 # Integrantes

- Alejandro Ruiz Luna
- Robinson Álvarez Patiño
- Ronald Ortíz García

```
!wget --no-cache -O init.py -q https://raw.githubusercontent.com/UDEA-Esp-Analitica-y-Ciencia-de-Datos/EACD-01-FUNDAMENTOS/ma
import init; init.init(force_download=False);
```

Precios de Casas

El objetivo de este taller es realizar un análisis exploratorio de un dataset. El dataset no llega limpio, el proceso de limpieza se encuentra implementado. Después de este proceso de limpieza se debe llevar a cabo el análisis exploratorio.

```
from collections import Counter, defaultdict

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Entendiendo y limpiando el dataset

```
!cat local/data/houseprices_description.txt
```

```
    ScreenPorch: Screen porch area in square feet

    PoolArea: Pool area in square feet

    PoolQC: Pool quality
```

```
       Ex       Excellent
       Gd       Good
       TA       Average/Typical
       Fa       Fair
       NA       No Pool

Fence: Fence quality

       GdPrv    Good Privacy
       MnPrv    Minimum Privacy
       GdWo     Good Wood
       MnWw     Minimum Wood/Wire
       NA       No Fence

MiscFeature: Miscellaneous feature not covered in other categories

       Elev     Elevator
       Gar2     2nd Garage (if not described in garage section)
       Othr     Other
       Shed     Shed (over 100 SF)
       TenC     Tennis Court
       NA       None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

       WD       Warranty Deed - Conventional
       CWD      Warranty Deed - Cash
       VWD      Warranty Deed - VA Loan
       New      Home just constructed and sold
       COD      Court Officer Deed/Estate

       Con      Contract 15% Down payment regular terms
       ConLw    Contract Low Down payment and low interest
       ConLI    Contract Low Interest
       ConLD    Contract Low Down
       Oth      Other
```

```
SaleCondition: Condition of sale

       Normal   Normal Sale
       Abnorml  Abnormal Sale -  trade, foreclosure, short sale
       AdjLand  Adjoining Land Purchase
       Alloca   Allocation - two linked properties with separate deeds, typically condo with a garage unit
       Family   Sale between family members
       Partial  Home was not completed when last assessed (associated with New Homes)
```

La descripcion de cada variable puede verse ejecutando la siguiente celda

Ahora carguemos los datos y hagamos una breve exploración

```
df = pd.read_csv("local/data/houseprices.csv")
df.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	Pool
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	Na
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	Na
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	Na
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	Na
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	Na

5 rows × 81 columns

```
df.info()
```

```
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
```

```
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
67  OpenPorchSF    1460 non-null   int64
68  EnclosedPorch  1460 non-null   int64
69  3SsnPorch      1460 non-null   int64
70  ScreenPorch    1460 non-null   int64
```

```
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

Podemos ver que hay una gran cantidad de valores nulos en algunas de las variables. Alguien, algo despistado, podría sugerir simplemente eliminar esas variables; sin embargo, la descripción de las variables que observamos anteriormente, nos permite entender la razón de ser de estos valores nulos y tratarlos de una manera inteligente. Por ejemplo, consideremos la variable `PoolQC` la cual nos muestra únicamente 7 valores no nulos de los 1460 registros que tenemos en total. Su descripción dice:

```
PoolQC: Pool quality

       Ex    Excellent
       Gd    Good
       TA    Average/Typical
       Fa    Fair
       NA    No Pool
```

Esta variable hace referencia a la calidad de la piscina en la casa y vemos que `NA` significa que no tiene piscina, lo cual posiblemente tiene mucho impacto en el precio de una casa (pregúntese, ¿estaría dispuesto a pagar más por una casa que tenga piscina?). Además, vemos que existe la variable `PoolArea`, la cual almacena el area de la piscina en ft^2, la cual no tiene valores nulos. Dado esto, los valores nulos de la variable `PoolQC` deben corresponder a casos en los que la variable `PoolArea` es cero; validemos esto.

```
num_total_nulls = df["PoolQC"].isna().sum()
```

```
num_nulls_when_poolarea_is_zero = df[df["PoolArea"] == 0]["PoolQC"].isna().sum()
assert num_nulls_when_poolarea_is_zero == num_total_nulls

num_nulls_when_poolarea_is_not_zero = df[df["PoolArea"] != 0]["PoolQC"].isna().sum()
assert num_nulls_when_poolarea_is_not_zero == 0
```

Concluimos que se cumple que todos los valores nulos de la variable `PoolQC` corresponden a casos en los que no hay piscina; por lo tanto, vamos a reemplzar los valores nulos por otro valor que podamos usar en nuestros modelos.

```
df["PoolQC"] = df["PoolQC"].fillna("NP")
```

Esta misma lógica debemos usarla a la hora de tratar el resto de las variables con valores nulos de este dataset. Escribiremos algún razonamiento addiconal únicamente cuando haga falta

```
num_total_nulls = df["MiscFeature"].isna().sum()
num_nulls_when_miscval_is_zero = df[df["MiscVal"] == 0]["MiscFeature"].isna().sum()
num_nulls_when_miscval_is_not_zero = df[df["MiscVal"] != 0]["MiscFeature"].isna().sum()
assert num_nulls_when_miscval_is_zero == num_total_nulls
assert num_nulls_when_miscval_is_not_zero == 0
df["MiscFeature"] = df["MiscFeature"].fillna("No MF")
```

```
num_total_nulls = df["FireplaceQu"].isna().sum()
num_nulls_when_fireplaces_is_zero = df[df["Fireplaces"] == 0]["FireplaceQu"].isna().sum()
num_nulls_when_fireplaces_is_not_zero = df[df["Fireplaces"] != 0]["FireplaceQu"].isna().sum()
assert num_nulls_when_fireplaces_is_zero == num_total_nulls
assert num_nulls_when_fireplaces_is_not_zero == 0
df["FireplaceQu"] = df["FireplaceQu"].fillna("No FP")
```

```
num_area_zeros = (df["GarageArea"] == 0).sum()
num_cars_zeros = (df["GarageCars"] == 0).sum()
num_both_zeros = ((df["GarageArea"] == 0) & (df["GarageCars"] == 0.0)).sum()
assert num_both_zeros == num_area_zeros == num_cars_zeros
for colname in ["GarageType", "GarageFinish", "GarageQual", "GarageCond"]:
```

```
    num_total_nulls = df[colname].isna().sum()
    num_nulls_when_area_and_cars_capacity_is_zero = df[(df["GarageArea"] == 0.0) & (df["GarageCars"] == 0.0)][colname].isna(
    num_nulls_when_area_and_cars_capacity_is_not_zero = df[(df["GarageArea"] != 0.0) & (df["GarageCars"] != 0.0)][colname].i
    assert num_total_nulls == num_nulls_when_area_and_cars_capacity_is_zero
    assert num_nulls_when_area_and_cars_capacity_is_not_zero == 0
    df[colname] = df[colname].fillna("No Ga")
```

Para la variable `GarageYrBlt` debemos ser más cuidadosos, ya que son números y no strings. Esta variable nos dice el año en que fue construido el garaje y, según lo visto con otras variables relacionadas al garage, los valores nulos corresponden a casos en los que no hay garage. En este caso, vamos a imputar esa variable con un año posterior a la fecha de venta. Esta aproximación podría no funcionar muy bien con modelos lineales, o cuando escalamos los datos, pero no se me ocurre otra!

```
num_total_nulls = df["GarageYrBlt"].isna().sum()
num_nulls_when_area_and_cars_is_zero = df[(df["GarageArea"] == 0.0) & (df["GarageCars"] == 0.0)]["GarageYrBlt"].isna().sum()
num_nulls_when_area_and_cars_is_not_zero = df[(df["GarageArea"] != 0.0) & (df["GarageCars"] != 0.0)]["GarageYrBlt"].isna().s
assert num_nulls_when_area_and_cars_is_zero == num_total_nulls
assert num_nulls_when_area_and_cars_is_not_zero == 0
df["GarageYrBlt"].where(~df["GarageYrBlt"].isna(), other=df["YrSold"] + 1, inplace=True)
```

```
 LotFrontage: Linear feet of street connected to property)
```

Valores nulos en esta variable podrían ser ocasionados porque sencillamente no hay conexión de la calle a la propiedad, es decir, que esa longitud medida por esta variable es igual a 0. Podemos asumir que este es el caso únicamente si no hay otro 0 en los valores que ha tomado esta variable, de otra manera ¿por qué algunos tendrían 0 y otros nulos?

```
assert (df["LotFrontage"] == 0).sum() == 0
df["LotFrontage"].fillna(0, inplace=True)
```

```
df["Alley"].fillna("NA", inplace=True)
df["Fence"].fillna("NF", inplace=True)
```

```
 MasVnrType: Masonry veneer type

        BrkCmn    Brick Common
        BrkFace    Brick Face
        CBlock    Cinder Block
        None    None
        Stone    Stone

 MasVnrArea: Masonry veneer area in square feet
```

Una posibilidad es que los valores nulos de `MasVnrType` y `MasVnrArea` correspondan a casos en los que no hay "Masonry veneer" (chapa de albañilería). Esto sería en los casos en que la variable `MasVnrArea` sea 0, esto lo validaremos de la misma manera en que

```
(df["MasVnrArea"] == 0).sum() == df["MasVnrType"].isnull().sum()
```

```
     False
```

Acá la situación es diferente, el error anterior nos dice que los casos para los cuales no tenemos área son distintos de los que el tipo es nulo, entonces nos toca inspeccionar más en detalle. Sabemos que hay 8 valores nulos en cada una de las dos variables, miremos si corresponden a los mismos registros:

```
np.logical_and(df["MasVnrType"].isnull().values, df["MasVnrArea"].isnull().values).sum()
```

```
     8
```

En efecto, dado que hay 8 registros para los cuales las dos variables tienen valores nulos, podemos decir que son nulos en ambas partes. A falta de información, vamos a decidir eliminar esos registros por completo, no debe ser muy grave, pues apenas son 8 filas en todo el dataset.

```
df = df.dropna(subset=["MasVnrType", "MasVnrArea"])
```

Ahora miremos los valores de cada una

```
df["MasVnrType"].value_counts()
```

```
None       864
BrkFace    445
Stone      128
BrkCmn      15
Name: MasVnrType, dtype: int64
```

```
df["MasVnrArea"].value_counts()
```

```
0.0      861
180.0      8
72.0       8
108.0      8
120.0      7
         ...
562.0      1
89.0       1
921.0      1
762.0      1
119.0      1
Name: MasVnrArea, Length: 327, dtype: int64
```

Acá podemos observar lo siguiente:

- Hay 864 registros con `MasVnrType="None"`
- Hay 861 registros con `MasVnrArea=0`

Eso quiere decir que hay algunos registros que deberían tener un área de 0 y no es así. Vamos a mirar en detalle cuáles son:

```
df[(df["MasVnrType"] == "None") & (df["MasVnrArea"] != 0.0)]
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea
624	625	60	RL	80.0	10400	Pave	NA	Reg	Lvl	AllPub	...	0
773	774	20	RL	70.0	10150	Pave	NA	Reg	Lvl	AllPub	...	0
1230	1231	90	RL	0.0	18890	Pave	NA	IR1	Lvl	AllPub	...	0
1300	1301	60	RL	0.0	10762	Pave	NA	IR1	Lvl	AllPub	...	0
1334	1335	160	RM	24.0	2368	Pave	NA	Reg	Lvl	AllPub	...	0

5 rows × 81 columns

Francamente yo no soy un experto en casas, por lo que no tengo forma de deducir a qué se debe esa incoherencia en los datos. Por lo anterior, simplemente voy a eliminar esas filas, ya que tengo dudas de la veracidad de esos datos.

```
df = df[~((df["MasVnrType"] == "None") & (df["MasVnrArea"] != 0.0))]
```

La variable `Electrical` tampoco nos ofrece una forma de recuperar esos valores nulos, por lo que también vamos a eliminar ese registro. Nótese que otra opción podría ser reemplazarlo con el valor más común en la misma variable, dado que esta es categórica.

```
df.dropna(subset=["Electrical"], inplace=True)
```

```
df.info()
```

```
 25  MasVnrType     1440 non-null   object
 26  MasVnrArea     1446 non-null   float64
 27  ExterQual      1446 non-null   object
 28  ExterCond      1446 non-null   object
 29  Foundation     1446 non-null   object
 30  BsmtQual       1409 non-null   object
 31  BsmtCond       1409 non-null   object
 32  BsmtExposure   1408 non-null   object
 33  BsmtFinType1   1409 non-null   object
 34  BsmtFinSF1     1446 non-null   int64
 35  BsmtFinType2   1408 non-null   object
```

```
36  BsmtFinSF2     1446 non-null   int64
37  BsmtUnfSF      1446 non-null   int64
38  TotalBsmtSF    1446 non-null   int64
39  Heating        1446 non-null   object
40  HeatingQC      1446 non-null   object
41  CentralAir     1446 non-null   object
42  Electrical     1446 non-null   object
43  1stFlrSF       1446 non-null   int64
44  2ndFlrSF       1446 non-null   int64
45  LowQualFinSF   1446 non-null   int64
46  GrLivArea      1446 non-null   int64
47  BsmtFullBath   1446 non-null   int64
48  BsmtHalfBath   1446 non-null   int64
49  FullBath       1446 non-null   int64
50  HalfBath       1446 non-null   int64
51  BedroomAbvGr   1446 non-null   int64
52  KitchenAbvGr   1446 non-null   int64
53  KitchenQual    1446 non-null   object
54  TotRmsAbvGrd   1446 non-null   int64
55  Functional     1446 non-null   object
56  Fireplaces     1446 non-null   int64
57  FireplaceQu    1446 non-null   object
58  GarageType     1446 non-null   object
59  GarageYrBlt    1446 non-null   float64
60  GarageFinish   1446 non-null   object
61  GarageCars     1446 non-null   int64
62  GarageArea     1446 non-null   int64
63  GarageQual     1446 non-null   object
64  GarageCond     1446 non-null   object
65  PavedDrive     1446 non-null   object
66  WoodDeckSF     1446 non-null   int64
67  OpenPorchSF    1446 non-null   int64
68  EnclosedPorch  1446 non-null   int64
69  3SsnPorch      1446 non-null   int64
70  ScreenPorch    1446 non-null   int64
71  PoolArea       1446 non-null   int64
72  PoolQC         1446 non-null   object
73  Fence          1446 non-null   object
74  MiscFeature    1446 non-null   object
75  MiscVal        1446 non-null   int64
76  MoSold         1446 non-null   int64
77  YrSold         1446 non-null   int64
```

```
 78  SaleType       1446 non-null   object
 79  SaleCondition  1446 non-null   object
 80  SalePrice      1446 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 926.3+ KB
```

Ahora miremos las variables relacionadas con el basement:

```
BsmtQual: Evaluates the height of the basement

       Ex    Excellent (100+ inches)
       Gd    Good (90-99 inches)
       TA    Typical (80-89 inches)
       Fa    Fair (70-79 inches)
       Po    Poor (<70 inches
       NA    No Basement

BsmtCond: Evaluates the general condition of the basement

       Ex    Excellent
       Gd    Good
       TA    Typical - slight dampness allowed
       Fa    Fair - dampness or some cracking or settling
       Po    Poor - Severe cracking, settling, or wetness
       NA    No Basement

BsmtExposure: Refers to walkout or garden level walls

       Gd    Good Exposure
       Av    Average Exposure (split levels or foyers typically score average or above)
       Mn    Mimimum Exposure
       No    No Exposure
```

NA	No Basement

BsmtFinType1: Rating of basement finished area

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF1: Type 1 finished square feet

BsmtFinType2: Rating of basement finished area (if multiple types)

GLQ	Good Living Quarters
ALQ	Average Living Quarters
BLQ	Below Average Living Quarters
Rec	Average Rec Room
LwQ	Low Quality
Unf	Unfinshed
NA	No Basement

BsmtFinSF2: Type 2 finished square feet

BsmtUnfSF: Unfinished square feet of basement area

TotalBsmtSF: Total square feet of basement area

Dado que los valores nulos en varias de estas variables corresponden a `No Basement`, ahora miremos si los registros nulos en algunas corresponden a los registros nulos en todas. Primero, dado que ya hemos eliminado varios registros, vamos a ver cuántos valores nulos hay en esas variables

```
colnames = ["BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1",
    "BsmtFinSF1", "BsmtFinType2", "BsmtFinSF2", "BsmtUnfSF", "TotalBsmtSF"]
for c in colnames:
    print(f"{c} has {df[c].isnull().sum()} null values")
```

```
 BsmtQual has 37 null values
 BsmtCond has 37 null values
 BsmtExposure has 38 null values
 BsmtFinType1 has 37 null values
 BsmtFinSF1 has 0 null values
 BsmtFinType2 has 38 null values
 BsmtFinSF2 has 0 null values
 BsmtUnfSF has 0 null values
 TotalBsmtSF has 0 null values
```

```
df["TotalBsmtSF"].value_counts()
```

```
 0       37
 864     35
 672     17
 912     14
 1040    14
         ..
 1581     1
 707      1
 611      1
 1452     1
 1542     1
 Name: TotalBsmtSF, Length: 716, dtype: int64
```

Acá vemos que algunas tienen más variables nulas que otras, lo cual es confuso porque en cualquier caso los valores nulos deberían significar que no hay basement. En este caso vamos a reemplazar los valores en los que los valores nulos sean en todas las variables no numéricas pero los que sobren los eliminaremos.

```
colnames = ["BsmtQual", "BsmtCond", "BsmtExposure", "BsmtFinType1", "BsmtFinType2"]
cond = ~(df["BsmtQual"].isna() & df["BsmtCond"].isna() & df["BsmtExposure"].isna() & df["BsmtFinType1"].isna() & df["BsmtFin
for c in colnames:
    df[c].where(cond, other="NB", inplace=True)
```

```
df.dropna(inplace=True)
print(f"Number of null values {df.isna().sum().sum()} in a dataframe of shape {df.shape}")
```

```
Number of null values 0 in a dataframe of shape (1444, 81)
```

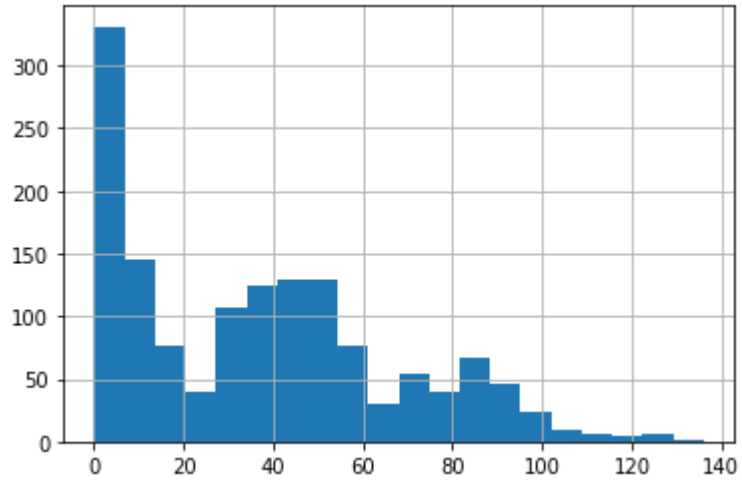
Análisis exploratorio de datos

En esta sección se dejarán preguntas que deben ser respondidas utilizando los datos.

¿Qué tan viejas son las casas?

```
df["HouseAge"] = df["YrSold"] - df["YearBuilt"]
df["HouseAge"].hist(bins=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f033d76fb10>
```

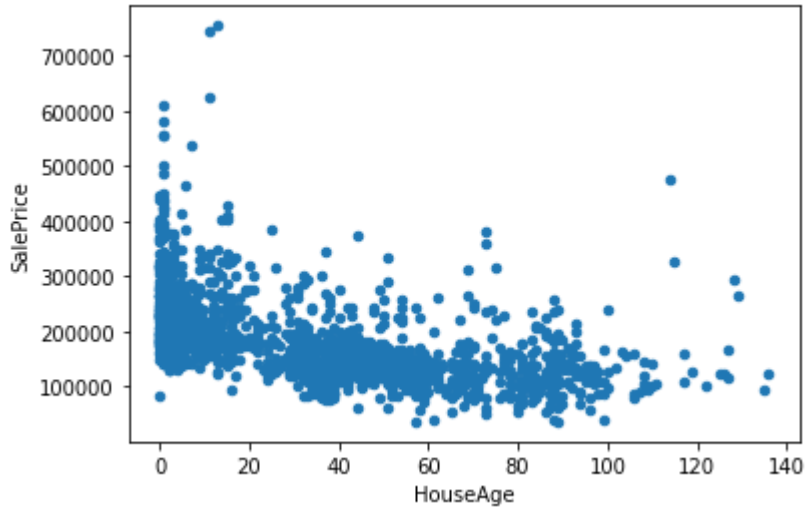


¿Cómo se relaciona el precio con la edad de la casa? Graficando las dos variables en un Scatter podemos notar que existe una relacion inversa entre la edad de casa y el precio. Entre mas reciente es la construccion mayor es el precio.

```
data = pd.concat([df["SalePrice"],df["HouseAge"]],axis=1)   # Reducimos el dataframe a las columnas precio y edad de la casa
```

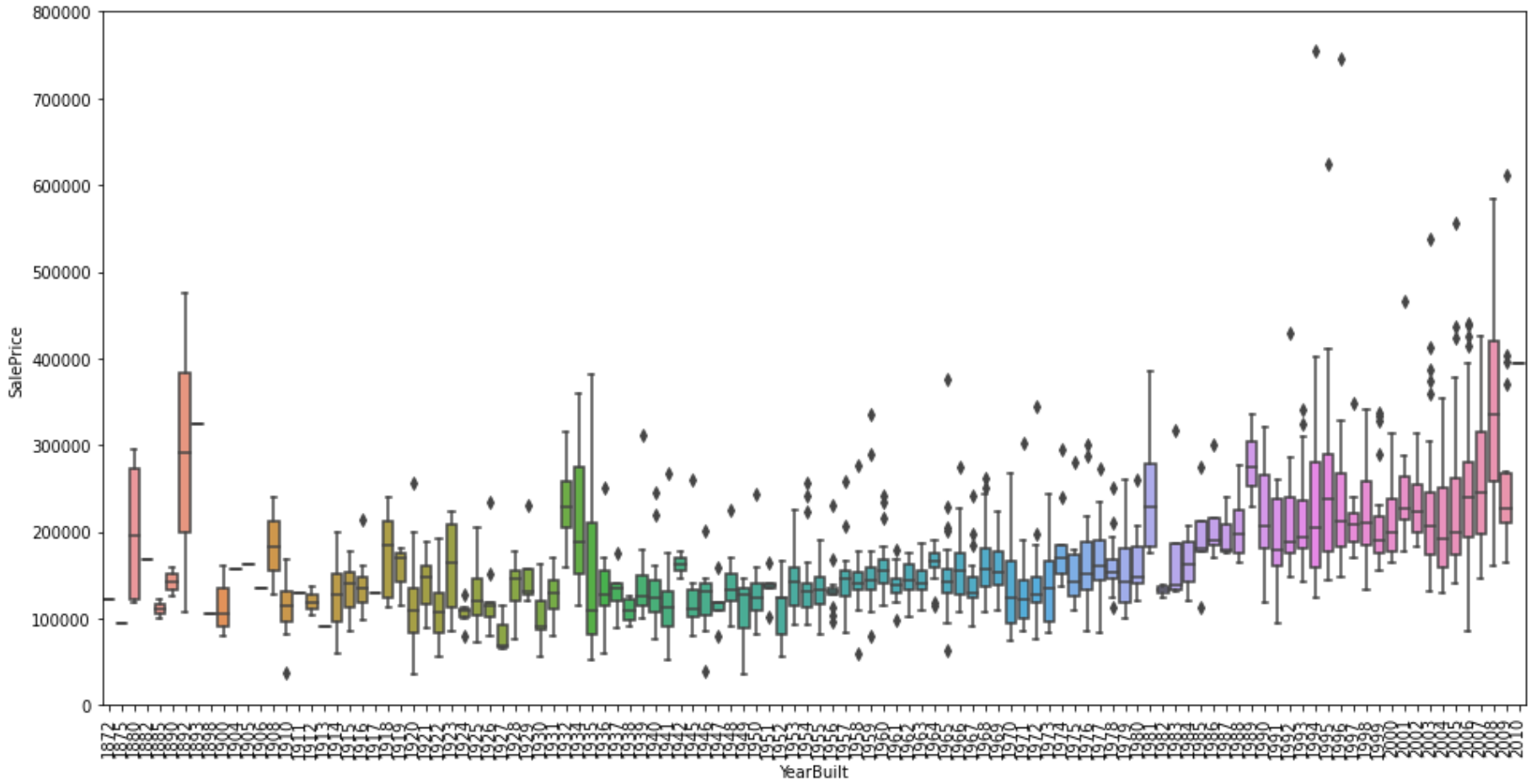
```
data.plot.scatter(x="HouseAge",y="SalePrice",ylim=(0.8000))   # Graficamos
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f033ccfd050>
```



Para entender porque la relacion entre precio y año no es "perfectamente" inversa, vamos graficar en un boxplot año a año con la dispersion del precio versus el año de construccion.

```
var = 'YearBuilt'
data = pd.concat([df['SalePrice'], df[var]], axis=1) #  Reducimos el dataframe a las columnas precio y edad de la casa
f, ax = plt.subplots(figsize=(16, 8))   #ajustamos el tamaño del grafico
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
plt.xticks(rotation=90);
```



Esta grafica nos muestra unos outlayer que representan casas que a pesar de tener mucho tiempo de construccion, se han vendido por precios muy altos, lo cual explica el porque no se ve una relacion perfectamente inversa en todos los casos.

¿Cuál es el barrio más pobre?

```
data = pd.concat([df["SalePrice"],df["Neighborhood"],df['Utilities'],df['MSZoning'],df["OverallCond"]],axis=1)
data.groupby(['Neighborhood']).size()
```

```
Neighborhood
Blmngtn     17
```

```
Blueste      2
BrDale      15
BrkSide     58
ClearCr     28
CollgCr    148
Crawfor     50
Edwards    100
Gilbert     77
IDOTRR      37
MeadowV     17
Mitchel     49
NAmes      224
NPkVill      9
NWAmes      72
NoRidge     41
NridgHt     75
OldTown    113
SWISU       25
Sawyer      73
SawyerW     58
Somerst     83
StoneBr     25
Timber      37
Veenker     11
dtype: int64
```

```
df["SalePrice"].describe()
```

```
count      1444.000000
mean     180602.092798
std       79414.474218
min       34900.000000
25%      129900.000000
50%      162000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

Para identificar el barrio mas pobre iniciamos reduciendo nuestro dataframe a las variables que consideramos que se relacionan a la pobreza, para este caso serían "Utilities","SalePrice", "Neighborhood" y "OverallCond"

La variable "Utilities" Cateregoriza la cantidad de servicios que tiene una casa, por otro lado, "OverallCond" Califica el estado general

```
data = pd.concat([df["SalePrice"],df["Neighborhood"],df['Utilities'],df["OverallCond"]],axis=1)
data.groupby(['Utilities','Neighborhood']).mean().sort_values('Utilities',ascending = False).head(10)
```

		SalePrice	OverallCond
Utilities	**Neighborhood**		
NoSeWa	**Timber**	137500.000000	6.000000
AllPub	**Blueste**	137500.000000	6.000000
	Veenker	238772.727273	6.272727
	Timber	247233.416667	5.111111
	StoneBr	310499.000000	5.000000
	Somerst	226443.566265	5.024096
	SawyerW	186584.344828	5.155172
	Sawyer	136064.273973	5.821918
	SWISU	142591.360000	5.920000
	OldTown	128225.300885	6.353982

Luego de aplicar los criterios para segmentar nuestro conjunto de datos,concluimos que el barrio mas pobre es **Timber** dado que la mayoria de las casa no tienen todos los servicios, el precio de las casas es mas bajo que el promedio y además el estado promedio de las casas esta deteriorado

¿Cuál es el barrio más cercano a vías férreas?

```
df["Condition1"].describe()
```

```
count     1444
unique       9
top       Norm
freq      1246
Name: Condition1, dtype: object
```

Para identificar el barrio mas próximo a las vías férreas, iniciamos reduciendo nuestro dataframe a las variables que consideramos que se relacionan a la proximidad, para este caso serían "Condition1", "Condition2" y "Neighborhood"

Considerando que el criterio para determinar la proximidad a las vías ferreas son aquellos que cumplan con alguno/s de estos valores 'RRNn','RRAn','RRNe','RRAe', sabiendo esto, segmentamos los datos a los barrios que cumplan con estos valores

```
near_train = pd.concat([df["Id"],df["Neighborhood"],df['Condition1'],df['Condition2']],axis=1)
data = near_train [near_train.Condition1.isin(['RRNn','RRAn','RRNe','RRAe']) | near_train.Condition2.isin(['RRNn','RRAn','RRI
data.groupby("Neighborhood").size()
```

```
Neighborhood
BrkSide    9
Gilbert    9
IDOTRR     2
NWAmes     7
OldTown    2
Sawyer     6
SawyerW    6
Somerst    6
dtype: int64
```

Notamos que para dos **BrkSide, Gilbert** barrios se cumplen que 9 de sus casas se encuentran proximas a las vías ferreas

¿Cuál es la cobertura más común en las casas que se encuentran en el top 10% en precio?

Para identificar cuales son las casas que se encuentran en el top 10% en precio creamos un nuevo DataFrame "top_10_percent" que contiene el 10% de la columna ordenada de "SalePrice". Luego de eso asociamos las variables de cobertura que para este caso son

"Exterior1st" y "Exterior2nd".

Por ultimo hacemos el conteo de cada criterio de las variables anteriores

```
top_10_percent= df[df["SalePrice"].sort_values(ascending= False) > df.SalePrice.quantile(0.9)]
cobertur=top_10_percent[["Exterior1st","Exterior2nd"]].value_counts()
cobertur
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: UserWarning: Boolean Series key will be reindexed to ma
  """Entry point for launching an IPython kernel.
Exterior1st  Exterior2nd
VinylSd      VinylSd        83
CemntBd      CmentBd        19
MetalSd      MetalSd         9
Wd Sdng      Wd Sdng         6
BrkFace      BrkFace         5
HdBoard      HdBoard         5
Plywood      Plywood         4
Wd Sdng      ImStucc         2
WdShing      Wd Shng         2
BrkFace      Wd Sdng         2
             HdBoard         1
Stucco       CmentBd         1
             Stucco          1
VinylSd      ImStucc         1
             Other           1
Plywood      Wd Sdng         1
CemntBd      Wd Shng         1
Stone        HdBoard         1
dtype: int64
```

Podemos notar que, para el top 10% de los precios de las casas, la cobertura mas común es la de ***VinylSd*** usada por 83 casas

¿En qué barrio hay mayor desigualdad?. Vamos a considerar "desigualdad" como la diferencia en los precios de venta de las casa.

- Vamos a filtrar Barrios por encima de 10 viviendas (con el objetivo de tomar barrios representativos)
- Buscaremos la Desviacion estandar de sus precios como elemento para determinar la desigualdad.

```
df_ori_rap = df.copy()  # copia del dataframe


group_x_neigh = df_ori_rap.groupby('Neighborhood').agg(np.count_nonzero)


filtro_barrios_grandes = group_x_neigh['Id'] >= 10   # Filtro todos aquellos que no tengamos mas de 10 viviendas


std = df_ori_rap.groupby('Neighborhood').agg(np.std)    # saco la desviacion estandar de todos
std['SalePrice'].sort_values()
```

```
Neighborhood
NPkVill      9377.314529
BrDale      13710.708363
Blueste     19091.883092
Sawyer      21595.877724
MeadowV     23491.049610
Blmngtn     30393.229219
SWISU       32622.917679
NAmes       33082.594869
IDOTRR      33376.710117
Gilbert     36161.352410
Mitchel     36486.625334
NWAmes      37323.965554
BrkSide     40348.689270
Edwards     43208.616459
ClearCr     50231.538993
CollgCr     51649.462283
OldTown     52650.583185
SawyerW     56137.614907
Somerst     56874.974640
Timber      64516.423327
Crawfor     69550.599180
Veenker     72369.317959
NridgHt     96058.334691
StoneBr    112969.676640
```

```
     NoRidge    121412.658640
     Name: SalePrice, dtype: float64
```

```
std.loc[filtro_barrios_grandes, 'SalePrice'].max()   #aplico el fitro y selecciono el barrio que tiene la mas alta desviacio
```

```
     121412.65864036564
```

Para nosotros, teniendo en cuenta los barrios donde tenemos mas datos (mas de 10 ventas), el barrio que mas desigualdad presenta en funcion de la desviacion estandar de los precios de sus casas es: **NoRidge** con una desviacion de US 121.412 en los precios de sus viviendas.

¿En qué año hubo más movimiento del mercado inmobiliario?

Para eso agrupamos todas las transacciones que se hayan hecho por cada año en el DataSet y luego lo ordenamos para encontrar el año con mayor moviemiento. La respuesta es : **2009** con 334 ventas

```
resultado = df.groupby('YrSold').agg(np.count_nonzero).sort_values('Id', ascending=False)   # agrupacion por año
resultado = resultado.rename(columns={'Id':'Houses sold'})       # cambio nombre de columnas
resultado['Houses sold']
```

```
     YrSold
     2009    334
     2007    325
     2006    311
     2008    300
     2010    174
     Name: Houses sold, dtype: int64
```

¿Cuáles son los 2 barrios con mayor industria cerca?

```
df["MSZoning"].value_counts()
```

```
     RL         1139
     RM          217
     FV           62
```

```
RH           16
C (all)      10
Name: MSZoning, dtype: int64
```

Partiendo de la clasificación de la zona de venta, evidenciamos que no existe ninguna casa que cumpla con el criterio de "I" = Industrial, o casa dentro de una zona industrial.

En vista de que no existe algún/a otra variable que de información acerca de si un barrio se encuentra cerca o no de una zona industrial. Concluimos que no hay ventas de casa en barrios cerca a zonas industriales.

¿Cuáles son los 2 barrios con mayor comercio cerca? Vamos a considerar los barrios que se han vendido en zona comercial, para mirar esto tenemos en cuenta la variable MSZoning de cada barrio con el fin de encontrar aquellas ventas que se han realizado en barrios comerciales.

```
data2 = pd.concat([df["MSZoning"],df["Neighborhood"]],axis=1)
data2.plot.scatter(x="MSZoning",y="Neighborhood",ylim=(0.8000), figsize=(12, 8))  # utilizamos Scatter para identificar las v
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f033bed4590>
```


```
df_ori_rap = df[(df["MSZoning"] == "C (all)")]
df_ori_rap.Neighborhood   # Corroborando la informacion en el DataSet
```

```
30        IDOTRR
88        IDOTRR
93       OldTown
495       IDOTRR
557       IDOTRR
711       IDOTRR
812       IDOTRR
916       IDOTRR
1061      IDOTRR
1279      IDOTRR
Name: Neighborhood, dtype: object
```

Dado los resultados anteriores las casas que se han vendido en la zonas comerciales estan ubicadas en los barrios: **IDOTRR** (9 viviendas) Y **OldTown** (una sola vivienda)

Colab paid products - Cancel contracts here

0s completed at 2:51 PM