Import packages

[ ] ↳ *1 cell hidden*

Importing data

```
##Uploading the data
house_df=pd.read_csv('/content/drive/MyDrive/EnhanceIT/Aldo Cao Romero - raw_house_data.csv -
##Shape of the Data
print(house_df.shape)
##Observing the first 5 rows
house_df.head(5)
```

```
(5000, 16)
```

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	taxes	year_buil
0	21530491	5300000.0	85637	-1.103.782	31.356.362	2154.00	5272.00	194
1	21529082	4200000.0	85646	-111.045.371	31.594.213	1707.00	10422.36	199
2	3054672	4200000.0	85646	-111.040.707	31.594.844	1707.00	10482.00	199
3	21919321	4500000.0	85646	-111.035.925	31.645.878	636.67	8418.58	193
4	21306357	3411450.0	85750	-110.813.768	32.285.162	3.21	15393.00	199

```
##Identifying data types
house_df.dtypes
```

```
MLS                 int64
sold_price        float64
zipcode             int64
```

```
longitude            object
latitude             object
lot_acres           float64
taxes               float64
year_built            int64
bedrooms              int64
bathrooms            object
sqrt_ft              object
garage               object
kitchen_features     object
fireplaces          float64
floor_covering       object
HOA                  object
dtype: object
```

Locating None values

```
##We get a list of every column
columns=house_df.columns.values.tolist()
#Separate beteween cathegorical and numerical columns
numerical=[]
cathegorical=[]

for i in columns:
  if(house_df[i].dtype!="O"):
    numerical.append(i)
  else:
    cathegorical.append(i)


 for j in cathegorical:
   if(j!='longitude' and j!='latitude'):
    print(j+':')
    print( house_df[j].value_counts()["None"])
##Getting the number of None values in each categorical column
```

```
bathrooms:
6
sqrt_ft:
56
garage:
7
kitchen_features:
33
floor_covering:
1
HOA:
562
```

Correct object numerical values

```
##In this case we have garage, bathrooms, HOA and sqrt_ft as object type variables when they
#First, let's change the None values by zeros and transform the whoe columns into numbers
n=len(house_df['bathrooms'])
##Making a loop that makes all the work
#For batrooms we have
for i in range(n):
  if(house_df['bathrooms'][i]=='None'):
    house_df['bathrooms'][i]=0.0
house_df['bathrooms']=house_df['bathrooms'].astype(float)

#Sqrt_ft
for i in range(n):
  if(house_df['sqrt_ft'][i]=='None'):
    house_df['sqrt_ft'][i]=0.0
house_df['sqrt_ft']=house_df['sqrt_ft'].astype(float)

#Garage
for i in range(n):
  if(house_df['garage'][i]=='None'):
    house_df['garage'][i]=0
house_df['garage']=house_df['garage'].astype(float)

house_df['garage']=house_df['garage'].astype(int)

#HOA
for i in range(n):
  if(house_df['HOA'][i]=='None'):
    house_df['HOA'][i]=0
    #print(i)
  else:
   # print(i)
    vv=house_df['HOA'][i].split(',')
    house_df['HOA'][i]=house_df['HOA'][i][0]
house_df['HOA']=house_df['HOA'].astype(int)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:14: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:20: SettingWithCopyWarning
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:33: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:28: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
```

```
#Also let's note that we have zero values data that most likely should not be zero.
r=0
n=len(house_df['year_built'])
for i in range(n):
   if(house_df['year_built'][i]==0.0):
     r=r+1
print(r)
k=0
for i in range(n):
   if(house_df['sold_price'][i]==0.0):
     k=k+1
print(k)
m=0
for i in range(n):
   if(house_df['HOA'][i]==0.0):
     k=k+1
print(k)

r=0
for i in range(n):
   if(house_df['sqrt_ft'][i]==0.0):
     r=r+1
print(r)
b=0
for i in range(n):
   if(house_df['bedrooms'][i]==0.0):
     b=b+1
print(b)

t=0
for i in range(n):
   if(house_df['taxes'][i]==0.0):
     t=t+1
print(t)
a=0
for i in range(n):
   if(house_df['lot_acres'][i]==0.0):
```

```
        a=a+1
print(a)
```

```
        5
        0
        1388
        56
        0
        22
```

```
##We opbserve that we don't have None or zero values in the bedrooms columns, so, it is not p
#To solve this, let's strack some statisitics
#For the HOA variable is clear that the mode is zero. We compute it anyway
mode=house_df['year_built'].mode()[0]
for i in range(n):
   if(house_df['year_built'][i]==0.0):
     house_df['year_built'][i]=mode
#For HOA
modeHOA=house_df['HOA'].mode()[0]
for i in range(n):
   if(house_df['HOA'][i]==0.0):
     house_df['HOA'][i]=mode
```

```
        /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: SettingWithCopyWarning:
        A value is trying to be set on a copy of a slice from a DataFrame

        See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_

        /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: SettingWithCopyWarning
        A value is trying to be set on a copy of a slice from a DataFrame

        See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
          # This is added back by InteractiveShellApp.init_path()
```

```
#We analizy some information on the sqrt_ft column to see if we can be able of choosing a val
mean=house_df['sqrt_ft'].mean()
median=house_df['sqrt_ft'].median()
mode2=house_df['sqrt_ft'].mode()
print(mean)
print(median)
print(mode2)
#We observe how different is the media with the madianit seems they behave as a normal distri
```

```
        3715.9006474239914
        3524.0
        0    3674.74352
        dtype: float64
```

```
#We use the mean to fill all the zero values for sqrt_ft
for i in range(n):
```

```
    if(house_df['sqrt_ft'][i]==0.0):
      house_df['sqrt_ft'][i]=mean

      /usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:
      A value is trying to be set on a copy of a slice from a DataFrame

      See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
        This is separate from the ipykernel package so we can avoid doing imports until
```

Searching for Null/NaN values

```
#Searching for the total amount of NaN or Null values
house_df.isnull().sum()
```

```
MLS                 0
sold_price          0
zipcode             0
longitude           0
latitude            0
lot_acres          10
taxes               0
year_built          0
bedrooms            0
bathrooms           0
sqrt_ft             0
garage              0
kitchen_features    0
fireplaces         25
floor_covering      0
HOA                 0
dtype: int64
```

```
##We opbserve that we don't have Null or NaN values in the bedrooms columns, so, it is not po
```

```
#We eliminate all nun rows considerig they are less than 5% of the data
df_drop=house_df.dropna(axis=0)
df_drop.shape
```

```
(4973, 16)
```

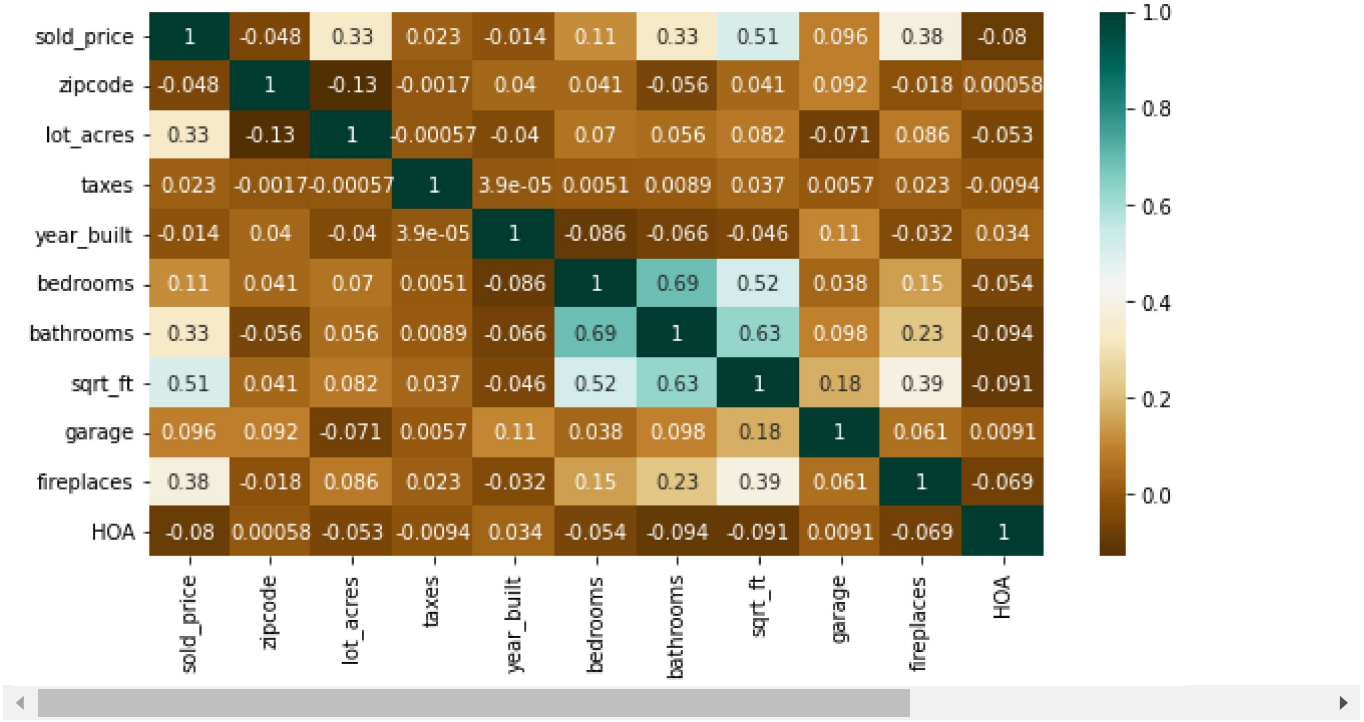
Dropping unnecesary values

[ ] ↳ *3 cells hidden*

Correlated variables

```
#Getting the correlation matrix of the data frame
plt.figure(figsize=(10,5))
c= df_drop.corr()
sns.heatmap(c,cmap="BrBG",annot=True)
c
```

	sold_price	zipcode	lot_acres	taxes	year_built	bedrooms	bathrooms
sold_price	1.000000	-0.047941	0.332954	0.023265	-0.013650	0.114050	0.326405
zipcode	-0.047941	1.000000	-0.128443	-0.001697	0.040368	0.040643	-0.056332
lot_acres	0.332954	-0.128443	1.000000	-0.000569	-0.039643	0.069806	0.055510
taxes	0.023265	-0.001697	-0.000569	1.000000	0.000039	0.005146	0.008946
year_built	-0.013650	0.040368	-0.039643	0.000039	1.000000	-0.085681	-0.066275
bedrooms	0.114050	0.040643	0.069806	0.005146	-0.085681	1.000000	0.687501
bathrooms	0.326405	-0.056332	0.055510	0.008946	-0.066275	0.687501	1.000000
sqrt_ft	0.510705	0.041285	0.081832	0.036760	-0.046220	0.523298	0.628779
garage	0.095537	0.092184	-0.070652	0.005666	0.107829	0.038145	0.097742
fireplaces	0.384310	-0.018166	0.086382	0.022548	-0.032132	0.145279	0.225633
HOA	-0.080293	0.000583	-0.052820	-0.009352	0.034318	-0.053593	-0.093886

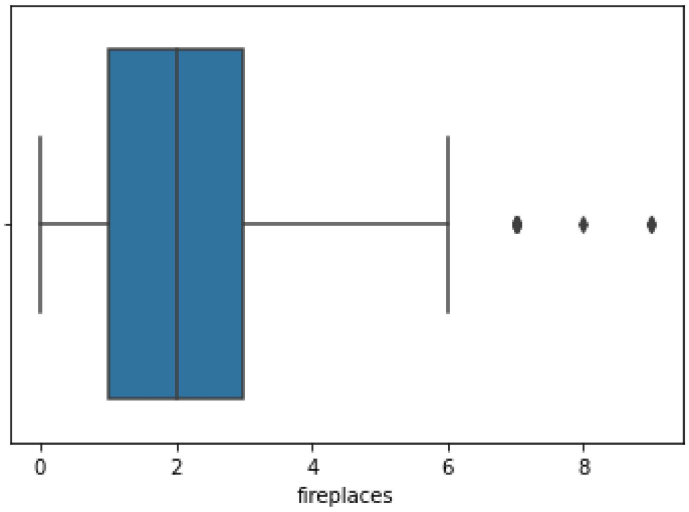



```
#There is a high correlation between the size of the house (sqrt_ft) and the sold price, as e
```

Searching for outliers

```
#Getting the box plots of all numerical variables
df=df_drop
print(df.shape)
#sns.boxplot(x=df['sqrt_ft'])
#sns.boxplot(x=df['sold_price'])
#sns.boxplot(x=df['zipcode'])
#sns.boxplot(x=df['lot_acres'])
#sns.boxplot(x=df['taxes'])
#sns.boxplot(x=df['bedrooms'])
#sns.boxplot(x=df['year_built'])
#sns.boxplot(x=df['bathrooms'])
#sns.boxplot(x=df['garage'])
sns.boxplot(x=df['fireplaces'])
plt.show()
```

```
(4973, 15)
```




```
##Getting an IQR analysis
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
sold_price    252500.00
zipcode           32.00
lot_acres          1.17
taxes           3283.00
year_built        19.00
bedrooms           1.00
bathrooms          1.00
sqrt_ft         1084.00
garage             1.00
fireplaces         2.00
```

```
HOA             2005.00
dtype: float64
```

```
df = df[~(df['fireplaces'] > 6)]
df.shape
```

```
(4961, 15)
```

```
#Based on the plots, we eliminate the significant outliers
df = df[~(df['garage'] > 15)]
df = df[~(df['bathrooms'] > 20)]
df = df[~(df['bedrooms'] > 15)]
df = df[~(df['taxes'] ==0)]
```

```
df.shape
```

```
(4930, 15)
```

```
df = df[~(df['lot_acres'] > 500)]
df = df[~(df['lot_acres']==0)]
df = df[~(df['sold_price'] > 3000000)]
df = df[~(df['sqrt_ft'] >10000)]
```

```
#We get the final shape of the data
df.shape
```

```
(4877, 15)
```

Encoding Kitchen features and floor covering

```
##We observe in the data, we have a lot of data in the kitchen features column. We need a way
#Fisrt we select the kitchen column
features=df_drop['kitchen_features']
elements=[]
coder=[]
label=[]

##We loop over the total elements in the kitchn columns to extract the unique ones.
for i in features:
  el=i.lower().split(',')
  elements=elements+el
```

```
##Then we observe that existed elements with some descriptions, so we would need to add these
for j in elements:
  if(':' in j):
    x=i.split(':')[0]
    label.append(x)
  else:
    label.append(j)
##In this way we construct our vector of unique kitchen items
label=np.unique(label)#This vector will be the base of the order of the coder's elements.

#Now,·we·create·our·vector·of·zeros·and·ones·depending·on·the·case.·
#If·the·house·has·the·item·it·would·be·one·and·zero·otherwise.·Once·again,·depending·on·the·l

for s in features:
  vec=np.zeros(len(label))
  ele=i.lower().split(',')
  for k in ele:
    if ':' in k:
      k=k.split(':')[0]
    for r in range(len(label)):
      if(label[r] ==k):
        vec[r]=1
  coder.append(vec)

##The order of the coder depends on the order of the label list
df_drop['kitchen coder']=coder


#Fisrt we select the floor_covering column
features=df_drop['floor_covering']
elements=[]
coder=[]
label=[]

##We loop over the total elements in the kitchn columns to extract the unique ones.
for i in features:
  el=i.lower().split(',')
  elements=elements+el

##Then we observe that exists elementes with the other feature, so we would need to add this
for j in elements:
  if(':' in j):
    x=i.split(':')[0]
    label.append(x)
  else:
    label.append(j)
##In this way we construct our vector of unique kitchen items
label=np.unique(label)
##print(label)

#Now, we create our vector of zeros and ones depending on the case.
#If the house has the item it would be one and zero otherwise. Once again, depending on the l
```

```
#if the house has the item it would be one and zero otherwise. once again, depending on the f

for s in features:
  vec=np.zeros(len(label))
  ele=i.lower().split(',')
  for k in ele:
    if ':' in k:
      k=k.split(':')[0]
    for r in range(len(label)):
      if(label[r] ==k):
        vec[r]=1
  coder.append(vec)

##The order of the coder depends on the order of the label list
df_drop['floor coder']=coder


df_drop.head(5)
```

longitude	latitude	lot_acres	taxes	year_built	bedrooms	bathrooms	sqrt_
-1.103.782	31.356.362	2154.00	5272.00	1941	13	10.0	1050
-111.045.371	31.594.213	1707.00	10422.36	1997	2	2.0	730
-111.040.707	31.594.844	1707.00	10482.00	1997	2	3.0	
-111.035.925	31.645.878	636.67	8418.58	1930	7	5.0	901
-110.813.768	32.285.162	3.21	15393.00	1995	4	6.0	639

✓ 0s completed at 3:06 PM