITED STATES A HUD',
'U S GOVT LAND & BLDGS',
'NYS DEPT PARKS, RECRE',
'NYC-DEPT OF HIGHWAYS',
'DEPT OF HIGHWAY',
'NYC-DEPT WATER RESOUR',
'NYC DEPT HWYS',
'U S GOVERNMENT',
'CITY OF NY-DEPT HWY',
'GOVERNMENT OF ST. LUC',
'NYC DEPT OF PUBLIC WK',
'NYC DEPT OF REAL ESTA',
'NYC DEPT OF WATER RES',
'GOVERNMENT OF THE RUS',
'DEPT OF HWAYS',
'NYC - DEPT OF HIGHWAY',
'GOVERNMENT OF EGYPT',
'NY CITY - DEPT OF HWA',
'HILLSIDE CEMETERY OF',
'U S GOVT INTERIOR',
'N Y C DEPT PUBLIC WK',
'NYC DEPT OF GEN SERV',
'N Y C DEPT OF PUBLIC',
'NYS DEPT OF TRANSP',
'THE GOVERNMENT OF THE',
'NYC DEPT OF PUBLIC WO',
'NYS DEPT OF PUB WORKS',
'KNOLLWOOD PK CEMETERY',
'NYC DEPT ENVIRON PROT',
'ELMWIER CEMETERY ASOC',
'DEPT RE-CITY OF NY',
'MT CARMEL CEMETERY',
'GOVERNMENT/THE REPUBL',
'NYS DEPARTMENT OF TRA',
'FAIR VIEW CEMETERY',
'GOVERNMENT FEDERAL ET',
'DEPT OF CULTURAL AFFA',
'UNITED STATES AVIATNU',
'NYC DEPT PUBLIC LIBRA',
'SALEM FIELDS CEMETERY',
'GOVERNMENT REPUBLICTO',
'DEPT OF PARKS AND REC',
'DEPT HOUSING PRESERVA',
'GOVERNMENT OF GUINEA',
'NYC DEPT ENVIR PROTEC',
'NYS DEPT OF TRANSPORT',
'US DEPARTMENT OF TRAN',
'GOVERNMENT OF THE SUL',
'LUTHERAN CEMETERY',
'NYC DEPT WATER RES',
'UNITED STATES A-VA',
'NYC DEPT PUBLIC WRKS',
'GOVERNMENT OF THE REP',
'GREEN-WOOD CEMETERYIN',
'U S GOVERNMENT OWNRD',
'FEDERATED DEPARTMENT',
'HOLY TRINITY CEMETERY',
'NYS DEPT OF ENV. CONS',
'US GOVERNMENT',
'UNITED STATES TRUST C',
'MACHPELAH CEMETERY',
'LIBERTY DEPARTMENT ST',
'GOVERNMENT OF BULGARI',
'DEPT OF CONSUMER AFFA',
'DEPT PUBLIC WORKS',
'UNITED STATES OF AMER',
'UNITED STATES POSTALE',
'US GOVERNMENT GEN SER',
'UNITED STATES A OF VA',
'DEPARTMENT OF GENERAL',
'UNITED STATES A-HUD',
'GOVERNMENT OF ALGERIA',
'GOVERNMENT SOCIALSTET',
'GOVERNMENT MALAYSIA',
'NYS DEPT TRANSPORT',
```

```
 'DEPT OF HIGHWAYS',
 'NYC DEPT PUBLIC WORK',
 'DEPT OF HWYS',
 'NYC DEPT OF HIGHWAYS',
 'DEPARTMENT OF EDUC.AR',
 'GOVERNMENT OF JAPAN',
 'NYC DEPT REAL ESTATE',
 'DEPT WATER RESOURCES',
 'OCEAN VIEW CEMETERY',
 'GOVERNMENT OF THE GRA',
 'GOVERNMENT OF MALAYSI',
 'UNITED STATES FUND FO',
 'DEPT PUBLIC WORKS N Y',
 'NYC DEPT OF HWYS',
 'N Y C DEPT REAL ESTAT',
 'NYC DEPT HGHWAYS',
 'WOODLAND CEMETERY ASS',
 'GOVERNMENT OF THE FED',
 'DEPT OF R E IN-REM',
 'DEPARTMENT OF TRANSPO',
 'ST PETERS CEMETERY',
 'UNITED STATES OF AMFB',
 'GOVERNMENT OF REPUBLI',
 'NYS DEPT OF CORRECTIO',
 'THE GOVERNMENT OF COT',
 'N Y STATE DEPT TRANSP',
 'NYC DEPT PUB WORKS',
 'GOVERNMENT FED REP BR',
 'U S GOVT VET ADMIN',
 'FIRE DEPARTMENT',
 'NYCITY - DEPT OF HWAY',
 'US DEPT OF HOUSING &',
 'UNITED STATES A- VA',
 'NYC DEPT PUBLIC WORKS',
 'GOVERNMENT OF UKRAINE',
 'DEPT OF WATER RES',
 'DEPARTMENT OF PARKS A',
 'NYC DEPT PUBLIC WKS',
 'GOVERNMENT OF MEXICO',
 'UNITED STATES AMERICA',
 'LAW DEPARTMENT',
 'N YCITY- DEPT OF HWAY',
 'THE LUTHERAN CEMETERY']
```

In [9]:

```
# Look at the most frequent owners. This might show some other properties we aren't interested in.
remove_list2 = data['OWNER'].value_counts().head(20).index.tolist()
remove_list2
```

Out[9]:

```
['PARKCHESTER PRESERVAT',
 'PARKS AND RECREATION',
 'DCAS',
 'HOUSING PRESERVATION',
 'CITY OF NEW YORK',
 'DEPT OF ENVIRONMENTAL',
 'BOARD OF EDUCATION',
 'NEW YORK CITY HOUSING',
 'CNY/NYCTA',
 'NYC HOUSING PARTNERSH',
 'YORKVILLE TOWERS ASSO',
 'DEPARTMENT OF BUSINES',
 'DEPT OF TRANSPORTATIO',
 'MTA/LIRR',
 'PARCKHESTER PRESERVAT',
 'MH RESIDENTIAL 1, LLC',
 '434 M LLC',
 'LINCOLN PLAZA ASSOCIA',
 'DEUTSCHE BANK NATIONA',
 '561 11TH AVENUE TMG L']
```

In [10]:

```
# add some others to also be removed
remove_list2.append('THE CITY OF NEW YORK')
remove_list2.append('NYS URBAN DEVELOPMENT')
remove_list2.append('CULTURAL AFFAIRS')
remove_list2.append('NY STATE PUBLIC WORKS')
remove_list2.append("NYC DEP'T OF HIGHWAYS")
remove_list2.append('CITY WIDE ADMINISTRAT')
remove_list2.append('NEW YORK CITY')
remove_list2.append('THE PORT OFNY & NJ')
remove_list2.append('NEW YORK STATE DEPART')
remove_list2.append('CITY AND NON-CITY OWN')
remove_list2.append('SANITATION')
remove_list2.append('NYS DOT')
remove_list2.append('NEW YORK CITY TRANSIT')
remove_list2.append('PORT AUTHORITY OF NY')
remove_list2.append('NEW YORK STATE OWNED')
remove_list2.append('NYC PARK DEPT')
remove_list2.append('PORT OF NEW YORK AUTH')
remove_list2.append('NYC PARK DEPT')
remove_list2.append('LIRR')
remove_list2.append('NY STATE PUBLIC SERV')
remove_list2.append('STATE OF NEW YORK')
remove_list2.append('NYC HIGHWAY DEPT')
remove_list2.append('CITY OF NY/PARKS AND')
```

In [11]:
```
for i in remove_list2:
    if i not in remove_list:
        remove_list.append(i)
    else:
        print(i)
```

```
DEPT OF ENVIRONMENTAL
DEPARTMENT OF BUSINES
DEPT OF TRANSPORTATIO
NYC PARK DEPT
```

In [12]:
```
# delete some of the removes...
remove_list.remove('YORKVILLE TOWERS ASSO')
remove_list.remove('434 M LLC')
remove_list.remove('DEUTSCHE BANK NATIONA')
remove_list.remove('561 11TH AVENUE TMG L')
remove_list.remove('MH RESIDENTIAL 1, LLC')
```

In [13]:
```
len(remove_list)
```

Out[13]: 264

In [14]:
```
numrecords = len(data)
removed = data[data['OWNER'].isin(remove_list)].reset_index(drop=True)
data = data[~data['OWNER'].isin(remove_list)].reset_index(drop=True)
numremoved = numrecords - len(data)
print('# records removed:', numremoved)
```

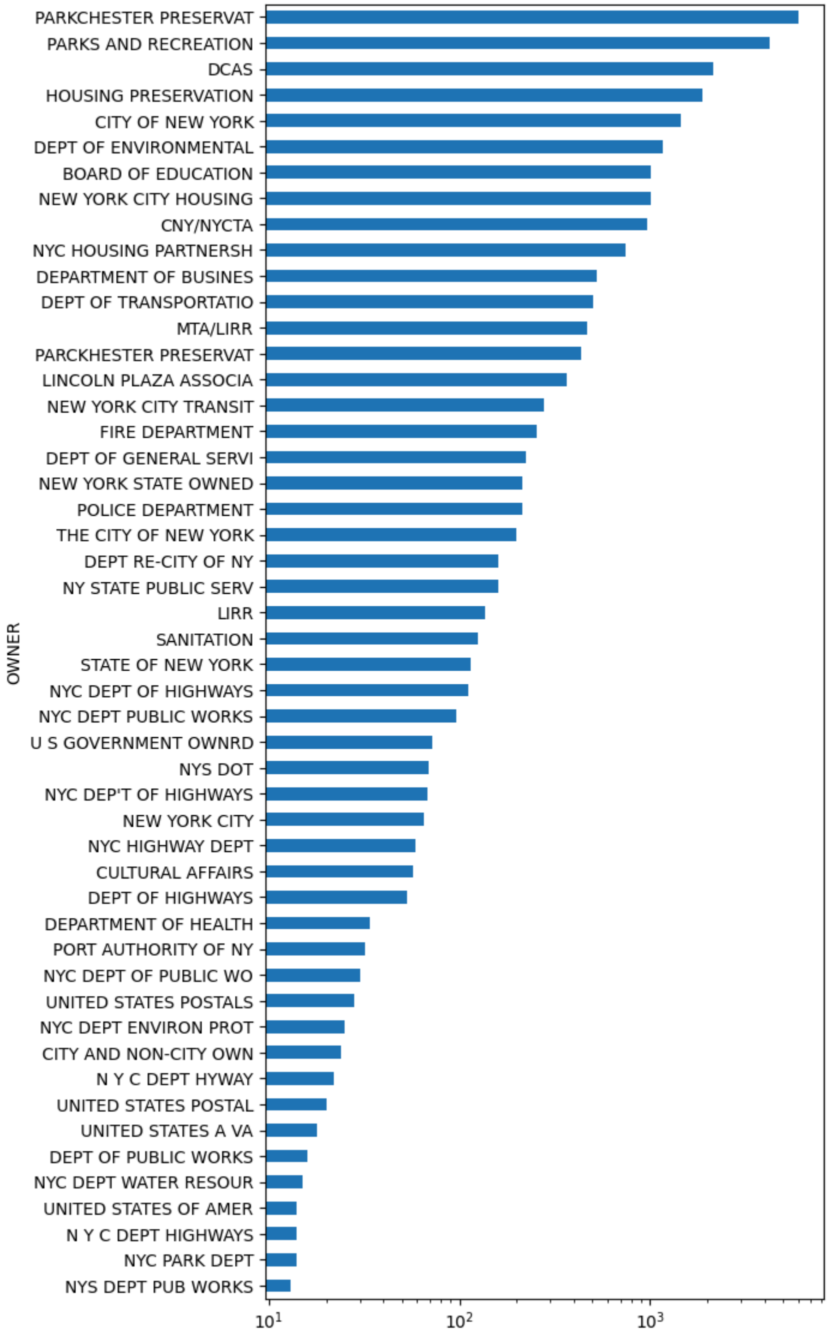
```
# records removed: 26501
```

In [15]:
```
removed.shape
```

Out[15]: (26501, 32)

In [16]:
```
# any on this list that we shouldn't remove? If so, go back and remove them from the remove list.
# plt.rcParams.update({'figure.figsize':(6,14)})
plt.figure(figsize=(6,14))
plt.xscale('log')
removed['OWNER'].value_counts().head(50).sort_values().plot(kind='barh')
```

Out[16]: <AxesSubplot: ylabel='OWNER'>



In [17]:
```
data.shape
```

Out[17]: (1044492, 32)

In [18]:
```
# this is how many records we removed
numrecords_orig - len(data)
```

Out[18]: 26502

In [19]:
```
data.head(10)
```

Out[19]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	BLDFRONT	BLDDEPTH	AVLAND2	AVTOT2
0	9	1000041001	1	4	1001	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	636093.0	2049290.0
1	10	1000041002	1	4	1002	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	919276.0	2961617.0
2	11	1000041003	1	4	1003	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	967500.0	5483912.0
3	12	1000041004	1	4	1004	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	163174.0	525692.0
4	13	1000041005	1	4	1005	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	373783.0	1204211.0
5	14	1000041006	1	4	1006	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	353383.0	1138493.0
6	15	1000041007	1	4	1007	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	1246572.0	4016063.0
7	16	1000041008	1	4	1008	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	1213369.0	3909089.0
8	17	1000041009	1	4	1009	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	1213369.0	3909089.0
9	18	1000041010	1	4	1010	NaN	TRZ HOLDINGS, LLC	R5	4	0	...	0	0	1213369.0	3909089.0

10 rows × 32 columns

Fill in missing ZIP

In [20]:
```
# How many zips are missing? Replace NAN with 0 and count them.
missing_zips = np.where(pd.isnull(data['ZIP']))[0]
num_missing_zips_orig = len(missing_zips)
num_missing_zips_orig
```

Out[20]:
```
20431
```

In [21]:
```
sum(data['BORO'].isna())
```

Out[21]:
```
0
```

In [22]:
```
sum(data['STADDR'].isna())
```

Out[22]:
```
364
```

In [23]:
```
# concatenate the 'staddr' and 'boro' columns into a new 'staddr_boro' column
data['staddr_boro'] = data[data['STADDR'].notnull()]['STADDR'] + '_' + data[data['BORO'].notnull()]['BORO'].astype(str)
data['staddr_boro']
```

Out[23]:
```
0              1 WATER STREET_1
1              1 WATER STREET_1
2              1 WATER STREET_1
3              1 WATER STREET_1
4              1 WATER STREET_1
                    ...
1044487    142 BENTLEY STREET_5
1044488    146 BENTLEY STREET_5
1044489    150 BENTLEY STREET_5
1044490    156 BENTLEY STREET_5
1044491    162 BENTLEY STREET_5
Name: staddr_boro, Length: 1044492, dtype: object
```

In [24]:
```
staddr_boro_zip = {}
for index, staddrboro in data['staddr_boro'].items():
    if staddrboro not in staddr_boro_zip :
        staddr_boro_zip [staddrboro] = data.loc[index, 'ZIP']


# fill in by mapping with street addrees boroughs
data['ZIP'] = data['ZIP'].fillna(data['staddr_boro'].map(staddr_boro_zip))
```

In [25]:
```
# how many missing zips did we fill in with this last step?
num_filled_in = num_missing_zips_orig - len(np.where(pd.isnull(data['ZIP']))[0])
num_filled_in
```

Out[25]:
```
2832
```

In [26]:
```
# How many are still left to fill in?
missing_zips = np.where(pd.isnull(data['ZIP']))[0]
len(missing_zips)
```

Out[26]: 17599

In [27]:
```
%%time
# Assume data is sorted by zip. Fill in a missing zip if the previous and next record have the same zip
zip_forward_filled = data['ZIP'].fillna(method='ffill')
zip_backward_filled = data['ZIP'].fillna(method='bfill')
data['ZIP'] = data['ZIP'].mask((zip_forward_filled == zip_backward_filled) & data['ZIP'].isna(),zip_forward_filled)
```

```
CPU times: user 8.45 ms, sys: 2.24 ms, total: 10.7 ms
Wall time: 9.68 ms
```

```
<timed exec>:2: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill
() instead.
<timed exec>:3: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill
() instead.
```

In [28]:
```
# how many missing zips did we fill in with this last step?
num_filled_in = len(missing_zips) - len(np.where(pd.isnull(data['ZIP']))[0])
num_filled_in
```

Out[28]: 16126

In [29]:
```
# How many are still left to fill in?
missing_zips = np.where(pd.isnull(data['ZIP']))[0]
len(missing_zips)
```

Out[29]: 1473

In [30]:
```
%time
data['ZIP'].fillna(method='ffill', inplace=True)
```

```
CPU times: user 0 ns, sys: 1e+03 ns, total: 1e+03 ns
Wall time: 3.1 µs
```

```
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/2418375584.py:2: FutureWarning: A value is trying to be set on a copy of
a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].meth
od(value) instead, to perform the operation inplace on the original object.


  data['ZIP'].fillna(method='ffill', inplace=True)
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/2418375584.py:2: FutureWarning: Series.fillna with 'method' is deprecated
and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
  data['ZIP'].fillna(method='ffill', inplace=True)
```

In [31]:
```
# For the remaining missing zips, just fill in with the previous record's zip.
missing_zips = np.where(pd.isnull(data['ZIP']))[0]
len(missing_zips)
```

Out[31]: 0

In [32]:
```
data = data.drop('staddr_boro', axis=1)
```

FULLVAL, AVLAND, AVTOT

FULLVAL

In [33]:
```
len(data[data['FULLVAL']==0])
```

Out[33]: 10025

In [34]:
```
data['FULLVAL'].isnull().sum()
```

Out[34]: 0

In [35]:
```
data['FULLVAL'].replace(0, np.nan, inplace=True)
data['FULLVAL'].isnull().sum()
```

```
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/3840300546.py:1: FutureWarning: A value is trying to be set on a copy of
a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].meth
od(value) instead, to perform the operation inplace on the original object.


  data['FULLVAL'].replace(0, np.nan, inplace=True)
```

Out[35]: 10025

In [36]:
```
data["FULLVAL"] = data.\
                        groupby(['TAXCLASS','BORO','BLDGCL'])['FULLVAL'].transform(lambda x: x.fillna(x.mean()))
data['FULLVAL'].isnull().sum()
```

Out[36]: 7307

In [37]:
```
data["FULLVAL"] = data.\
                        groupby(['TAXCLASS','BORO'])['FULLVAL'].transform(lambda x: x.fillna(x.mean()))
data['FULLVAL'].isnull().sum()
```

Out[37]: 386

In [38]:
```
data["FULLVAL"] = data.\
                        groupby(['TAXCLASS'])['FULLVAL'].transform(lambda x: x.fillna(x.mean()))
data['FULLVAL'].isnull().sum()
```

Out[38]: 0

AVLAND

In [39]:
```
len(data[data['AVLAND']==0])
```

Out[39]: 10027

In [40]:
```
data['AVLAND'].isnull().sum()
```

Out[40]: 0

In [41]:
```
data['AVLAND'].replace(0, np.nan, inplace=True)
data['AVLAND'].isnull().sum()
```

```
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/116382313.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['AVLAND'].replace(0, np.nan, inplace=True)
```

Out[41]: 10027

In [42]:
```
data["AVLAND"] = data.\
                        groupby(['TAXCLASS','BORO','BLDGCL'])['AVLAND'].transform(lambda x: x.fillna(x.mean()))
data['AVLAND'].isnull().sum()
```

Out[42]: 7307

In [43]:
```
data["AVLAND"] = data.\
                        groupby(['TAXCLASS','BORO'])['AVLAND'].transform(lambda x: x.fillna(x.mean()))
data['AVLAND'].isnull().sum()
```

Out[43]: 386

In [44]:
```
data["AVLAND"] = data.\
                        groupby(['TAXCLASS'])['AVLAND'].transform(lambda x: x.fillna(x.mean()))
data['AVLAND'].isnull().sum()
```

Out[44]: 0

AVTOT

In [45]:
```
len(data[data['AVTOT']==0])
```

Out[45]: 10025

In [46]:
```
data['AVTOT'].isnull().sum()
```

Out[46]: 0

In [47]:
```
data['AVTOT'].replace(0, np.nan, inplace=True)
data['AVTOT'].isnull().sum()
```

```
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/3655551349.py:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data['AVTOT'].replace(0, np.nan, inplace=True)
```

Out[47]: 10025

In [48]:
```
data["AVTOT"] = data.\
                        groupby(['TAXCLASS','BORO','BLDGCL'])['AVTOT'].transform(lambda x: x.fillna(x.mean()))
data['AVTOT'].isnull().sum()
```

Out[48]: 7307

In [49]:
```
data["AVTOT"] = data.\
                                groupby(['TAXCLASS','BORO'])['AVTOT'].transform(lambda x: x.fillna(x.mean()))
data['AVTOT'].isnull().sum()
```

Out[49]: 386

In [50]:
```
data["AVTOT"] = data.\
                                groupby(['TAXCLASS'])['AVTOT'].transform(lambda x: x.fillna(x.mean()))
data['AVTOT'].isnull().sum()
```

Out[50]: 0

In [51]:
```
data.head().transpose()
```

Out[51]:

	0	1	2	3	4
RECORD	9	10	11	12	13
BBLE	1000041001	1000041002	1000041003	1000041004	1000041005
BORO	1	1	1	1	1
BLOCK	4	4	4	4	4
LOT	1001	1002	1003	1004	1005
EASEMENT	NaN	NaN	NaN	NaN	NaN
OWNER	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC
BLDGCL	R5	R5	R5	R5	R5
TAXCLASS	4	4	4	4	4
LTFRONT	0	0	0	0	0
LTDEPTH	0	0	0	0	0
EXT	NaN	NaN	NaN	NaN	NaN
STORIES	50.0	50.0	50.0	50.0	50.0
FULLVAL	3944762.0	5700930.0	10600000.0	1011928.0	2318026.0
AVLAND	636093.0	919276.0	967500.0	163174.0	373783.0
AVTOT	1775143.0	2565419.0	4770000.0	455368.0	1043112.0
EXLAND	0.0	0.0	0.0	0.0	0.0
EXTOT	0.0	0.0	0.0	0.0	0.0
EXCD1	NaN	NaN	NaN	NaN	NaN
STADDR	1 WATER STREET	1 WATER STREET	1 WATER STREET	1 WATER STREET	1 WATER STREET
ZIP	10004.0	10004.0	10004.0	10004.0	10004.0
EXMPTCL	NaN	NaN	NaN	NaN	NaN
BLDFRONT	0	0	0	0	0
BLDDEPTH	0	0	0	0	0
AVLAND2	636093.0	919276.0	967500.0	163174.0	373783.0
AVTOT2	2049290.0	2961617.0	5483912.0	525692.0	1204211.0
EXLAND2	NaN	NaN	NaN	NaN	NaN
EXTOT2	NaN	NaN	NaN	NaN	NaN
EXCD2	NaN	NaN	NaN	NaN	NaN
PERIOD	FINAL	FINAL	FINAL	FINAL	FINAL
YEAR	2010/11	2010/11	2010/11	2010/11	2010/11
VALTYPE	AC-TR	AC-TR	AC-TR	AC-TR	AC-TR

Fill in the missing STORIES

In [52]:
```
data['STORIES'].isnull().sum()
```

Out[52]: 42029

In [53]:
```
modes = data.groupby(['BORO', 'BLDGCL'])['STORIES'] \
            .transform(lambda x: x.mode(dropna=False).iloc[0])
data['STORIES'] = data['STORIES'].fillna(modes)
```

In [54]:
```
data['STORIES'].isnull().sum()
```

Out[54]: 37921

In [55]:
```
data["STORIES"] = data.\
                                groupby(['TAXCLASS'])['STORIES'].transform(lambda x: x.fillna(x.mean()))
```

In [56]:
```
data['STORIES'].isnull().sum()
```

Out[56]: 0

In [57]:
```
data.head().transpose()
```

Out[57]:

	0	1	2	3	4
RECORD	9	10	11	12	13
BBLE	1000041001	1000041002	1000041003	1000041004	1000041005
BORO	1	1	1	1	1
BLOCK	4	4	4	4	4
LOT	1001	1002	1003	1004	1005
EASEMENT	NaN	NaN	NaN	NaN	NaN
OWNER	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC	TRZ HOLDINGS, LLC
BLDGCL	R5	R5	R5	R5	R5
TAXCLASS	4	4	4	4	4
LTFRONT	0	0	0	0	0
LTDEPTH	0	0	0	0	0
EXT	NaN	NaN	NaN	NaN	NaN
STORIES	50.0	50.0	50.0	50.0	50.0
FULLVAL	3944762.0	5700930.0	10600000.0	1011928.0	2318026.0
AVLAND	636093.0	919276.0	967500.0	163174.0	373783.0
AVTOT	1775143.0	2565419.0	4770000.0	455368.0	1043112.0
EXLAND	0.0	0.0	0.0	0.0	0.0
EXTOT	0.0	0.0	0.0	0.0	0.0
EXCD1	NaN	NaN	NaN	NaN	NaN
STADDR	1 WATER STREET	1 WATER STREET	1 WATER STREET	1 WATER STREET	1 WATER STREET
ZIP	10004.0	10004.0	10004.0	10004.0	10004.0
EXMPTCL	NaN	NaN	NaN	NaN	NaN
BLDFRONT	0	0	0	0	0
BLDDEPTH	0	0	0	0	0
AVLAND2	636093.0	919276.0	967500.0	163174.0	373783.0
AVTOT2	2049290.0	2961617.0	5483912.0	525692.0	1204211.0
EXLAND2	NaN	NaN	NaN	NaN	NaN
EXTOT2	NaN	NaN	NaN	NaN	NaN
EXCD2	NaN	NaN	NaN	NaN	NaN
PERIOD	FINAL	FINAL	FINAL	FINAL	FINAL
YEAR	2010/11	2010/11	2010/11	2010/11	2010/11
VALTYPE	AC-TR	AC-TR	AC-TR	AC-TR	AC-TR

Fill in LTFRONT, LTDEPTH, BLDDEPTH, BLDFRONT with averages by TAXCLASS

In [58]:
```
# Because these 4 fields do not have NAs, we just need to replace 0s.
# We think zero and 1 are invalid values for these fields, so replace them with NA.
# Probably OK for BLD dimensions to be zero. Property could have no building.
# Calculate groupwise average. Replace 0 and 1's by NAs so they are not counted in calculating mean.
# Not sure which values to treat as missing. Here are some choices.
data.loc[data['LTFRONT']==0,'LTFRONT']=np.nan
data.loc[data['LTDEPTH']==0,'LTDEPTH']=np.nan
# data.loc[data['BLDFRONT']==0,'BLDFRONT']=np.nan
# data.loc[data['BLDDEPTH']==0,'BLDDEPTH']=np.nan
data.loc[data['LTFRONT']==1,'LTFRONT']=np.nan
data.loc[data['LTDEPTH']==1,'LTDEPTH']=np.nan
data.loc[data['BLDFRONT']==1,'BLDFRONT']=np.nan
data.loc[data['BLDDEPTH']==1,'BLDDEPTH']=np.nan
```

In [59]:
```
data.head()
```

Out[59]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	BLDFRONT	BLDDEPTH	AVLAND2	AVTOT2
0	9	1000041001	1	4	1001	NaN	TRZ HOLDINGS, LLC	R5	4	NaN	...	0.0	0.0	636093.0	2049290.0
1	10	1000041002	1	4	1002	NaN	TRZ HOLDINGS, LLC	R5	4	NaN	...	0.0	0.0	919276.0	2961617.0
2	11	1000041003	1	4	1003	NaN	TRZ HOLDINGS, LLC	R5	4	NaN	...	0.0	0.0	967500.0	5483912.0
3	12	1000041004	1	4	1004	NaN	TRZ HOLDINGS, LLC	R5	4	NaN	...	0.0	0.0	163174.0	525692.0
4	13	1000041005	1	4	1005	NaN	TRZ HOLDINGS, LLC	R5	4	NaN	...	0.0	0.0	373783.0	1204211.0

5 rows × 32 columns

LTFRONT

In [60]:
```
data['LTFRONT'].isnull().sum()
```

Out[60]: 161133

In [61]:
```
data["LTFRONT"] = data.\
                        groupby(['TAXCLASS','BORO'])['LTFRONT'].transform(lambda x: x.fillna(x.mean()))
data[data['LTFRONT'].isnull()]
```

Out[61]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	BLDFRONT	BLDDEPTH	AVLAND2	AVTOT2
126002	127752	1018259034	1	1825	9034	NaN	NaN	V0	1B	NaN	...	0.0	0.0	NaN	NaN
126003	127753	1018259036	1	1825	9036	NaN	NaN	V0	1B	NaN	...	0.0	0.0	NaN	NaN

2 rows × 32 columns

In [62]:
```
data["LTFRONT"] = data.\
                        groupby(['TAXCLASS'])['LTFRONT'].transform(lambda x: x.fillna(x.mean()))
data['LTFRONT'].isnull().sum()
```

Out[62]: 0

LTDEPTH

In [63]:
```
data['LTDEPTH'].isnull().sum()
```

Out[63]: 161715

In [64]:
```
data["LTDEPTH"] = data.\
                        groupby(['TAXCLASS','BORO'])['LTDEPTH'].transform(lambda x: x.fillna(x.mean()))
data[data['LTDEPTH'].isnull()]
```

Out[64]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	BLDFRONT	BLDDEPTH	AVLAND2	AVTOT
126002	127752	1018259034	1	1825	9034	NaN	NaN	V0	1B	45.465048	...	0.0	0.0	NaN	Na
126003	127753	1018259036	1	1825	9036	NaN	NaN	V0	1B	45.465048	...	0.0	0.0	NaN	Na

2 rows × 32 columns

In [65]:
```
data["LTDEPTH"] = data.\
                        groupby(['TAXCLASS'])['LTDEPTH'].transform(lambda x: x.fillna(x.mean()))
data['LTDEPTH'].isnull().sum()
```

Out[65]: 0

BLDFRONT

In [66]:
```
data['BLDFRONT'].isnull().sum()
```

Out[66]: 75

In [67]:
```
data['BLDFRONT'] = data.\
                        groupby(['TAXCLASS','BORO','BLDGCL'])['BLDFRONT'].transform(lambda x: x.fillna(x.mean()))
data['BLDFRONT'].isnull().sum()
```

Out[67]: 0

In [68]:
```
data['BLDFRONT'] = data.\
                        groupby(['TAXCLASS','BORO'])['BLDFRONT'].transform(lambda x: x.fillna(x.mean()))
data['BLDFRONT'].isnull().sum()
```

Out[68]: 0

In [69]:
```
data['BLDFRONT'] = data.\
                        groupby(['TAXCLASS'])['BLDFRONT'].transform(lambda x: x.fillna(x.mean()))
data['BLDFRONT'].isnull().sum()
```

Out[69]: 0

BLDEPTH

In [70]:
```
data['BLDDEPTH'].isnull().sum()
```

Out[70]: 58

In [71]:
```
data['BLDDEPTH'] = data.\
                        groupby(['TAXCLASS','BORO','BLDGCL'])['BLDDEPTH'].transform(lambda x: x.fillna(x.mean()))
data['BLDDEPTH'].isnull().sum()
```

Out[71]: 0

In [72]:
```
data['BLDDEPTH'] = data.\
                        groupby(['TAXCLASS','BORO'])['BLDDEPTH'].transform(lambda x: x.fillna(x.mean()))
data['BLDDEPTH'].isnull().sum()
```

Out[72]: 0

In [73]:
```
data['BLDDEPTH'] = data.\
                        groupby(['TAXCLASS'])['BLDDEPTH'].transform(lambda x: x.fillna(x.mean()))
data['BLDDEPTH'].isnull().sum()
```

Out[73]: 0

In [74]:
```
data.dtypes
```

Out[74]:
```
RECORD        int64
BBLE         object
BORO          int64
BLOCK         int64
LOT           int64
EASEMENT     object
OWNER        object
BLDGCL       object
TAXCLASS     object
LTFRONT     float64
LTDEPTH     float64
EXT          object
STORIES     float64
FULLVAL     float64
AVLAND      float64
AVTOT       float64
EXLAND      float64
EXTOT       float64
EXCD1       float64
STADDR       object
ZIP         float64
EXMPTCL      object
BLDFRONT    float64
BLDDEPTH    float64
AVLAND2     float64
AVTOT2      float64
EXLAND2     float64
EXTOT2      float64
EXCD2       float64
PERIOD       object
YEAR         object
VALTYPE      object
dtype: object
```

In [75]:
```
# convert ZIP to a string rather than a float
# We call the first three digits of the zip zip3
data['ZIP'] = data['ZIP'].astype(str)
data['zip3'] = data['ZIP'].str[:3]
```

In [76]:
```
data.count()
```

Out[76]:
```
RECORD      1044492
BBLE        1044492
BORO        1044492
BLOCK       1044492
LOT         1044492
EASEMENT       1976
OWNER       1012748
BLDGCL      1044492
TAXCLASS    1044492
LTFRONT     1044492
LTDEPTH     1044492
EXT          353646
STORIES     1044492
FULLVAL     1044492
AVLAND      1044492
AVTOT       1044492
EXLAND      1044492
EXTOT       1044492
EXCD1        623528
STADDR      1044128
ZIP         1044492
EXMPTCL        9295
BLDFRONT    1044492
BLDDEPTH    1044492
AVLAND2      266065
AVTOT2       266071
EXLAND2       80844
EXTOT2       117833
EXCD2         92904
PERIOD      1044492
YEAR        1044492
VALTYPE     1044492
zip3        1044492
dtype: int64
```

In [77]:
```
cols = data.columns
print(cols)
```

```
Index(['RECORD', 'BBLE', 'BORO', 'BLOCK', 'LOT', 'EASEMENT', 'OWNER', 'BLDGCL',
       'TAXCLASS', 'LTFRONT', 'LTDEPTH', 'EXT', 'STORIES', 'FULLVAL', 'AVLAND',
       'AVTOT', 'EXLAND', 'EXTOT', 'EXCD1', 'STADDR', 'ZIP', 'EXMPTCL',
       'BLDFRONT', 'BLDDEPTH', 'AVLAND2', 'AVTOT2', 'EXLAND2', 'EXTOT2',
       'EXCD2', 'PERIOD', 'YEAR', 'VALTYPE', 'zip3'],
      dtype='object')
```

Now build variables that try to find properties that are unusual in ways we're interested in

In [78]:
```
# epsilon is an arbitrary small number to make sure we don't divide by zero
epsilon = .0001
data['ltsize'] = data['LTFRONT'] * data['LTDEPTH'] + epsilon
data['bldsize'] = data['BLDFRONT'] * data['BLDDEPTH'] + epsilon
data['bldvol'] = data['bldsize'] * data['STORIES'] + epsilon
```

In [79]:
```
data['r1'] = data['FULLVAL'] / data['ltsize']
data['r2'] = data['FULLVAL'] / data['bldsize']
data['r3'] = data['FULLVAL'] / data['bldvol']
data['r4'] = data['AVLAND'] / data['ltsize']
data['r5'] = data['AVLAND'] / data['bldsize']
data['r6'] = data['AVLAND'] / data['bldvol']
data['r7'] = data['AVTOT'] / data['ltsize']
data['r8'] = data['AVTOT'] / data['bldsize']
data['r9'] = data['AVTOT'] / data['bldvol']
```

In [80]:
```
data.describe()
```

Out[80]:

	RECORD	BORO	BLOCK	LOT	LTFRONT	LTDEPTH	STORIES	FULLVAL	AVLAND	AVTO
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+0
mean	5.368071e+05	3.220281e+00	4.756780e+03	3.509016e+02	5.045399e+01	1.073810e+02	4.969850e+00	8.163420e+05	6.643735e+04	1.998216e+0
std	3.080025e+05	1.199074e+00	3.677416e+03	8.267098e+02	5.999403e+01	5.153434e+01	8.225043e+00	6.394366e+06	2.009129e+06	5.391132e+0
min	9.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	2.000000e+00	2.000000e+00	1.000000e+00	4.000000e+00	1.000000e+00	1.000000e+0
25%	2.729098e+05	3.000000e+00	1.542000e+03	2.300000e+01	2.100000e+01	1.000000e+02	2.000000e+00	3.181550e+05	9.679000e+03	1.892600e+0
50%	5.387725e+05	3.000000e+00	4.078000e+03	4.900000e+01	3.000000e+01	1.000000e+02	2.000000e+00	4.540000e+05	1.387800e+04	2.579100e+0
75%	8.022752e+05	4.000000e+00	6.920000e+03	1.400000e+02	6.000000e+01	1.120598e+02	4.000000e+00	6.240000e+05	1.998000e+04	4.724400e+0
max	1.070994e+06	5.000000e+00	1.635000e+04	9.450000e+03	9.999000e+03	9.619000e+03	1.190000e+02	1.663775e+09	1.792809e+09	4.668309e+0

8 rows × 32 columns

I want outliers in these 9 variables, either very high or very low. Very high is easy to find but very low might be close to zero and probably not many standard deviations below the average. A simple way to look for outliers that are very low is to also include 1/over these variables, which will be very large outliers when the variables are very low. First I scale them all to have reasonable average.

In [81]:
```
vars9 = ['r1','r2','r3','r4','r5','r6','r7','r8','r9']
for vars in vars9:
    data[vars] = data[vars]/data[vars].median()
```

```
data.describe()
```

Out[81]:

	RECORD	BORO	BLOCK	LOT	LTFRONT	LTDEPTH	STORIES	FULLVAL	AVLAND	AVTO
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+0
mean	5.368071e+05	3.220281e+00	4.756780e+03	3.509016e+02	5.045399e+01	1.073810e+02	4.969850e+00	8.163420e+05	6.643735e+04	1.998216e+0
std	3.080025e+05	1.199074e+00	3.677416e+03	8.267098e+02	5.999403e+01	5.153434e+01	8.225043e+00	6.394366e+06	2.009129e+06	5.391132e+0
min	9.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	2.000000e+00	2.000000e+00	1.000000e+00	4.000000e+00	1.000000e+00	1.000000e+0
25%	2.729098e+05	3.000000e+00	1.542000e+03	2.300000e+01	2.100000e+01	1.000000e+02	2.000000e+00	3.181550e+05	9.679000e+03	1.892600e+0
50%	5.387725e+05	3.000000e+00	4.078000e+03	4.900000e+01	3.000000e+01	1.000000e+02	2.000000e+00	4.540000e+05	1.387800e+04	2.579100e+0
75%	8.022752e+05	4.000000e+00	6.920000e+03	1.400000e+02	6.000000e+01	1.120598e+02	4.000000e+00	6.240000e+05	1.998000e+04	4.724400e+0
max	1.070994e+06	5.000000e+00	1.635000e+04	9.450000e+03	9.999000e+03	9.619000e+03	1.190000e+02	1.663775e+09	1.792809e+09	4.668309e+0

8 rows × 32 columns

In [82]:
```
# add in the inverse of all the 9 primary variables.
for vars in vars9:
    data[vars+'inv'] = 1/(data[vars] + epsilon)
```

In [83]:
```
data.head()
```

Out[83]:

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	r9	r1inv	r2inv	r3
0	9	1000041001	1	4	1001	NaN	TRZ HOLDINGS, LLC	R5	4	70.813856	...	2.157505e+07	0.319454	1.613565e-08	4.01164
1	10	1000041002	1	4	1002	NaN	TRZ HOLDINGS, LLC	R5	4	70.813856	...	3.118004e+07	0.221048	1.116507e-08	2.77586
2	11	1000041003	1	4	1003	NaN	TRZ HOLDINGS, LLC	R5	4	70.813856	...	5.797447e+07	0.118886	6.004840e-09	1.49292
3	12	1000041004	1	4	1004	NaN	TRZ HOLDINGS, LLC	R5	4	70.813856	...	5.534532e+06	1.245200	6.290102e-08	1.56384
4	13	1000041005	1	4	1005	NaN	TRZ HOLDINGS, LLC	R5	4	70.813856	...	1.267796e+07	0.543627	2.745927e-08	6.82692

5 rows × 54 columns

Now I want the large outliers where the variables are either very low or very high, so I'll keep only one of the two, r or rinv, depending on which is largest. This allows me to find both the very low and high outliers.

In [84]:
```
for vars in vars9:
    data[vars] = data[[vars,vars+'inv']].max(axis=1)
```

Now I can remove the inverse columns since I have the 9 variables that I need

In [85]:
```
for vars in vars9:
    data.drop(columns=(vars+'inv'),inplace=True)

data.describe()
```

Out[85]:

	RECORD	BORO	BLOCK	LOT	LTFRONT	LTDEPTH	STORIES	FULLVAL	AVLAND	AVTO
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+0
mean	5.368071e+05	3.220281e+00	4.756780e+03	3.509016e+02	5.045399e+01	1.073810e+02	4.969850e+00	8.163420e+05	6.643735e+04	1.998216e+0
std	3.080025e+05	1.199074e+00	3.677416e+03	8.267098e+02	5.999403e+01	5.153434e+01	8.225043e+00	6.394366e+06	2.009129e+06	5.391132e+0
min	9.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	2.000000e+00	2.000000e+00	1.000000e+00	4.000000e+00	1.000000e+00	1.000000e+0
25%	2.729098e+05	3.000000e+00	1.542000e+03	2.300000e+01	2.100000e+01	1.000000e+02	2.000000e+00	3.181550e+05	9.679000e+03	1.892600e+0
50%	5.387725e+05	3.000000e+00	4.078000e+03	4.900000e+01	3.000000e+01	1.000000e+02	2.000000e+00	4.540000e+05	1.387800e+04	2.579100e+0
75%	8.022752e+05	4.000000e+00	6.920000e+03	1.400000e+02	6.000000e+01	1.120598e+02	4.000000e+00	6.240000e+05	1.998000e+04	4.724400e+0
max	1.070994e+06	5.000000e+00	1.635000e+04	9.450000e+03	9.999000e+03	9.619000e+03	1.190000e+02	1.663775e+09	1.792809e+09	4.668309e+0

8 rows × 32 columns

Now I add more variables where I standardize each of these 9 basic variables by a few logical groupings. For example, is a property's value of r1 typical for that zip code? for that taxclass?

In [86]:
```
# Standardized variables by appropriate logical group
zip5_mean = data.groupby('ZIP')[vars9].mean()
taxclass_mean = data.groupby('TAXCLASS')[vars9].mean()
data = data.join(zip5_mean, on='ZIP', rsuffix='_zip5')
data = data.join(taxclass_mean, on='TAXCLASS', rsuffix='_taxclass')
```

```
rsuffix = ['_zip5', '_taxclass']
for var in vars9:
    for r in rsuffix:
        data[str(var)+r] = data[var] / data[str(var)+r]
```

In [87]:
```
# include two more possibly interesting variables
data['value_ratio'] = data['FULLVAL']/(data['AVLAND']+data['AVTOT'])
data['value_ratio'] = data['value_ratio']/data['value_ratio'].mean()
# again, use 1/variable if that's larger, in order to find the low outliers
data['value_ratio'] = np.where(data['value_ratio'] < 1, 1/(data['value_ratio']+epsilon), data['value_ratio'])
data['size_ratio'] = data['bldsize'] / (data['ltsize']+1)
```

In [88]:
```
data.head().transpose()
```

Out[88]:

	0	1	2	3	4
RECORD	9	10	11	12	13
BBLE	1000041001	1000041002	1000041003	1000041004	1000041005
BORO	1	1	1	1	1
BLOCK	4	4	4	4	4
LOT	1001	1002	1003	1004	1005
...	...	...	...	...	...
r7_taxclass	1.288211	1.86171	3.461563	0.330458	0.756981
r8_taxclass	7.20413	10.411337	19.358271	1.848037	4.233301
r9_taxclass	1.127871	1.629988	3.03071	0.289327	0.662761
value_ratio	8.056712	8.056715	7.13503	8.056724	8.05672
size_ratio	0.0	0.0	0.0	0.0	0.0

65 rows × 5 columns

In [89]:
```
data.columns
```

Out[89]:
```
Index(['RECORD', 'BBLE', 'BORO', 'BLOCK', 'LOT', 'EASEMENT', 'OWNER', 'BLDGCL',
       'TAXCLASS', 'LTFRONT', 'LTDEPTH', 'EXT', 'STORIES', 'FULLVAL', 'AVLAND',
       'AVTOT', 'EXLAND', 'EXTOT', 'EXCD1', 'STADDR', 'ZIP', 'EXMPTCL',
       'BLDFRONT', 'BLDDEPTH', 'AVLAND2', 'AVTOT2', 'EXLAND2', 'EXTOT2',
       'EXCD2', 'PERIOD', 'YEAR', 'VALTYPE', 'zip3', 'ltsize', 'bldsize',
       'bldvol', 'r1', 'r2', 'r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9',
       'r1_zip5', 'r2_zip5', 'r3_zip5', 'r4_zip5', 'r5_zip5', 'r6_zip5',
       'r7_zip5', 'r8_zip5', 'r9_zip5', 'r1_taxclass', 'r2_taxclass',
       'r3_taxclass', 'r4_taxclass', 'r5_taxclass', 'r6_taxclass',
       'r7_taxclass', 'r8_taxclass', 'r9_taxclass', 'value_ratio',
       'size_ratio'],
      dtype='object')
```

In [90]:
```
save_record = data['RECORD']
save_record.head()
```

Out[90]:
```
0     9
1    10
2    11
3    12
4    13
Name: RECORD, dtype: int64
```

In [91]:
```
dropcols = ['RECORD','BBLE', 'BORO', 'BLOCK', 'LOT', 'EASEMENT',
       'OWNER', 'BLDGCL', 'TAXCLASS', 'LTFRONT', 'LTDEPTH', 'EXT', 'STORIES',
       'FULLVAL', 'AVLAND', 'AVTOT', 'EXLAND', 'EXTOT', 'EXCD1', 'STADDR',
       'ZIP', 'EXMPTCL', 'BLDFRONT', 'BLDDEPTH', 'AVLAND2', 'AVTOT2',
       'EXLAND2', 'EXTOT2', 'EXCD2', 'PERIOD', 'YEAR', 'VALTYPE', 'zip3','ltsize','bldsize','bldvol']
data = data.drop(columns = dropcols)
data.shape
```

Out[91]:
```
(1044492, 29)
```

In [92]:
```
# this dataframe is now just the variables for our unsupervised fraud models
data.head().transpose()
```

Out[92]:

	0	1	2	3	4
r1	3.130243e+00	4.523796e+00	8.411301e+00	1.245200e+00	1.839398e+00
r2	6.197457e+07	8.956501e+07	1.665323e+08	1.589799e+07	3.641757e+07
r3	2.492741e+06	3.602484e+06	6.698264e+06	6.394491e+05	1.464788e+06
r4	1.686504e+01	2.437321e+01	2.565179e+01	4.326310e+00	9.910288e+00
r5	3.375292e+08	4.877942e+08	5.133833e+08	8.658481e+07	1.983400e+08
r6	1.344811e+07	1.943509e+07	2.045463e+07	3.449782e+06	7.902422e+06
r7	2.506448e+01	3.622294e+01	6.735096e+01	6.429659e+00	1.472843e+01
r8	5.375602e+08	7.768767e+08	1.444482e+09	1.378975e+08	3.158819e+08
r9	2.157505e+07	3.118004e+07	5.797447e+07	5.534532e+06	1.267796e+07
r1_zip5	3.448109e-01	4.983172e-01	9.265440e-01	1.371645e-01	2.026182e-01
r2_zip5	3.641686e+00	5.262927e+00	9.785601e+00	9.341815e-01	2.139932e+00
r3_zip5	1.255151e+00	1.813931e+00	3.372725e+00	3.219768e-01	7.375532e-01
r4_zip5	1.040068e+00	1.503097e+00	1.581947e+00	2.668038e-01	6.111680e-01
r5_zip5	3.886179e+00	5.616272e+00	5.910894e+00	9.969036e-01	2.283609e+00
r6_zip5	9.720449e-01	1.404791e+00	1.478484e+00	2.493542e-01	5.711961e-01
r7_zip5	1.368515e+00	1.977764e+00	3.677347e+00	3.510579e-01	8.041688e-01
r8_zip5	3.667110e+00	5.299671e+00	9.853918e+00	9.407042e-01	2.154872e+00
r9_zip5	1.274764e+00	1.842276e+00	3.425428e+00	3.270084e-01	7.490786e-01
r1_taxclass	5.854262e-02	8.460519e-02	1.573103e-01	2.328804e-02	3.440089e-02
r2_taxclass	8.893294e+00	1.285250e+01	2.389724e+01	2.281348e+00	5.225889e+00
r3_taxclass	1.818169e+00	2.627600e+00	4.885616e+00	4.664049e-01	1.068395e+00
r4_taxclass	5.456175e-01	7.885217e-01	8.298865e-01	1.399647e-01	3.206175e-01
r5_taxclass	6.733415e+00	9.731073e+00	1.024155e+01	1.727292e+00	3.956711e+00
r6_taxclass	9.166989e-01	1.324805e+00	1.394303e+00	2.351565e-01	5.386735e-01
r7_taxclass	1.288211e+00	1.861710e+00	3.461563e+00	3.304580e-01	7.569806e-01
r8_taxclass	7.204130e+00	1.041134e+01	1.935827e+01	1.848037e+00	4.233301e+00
r9_taxclass	1.127871e+00	1.629988e+00	3.030710e+00	2.893267e-01	6.627610e-01
value_ratio	8.056712e+00	8.056715e+00	7.135030e+00	8.056724e+00	8.056720e+00
size_ratio	1.216887e-08	1.216887e-08	1.216887e-08	1.216887e-08	1.216887e-08

In [93]:

```
# Calculate and write the basic statistics of all the variables to check if everything looks OK
stats = data.describe().transpose()
# stats.to_excel('stats_on_vars.xlsx')
stats
```

Out[93]:

	count	mean	std	min	25%	50%	75%	max
r1	1044492.0	1.294407e+01	1.088700e+02	9.999570e-01	1.269474e+00	1.701746	3.207889	9.769741e+03
r2	1044492.0	1.365419e+06	4.578976e+07	9.999527e-01	1.175687e+00	1.463995	5.341069	2.613890e+10
r3	1044492.0	3.306131e+05	1.088307e+07	9.999526e-01	1.193296e+00	1.492702	5.831286	7.659900e+09
r4	1044492.0	8.518748e+00	7.580206e+01	9.999539e-01	1.262922e+00	1.682360	3.260424	9.821693e+03
r5	1044492.0	6.522880e+06	9.513500e+08	9.999542e-01	1.164241e+00	1.461362	7.953975	9.513159e+11
r6	1044492.0	1.740853e+06	4.744712e+08	9.999570e-01	1.190033e+00	1.524923	9.407930	4.832645e+11
r7	1044492.0	5.746087e+00	5.249257e+01	9.999514e-01	1.233189e+00	1.585997	2.710417	9.904245e+03
r8	1044492.0	1.141183e+07	1.437380e+09	9.999505e-01	1.167393e+00	1.468319	4.898251	1.413687e+12
r9	1044492.0	2.583144e+06	7.133054e+08	9.999505e-01	1.193659e+00	1.502809	4.631073	7.234161e+11
r1_zip5	1044492.0	1.000000e+00	8.217454e+00	5.039810e-03	1.620008e-01	0.321577	0.664211	2.284920e+03
r2_zip5	1044492.0	1.000000e+00	1.883100e+01	2.214009e-08	2.643649e-06	0.000005	0.000018	1.109450e+04
r3_zip5	1044492.0	1.000000e+00	2.030114e+01	9.732751e-08	5.730721e-06	0.000013	0.000054	1.327377e+04
r4_zip5	1044492.0	1.000000e+00	6.586975e+00	7.197847e-03	2.092844e-01	0.399771	0.721085	2.092640e+03
r5_zip5	1044492.0	1.000000e+00	2.235314e+01	3.394046e-09	7.360902e-07	0.000002	0.000006	1.103994e+04
r6_zip5	1044492.0	1.000000e+00	2.448087e+01	1.149107e-08	1.921770e-06	0.000004	0.000020	1.358301e+04
r7_zip5	1044492.0	1.000000e+00	7.881339e+00	9.727321e-03	2.953312e-01	0.473400	0.743049	2.296383e+03
r8_zip5	1044492.0	1.000000e+00	2.425117e+01	4.014298e-09	5.247339e-07	0.000001	0.000005	1.352037e+04
r9_zip5	1044492.0	1.000000e+00	2.635416e+01	9.391163e-09	1.390242e-06	0.000003	0.000012	1.541916e+04
r1_taxclass	1044492.0	1.000000e+00	3.494689e+00	7.711736e-03	5.608901e-01	0.719318	0.992428	1.013376e+03
r2_taxclass	1044492.0	1.000000e+00	5.655854e+01	1.052038e-07	1.204043e-04	0.000145	0.000214	3.637420e+04
r3_taxclass	1044492.0	1.000000e+00	5.194407e+01	2.111598e-07	1.715738e-04	0.000209	0.000337	3.457339e+04
r4_taxclass	1044492.0	1.000000e+00	4.296367e+00	2.663460e-02	5.986215e-01	0.758363	1.017098	1.303541e+03
r5_taxclass	1044492.0	1.000000e+00	4.929886e+01	1.994907e-08	1.647587e-04	0.000198	0.000335	1.897793e+04
r6_taxclass	1044492.0	1.000000e+00	5.211298e+01	6.816485e-08	2.421707e-04	0.000298	0.000570	3.294202e+04
r7_taxclass	1044492.0	1.000000e+00	4.675364e+00	2.318580e-02	6.099784e-01	0.758144	1.000537	2.082944e+03
r8_taxclass	1044492.0	1.000000e+00	5.391943e+01	1.340233e-08	2.162078e-04	0.000261	0.000420	2.839998e+04
r9_taxclass	1044492.0	1.000000e+00	5.730701e+01	5.227777e-08	3.215937e-04	0.000396	0.000659	3.781776e+04
value_ratio	1044492.0	3.255187e+00	1.823360e+01	9.999022e-01	1.119628e+00	1.284059	6.390243	1.000271e+04
size_ratio	1044492.0	3.620129e-01	1.222355e+01	3.046290e-12	1.499625e-01	0.296964	0.463768	1.019954e+04

In [94]:
```
# data.to_csv('NY vars.csv', index=False)
```

In [95]:
```
data.isna().sum().sum()
```

Out[95]: 0

In [96]:
```
# zscale all the variables
data_zs = (data - data.mean()) / data.std()
data_zs_save = data_zs.copy()
data_zs.describe().transpose()
```

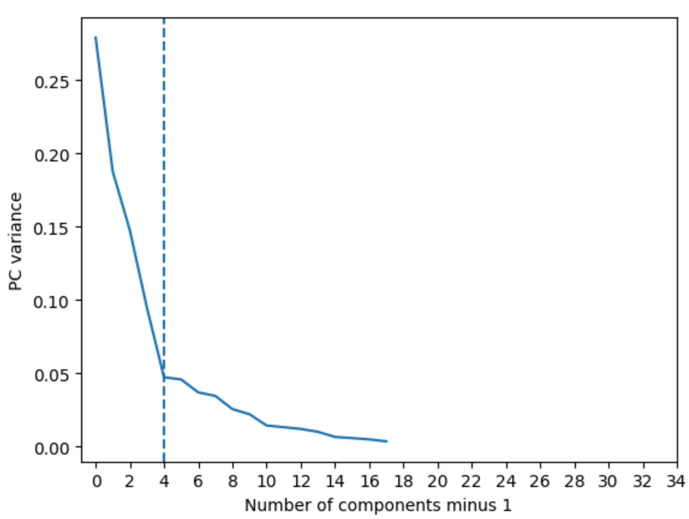
Out[96]:

	count	mean	std	min	25%	50%	75%	max
r1	1044492.0	4.040839e-17	1.0	-0.109710	-0.107234	-0.103264	-0.089429	89.618766
r2	1044492.0	4.568733e-17	1.0	-0.029819	-0.029819	-0.029819	-0.029819	570.816079
r3	1044492.0	2.157665e-17	1.0	-0.030379	-0.030379	-0.030379	-0.030378	703.806269
r4	1044492.0	7.191877e-17	1.0	-0.099190	-0.095721	-0.090187	-0.069369	129.457889
r5	1044492.0	5.272138e-19	1.0	-0.006856	-0.006856	-0.006856	-0.006856	999.957285
r6	1044492.0	-9.974545e-19	1.0	-0.003669	-0.003669	-0.003669	-0.003669	1018.529092
r7	1044492.0	7.710247e-17	1.0	-0.090415	-0.085972	-0.079251	-0.057830	188.569522
r8	1044492.0	-7.136094e-18	1.0	-0.007939	-0.007939	-0.007939	-0.007939	983.508585
r9	1044492.0	2.151373e-18	1.0	-0.003621	-0.003621	-0.003621	-0.003621	1014.170858
r1_zip5	1044492.0	-2.680287e-17	1.0	-0.121079	-0.101978	-0.082559	-0.040863	277.935175
r2_zip5	1044492.0	-1.131979e-17	1.0	-0.053104	-0.053104	-0.053104	-0.053103	589.108225
r3_zip5	1044492.0	4.734720e-18	1.0	-0.049258	-0.049258	-0.049258	-0.049256	653.794235
r4_zip5	1044492.0	-3.327910e-17	1.0	-0.150722	-0.120042	-0.091124	-0.042343	317.541865
r5_zip5	1044492.0	4.367371e-18	1.0	-0.044736	-0.044736	-0.044736	-0.044736	493.842636
r6_zip5	1044492.0	9.523863e-20	1.0	-0.040848	-0.040848	-0.040848	-0.040847	554.800949
r7_zip5	1044492.0	-2.715661e-17	1.0	-0.125648	-0.089410	-0.066816	-0.032602	291.242767
r8_zip5	1044492.0	8.925220e-18	1.0	-0.041235	-0.041235	-0.041235	-0.041235	557.472942
r9_zip5	1044492.0	-8.476238e-18	1.0	-0.037945	-0.037945	-0.037945	-0.037944	585.036939
r1_taxclass	1044492.0	1.717969e-16	1.0	-0.283942	-0.125651	-0.080317	-0.002167	289.689841
r2_taxclass	1044492.0	-2.693893e-18	1.0	-0.017681	-0.017679	-0.017678	-0.017677	643.107017
r3_taxclass	1044492.0	1.768717e-19	1.0	-0.019251	-0.019248	-0.019247	-0.019245	665.569626
r4_taxclass	1044492.0	2.274843e-17	1.0	-0.226555	-0.093423	-0.056242	0.003980	303.172722
r5_taxclass	1044492.0	3.809545e-19	1.0	-0.020284	-0.020281	-0.020280	-0.020278	384.936473
r6_taxclass	1044492.0	-9.204133e-18	1.0	-0.019189	-0.019184	-0.019183	-0.019178	632.107821
r7_taxclass	1044492.0	8.707532e-19	1.0	-0.208928	-0.083421	-0.051730	0.000115	445.300868
r8_taxclass	1044492.0	9.278963e-18	1.0	-0.018546	-0.018542	-0.018541	-0.018538	526.692811
r9_taxclass	1044492.0	-8.673518e-18	1.0	-0.017450	-0.017444	-0.017443	-0.017438	659.897578
value_ratio	1044492.0	3.619068e-17	1.0	-0.123688	-0.117122	-0.108104	0.171938	548.407916
size_ratio	1044492.0	1.021774e-17	1.0	-0.029616	-0.017348	-0.005322	0.008325	834.387426

In [97]:

```
# do a complete PCA and look at the scree and cumulative variance plots
pca = PCA(n_components = .99, svd_solver = 'full')
pca.fit(data_zs)
plt.plot(pca.explained_variance_ratio_)
plt.xlabel('Number of components minus 1')
plt.ylabel('PC variance')
plt.xticks(np.arange(0, 36, step=2))
plt.axvline(x=4, linestyle='--')
```

Out[97]: <matplotlib.lines.Line2D at 0x314b852a0>

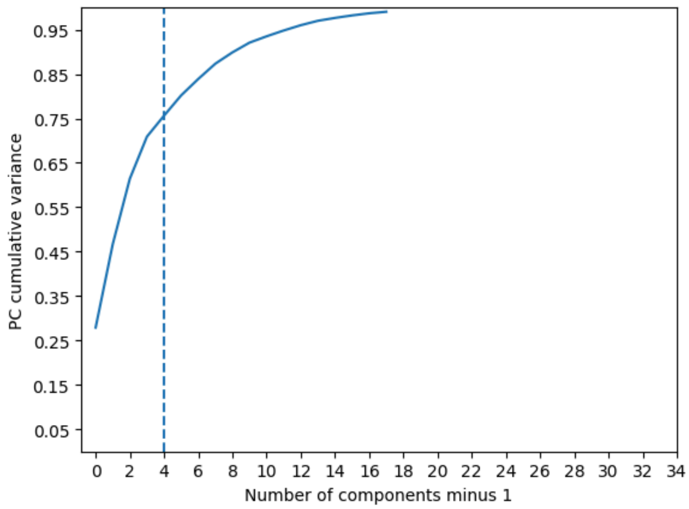


In [98]:

```
plt.xlabel('Number of components minus 1')
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.ylabel('PC cumulative variance')
plt.yticks(np.arange(0.05, 1.1, step=.1))
```

```
plt.xticks(np.arange(0, 36, step=2))
plt.axvline(x=4, linestyle='--')
plt.ylim(0,1)
```

Out[98]: (0.0, 1.0)



In [99]:

```
%%time
# now redo the PCA but just keep the top few PCs
data_zs = data_zs_save.copy()
pca = PCA(n_components = 5, svd_solver = 'full')
princ_comps = pca.fit_transform(data_zs)
pca.n_components_
```

```
CPU times: user 4.28 s, sys: 181 ms, total: 4.46 s
Wall time: 756 ms
```

Out[99]: 5

In [100…

```
print(np.cumsum(pca.explained_variance_ratio_))
```

```
[0.27908514 0.4667656  0.61435657 0.7090587  0.7562435 ]
```

In [101…

```
data_pca = pd.DataFrame(princ_comps, columns = ['PC' + str(i) for i in range(1, pca.n_components_+1)])
data_pca.shape
```

Out[101… (1044492, 5)

In [102…

```
data_pca.head(5)
```

Out[102…

	PC1	PC2	PC3	PC4	PC5
0	0.625151	0.037534	0.077495	0.126964	0.863770
1	0.962183	0.258705	0.119726	0.163168	1.152296
2	1.667141	0.690240	0.258814	0.152342	2.000525
3	0.062475	-0.328045	0.006789	0.066493	0.381345
4	0.312960	-0.167336	0.038376	0.093430	0.596509

In [103…

```
data_pca.describe()
```

Out[103…

	PC1	PC2	PC3	PC4	PC5
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06
mean	2.995663e-16	-4.342881e-17	1.310347e-16	7.790520e-17	1.170075e-18
std	2.844902e+00	2.332967e+00	2.068849e+00	1.657215e+00	1.169769e+00
min	-5.015535e-01	-1.225369e+02	-6.098382e+02	-5.798701e+02	-2.231438e+02
25%	-1.243610e-01	-2.692587e-01	-4.136593e-02	4.618834e-02	-8.575840e-02
50%	-1.199425e-01	-2.119116e-01	-3.765687e-02	5.218774e-02	-4.856385e-02
75%	-9.894908e-02	-1.209430e-01	-3.125217e-02	5.267692e-02	1.064451e-01
max	2.078522e+03	4.562133e+02	9.392000e+02	1.101210e+03	4.471563e+02

zscale the pcs.

I do this (make all the retained PCs equally important) if I only keep a small number of PCs. Alternatively you can keep maybe up to 6 to 8 or so, and don't do this second z scale. I prefer to keep a somewhat small number of PCs and then make them all equally important via zscaling. This second zscale step makes the later Minkowski distance to be similar to a Mahalanobis distance. Many people don't do this second zscaling but I think it's better.

My point of view: Keep the top few PC's thinking that each of these is measureing a mostly independent phenonenom (since they're orthogonal). Then make them all equally important via zscale.

My choice: keep only a few top PCs and zscale them. Alternative: keep some more of the PCs and don't zscale the PCs. Then these later PCs don't add much.

In [104…

```
data_pca_zs = (data_pca - data_pca.mean()) / data_pca.std()
data_pca_zs.describe()
```

Out[104…

	PC1	PC2	PC3	PC4	PC5
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06
mean	-3.394577e-18	2.108855e-18	-4.761931e-20	1.568036e-18	-1.175517e-17
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-1.762990e-01	-5.252409e+01	-2.947717e+02	-3.499064e+02	-1.907589e+02
25%	-4.371363e-02	-1.154147e-01	-1.999466e-02	2.787106e-02	-7.331226e-02
50%	-4.216051e-02	-9.083353e-02	-1.820184e-02	3.149123e-02	-4.151577e-02
75%	-3.478119e-02	-5.184085e-02	-1.510606e-02	3.178641e-02	9.099668e-02
max	7.306128e+02	1.955507e+02	4.539721e+02	6.644945e+02	3.822603e+02

In [105…

```
data_pca_zs.shape
```

Out[105…

```
(1044492, 5)
```

In [106…

```
data_pca_zs.head(5)
```

Out[106…

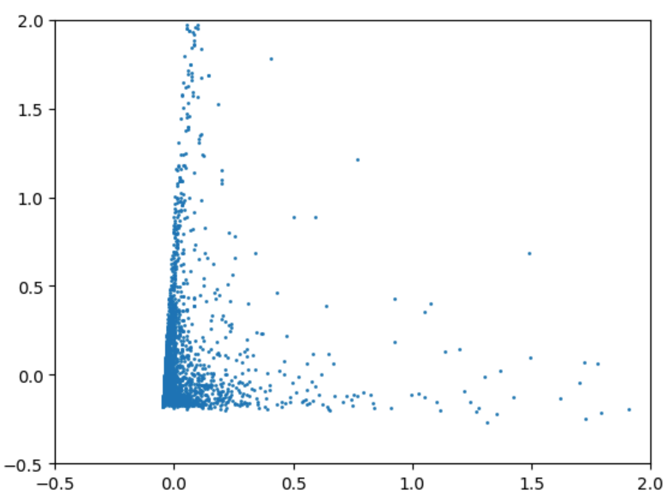
	PC1	PC2	PC3	PC4	PC5
0	0.219744	0.016089	0.037458	0.076613	0.738411
1	0.338213	0.110891	0.057871	0.098459	0.985063
2	0.586010	0.295864	0.125100	0.091926	1.710188
3	0.021960	-0.140613	0.003281	0.040123	0.326000
4	0.110007	-0.071727	0.018550	0.056378	0.509937

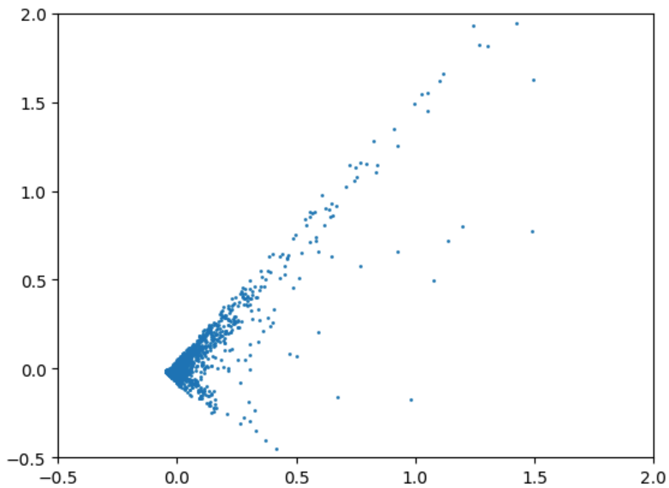
Look to see if clustering might be indicated

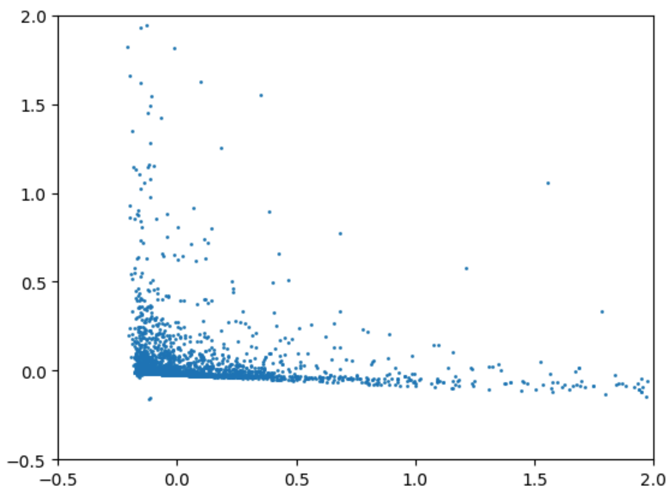
In [107…

```
samp = data_pca_zs.sample(n=10000)

plt.xlim([-.5,2])
plt.ylim([-.5,2])
plt.scatter(samp['PC1'],samp['PC2'],s=1)
plt.show()
plt.xlim([-.5,2])
plt.ylim([-.5,2])
plt.scatter(samp['PC1'],samp['PC3'],s=1)
plt.show()
plt.xlim([-.5,2])
plt.ylim([-.5,2])
plt.scatter(samp['PC2'],samp['PC3'],s=1)
plt.show()
```







From these three plots I see PC1 and PC2 are relatively uncorrelated and PC3 is largely correlated to PC1.

In the plot of the first 2 PCs, which are the dominant dimensions, it doesn't look like the data has any natural subgroups/clusters. So it really doesn't make sense to separate the data into clusters/goups for the rest of the work.

Now calculate two unsupervised fraud scores

In [108…
```
# Set the powers for the two Minkowski distances. The final results are relatively insensitive to these choices.
# Reasonable choices are anywhere from 1 to about 4.
# The higher the power the more the distance measure focuses on the large dimensional displacements.
p1 = 2
p2 = 2
ntop = 10000
```

Calculate score 1

In [109…
```
oop1 = 1/p1
score1 = (((data_pca_zs).abs()**p1).sum(axis=1))**oop1
score1.head(10)
```

Out[109…
```
0    0.775287
1    1.053602
2    1.838420
3    0.357982
4    0.529910
5    0.511855
6    1.381865
7    1.348378
8    1.348378
9    1.348378
dtype: float64
```

In [110…
```
data_pca_zs.head(10)
```

Out[110...

	PC1	PC2	PC3	PC4	PC5
0	0.219744	0.016089	0.037458	0.076613	0.738411
1	0.338213	0.110891	0.057871	0.098459	0.985063
2	0.586010	0.295864	0.125100	0.091926	1.710188
3	0.021960	-0.140613	0.003281	0.040123	0.326000
4	0.110007	-0.071727	0.018550	0.056378	0.509937
5	0.101474	-0.078555	0.017079	0.054804	0.492171
6	0.475136	0.220461	0.081463	0.123708	1.270139
7	0.461246	0.209346	0.079070	0.121146	1.241218
8	0.461246	0.209346	0.079070	0.121146	1.241218
9	0.461246	0.209346	0.079070	0.121146	1.241218

In [111...

```
score1.max()
```

Out[111...

```
1006.4007283462964
```

Autoencoder for score 2

In [112...

```
%%time
# you don't need the autoencoder to be really good, just a little bit trained so it can find the really unusual records
# you can set max_iter to 100 and it takes about a minute, or 500 and it takes about 7 minutes. But even 50 is good enough.
NNmodel = MLPRegressor(hidden_layer_sizes=(3),activation='logistic',max_iter=100,random_state=1)
NNmodel.fit(data_pca_zs,data_pca_zs)
```

```
CPU times: user 36.2 s, sys: 82.2 ms, total: 36.3 s
Wall time: 36 s
/opt/homebrew/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimizer:
Maximum iterations (100) reached and the optimization hasn't converged yet.
  warnings.warn(
```

Out[112...

```
▾                           MLPRegressor
MLPRegressor(activation='logistic', hidden_layer_sizes=3, max_iter=100,
             random_state=1)
```

In [113...

```
# calculate score 2 as the error of an autoencoder
# Again we'll use Minkowski distance for the error (difference betwen the input and output vectors)
pca_out = NNmodel.predict(data_pca_zs)
error = pca_out - data_pca_zs
oop2 = 1/p2
score2 = ((error.abs()**p2).sum(axis=1))**oop2
```

In [114...

```
scores = pd.DataFrame(score1)
scores.columns=['score1']
scores['score2'] = score2
scores['RECORD'] = save_record
scores.head(10)
```

Out[114...

	score1	score2	RECORD
0	0.775287	0.226800	9
1	1.053602	0.313333	10
2	1.838420	0.521354	11
3	0.357982	0.095664	12
4	0.529910	0.149259	13
5	0.511855	0.143530	14
6	1.381865	0.413418	15
7	1.348378	0.403301	16
8	1.348378	0.403301	17
9	1.348378	0.403301	18

In [115...

```
# do a rank-order scaling. Replace the score with the sorted rank order
scores['score1 rank'] = scores['score1'].rank()
scores['score2 rank'] = scores['score2'].rank()
scores.head(20)
```

Out[115...

	score1	score2	RECORD	score1 rank	score2 rank
0	0.775287	0.226800	9	1004949.0	999931.0
1	1.053602	0.313333	10	1015068.0	1012941.0
2	1.838420	0.521354	11	1028447.0	1028273.0
3	0.357982	0.095664	12	956629.0	923125.0
4	0.529910	0.149259	13	987777.0	977115.0
5	0.511855	0.143530	14	985854.0	974487.0
6	1.381865	0.413418	15	1022390.0	1022270.0
7	1.348378	0.403301	16	1021860.0	1021564.0
8	1.348378	0.403301	17	1021858.0	1021562.0
9	1.348378	0.403301	18	1021856.0	1021561.0
10	1.348378	0.403301	19	1021859.0	1021563.0
11	1.348378	0.403301	20	1021861.0	1021565.0
12	1.348378	0.403301	21	1021862.0	1021566.0
13	1.348378	0.403301	22	1021857.0	1021560.0
14	1.347900	0.403157	23	1021852.0	1021554.0
15	1.332125	0.398384	24	1021562.0	1021220.0
16	1.332125	0.398384	25	1021561.0	1021221.0
17	1.371337	0.410240	26	1022223.0	1022047.0
18	1.371337	0.410240	27	1022226.0	1022048.0
19	1.371337	0.410240	28	1022225.0	1022045.0

In [116...

```
# calculate the final score as the average of the two scores
# you could do other possible combinations of these if you want
# You could do different weightings, or a max or min
weight = .5
scores['final'] = (weight*scores['score1 rank'] + (1-weight)*scores['score2 rank'])
scores_sorted = scores.sort_values(by='final', ascending=False)
scores_sorted.head(20)
```

Out[116...

	score1	score2	RECORD	score1 rank	score2 rank	final
897397	1006.400728	1005.288761	917942	1044492.0	1044492.0	1044492.0
544655	639.468504	587.387598	561383	1044491.0	1044491.0	1044491.0
1028917	485.727334	446.325893	1053832	1044490.0	1044490.0	1044490.0
148176	417.114776	381.496289	151044	1044489.0	1044489.0	1044489.0
383033	384.202091	348.415672	398266	1044488.0	1044488.0	1044488.0
676142	321.959426	289.530523	694272	1044487.0	1044487.0	1044487.0
383044	293.476973	259.276466	398284	1044486.0	1044486.0	1044486.0
29547	275.488029	242.650688	30042	1044484.0	1044485.0	1044484.5
230836	282.996129	241.479071	241946	1044485.0	1044484.0	1044484.5
1029144	255.549985	217.356242	1054166	1044481.0	1044483.0	1044482.0
960229	260.369666	201.150553	982930	1044483.0	1044481.0	1044482.0
632013	244.472011	202.883558	649717	1044480.0	1044482.0	1044481.0
957705	259.817225	200.663382	980276	1044482.0	1044480.0	1044481.0
668926	244.196260	195.497789	686922	1044479.0	1044479.0	1044479.0
166271	202.812446	168.280960	170125	1044477.0	1044478.0	1044477.5
973750	205.788741	162.505238	996722	1044478.0	1044477.0	1044477.5
792379	202.414160	155.082585	811390	1044476.0	1044476.0	1044476.0
443551	186.184642	145.608953	459429	1044475.0	1044475.0	1044475.0
55457	160.455860	126.580953	56136	1044473.0	1044474.0	1044473.5
401073	162.479210	117.264941	416506	1044474.0	1044473.0	1044473.5

In [117...

```
scores_sorted.tail(10)
```

Out [117…

	score1	score2	RECORD	score1 rank	score2 rank	final
561760	0.015387	0.006203	578627	50.0	8.0	29.0
561777	0.015264	0.006158	578644	49.0	6.0	27.5
561799	0.015149	0.006178	578666	48.0	7.0	27.5
560940	0.011941	0.006786	577791	26.0	28.0	27.0
561756	0.015028	0.006139	578623	47.0	5.0	26.0
561007	0.011907	0.006775	577858	25.0	27.0	26.0
560953	0.012098	0.006656	577804	28.0	22.0	25.0
561805	0.014898	0.006128	578672	45.0	4.0	24.5
561767	0.014443	0.006100	578634	40.5	2.5	21.5
561770	0.014443	0.006100	578637	40.5	2.5	21.5

In [118…

```
scores_sorted.describe()
```

Out [118…

	score1	score2	RECORD	score1 rank	score2 rank	final
count	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06	1.044492e+06
mean	2.936232e-01	1.018640e-01	5.368071e+05	5.222465e+05	5.222465e+05	5.222465e+05
std	2.216706e+00	1.723090e+00	3.080025e+05	3.015190e+05	3.015190e+05	2.983085e+05
min	8.744326e-03	5.374115e-03	9.000000e+00	1.000000e+00	1.000000e+00	2.150000e+01
25%	1.225182e-01	3.584060e-02	2.729098e+05	2.611238e+05	2.611241e+05	2.677320e+05
50%	1.416069e-01	4.230228e-02	5.387725e+05	5.222455e+05	5.222468e+05	5.215945e+05
75%	2.084633e-01	5.958568e-02	8.022752e+05	7.833692e+05	7.833235e+05	7.842996e+05
max	1.006401e+03	1.005289e+03	1.070994e+06	1.044492e+06	1.044492e+06	1.044492e+06

In [119…

```
scores_sorted.set_index('RECORD', drop=True, inplace=True)
scores_sorted.head(10)
```

Out [119…

	score1	score2	score1 rank	score2 rank	final
RECORD					
917942	1006.400728	1005.288761	1044492.0	1044492.0	1044492.0
561383	639.468504	587.387598	1044491.0	1044491.0	1044491.0
1053832	485.727334	446.325893	1044490.0	1044490.0	1044490.0
151044	417.114776	381.496289	1044489.0	1044489.0	1044489.0
398266	384.202091	348.415672	1044488.0	1044488.0	1044488.0
694272	321.959426	289.530523	1044487.0	1044487.0	1044487.0
398284	293.476973	259.276466	1044486.0	1044486.0	1044486.0
30042	275.488029	242.650688	1044484.0	1044485.0	1044484.5
241946	282.996129	241.479071	1044485.0	1044484.0	1044484.5
1054166	255.549985	217.356242	1044481.0	1044483.0	1044482.0

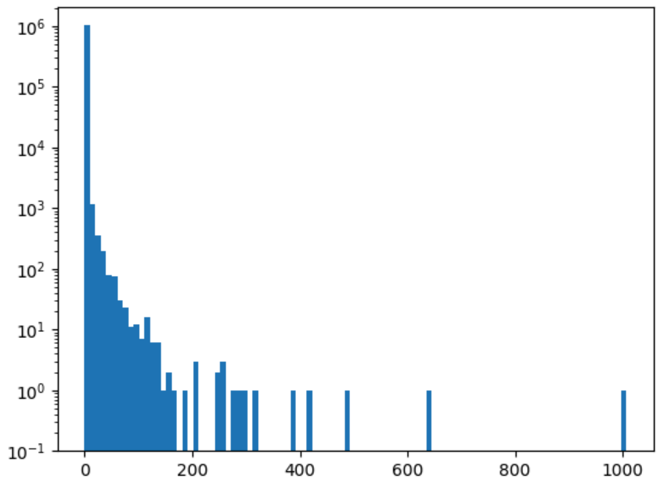
Look at the score distributions

In [120…

```
sc1max = int(score1.max())
plt.hist(score1, bins =100, range=(0,sc1max+1))
plt.yscale('log')
plt.ylim(ymin=.1)
```

Out [120…

```
(0.1, 2084432.0408312716)
```



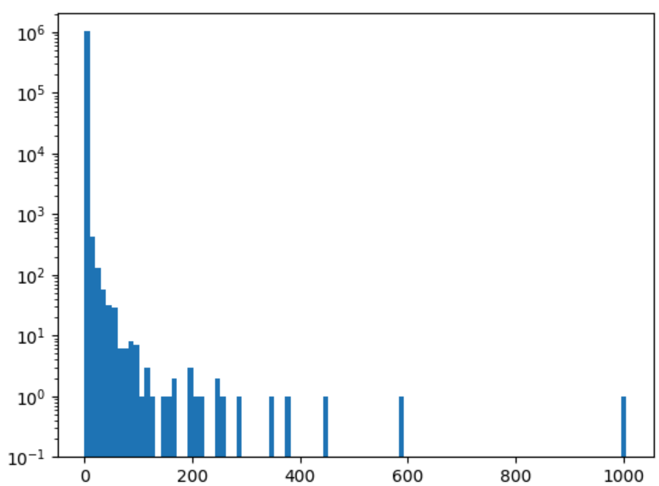
In [121…
```
sc2max = int(score2.max())
sc2max
```

Out[121… 1005

In [122…
```
sc2max = int(score2.max())
print(sc2max)
plt.hist(score2, bins =100, range=(0,sc2max+1))
plt.yscale('log')
plt.ylim(ymin=.1)
```

```
1005
```

Out[122… (0.1, 2087077.3542771842)

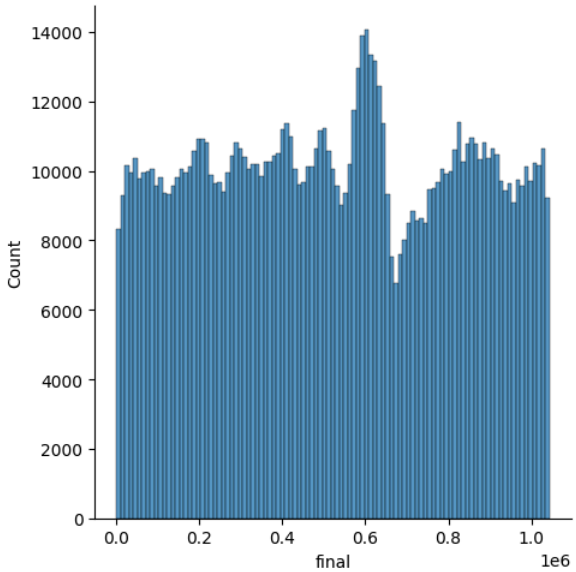


The flatter the next plot, the more similar are the two scores. If the two scores are very similar then the rank order hardly changes betwen the two scores and the plot is flat. The plot basically shows how much and in what sore regions the two scores differ.

In [123…
```
sns.displot(scores['final'])
```

```
/opt/homebrew/lib/python3.10/site-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[123… <seaborn.axisgrid.FacetGrid at 0x317372710>



In [124…
```
top_records = scores_sorted.head(ntop).index
print(top_records)
```

```
Index([ 917942,  561383, 1053832,  151044,  398266,  694272,  398284,   30042,
        241946, 1054166,
       ...
         78352,   78353,   78354,   78356,   78357,   78358,   78359,   78360,
         78361,   78363],
      dtype='int64', name='RECORD', length=10000)
```

In [125…
```
data_zs['RECORD'] = save_record
data_zs.set_index('RECORD', inplace=True, drop=True)
data_zs.head()
```

Out[125…

	r1	r2	r3	r4	r5	r6	r7	r8	r9	r1_zip5	...	r2_taxclass	r3_taxclass	r4_taxcla
RECORD														
9	-0.090143	1.323640	0.198669	0.110106	0.347933	0.024674	0.368022	0.366047	0.026625	-0.079731	...	0.139560	0.015751	-0.1057
10	-0.077342	1.926186	0.300639	0.209156	0.505883	0.037293	0.580594	0.532542	0.040091	-0.061051	...	0.209562	0.031334	-0.0492
11	-0.041635	3.607071	0.585097	0.226023	0.532780	0.039441	1.173592	0.997001	0.077654	-0.008939	...	0.404841	0.074804	-0.0395
12	-0.107457	0.317376	0.028378	-0.055308	0.084156	0.003602	0.013022	0.087997	0.004138	-0.105000	...	0.022655	-0.010272	-0.2001
13	-0.101999	0.765502	0.104215	0.018358	0.201626	0.012986	0.171116	0.211823	0.014152	-0.097035	...	0.074717	0.001317	-0.1581

5 rows × 29 columns

In [126…
```
scores.set_index('RECORD',inplace=True)
scores.drop(columns=['score1','score2'],inplace=True)
scores.head(30)
```

Out[126…

RECORD	score1 rank	score2 rank	final
9	1004949.0	999931.0	1002440.0
10	1015068.0	1012941.0	1014004.5
11	1028447.0	1028273.0	1028360.0
12	956629.0	923125.0	939877.0
13	987777.0	977115.0	982446.0
14	985854.0	974487.0	980170.5
15	1022390.0	1022270.0	1022330.0
16	1021860.0	1021564.0	1021712.0
17	1021858.0	1021562.0	1021710.0
18	1021856.0	1021561.0	1021708.5
19	1021859.0	1021563.0	1021711.0
20	1021861.0	1021565.0	1021713.0
21	1021862.0	1021566.0	1021714.0
22	1021857.0	1021560.0	1021708.5
23	1021852.0	1021554.0	1021703.0
24	1021562.0	1021220.0	1021391.0
25	1021561.0	1021221.0	1021391.0
26	1022223.0	1022047.0	1022135.0
27	1022226.0	1022048.0	1022137.0
28	1022225.0	1022045.0	1022135.0
29	1022224.0	1022046.0	1022135.0
30	1022192.0	1021995.0	1022093.5
31	1011369.0	1008111.0	1009740.0
32	1011368.0	1008112.0	1009740.0
33	1011360.0	1008114.0	1009737.0
34	1011367.0	1008113.0	1009740.0
35	1011366.0	1008120.0	1009743.0
36	1011365.0	1008115.0	1009740.0
37	1011362.5	1008117.5	1009740.0
38	1011362.5	1008117.5	1009740.0

In [127…

```
scores.tail(30)
```

Out[127...

RECORD	score1 rank	score2 rank	final
1070965	576550.0	674980.0	625765.0
1070966	681323.0	705298.0	693310.5
1070967	914713.0	878871.0	896792.0
1070968	964628.0	927912.0	946270.0
1070969	918060.0	879149.0	898604.5
1070970	794971.0	805418.0	800194.5
1070971	899298.0	869720.0	884509.0
1070972	459475.0	554725.0	507100.0
1070973	812270.0	817371.0	814820.5
1070974	784986.0	796039.0	790512.5
1070975	619920.0	685333.0	652626.5
1070976	836533.0	832043.0	834288.0
1070977	713364.0	732667.0	723015.5
1070978	439194.0	551848.0	495521.0
1070979	538089.0	660523.0	599306.0
1070980	767763.0	876458.0	822110.5
1070981	608542.0	687231.0	647886.5
1070982	621420.0	690839.0	656129.5
1070983	426086.0	537059.0	481572.5
1070984	883270.0	860360.0	871815.0
1070985	728544.0	748375.0	738459.5
1070986	774862.0	788553.0	781707.5
1070987	490705.0	620994.0	555849.5
1070988	460608.0	533917.0	497262.5
1070989	895192.0	867821.0	881506.5
1070990	717500.0	737804.0	727652.0
1070991	985216.0	962129.0	973672.5
1070992	963758.0	933373.0	948565.5
1070993	700096.0	719210.0	709653.0
1070994	490920.0	527459.0	509189.5

In [128...

```
NY_data_with_scores = NY_data_orig.join(scores, on='RECORD')
NY_data_with_scores['final'].fillna(1,inplace=True)
NY_data_with_scores
```

```
/var/folders/xx/xzxtm5cd4_qc7z_7jjysft080000gn/T/ipykernel_25633/3385311105.py:2: FutureWarning: A value is trying to be set on a copy of
a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].meth
od(value) instead, to perform the operation inplace on the original object.


  NY_data_with_scores['final'].fillna(1,inplace=True)
```

Out[128…

	RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	...	AVTOT2	EXLAND2	EXTOT
0	1	1000010101	1	1	101	NaN	U S GOVT LAND & BLDGS	P7	4	500	...	8613000.0	3775500.0	8613000
1	2	1000010201	1	1	201	NaN	U S GOVT LAND & BLDGS	Z9	4	27	...	80690400.0	11111400.0	80690400
2	3	1000020001	1	2	1	NaN	DEPT OF GENERAL SERVI	Y7	4	709	...	40179510.0	32321790.0	40179510
3	4	1000020023	1	2	23	NaN	DEPARTMENT OF BUSINES	T2	4	793	...	15750000.0	13644000.0	15750000
4	5	1000030001	1	3	1	NaN	PARKS AND RECREATION	Q1	4	323	...	107758350.0	106348680.0	107758350
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1070989	1070990	5080500083	5	8050	83	NaN	TOBIN, GALE	A1	1	60	...	NaN	NaN	Na
1070990	1070991	5080500086	5	8050	86	NaN	SHERRI MILINAZZO	A1	1	62	...	NaN	NaN	Na
1070991	1070992	5080500089	5	8050	89	NaN	JOHN GERVASI	A1	1	53	...	NaN	NaN	Na
1070992	1070993	5080500092	5	8050	92	NaN	RITA M MOOG	A1	1	52	...	NaN	NaN	Na
1070993	1070994	5080500094	5	8050	94	NaN	EDWARD DONOHUE	A1	1	50	...	NaN	NaN	Na

1070994 rows × 35 columns

OK, now let's look at the top records, see if they make sense, and then we can go back and adjust the algorithm/variables if we want.

In [129…

```
NY_data_scored_and_sorted = NY_data_with_scores.sort_values(by=['final','RECORD'], ascending = [False,True])
NY_data_scored_zs = NY_data_with_scores.join(data_zs, on='RECORD')
NY_data_scored_zs.set_index('RECORD',inplace=True)
NY_data_scored_zs.head(20)
```

Out[129…

RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	LTDEPTH	...	r2_taxclass	r3_taxclass	r4_taxclass
1	1000010101	1	1	101	NaN	U S GOVT LAND & BLDGS	P7	4	500	1046	...	NaN	NaN	NaN
2	1000010201	1	1	201	NaN	U S GOVT LAND & BLDGS	Z9	4	27	0	...	NaN	NaN	NaN
3	1000020001	1	2	1	NaN	DEPT OF GENERAL SERVI	Y7	4	709	564	...	NaN	NaN	NaN
4	1000020023	1	2	23	NaN	DEPARTMENT OF BUSINES	T2	4	793	551	...	NaN	NaN	NaN
5	1000030001	1	3	1	NaN	PARKS AND RECREATION	Q1	4	323	1260	...	NaN	NaN	NaN
6	1000030002	1	3	2	NaN	PARKS AND RECREATION	Q1	4	496	76	...	NaN	NaN	NaN
7	1000030003	1	3	3	NaN	PARKS AND RECREATION	Q1	4	180	370	...	NaN	NaN	NaN
8	1000030010	1	3	10	NaN	DEPT RE-CITY OF NY	Z9	4	362	177	...	NaN	NaN	NaN
9	1000041001	1	4	1001	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.139560	0.015751	-0.105760
10	1000041002	1	4	1002	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.209562	0.031334	-0.049223
11	1000041003	1	4	1003	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.404841	0.074804	-0.039595
12	1000041004	1	4	1004	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.022655	-0.010272	-0.200177
13	1000041005	1	4	1005	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.074717	0.001317	-0.158130
14	1000041006	1	4	1006	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.069675	0.000194	-0.162202
15	1000041007	1	4	1007	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.290468	0.049344	0.016122
16	1000041008	1	4	1008	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.282260	0.047517	0.009493
17	1000041009	1	4	1009	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.282260	0.047517	0.009493
18	1000041010	1	4	1010	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.282260	0.047517	0.009493
19	1000041011	1	4	1011	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.282260	0.047517	0.009493
20	1000041012	1	4	1012	NaN	TRZ HOLDINGS, LLC	R5	4	0	0	...	0.282260	0.047517	0.009493

20 rows × 63 columns

In [130…

```
NY_data_scored_zs_sorted = NY_data_scored_zs.sort_values(by=['final','RECORD'], ascending = [False,True])
NY_data_top_n = NY_data_scored_zs_sorted.head(ntop)
NY_data_top_n
```

Out[130...

RECORD	BBLE	BORO	BLOCK	LOT	EASEMENT	OWNER	BLDGCL	TAXCLASS	LTFRONT	LTDEPTH	...	r2_taxclass	r3_taxclass	r4_taxclass
917942	4142600001	4	14260	1	NaN	LOGAN PROPERTY, INC.	T1	4	4910	0	...	14.890970	42.294588	4.555461
561383	3084700055	3	8470	55	NaN	YILDIZ HOLDING A.S.	K6	4	930	650	...	10.266350	38.898361	-0.122604
1053832	5064310001	5	6431	1	NaN	MARKOW, REGINA	A3	1	615	1054	...	643.107017	665.569626	2.296469
151044	2024930001	2	2493	1	NaN	NaN	Q6	4	798	611	...	66.301360	107.539139	0.032199
398266	3044520090	3	4452	90	NaN	STARRETT CITY, INC.	Z0	1	907	201	...	342.137331	341.483632	0.283983
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
78345	1011712303	1	1171	2303	NaN	SHAPIRO, ROBERT I	R5	4	0	0	...	-0.017617	-0.019230	1.770533
78346	1011712304	1	1171	2304	NaN	SHAPIRO, ROBERT I	R5	4	0	0	...	-0.017617	-0.019230	1.770533
78347	1011712305	1	1171	2305	NaN	DINSMORE, MARIANNE	R5	4	0	0	...	-0.017617	-0.019230	1.770533
78348	1011712306	1	1171	2306	NaN	MORAN, TREVOR C	R5	4	0	0	...	-0.017617	-0.019230	1.770533
78349	1011712307	1	1171	2307	NaN	SCHWARZ, JEFFREY A	R5	4	0	0	...	-0.017617	-0.019230	1.770533

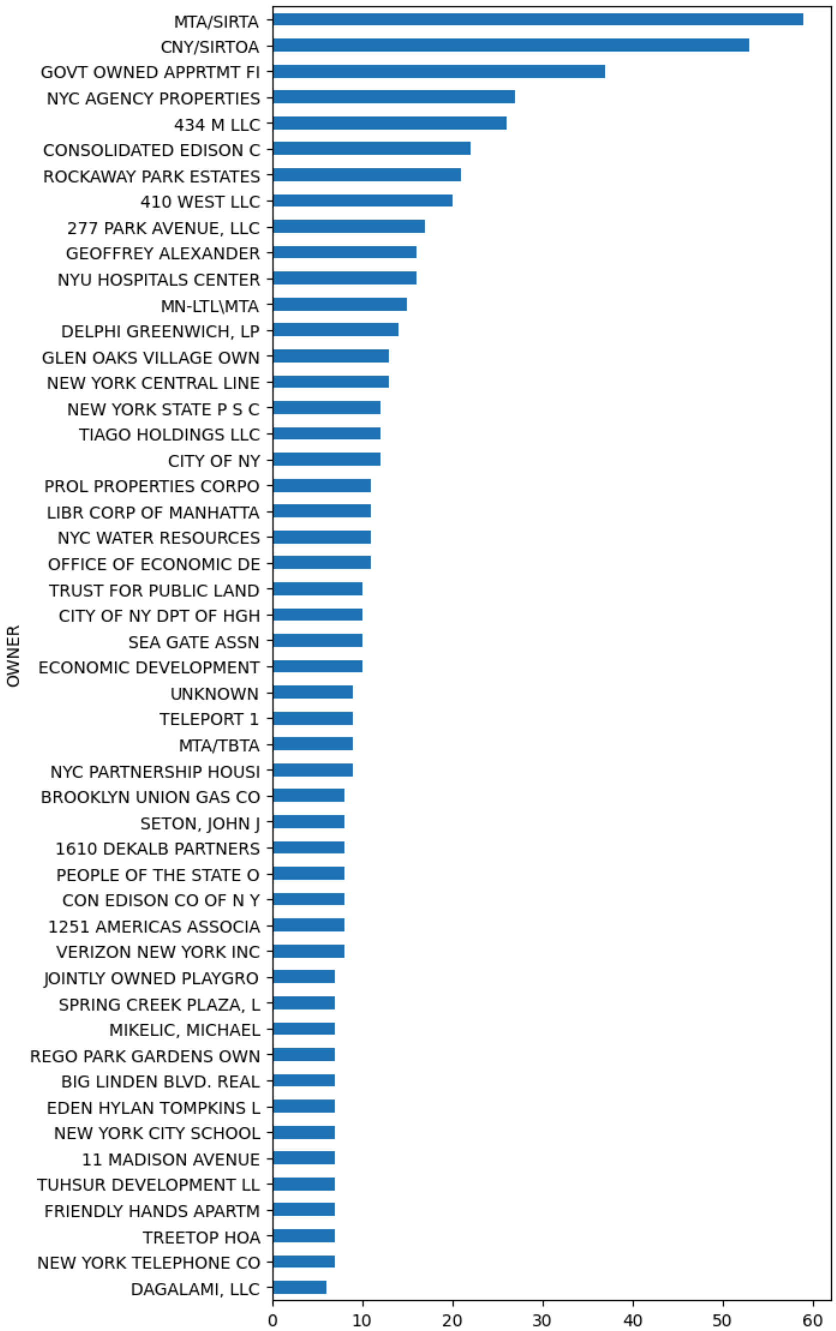
10000 rows × 63 columns

In [131...

```
# you can look at this list and add some to the exclusions if you want
plt.figure(figsize=(6,14))
NY_data_top_n['OWNER'].value_counts().head(50).sort_values().plot(kind='barh')
```

Out[131...

```
<AxesSubplot: ylabel='OWNER'>
```



In [132…
```
NY_data_top_n.shape
```

Out[132… (10000, 63)

In [133…
```
# Write out some data sets that you can use for examination.
# Sometimes we want to ignore the ones with zero sizes since they're somewhat trivial cases.
NY_data_top_n.to_excel('NY_top_with_zs.xlsx', index=True)
NY_top_lotsize_ne_0 = NY_data_top_n[NY_data_top_n['LTFRONT'] != 0]
NY_top_lotsize_ne_0.to_excel('NY_top_lotsize_ne_0.xlsx', index=True)
NY_top_sizes_ne_0 = NY_top_lotsize_ne_0[NY_top_lotsize_ne_0['BLDDEPTH'] != 0]
NY_top_sizes_ne_0.to_excel('NY_top_sizes_ne_0.xlsx', index=True)
```

In [134…
```
nfields = 34
data_base_vars = NY_data_top_n.iloc[:,nfields:nfields+9]
data_base_vars.head()
```

Out[134...

RECORD	r1	r2	r3	r4	r5	r6	r7	r8	r9
917942	-0.082426	128.297458	276.861374	8.276290	999.957285	1018.529092	16.688653	983.508585	1014.170858
561383	-0.093334	88.490709	254.637267	0.080595	22.632781	30.743050	0.315018	24.451939	33.626160
1053832	-0.073933	6.935168	20.007424	0.127208	0.084358	0.120209	0.507323	0.028299	0.046203
151044	0.085466	570.816079	703.806269	0.351801	43.917052	25.561785	3.284346	157.727464	92.940547
398266	-0.095121	3.675691	10.250694	-0.063432	0.118890	0.161028	0.010209	0.044727	0.066212

In [135...

```
data_all_vars = NY_data_top_n.iloc[:,nfields:]
data_all_vars.head()
```

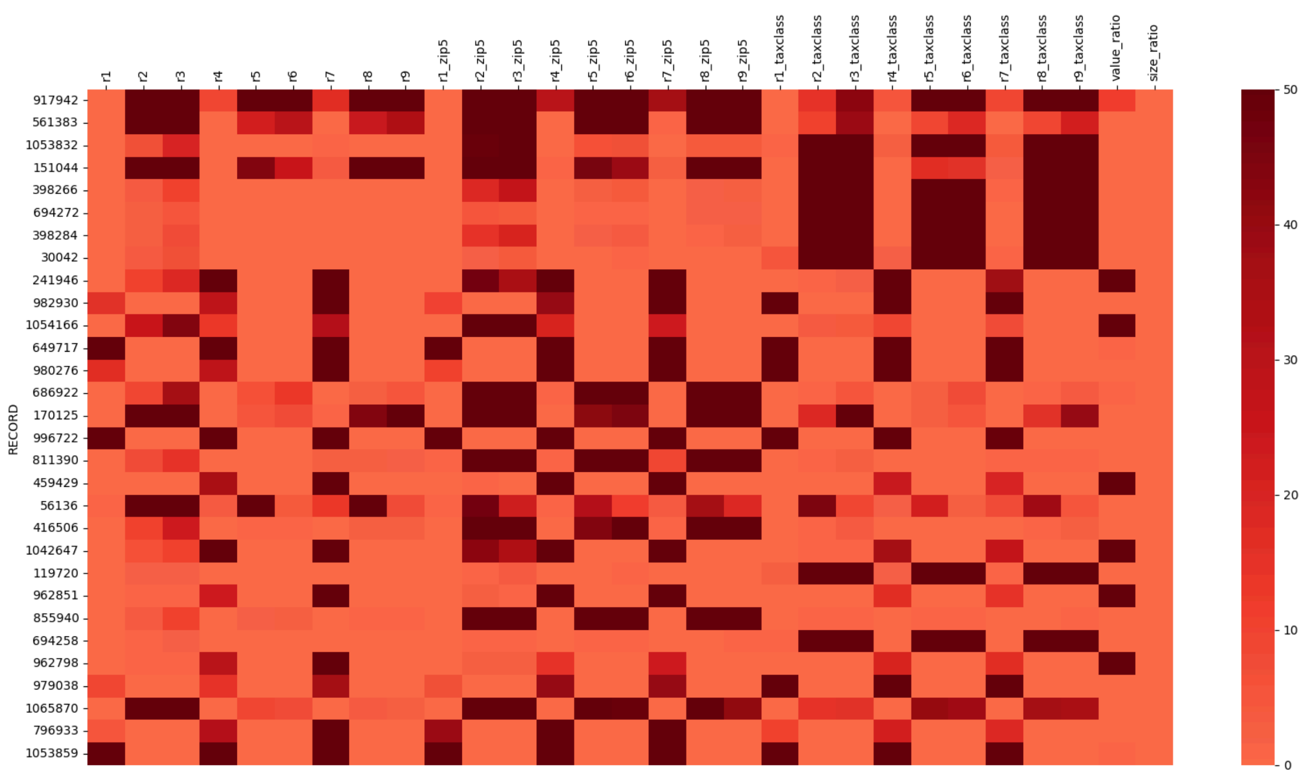
Out[135...

RECORD	r1	r2	r3	r4	r5	r6	r7	r8	r9	r1_zip5	...	r2_taxclass	r3_tax
917942	-0.082426	128.297458	276.861374	8.276290	999.957285	1018.529092	16.688653	983.508585	1014.170858	0.084078	...	14.890970	42.2
561383	-0.093334	88.490709	254.637267	0.080595	22.632781	30.743050	0.315018	24.451939	33.626160	-0.051717	...	10.266350	38.8
1053832	-0.073933	6.935168	20.007424	0.127208	0.084358	0.120209	0.507323	0.028299	0.046203	-0.071542	...	643.107017	665.56
151044	0.085466	570.816079	703.806269	0.351801	43.917052	25.561785	3.284346	157.727464	92.940547	0.027438	...	66.301360	107.5
398266	-0.095121	3.675691	10.250694	-0.063432	0.118890	0.161028	0.010209	0.044727	0.066212	-0.060280	...	342.137331	341.48

5 rows × 29 columns

In [136...

```
# The heatmaps are good for seeing which variables are driving the high scores
data_heatmap = data_all_vars.abs().head(30)
plt.rcParams['figure.figsize'] = (20,10)
ax = sns.heatmap(data_heatmap, center=0, vmin=0, vmax=50, cmap='Reds')
ax.xaxis.tick_top()
ax.xaxis.set_label_position('top')
plt.xticks(rotation=90)
plt.savefig('heatmap.png')
```



In [137...

```
top_records_df = pd.DataFrame(top_records)
```

In [138...

```
# Use this cell if you want to write out the top n record numbers
# top_records_df.to_csv('top_n_record_numbers_baseline.csv', index=False)
```

In [139...

```
# # Use this cell if you want to compare to a previous top n record numbers.
# # You can run a baseline model, see which records score the highest, then change some of the algorithm parameters
# # to see what % of these top scoring records change. The top records are insensitive to changes in the
# # powers for the Minkowski distance measures for the two scores

# top_records_previous = pd.read_csv('top_n_record_numbers_baseline.csv')
# print(top_records_df.head())
# print(top_records_previous.head())
```

```
# num_common = len(pd.merge(top_records_df,top_records_previous, on='RECORD'))
# percent_common = 100*num_common/ntop
# percent_common
```

In [140…

```
print('Duration: ', datetime.now() - start_time)
```

```
Duration:  0:01:08.180996
```