Informe Trabajo Práctico 2

Integrantes:

Condorí, Guillermo 98688

Cajachuán, Kevin 98725

Avigliano, Patricio 98861

Adaptación del set de datos

Join de los datos

En este notebook nos encargamos de juntar todos los csvs de datos a utilizar en un unico csv para trabajar.

In [12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [13]:
```
data1 = pd.read_csv('csvs/properati-AR-2016-01-01-properties-sell.csv')
data2 = pd.read_csv('csvs/properati-AR-2016-02-01-properties-sell.csv')
data3 = pd.read_csv('csvs/properati-AR-2016-03-01-properties-sell.csv')
data4 = pd.read_csv('csvs/properati-AR-2016-04-01-properties-sell.csv')
data5 = pd.read_csv('csvs/properati-AR-2016-05-01-properties-sell.csv')
data6 = pd.read_csv('csvs/properati-AR-2016-06-01-properties-sell.csv')
data7 = pd.read_csv('csvs/properati-AR-2016-07-01-properties-sell.csv')
data8 = pd.read_csv('csvs/properati-AR-2016-08-01-properties-sell.csv')
data9 = pd.read_csv('csvs/properati-AR-2016-09-01-properties-sell.csv')
data10 = pd.read_csv('csvs/properati-AR-2016-10-01-properties-sell.csv')
data11 = pd.read_csv('csvs/properati-AR-2016-11-01-properties-sell.csv')
data12 = pd.read_csv('csvs/properati-AR-2016-12-01-properties-sell.csv')
data13 = pd.read_csv('csvs/properati-AR-2017-01-01-properties-sell.csv')
data14 = pd.read_csv('csvs/properati-AR-2017-02-01-properties-sell.csv')
data15 = pd.read_csv('csvs/properati-AR-2017-03-01-properties-sell.csv')
data16 = pd.read_csv('csvs/properati-AR-2017-04-01-properties-sell.csv')
data17 = pd.read_csv('csvs/properati-AR-2017-05-01-properties-sell.csv')
data18 = pd.read_csv('csvs/properati-AR-2017-06-01-properties-sell.csv')
data19 = pd.read_csv('csvs/properati-AR-2017-06-06-properties-sell.csv')
data20 = pd.read_csv('csvs/properati-AR-2017-07-03-properties-sell.csv')
data21 = pd.read_csv('csvs/properati-AR-2017-08-01-properties-sell.csv')
data22 = pd.read_csv('csvs/properati-AR-2017-09-01-properties-sell-six_months.csv')
data23 = pd.read_csv('csvs/properati-AR-2017-10-01-properties-sell.csv')
```

In [15]:
```
data = pd.concat([data1, data2, data3, data4, data5, data6, data7, data8, data9, data10, \
                  data11, data12, data13, data14, data15, data16, data17, data18, data19, data20, \
                  data21, data22, data23])
```

In [16]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1316434 entries, 0 to 196718
Data columns (total 27 columns):
country_name                  1316434 non-null object
created_on                    1316434 non-null object
currency                      1145291 non-null object
description                   1316356 non-null object
expenses                      142854 non-null float64
floor                         139886 non-null float64
geonames_id                   1053967 non-null float64
id                            1316434 non-null object
image_thumbnail               318274 non-null object
lat                           952738 non-null float64
lat-lon                       952738 non-null object
lon                           952738 non-null float64
operation                     324886 non-null object
place_name                    1316152 non-null object
place_with_parent_names       1316434 non-null object
price                         1162149 non-null float64
price_aprox_local_currency    1162149 non-null float64
price_aprox_usd               1162149 non-null float64
price_per_m2                  1011651 non-null float64
price_usd_per_m2              819976 non-null float64
properati_url                 324886 non-null object
property_type                 1316434 non-null object
rooms                         789737 non-null float64
state_name                    1316434 non-null object
surface_covered_in_m2         1147593 non-null float64
surface_total_in_m2           976190 non-null float64
title                         1316434 non-null object
dtypes: float64(13), object(14)
memory usage: 281.2+ MB
```

In [17]:
```python
data=data.dropna(subset=['lat-lon']).reset_index(drop = True)
```

In [18]:
```python
data = data.loc[(data.place_with_parent_names.str.contains('Capital Federal')) | \
                (data.place_with_parent_names.str.contains('G.B.A.'))]
```

In [19]:
```
data.to_csv("csvs/datosConDuplicados.csv", index = False)
```

In [20]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 875661 entries, 0 to 952737
Data columns (total 27 columns):
country_name                  875661 non-null object
created_on                    875661 non-null object
currency                      767011 non-null object
description                   875641 non-null object
expenses                      110161 non-null float64
floor                         96544 non-null float64
geonames_id                   690233 non-null float64
id                            875661 non-null object
image_thumbnail               130399 non-null object
lat                           875661 non-null float64
lat-lon                       875661 non-null object
lon                           875661 non-null float64
operation                     130399 non-null object
place_name                    875379 non-null object
place_with_parent_names       875661 non-null object
price                         781422 non-null float64
price_aprox_local_currency    781422 non-null float64
price_aprox_usd               781422 non-null float64
price_per_m2                  677973 non-null float64
price_usd_per_m2              556249 non-null float64
properati_url                 130399 non-null object
property_type                 875661 non-null object
rooms                         554343 non-null float64
state_name                    875661 non-null object
surface_covered_in_m2         763140 non-null float64
surface_total_in_m2           663887 non-null float64
title                         875661 non-null object
dtypes: float64(13), object(14)
memory usage: 187.1+ MB
```

Eliminamos algunos registros que no tienen campos fundamentales o estan fuera del rango geografico donde vamos a trabajar.

Eliminar duplicados

En este notebook nos encargamos de eliminar registros que aparecen repetidos en nuestro set de datos. Para hacer esto optamos por ordenar el arreglo por ubicacion geografica y por descripcion, de manera que registros con la misma ubicacion y descripcion (esos son los que vamos a considerar repetidos) queden juntos. Luego al recorrer el arreglo agregamos una columna que indica si el registro debe eliminarse o no.

In [2]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('csvs/datosConDuplicados.csv', low_memory=False)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 875661 entries, 0 to 875660
Data columns (total 27 columns):
country_name                  875661 non-null object
created_on                    875661 non-null object
currency                      767011 non-null object
description                   875641 non-null object
expenses                      110161 non-null float64
floor                         96544 non-null float64
geonames_id                   690233 non-null float64
id                            875661 non-null object
image_thumbnail               130399 non-null object
lat                           875661 non-null float64
lat-lon                       875661 non-null object
lon                           875661 non-null float64
operation                     130399 non-null object
place_name                    875379 non-null object
place_with_parent_names       875661 non-null object
price                         781422 non-null float64
price_aprox_local_currency    781422 non-null float64
price_aprox_usd               781422 non-null float64
price_per_m2                  677973 non-null float64
price_usd_per_m2              556249 non-null float64
properati_url                 130399 non-null object
property_type                 875661 non-null object
rooms                         554343 non-null float64
state_name                    875661 non-null object
surface_covered_in_m2         763140 non-null float64
surface_total_in_m2           663887 non-null float64
title                         875661 non-null object
dtypes: float64(13), object(14)
memory usage: 180.4+ MB
```

In [5]:
```
data=data.dropna(subset=['description'])
```

In [6]:
```
data.sort_values(by = ['lat-lon','description'], inplace=True)
```

```
In [7]: data = data[data.price_aprox_usd.notnull()]
```

```
In [8]: data = data.reset_index(drop = True)
```

```
In [9]: data['duplicado']=True;
i=0
while i < (len(data.index)):
    j = i + 1
    while((data.loc[i,'lat-lon'] == data.loc[j,'lat-lon']) & (data.loc[i,'description'] == data.loc[j,'descri
        j = j + 1
        if(j == len(data.index)):
            break;
    maxi = -1
    max_idx = i
    for x in range(i,j):
        if data.loc[x,'price_aprox_usd'] > maxi:
            maxi = data.loc[x,'price_aprox_usd']
            max_idx = x
    data.loc[max_idx,'duplicado']=False
    i = j;
```

```
In [10]: data = data.loc[data.duplicado == False]
```

Aqui nos deshacemos de los elementos que aparecen duplicados. Un detalle para nada menor es el criterio con que se selecciona el registro que sobrevivirá al encontrar varios registros que hacen referencia a la misma propiedad. Probamos 3 criterios distintos:

1) El registro que tiene menor precio sobrevive

2) El registro que tiene mayor precio sobrevive

3) Se calcula el promedio entre el mayor y menor precio encontrado

Despues de probar los 3 criterios observamos que el que arrojaba mejores resultados era el numero 2 y ese fue el criterio que elegimos.

In [15]:
```
data[data['lat-lon'] == '-34.3402525,-58.7849434']
```

Out[15]:

	country_name	created_on	currency	description	expenses	floor	geonames_id	id	
3048	Argentina	2017-08-27	USD	EXCELENTE CASA DESARROLL...	NaN	NaN	3434130.0	f60bd2e07f7a54b9c4909b41379e2e0588f5f011	h
3049	Argentina	2017-08-27	USD	CONTACTO 155411811313 ...	NaN	NaN	NaN	ce2b63f1ab2931d49bcb9d68910cebd39c470bd3	I
3051	Argentina	2017-08-27	USD	MUY BUENA CASA EN CONSTRUCC...	NaN	NaN	3434130.0	99800c7ceb8090ca8cde2ee4711fc6fa97d301ee	
3052	Argentina	2017-08-27	USD	CONTACTO	NaN	NaN	NaN	988704da493999ff7999f03f4913b2478a706eb1	http

In [12]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 182407 entries, 0 to 781399
Data columns (total 28 columns):
country_name                  182407 non-null object
created_on                    182407 non-null object
currency                      178752 non-null object
description                   182407 non-null object
expenses                      25400 non-null float64
floor                         22438 non-null float64
geonames_id                   145856 non-null float64
id                            182407 non-null object
image_thumbnail               34078 non-null object
lat                           182407 non-null float64
lat-lon                       182407 non-null object
lon                           182407 non-null float64
operation                     34078 non-null object
place_name                    182343 non-null object
place_with_parent_names       182407 non-null object
price                         182407 non-null float64
price_aprox_local_currency    182407 non-null float64
price_aprox_usd               182407 non-null float64
price_per_m2                  158759 non-null float64
price_usd_per_m2              126164 non-null float64
properati_url                 34078 non-null object
property_type                 182407 non-null object
rooms                         112645 non-null float64
state_name                    182407 non-null object
surface_covered_in_m2         162848 non-null float64
surface_total_in_m2           134937 non-null float64
title                         182407 non-null object
duplicado                     182407 non-null bool
dtypes: bool(1), float64(13), object(14)
memory usage: 39.1+ MB
```

In [13]:
```python
data.to_csv("csvs/datosSinDuplicados.csv", index = False)
```

Filtrando anomalos

En este notebook lo que hicimos fue identificar los datos anomalos, por ejemplo no tiene sentido que una propiedad del tipo 'casa' este el el floor 250. Lo que elegimos hacer fue reemplazar esos datos anomalos por NaNs para no perder todo el registro solo por un dato incorrecto.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
data = pd.read_csv('csvs/datosSinDuplicados.csv', low_memory=False)
```

In [3]:
```
data.drop_duplicates('id', keep = 'last', inplace = True)
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 163260 entries, 0 to 182406
Data columns (total 28 columns):
country_name                  163260 non-null object
created_on                    163260 non-null object
currency                      159636 non-null object
description                   163260 non-null object
expenses                      23131 non-null float64
floor                         19585 non-null float64
geonames_id                   130373 non-null float64
id                            163260 non-null object
image_thumbnail               30237 non-null object
lat                           163260 non-null float64
lat-lon                       163260 non-null object
lon                           163260 non-null float64
operation                     30237 non-null object
place_name                    163206 non-null object
place_with_parent_names       163260 non-null object
price                         163260 non-null float64
price_aprox_local_currency    163260 non-null float64
price_aprox_usd               163260 non-null float64
price_per_m2                  141503 non-null float64
price_usd_per_m2              113089 non-null float64
properati_url                 30237 non-null object
property_type                 163260 non-null object
rooms                         100416 non-null float64
state_name                    163260 non-null object
surface_covered_in_m2         145505 non-null float64
surface_total_in_m2           121590 non-null float64
title                         163260 non-null object
duplicado                     163260 non-null bool
dtypes: bool(1), float64(13), object(14)
memory usage: 35.0+ MB
```

In [5]:
```
del data['country_name']
del data['geonames_id']
del data['image_thumbnail']
del data['id']
del data['lat-lon']
del data['operation']
del data['place_name']
del data['properati_url']
del data['state_name']
del data['duplicado']
del data['title']
del data['price']
del data['currency']
del data['price_per_m2']
del data['price_aprox_local_currency']
del data['price_usd_per_m2']
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 163260 entries, 0 to 182406
Data columns (total 12 columns):
created_on                 163260 non-null object
description                163260 non-null object
expenses                   23131 non-null float64
floor                      19585 non-null float64
lat                        163260 non-null float64
lon                        163260 non-null float64
place_with_parent_names    163260 non-null object
price_aprox_usd            163260 non-null float64
property_type              163260 non-null object
rooms                      100416 non-null float64
surface_covered_in_m2      145505 non-null float64
surface_total_in_m2        121590 non-null float64
dtypes: float64(8), object(4)
memory usage: 16.2+ MB
```

In [6]:
```
data.loc[data.floor > 70, 'floor'] = np.NaN
```

In [7]:
```
data.loc[data.rooms > 12, 'rooms'] = np.NaN
```

In [8]:
```
data.loc[data.surface_total_in_m2 < 1, 'surface_total_in_m2'] = np.NaN
```

In [9]:
```
data.loc[((data.surface_total_in_m2 > 70000) & (data.property_type == "apartment")), 'surface_total_in_m2'] =
```

In [10]:
```
data.loc[data.surface_covered_in_m2 < 1, 'surface_covered_in_m2'] = np.NaN
```

In [11]:
```
data.loc[data.surface_covered_in_m2 > data.surface_total_in_m2, 'surface_covered_in_m2'] = np.NaN
```

In [12]:
```
data.loc[data.expenses < 150, 'expenses'] = np.NaN
```

In [13]:
```
data.loc[data.expenses > 150000, 'expenses'] = np.NaN
```

In [14]:
```
data = data.loc[(data.lat<-34) & (data.lat>-35) & (data.lon>-60) & (data.lon<-57.5)]
```

In [15]:
```
data = data[data.price_aprox_usd!=0]
```

Esto ultimo es muy importante, eliminar todo registro cuyo precio sea igual a 0, ya que si estuvieran se usarian para predecir y afectaria mucho los resultados.

In [16]:
```
data = data.reset_index(drop = True)
```

In [17]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158352 entries, 0 to 158351
Data columns (total 12 columns):
created_on                  158352 non-null object
description                 158352 non-null object
expenses                    21572 non-null float64
floor                       18702 non-null float64
lat                         158352 non-null float64
lon                         158352 non-null float64
place_with_parent_names     158352 non-null object
price_aprox_usd             158352 non-null float64
property_type               158352 non-null object
rooms                       97155 non-null float64
surface_covered_in_m2       139468 non-null float64
surface_total_in_m2         112609 non-null float64
dtypes: float64(8), object(4)
memory usage: 14.5+ MB
```

In [18]:
```
data.to_csv("csvs/datosFiltrados.csv", index = False)
```

Reemplazando NaNs

Para poder correr algoritmos es necesario tener todos los campos completos en nuestro dataset. Por esto en este notebook nos ocupamos de estimar los campos que no estan para luego poder entrenar nuestros estimadores.

In [113]:
```
import pandas as pd
import numpy as np
```

In [114]:
```
data = pd.read_csv('csvs/datosFiltrados.csv', low_memory=False)
```

In [115]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158352 entries, 0 to 158351
Data columns (total 12 columns):
created_on                  158352 non-null object
description                 158352 non-null object
expenses                    21572 non-null float64
floor                       18702 non-null float64
lat                         158352 non-null float64
lon                         158352 non-null float64
place_with_parent_names     158352 non-null object
price_aprox_usd             158352 non-null float64
property_type               158352 non-null object
rooms                       97155 non-null float64
surface_covered_in_m2       139468 non-null float64
surface_total_in_m2         112609 non-null float64
dtypes: float64(8), object(4)
memory usage: 14.5+ MB
```

In [116]:
```
data.loc[data.expenses.notnull(), 'expenses/price'] = data['expenses']/data['price_aprox_usd']
```

In [117]:
```
meanHouse = data[data['property_type'] == 'house']['expenses/price'].mean()
meanApart = data[data['property_type'] == 'apartment']['expenses/price'].mean()
meanPH = data[data['property_type'] == 'PH']['expenses/price'].mean()
meanStore = data[data['property_type'] == 'store']['expenses/price'].mean()

data.loc[data['property_type'] == "apartment",'expenses/price'] = data['expenses/price'].apply(lambda x: mear
data.loc[data['property_type'] == "house",'expenses/price'] = data['expenses/price'].apply(lambda x: meanHous
data.loc[data['property_type'] == "PH",'expenses/price'] = data['expenses/price'].apply(lambda x: meanPH if r
data.loc[data['property_type'] == "store",'expenses/price'] = data['expenses/price'].apply(lambda x: meanStor
```

In [118]:
```
data.loc[data.expenses.isnull(), 'expenses'] = data['expenses/price']*data['price_aprox_usd']
del data['expenses/price']
```

In [119]:
```
data.loc[data.surface_total_in_m2.notnull(), 'surface/price'] = data['surface_total_in_m2']/data['price_aprox
```

```
In [120]: meanHouse = data[data['property_type'] == 'house']['surface/price'].mean()
meanApart = data[data['property_type'] == 'apartment']['surface/price'].mean()
meanPH = data[data['property_type'] == 'PH']['surface/price'].mean()
meanStore = data[data['property_type'] == 'store']['surface/price'].mean()

data.loc[data['property_type'] == "apartment",'surface/price'] = data['surface/price'].apply(lambda x: meanAp
data.loc[data['property_type'] == "house",'surface/price'] = data['surface/price'].apply(lambda x: meanHouse
data.loc[data['property_type'] == "PH",'surface/price'] = data['surface/price'].apply(lambda x: meanPH if np.
data.loc[data['property_type'] == "store",'surface/price'] = data['surface/price'].apply(lambda x: meanStore
```

```
In [122]: data.loc[data.surface_total_in_m2.isnull(), 'surface_total_in_m2'] = data['surface/price']*data['price_aprox_
del data['surface/price']
```

```
In [123]: data.loc[data.surface_covered_in_m2.notnull(), 'surface/price'] = data['surface_covered_in_m2']*data['price_a
```

```
In [124]: meanHouse = data[data['property_type'] == 'house']['surface/price'].mean()
meanApart = data[data['property_type'] == 'apartment']['surface/price'].mean()
meanPH = data[data['property_type'] == 'PH']['surface/price'].mean()
meanStore = data[data['property_type'] == 'store']['surface/price'].mean()

data.loc[data['property_type'] == "apartment",'surface/price'] = data['surface/price'].apply(lambda x: meanAp
data.loc[data['property_type'] == "house",'surface/price'] = data['surface/price'].apply(lambda x: meanHouse
data.loc[data['property_type'] == "PH",'surface/price'] = data['surface/price'].apply(lambda x: meanPH if np.
data.loc[data['property_type'] == "store",'surface/price'] = data['surface/price'].apply(lambda x: meanStore
```

```
In [125]: data.loc[data.surface_covered_in_m2.isnull(),'surface_covered_in_m2'] = data['surface/price']*data['surface_t
del data['surface/price']
```

```
In [126]: data.loc[data.rooms.notnull(), 'rooms*price'] = data['rooms']*data['price_aprox_usd']
```

In [127]:
```
meanHouse = data[data['property_type'] == 'house']['rooms*price'].mean()
meanApart = data[data['property_type'] == 'apartment']['rooms*price'].mean()
meanPH = data[data['property_type'] == 'PH']['rooms*price'].mean()
meanStore = data[data['property_type'] == 'store']['rooms*price'].mean()

data.loc[data['property_type'] == "apartment",'rooms*price'] = data['rooms*price'].apply(lambda x: meanApart
data.loc[data['property_type'] == "house",'rooms*price'] = data['rooms*price'].apply(lambda x: meanHouse if r
data.loc[data['property_type'] == "PH",'rooms*price'] = data['rooms*price'].apply(lambda x: meanPH if np.isna
data.loc[data['property_type'] == "store",'rooms*price'] = data['rooms*price'].apply(lambda x: meanStore if r
```

In [128]:
```
data.loc[data.rooms.isnull(),'rooms'] = data['rooms*price']/data['price_aprox_usd']
del data['rooms*price']
```

In [129]:
```
data['rooms'] = data['rooms'].apply(lambda x: (float)((int)(x)))
```

In [130]:
```
data.loc[data.floor.notnull(), 'floor*price'] = data['floor']*data['price_aprox_usd']
```

In [131]:
```
meanHouse = data[data['property_type'] == 'house']['floor*price'].mean()
meanApart = data[data['property_type'] == 'apartment']['floor*price'].mean()
meanPH = data[data['property_type'] == 'PH']['floor*price'].mean()
meanStore = data[data['property_type'] == 'store']['floor*price'].mean()

data.loc[data['property_type'] == "apartment",'floor*price'] = data['floor*price'].apply(lambda x: meanApart
data.loc[data['property_type'] == "house",'floor*price'] = data['floor*price'].apply(lambda x: meanHouse if r
data.loc[data['property_type'] == "PH",'floor*price'] = data['floor*price'].apply(lambda x: meanPH if np.isna
data.loc[data['property_type'] == "store",'floor*price'] = data['floor*price'].apply(lambda x: meanStore if r
```

In [132]:
```
data.loc[data.floor.isnull(),'floor'] = data['floor*price']/data['price_aprox_usd']
del data['floor*price']
```

In [133]:
```
data['floor'] = data['floor'].apply(lambda x: (float)((int)(x)))
```

In [141]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158352 entries, 0 to 158351
Data columns (total 12 columns):
created_on                 158352 non-null object
description                158352 non-null object
expenses                   158352 non-null float64
floor                      158352 non-null float64
lat                        158352 non-null float64
lon                        158352 non-null float64
place_with_parent_names    158352 non-null object
price_aprox_usd            158352 non-null float64
property_type              158352 non-null object
rooms                      158352 non-null float64
surface_covered_in_m2      158352 non-null float64
surface_total_in_m2        158352 non-null float64
dtypes: float64(8), object(4)
memory usage: 14.5+ MB
```

In [135]:
```
data.to_csv("csvs/datosSinNan.csv", index = False)
```

Agregando distancias

Tambien probamos agregar campos como distancia minima a una estacion de subte, metrobus y ferrocarril. Pero empeoraron nuestros resultados asi que dejamos de lado esos campos.

In [26]:
```
import pandas as pd
import numpy as np
```

In [27]:
```
data = pd.read_csv('csvs/datosFiltrados.csv', low_memory=False)
```

In [28]:
```
subte = pd.read_csv('../Datos Capital/estaciones-de-subte.csv',low_memory=False)
```

In [29]:
```
metrobus = pd.read_csv('../Datos Capital/estaciones-de-metrobus.csv',low_memory=False)
```

```
In [30]: tren = pd.read_csv('../Datos Capital/estaciones-de-ferrocarril.csv',low_memory=False)
```

```
In [31]: for i in range(len(data.index)):
             data.loc[i,'dist_subte'] = 99999999999
             for j in range(len(subte.index)):
                     aux = abs(data.loc[i,'lat'] - subte.loc[j,'lat']) + abs(data.loc[i,'lon'] - subte.loc[j,'lon'])
                     if aux < data.loc[i,'dist_subte']:
                         data.loc[i,'dist_subte'] = aux
             data.loc[i,'dist_tren'] = 99999999999
             for j in range(len(tren.index)):
                     aux = abs(data.loc[i,'lat'] - tren.loc[j,'lat']) + abs(data.loc[i,'lon'] - tren.loc[j,'lon'])
                     if aux < data.loc[i,'dist_tren']:
                         data.loc[i,'dist_tren'] = aux
             data.loc[i,'dist_metrobus'] = 99999999999
             for j in range(len(metrobus.index)):
                     aux = abs(data.loc[i,'lat'] - metrobus.loc[j,'lat']) + abs(data.loc[i,'lon'] - metrobus.loc[j,'lo
                     if aux < data.loc[i,'dist_metrobus']:
                         data.loc[i,'dist_metrobus'] = aux
```

```
In [33]: data.to_csv("csvs/datosAgregados.csv", index = False)
```

Arreglando el set de pruebas

En este notebook nos ocupamos de completar todos los campos nulos del set de pruebas.

```
In [268]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          %matplotlib inline
          from sklearn.neighbors import KNeighborsRegressor
```

```
In [270]: datatest = pd.read_csv('csvs/properati_dataset_testing_noprice.csv', low_memory=False)
```

In [271]:
```
datatest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14166 entries, 0 to 14165
Data columns (total 17 columns):
id                          14166 non-null int64
created_on                  14166 non-null object
property_type               14166 non-null object
operation                   14166 non-null object
place_name                  14166 non-null object
place_with_parent_names     14166 non-null object
country_name                14166 non-null object
state_name                  14166 non-null object
lat-lon                     10487 non-null object
lat                         10487 non-null float64
lon                         10487 non-null float64
surface_total_in_m2         11853 non-null float64
surface_covered_in_m2       13005 non-null float64
floor                       1368 non-null float64
rooms                       7500 non-null float64
expenses                    2543 non-null object
description                 14166 non-null object
dtypes: float64(6), int64(1), object(10)
memory usage: 1.8+ MB
```

In [272]:
```
meanLat = datatest[['place_name','lat']].groupby('place_name').agg(np.mean)
meanLon = datatest[['place_name','lon']].groupby('place_name').agg(np.mean)
```

In [273]:
```
for place in meanLon.index:
    datatest.loc[((datatest.lon.isnull()) & (datatest.place_name == place)), 'lon'] = meanLon.loc[place]['lon
    datatest.loc[((datatest.lat.isnull()) & (datatest.place_name == place)), 'lat'] = meanLat.loc[place]['lat
```

Para los campos que no tienen ubicacion geografica, nos fijamos en que barrio estaban y le asignamos una ubicacion que sea coherente con el barrio en que estan.

In [274]:
```
noLatLon = datatest[datatest.lat.isnull()].groupby('place_name').agg(np.size)
noLatLon
```

Out[274]:

place_name	id	created_on	property_type	operation	place_with_parent_names	country_name	state_name	lat-lon	lat	lon	surface_total_in_m2
Abril Club de Campo	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Altos de Hudson I	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Bs.As. G.B.A. Zona Oeste	13	13	13	13	13	13	13	13	13.0	13.0	13.0
Buenos Aires Interior	1	1	1	1	1	1	1	1	1.0	1.0	1.0
El Rocío	1	1	1	1	1	1	1	1	1.0	1.0	1.0
González Catán	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Gregorio de Laferrere	5	5	5	5	5	5	5	5	5.0	5.0	5.0
Haras San Pablo	2	2	2	2	2	2	2	2	2.0	2.0	2.0
La horqueta de Echeverría	3	3	3	3	3	3	3	3	3.0	3.0	3.0
Malvinas Argentinas	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Prados del Oeste	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Solar del Bosque	2	2	2	2	2	2	2	2	2.0	2.0	2.0
Sourigues	1	1	1	1	1	1	1	1	1.0	1.0	1.0
Terralagos	3	3	3	3	3	3	3	3	3.0	3.0	3.0
Villa Celina	7	7	7	7	7	7	7	7	7.0	7.0	7.0

In [275]:
```
datatest.loc[datatest.place_name == 'Abril Club de Campo', 'lat'] = -34.802221
datatest.loc[datatest.place_name == 'Abril Club de Campo', 'lon'] = -58.164291
```

In [276]:
```
datatest.loc[datatest.place_name == 'Bs.As. G.B.A. Zona Oeste', 'lat'] = datatest.loc[datatest.state_name ==
datatest.loc[datatest.place_name == 'Bs.As. G.B.A. Zona Oeste', 'lon'] = datatest.loc[datatest.state_name ==
```

In [277]:
```
datatest.loc[datatest.place_name == 'El Rocío', 'lat'] = -34.867969
datatest.loc[datatest.place_name == 'El Rocío', 'lon'] = -58.488543
```

In [278]:
```
datatest.loc[datatest.place_name == "Buenos Aires Interior",'lat'] = -34.708663
datatest.loc[datatest.place_name == "Buenos Aires Interior",'lon'] = -58.973401
```

In [279]:
```
datatest.loc[datatest.place_name == "Altos de Hudson I",'lat'] = -34.862623
datatest.loc[datatest.place_name == "Altos de Hudson I",'lon'] = -58.168622
```

In [280]:
```
datatest.loc[datatest.place_name == "González Catán",'lat'] = -34.769133
datatest.loc[datatest.place_name == "González Catán",'lon'] = -58.628477
```

In [281]:
```
datatest.loc[datatest.place_name == "Gregorio de Laferrere",'lat'] = -34.743953
datatest.loc[datatest.place_name == "Gregorio de Laferrere",'lon'] = -58.592342
```

In [282]:
```
datatest.loc[datatest.place_name == "Haras San Pablo",'lat'] = -34.606195
datatest.loc[datatest.place_name == "Haras San Pablo",'lon'] = -59.038684
```

In [283]:
```
datatest.loc[datatest.place_name == "La horqueta de Echeverría",'lat'] = -34.897656
datatest.loc[datatest.place_name == "La horqueta de Echeverría",'lon'] = -58.484175
```

In [284]:
```
datatest.loc[datatest.place_name == "Malvinas Argentinas",'lat'] = -34.482412
datatest.loc[datatest.place_name == "Malvinas Argentinas",'lon'] = -58.717893
```

In [285]:
```
datatest.loc[datatest.place_name == "Prados del Oeste",'lat'] = -34.594077
datatest.loc[datatest.place_name == "Prados del Oeste",'lon'] = -58.829508
```

In [286]:
```
datatest.loc[datatest.place_name == "Solar del Bosque",'lat'] = -34.905555
datatest.loc[datatest.place_name == "Solar del Bosque",'lon'] = -58.507635
```

In [287]:
```
datatest.loc[datatest.place_name == "Sourigues",'lat'] = -34.800140
datatest.loc[datatest.place_name == "Sourigues",'lon'] = -58.220011
```

In [288]:
```
datatest.loc[datatest.place_name == "Terralagos",'lat'] = -34.907001
datatest.loc[datatest.place_name == "Terralagos",'lon'] = -58.514885
```

In [289]:
```
datatest.loc[datatest.place_name == "Villa Celina",'lat'] = -34.706311
datatest.loc[datatest.place_name == "Villa Celina",'lon'] = -58.483025
```

In [290]:
```
del datatest['lat-lon']
```

In [291]:
```
datatest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14166 entries, 0 to 14165
Data columns (total 16 columns):
id                          14166 non-null int64
created_on                  14166 non-null object
property_type               14166 non-null object
operation                   14166 non-null object
place_name                  14166 non-null object
place_with_parent_names     14166 non-null object
country_name                14166 non-null object
state_name                  14166 non-null object
lat                         14166 non-null float64
lon                         14166 non-null float64
surface_total_in_m2         11853 non-null float64
surface_covered_in_m2       13005 non-null float64
floor                       1368 non-null float64
rooms                       7500 non-null float64
expenses                    2543 non-null object
description                 14166 non-null object
dtypes: float64(6), int64(1), object(9)
memory usage: 1.7+ MB
```

In [292]:
```
def property_type_to_num(x):
    if x == "departamento":
        return 0
    if x == "casa":
        return 1
    return 2
```

In [293]:
```
datatest['property_type'] = datatest['property_type'].apply(lambda x: property_type_to_num(x))
```

In [294]:
```
datatest.loc[datatest.surface_covered_in_m2 > datatest.surface_total_in_m2,'aux'] = \
datatest.loc[datatest.surface_covered_in_m2 > datatest.surface_total_in_m2, 'surface_covered_in_m2']

datatest.loc[datatest.surface_covered_in_m2 > datatest.surface_total_in_m2, 'surface_covered_in_m2'] = \
datatest.loc[datatest.surface_covered_in_m2 > datatest.surface_total_in_m2, 'surface_total_in_m2']

datatest.loc[datatest.aux > datatest.surface_total_in_m2, 'surface_total_in_m2'] = \
datatest.loc[datatest.aux > datatest.surface_total_in_m2, 'aux']
```

In [295]:
```
del datatest['aux']
```

In [296]:
```
datatest.loc[datatest.surface_total_in_m2 == 0, 'surface_total_in_m2'] = np.NaN
```

Para el set de pruebas nos ayudamos de knn para predecir los campos nulos.

In [297]:
```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor(n_neighbors = 3, p = 1, weights = 'distance')
```

In [298]:
```
cond = ((datatest.surface_covered_in_m2.isnull()) & (datatest.surface_total_in_m2.isnull()))
rest = datatest[cond == False]
train = rest[rest.surface_covered_in_m2.notnull()]
rest = rest[rest.surface_covered_in_m2.isnull()]
test = datatest[cond]
```

In [299]:
```
knn.fit(train[['property_type', 'lat', 'lon']], train['surface_covered_in_m2'])
```

Out[299]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [300]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon']])
```

In [301]:
```
test= test.reset_index(drop = True)
test.surface_covered_in_m2 = pd.Series(predictions)
```

In [302]:
```
datatest = pd.concat([train, test, rest])
```

In [303]:
```
datatest = datatest.reset_index(drop = True)
```

In [304]:
```
datatest['covered/total'] = datatest['surface_covered_in_m2']/datatest['surface_total_in_m2']
```

In [305]:
```
rest = datatest[datatest.surface_total_in_m2.isnull()]
train = datatest[datatest.surface_total_in_m2.notnull()]
test = train[train['covered/total'].isnull()]
train = train[train['covered/total'].notnull()]
```

In [306]:
```
knn.fit(train[['property_type', 'lat', 'lon','surface_total_in_m2']], train['covered/total'])
```

Out[306]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [307]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon','surface_total_in_m2']])
```

In [308]:
```
test= test.reset_index(drop = True)
test['covered/total'] = pd.Series(predictions)
datatest = pd.concat([train, test, rest])
datatest = datatest.reset_index(drop = True)
datatest.loc[datatest['surface_covered_in_m2'].isnull(),'surface_covered_in_m2'] = datatest['surface_total_ir
```

In [309]:
```
del datatest['covered/total']
datatest['covered/total'] = datatest['surface_covered_in_m2']/datatest['surface_total_in_m2']
```

In [310]:
```
test = datatest[datatest['covered/total'].isnull()]
train = datatest[datatest['covered/total'].notnull()]
```

In [311]:
```
knn.fit(train[['property_type', 'lat', 'lon','surface_covered_in_m2']], train['covered/total'])
```

Out[311]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [312]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon','surface_covered_in_m2']])
```

In [313]:
```
test= test.reset_index(drop = True)
test['covered/total'] = pd.Series(predictions)
datatest = pd.concat([train, test])
datatest = datatest.reset_index(drop = True)
```

In [314]:
```
datatest.loc[datatest.surface_total_in_m2.isnull(),'surface_total_in_m2'] = datatest['surface_covered_in_m2']
del datatest['covered/total']
```

In [315]:
```
datatest.loc[datatest.floor > 100, 'floor'] = np.NaN
```

In [316]:
```
test = datatest[datatest['floor'].isnull()]
train = datatest[datatest['floor'].notnull()]
```

In [317]:
```
knn.fit(train[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2']], train['floor'])
```

Out[317]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [318]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2']]
```

In [319]:
```
test= test.reset_index(drop = True)
test['floor'] = pd.Series(predictions)
datatest = pd.concat([train, test])
datatest = datatest.reset_index(drop = True)
```

In [320]:
```
datatest['floor'] = datatest['floor'].apply(lambda x: (float)((int)(x)))
```

In [321]:
```
datatest.loc[datatest.rooms>16,'rooms'] = np.NaN
```

In [322]:
```
test = datatest[datatest['rooms'].isnull()]
train = datatest[datatest['rooms'].notnull()]
```

In [323]:
```
knn.fit(train[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2','floor']], train['
```

Out[323]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [324]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2','
```

In [325]:
```
test= test.reset_index(drop = True)
test['rooms'] = pd.Series(predictions)
datatest = pd.concat([train, test])
datatest = datatest.reset_index(drop = True)
```

In [326]:
```
datatest['rooms'] = datatest['rooms'].apply(lambda x: (float)((int)(x)))
```

In [327]:
```
datatest['rooms'] = datatest['rooms'].apply(lambda x: 1.0 if x<1 else x)
```

In [328]:
```
datatest.loc[datatest.expenses.str.isdigit() == False,'expenses'] = np.NaN
```

In [329]:
```
datatest['expenses'] = datatest['expenses'].apply(lambda x : float(x))
```

In [330]:
```
datatest.loc[datatest.expenses >50000,'expenses'] = np.NaN
```

In [331]:
```
test = datatest[datatest['expenses'].isnull()]
train = datatest[datatest['expenses'].notnull()]
```

In [332]:
```
knn.fit(train[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2','floor','rooms']],
```

Out[332]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [333]:
```
predictions = knn.predict(test[['property_type', 'lat', 'lon','surface_covered_in_m2','surface_total_in_m2','
```

In [334]:
```
test= test.reset_index(drop = True)
test['expenses'] = pd.Series(predictions)
datatest = pd.concat([train, test])
datatest = datatest.reset_index(drop = True)
```

In [335]:
```
datatest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14166 entries, 0 to 14165
Data columns (total 16 columns):
id                            14166 non-null int64
created_on                    14166 non-null object
property_type                 14166 non-null int64
operation                     14166 non-null object
place_name                    14166 non-null object
place_with_parent_names       14166 non-null object
country_name                  14166 non-null object
state_name                    14166 non-null object
lat                           14166 non-null float64
lon                           14166 non-null float64
surface_total_in_m2           14166 non-null float64
surface_covered_in_m2         14166 non-null float64
floor                         14166 non-null float64
rooms                         14166 non-null float64
expenses                      14166 non-null float64
description                   14166 non-null object
dtypes: float64(7), int64(2), object(7)
memory usage: 1.7+ MB
```

In [337]:
```
datatest.to_csv("csvs/datatestSinNan.csv", index = False)
```

Algoritmos

Grid Search

Para poder hallar los hiperparámetros de cada algoritmo que probamos, usamos Grid Search con valores que se consideraban lógicos. A continuación se muestra el código de Grid Search para KNN, pero se utilizó para todos los algoritmos, cambiando en cada caso, el diccionario con los hiperparámetros.

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

In [2]:
```
data = pd.read_csv("datosSinNan.csv")
```

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(data[['expenses', 'floor', 'lat', 'lon',\
                                                           'rooms', 'surface_covered_in_m2', 'surface_total_ir
                                                     data[['price_aprox_usd']], test_size=0.1, random_state=0)
```

In [4]:
```
parameters = {'n_neighbors':[1,2,3,4,5,6,7,8,9,10], 'weights':['uniform', 'distance'], 'p':[1,2,3,4,5]}
```

In [5]:
```
gs = GridSearchCV(KNeighborsRegressor(), parameters, cv = 5, scoring = 'neg_mean_squared_error')
gs.fit(X_train, y_train)
```

Out[5]:
```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], 'weights': ['uniform', 'distance'], 'p':
 [1, 2, 3, 4, 5]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

In [6]:
```
gs.best_params_
```

Out[6]:
```
{'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
```

In [7]:
```
y_true, y_pred = y_test, gs.predict(X_test)
```

In [8]:
```
mean_squared_error(y_true, y_pred)
```

Out[8]:
```
10270721834.252747
```

KNN

El primer algoritmo que probamos fue KNN sin filtrar aquellas propiedades que tenían "store" en el campo property_type. El score en Kaggle fue 973339015524.93700

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsRegressor
```

In [2]:
```
data = pd.read_csv("datosSinNan.csv")
test = pd.read_csv("datatestSinNan.csv")
```

In [3]:
```
def property_type_to_num(x):
    if x == "apartment":
        return 0
    if x == "house":
        return 1
    if x == "PH":
        return 2
    return 3
```

In [4]:
```
data['property_type'] = data['property_type'].apply(lambda x: property_type_to_num(x))
```

In [6]:
```
knn = KNeighborsRegressor(n_neighbors=3, weights='distance', p=1)
```

In [7]:
```
knn.fit(data[['expenses', 'floor', 'lat', 'lon', 'property_type', \
              'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']], \
        data[['price_aprox_usd']])
```

Out[7]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=1,
          weights='distance')
```

In [8]:
```
predictions = knn.predict(test[['expenses', 'floor', 'lat', 'lon', 'property_type',\
                                'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']])
```

In [9]:
```
pr = [predictions[i][0] for i in range(len(predictions))]
```

In [10]:
```
test['price_usd'] = pd.Series(pr)
```

In [11]:
```
submit = test[['id', 'price_usd']]
```

In [13]:
```
submit.to_csv("properati_dataset_sample_submision.csv", index = False)
```

Al filtrar las propiedades de tipo "store", el score en Kaggle mejoró (973189725366.30400), por lo que decidimos filtrar estas propiedades en todos los algoritmos que probamos.

Decision Tree

Con este algoritmo, al momento de hacer el Grid Search, el score que daba con la validación era menor que el que se obtuvo con KNN, sin embargo, al subir las predicciones a Kaggle, el score de este algoritmo fue mucho peor que con KNN (1005876469824.70000). Esto empeoró comparandolo incluso con la predicción utilizando Decision Tree con los valores por defecto (992801565782.56500).

In [14]:
```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeRegressor
```

In [15]:
```
data = pd.read_csv("datosSinNan.csv")
test = pd.read_csv("datatestSinNan.csv")
```

In [16]:
```
data = data[data.property_type != "store"]
```

In [17]:
```
def property_type_to_num(x):
    if x == "apartment":
        return 0
    if x == "house":
        return 1
    if x == "PH":
        return 2
```

In [18]:
```
data['property_type'] = data['property_type'].apply(lambda x: property_type_to_num(x))
```

In [19]:
```
dt = DecisionTreeRegressor(max_depth = 10, min_samples_split = 3)
```

In [20]:
```
dt.fit(data[['expenses', 'floor', 'lat', 'lon', 'property_type', \
             'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']], \
        data[['price_aprox_usd']])
```

Out[20]:
```
DecisionTreeRegressor(criterion='mse', max_depth=10, max_features=None,
           max_leaf_nodes=None, min_impurity_decrease=0.0,
           min_impurity_split=None, min_samples_leaf=1,
           min_samples_split=3, min_weight_fraction_leaf=0.0,
           presort=False, random_state=None, splitter='best')
```

In [21]:
```
predictions = dt.predict(test[['expenses', 'floor', 'lat', 'lon', 'property_type',\
                                'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']])
```

In [22]:
```
test['price_usd'] = pd.Series(predictions)
```

In [23]:
```
submit = test[['id', 'price_usd']]
```

In [24]:
```
submit.to_csv("properati_dataset_sample_submision.csv", index = False)
```

Random Forest

Utilizando este algoritmo con los hiperparámetros obtenidos con el Grid Search, pudimos mejorar el score obtenido con KNN en Kaggle (967248246642.93500).

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

In [2]:
```
data = pd.read_csv("datosSinNan.csv")
test = pd.read_csv("datatestSinNan.csv")
```

In [3]:
```
data = data[data.property_type != "store"]
```

In [4]:
```
def property_type_to_num(x):
    if x == "apartment":
        return 0
    if x == "house":
        return 1
    if x == "PH":
        return 2
```

In [5]:
```
data['property_type'] = data['property_type'].apply(lambda x: property_type_to_num(x))
```

In [6]:
```
rfr = RandomForestRegressor(n_estimators=20, min_samples_split=3, random_state=0)
```

In [7]:
```
rfr.fit(data[['expenses', 'floor', 'lat', 'lon', 'property_type', \
              'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']], \
        data[['price_aprox_usd']].values.ravel())
```

Out[7]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=3,
           min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

In [8]:
```
predictions = rfr.predict(test[['expenses', 'floor', 'lat', 'lon', 'property_type',\
                                'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']])
```

In [9]:
```
test['price_usd'] = pd.Series(predictions)
```

In [10]:
```
submit = test[['id', 'price_usd']]
```

In [11]:
```
submit.to_csv("properati_dataset_sample_submision.csv", index = False)
```

AdaBoost

Con este algoritmo hicimos varias pruebas cambiando el estimador base. Obtuvimos los siguientes score:

1) AdaBoost con KNN: 973395588613.77500 (Empeoró respecto a KNN)

2) AdaBoost sin estimador base: 964367202699.85600 (Mejoró en comparación a todos los anteriores)

3) AdaBoost con Random Forest: 963314256967.69800

4) AdaBoost con Bagging Regressor cuyo estimador base era Gradient Boosting: 960838188443.48500

5) AdaBoost con Gradient Boosting: 957297203685.38900

Este ultimo fue el algoritmo con el que obtuvimos el mejor score. El código que sigue es de este algoritmo.

In [105]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

In [106]:
```
data = pd.read_csv("datosSinNan.csv")
test = pd.read_csv("datatestSinNan.csv")
```

In [107]:
```
data = data[data.property_type != "store"]
```

In [108]:
```
def property_type_to_num(x):
    if x == "apartment":
        return 0
    if x == "house":
        return 1
    if x == "PH":
        return 2
```

In [109]:
```
data['property_type'] = data['property_type'].apply(lambda x: property_type_to_num(x))
```

```
In [111]: ab = AdaBoostRegressor(base_estimator=GradientBoostingRegressor(n_estimators = 2000, \
                                                                          max_depth = 5, min_samples_split = 3), \
                                 n_estimators=50, learning_rate=0.1, loss='exponential', random_state=3)
```

```
In [112]: ab.fit(data[['expenses', 'floor', 'lat', 'lon', 'property_type', \
                       'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']], \
                  data[['price_aprox_usd']].values.ravel())
```

```
Out[112]: AdaBoostRegressor(base_estimator=GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                       learning_rate=0.1, loss='ls', max_depth=5, max_features=None,
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=3, min_weight_fraction_leaf=0.0,
                       n_estimators=2000, presort='auto', random_state=None,
                       subsample=1.0, verbose=0, warm_start=False),
                   learning_rate=0.1, loss='exponential', n_estimators=50,
                   random_state=3)
```

Esta linea nos permite guardar el estimador ya entrenado para después poder cargarlo y predecir los valores.

```
In [113]: from sklearn.externals import joblib

          joblib.dump(ab, 'adaboost5000.pkl')
```

```
Out[113]: ['adaboost5000.pkl']
```

```
In [114]: predictions = ab.predict(test[['expenses', 'floor', 'lat', 'lon', 'property_type',\
                                         'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']])
```

```
In [117]: test['price_usd'] = pd.Series(predictions)
```

```
In [118]: submit = test[['id', 'price_usd']]
```

```
In [119]: submit.to_csv("properati_dataset_sample_submision2.csv", index = False)
```

Código para cargar el estimador

Utilizamos la libreria joblib para no tener que entrenar al estimador cada vez que queremos predecir, asi a la hora de presentar el trabajo podemos predecir rapidamente.

In [2]:
```
from sklearn.externals import joblib

ab = joblib.load('adaboost.pkl')
```

In [3]:
```
import pandas as pd

test = pd.read_csv("datatestSinNan.csv")
```

In [4]:
```
predictions = ab.predict(test[['expenses', 'floor', 'lat', 'lon', 'property_type',\
                                'rooms', 'surface_covered_in_m2', 'surface_total_in_m2']])
```

In [5]:
```
test['price_usd'] = pd.Series(predictions)
```

In [6]:
```
submit = test[['id', 'price_usd']]
```

In [7]:
```
submit.to_csv("properati_dataset_sample_submision3.csv", index = False)
```