Predicting Housing Price Using Neural Network
[Tensorflow and Keras]

The Data:

Real DB from Kaggle: (https://www.kaggle.com/datasets/harlfoxem/housesalesprediction)

2014-15 Home Sales in King County, WA, one of three Washington counties that are included in the Seattle metropolitan area.

DB Dictionary (https://geodacenter.github.io/data-and-lab/KingCounty-HouseSales2015/):

- id: Identification
- date: Date sold
- price: Sale price
- bedrooms: Number of bedrooms
- bathrooms: Number of bathrooms
- sqft_liv: Size of living area in square feet
- sqft_lot: Size of the lot in square feet
- floors: Number of floors
- waterfront: '1' if the property has a waterfront, '0' if not.
- view An index from 0 to 4 of how good the view of the property was
- condition Condition of the house, ranked from 1 to 5
- grade Classification by construction quality which refers to the types of materials used and the quality of workmanship. Buildings of better quality (higher grade) cost more to build per unit of measure and command higher value. Additional information in: KingCounty
- sqft_above: Square feet above ground
- sqft_basmt: Square feet below ground
- yr_built: Year built
- yr_renov: Year renovated. '0' if never renovated
- zipcode: 5 digit zip code
- lat: Latitude
- long: Longitude
- squft_liv15: Average size of interior housing living space for the closest 15 houses, in square feet
- squft_lot15: Average size of land lots for the closest 15 houses, in square feet
- Shape_leng: Polygon length in meters
- Shape_Area: Polygon area in meters

Goal / Target:

Re...

Tools:

- Python
- Numpy, Pandas
- Matplotlib, Seaborn
- Scikitlearn
- Keras, Tensorflow

Step 1: Preparing the Data

There isn't missing data (null) or duplicate values.

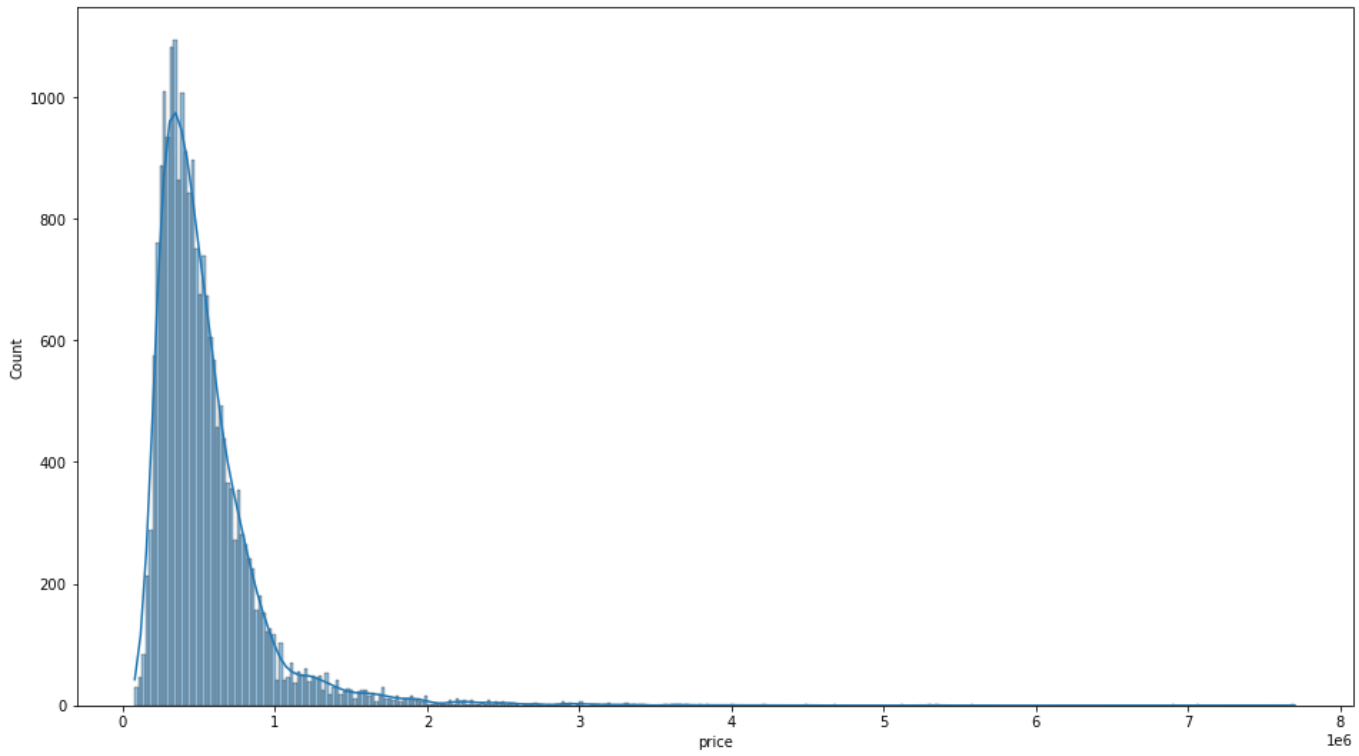
```
df.isnull().sum()

id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
```

```
sqft_lot15       0
dtype: int64
```

Step 2: EDA - Exploratory Data Analysis

Probably we can predict very well the house prices between 0 and 2 million dolars. So, we can exclude the outliers.

```
sns.histplot(df['price'],kde=True)
```



Exploring posible correlations.

Seems that the price is very correlated with sqft_living feature:

```
df.corr()['price'].sort_values()

zipcode         -0.053402
id              -0.016772
long             0.022036
```

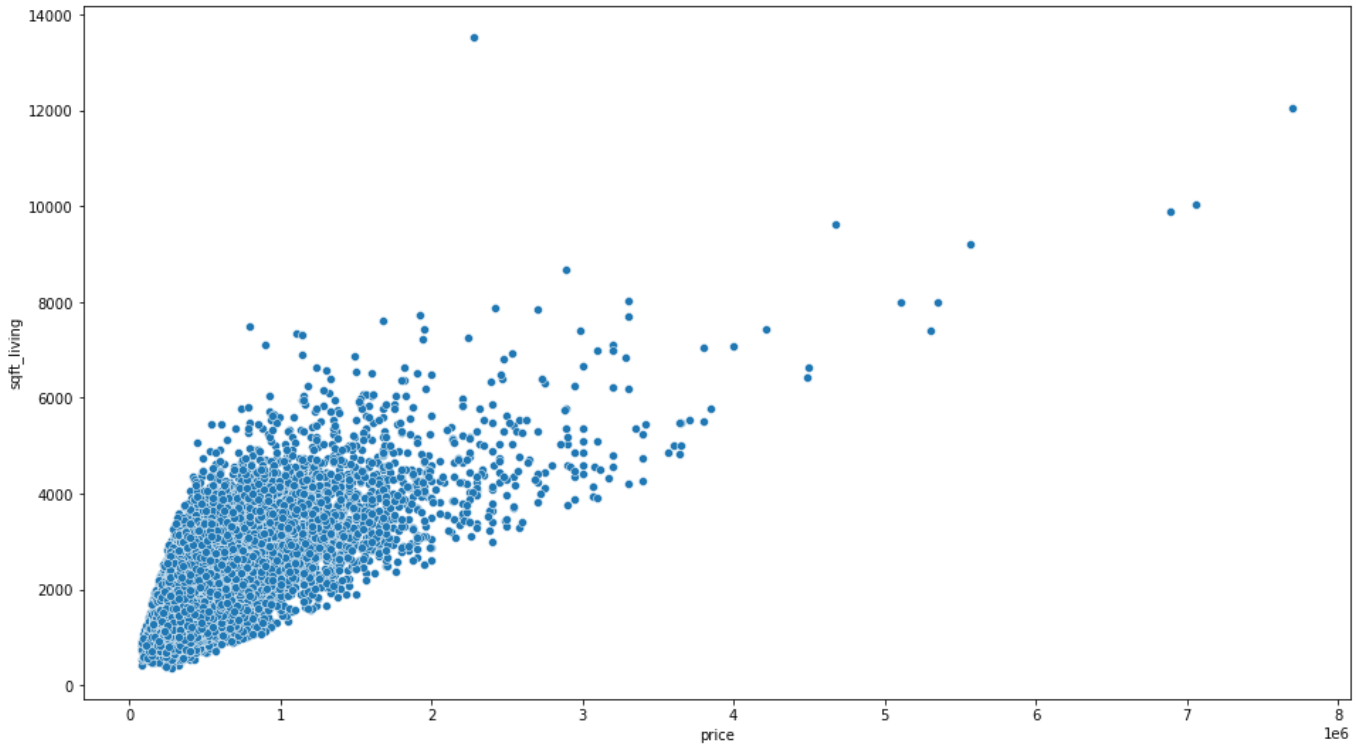
```
condition        0.036056
yr_built         0.053953
sqft_lot15       0.082845
sqft_lot         0.089876
yr_renovated     0.126424
floors           0.256804
waterfront       0.266398
lat              0.306692
bedrooms         0.308787
sqft_basement    0.323799
view             0.397370
bathrooms        0.525906
sqft_living15    0.585241
sqft_above       0.605368
grade            0.667951
sqft_living      0.701917
price            1.000000
Name: price, dtype: float64
```

We can see the same correlation on the pairplot and with others faetures. #But price and sqft_living is really correlated.

Not showing it here beacause its not a good way to see in pairplot with a lot of attributes, the plots got very small.

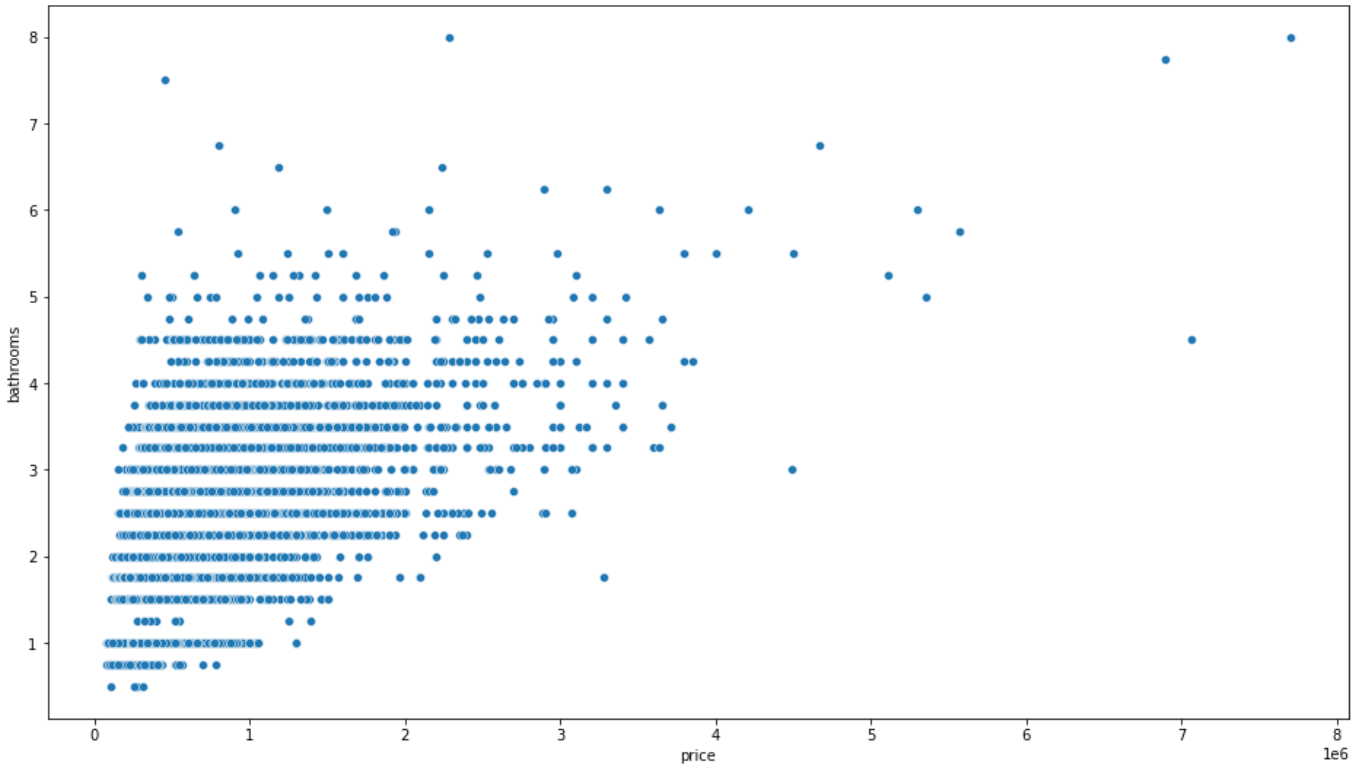
Let´s see that correlation closely:

```
sns.scatterplot(data=df,x='price',y='sqft_living')
```

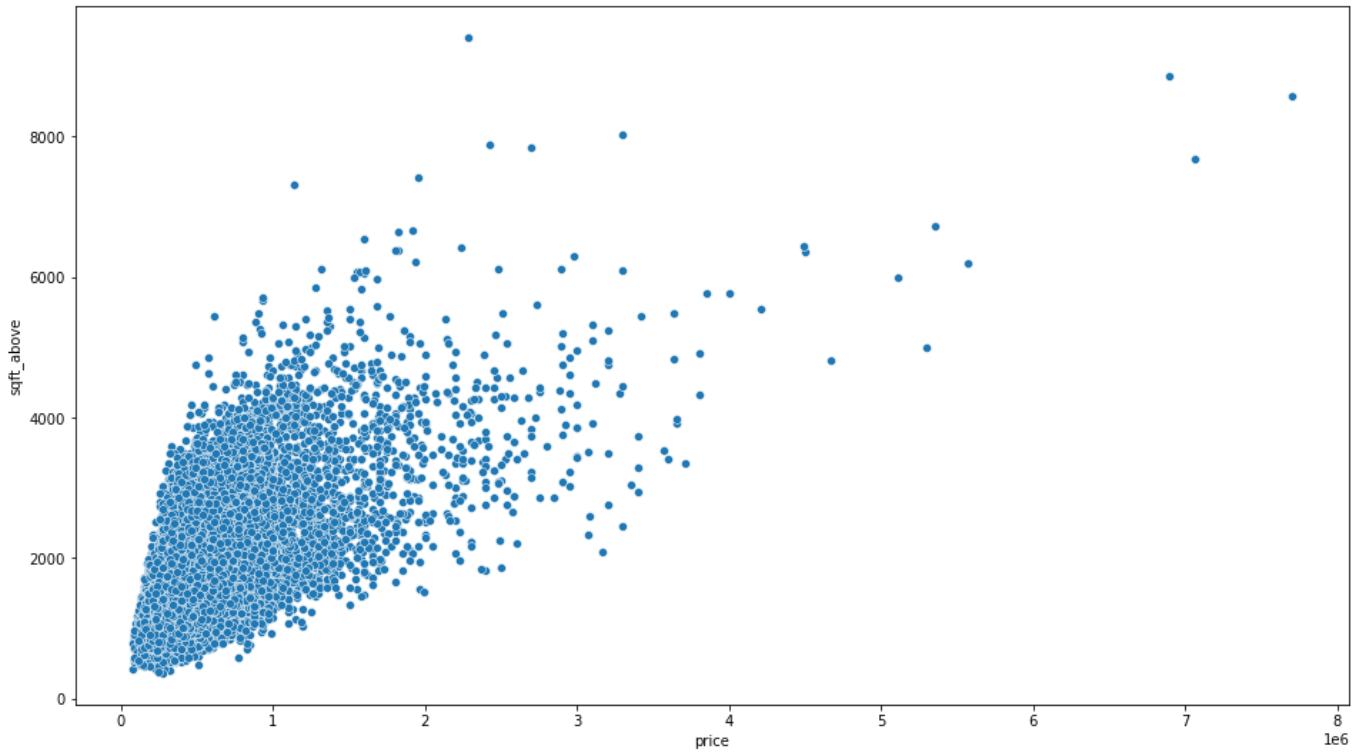


Other correlations:

```
sns.scatterplot(data=df,x='price',y='bathrooms')
```

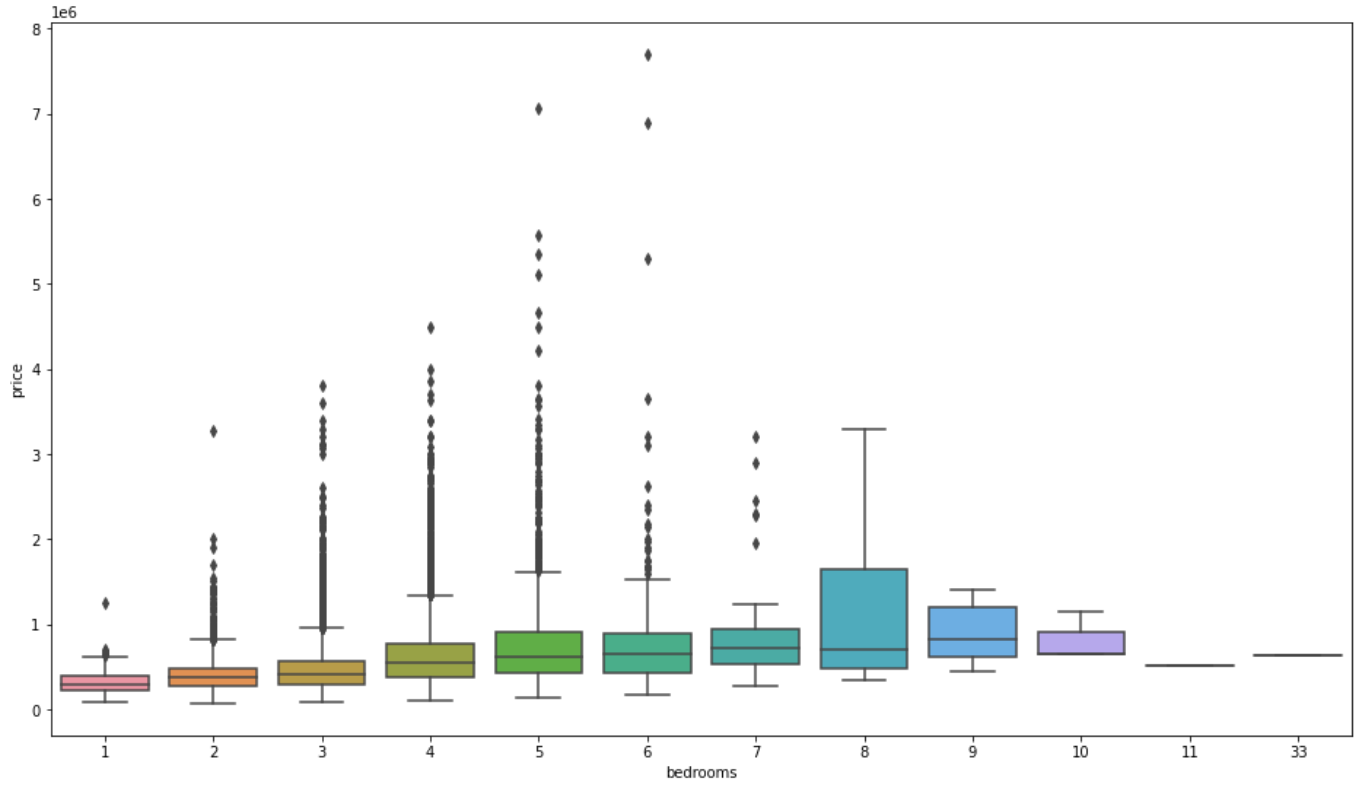


```
sns.scatterplot(data=df,x='price',y='sqft_above')
```



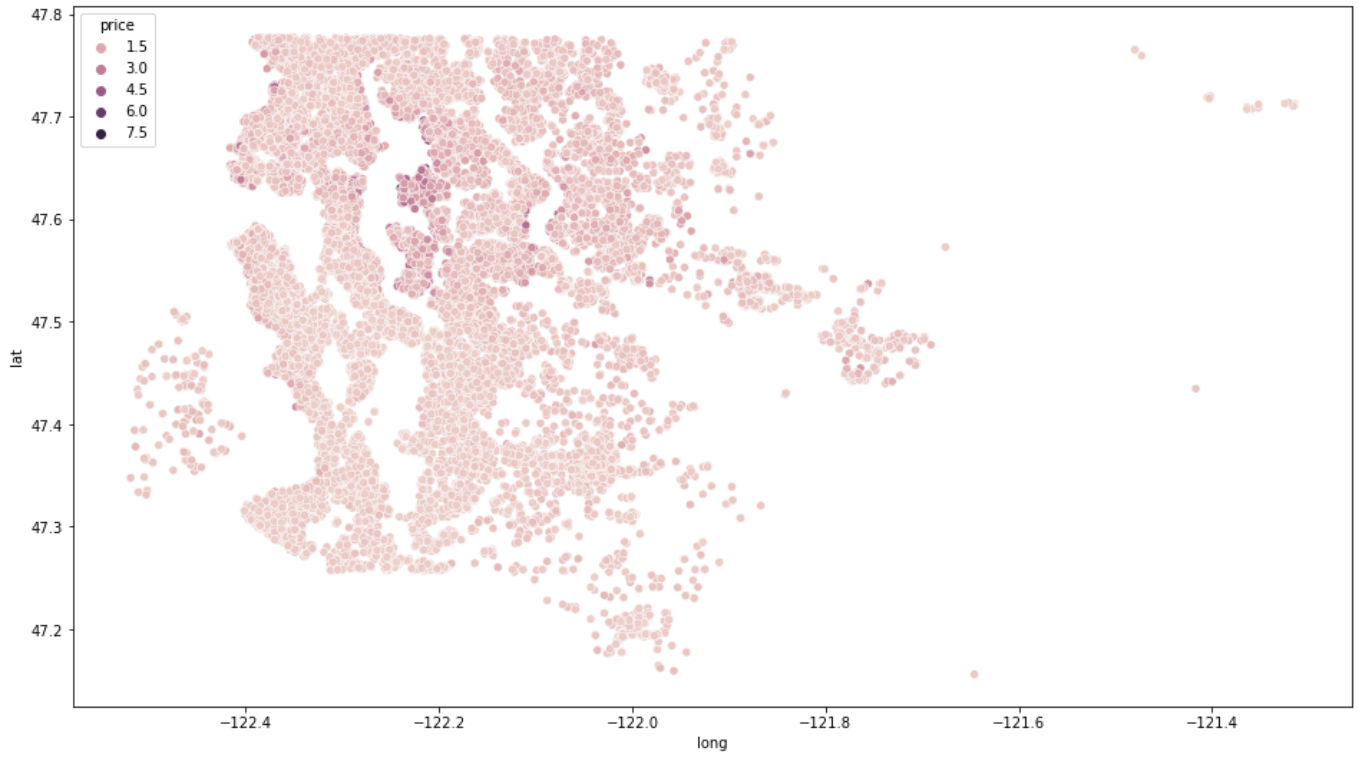
Distributions of price per number of bedrooms:

```
sns.boxplot(data=df,x='bedrooms',y='price')
```



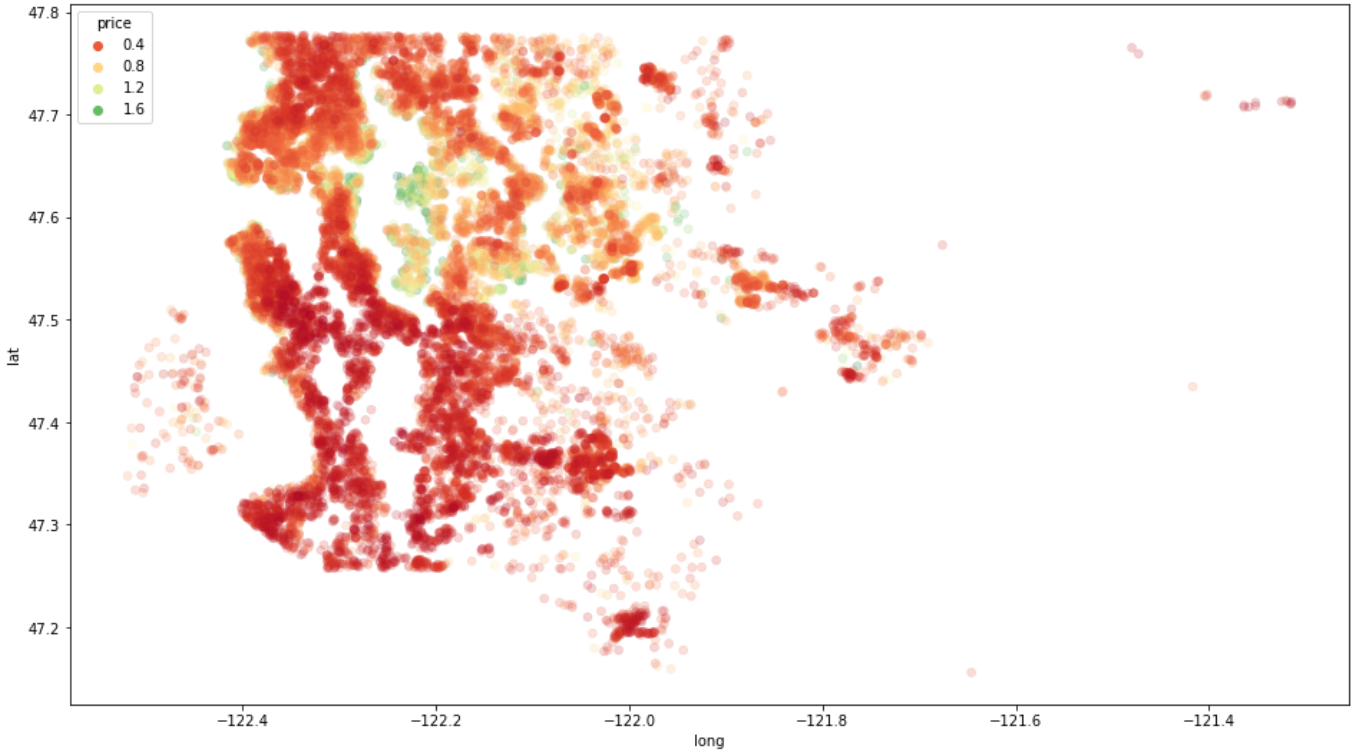
Let's use lat (latitude) and long (longitude feature) to build a King County's map and check if that interferes on the house prices:

```
sns.scatterplot(data=df,x='long',y='lat',hue='price')
```



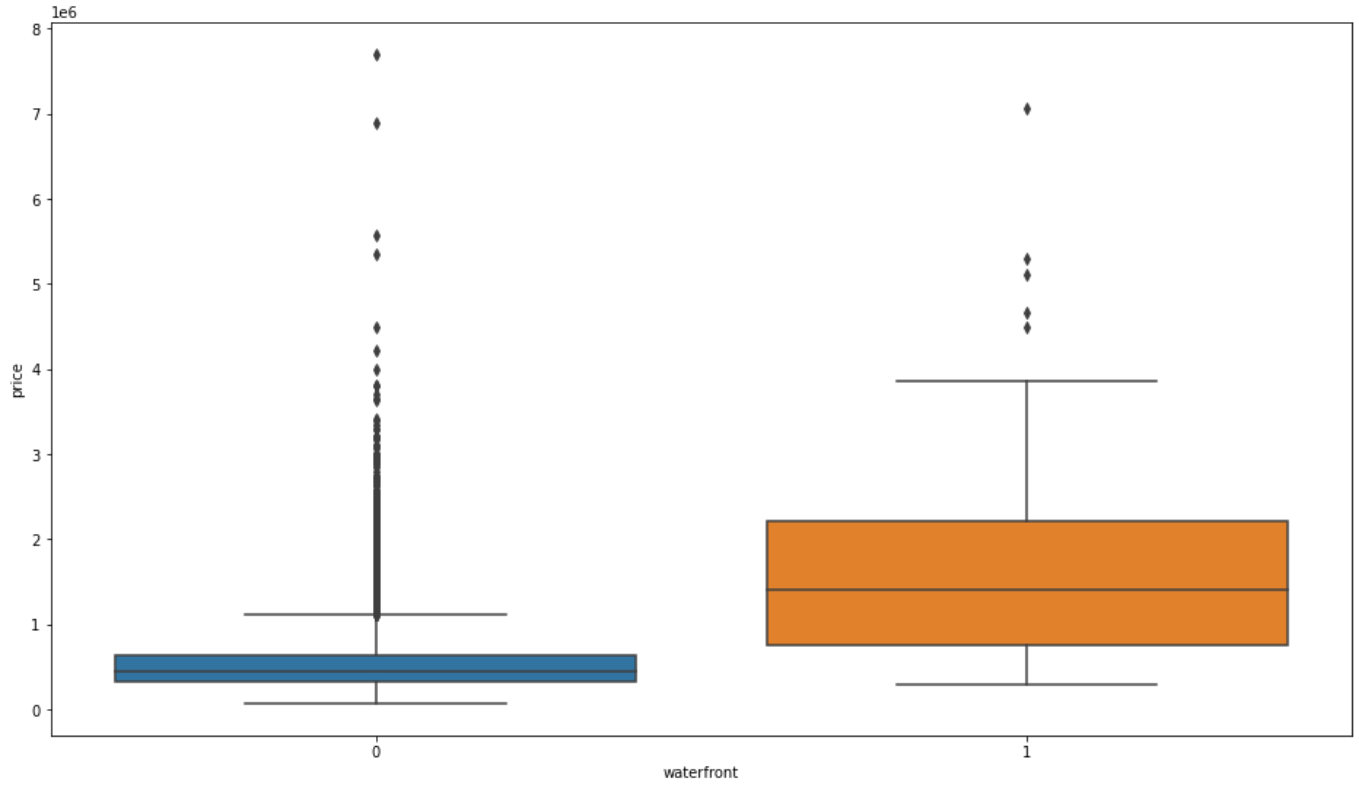
Let's drop 1% of the house/prices (the outliers):

```
len(df)*0.01
215.97

df_99perc = df.sort_values('price',ascending=False).iloc[216:]

sns.scatterplot(data=df_99perc,x='long',y='lat',hue='price',
                edgecolor=None, alpha=0.2, palette='RdYlGn')

# Red less expensive
# Green more expensive
```



Water front feature vs. prices:

```
sns.boxplot(data=df,x='waterfront',y='price')
```



Step 3: Feature Engineering Process

Transformming the most relevant variables from raw data to improve the performance of machine learning (ML) algorithms.

Lets drop features that are not important to us and use just 99% of the data (taking out those outliers):

```
df_99perc = df.drop('id',axis=1)
```

Converting to datetime object:

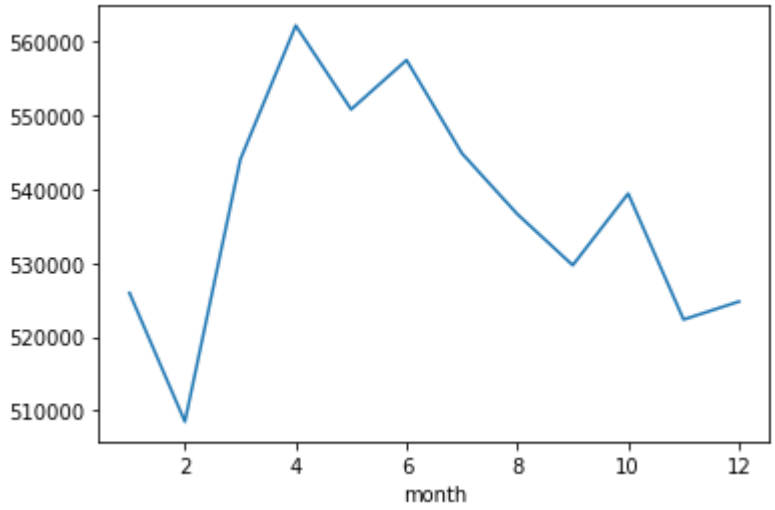
```
df_99perc['date'] = pd.to_datetime(df_99perc['date'])
```

Creating two new columns to observe the datetime better:

```
df_99perc['year'] = df_99perc['date'].apply(lambda date: date.year)

df_99perc['month'] = df_99perc['date'].apply(lambda date: date.month)

df_99perc = df_99perc.drop('date',axis=1)
```

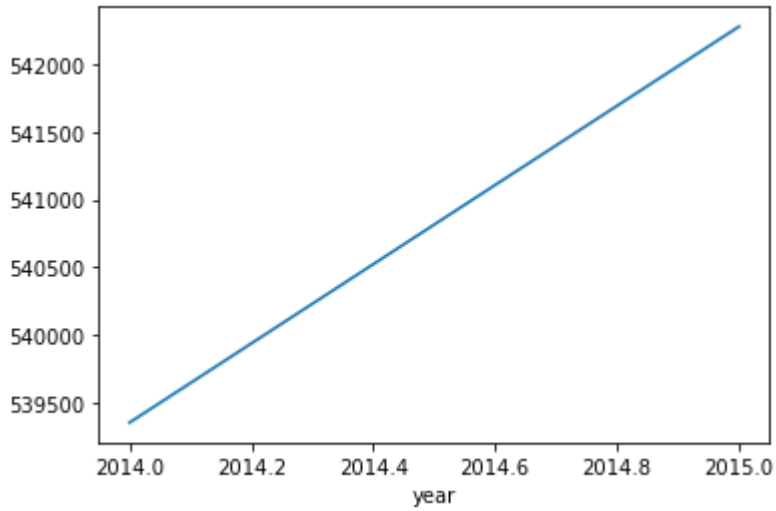
There is some influence in on sales price by month? A bit of influence, yes.

```
df_99perc.groupby('month').mean()['price'].plot()
```

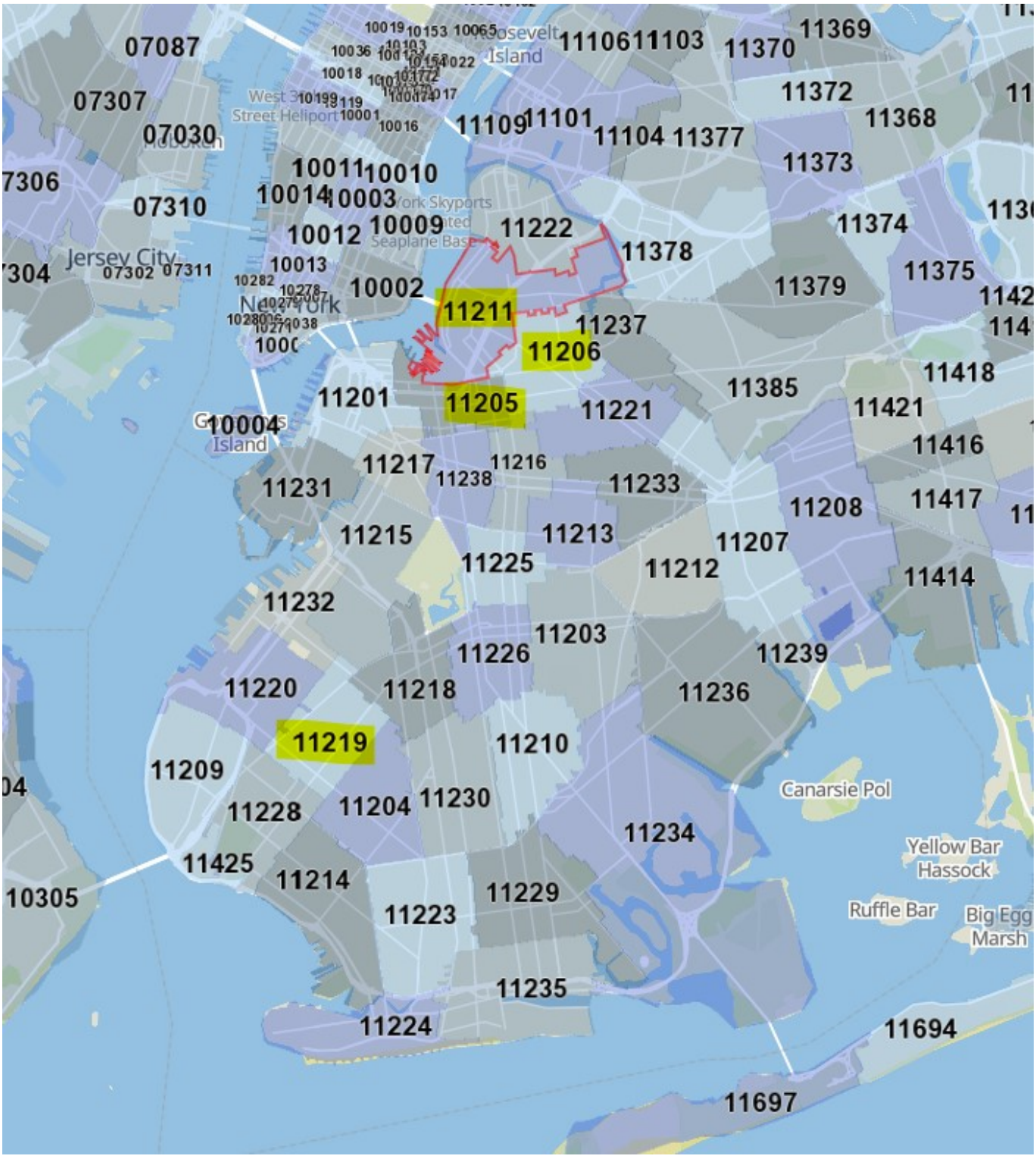


The prices go up one year after other, what makes sense:

```
df_99perc.groupby('year').mean()['price'].plot()
```



Let´s drop the zipcode too because it's not logical distributted and we already have latitude and longitude.




```
df_99perc = df_99perc.drop('zipcode',axis=1)
```

Step 4: Predictions

Train test split:

```
X = df_99perc.drop('price',axis=1).values
y = df_99perc['price'].values

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=101)
```

Scalling the data:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Creating a deep learning model with Tensorflow and Keras:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

X_train.shape

(15117, 19)
```

Let´s use 19 neurons per layer:

```
model = Sequential()

model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))
model.add(Dense(19,activation='relu'))

model.add(Dense(1))

model.compile(optimizer='adam',loss='mse')
```

Obs.: Adam optimization is a stochastic gradient descent method beacuse it 'combine' the benefits of AdaGrad (Adaptive Gradient Algorithm) and RMSProp (Root Mean Square Propagation).

Trainning our model:

```
model.fit(x=X_train,y=y_train,
          validation_data=(X_test,y_test),
          batch_size=128,
          epochs=400,
          verbose=0)
```

Evaluating the Model:

History of the **calculated losses**:

```
pd.DataFrame(model.history.history)

        loss            val_loss
0   4.302356e+11    4.188866e+11
1   4.287633e+11    4.137970e+11
2   4.083790e+11    3.674646e+11
3   3.131799e+11    2.215630e+11
4   1.561175e+11    1.030742e+11
...       ...             ...
```

Plot:

```
losses = pd.DataFrame(model.history.history)
losses.plot()
```

![img calculated losses] ()

Predicting:

```
from sklearn.metrics import mean_squared_error, mean_absolute_error,
explained_variance_score

pred = model.predict(X_test)

pred

array([[432760.72],
       [586596.44],
       [575481.7 ],
       ...,
       [406132.4 ],
       [597030.94],
       [683578.3 ]], dtype=float32)
```

RMSE:

```
np.sqrt(mean_squared_error(y_test,pred))

162532.95927484107
```

MAE:

```
mean_absolute_error(y_test,pred)

100852.6434148341
```

Mean of the house prices:

```
df_99perc['price'].describe()

count    2.159700e+04
mean     5.402966e+05
std      3.673681e+05
min      7.800000e+04
25%      3.220000e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: price, dtype: float64
```

Mean of the house prices = 540.000

RMSE = 163.000

MAE = 101.000

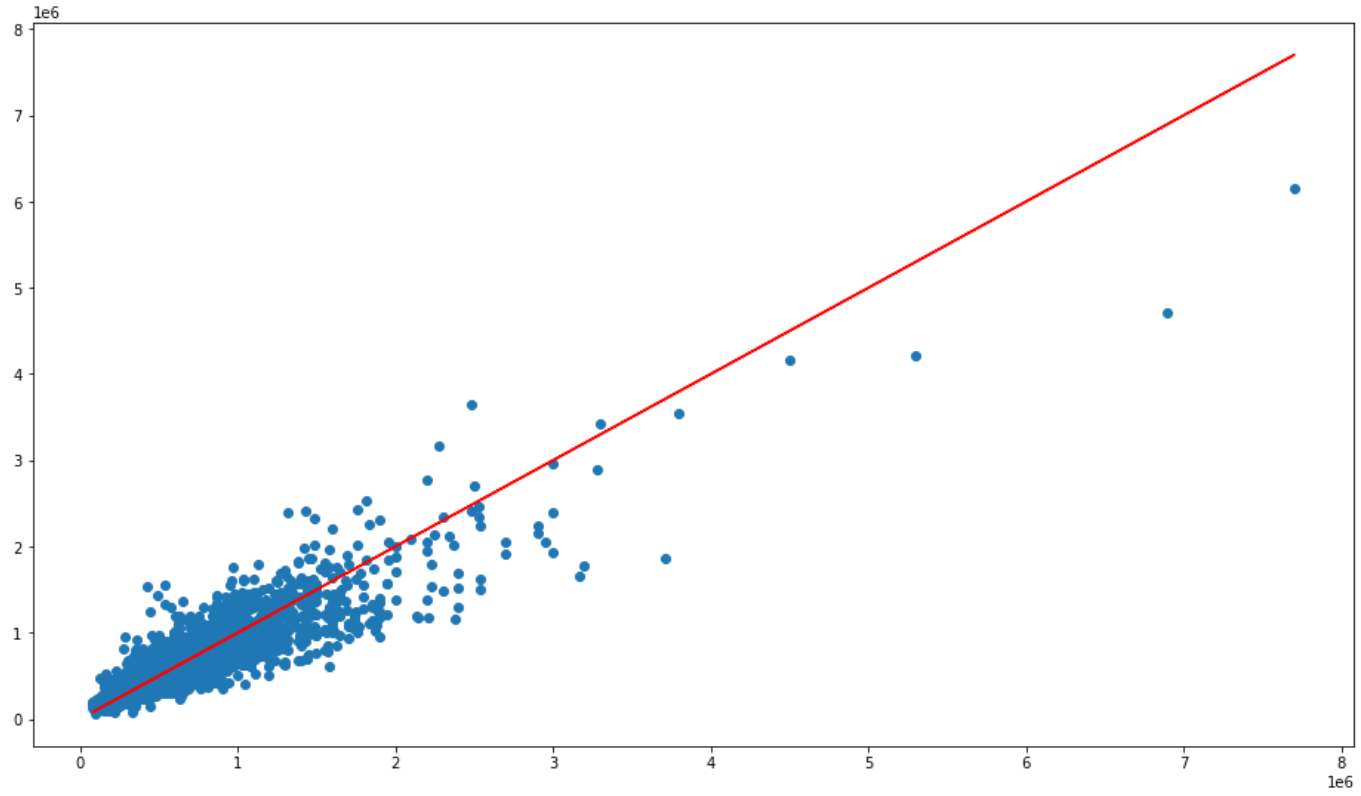
Our model is predicting with 18-20% of error, not good.

How much of the variance is explained by our model:

```
explained_variance_score(y_test,pred)

0.8008035596774933
```

Plotting predictions vs. real values:

```
plt.figure(figsize=(16,9))
plt.scatter(y_test,pred)
plt.plot(y_test,y_test,'r')
```



Conclusion:

Looks like we should train our model again using less than 99% of the houses sale prices or try a multidimensional model.

Calculating a price for a new house:

```
new_house = df_99perc.drop('price',axis=1).iloc[0]

new_house = scaler.transform(new_house.values.reshape(-1,19))

model.predict(new_house)

array([[280802.2]], dtype=float32)
```