```
# data analysis
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# visualization
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# scaling and train test split
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

# creating a model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam

# evaluation on test data
from sklearn.metrics import mean_squared_error,mean_absolute_error,r2_score
from sklearn.metrics import classification_report,confusion_matrix
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df=pd.read_csv('/content/drive/MyDrive/house prediction/kc_house_data.csv')
df.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...

5 rows × 21 columns

```
# No missing 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
```

What are the data types for various features?

Five features are floats, fifteen are integers and one is an object.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

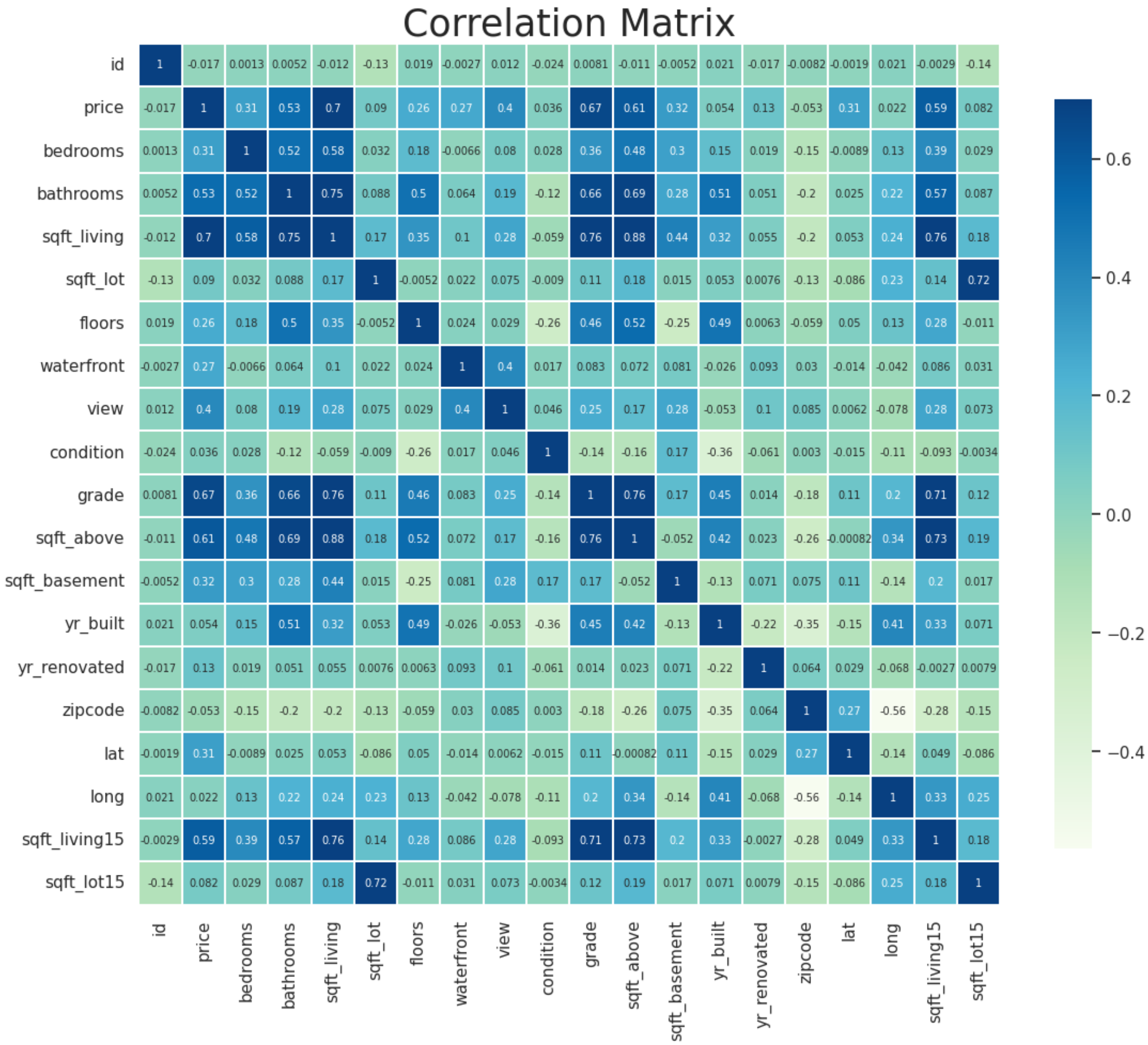
What is the distribution of numerical feature values across the samples?

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	
id	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.90000
price	21613.0	5.400881e+05	3.671272e+05	7.500000e+04	3.219500e+05	4.500000e+05	6.450000e+05	7.70000
bedrooms	21613.0	3.370842e+00	9.300618e-01	0.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.30000
bathrooms	21613.0	2.114757e+00	7.701632e-01	0.000000e+00	1.750000e+00	2.250000e+00	2.500000e+00	8.00000
sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02	1.427000e+03	1.910000e+03	2.550000e+03	1.35400
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02	5.040000e+03	7.618000e+03	1.068800e+04	1.65135
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.50000
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.00000
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.00000
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.00000
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.30000
sqft_above	21613.0	1.788391e+03	8.280910e+02	2.900000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.41000
sqft_basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00	0.000000e+00	0.000000e+00	5.600000e+02	4.82000
yr_built	21613.0	1.971005e+03	2.937341e+01	1.900000e+03	1.951000e+03	1.975000e+03	1.997000e+03	2.01500
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.01500
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04	9.803300e+04	9.806500e+04	9.811800e+04	9.81990
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01	4.747100e+01	4.757180e+01	4.767800e+01	4.77776
long	21613.0	-1.222139e+02	1.408283e-01	-1.225190e+02	-1.223280e+02	-1.222300e+02	-1.221250e+02	-1.21315
sqft_living15	21613.0	1.986552e+03	6.853913e+02	3.990000e+02	1.490000e+03	1.840000e+03	2.360000e+03	6.21000
sqft_lot15	21613.0	1.276846e+04	2.730418e+04	6.510000e+02	5.100000e+03	7.620000e+03	1.008300e+04	8.71200

```
sns.set(style="whitegrid", font_scale=1)

plt.figure(figsize=(13,13))
plt.title('Correlation Matrix',fontsize=25)
sns.heatmap(df.corr(),linewidths=0.25,vmax=0.7,square=True,cmap="GnBu",linecolor='w',
            annot=True, annot_kws={"size":7}, cbar_kws={"shrink": .7})
```

```
<Axes: title={'center': 'Correlation Matrix'}>
```



Price correlation

- This allow us to explore labels that are highly correlated to the price.
- sqft_living looks like a highly correlated label to the price, as well as grade, sqft_above, sqft_living15 and bathrooms

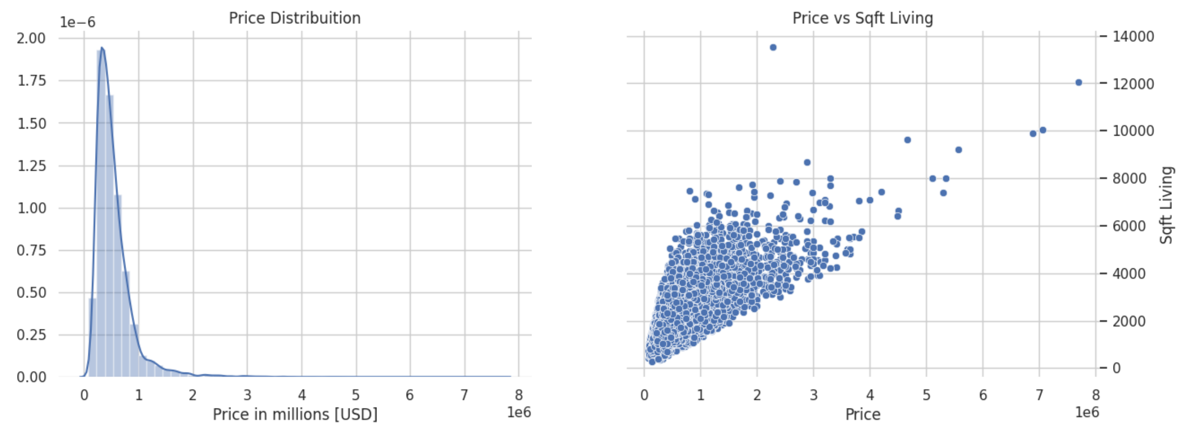
```
price_corr = df.corr()['price'].sort_values(ascending=False)
print(price_corr)
```

```
price            1.000000
sqft_living      0.702035
grade            0.667434
sqft_above       0.605567
sqft_living15    0.585379
bathrooms        0.525138
view             0.397293
sqft_basement    0.323816
bedrooms         0.308350
lat              0.307003
waterfront       0.266369
floors           0.256794
yr_renovated     0.126434
sqft_lot         0.089661
sqft_lot15       0.082447
yr_built         0.054012
condition        0.036362
long             0.021626
id              -0.016762
zipcode         -0.053203
Name: price, dtype: float64
```

Price feature

- Most of the house prices are between $0and$1,500,000.
- The average house price is $540,000.
- Keep in mind that it may be a good idea to drop extreme values. For instance, we could focus on house from $0to$3,000,000 and drop the other ones.
- It seems that there is a positive linear relationship between the price and sqft_living.
- An increase in living space generally corresponds to an increase in house price.

```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.distplot(df['price'], ax=axes[0])
sns.scatterplot(x='price',y='sqft_living', data=df, ax=axes[1])
sns.despine(bottom=True, left=True)
axes[0].set(xlabel='Price in millions [USD]', ylabel='', title='Price Distribuition')
axes[1].set(xlabel='Price', ylabel='Sqft Living', title='Price vs Sqft Living')
axes[1].yaxis.set_label_position("right")
axes[1].yaxis.tick_right()
```



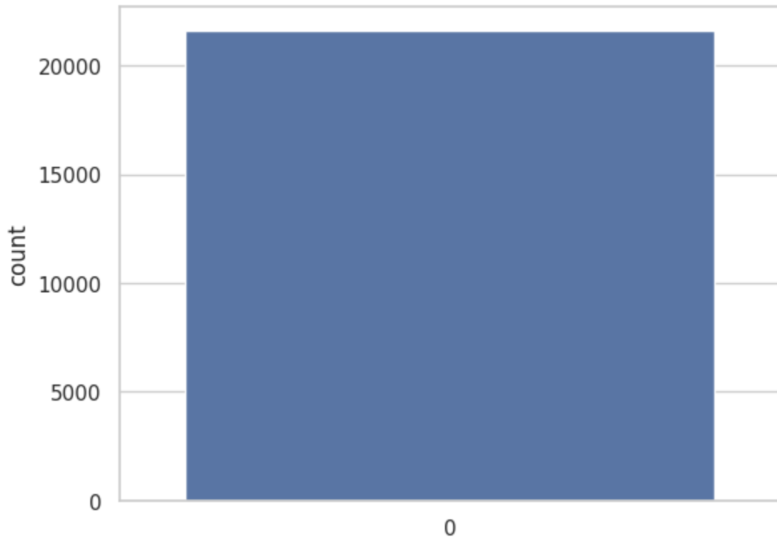
Bedrooms and floors box plots

- We can see outliers plotted as individual points; this probably are the more expensive houses.
- We can see that the price tends to go up when the house has more bedrooms.

```
sns.set(style="whitegrid", font_scale=1)
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=df['bedrooms'],y=df['price'], ax=axes[0])
sns.boxplot(x=df['floors'],y=df['price'], ax=axes[1])
axes[0].set(xlabel='Bedrooms', ylabel='Price', title='Bedrooms vs Price Box Plot')
axes[1].set(xlabel='Floors', ylabel='Price', title='Floors vs Price Box Plot')
```

```
[Text(0.5, 0, 'Floors'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'Floors vs Price Box Plot')]
```


```
sns.countplot(df['bedrooms'])
plt.show()
```

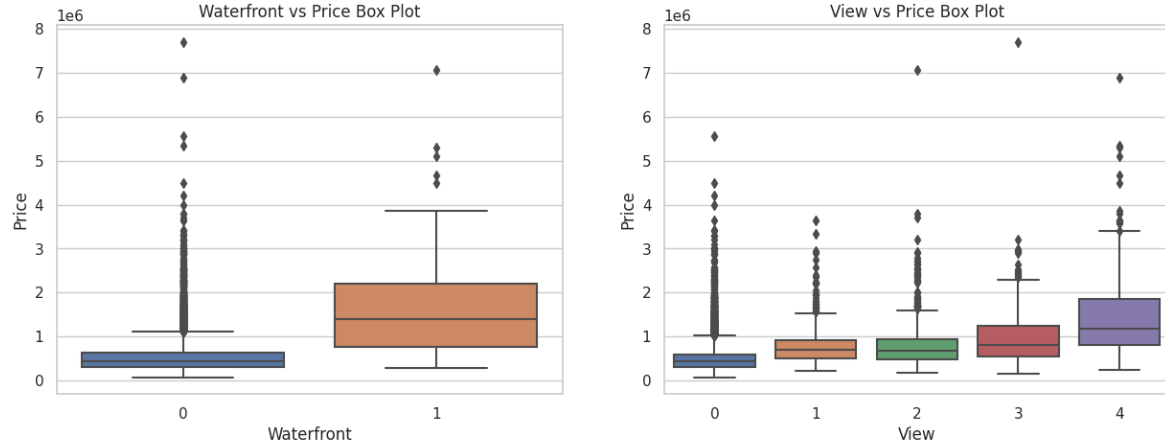


Waterfront, view and grade box plots

- Waterfront houses tends to have a better price value.
- The price of waterfront houses tends to be more disperse and the price of houses without waterfront tend to be more concentrated.
- Grade and waterfront effect price. View seem to effect less but it also has an effect on price

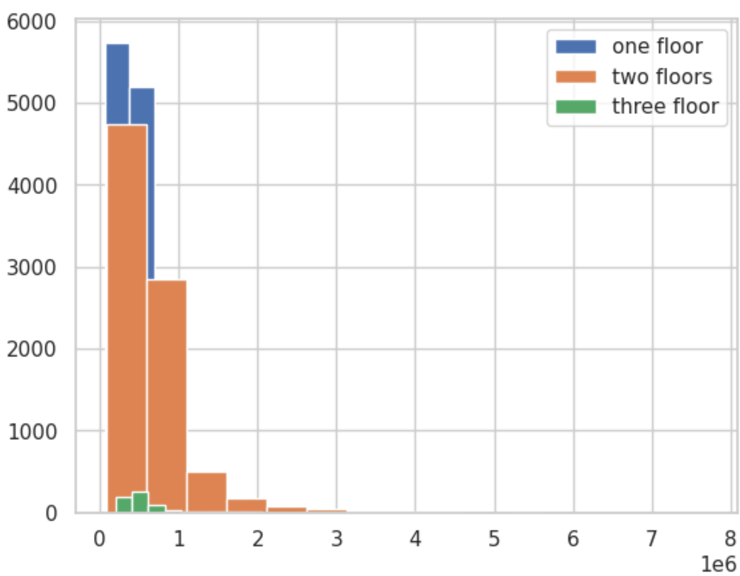
```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=df['waterfront'],y=df['price'], ax=axes[0])
sns.boxplot(x=df['view'],y=df['price'], ax=axes[1])
axes[0].set(xlabel='Waterfront', ylabel='Price', title='Waterfront vs Price Box Plot')
axes[1].set(xlabel='View', ylabel='Price', title='View vs Price Box Plot')

f, axe = plt.subplots(1, 1,figsize=(15,5))
sns.boxplot(x=df['grade'],y=df['price'], ax=axe)
axe.set(xlabel='Grade', ylabel='Price', title='Grade vs Price Box Plot')
```

```
[Text(0.5, 0, 'Grade'),
 Text(0, 0.5, 'Price'),
 Text(0.5, 1.0, 'Grade vs Price Box Plot')]
```

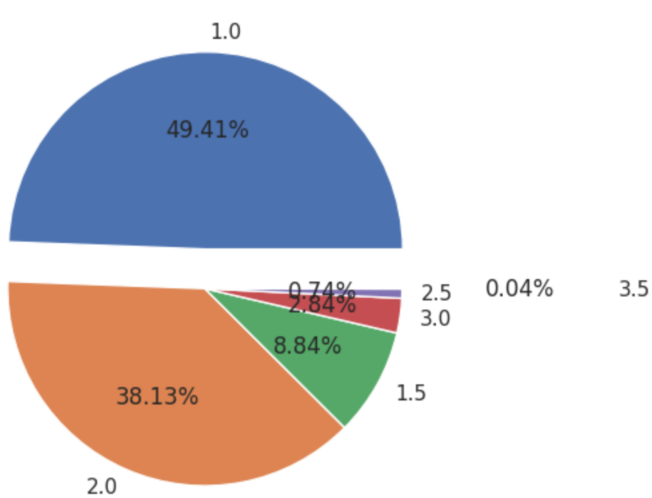



```
one = plt.hist(df[(df.floors == 1) | (df.floors ==1.5) ].price ,bins =15 ,label = "one floor")
two = plt.hist(df[(df.floors == 2) | (df.floors == 2.5)].price ,bins =15 ,label = "two floors")
three = plt.hist(df[(df.floors ==3) | (df.floors == 3.5)].price ,bins =15 ,label = "three floor")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb6280b7460>
```

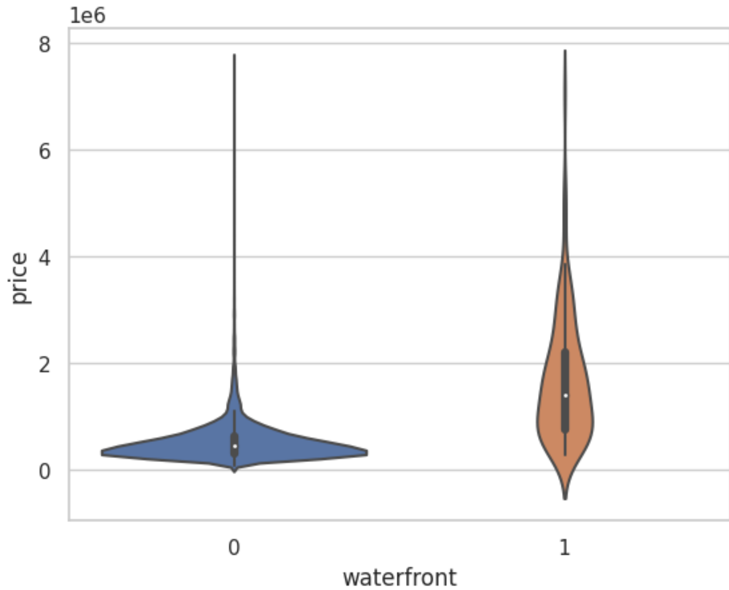



```
plt.pie(df.floors.value_counts(normalize =True) , explode =[0.2,0,0,0,0,1] ,labels =df.floors.value_counts().index,
         autopct ="%.2f%%"
        )
plt.show()
```



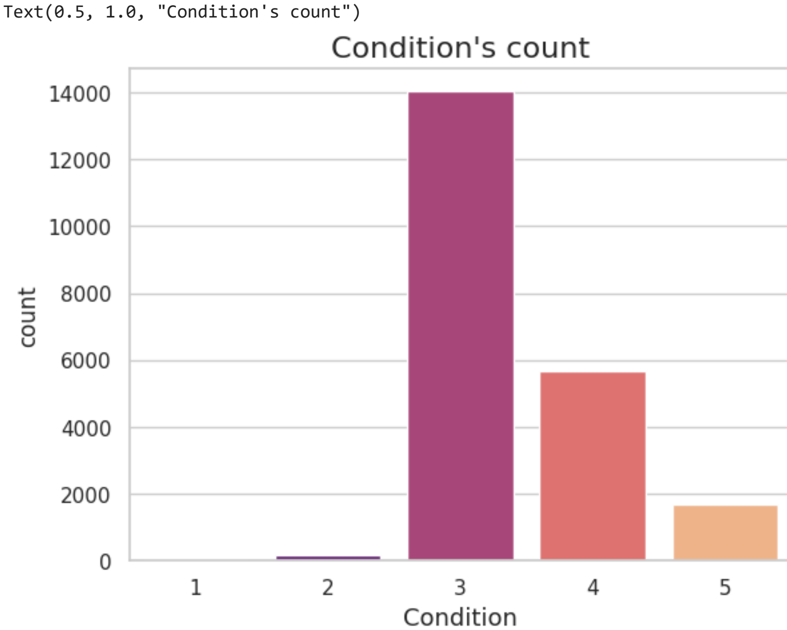

```
sns.violinplot(data =df,x = "waterfront" ,y ="price")
```

<Axes: xlabel='waterfront', ylabel='price'>




```
sns.countplot(x='condition', data=df, palette='magma')
plt.xlabel('Condition', fontsize=13)
plt.title("Condition's count", fontsize=16)
```

Text(0.5, 1.0, "Condition's count")




```
df = df.drop(['id','zipcode'], axis=1)
```

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

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

df = df.drop('date',axis=1)

# Check the new columns
print(df.columns.values)
```

```
['price' 'bedrooms' 'bathrooms' 'sqft_living' 'sqft_lot' 'floors'
 'waterfront' 'view' 'condition' 'grade' 'sqft_above' 'sqft_basement'
 'yr_built' 'yr_renovated' 'lat' 'long' 'sqft_living15' 'sqft_lot15'
 'month' 'year']
```

House price trends

- Looking the box plots we notice that there is not a big difference between 2014 and 2015.
- The number of houses sold by month tends to be similar every month.
- The line plot show that around April there is an increase in house prices.

```
X = df.drop('price',axis=1)

# Label
y = df['price']

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

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(15129, 19)
(6484, 19)
(15129,)
(6484,)
```

```
scaler = MinMaxScaler()

# fit and transfrom
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
model = Sequential()
y_size,x_size = X_train.shape
model.add(Dense(x_size,activation='relu')) # x_size - 64 - 64 - 128 - 64 - 64 - x_size
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(x_size,activation='relu'))
model.add(Dense(1)) # since we want only one feature as outcome (price) I added 1 as last dense
```

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

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

```
119/119 [==============================] - 1s 4ms/step - loss: 63535.7773 - val_loss: 67579.9609
Epoch 392/400
119/119 [==============================] - 1s 4ms/step - loss: 63650.1211 - val_loss: 67434.2344
Epoch 393/400
119/119 [==============================] - 1s 4ms/step - loss: 62987.6992 - val_loss: 68243.4766
Epoch 394/400
119/119 [==============================] - 0s 4ms/step - loss: 63194.1719 - val_loss: 66734.5781
Epoch 395/400
119/119 [==============================] - 1s 4ms/step - loss: 63148.0078 - val_loss: 66881.0547
Epoch 396/400
119/119 [==============================] - 0s 4ms/step - loss: 63796.0312 - val_loss: 70100.3125
Epoch 397/400
119/119 [==============================] - 1s 4ms/step - loss: 62828.8398 - val_loss: 67862.0625
Epoch 398/400
119/119 [==============================] - 0s 4ms/step - loss: 62888.4414 - val_loss: 67653.5156
Epoch 399/400
119/119 [==============================] - 1s 4ms/step - loss: 63058.4258 - val_loss: 66761.6250
Epoch 400/400
119/119 [==============================] - 0s 4ms/step - loss: 63954.6250 - val_loss: 69320.8672
```

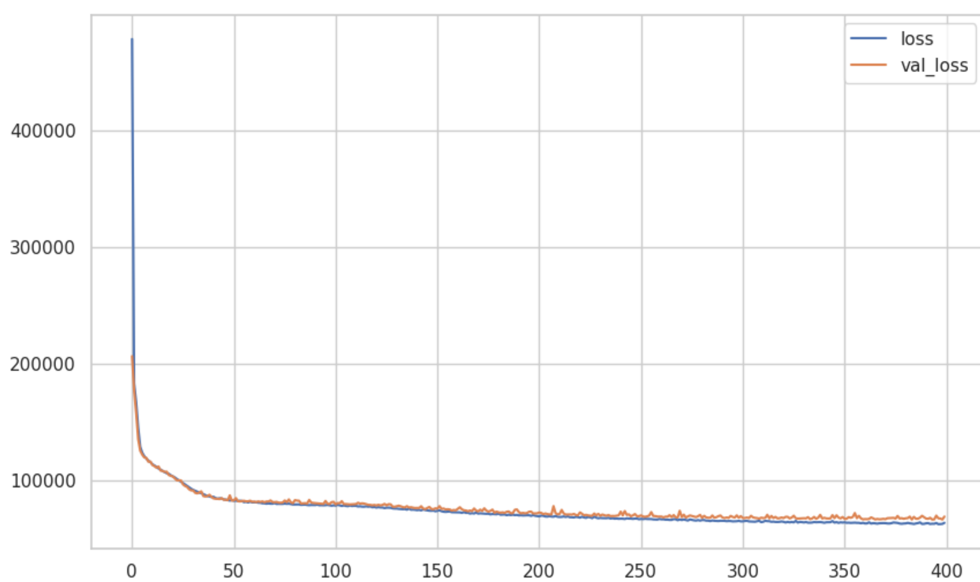
```
pd.DataFrame(history.history)
```

	loss	val_loss
0	477793.062500	206515.953125
1	183283.031250	177072.343750
2	167162.265625	158134.875000
3	145236.718750	135565.812500
4	129024.015625	125616.851562
...	...	...
395	63796.031250	70100.312500
396	62828.839844	67862.062500
397	62888.441406	67653.515625
398	63058.425781	66761.625000
399	63954.625000	69320.867188

400 rows × 2 columns

```
pd.DataFrame(history.history).plot(figsize=(10,6))
```

```
<Axes: >
```




```
predictions=model.predict(X_test)
```

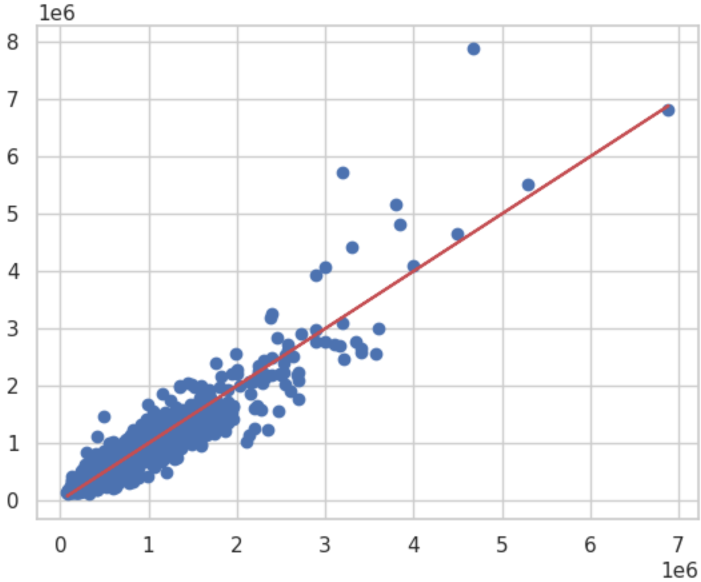
```
203/203 [==============================] - 0s 1ms/step
```

```
print("The absolute mean error :",mean_absolute_error(y_test, predictions))
print("The r2_score :",r2_score(y_test, predictions))
```

```
The absolute mean error : 69320.8604015654
```

```
plt.scatter(y_test,predictions)
plt.plot(y_test,y_test,'r')
```

```
[<matplotlib.lines.Line2D at 0x7fb6189e0970>]
```




```
pd.DataFrame({"Actual value":y_test.astype(int),"predicted value":predictions.flatten().astype(int)})
```

	Actual value	predicted value
3834	349950	256503
1348	450000	434910
20366	635000	795017
16617	355500	336544
20925	246950	258052
...	...	...
1398	465000	421664
3364	418000	368092
18958	394250	350360
17845	249500	201135
16335	1350000	1213212

6484 rows × 2 columns