In [4]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as TF
from matplotlib.pyplot import figure
import pylab as pl
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

In [5]:

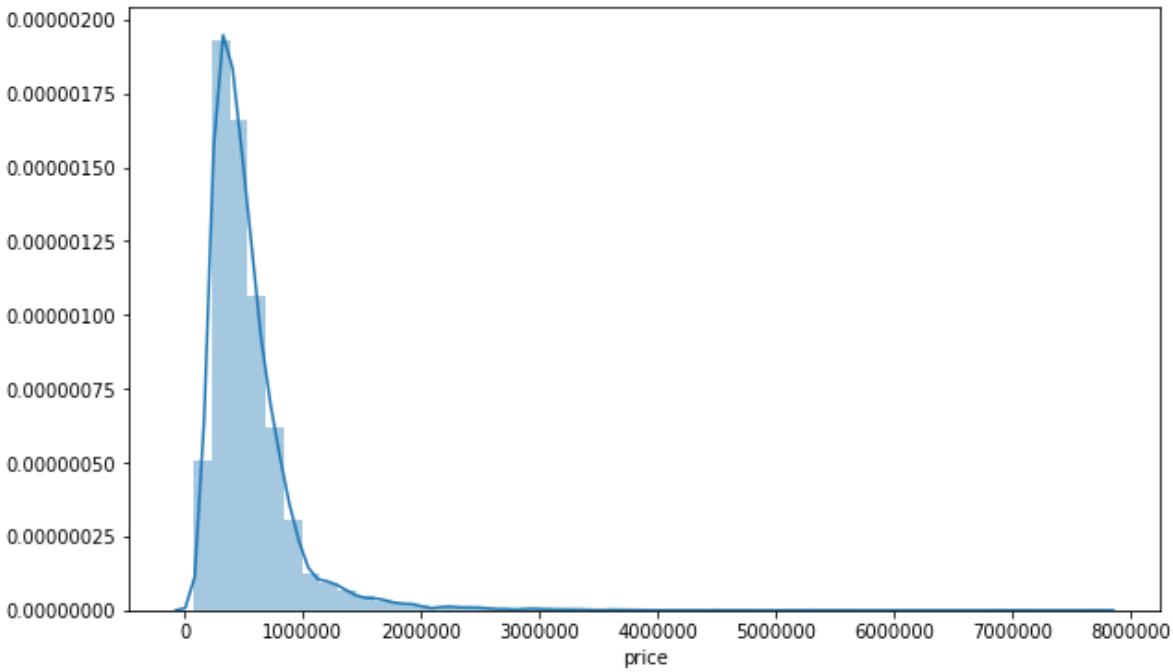
```
data1 = pd.read_csv('C:/Users/affaa/OneDrive/Dokumen/Latihan programming/TF_2_Notebooks_
```

In [8]:

```
plt.figure(figsize=(10,6))
sns.distplot(data1['price'])
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22835efe2c8>
```

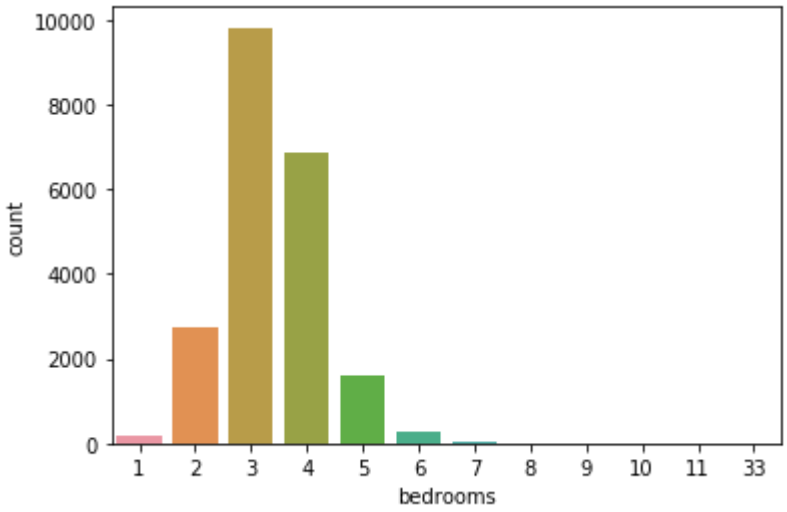


In [22]:

```
sns.countplot(data1['bedrooms'])
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x28bbb042848>
```



In [23]:

```
data1.corr()['price'].sort_values()
```

Out[23]:

```
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
```

In [9]:

```
from sklearn.model_selection import train_test_split
```

In [36]:

```
help(train_test_split)
```

```
Help on function train_test_split in module sklearn.model_selection._spli
t:

train_test_split(*arrays, **options)
    Split arrays or matrices into random train and test subsets

    Quick utility that wraps input validation and
    ``next(ShuffleSplit().split(X, y))`` and application to input data
    into a single call for splitting (and optionally subsampling) data in
a
    oneliner.

    Read more in the :ref:`User Guide <cross_validation>`.

    Parameters
    ----------
    *arrays : sequence of indexables with same length / shape[0]
        Allowed inputs are lists, numpy arrays, scipy-sparse
        matrices or pandas dataframes.

    test_size : float, int or None, optional (default=None)
        If float, should be between 0.0 and 1.0 and represent the proporti
on
        of the dataset to include in the test split. If int, represents th
e
        absolute number of test samples. If None, the value is set to the
        complement of the train size. If ``train_size`` is also None, it w
ill
        be set to 0.25.

    train_size : float, int, or None, (default=None)
        If float, should be between 0.0 and 1.0 and represent the
        proportion of the dataset to include in the train split. If
        int, represents the absolute number of train samples. If None,
        the value is automatically set to the complement of the test size.

    random_state : int, RandomState instance or None, optional (default=No
ne)
        If int, random_state is the seed used by the random number generat
or;
        If RandomState instance, random_state is the random number generat
or;
        If None, the random number generator is the RandomState instance u
sed
        by `np.random`.

    shuffle : boolean, optional (default=True)
        Whether or not to shuffle the data before splitting. If shuffle=Fa
lse
        then stratify must be None.

    stratify : array-like or None (default=None)
        If not None, data is split in a stratified fashion, using this as
        the class labels.

    Returns
    -------
```

```
    splitting : list, length=2 * len(arrays)
        List containing train-test split of inputs.

        .. versionadded:: 0.16
            If the input is sparse, the output will be a
            ``scipy.sparse.csr_matrix``. Else, output type is the same as
the
            input type.

    Examples
    --------
    >>> import numpy as np
    >>> from sklearn.model_selection import train_test_split
    >>> X, y = np.arange(10).reshape((5, 2)), range(5)
    >>> X
    array([[0, 1],
           [2, 3],
           [4, 5],
           [6, 7],
           [8, 9]])
    >>> list(y)
    [0, 1, 2, 3, 4]

    >>> X_train, X_test, y_train, y_test = train_test_split(
    ...     X, y, test_size=0.33, random_state=42)
    ...
    >>> X_train
    array([[4, 5],
           [0, 1],
           [6, 7]])
    >>> y_train
    [2, 0, 3]
    >>> X_test
    array([[2, 3],
           [8, 9]])
    >>> y_test
    [1, 4]

    >>> train_test_split(y, shuffle=False)
    [[0, 1, 2], [3, 4]]
```

In [10]:

```
data1 = data1.drop('id',axis=1)
data1['date'] = pd.to_datetime(data1['date'])
data1['year'] = data1['date'].apply(lambda date:date.year)
data1['month'] = data1['date'].apply(lambda date:date.month)
data1 = data1.drop('date',axis=1)
data1 = data1.drop('zipcode',axis=1)
```

In [21]:

```
X = data1.drop('price',axis=1).values
y = data1['price'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Out[21]:

```
array([[3.000e+00, 1.000e+00, 1.180e+03, ..., 5.650e+03, 2.014e+03,
        1.000e+01],
       [3.000e+00, 2.250e+00, 2.570e+03, ..., 7.639e+03, 2.014e+03,
        1.200e+01],
       [2.000e+00, 1.000e+00, 7.700e+02, ..., 8.062e+03, 2.015e+03,
        2.000e+00],
       ...,
       [2.000e+00, 7.500e-01, 1.020e+03, ..., 2.007e+03, 2.014e+03,
        6.000e+00],
       [3.000e+00, 2.500e+00, 1.600e+03, ..., 1.287e+03, 2.015e+03,
        1.000e+00],
       [2.000e+00, 7.500e-01, 1.020e+03, ..., 1.357e+03, 2.014e+03,
        1.000e+01]])
```

In [41]:

```
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')
```

In [42]:

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

```
Train on 15117 samples, validate on 6480 samples
Epoch 1/400
15117/15117 [==============================] - 1s 74us/sample - loss: 43
0241691427.5796 - val_loss: 418920676526.4593
Epoch 2/400
15117/15117 [==============================] - 0s 20us/sample - loss: 42
9100469091.4568 - val_loss: 414854062949.7679
Epoch 3/400
15117/15117 [==============================] - 0s 20us/sample - loss: 41
2614656548.4093 - val_loss: 377063755523.1605
Epoch 4/400
15117/15117 [==============================] - 0s 20us/sample - loss: 33
2803225956.1680 - val_loss: 250119244923.8914
Epoch 5/400
15117/15117 [==============================] - 0s 21us/sample - loss: 18
1072001864.6324 - val_loss: 114358537036.4839
Epoch 6/400
15117/15117 [==============================] - 0s 21us/sample - loss: 10
3722396475.7621 - val_loss: 95523096186.6272
```

In [45]:

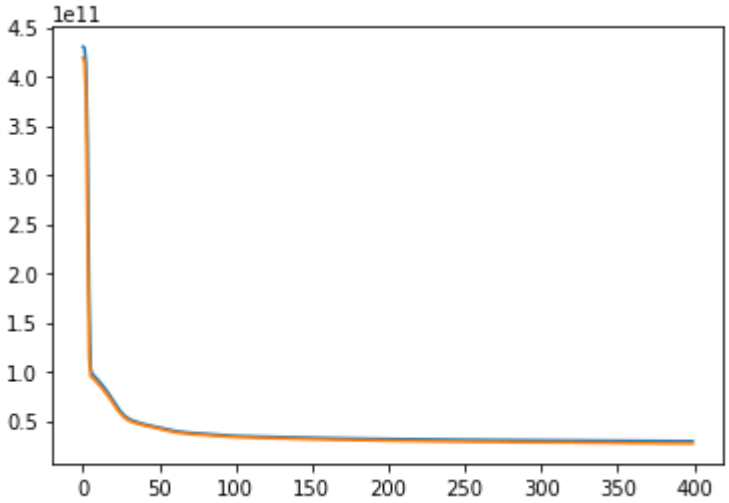
```
losses = pd.DataFrame(model.history.history)
```

In [51]:

```
plt.plot(losses)
```

Out[51]:

```
[<matplotlib.lines.Line2D at 0x28bbd65dcc8>,
 <matplotlib.lines.Line2D at 0x28bbd692888>]
```



In [53]:

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

In [58]:

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

In [60]:

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

Out[60]:

```
164041.6853620155
```

In [61]:

```
mean_absolute_error(y_test,prediction)
```

Out[61]:

```
100858.39065634646
```

In [62]:

```
data1['price'].describe()
```

Out[62]:

```
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
```

In [63]:

```
explained_variance_score(y_test,prediction)
```

Out[63]:

```
0.7971299934108212
```

In [64]:

```
test_house = data1.drop('price',axis=1).iloc[0]
```

In [66]:

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

In [67]:

```
model.predict(test_house)
```

Out[67]:

```
array([[285817.72]], dtype=float32)
```

In [ ]:

```

```