Preparazione dei Dati

Il notebook si occupa di trasformare i dati grezzi messi a disposizioni dal task di Kaggle; il task di preparazione si propone di:

- fornire una prima analisi generale del dataset.
- effettuare una pulizia di dati mancanti, ridondanti o poco significativi.
- aggiungere in forma esplicita nuova informazione utile.
- trasformare i dati affinché possano essere processati in maniera corretta da un algoritmo di Machine-Learning.

Lettura dei dati

In [1]:

```python
# Librerie esterne
import math
import re
import warnings
import pandas as pd
import numpy  as np

from sklearn.preprocessing   import OneHotEncoder
from sklearn.model_selection import train_test_split

from datetime import datetime as dt

warnings.filterwarnings('ignore')
```

Lettura dei dataset forniti da Kaggle.

In [2]:

```python
# local file paths
dir_name = 'datasets'

fp_properties2016 = dir_name + "/properties_2016.csv"
fp_properties2017 = dir_name + "/properties_2017.csv"
fp_train2016      = dir_name +   "/train_2016_v2.csv"
fp_train2017      = dir_name +      "/train_2017.csv"
```

In [3]:

```python
# Lettura dei dataframe
df_properties2016 = pd.read_csv(fp_properties2016, low_memory=False)
df_train2016      = pd.read_csv(fp_train2016,      low_memory=False)
df_properties2017 = pd.read_csv(fp_properties2017, low_memory=False)
df_train2017      = pd.read_csv(fp_train2017,      low_memory=False)
```

In [4]:

```python
# Dimensionalità
print(f'Properites 2016 {df_properties2016.shape}')
print(f'     Train 2016 {     df_train2016.shape}')
print(f'Properites 2017 {df_properties2017.shape}')
print(f'     Train 2017 {     df_train2017.shape}')
```

```
Properites 2016 (2985217, 58)
     Train 2016 (90275, 3)
Properites 2017 (2985217, 58)
     Train 2017 (77613, 3)
```

Non dispongono del log-error di ogni casa, ma solo di quelle che sono state vendute: seleziono solo l'insieme di case di cui ho a disposizione il log-error.

Unione in un unico dataset: matengo le sole case di cui conosco il `log-error`.
Se una casa ha più `log-error`, la colonna è copiata e abbinata a ciascuna data di vendita.

In [5]:

```python
# Right-join
df_2016 = pd.merge(df_properties2016, df_train2016, how='right', left_on=['parcelid'], right_on=['parcelid'])
df_2017 = pd.merge(df_properties2017, df_train2017, how='right', left_on=['parcelid'], right_on=['parcelid'])
```

In [6]:

```python
# Dimensionalità
print(f'Properites 2016 {df_2016.shape}')
print(f'Properites 2017 {df_2017.shape}')
```

```
Properites 2016 (90275, 60)
Properites 2017 (77613, 60)
```

Unisco in un unico dataset i dati del 2016 e del 2017.

In [7]:

```python
# Concat
dfAll = pd.concat([df_2016, df_2017], ignore_index=True)
```

In [8]:

```python
del(df_2016, df_2017)
```

In [9]:

```python
dfAll.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167888 entries, 0 to 167887
Data columns (total 60 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   parcelid                      167888 non-null  int64  
 1   airconditioningtypeid         53788 non-null   float64
 2   architecturalstyletypeid      468 non-null     float64
 3   basementsqft                  93 non-null      float64
 4   bathroomcnt                   167854 non-null  float64
 5   bedroomcnt                    167854 non-null  float64
 6   buildingclasstypeid           31 non-null      float64
 7   buildingqualitytypeid         107173 non-null  float64
 8   calculatedbathnbr             166056 non-null  float64
 9   decktypeid                    1272 non-null    float64
 10  finishedfloor1squarefeet      12893 non-null   float64
 11  calculatedfinishedsquarefeet  166992 non-null  float64
 12  finishedsquarefeet12          159519 non-null  float64
 13  finishedsquarefeet13          75 non-null      float64
 14  finishedsquarefeet15          6591 non-null    float64
 15  finishedsquarefeet50          12893 non-null   float64
 16  finishedsquarefeet6           807 non-null     float64
 17  fips                          167854 non-null  float64
 18  fireplacecnt                  17896 non-null   float64
 19  fullbathcnt                   166056 non-null  float64
 20  garagecarcnt                  55457 non-null   float64
 21  garagetotalsqft               55457 non-null   float64
 22  hashottuborspa                3904 non-null    object 
 23  heatingorsystemtypeid         105651 non-null  float64
 24  latitude                      167854 non-null  float64
 25  longitude                     167854 non-null  float64
 26  lotsizesquarefeet             149446 non-null  float64
 27  poolcnt                       34075 non-null   float64
 28  poolsizesum                   1838 non-null    float64
 29  pooltypeid10                  1626 non-null    float64
 30  pooltypeid2                   2278 non-null    float64
 31  pooltypeid7                   31776 non-null   float64
 32  propertycountylandusecode     167853 non-null  object 
 33  propertylandusetypeid         167854 non-null  float64
 34  propertyzoningdesc            108789 non-null  object 
 35  rawcensustractandblock        167854 non-null  float64
 36  regionidcity                  164579 non-null  float64
 37  regionidcounty                167854 non-null  float64
 38  regionidneighborhood          66986 non-null   float64
 39  regionidzip                   167769 non-null  float64
 40  roomcnt                       167854 non-null  float64
 41  storytypeid                   93 non-null      float64
 42  threequarterbathnbr           22115 non-null   float64
 43  typeconstructiontypeid        522 non-null     float64
 44  unitcnt                       109056 non-null  float64
 45  yardbuildingsqft17            5039 non-null    float64
 46  yardbuildingsqft26            165 non-null     float64
 47  yearbuilt                     166828 non-null  float64
 48  numberofstories               38169 non-null   float64
 49  fireplaceflag                 394 non-null     object 
 50  structuretaxvaluedollarcnt    167359 non-null  float64
 51  taxvaluedollarcnt             167852 non-null  float64
 52  assessmentyear                167854 non-null  float64
 53  landtaxvaluedollarcnt         167851 non-null  float64
 54  taxamount                     167843 non-null  float64
 55  taxdelinquencyflag            4683 non-null    object 
 56  taxdelinquencyyear            4683 non-null    float64
 57  censustractandblock           167002 non-null  float64
 58  logerror                      167888 non-null  float64
 59  transactiondate               167888 non-null  object 
dtypes: float64(53), int64(1), object(6)
memory usage: 76.9+ MB
```

Casting dei tipi

Prima di processare i dati è effettuato un **casting tutti i tipi di dato numerici** da *64 bit* (tipo di default) a *32 bit* con il doppio scopo di ridurre l'impiego la memoria e effettuare calcoli più efficienti.

In [10]:

```python
# Given a dataframe cast all numeric type from 64 bit to 32 bit
def int_float_to32(df):
    for c, dtype in zip(df.columns, df.dtypes):
        if dtype == np.float64:
            df[c] = df[c].astype(np.float32)
        if dtype == np.int64:
            df[c] = df[c].astype(np.int32)
    return df
```

In [11]:

```python
dfAll = int_float_to32(dfAll)
```

In [12]:

```python
dfAll.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 167888 entries, 0 to 167887
Data columns (total 60 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   parcelid                      167888 non-null  int32  
 1   airconditioningtypeid         53788 non-null   float32
 2   architecturalstyletypeid      468 non-null     float32
 3   basementsqft                  93 non-null      float32
 4   bathroomcnt                   167854 non-null  float32
 5   bedroomcnt                    167854 non-null  float32
 6   buildingclasstypeid           31 non-null      float32
 7   buildingqualitytypeid         107173 non-null  float32
 8   calculatedbathnbr             166056 non-null  float32
 9   decktypeid                    1272 non-null    float32
 10  finishedfloor1squarefeet      12893 non-null   float32
 11  calculatedfinishedsquarefeet  166992 non-null  float32
 12  finishedsquarefeet12          159519 non-null  float32
 13  finishedsquarefeet13          75 non-null      float32
 14  finishedsquarefeet15          6591 non-null    float32
 15  finishedsquarefeet50          12893 non-null   float32
 16  finishedsquarefeet6           807 non-null     float32
 17  fips                          167854 non-null  float32
 18  fireplacecnt                  17896 non-null   float32
 19  fullbathcnt                   166056 non-null  float32
 20  garagecarcnt                  55457 non-null   float32
 21  garagetotalsqft               55457 non-null   float32
 22  hashottuborspa                3904 non-null    object 
 23  heatingorsystemtypeid         105651 non-null  float32
 24  latitude                      167854 non-null  float32
 25  longitude                     167854 non-null  float32
 26  lotsizesquarefeet             149446 non-null  float32
 27  poolcnt                       34075 non-null   float32
 28  poolsizesum                   1838 non-null    float32
 29  pooltypeid10                  1626 non-null    float32
 30  pooltypeid2                   2278 non-null    float32
 31  pooltypeid7                   31776 non-null   float32
 32  propertycountylandusecode     167853 non-null  object 
 33  propertylandusetypeid         167854 non-null  float32
 34  propertyzoningdesc            108789 non-null  object 
 35  rawcensustractandblock        167854 non-null  float32
 36  regionidcity                  164579 non-null  float32
 37  regionidcounty                167854 non-null  float32
 38  regionidneighborhood          66986 non-null   float32
 39  regionidzip                   167769 non-null  float32
 40  roomcnt                       167854 non-null  float32
 41  storytypeid                   93 non-null      float32
 42  threequarterbathnbr           22115 non-null   float32
 43  typeconstructiontypeid        522 non-null     float32
 44  unitcnt                       109056 non-null  float32
 45  yardbuildingsqft17            5039 non-null    float32
 46  yardbuildingsqft26            165 non-null     float32
 47  yearbuilt                     166828 non-null  float32
 48  numberofstories               38169 non-null   float32
 49  fireplaceflag                 394 non-null     object 
 50  structuretaxvaluedollarcnt    167359 non-null  float32
 51  taxvaluedollarcnt             167852 non-null  float32
 52  assessmentyear                167854 non-null  float32
 53  landtaxvaluedollarcnt         167851 non-null  float32
 54  taxamount                     167843 non-null  float32
```

```
 55  taxdelinquencyflag            4683 non-null    object
 56  taxdelinquencyyear            4683 non-null    float32
 57  censustractandblock           167002 non-null  float32
 58  logerror                      167888 non-null  float32
 59  transactiondate               167888 non-null  object
dtypes: float32(53), int32(1), object(6)
memory usage: 42.3+ MB
```

Il casting è avvenuto in maniera corretta.

In [13]:

```
dfAll.shape
```

Out[13]:

```
(167888, 60)
```

Prima analisi delle occorezze dei parcelid

Analisi delle occorrenze dei `parcelid` : quante volte una casa è stata venduta tra 2016 e 2017.

In [14]:

```
dfAll.loc[:,'parcelid'].value_counts().head(20)
```

Out[14]:

```
10857130    3
11991059    3
11842707    3
14010551    3
12478591    3
14672826    3
17164212    3
17237150    3
12612211    3
11186156    2
11991474    2
12273962    2
11061551    2
14659784    2
12467034    2
11266654    2
14008322    2
12752161    2
11887100    2
12239653    2
Name: parcelid, dtype: int64
```

Una casa è stata venduta al massimo tre volte: estraggo le case vendute tre volte tra 2016 e 2017.

In [15]:

```
houses = list(dfAll.loc[:,'parcelid'].value_counts()[dfAll.loc[:,'parcelid'].value_counts() == 3].to_dict().key
houses
```

Out[15]:

```
[10857130,
 11991059,
 11842707,
 14010551,
 12478591,
 14672826,
 17164212,
 17237150,
 12612211]
```

Ispeziono `logerror` e `transactiondate` di queste case.

In [16]:

```
dfAll[dfAll.loc[:,'parcelid'].isin(houses)]\
   .sort_values(by=['parcelid', 'transactiondate']).loc[:, ['parcelid', 'logerror', 'transactiondate']]
```

Out[16]:

	parcelid	logerror	transactiondate
135236	10857130	0.053244	2017-06-09
135237	10857130	0.053244	2017-06-30
135238	10857130	0.290908	2017-08-25
55794	11842707	-0.028400	2016-07-14
55795	11842707	0.057300	2016-08-22
55796	11842707	0.207800	2016-09-29
134115	11991059	2.619876	2017-06-06
134116	11991059	2.670239	2017-06-09
134117	11991059	2.508444	2017-06-13
48461	12478591	0.424000	2016-06-23
97365	12478591	0.012482	2017-02-01
97366	12478591	0.039378	2017-09-18
136926	12612211	-0.007561	2017-06-15
136927	12612211	0.074989	2017-08-31
136928	12612211	0.089218	2017-09-18
20217	14010551	1.021000	2016-03-29
114234	14010551	-0.005612	2017-04-06
114235	14010551	0.082468	2017-08-11
33582	14672826	-0.013100	2016-05-10
33583	14672826	0.007000	2016-09-29
155870	14672826	-0.000484	2017-08-11
15385	17164212	-0.080100	2016-03-10
15386	17164212	-0.018200	2016-08-12
165695	17164212	-0.003952	2017-09-11
10416	17237150	0.213500	2016-02-19
10417	17237150	0.288900	2016-07-11
104777	17237150	-0.053883	2017-03-03

Si nota che il `logerror` della stessa casa varia di molto a seconda della data di vendita; nella preprazione dei dati sarà dunque molto importante prendere in considerazione anche il fattore temporale.

Split in Train, Validation e Test

Separazione del dataframe mantenendo in X tutte le colonne fatta eccezione per il `logerror` , che sarà l'unica colonna di y.

In [17]:

```
# Given a dataframe and the column-target name,  returns due dataframes:
#    - X with all colummns except for the target
#    - y with the only target column
def split_X_y(df, yname):
    Xnames = list(dfAll.columns)
    Xnames.remove(yname)
    X = df.loc[:,Xnames]
    y = df.loc[:,yname]
    return X, y
```

In [18]:

```
df_X, df_y = split_X_y(dfAll, 'logerror')
```

Divisione in **Train**, **Validation** e **Test** con proporzioni 6:2:2

In [19]:

```
# Splits the given X and y dataset in three parts:
#    - train      0.6
#    - validation 0.2
#    - test       0.2
def train_validation_test(X, y):
    X_train_80, X_test, y_train_80, y_test = train_test_split(X, y,
                                                       test_size=0.20, random_state=42)
    X_train, X_val, y_train, y_val  = train_test_split(X_train_80, y_train_80,
                                                     test_size=0.25, random_state=42)
    return X_train, y_train, X_val, y_val, X_test, y_test
```

In [20]:

```
X_train, y_train, X_val, y_val, X_test, y_test = train_validation_test(df_X, df_y)
```

In [21]:

```
del(dfAll)
```

Definizione di una funzione che dia informazione sulle **dimensionalità dei dataset**, che ricorrera nel corso delle operazioni per verificare il corretto esito delle trasformazioni impiegate.

In [22]:

```
# Prints shape of X_train, X_val and X_test
# If y flag is on, also prints y shapes
def dimensionality(y=False):
    print(f'X_train {  X_train.shape}')
    print(f'X_val   {    X_val.shape}')
    print(f'X_test  {   X_test.shape}')
    if y:
        print(f'y_train { y_train.shape}')
        print(f'y_val   {   y_val.shape}')
        print(f'y_test  {  y_test.shape}')
```

In [23]:

```
dimensionality(y=True)
```

```
X_train (100732, 59)
X_val   (33578, 59)
X_test  (33578, 59)
y_train (100732,)
y_val   (33578,)
y_test  (33578,)
```

In [24]:

```
X_train.loc[: , ['parcelid', 'transactiondate']].head()
```

Out[24]:

	parcelid	transactiondate
153597	14217523	2017-08-02
146235	11199964	2017-07-11
25650	12627031	2016-04-15
122564	13992985	2017-05-02
84846	12086693	2016-10-13

In [25]:

```
y_train.head()
```

Out[25]:

```
153597    0.057681
146235   -0.010815
25650     0.020800
122564    0.001967
84846    -0.020200
Name: logerror, dtype: float32
```

I numeri di riga sono ora causali: ripristino del numero di riga.

In [26]:

```
# Given a dataframe set its rows in range from 0 to n in ascending order
def arange_rows(df):
    df.index = np.arange(len(df))
    return df
```

In [27]:

```
for df in [X_train, X_val, X_test, y_train, y_val, y_test]:
    df = arange_rows(df)
```

In [28]:

```
X_train.loc[: , ['parcelid', 'transactiondate']].head()
```

Out[28]:

	parcelid	transactiondate
0	14217523	2017-08-02
1	11199964	2017-07-11
2	12627031	2016-04-15
3	13992985	2017-05-02
4	12086693	2016-10-13

In [29]:

```
y_train.head()
```

Out[29]:

```
0    0.057681
1   -0.010815
2    0.020800
3    0.001967
```

```
4   -0.020200
Name: logerror, dtype: float32
```

Rappresentazione non corretta dei Nan

Analizzando il datset di evince che alcune feature rappresentano la assenza di una caratteristica con un Nan, quando in realtà ai fini di algoritmi di **Machine Learning** sarebbe più opportuno rappresentarle con zero o con False, ad esempio:

- `fireplaceflag` ha valori Nan e True, sarebbe opportuna la conversione in una variabile binaria
- `fireplacecnt` e `poolcnt` hanno un valore numerico se l'elmento è presente, Nan se non è presente. Conversione della rappresentazione dell'assenza con 0.

In [30]:

```
# Given a dataframe and a column name, column's values are set to zero if Nan, one otherwise
def set_zero_one(df, col_names):
    for col_name in col_names:
        is_na = df.loc[:,col_name].isna()
        df.loc[:,col_name][is_na]  = 0.
        df.loc[:,col_name][~is_na] = 1.
    return df
```

In [31]:

```
# Given a dataframe and a column name, values of that column are set to zero if Nan
def nan_to_zero(df, col_names):
    for col_name in col_names:
        df.loc[:,col_name].fillna(0., inplace=True)
    return df
```

In [32]:

```
for X in [X_train, X_val, X_test]:
    X = set_zero_one(X, ['fireplaceflag'])
    X = nan_to_zero(X, ['fireplacecnt', 'poolcnt'])
```

Rimozione degli Outlier

Rimuovo dal Train righe che presentano `logerror` estremi: potrebbero costituire dei punti di rumore e inficiare un corretto funzionamento degli algoritmi di Machine-Learning.

In [33]:

```
# Given X and y dataframe remove all rows which target value is under the first or over the last percentile
def remove_outlier(X, y):
    out1 = y < np.percentile(y, 99.5)
    out2 = y > np.percentile(y, 00.5)
    out  = list(map(lambda o1, o2: o1 and o2, out1, out2))
    X = X[out]
    y = y[out]
    return X, y
```

In [34]:

```
dimensionality()
```

```
X_train (100732, 59)
X_val   (33578, 59)
X_test  (33578, 59)
```

In [35]:

```
X_train, y_train = remove_outlier(X_train, y_train)
```

In [36]:

```
dimensionality(y=True)
```

```
X_train (99724, 59)
X_val   (33578, 59)
X_test  (33578, 59)
y_train (99724,)
y_val   (33578,)
y_test  (33578,)
```

Sono state rimosse circa un migliaio di righe dal Train.

Rimozione colonne con alta percentuale di Nan

Rimozione delle colonne con un'alta percentuale di valori assenti: queste arricchiscono l'informazione del dataset in maniera molto limitata.

In [37]:

```
# Given the dataframe and the name of a column returns the column
def get_col(df, colName):
    return df.loc[:, colName]

# Given a column returns Nan-count and Nan-percentage
def get_col_nan_info(col):
    count = col.isna().sum()
    tot = len(col)
    perc = count/tot
    return count, perc

# Given the df and a cut-off returns a list of column names with Nan-percentage greater or equal to the cut-of
def get_cols_over_nan_percentage(df, cutoff):
    names = df.columns
    overPercentage = []
    for name in names:
        col = get_col(df, name)
        _ , perc = get_col_nan_info(col)
        if perc > cutoff:
            overPercentage.append(name)
    return overPercentage
```

In [38]:

```
col_to_delete = get_cols_over_nan_percentage(X_train, 0.6)

for o in col_to_delete:
    print(f'{o} : {get_col_nan_info(get_col(X_train, o))}')
print(f'Length: {len(col_to_delete)}')
```

```
airconditioningtypeid : (67655, 0.6784224459508242)
architecturalstyletypeid : (99455, 0.9973025550519433)
basementsqft : (99676, 0.9995186715334323)
buildingclasstypeid : (99708, 0.9998395571778108)
decktypeid : (99016, 0.992900405118126)
finishedfloor1squarefeet : (91940, 0.9219445670049337)
finishedsquarefeet13 : (99676, 0.9995186715334323)
finishedsquarefeet15 : (95882, 0.9614736673218082)
finishedsquarefeet50 : (91940, 0.9219445670049337)
finishedsquarefeet6 : (99264, 0.9953872688620593)
garagecarcnt : (66611, 0.6679535518029762)
garagetotalsqft : (66611, 0.6679535518029762)
hashottuborspa : (97394, 0.9766355140186916)
poolsizesum : (98619, 0.9889194175925554)
pooltypeid10 : (98711, 0.9898419638201436)
pooltypeid2 : (98407, 0.986793550198548)
pooltypeid7 : (80817, 0.8104067225542497)
regionidneighborhood : (59904, 0.6006979262765232)
storytypeid : (99676, 0.9995186715334323)
threequarterbathnbr : (86493, 0.8673238137258834)
typeconstructiontypeid : (99417, 0.9969215033492439)
yardbuildingsqft17 : (96665, 0.9693253379326943)
yardbuildingsqft26 : (99626, 0.9990172877140909)
numberofstories : (76959, 0.7717199470538687)
taxdelinquencyflag : (96979, 0.9724740283181581)
taxdelinquencyyear : (96979, 0.9724740283181581)
Length: 26
```

Esistono ben 26 righe con una percentuale di valori assenti oltre il 70%, molte delle quali sono superiori al 95%.

In [39]:

```
# Given a dataframe and some column names returns the dataframe within that columns
def remove_column(df, col_names):
    df.drop(col_names, axis=1, inplace=True)
    return df
```

In [40]:

```
col_to_delete
```

Out[40]:

```
['airconditioningtypeid',
 'architecturalstyletypeid',
 'basementsqft',
 'buildingclasstypeid',
 'decktypeid',
 'finishedfloor1squarefeet',
 'finishedsquarefeet13',
 'finishedsquarefeet15',
 'finishedsquarefeet50',
 'finishedsquarefeet6',
 'garagecarcnt',
 'garagetotalsqft',
 'hashottuborspa',
 'poolsizesum',
 'pooltypeid10',
 'pooltypeid2',
 'pooltypeid7',
 'regionidneighborhood',
 'storytypeid',
 'threequarterbathnbr',
 'typeconstructiontypeid',
 'yardbuildingsqft17',
 'yardbuildingsqft26',
 'numberofstories',
 'taxdelinquencyflag',
 'taxdelinquencyyear']
```

In [41]:

```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, col_to_delete)
```

In [42]:

```
dimensionality()
```

```
X_train (99724, 33)
X_val   (33578, 33)
X_test  (33578, 33)
```

Le colonne sono state rimosse correttamente.

Rimozione di Feature ridondanti

Alcune feature portano informazione ripetuta: due colonne diverse contribuiscono con lo stesso tipo di informazione.

fireplaceflag & fireplacecnt

`fireplaceflag` e `fireplacecnt` : la prima dice se esiste almeno un impianto, la seconda quanti impianti sono presenti. La seconda feature porta una informazione almeno uguale a quello della prima.

In [43]:

```
X_train.loc[:,['fireplacecnt', 'fireplaceflag']].head(20)
```

Out[43]:

	fireplacecnt	fireplaceflag
0	0.0	0.0
1	0.0	0.0
2	0.0	0.0
3	0.0	0.0
4	0.0	0.0
5	1.0	0.0
6	1.0	0.0
7	0.0	0.0
8	0.0	0.0
9	0.0	0.0
10	0.0	0.0
11	1.0	0.0
12	0.0	0.0
13	0.0	0.0
14	0.0	0.0
15	0.0	0.0
16	0.0	0.0
17	1.0	0.0
18	0.0	0.0
19	0.0	0.0

Non c'è coerenza tra le due feature.

In [44]:

```
sum((get_col(X_train,['fireplacecnt'])[X_train.loc[:,'fireplaceflag'] == 0] > 0).values.ravel())
```

Out[44]:

```
10773
```

In 10000 osservazioni in cui il flag dice che non ci sono impianti, se ne conta almeno uno.

In [45]:

```
sum((get_col(X_train,['fireplacecnt'])[X_train.loc[:,'fireplaceflag'] == 1] == 0).values.ravel())
```

Out[45]:

```
229
```

In 200 osservazioni in cui il flag dice che è presente un impianto non se ne contano E in egual misura in 200 case dove non si contano impianti il flag ne segnala la presenza.

Scelgo di mantenere l'informazione portata da `fireplacecnt` perché più ricca.

In [46]:

```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, ['fireplaceflag'])
```

In [47]:

```
dimensionality()
```

```
X_train (99724, 32)
X_val   (33578, 32)
X_test  (33578, 32)
```

fullbathcnt & bathroomcnt

Entrambe le feature conteggiano il numero di bagni.

In [48]:

```
X_train.loc[:,['fullbathcnt','bathroomcnt']]
```

Out[48]:

	fullbathcnt	bathroomcnt
0	2.0	2.5
1	3.0	3.0
2	2.0	2.0
3	2.0	2.0
4	1.0	1.0
...	...	...
100727	2.0	2.0
100728	2.0	2.0
100729	2.0	2.5
100730	2.0	2.5
100731	1.0	1.0

99724 rows × 2 columns

`bathroomcnt` porta un informazione decimale, infatti la sua descrizione cita: *including fractional bathrooms*.

In [49]:

```
sum((get_col(X_train, 'bathroomcnt') - get_col(X_train, 'fullbathcnt') > 1).values.ravel())
```

Out[49]:

```
11
```

In solo una decina di istanze il dato non ha lo stessa parte intera.

In [50]:

```
sum((get_col(X_train, 'bathroomcnt') - get_col(X_train, 'fullbathcnt') > 1.5).values.ravel())
```

Out[50]:

```
1
```

E in solo una è maggiore di 1.5.

Scelgo di mantenere solo la colonna bathroomcnt poiché più ricca.

In [51]:

```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, 'fullbathcnt')
```

In [52]:

```
dimensionality()
```

```
X_train (99724, 31)
X_val   (33578, 31)
X_test  (33578, 31)
```

fips & censurtrackblock

`fips` e `censurtackblock` contribuiscono con esattamente la stessa informazione numerica, semplicemente su scala esponenziale differente.

In [53]:

```
get_col(X_train, ['fips', 'censustractandblock'])
```

Out[53]:

	fips	censustractandblock
0	6059.0	6.059022e+13
1	6037.0	6.037910e+13
2	6037.0	6.037294e+13
3	6059.0	6.059087e+13
4	6037.0	6.037302e+13
...	...	...
100727	6037.0	6.037571e+13
100728	6059.0	6.059089e+13
100729	6111.0	6.111007e+13
100730	6059.0	6.059022e+13
100731	6037.0	6.037530e+13

99724 rows × 2 columns

Controllo per quante istanze vale questa relazione.

In [54]:

```
equal = []
for i, j in zip(
        get_col(X_train, ['fips']).values.ravel(),\
        (get_col(X_train, ['censustractandblock']) / 10**10).fillna(0).astype('int32').astype('float32').values
    ):
    equal.append(i==j)

len(equal)
```

Out[54]:

```
99724
```

Sembra non valere per circa 500 osservazioni, analizzo per quali valori non vale.

In [55]:

```
get_col(X_train, ['fips', 'censustractandblock'])[[not e for e in equal]]
```

Out[55]:

	fips	censustractandblock
301	6059.0	NaN
465	6037.0	NaN
727	6037.0	NaN
804	6037.0	NaN
1021	6059.0	NaN
...	...	...
99663	6037.0	NaN
99733	6037.0	NaN
100174	6037.0	NaN
100179	6037.0	NaN
100345	6037.0	NaN

490 rows × 2 columns

In [56]:
```
sum((get_col(X_train, 'censustractandblock')[[not e for e in equal]]).isna())
```
Out[56]:
```
481
```

Per la stragrande maggioranza dei valori in cui la relazione non vale `censustractandblock` ha valore Nan.

Rimuovo la colonna `censustractandblock` che presenza l'assenza di qualche valore a differenza di `fips`.

In [57]:
```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, 'censustractandblock')
```

In [58]:
```
dimensionality()
```
```
X_train (99724, 30)
X_val   (33578, 30)
X_test  (33578, 30)
```

Rimozione di righe con molti Nan

Rimuovo istanze poco significative dal Train

In [59]:
```
# Given the dataframe and the index of the row returns the row
def get_row(df, index):
    return df.loc[index, :]

# Given a row returns Nan-count and Nan-percentage
def get_row_nan_info(row):
    count = row.isna().sum()
    tot = len(row)
    perc = count/tot
    return count, perc

# Given the df and a cut-off returns a list of row ids with Nan-percentage greater or equal the cut-off
def get_rows_over_nan_percentage(df, cutoff):
    overPercentage_indexes = []
    for i in df.index:
        row = get_row(df, i)
        _ , perc = get_row_nan_info(row)
        if perc > cutoff:
            overPercentage_indexes.append(i)
    return overPercentage_indexes
```

In [60]:
```
# Given X and y dataframe and a cut-off removes from both all rows whith a percentage of Nan greater than the c
def drop_fullnan_rows(df, dfy, cutoff):
    indexes = get_rows_over_nan_percentage(df, cutoff)
    return df.drop(indexes, axis=0, inplace=True), dfy.drop(indexes, axis=0, inplace=True)
```

In [61]:
```
dimensionality(y=True)
```
```
X_train (99724, 30)
X_val   (33578, 30)
X_test  (33578, 30)
y_train (99724,)
y_val   (33578,)
y_test  (33578,)
```

Rimuovo le righe con una percentuale di Nan oltre il 50%.

In [62]:
```
for X, y in [[X_train, y_train], [X_val, y_val], [X_test, y_test]]:
    X, y = drop_fullnan_rows(X, y, 0.5)
```

In [63]:
```
dimensionality(y=True)
```
```
X_train (99709, 30)
X_val   (33572, 30)
X_test  (33567, 30)
y_train (99709,)
y_val   (33572,)
y_test  (33567,)
```

Sono state rimosse 15 righe che portavano un'informazione ridotta.

Conversione di valor non numerici

Nel dataset sono presenti dei valori non numerici, di cui è opportuno effettuare una trasformazione numerica ai fini di algoritmi di Machine-Learning.

In [64]:
```
X_train.info()
```
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 99709 entries, 0 to 100731
Data columns (total 30 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   parcelid                      99709 non-null  int32
 1   bathroomcnt                   99709 non-null  float32
 2   bedroomcnt                    99709 non-null  float32
 3   buildingqualitytypeid         63558 non-null  float32
 4   calculatedbathnbr             98697 non-null  float32
 5   calculatedfinishedsquarefeet  99224 non-null  float32
 6   finishedsquarefeet12          94874 non-null  float32
 7   fips                          99709 non-null  float32
 8   fireplacecnt                  99709 non-null  float32
 9   heatingorsystemtypeid         62661 non-null  float32
 10  latitude                      99709 non-null  float32
 11  longitude                     99709 non-null  float32
 12  lotsizesquarefeet             88847 non-null  float32
 13  poolcnt                       99709 non-null  float32
 14  propertycountylandusecode     99709 non-null  object
 15  propertylandusetypeid         99709 non-null  float32
 16  propertyzoningdesc            64482 non-null  object
 17  rawcensustractandblock        99709 non-null  float32
 18  regionidcity                  97773 non-null  float32
 19  regionidcounty                99709 non-null  float32
 20  regionidzip                   99661 non-null  float32
 21  roomcnt                       99709 non-null  float32
 22  unitcnt                       64641 non-null  float32
 23  yearbuilt                     99138 non-null  float32
 24  structuretaxvaluedollarcnt    99436 non-null  float32
 25  taxvaluedollarcnt             99708 non-null  float32
 26  assessmentyear                99709 non-null  float32
 27  landtaxvaluedollarcnt         99708 non-null  float32
 28  taxamount                     99699 non-null  float32
 29  transactiondate               99709 non-null  object
dtypes: float32(26), int32(1), object(3)
memory usage: 13.3+ MB
```

In [65]:
```
# Given a dataframe returns al list of column-names which type is different from int32 and float32
def get_not_numeric_cols(df):

    def is_numeric(value):
        return value != np.int32 and\
               value != np.float32

    not_numeric = []
    for k, v in dict(df.dtypes).items():
        if(is_numeric(v)):
            not_numeric.append(k)
    return not_numeric
```

Feature non numeriche:

In [66]:
```
not_numeric = get_not_numeric_cols(X_train)
print(*not_numeric, sep='\n')
```
```
propertycountylandusecode
propertyzoningdesc
transactiondate
```

Ho tre valori non numerici.

propertycountylandusecode & propertyzoningdesc

In [67]:
```
# propertycountylandusecode
values = get_col(X_train, 'propertycountylandusecode')
print(f'Values:\n{values.unique()      }\n')
print(f'Occurences:\n{values.value_counts()}')
```
```
Values:
['122' '0100' '010E' '1111' '34' '010C' '1129' '0101' '1' '010D' '0300'
 '1128' '012C' '01DC' '0200' '1117' '010H' '010F' '0104' '1110' '0108'
 '010V' '0400' '020G' '0103' '010G' '96' '0201' '1210' '01HC' '135' '38'
 '070D' '1222' '010M' '0700' '1014' '0131' '1410' '0109' '1116' '012D'
 '100V' '1720' '1112' '73' '0102' '01HE' '1722' '0110' '0111' '0113'
 '012E' '105' '1420' '040V' '1310' '0301' '6050' '030G' '0114' '0401'
 '0130' '0115' '8800' '1432' '020E' '1012' '1321' '0303' '0105' '1011'
 '1120' '1421' '0' '040B' '1333' '0141' '0204']

Occurences:
0100    34060
122     16979
010C    11428
0101     8196
34       6396
        ...
0           1
0303        1
1012        1
0115        1
1420        1
Name: propertycountylandusecode, Length: 79, dtype: int64
```

In [68]:
```
# propertyzoningdesc
values = get_col(X_train, 'propertyzoningdesc')
print(f'Values:\n{values.unique()      }\n')
print(f'Occurences:\n{values.value_counts()}')
```
```
Values:
[nan 'PDR1*' 'LAR1' ... 'BRR1*' 'LVPID*' 'LC105*']

Occurences:
LAR1           8508
LAR3           3131
LARS           1720
LBR1N          1600
LARD1.5        1442
               ...
COML-PRH*         1
WHR112000*        1
LRA11*            1
LCC4-R1600        1
MBR3*             1
Name: propertyzoningdesc, Length: 2052, dtype: int64
```

`propertycountylandusecode` conta ben 80 valori distinti; `propertyzoningdesc` ne conta oltre 2000.

Per queste due feature si sceglie di mantenere il valore solamente delle cinque più frequenti; tutte la altre sono settate con il valore *rare*.

Questo consente di coprire l'informazione delle maggioranza della popalazione, senza però far esplodere il numero di colonne successivamente al **One-Hot Econding**.

In [69]:
```
# Given a dataframe, a column-name and a importcance treshold, returns a list of his most frequent values
def get_frequent(df, col_name, important):
    col = get_col(df, col_name)
    names = list(col.value_counts().to_dict().keys())[:important]
    return names

# Given the train-dataframe, some column-names, and importance treshold and a list of dataframes,
# foreach dataframe set a value of the specified columns to 'rare' if is not frequent.
def set_rare(df, col_name, important, dfs):
    frequent = get_frequent(df, col_name, important)
    for d in dfs:
        d.loc[:,col_name][~d.loc[:,col_name].isin(frequent)] = 'rare'
```

In [70]:
```
for col in ['propertycountylandusecode', 'propertyzoningdesc']:
    set_rare(X_train, col, 5, [X_train, X_val, X_test])
```

Verifica se *rare* è stato assegnato correttamente.

In [71]:
```
values = get_col(X_train, 'propertycountylandusecode')
print(f'Values:\n{values.unique()      }\n')
print(f'Occurences:\n{values.value_counts()}')
```
```
Values:
['122' '0100' 'rare' '34' '010C' '0101']

Occurences:
0100    34060
rare    22650
122     16979
010C    11428
0101     8196
34       6396
Name: propertycountylandusecode, dtype: int64
```

L'operazione avvenuta correttamentamente.

transactiondate

In [72]:
```
# transactiondate
values = get_col(X_train, 'transactiondate')
print(f'Values:\n{values.unique()}\n')
print(f'Occurences:\n{values.value_counts().head(10)}')
```
```
Values:
['2017-08-02' '2017-07-11' '2016-04-15' '2017-05-02' '2016-10-13'
 '2016-06-13' '2017-01-03' '2016-08-15' '2016-10-20' '2016-08-11'
 '2016-05-27' '2017-08-10' '2017-06-09' '2017-05-16' '2016-08-03'
 '2017-01-30' '2017-05-12' '2017-06-28' '2016-07-28' '2016-09-02'
 '2016-03-11' '2017-06-06' '2017-09-08' '2017-09-01' '2016-07-13'
 '2017-07-27' '2017-04-05' '2016-12-07' '2017-05-05' '2016-06-30'
 '2017-01-27' '2016-07-21' '2016-10-07' '2017-07-31' '2017-07-24'
 '2016-05-24' '2016-04-22' '2017-06-29' '2016-04-13' '2017-03-16'
 '2017-06-26' '2016-08-25' '2017-07-14' '2016-09-15' '2017-07-13'
 '2016-03-18' '2016-07-07' '2017-01-05' '2017-01-25' '2017-07-28'
 '2017-02-28' '2017-08-04' '2016-08-29' '2017-04-12' '2016-06-23'
 '2017-09-06' '2016-01-14' '2017-06-02' '2016-08-31' '2016-06-24'
 '2016-05-30' '2017-02-24' '2017-06-08' '2017-06-16' '2017-09-14'
 '2016-02-28' '2017-04-20' '2016-09-20' '2016-04-21' '2017-03-31'
 '2017-03-08' '2017-06-01' '2016-02-08' '2016-03-02' '2017-05-24'
 '2016-06-03' '2016-09-21' '2016-04-01' '2016-06-29' '2016-08-10'
 '2016-07-05' '2017-01-26' '2017-08-22' '2016-01-13' '2016-08-08'
 '2016-10-12' '2016-05-10' '2016-07-15' '2017-08-31' '2016-12-05'
 '2016-04-25' '2016-11-13' '2017-03-21' '2017-03-07' '2016-08-12'
 '2017-08-29' '2016-03-17' '2016-09-13' '2016-06-21' '2016-01-27'
 '2016-01-28' '2016-01-21' '2017-05-26' '2017-02-21' '2017-09-19'
 '2017-01-31' '2016-09-26' '2016-10-14' '2017-04-25' '2016-05-18'
 '2017-03-27' '2016-08-26' '2017-02-16' '2016-10-03' '2017-08-07'
 '2016-05-02' '2017-03-09' '2017-02-23' '2016-12-14' '2016-10-19'
 '2016-02-07' '2016-11-29' '2016-09-16' '2017-03-30' '2017-03-15'
 '2017-05-10' '2016-05-31' '2016-07-20' '2016-01-05' '2017-02-14'
 '2016-01-29' '2017-07-12' '2017-07-07' '2016-06-15' '2017-06-07'
 '2016-10-06' '2016-07-29' '2016-11-01' '2017-02-06' '2016-05-25'
 '2017-08-25' '2017-02-17' '2016-02-26' '2016-05-19' '2016-05-09'
 '2017-04-18' '2017-03-03' '2017-05-01' '2016-08-30' '2016-05-20'
 '2017-06-19' '2017-08-17' '2017-09-11' '2016-11-18' '2017-05-11'
 '2016-07-27' '2016-01-19' '2017-05-31' '2016-02-11' '2017-01-13'
 '2017-04-28' '2016-12-29' '2016-06-22' '2017-05-23' '2016-09-29'
 '2016-06-14' '2017-06-30' '2016-01-11' '2017-06-27' '2016-04-08'
 '2016-03-29' '2017-06-13' '2016-08-24' '2016-04-20' '2016-03-01'
 '2017-05-04' '2016-06-17' '2017-01-09' '2016-10-05' '2017-04-14'
 '2016-07-01' '2016-04-12' '2016-09-06' '2016-06-10' '2017-08-21'
 '2017-05-22' '2017-07-18' '2017-08-08' '2016-03-04' '2016-06-01'
 '2016-11-06' '2016-05-12' '2016-04-29' '2017-05-03' '2016-09-30'
 '2016-09-23' '2016-06-16' '2017-06-05' '2017-06-22' '2016-08-16'
 '2017-06-23' '2016-01-22' '2016-08-09' '2017-05-19' '2016-08-19'
 '2016-12-15' '2016-03-10' '2017-04-06' '2016-01-26' '2017-04-27'
 '2016-10-24' '2016-03-30' '2017-01-04' '2017-07-19' '2017-07-03'
 '2016-01-06' '2017-02-08' '2017-07-17' '2016-09-09' '2016-08-17'
 '2017-01-17' '2016-08-18' '2016-11-21' '2017-05-17' '2016-03-28'
 '2017-03-10' '2017-02-27' '2016-06-27' '2017-05-09' '2017-09-12'
 '2017-09-13' '2017-03-02' '2016-11-03' '2016-03-23' '2016-08-22'
 '2016-04-26' '2016-03-16' '2016-12-28' '2016-09-27' '2017-07-26'
 '2016-04-07' '2017-04-19' '2017-08-18' '2017-05-18' '2017-08-11'
 '2017-07-25' '2016-07-18' '2016-02-10' '2016-05-11' '2017-07-06'
 '2017-01-12' '2017-08-30' '2016-02-17' '2017-09-07' '2016-06-02'
 '2017-06-14' '2016-01-02' '2016-03-31' '2017-02-22' '2017-02-10'
 '2016-11-14' '2016-12-23' '2016-07-22' '2016-07-26' '2017-06-20'
 '2017-06-21' '2016-07-12' '2016-06-06' '2017-04-03' '2016-07-19'
 '2017-08-24' '2017-01-24' '2016-03-24' '2016-09-24' '2016-09-07'
 '2016-08-05' '2016-06-20' '2017-03-18' '2017-04-10' '2016-04-27'
 '2017-04-26' '2016-08-23' '2017-04-04' '2017-08-09' '2016-06-28'
 '2016-11-30' '2016-01-31' '2017-03-24' '2017-08-01' '2017-05-08'
 '2017-02-26' '2016-06-08' '2017-01-10' '2016-05-26' '2016-05-05'
 '2016-08-01' '2016-03-06' '2016-09-19' '2017-09-15' '2016-01-12'
 '2017-07-10' '2016-04-06' '2017-02-09' '2016-06-09' '2016-01-25'
 '2016-02-25' '2016-04-18' '2017-01-16' '2016-09-22' '2016-05-03'
 '2017-04-17' '2016-10-04' '2016-07-14' '2017-03-29' '2016-02-02'
 '2016-11-28' '2016-04-14' '2016-02-18' '2016-03-15' '2017-04-13'
 '2016-12-27' '2016-02-01' '2016-06-07' '2017-02-05' '2017-07-06'
 '2016-03-22' '2017-03-23' '2016-11-15' '2016-09-14' '2017-09-18'
 '2016-10-25' '2016-12-30' '2016-12-02' '2017-01-11' '2016-10-27'
 '2017-08-15' '2016-05-16' '2017-08-23' '2016-01-08' '2016-03-21'
 '2017-03-28' '2017-01-23' '2017-01-19' '2016-02-24' '2016-02-29'
 '2017-08-14' '2016-02-23' '2016-04-19' '2016-02-05' '2016-05-13'
 '2016-02-22' '2016-12-01' '2017-02-13' '2016-02-03' '2016-12-16'
 '2017-06-12' '2016-01-04' '2016-09-28' '2016-03-08' '2016-02-12'
 '2017-02-07' '2017-04-11' '2017-02-01' '2016-12-08' '2016-07-08'
 '2017-04-21' '2016-04-28' '2016-11-09' '2016-07-11' '2016-11-22'
 '2017-07-05' '2017-07-21' '2017-01-29' '2017-01-06' '2016-01-07'
 '2016-01-15' '2017-04-07' '2016-04-16' '2017-01-20' '2017-02-15'
 '2016-09-01' '2017-03-17' '2016-10-17' '2016-03-25' '2017-08-03'
 '2016-02-16' '2016-07-25' '2017-01-18' '2017-05-25' '2016-05-04'
 '2016-05-08' '2017-08-28' '2016-02-09' '2016-02-19' '2016-10-11'
 '2016-03-03' '2017-01-15' '2017-05-15' '2016-05-06' '2016-09-12'
 '2016-09-08' '2016-02-04' '2016-01-20' '2017-03-06' '2017-04-24'
 '2017-07-20' '2017-03-14' '2016-11-23' '2016-05-23' '2017-02-03'
 '2017-03-01' '2016-08-04' '2017-06-15' '2017-07-30' '2016-02-15'
 '2016-03-09' '2016-01-24' '2017-03-14' '2016-12-19' '2016-04-11'
 '2016-11-08' '2016-10-31' '2016-11-27' '2016-11-04' '2017-03-20'
 '2016-04-05' '2016-08-02' '2017-03-13' '2016-12-04' '2017-08-16'
 '2017-09-05' '2017-02-02' '2016-12-09' '2017-08-12' '2016-01-10'
 '2016-11-16' '2016-04-04' '2016-11-07' '2017-02-20' '2016-05-17'
 '2016-09-25' '2016-12-12' '2016-10-28' '2016-01-17' '2017-01-01'
 '2016-12-20' '2016-11-20' '2016-07-16' '2016-12-18' '2016-03-07'
 '2016-12-22' '2016-11-02' '2016-10-21' '2017-07-02' '2016-12-06'
 '2017-05-29' '2017-03-22' '2016-11-10' '2017-07-29' '2016-12-13'
 '2016-06-05' '2016-12-21' '2016-02-21' '2017-01-02' '2016-04-30'
 '2017-07-08' '2017-01-22' '2016-12-24' '2016-10-18' '2017-03-05'
 '2016-01-03' '2016-02-06' '2017-01-14' '2017-01-08' '2016-01-18'
 '2016-10-02' '2016-10-26' '2017-07-01' '2016-06-12' '2016-11-17'
 '2016-07-24' '2016-04-03' '2016-04-10' '2017-01-07' '2016-01-23'
 '2016-07-02' '2016-06-25' '2017-01-28' '2017-09-17' '2017-03-19'
 '2017-02-19' '2016-08-28' '2016-09-18' '2016-08-21' '2017-08-20'
 '2016-02-27' '2017-03-04' '2016-10-10' '2017-04-22' '2016-11-24'
 '2016-03-05' '2016-07-10' '2016-12-26' '2016-08-13' '2016-04-24'
 '2017-02-12' '2016-04-09' '2016-09-10' '2016-08-14' '2017-09-04'
 '2017-07-15' '2016-02-20' '2017-05-14' '2016-02-14' '2016-10-08'
 '2016-06-25' '2016-10-22' '2016-10-09' '2016-05-15' '2017-04-01'
 '2016-07-04' '2016-07-31' '2017-05-13' '2017-04-15' '2016-07-17'
 '2016-09-17' '2016-03-19' '2017-04-29' '2016-08-07' '2017-03-11'
 '2016-06-26' '2017-02-25' '2016-10-29' '2017-09-20' '2017-02-11'
 '2016-12-11' '2016-05-30' '2017-06-04' '2017-08-19' '2017-04-30'
 '2016-06-18' '2017-08-27' '2017-05-21' '2016-11-12' '2016-05-29'
 '2017-08-26' '2016-06-04' '2016-07-09' '2016-01-09' '2017-01-21'
 '2016-10-01' '2017-06-17' '2016-06-11' '2016-04-02' '2016-12-25'
 '2016-02-13' '2017-07-04' '2016-06-19' '2017-05-28' '2016-05-01'
 '2017-09-10' '2017-04-16' '2017-03-26' '2017-04-23' '2016-05-14'
 '2016-01-01' '2016-08-27' '2016-03-20' '2017-06-18' '2017-02-04'
 '2016-07-23' '2016-01-30' '2017-04-09' '2017-06-24' '2016-04-17'
 '2017-09-02' '2016-09-04' '2017-09-09' '2016-03-27' '2016-09-05'
 '2016-07-30' '2016-03-13' '2017-07-16' '2016-10-30' '2016-01-16'
 '2017-06-03' '2016-04-23' '2017-09-16' '2016-09-11' '2017-03-12'
 '2016-09-03' '2016-08-20' '2017-05-27' '2016-05-22' '2016-05-07'
 '2017-05-07' '2017-05-20' '2017-07-23' '2017-08-06' '2016-11-19'
 '2016-08-06' '2017-04-02' '2017-06-10' '2017-06-11' '2017-05-06']

Occurences:
2017-06-30    689
2016-07-29    561
2017-04-28    543
2016-05-27    524
2016-09-30    523
2017-03-31    520
2016-04-29    508
2016-06-30    504
2017-07-28    478
2017-08-31    472
Name: transactiondate, dtype: int64
```

È interessante notare come la maggior parte delle transizioni si concentrino negli ultimi giorni del mese.

Trasformazione dei dati come giorni passati dal 1 Gennaio di quell'anno usando dunque un'informazione di tipo intero.

In [73]:
```
# Add a row to the given dataframe which value is the days passed from the 1th Genuary of that year
def date_to_int(df):

    def string_to_date(date_str):
        return dt.strptime(date_str.replace('-', '/'), '%Y/%m/%d')

    start = string_to_date('2016-01-01')

    df.loc[:,'int_transactiondate'] = pd.to_datetime(df.loc[:,'transactiondate'], format='%Y/%m/%d')
    df.loc[:,'int_transactiondate'] = (df.loc[:,'int_transactiondate'] - start).astype('timedelta64[D]') % 365

    return df
```

In [74]:
```
for X in [X_train, X_val, X_test]:
    X = date_to_int(X)
```

Verifico se l'operazione è avvenuta correttamente:

In [75]:
```
X_train.loc[:,['parcelid', 'transactiondate', 'int_transactiondate']].head(20)
```

Out[75]:

	parcelid	transactiondate	int_transactiondate
0	14217523	2017-08-02	214.0
1	11199964	2017-07-11	192.0
2	12627031	2016-04-15	105.0
3	13992985	2017-05-02	122.0
4	12086693	2016-10-13	286.0
5	14152642	2016-06-13	164.0
6	17210760	2017-01-03	3.0
7	14145516	2016-08-15	227.0
8	14696645	2016-10-20	293.0
9	11697737	2016-08-11	223.0
10	14071157	2016-05-27	147.0
11	14209240	2017-08-10	222.0
12	12375928	2017-06-09	160.0
13	12480158	2017-05-16	136.0
14	10988496	2016-08-03	215.0
15	10982015	2017-01-30	30.0
16	11228135	2017-05-12	132.0
17	17289071	2017-06-28	179.0
18	11093150	2016-07-28	209.0
19	12831060	2016-09-02	245.0

L'operazione sembra essere avvenuta correttamente.

Aggiunta di nuova informazione

[illegible]

Media delle transazioni di quel periodo in quella regione

[illegible]

In [76]: [illegible]

In [77]: [illegible]

In [78]: [illegible]

Out[78]:

	parcelid	regionidcounty	transactiondate	int_transactiondate	period_mean_price
0	14217523	[illegible]	2017-08-02	214.0	[illegible]
1	11199964	[illegible]	2017-07-11	192.0	[illegible]
2	12627031	[illegible]	2016-04-15	105.0	[illegible]
3	13992985	[illegible]	2017-05-02	122.0	[illegible]
4	12086693	3101.0	2016-10-13	286.0	0.015765
5	14152642	1286.0	2016-06-13	164.0	0.008718
6	17210760	2061.0	2017-01-03	3.0	0.020255
7	14145516	1286.0	2016-08-15	227.0	0.012602
8	14696645	1286.0	2016-10-20	293.0	0.022827
9	11697737	3101.0	2016-08-11	223.0	0.008038
10	14071157	1286.0	2016-05-27	147.0	0.008571
11	14209240	1286.0	2017-08-10	222.0	0.018485
12	12375928	3101.0	2017-06-09	160.0	0.009990
13	12480158	3101.0	2017-05-16	136.0	0.005966
14	10988496	3101.0	2016-08-03	215.0	0.008038
15	10982015	3101.0	2017-01-30	30.0	0.018386
16	11228135	3101.0	2017-05-12	132.0	0.005966
17	17289071	2061.0	2017-06-28	179.0	0.010030
18	11093150	3101.0	2016-07-28	209.0	0.011585
19	12831060	3101.0	2016-09-02	245.0	0.016233

L'informazione `transactiondate` è stata converita in intero in `int_transactiondate` e ne è stata ricavata un'ulteriore informazione in `period_mean_price`. Ora è possibile rimuovere la colonna.

In [79]:
```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, ['transactiondate'])
```

Media delle transazioni delle case vicine

Aggiungo l'informazione del `logerror` medio in un area circoscritta e discretizzata.

La longitudine ha sempre valori negativi, ne effettuo una trasformazione in valore assoluto per semplificare i conti pur mantendendo la stessa informazione.

In [80]:
```
# Given a dataframe set its longitude to positive
def abs_longitude(df):
    df.loc[:,'longitude'] = abs(df.loc[:,'longitude'])
    return df
```

In [81]:
```
for X in [X_train, X_val, X_test]:
     X = abs_longitude(X)
```

Per rendere efficiente il calcolo, lo spazio è discretizzato ed è costruito un dizionario con coppia **(longitudine, latitudine)** discretizzate.

In [82]:
```
# Given a dataframe, longitude, latitude and distance returns all rows which distance is less or equal to the g
def get_neighborhood(df, lon, lat, distance):

    cond1 = lon - distance <= df.loc[:,'longitude']
    cond2 = df.loc[:,'longitude'] <= lon + distance
    cond3 = lat - distance <= df.loc[:,'latitude']
    cond4 = df.loc[:,'latitude'] <= lat + distance

    cond =  list(map(lambda c1, c2, c3, c4: c1 and c2 and c3 and c4, cond1, cond2, cond3, cond4))

    return df[cond]

# Given dataframe and its target, latitude, longitude and distance returns the mean logerror
# of rows which distance is less or equal to the given distance
def get_neighborhood_mean(df, dfy, lat, lon, distance):
    ret = get_neighborhood(df, lat, lon, distance)
    indexes = ret.index
    return dfy[indexes].mean()
```

In [83]:
```
# Floor input value in a multiple of the given distance
def round_lat_lon(dim, dist):
        return dim - (dim % dist)
```

In [84]:
```
# Given longitude, latitude and distance generate the corresponding key flooring two values
def generate_key(lon, lat, dist):
    lon = round_lat_lon(lon, dist)
    lat = round_lat_lon(lat, dist)
    key = f'{lon}_{lat}'
    return key
```

In [85]:
```
# Generate a dictionary with the mean log-error of the neighborhood, defined by the given distance.
def create_distance_dict(df, dfy, dist):

    lon_start = round_lat_lon(df.loc[:,'longitude'].min(), dist)
    lon_end   = round_lat_lon(df.loc[:,'longitude'].max(), dist)
    lat_start = round_lat_lon(df.loc[:,'latitude'].min(), dist)
    lat_end   = round_lat_lon(df.loc[:,'latitude' ].max(), dist)

    dict_dist = {}

    lon = lon_start
    lat = lat_start

    while(lon < lon_end):
        lat = lat_start
        while(lat < lat_end):
            key = generate_key(lon, lat, dist)
            dict_dist[key] = get_neighborhood_mean(df, dfy, lon, lat, dist)
            lat += dist
        lon += dist

    return dict_dist
```

In [86]:
```
def add_neighborhood_logerror_column_aux(df, dist, dict_dist):

    df['neighborhood_mean_price'] = np.nan

    for i in df.index:
        lon = df.at[i,'longitude']
        lat = df.at[i,'latitude']
        key = generate_key(lon, lat, dist)
        try:
            df.at[i, 'neighborhood_mean_price'] = dict_dist[key]
        except:
            print(f'Chiave {key} non esiste')
            pass

    return df

# Given a list of dataframes, train dataset and its target and a distance
# add a column with the mean-price of the neighborhood defined from the distance given
def add_neighborhood_logerror_column(df_list, dfx, dfy, distance):

    dict_dist = create_distance_dict(
        dfx,
        dfy,
        distance
    )

    for df in df_list:
        df = add_neighborhood_logerror_column_aux(df, distance, dict_dist)
```

In [87]:
```
DIST = 30000
```

In [88]:
```
add_neighborhood_logerror_column([X_train, X_val, X_test], X_train, y_train, DIST)
```
```
Chiave 118260000.0_34800000.0 non esiste
Chiave 118230000.0_34800000.0 non esiste
Chiave 119430000.0_34380000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 118230000.0_34800000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 118260000.0_34800000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119230000.0_34800000.0 non esiste
Chiave 118320000.0_34800000.0 non esiste
Chiave 118260000.0_34800000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
Chiave 119460000.0_34350000.0 non esiste
Chiave 118410000.0_34800000.0 non esiste
Chiave 119430000.0_34350000.0 non esiste
```

Il valori discreti sono stati costruiti sul range del Train, che a quanto pare non è stato in grado di coprire alcuni valori presenti nel Train e nel Test, che avranno valore Nan.

In [89]:
```
for X in [X_train, X_val, X_test]:
    print(sum(X.loc[:,'neighborhood_mean_price'].isna()))
```
```
0
12
24
```

Il fenomeno è accettabile poiché questo è un rischio che può incorrere nel processare dati "nuovi", non previsti dall'insieme di Train.

In [90]:
```
X_train.loc[:,['parcelid', 'longitude', 'latitude', 'neighborhood_mean_price']].head(20)
```

Out[90]:

	parcelid	longitude	latitude	neighborhood_mean_price
0	14217523	117760080.0	33834352.0	0.017423
1	11199964	118103008.0	34263684.0	-0.002890
2	12627031	118277552.0	33791928.0	0.017852
3	13992985	117946520.0	33829112.0	0.006387
4	12086693	118232000.0	34139900.0	-0.002875
5	14152642	117926168.0	33936776.0	0.003294
6	17210760	118869328.0	34224248.0	0.018469
7	14145516	117940552.0	33907560.0	0.007479
8	14696645	117833896.0	33608692.0	0.021820
9	11697737	118341384.0	33992188.0	-0.006688
10	14071157	117991024.0	33714372.0	0.009464
11	14209240	117825080.0	33848864.0	0.019038
12	12375928	118115272.0	34014304.0	0.015102
13	12480158	118108120.0	33865344.0	0.008038
14	10988496	118371936.0	34250192.0	0.007338
15	10982015	118379000.0	34275600.0	0.011258
16	11228135	117970096.0	34533360.0	0.020102
17	17289071	118805160.0	34159224.0	0.007006
18	11093150	118514960.0	34241808.0	0.014025
19	12831060	117964736.0	33948352.0	0.012256

Colonne aggiuntive calcolate su rapporti

Aggiunta di tre colonne, rapporto di altre due preesistenti:

`living_area_prop`, rapporto tra:

- **`calculatedfinishedsquarefeet`**: *Calculated total finished living area of the home.*
- **`lotsizesquarefeet`**: *Area of the lot in square feet.*

Rappresenta dunque la proporzione di metri quadri calpestabile quindi abitabili.

`tax_ratio`, rapporto tra:

- **`taxvaluedollarcnt`**: *The total tax assessed value of the parcel.*
- **`taxamount`**: *The total property tax assessed for that assessment year.*

Rappresenta la proporzione di tasse totali pagate per quell'anno, utile per discriminare in base all'anno di vendita.

`tax_prop`, rapporto tra:

- **`structuretaxvaluedollarcnt`**: *The assessed value of the built structure on the parcel.*
- **`landtaxvaluedollarcnt`**: *The assessed value of the land area of the parcel.*

Rappresenta dunque l'interazione tra il costo relativo alla struttura in sè e al terreno su cui è stata costruita.

In [91]:
```
# Add three columns to the given dataset
def add_tax_info(df):
    df['living_area_prop'] = df['calculatedfinishedsquarefeet'] / df['lotsizesquarefeet']
    df['tax_ratio']        = df['taxvaluedollarcnt']            / df['taxamount']
    df['tax_prop']         = df['structuretaxvaluedollarcnt']   / df['landtaxvaluedollarcnt']
    return df
```

In [92]:
```
for df in [X_train, X_val, X_test]:
    df = add_tax_info(df)
```

Valori discreti

Analisi di valori discreti usando un'euristica: un dato è considerato discreto se ha meno di 30 valori distinti.

In [93]:
```
# Given a dataframe and a cut-off returns a list of column-names which has less than cut-off different values
def get_discrete(df, cutoff):
    discretes = []
    for col_name in df.columns:
        values_count = len(get_col(df,col_name).unique())
        if values_count < cutoff:
            discretes.append(col_name)
    return discretes

# Given a list of columns prints for each column all his differente values
def discrete_info(df, discretes):
    for discrete in discretes:
        values = get_col(df, discrete).unique()
        print(f'{discrete}\n{values} ({len(values)})\n')
```

Valori discreti:

In [94]:
```
discrete_info(X_train, get_discrete(X_train, 30))
```

```
bathroomcnt
[ 2.5  3.   2.   1.   4.   1.5  5.   3.5  0.   4.5  6.   6.5  5.5  8.
 10.   7.5  7.  15.   9.  11.  18.  13.  12.   8.5] (24)

bedroomcnt
[ 3.  4.  5.  0.  2.  1.  6.  8.  7.  9. 10. 12. 11. 16. 14. 13.] (16)

buildingqualitytypeid
[nan  8.  7.  4.  6. 10.  9.  1.  5. 11. 12.  3.  2.] (13)

calculatedbathnbr
[ 2.5  3.   2.   1.   4.   1.5  5.   3.5  nan  4.5  6.   6.5  5.5  8.
 10.   7.5  7.  15.   9.  11.  18.  13.  12.   8.5] (24)

fips
[6059. 6037. 6111.] (3)

fireplacecnt
[0. 1. 2. 3. 4. 5.] (6)

heatingorsystemtypeid
[nan  2.  7. 24.  6. 13. 20. 18.  1. 11. 10. 12.] (12)

poolcnt
[0. 1.] (2)

propertycountylandusecode
['122' '0100' 'rare' '34' '010C' '0101'] (6)

propertylandusetypeid
[261. 266. 269. 247. 265. 246. 275. 267. 260. 248. 263.  31. 264.] (13)

propertyzoningdesc
['rare' 'LAR1' 'LAR3' 'LARD1.5' 'LBR1N' 'LARS'] (6)

regionidcounty
[1286. 3101. 2061.] (3)

roomcnt
[ 7.  0.  8.  6.  5.  4.  9. 10.  3.  2. 11. 15. 12. 13. 14.  1.] (16)

unitcnt
[ nan   1.   3.   2.   4. 143.   9.  70.   6. 237.  11.] (11)

assessmentyear
[2016. 2015.] (2)
```

Analisi dei tipi di valore e divisione in **categoriali** e **ordinali**; divido questi due tipi poiché andranno processati in maniera differente.
Per i primi sarò fatto **One-Hot Econding**, per i secondi invece no poiché comporterebbe una perdita di informazione.

In [95]:

```python
# Valori discreti: cateogirci e ordinali
categorical = ['assessmentyear',
               'fips',
               'heatingorsystemtypeid',
               'poolcnt',
               'propertycountylandusecode',
               'propertylandusetypeid',
               'propertyzoningdesc',
               'regionidcounty'
              ]
ordinal = ['bathroomcnt',
           'bedroomcnt',
           'buildingqualitytypeid',
           'calculatedbathnbr',
           'fireplacecnt',
           'roomcnt',
           'unitcnt'
          ]
```

Fatta eccezione per `parcelid`, il resto dei dati sono **numerici**.

In [96]:

```python
# Valori continui: numerici
numeric = list(set(X_train.columns) - set(categorical + ordinal) - {'parcelid'})
```

In [97]:

```python
X_train[numeric].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 99709 entries, 0 to 100731
Data columns (total 19 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   living_area_prop              88583 non-null  float32
 1   structuretaxvaluedollarcnt    99436 non-null  float32
 2   taxamount                     99699 non-null  float32
 3   finishedsquarefeet12          94874 non-null  float32
 4   period_mean_price             99709 non-null  float64
 5   calculatedfinishedsquarefeet  99224 non-null  float32
 6   longitude                     99709 non-null  float32
 7   tax_prop                      99436 non-null  float32
 8   tax_ratio                     99698 non-null  float32
 9   regionidzip                   99661 non-null  float32
 10  landtaxvaluedollarcnt         99708 non-null  float32
 11  taxvaluedollarcnt             99708 non-null  float32
 12  rawcensustractandblock        99709 non-null  float32
 13  regionidcity                  97773 non-null  float32
 14  neighborhood_mean_price       99700 non-null  float64
 15  lotsizesquarefeet             88847 non-null  float32
 16  int_transactiondate           99709 non-null  float64
 17  yearbuilt                     99138 non-null  float32
 18  latitude                      99709 non-null  float32
dtypes: float32(16), float64(3)
memory usage: 11.1 MB
```

In [98]:

```python
dimensionality()
```

```
X_train (99709, 35)
X_val   (33572, 35)
X_test  (33567, 35)
```

In [99]:

```python
len(numeric) + len(categorical) + len(ordinal) + 1
```

Out[99]:

```
35
```

Verifica che i sottoinsiemi numerici, categorici e ordinali costituiscano la totalità delle colonne

In [100…

```python
def dim_check():
    return X_train.shape[1] == len(numeric) + len(categorical) + len(ordinal) + 1 # 1: parcelid
```

In [101…

```python
dim_check()
```

Out[101…

```
True
```

In [102…

```python
print(f'Numeric:\n{numeric} ({len(numeric)})\n')
print(f'Categorical:\n{categorical} ({len(categorical)})\n')
print(f'Ordinal:\n{ordinal} ({len(ordinal)})\n')
```

```
Numeric:
['living_area_prop', 'structuretaxvaluedollarcnt', 'taxamount', 'finishedsquarefeet12', 'period_mean_price', 'c
alculatedfinishedsquarefeet', 'longitude', 'tax_prop', 'tax_ratio', 'regionidzip', 'landtaxvaluedollarcnt', 'ta
xvaluedollarcnt', 'rawcensustractandblock', 'regionidcity', 'neighborhood_mean_price', 'lotsizesquarefeet', 'in
t_transactiondate', 'yearbuilt', 'latitude'] (19)

Categorical:
['assessmentyear', 'fips', 'heatingorsystemtypeid', 'poolcnt', 'propertycountylandusecode', 'propertylandusetyp
eid', 'propertyzoningdesc', 'regionidcounty'] (8)

Ordinal:
['bathroomcnt', 'bedroomcnt', 'buildingqualitytypeid', 'calculatedbathnbr', 'fireplacecnt', 'roomcnt', 'unitcn
t'] (7)
```

Missing-flag: numerici e ordinali

Controllo la percentuale di missing value in questi tipi di dati per controllare se è sensato inserire le missing-flags.

In [103…

```python
# Returns rows with nan_percentage over the cut-off
def over_nan_percentage(colnames, cutoff, verbose=False):
    over = []
    for cn in colnames:
        col = get_col(X_train, cn)
        _, perc = get_col_nan_info(col)
        if verbose:
            print(f'{cn}: {perc}')
        if perc > cutoff:
            over.append(cn)
    return over
```

In [104…

```python
put_nan_flag = over_nan_percentage(numeric+ordinal, 0.2, verbose=True)
put_nan_flag
```

```
living_area_prop: 0.11158471151049554
structuretaxvaluedollarcnt: 0.002737967485382463
taxamount: 0.0001002918492814089
finishedsquarefeet12: 0.0484911091275612
period_mean_price: 0.0
calculatedfinishedsquarefeet: 0.004864154690148332
longitude: 0.0
tax_prop: 0.002737967485382463
tax_ratio: 0.00011032103420954979
regionidzip: 0.00048140087655076274
landtaxvaluedollarcnt: 1.002918492814089e-05
taxvaluedollarcnt: 1.002918492814089e-05
rawcensustractandblock: 0.0
regionidcity: 0.019416502020880765
neighborhood_mean_price: 9.0262664353268e-05
lotsizesquarefeet: 0.10893700668946635
int_transactiondate: 0.0
yearbuilt: 0.0057266645939684484
latitude: 0.0
bathroomcnt: 0.0
bedroomcnt: 0.0
buildingqualitytypeid: 0.3625650643372213
calculatedbathnbr: 0.010149535147278581
fireplacecnt: 0.0
roomcnt: 0.0
unitcnt: 0.35170345706004474
```

Out[104…

```
['buildingqualitytypeid', 'unitcnt']
```

I missing value hanno una bassissima percentaule. Solo `buildingqualitytypeid` e `unitcnt` hanno una percentuale superiore al 2%
Solo per queste colonne aggiungo un missing-flag.

In [105…

```python
# Given a dataframe and a column_name adds the missing flag
def add_missing_flag(df, col_name):
    df[col_name+'_na_flag'] = df.loc[:,col_name].isna().astype(int)
    return df
```

In [106…

```python
for df in [X_train, X_val, X_test]:
    for cname in put_nan_flag:
        df = add_missing_flag(df, cname)
```

In [107…

```python
for cname in put_nan_flag:
    print(X_train.loc[:, [cname, cname+'_na_flag']])
```

```
        buildingqualitytypeid  buildingqualitytypeid_na_flag
0                         NaN                              1
1                         8.0                              0
2                         7.0                              0
3                         NaN                              1
4                         4.0                              0
...                       ...                            ...
100727                    7.0                              0
100728                    NaN                              1
100729                    NaN                              1
100730                    NaN                              1
100731                    7.0                              0

[99709 rows x 2 columns]
        unitcnt  unitcnt_na_flag
0           NaN                1
1           1.0                0
2           1.0                0
3           NaN                1
4           1.0                0
...         ...              ...
100727      1.0                0
100728      NaN                1
100729      NaN                1
```

```
100730      NaN                1
100731      1.0                0

[99709 rows x 2 columns]
```

In [108…

```
dimensionality()
```

```
X_train (99709, 37)
X_val   (33572, 37)
X_test  (33567, 37)
```

L'operazione è avvenuta correttamente.

Riempimento dei Nan - numerici e ordinali

Vista la bassa percentuale di Nan individuata, questi sono riempiti con la mediana della colonna.

In [109…

```
# Given a dataframe and its column names fill its Nans with the median value of the column for that region
def fill_nan_with_median_same_country(df, col_names, country_ids):
    for country_id in country_ids:
        df_sub = df[df.loc[:,'regionidcounty'] == country_id]
        df_sub = fill_nan_with_median(df_sub, col_names)
    return df

# Given a dataframe and its column names fill its Nans with the median value of the column
def fill_nan_with_median(df, col_names):
    for col_name in col_names:
        df[col_name] = df[col_name].fillna(get_col(df, col_name).median())
    return df
```

In [110…

```
for X in [X_train, X_val, X_test]:
    X = fill_nan_with_median(X, numeric+ordinal)
    # X = fill_nan_with_median_same_country(X, numeric+ordinal, [1286., 2061., 3101.])
```

In [111…

```
X_train[numeric + ordinal].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 99709 entries, 0 to 100731
Data columns (total 26 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   living_area_prop              99709 non-null  float32
 1   structuretaxvaluedollarcnt    99709 non-null  float32
 2   taxamount                     99709 non-null  float32
 3   finishedsquarefeet12          99709 non-null  float32
 4   period_mean_price             99709 non-null  float64
 5   calculatedfinishedsquarefeet  99709 non-null  float32
 6   longitude                     99709 non-null  float32
 7   tax_prop                      99709 non-null  float32
 8   tax_ratio                     99709 non-null  float32
 9   regionidzip                   99709 non-null  float32
 10  landtaxvaluedollarcnt         99709 non-null  float32
 11  taxvaluedollarcnt             99709 non-null  float32
 12  rawcensustractandblock        99709 non-null  float32
 13  regionidcity                  99709 non-null  float32
 14  neighborhood_mean_price       99709 non-null  float64
 15  lotsizesquarefeet             99709 non-null  float32
 16  int_transactiondate           99709 non-null  float64
 17  yearbuilt                     99709 non-null  float32
 18  latitude                      99709 non-null  float32
 19  bathroomcnt                   99709 non-null  float32
 20  bedroomcnt                    99709 non-null  float32
 21  buildingqualitytypeid         99709 non-null  float32
 22  calculatedbathnbr             99709 non-null  float32
 23  fireplacecnt                  99709 non-null  float32
 24  roomcnt                       99709 non-null  float32
 25  unitcnt                       99709 non-null  float32
dtypes: float32(23), float64(3)
memory usage: 13.8 MB
```

One-Hot encoding delle variabili categoriali

Il One-Hot enconding è effettuato alla luce dei valori del Train e riapplicato in maniera opaca su Validation e Test.
Nell'eventualità che questi due dataset presentassero un valore inedito, il suo encoding sarebbe una riga di zeri per le colonne considerate.

In [112…

```
categorical
```

Out[112…

```
['assessmentyear',
 'fips',
 'heatingorsystemtypeid',
 'poolcnt',
 'propertycountylandusecode',
 'propertylandusetypeid',
 'propertyzoningdesc',
 'regionidcounty']
```

In [113…

```
# Given a train-dataframe, its column-names and a list of dataframes,
# trains a one-hot-encoder to the train
# makes a one-hot-enconding for each dataframe
def one_hot_encoding(df_fit, col_names, dfs):
    oh = OneHotEncoder(sparse=False, handle_unknown='ignore')
    oh.fit(df_fit[col_names])
    for df in dfs:
        encoded = oh.transform(df[col_names])
        for i, col in enumerate(oh.get_feature_names(col_names)):
            df[col] = encoded[:,i]
        df.drop(col_names, axis=1, inplace=True)
```

In [114…

```
one_hot_encoding(X_train, categorical, [X_train, X_val, X_test])
```

In [115…

```
dimensionality()
```

```
X_train (99709, 76)
X_val   (33572, 76)
X_test  (33567, 76)
```

Rimuovo colonne che codficano i Nan per One-Hot-Encoding: mantengo righe di soli zeri.

In [116…

```
nan_column = list(filter(re.compile("^.*_nan$").match, list(X_train.columns)))
print(nan_column)
```

```
['heatingorsystemtypeid_nan']
```

In [117…

```
X_train.columns
```

Out[117…

```
Index(['parcelid', 'bathroomcnt', 'bedroomcnt', 'buildingqualitytypeid',
       'calculatedbathnbr', 'calculatedfinishedsquarefeet',
       'finishedsquarefeet12', 'fireplacecnt', 'latitude', 'longitude',
       'lotsizesquarefeet', 'rawcensustractandblock', 'regionidcity',
       'regionidzip', 'roomcnt', 'unitcnt', 'yearbuilt',
       'structuretaxvaluedollarcnt', 'taxvaluedollarcnt',
       'landtaxvaluedollarcnt', 'taxamount', 'int_transactiondate',
       'period_mean_price', 'neighborhood_mean_price', 'living_area_prop',
       'tax_ratio', 'tax_prop', 'buildingqualitytypeid_na_flag',
       'unitcnt_na_flag', 'assessmentyear_2015.0', 'assessmentyear_2016.0',
       'fips_6037.0', 'fips_6059.0', 'fips_6111.0',
       'heatingorsystemtypeid_1.0', 'heatingorsystemtypeid_2.0',
       'heatingorsystemtypeid_6.0', 'heatingorsystemtypeid_7.0',
       'heatingorsystemtypeid_10.0', 'heatingorsystemtypeid_11.0',
       'heatingorsystemtypeid_12.0', 'heatingorsystemtypeid_13.0',
       'heatingorsystemtypeid_18.0', 'heatingorsystemtypeid_20.0',
       'heatingorsystemtypeid_24.0', 'heatingorsystemtypeid_nan',
       'poolcnt_0.0', 'poolcnt_1.0', 'propertycountylandusecode_0100',
       'propertycountylandusecode_0101', 'propertycountylandusecode_010C',
       'propertycountylandusecode_122', 'propertycountylandusecode_34',
       'propertycountylandusecode_rare', 'propertylandusetypeid_31.0',
       'propertylandusetypeid_246.0', 'propertylandusetypeid_247.0',
       'propertylandusetypeid_248.0', 'propertylandusetypeid_260.0',
       'propertylandusetypeid_261.0', 'propertylandusetypeid_263.0',
       'propertylandusetypeid_264.0', 'propertylandusetypeid_265.0',
       'propertylandusetypeid_266.0', 'propertylandusetypeid_267.0',
       'propertylandusetypeid_269.0', 'propertylandusetypeid_275.0',
       'propertyzoningdesc_LAR1', 'propertyzoningdesc_LAR3',
       'propertyzoningdesc_LARD1.5', 'propertyzoningdesc_LARS',
       'propertyzoningdesc_LBR1N', 'propertyzoningdesc_rare',
       'regionidcounty_1286.0', 'regionidcounty_2061.0',
       'regionidcounty_3101.0'],
      dtype='object')
```

E colonne che potrebbero essere binarie, come `poolcnt` o `assementyear`.

In [118…

```
for X in [X_train, X_val, X_test]:
    X = remove_column(X, nan_column)
    X = remove_column(X, 'poolcnt_0.0')
    X = remove_column(X, 'assessmentyear_2016.0')
```

In [119…

```
dimensionality()
```

```
X_train (99709, 73)
X_val   (33572, 73)
X_test  (33567, 73)
```

Scrittura csv

Salvo i dati processati in una specifica cartella.

In [120…

```
dir_name = 'preparazione'

X_train.to_csv(dir_name + '/X_train.csv', index=False)
y_train.to_csv(dir_name + '/y_train.csv', index=False)
X_val  .to_csv(dir_name + '/X_val.csv',   index=False)
y_val  .to_csv(dir_name + '/y_val.csv',   index=False)
X_test .to_csv(dir_name + '/X_test.csv',  index=False)
y_test .to_csv(dir_name + '/y_test.csv',  index=False)
```