```
#pip install shap


import shap
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns


df = pd.read_csv('/content/house_data.csv')
df.shape
```

```
(5000, 16)
```

There are 5000 observations and 16 variables in our dataset. Each observation correspond to a house. Now, let's see the data.

```
df.head(5)
```

	MLS	sold_price	zipcode	longitude	latitude	lot_acres	ta
0	21530491	5300000.0	85637	-1.103.782	31.356.362	2154.00	5272
1	21529082	4200000.0	85646	-111.045.371	31.594.213	1707.00	10422
2	3054672	4200000.0	85646	-111.040.707	31.594.844	1707.00	10482
3	21919321	4500000.0	85646	-111.035.925	31.645.878	636.67	8418

```
# Count the number of NA's in each column
df.isna().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
```

```
# Now we are check the type of variables of the table
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   MLS               5000 non-null   int64  
 1   sold_price        5000 non-null   float64
 2   zipcode           5000 non-null   int64  
 3   longitude         5000 non-null   object 
 4   latitude          5000 non-null   object 
 5   lot_acres         4990 non-null   float64
 6   taxes             5000 non-null   float64
 7   year_built        5000 non-null   int64  
 8   bedrooms          5000 non-null   int64  
 9   bathrooms         5000 non-null   object 
 10  sqrt_ft           5000 non-null   object 
 11  garage            5000 non-null   object 
 12  kitchen_features  5000 non-null   object 
 13  fireplaces        4975 non-null   float64
 14  floor_covering    5000 non-null   object 
 15  HOA               5000 non-null   object 
dtypes: float64(4), int64(4), object(8)
memory usage: 625.1+ KB
```

```
def IdentifyKeyWord(x, word):
    if word in x.lower():
        return 1
    else:
        return 0

# Kitchen Features
df['KF_Dishwasher'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'dishwasher'))
df['KF_GarbageDisposal'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'garbage di
df['KF_Refrigerator'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'refrigerator'
df['KF_DoubleSink'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'double sink'))
df['KF_Microwave'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'microwave'))
df['KF_Oven'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'oven'))
```

```
df['KF_Compactor'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'compactor'))
df['KF_Freezer'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'freezer'))
df['KF_ElectricRange'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'electric ran
df['KF_Island'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'island'))
df['KF_GasRange'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'gas range'))
df['KF_Countertops'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'countertops'))
df['KF_Desk'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'desk'))
df['KF_Granite'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'granite'))
df['KF_Concrete'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'concrete'))
df['KF_WaterPurifier'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'water purifi
df['KF_Quartz'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'quartz'))
df['KF_Pantry'] = df.kitchen_features.apply(lambda x: IdentifyKeyWord(x, 'pantry'))
```

```
df[['KF_Dishwasher','KF_GarbageDisposal','KF_Refrigerator','KF_DoubleSink','KF_Microwave','KF
    'KF_Freezer','KF_ElectricRange','KF_Island','KF_GasRange','KF_Countertops','KF_Desk','KF_
    'KF_WaterPurifier','KF_Quartz','KF_Pantry']].sum()
```

```
KF_Dishwasher         4857
KF_GarbageDisposal    4520
KF_Refrigerator       4234
KF_DoubleSink         1164
KF_Microwave          3625
KF_Oven               3977
KF_Compactor           432
KF_Freezer             395
KF_ElectricRange       401
KF_Island             1252
KF_GasRange           1307
KF_Countertops        1482
KF_Desk                327
KF_Granite            1114
KF_Concrete             28
KF_WaterPurifier       168
KF_Quartz              136
KF_Pantry             1461
dtype: int64
```

```
# Floor covering
df['FC_Stone'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'natural stone'))
df['FC_Ceramic'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'ceramic tile'))
df['FC_Laminate'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'laminate'))
df['FC_Wood'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'wood'))
df['FC_Carpet'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'carpet'))
df['FC_Concrete'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'concrete'))
df['FC_MexicanTile'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'mexican tile'))
df['FC_Other'] = df.floor_covering.apply(lambda x: IdentifyKeyWord(x, 'other'))
```

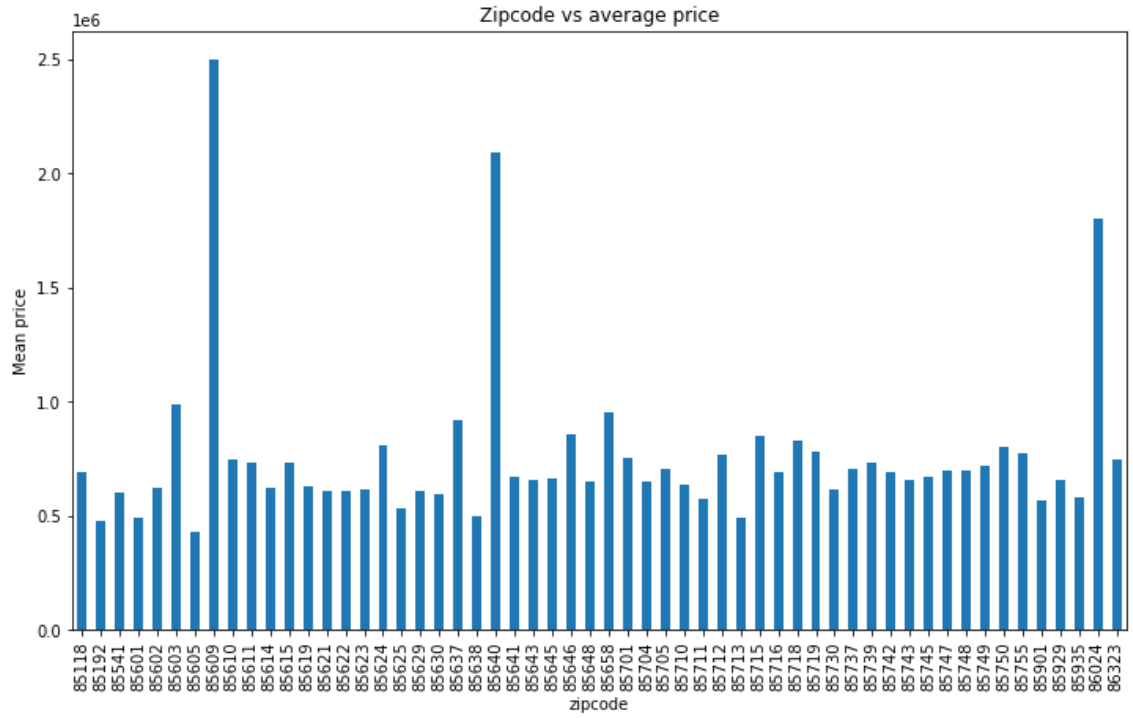
```
df[['FC_Stone','FC_Ceramic','FC_Laminate','FC_Wood','FC_Carpet','FC_Concrete','FC_MexicanTile
```

```
FC_Stone          1499
FC_Ceramic        2527
```

```
FC_Laminate          86
FC_Wood            1248
FC_Carpet          3509
FC_Concrete         756
FC_MexicanTile      660
FC_Other            496
dtype: int64
```

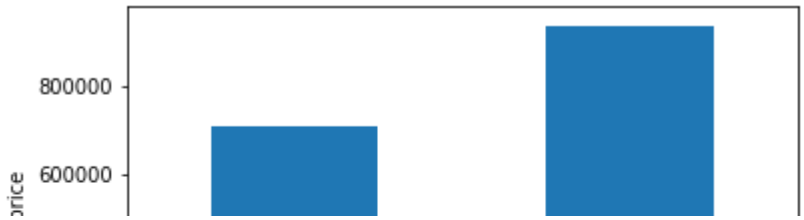
```
dfg = df.groupby(['zipcode'])['sold_price'].mean()
dfg.plot(kind='bar', title='Zipcode vs average price', ylabel='Mean price',
         xlabel='zipcode', figsize=(12, 7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f92dc187450>
```




```
dfg = df.groupby(['FC_Stone'])['sold_price'].mean()
dfg.plot(kind='bar', ylabel='Mean price',
         xlabel='FC_Other')#, figsize=(12, 7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f92dc3b1490>
```




```
df.columns
```

```
Index(['MLS', 'sold_price', 'zipcode', 'longitude', 'latitude', 'lot_acres',
       'taxes', 'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',
       'kitchen_features', 'fireplaces', 'floor_covering', 'HOA',
       'KF_Dishwasher', 'KF_GarbageDisposal', 'KF_Refrigerator',
       'KF_DoubleSink', 'KF_Microwave', 'KF_Oven', 'KF_Compactor',
       'KF_Freezer', 'KF_ElectricRange', 'KF_Island', 'KF_GasRange',
       'KF_Countertops', 'KF_Desk', 'KF_Granite', 'KF_Concrete',
       'KF_WaterPurifier', 'KF_Quartz', 'KF_Pantry', 'FC_Stone', 'FC_Ceramic',
       'FC_Laminate', 'FC_Wood', 'FC_Carpet', 'FC_Concrete', 'FC_MexicanTile',
       'FC_Other'],
      dtype='object')
```

```
def trans_lat(x):
  y = x.replace(',','').replace('.','')
  return (y[:2] + '.' + y[2:])
def trans_lon(x):
  y = x.replace(',','').replace('.','')
  return (y[:4] + '.' + y[4:])

df['latitude'] = df.latitude.apply(trans_lat).astype(float)
df['longitude'] = df.longitude.apply(trans_lon).astype(float)


# Filling missing values
df['HOA'] = df.HOA.str.replace('None','55') #0
df['HOA'] = df.HOA.str.replace(',','.')
df['HOA'] = df.HOA.astype(float)
#df['HOA'] = df.HOA.astype(int)
df['bathrooms'] = df.bathrooms.str.replace('None','4')
df['bathrooms'] = df.bathrooms.astype(float).round()
df['sqrt_ft'] = df.sqrt_ft.str.replace('None','3047') #'3512'
df['sqrt_ft'] = df.sqrt_ft.astype(float)
df['garage'] = df.garage.str.replace('None','3')
df['garage'] = df.garage.astype(float).round()
df['lot_acres'] = df.lot_acres.fillna(1)
df['fireplaces'] = df.fireplaces.fillna(2)
df['year_built'] = df.year_built.replace(0,2019)


df.drop(['MLS','floor_covering','kitchen_features'], axis = 1, inplace = True)
df.info()
```

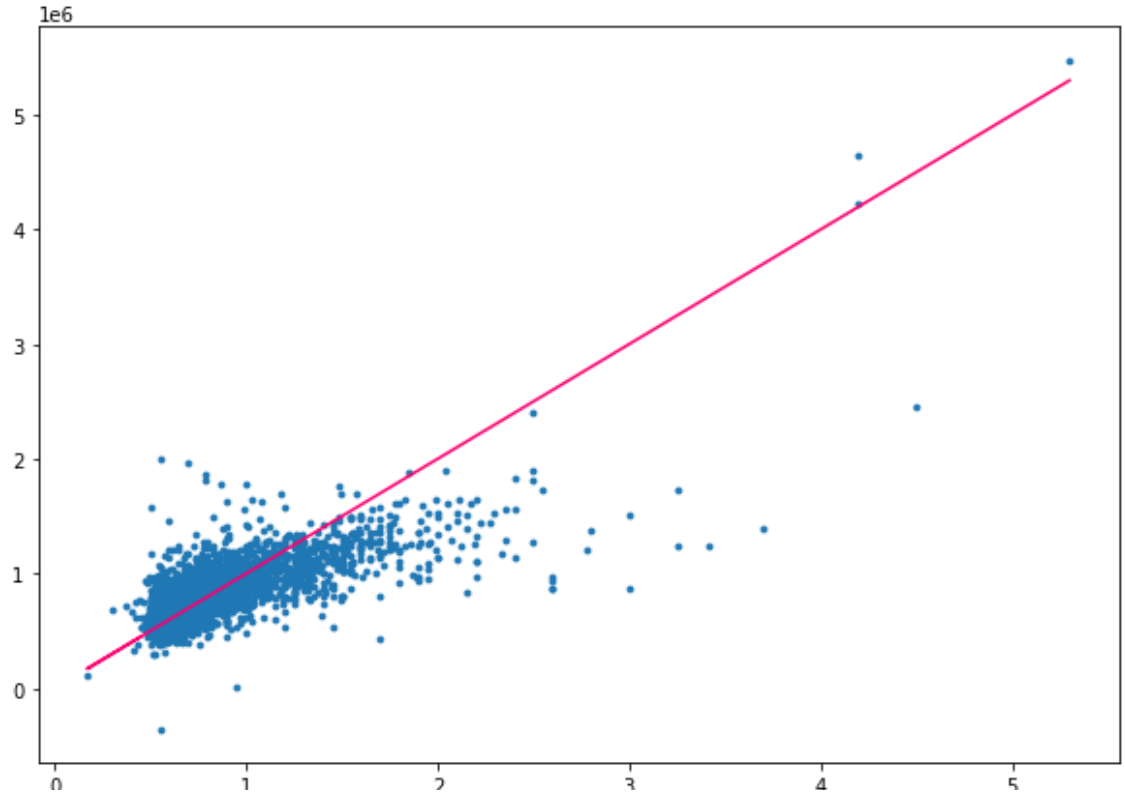
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 39 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   sold_price          5000 non-null   float64
 1   zipcode             5000 non-null   int64  
 2   longitude           5000 non-null   float64
 3   latitude            5000 non-null   float64
 4   lot_acres           5000 non-null   float64
 5   taxes               5000 non-null   float64
 6   year_built          5000 non-null   int64  
 7   bedrooms            5000 non-null   int64  
 8   bathrooms           5000 non-null   float64
 9   sqrt_ft             5000 non-null   float64
 10  garage              5000 non-null   float64
 11  fireplaces          5000 non-null   float64
 12  HOA                 5000 non-null   float64
 13  KF_Dishwasher       5000 non-null   int64  
 14  KF_GarbageDisposal  5000 non-null   int64  
 15  KF_Refrigerator     5000 non-null   int64  
 16  KF_DoubleSink       5000 non-null   int64  
 17  KF_Microwave        5000 non-null   int64  
 18  KF_Oven             5000 non-null   int64  
 19  KF_Compactor        5000 non-null   int64  
 20  KF_Freezer          5000 non-null   int64  
 21  KF_ElectricRange    5000 non-null   int64  
 22  KF_Island           5000 non-null   int64  
 23  KF_GasRange         5000 non-null   int64  
 24  KF_Countertops      5000 non-null   int64  
 25  KF_Desk             5000 non-null   int64  
 26  KF_Granite          5000 non-null   int64  
 27  KF_Concrete         5000 non-null   int64  
 28  KF_WaterPurifier    5000 non-null   int64  
 29  KF_Quartz           5000 non-null   int64  
 30  KF_Pantry           5000 non-null   int64  
 31  FC_Stone            5000 non-null   int64  
 32  FC_Ceramic          5000 non-null   int64  
 33  FC_Laminate         5000 non-null   int64  
 34  FC_Wood             5000 non-null   int64  
 35  FC_Carpet           5000 non-null   int64  
 36  FC_Concrete         5000 non-null   int64  
 37  FC_MexicanTile      5000 non-null   int64  
 38  FC_Other            5000 non-null   int64  
dtypes: float64(10), int64(29)
memory usage: 1.5 MB
```

```
fig = px.scatter_mapbox(df,lat = 'latitude', lon = 'longitude', color = 'sold_price',
                        center=dict(lon=-110.9, lat=32.3),
                       zoom = 9, mapbox_style = 'open-street-map')
fig
```


```
df['latitude2'] = df['latitude']**2
df['longitude2'] = df['longitude']**2
X = df.to_numpy()
y = X[:,0]
X = X[:,1:]


class MultiLinearRegression():

  def fit(self, X, y):
    self.W = np.linalg.solve(X.T@X, X.T@y)

  def predict(self, X):

    return np.matmul(X, self.W)


lr_mul = MultiLinearRegression()


lr_mul.fit(X,y)


y_hat = lr_mul.predict(X)


plt.figure(figsize=(10,7))
```

```
plt.scatter(y, y_hat, s=8)
plt.plot(y,y, color = '#FF0070')
```

```
[<matplotlib.lines.Line2D at 0x7f92dcc01c50>]
```




```
def R2(Y, Y_hat):
  return(1-(np.sum((Y-Y_hat)**2)/np.sum((Y-np.mean(Y))**2)))
```

```
R2(y, y_hat)
```

```
0.5614112573942174
```

```
class KNNRegressor:

  def fit(self, X, y):
    self.X = X
    self.y = y

  def predict(self, X, K, epsilon = 1e-3):
    N = len(X)
    y_hat = np.zeros(N)
    for i in range(N):
      dist2 = np.sum((self.X-X[i])**2, axis = 1)
      idxt = np.argsort(dist2)[:K]
      gamma_K = np.exp(-dist2[idxt]) / np.exp(-dist2[idxt]).sum()
      y_hat[i] = gamma_K.dot(self.y[idxt])
```

```
    return y_hat

knn_reg = KNNRegressor()
```

```
df.columns
```

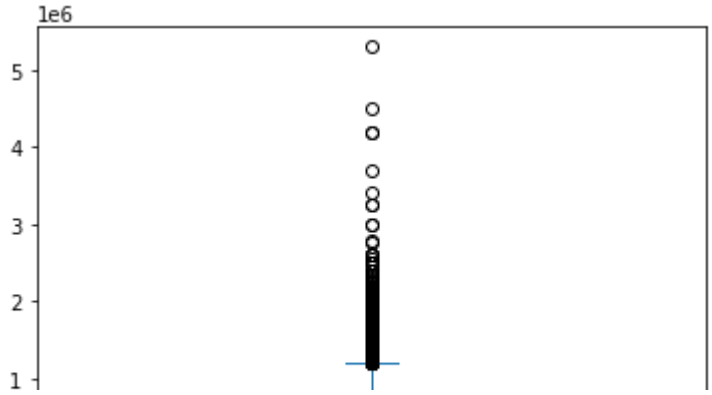
```
Index(['sold_price', 'zipcode', 'longitude', 'latitude', 'lot_acres', 'taxes',
       'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',
       'fireplaces', 'HOA', 'KF_Dishwasher', 'KF_GarbageDisposal',
       'KF_Refrigerator', 'KF_DoubleSink', 'KF_Microwave', 'KF_Oven',
       'KF_Compactor', 'KF_Freezer', 'KF_ElectricRange', 'KF_Island',
       'KF_GasRange', 'KF_Countertops', 'KF_Desk', 'KF_Granite', 'KF_Concrete',
       'KF_WaterPurifier', 'KF_Quartz', 'KF_Pantry', 'FC_Stone', 'FC_Ceramic',
       'FC_Laminate', 'FC_Wood', 'FC_Carpet', 'FC_Concrete', 'FC_MexicanTile',
       'FC_Other', 'latitude2', 'longitude2'],
      dtype='object')
```

```
df[['sold_price', 'longitude', 'latitude', 'lot_acres', 'taxes',
       'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',
       'fireplaces', 'HOA']].corr()
```

	sold_price	longitude	latitude	lot_acres	taxes	year_bui
sold_price	1.000000	-0.036819	0.035206	0.332329	0.023326	0.0984
longitude	-0.036819	1.000000	-0.251697	0.127891	-0.000959	-0.1090
latitude	0.035206	-0.251697	1.000000	-0.199936	0.000823	0.1725
lot_acres	0.332329	0.127891	-0.199936	1.000000	-0.000567	-0.0662
taxes	0.023326	-0.000959	0.000823	-0.000567	1.000000	-0.0042
year_built	0.098461	-0.109066	0.172578	-0.066292	-0.004243	1.0000
bedrooms	0.115932	0.058279	-0.091935	0.069328	0.005259	-0.1845
bathrooms	0.325982	0.014856	-0.081501	0.054924	0.008971	-0.0499
sqrt_ft	0.525068	0.064036	-0.096670	0.103044	0.037699	-0.0581
garage	0.100672	-0.041764	0.060698	-0.052492	0.005621	0.3157
fireplaces	0.383819	0.046265	-0.087933	0.086336	0.022540	-0.1269

```
df.sold_price.plot.box()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f92dc9dacd0>
```



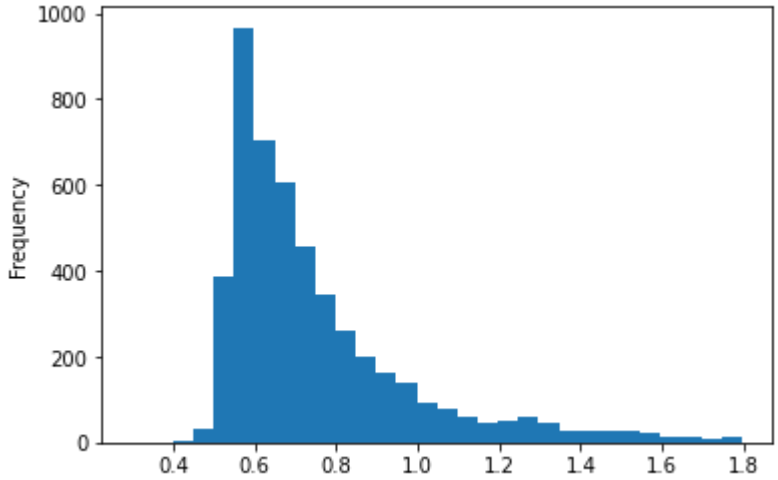
Treating/Removing **Outliers**

```
# Removing outliers from sold price
print(df.sold_price.quantile(0.05))
print(df.sold_price.quantile(0.98))
```

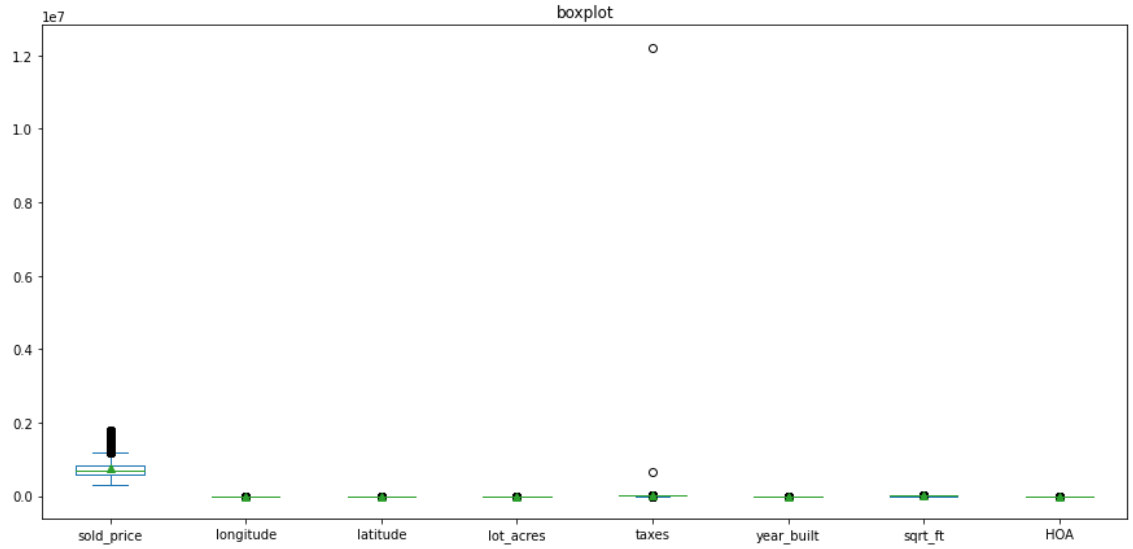
```
535000.0
1700299.999999993
```

```
#df['sold_price'] = np.where(df['sold_price'] > 1700000, 1700000, df['sold_price'])
df = df[df['sqrt_ft'] > 2000]
df = df[df['sold_price'] < 1800000]
df.sold_price.plot.hist(bins = 30)
df.shape
```

```
(4881, 41)
```




```
# Removing outliers from the rest of the data
ax = df[['sold_price','longitude','latitude','lot_acres','taxes','year_built','sqrt_ft','HOA'

plt.show()
```




```
df[df['taxes'] > 500000]
```

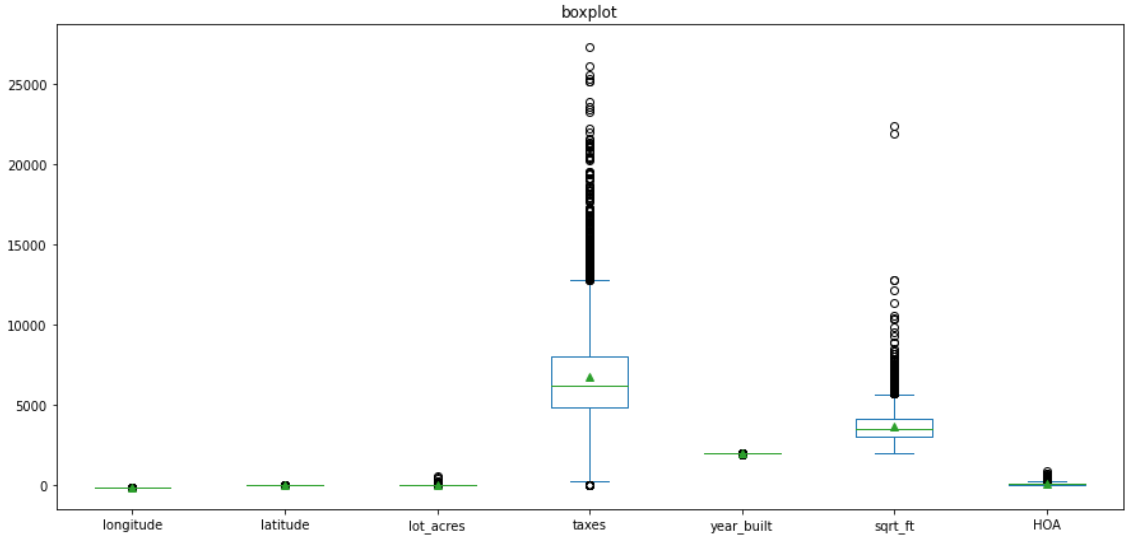
	sold_price	zipcode	longitude	latitude	lot_acres	taxes	y
715	1040000.0	85718	-110.909653	32.301099	1.43	12215075.0	
1135	885000.0	85718	-110.924210	32.309002	1.20	668059.0	

2 rows × 41 columns

```
df = df[df['taxes'] < 500000]
```

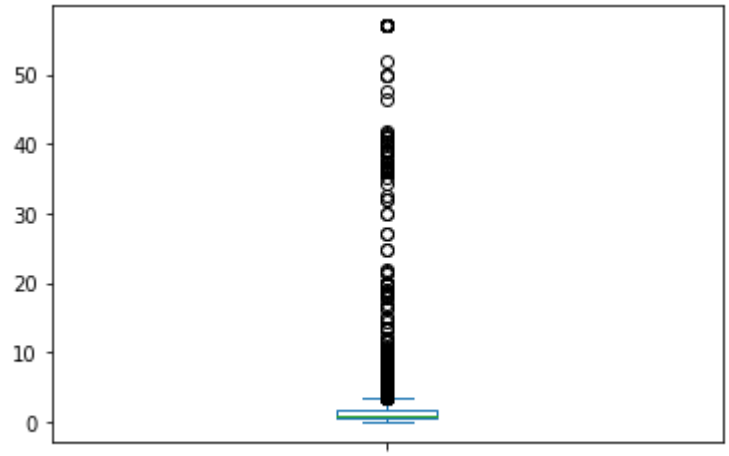
```
ax = df[['longitude','latitude','lot_acres','taxes','year_built','sqrt_ft','HOA']].plot(kind=
```

```
plt.show()
```



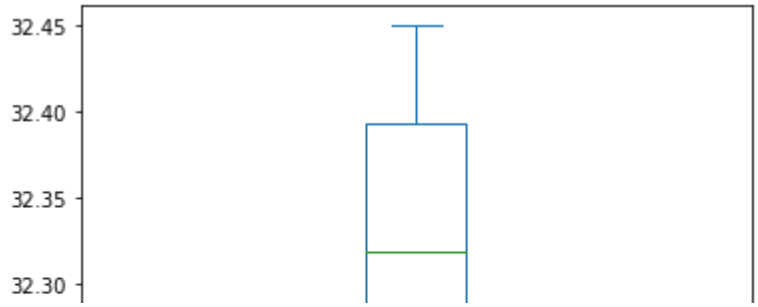

```
print(df.lot_acres.quantile(0.99))
print(df.lot_acres.quantile(0.5))
df['lot_acres'] = np.where(df['lot_acres'] > 57, 57, df['lot_acres'])
df.lot_acres.plot.box()
```

```
40.42760000000002
0.98
<matplotlib.axes._subplots.AxesSubplot at 0x7f92db8080d0>
```



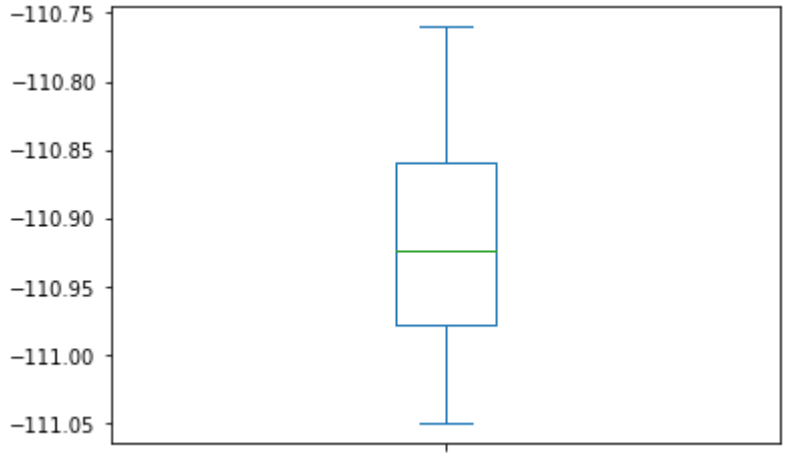

```
print(df.latitude.quantile(0.9))
print(df.latitude.quantile(0.5))
print(df.latitude.quantile(0.1))
df['latitude'] = np.where(df['latitude'] > 32.45, 32.45, df['latitude'])
df['latitude'] = np.where(df['latitude'] < 32.22, 32.22, df['latitude'])
df.latitude.plot.box()
```

```
32.4589508
32.318308
32.2256194
<matplotlib.axes._subplots.AxesSubplot at 0x7f92db777d50>
```



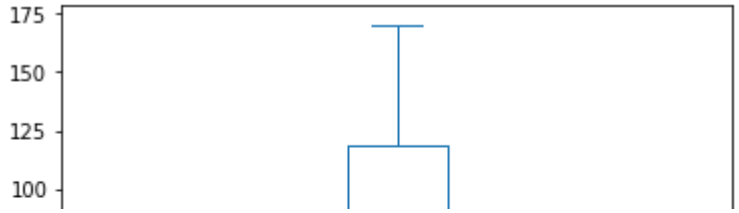

```
print(df.longitude.quantile(0.9))
print(df.longitude.quantile(0.5))
print(df.longitude.quantile(0.1))
df['longitude'] = np.where(df['longitude'] > -110.76, -110.76, df['longitude'])
df['longitude'] = np.where(df['longitude'] < -111.05, -111.05, df['longitude'])
df.longitude.plot.box()
```

```
-110.76590859999999
-110.92346
-111.051299
<matplotlib.axes._subplots.AxesSubplot at 0x7f92db0ec090>
```



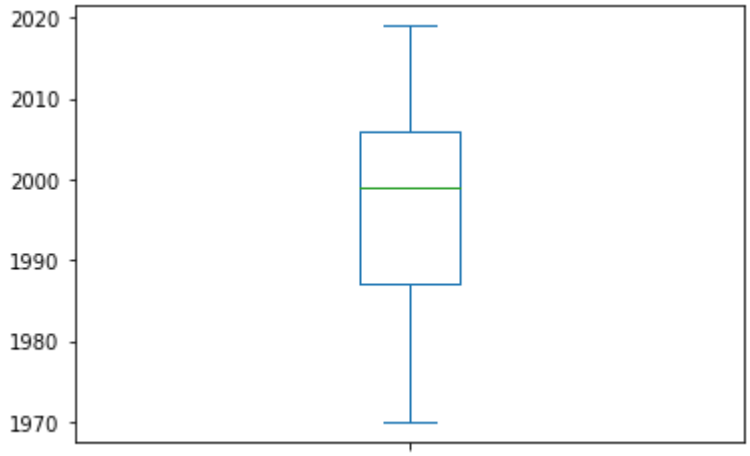

```
print(df.HOA.quantile(0.9))
print(df.HOA.quantile(0.5))
df['HOA'] = np.where(df['HOA'] > 170, 170, df['HOA'])
df.HOA.plot.box()
```

```
180.0
55.0
<matplotlib.axes._subplots.AxesSubplot at 0x7f92db06c250>
```




```
print(df.year_built.quantile(0.1))
print(df.year_built.quantile(0.5))
df['year_built'] = np.where(df['year_built'] < 1970, 1970, df['year_built'])
df.year_built.plot.box()
```

```
1971.0
1999.0
<matplotlib.axes._subplots.AxesSubplot at 0x7f92db11b150>
```

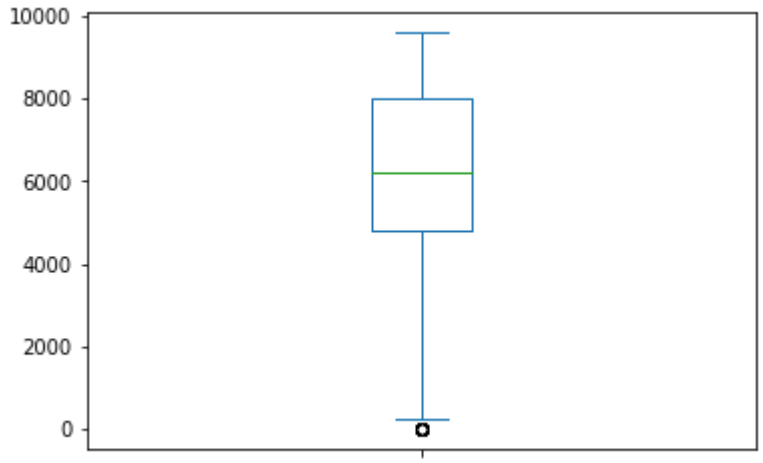



```
print(df.sqrt_ft.quantile(0.95))
print(df.sqrt_ft.quantile(0.5))
df['sqrt_ft'] = np.where(df['sqrt_ft'] > 5664, 5664, df['sqrt_ft'])#4592
df.sqrt_ft.plot.box()
```

```
5489.0
3495.0
```

```
print(df.taxes.quantile(0.9))
print(df.taxes.quantile(0.5))
df['taxes'] = np.where(df['taxes'] > 9595, 9595, df['taxes'])
df.taxes.plot.box()
```

```
10577.418
6218.85
<matplotlib.axes._subplots.AxesSubplot at 0x7f92daf38c90>
```




```
df[['sold_price', 'longitude', 'latitude', 'lot_acres', 'taxes',
       'year_built', 'bedrooms', 'bathrooms', 'sqrt_ft', 'garage',
       'fireplaces', 'HOA']].corr()
```

```
vars = ['sold_price','garage','sqrt_ft','longitude','latitude','bedrooms','fireplaces','taxes
        'HOA','year_built','lot_acres','FC_Ceramic'] #,,'FC_Wood''KF_Refrigerator'
        #'KF_ElectricRange','bathrooms','zipcode','FC_Stone','KF_Compactor','KF_Freezer'
df = df[vars]
```

latitude	0.109379	-0.479889	1.000000	-0.217506	0.066621	0.3640

```
## Split database
i = np.arange(len(df))
np.random.shuffle(i)
train = df.iloc[i[:int(len(df)*0.75)]]
test = df.iloc[i[int(len(df)*0.75):int(len(df)*0.85)]]
validation = df.iloc[i[int(len(df)*0.85):]]
```

bathrooms	0.322490	0.002420	0.065164	0.157388	0.273648	0.017

```
def normalize_function(data):
   min = np.amin(data,axis=0)
   max = np.amax(data,axis=0)
   return (data - min)/(max-min)
```

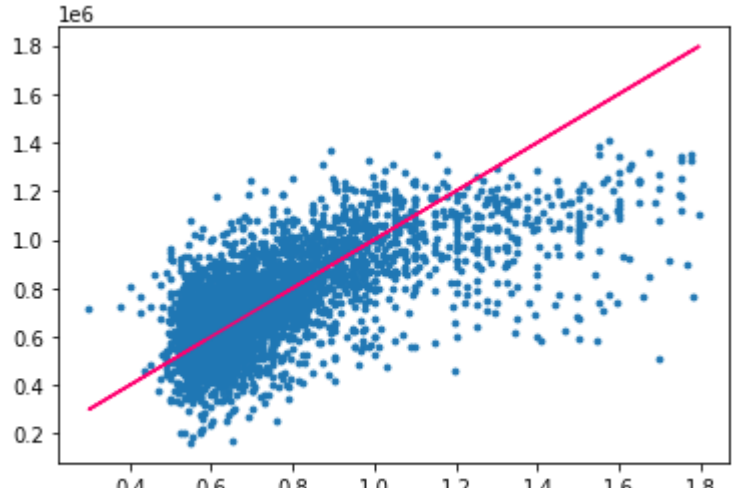
```
print(train.shape)
print(test.shape)
print(validation.shape)
```

```
(3659, 12)
(488, 12)
(732, 12)
```

```
# Transform all datasets into arrays
X_train = train.to_numpy()
y_train = X_train[:,0]
X_train = X_train[:,1:]
X_test = test.to_numpy()
y_test = X_test[:,0]
X_test = X_test[:,1:]
X_validation = validation.to_numpy()
y_validation = X_validation[:,0]
X_validation = X_validation[:,1:]
X_train = normalize_function(X_train)
X_test = normalize_function(X_test)
X_validation = normalize_function(X_validation)
```

```
lr_mul.fit(X_train,y_train)
y_hat = lr_mul.predict(X_train)
#plt.figure(figsize=(10,7))
plt.scatter(y_train, y_hat, s=8)
plt.plot(y_train,y_train, color = '#FF0070')
```

```
[<matplotlib.lines.Line2D at 0x7f92dae5ba50>]
```



Model 1: Simple Linear Regression

```
R2(y_train, y_hat)
```

```
0.385863109907968
```

```
y_hat = lr_mul.predict(X_test)
R2(y_test, y_hat)
```

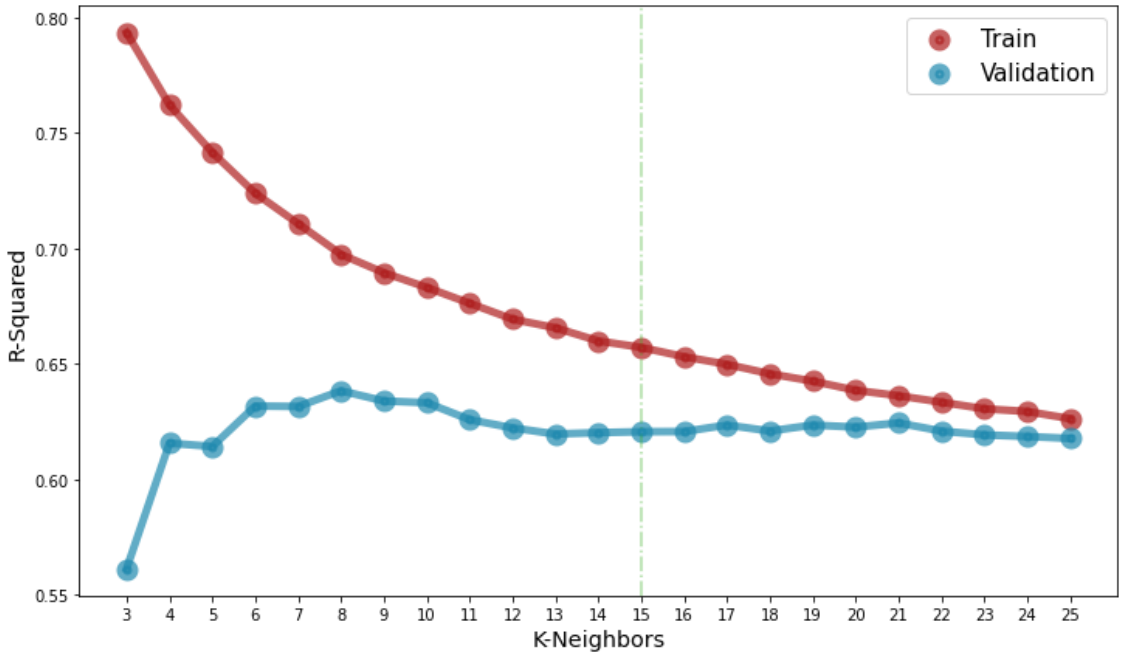
```
-0.9856062905442071
```

Model 2: K-Nearest Neightbors

```
def performance(XTrain, yTrain, XTest, yTest, K):
  knn_reg.fit(XTrain,yTrain)
  yTrain_hat = knn_reg.predict(XTrain, K)
  yTest_hat = knn_reg.predict(XTest, K)
  return np.array([R2(yTrain, yTrain_hat), R2(yTest, yTest_hat)])


K_grid = np.array([3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25])
R2_results = []

for k in K_grid:
  R2_results.append( performance(X_train, y_train, X_test, y_test, k) )

df_results = pd.DataFrame({'Neighbors': K_grid,
                           'Train': np.array(R2_results)[:,0],
                           'Test': np.array(R2_results)[:,1]})
df_results
```

	Neighbors	Train	Test
0	3	0.793363	0.561310
1	4	0.762019	0.615650
2	5	0.741470	0.614244
3	6	0.723838	0.631726
4	7	0.710675	0.631540
5	8	0.697354	0.638335
6	9	0.689428	0.633918
7	10	0.683050	0.633170
8	11	0.676203	0.625831
9	12	0.669323	0.622280
10	13	0.665584	0.619611
11	14	0.659904	0.620232
12	15	0.657112	0.620630
13	16	0.653081	0.620693
14	17	0.649793	0.623504
15	18	0.645694	0.620922
16	19	0.642501	0.623503
17	20	0.638542	0.622621
18	21	0.636201	0.624407
19	22	0.633400	0.620905
20	23	0.630546	0.619241

```
plt.figure(figsize = (12,7))
plt.axvline(x=15, alpha = 0.4, color = '#36AF1E', linestyle='dashdot')
plt.plot(df_results.Neighbors, df_results.Train, color ='#AF1E1E', linewidth=5, alpha = 0.7)
plt.plot(df_results.Neighbors, df_results.Test, color = '#1E89AF', linewidth=5, alpha = 0.7)
plt.scatter(df_results.Neighbors, df_results.Train, color ='#AF1E1E', label = 'Train', linewi
plt.scatter(df_results.Neighbors, df_results.Test, color = '#1E89AF', label = 'Validation', l
plt.xlabel('K-Neighbors', size = 'x-large')
plt.ylabel('R-Squared', size = 'x-large')
plt.legend(fontsize=15)
plt.xticks(np.arange(3, 26, 1))
plt.show()
```



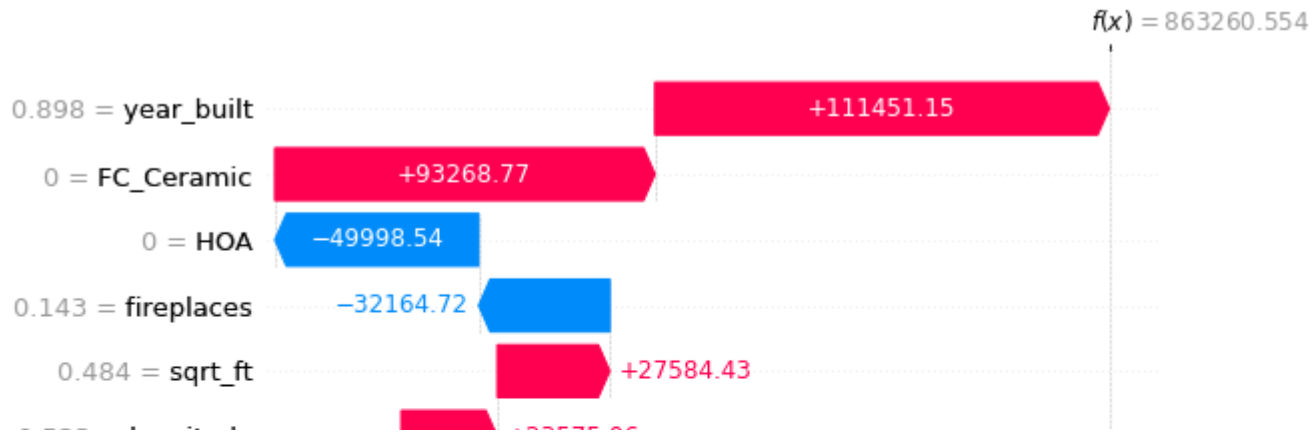

```
yvalidation_hat = knn_reg.predict(X_validation, 15)
R2(y_validation, yvalidation_hat)
```

```
0.6664056024818594
```

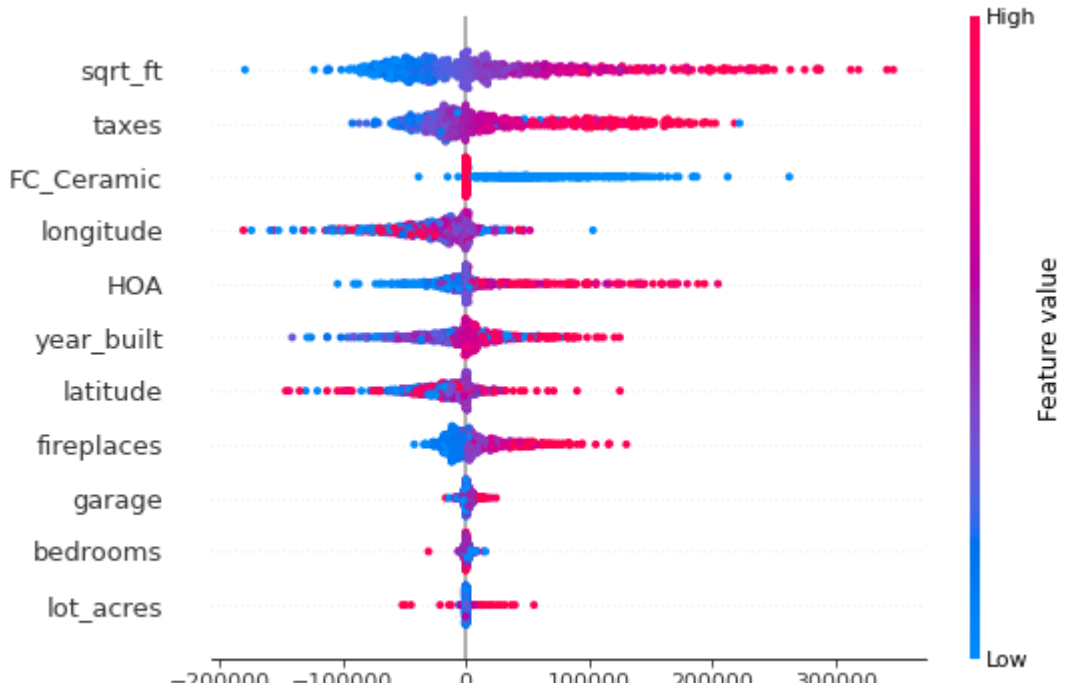
```
f = lambda x: knn_reg.predict(x, 9)
med = np.median(X_train, axis = 0).reshape(1, X_train.shape[1])

explainer = shap.Explainer(f, med,
                           feature_names = ['garage','sqrt_ft','longitude','latitude','bedroo
shap_values = explainer(X_validation)
```

```
Permutation explainer: 733it [03:07,  3.70it/s]
```

```
shap.plots.waterfall(shap_values[10])
```




```
shap.plots.beeswarm(shap_values, max_display=12)
```




```
shap.plots.bar(shap_values, max_display=12)
```