```
# 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
```

```
df=pd.read_csv('kc_house_data.csv')
df.head()
```

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

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
```

```
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
```

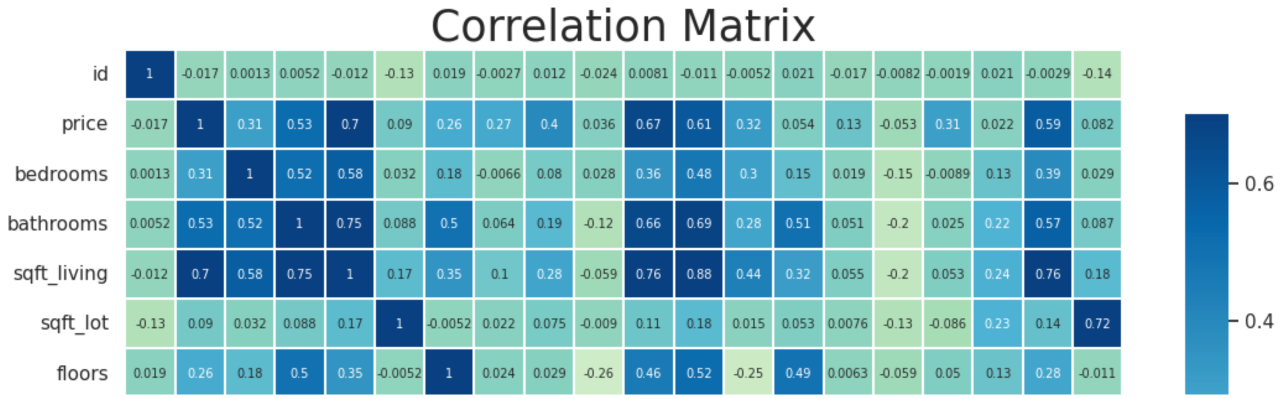
```
# Five features are floats, fifteen are integers and one is an object.
```

```
# Statistical distribution of dataset
df.describe()
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	21613.000000	21
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	0.234303	
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	0.766318	
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	0.000000	
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	0.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	0.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	0.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	4.000000	

```
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

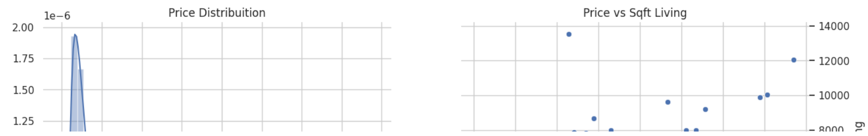
```
price_corr = df.corr()['price'].sort_values(ascending=False)
print(price_corr)
```

```
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
```

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 feature

```
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()
```

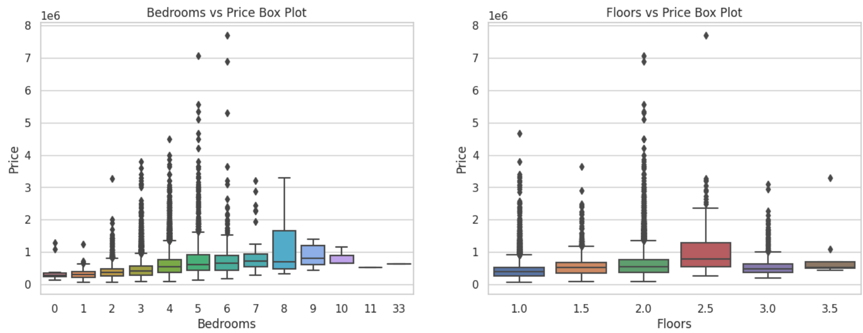


- Most of the house prices are between $0and$1,500,000.
- The average house price is $540,000.
- Keep in mind that it may be a good idea to drop extreme values. For instance, we could focus on house from $0to$3,000,000 and drop the other ones.
- It seems that there is a positive linear relationship between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

Bedrooms and floors box plots

```
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')]
```



We can see outliers plotted as individual points; this probably are the more expensive houses.

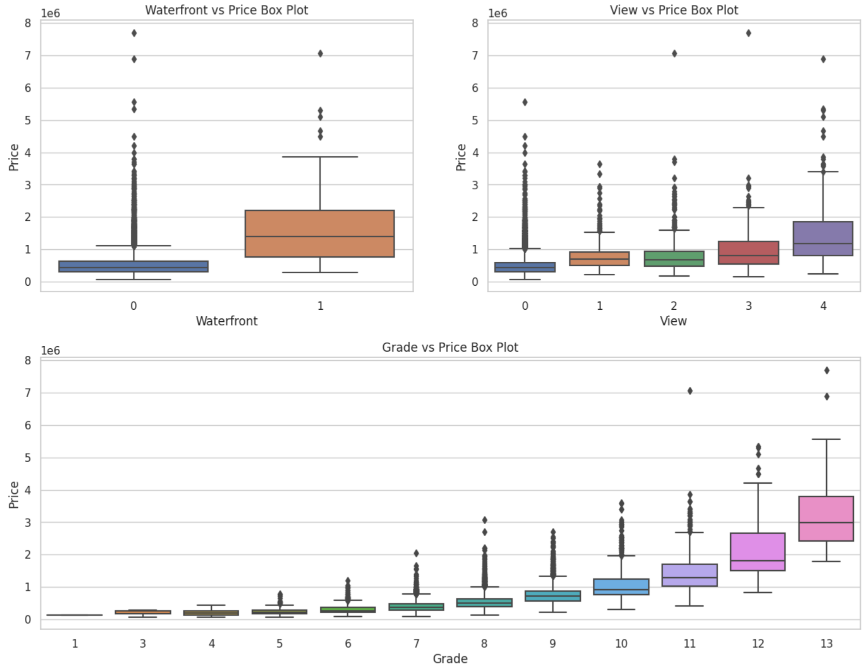
We can see that the price tends to go up when the house has more bedrooms.

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

Waterfront, view and grade box plots

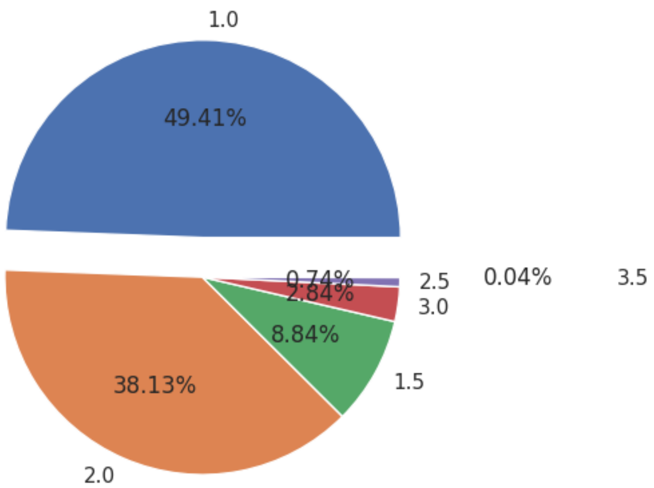
```
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.boxplot(x=df['waterfront'],y=df['price'], ax=axes[0])
sns.boxplot(x=df['view'],y=df['price'], ax=axes[1])
axes[0].set(xlabel='Waterfront', ylabel='Price', title='Waterfront vs Price Box Plot')
axes[1].set(xlabel='View', ylabel='Price', title='View vs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



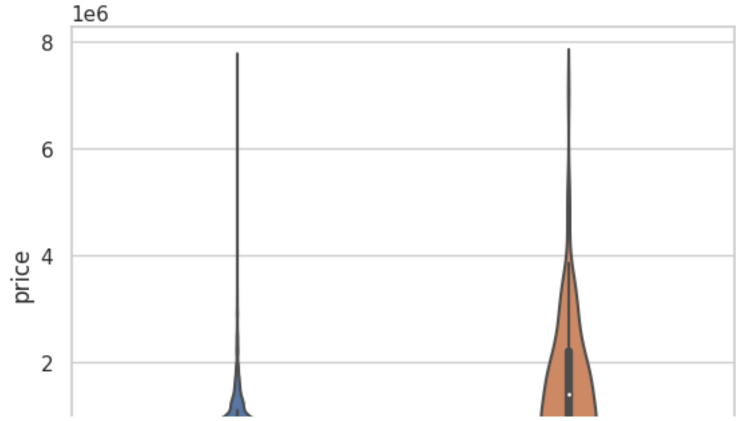
- Waterfront houses tends to have a better price value.
- The price of waterfront houses tends to be more disperse and the price of houses without waterfront tend to be more concentrated.
- Grade and waterfront effect price. View seem to effect less but it also has an effect on price

```
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()
```



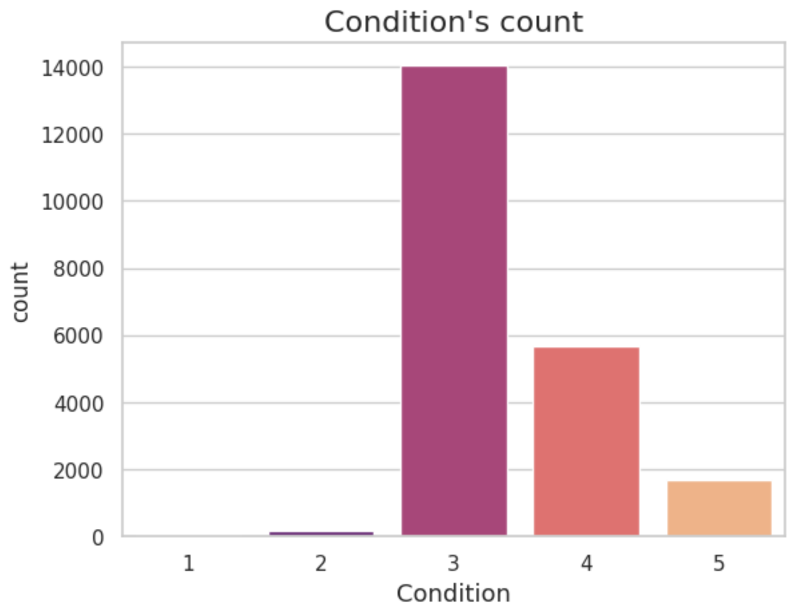

```
sns.violinplot(data =df,x = "waterfront" ,y ="price")
```

```
<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']
```

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

# Label
y = df['price']

# Spliting data into train_test_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,)
```

```
# Scalling
scaler = MinMaxScaler()

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

Building the ANN MOdel

```
# Step1: initialize the model
model = Sequential()
y_size,x_size = X_train.shape

# Step2: add layers into model
# Add Hidden layers
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'))

# Add output layer
model.add(Dense(1)) # since we want only one feature as outcome (price) I added 1 as last dense
```

```
# Step 3: established connection between the layers
model.compile(optimizer='adam',loss='mae')
```

```
# Step4: Train the model
history = model.fit(x= X_train,y= y_train,batch_size=128,epochs=200,validation_data=(X_test, y_test))
```

```
Epoch 1/200
119/119 [==============================] - 2s 6ms/step - loss: 493782.8750 - val_loss: 211599.7344
Epoch 2/200
119/119 [==============================] - 0s 3ms/step - loss: 186540.5625 - val_loss: 177971.2500
Epoch 3/200
119/119 [==============================] - 0s 3ms/step - loss: 168054.9062 - val_loss: 159044.8438
Epoch 4/200
119/119 [==============================] - 1s 4ms/step - loss: 143690.3438 - val_loss: 131721.4219
Epoch 5/200
119/119 [==============================] - 0s 4ms/step - loss: 127251.2891 - val_loss: 124610.2344
Epoch 6/200
119/119 [==============================] - 0s 4ms/step - loss: 122918.5156 - val_loss: 121547.6562
Epoch 7/200
119/119 [==============================] - 1s 5ms/step - loss: 119845.7578 - val_loss: 120129.3047
Epoch 8/200
119/119 [==============================] - 0s 3ms/step - loss: 118515.5781 - val_loss: 117178.2812
Epoch 9/200
119/119 [==============================] - 0s 3ms/step - loss: 116714.0000 - val_loss: 115825.2344
Epoch 10/200
119/119 [==============================] - 0s 3ms/step - loss: 115931.0781 - val_loss: 114656.7734
Epoch 11/200
119/119 [==============================] - 0s 3ms/step - loss: 114622.6016 - val_loss: 113709.8359
Epoch 12/200
119/119 [==============================] - 0s 3ms/step - loss: 113761.0703 - val_loss: 113510.8203
Epoch 13/200
119/119 [==============================] - 0s 3ms/step - loss: 112819.9922 - val_loss: 112468.8750
Epoch 14/200
119/119 [==============================] - 0s 3ms/step - loss: 111818.5547 - val_loss: 112120.7578
Epoch 15/200
119/119 [==============================] - 0s 3ms/step - loss: 111455.5234 - val_loss: 110779.8047
Epoch 16/200
119/119 [==============================] - 0s 3ms/step - loss: 110867.8516 - val_loss: 110243.6484
Epoch 17/200
119/119 [==============================] - 0s 3ms/step - loss: 110395.8750 - val_loss: 109971.7031
Epoch 18/200
119/119 [==============================] - 0s 3ms/step - loss: 110152.6250 - val_loss: 109090.5078
Epoch 19/200
119/119 [==============================] - 0s 3ms/step - loss: 109156.6016 - val_loss: 108846.0859
Epoch 20/200
119/119 [==============================] - 0s 3ms/step - loss: 109067.1875 - val_loss: 109142.1953
Epoch 21/200
119/119 [==============================] - 0s 3ms/step - loss: 109030.4688 - val_loss: 108153.3984
Epoch 22/200
119/119 [==============================] - 0s 3ms/step - loss: 108349.8203 - val_loss: 107553.0234
Epoch 23/200
```

```
119/119 [==============================] - 0s 3ms/step - loss: 107833.4531 - val_loss: 108202.8438
Epoch 24/200
119/119 [==============================] - 0s 3ms/step - loss: 107330.5703 - val_loss: 107632.7031
Epoch 25/200
119/119 [==============================] - 0s 3ms/step - loss: 107432.2422 - val_loss: 107421.3828
Epoch 26/200
119/119 [==============================] - 0s 3ms/step - loss: 107101.2031 - val_loss: 106260.4219
Epoch 27/200
119/119 [==============================] - 0s 3ms/step - loss: 106829.3828 - val_loss: 106031.3516
Epoch 28/200
119/119 [==============================] - 0s 3ms/step - loss: 106394.7109 - val_loss: 105809.5312
Epoch 29/200
```

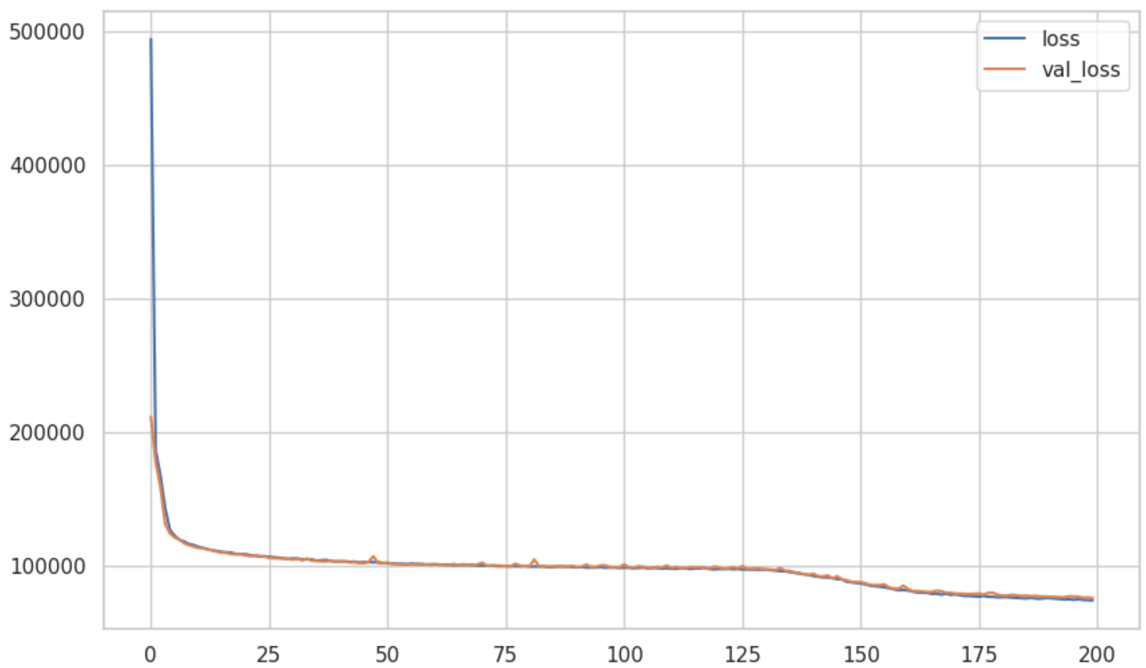
```
pd.DataFrame(history.history)
```

	loss	val_loss
0	493782.875000	211599.734375
1	186540.562500	177971.250000
2	168054.906250	159044.843750
3	143690.343750	131721.421875
4	127251.289062	124610.234375
...	...	...
195	74984.273438	77532.156250
196	75395.921875	77214.929688
197	74880.523438	76616.492188
198	74692.406250	76711.960938
199	74538.242188	76399.742188

200 rows × 2 columns

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

```
<Axes: >
```




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

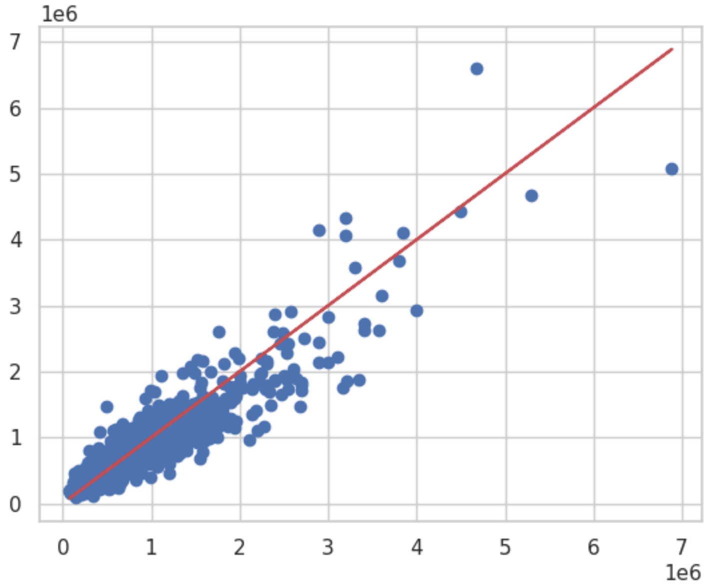
```
203/203 [==============================] - 0s 914us/step
```

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

```
The absolute mean error : 76399.72847947833
The r2_score : 0.8659049643655456
```

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

```
[<matplotlib.lines.Line2D at 0x7f9f2c776c70>]
```



Early Stopping Concept

To check the after using Early Stopping Concept,accuracy of the model is same or not

or if we get best accuracy in ann model then we check the early stop accuracy

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=40)


model1=Sequential()


model1.add(Dense(70,activation='relu'))
model1.add(Dense(70,activation='relu'))
model1.add(Dense(1))


model1.compile(optimizer='adam',loss='mae',metrics=['accuracy'])
model1.fit(X_train,y_train,epochs=1000,validation_data=(X_test,y_test),verbose=1,batch_size=25,callbacks=[early_stop])
```

```
Epoch 1/1000
606/606 [==============================] - 2s 2ms/step - loss: 533170.8750 - accuracy: 0.0000e+00 - val_loss: 514291.3750 - val_a
Epoch 2/1000
606/606 [==============================] - 1s 2ms/step - loss: 441782.5000 - accuracy: 0.0000e+00 - val_loss: 337797.0000 - val_a
Epoch 3/1000
606/606 [==============================] - 1s 2ms/step - loss: 236755.5781 - accuracy: 0.0000e+00 - val_loss: 193872.2656 - val_a
Epoch 4/1000
606/606 [==============================] - 1s 2ms/step - loss: 188630.8750 - accuracy: 0.0000e+00 - val_loss: 188453.9219 - val_a
Epoch 5/1000
606/606 [==============================] - 1s 2ms/step - loss: 185820.0625 - accuracy: 0.0000e+00 - val_loss: 186066.7188 - val_a
Epoch 6/1000
606/606 [==============================] - 1s 2ms/step - loss: 183351.7344 - accuracy: 0.0000e+00 - val_loss: 183587.9688 - val_a
Epoch 7/1000
606/606 [==============================] - 1s 2ms/step - loss: 180789.3906 - accuracy: 0.0000e+00 - val_loss: 180921.4531 - val_a
Epoch 8/1000
606/606 [==============================] - 1s 2ms/step - loss: 178089.0781 - accuracy: 0.0000e+00 - val_loss: 178107.8750 - val_a
Epoch 9/1000
606/606 [==============================] - 1s 2ms/step - loss: 175200.0312 - accuracy: 0.0000e+00 - val_loss: 175155.7969 - val_a
Epoch 10/1000
606/606 [==============================] - 1s 2ms/step - loss: 172242.3906 - accuracy: 0.0000e+00 - val_loss: 172119.0312 - val_a
Epoch 11/1000
606/606 [==============================] - 1s 2ms/step - loss: 169166.6562 - accuracy: 0.0000e+00 - val_loss: 169000.5312 - val_a
Epoch 12/1000
606/606 [==============================] - 1s 2ms/step - loss: 166034.4688 - accuracy: 0.0000e+00 - val_loss: 165699.5781 - val_a
Epoch 13/1000
606/606 [==============================] - 1s 2ms/step - loss: 162804.3594 - accuracy: 0.0000e+00 - val_loss: 162347.5938 - val_a
Epoch 14/1000
606/606 [==============================] - 1s 2ms/step - loss: 159494.1406 - accuracy: 0.0000e+00 - val_loss: 158871.0312 - val_a
Epoch 15/1000
606/606 [==============================] - 1s 2ms/step - loss: 156088.3750 - accuracy: 0.0000e+00 - val_loss: 155421.7812 - val_a
Epoch 16/1000
606/606 [==============================] - 1s 2ms/step - loss: 152568.3594 - accuracy: 0.0000e+00 - val_loss: 151679.2656 - val_a
Epoch 17/1000
606/606 [==============================] - 1s 2ms/step - loss: 148976.8438 - accuracy: 0.0000e+00 - val_loss: 148019.2031 - val_a
```

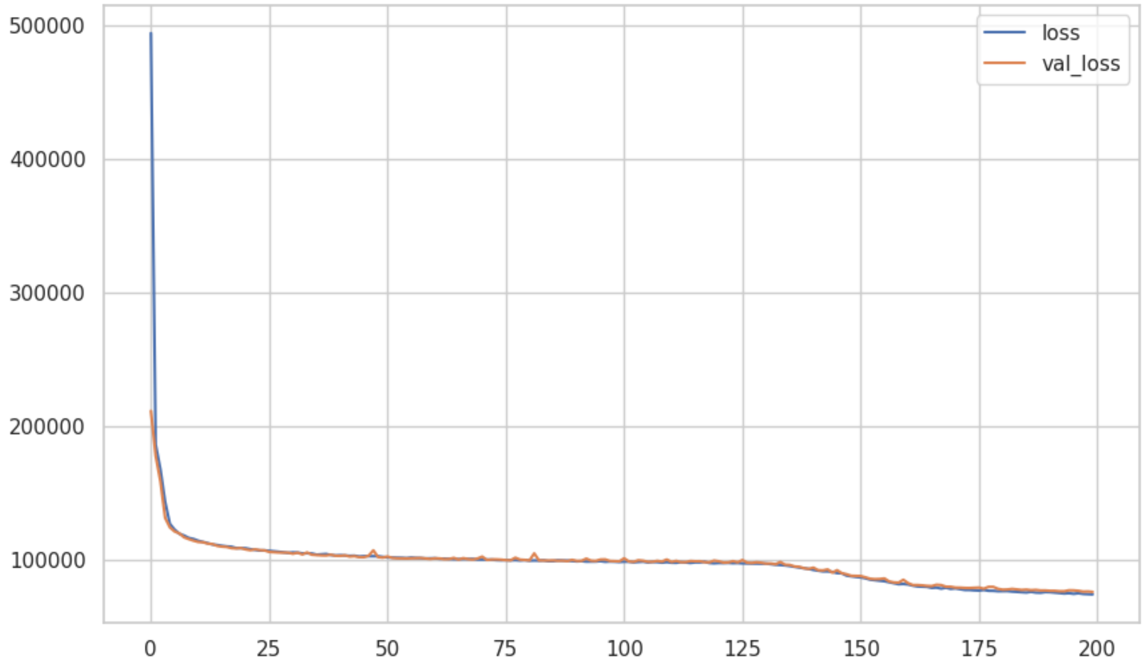
```
Epoch 18/1000
606/606 [==============================] - 1s 2ms/step - loss: 145471.2500 - accuracy: 0.0000e+00 - val_loss: 144504.5625 - val_a
Epoch 19/1000
606/606 [==============================] - 1s 2ms/step - loss: 142214.3906 - accuracy: 0.0000e+00 - val_loss: 141332.1875 - val_a
Epoch 20/1000
606/606 [==============================] - 1s 2ms/step - loss: 139262.5938 - accuracy: 0.0000e+00 - val_loss: 138519.7344 - val_a
Epoch 21/1000
606/606 [==============================] - 1s 2ms/step - loss: 136820.2344 - accuracy: 0.0000e+00 - val_loss: 136298.4219 - val_a
Epoch 22/1000
606/606 [==============================] - 1s 2ms/step - loss: 134847.2500 - accuracy: 0.0000e+00 - val_loss: 134505.9531 - val_a
Epoch 23/1000
606/606 [==============================] - 1s 2ms/step - loss: 133275.9375 - accuracy: 0.0000e+00 - val_loss: 133021.6562 - val_a
Epoch 24/1000
606/606 [==============================] - 1s 2ms/step - loss: 132009.0625 - accuracy: 0.0000e+00 - val_loss: 131795.0625 - val_a
Epoch 25/1000
606/606 [==============================] - 1s 2ms/step - loss: 130960.6641 - accuracy: 0.0000e+00 - val_loss: 130825.9219 - val_a
Epoch 26/1000
606/606 [==============================] - 1s 2ms/step - loss: 130070.3359 - accuracy: 0.0000e+00 - val_loss: 129878.2344 - val_a
Epoch 27/1000
606/606 [==============================] - 1s 2ms/step - loss: 129297.1094 - accuracy: 0.0000e+00 - val_loss: 129213.5000 - val_a
Epoch 28/1000
606/606 [==============================] - 1s 2ms/step - loss: 128653.7969 - accuracy: 0.0000e+00 - val_loss: 128537.2969 - val_a
```

```
model1.history.history
```

```
{'loss': [533170.875,
  441782.5,
  236755.578125,
  188630.875,
  185820.0625,
  183351.734375,
  180789.390625,
  178089.078125,
  175200.03125,
  172242.390625,
  169166.65625,
  166034.46875,
  162804.359375,
  159494.140625,
  156088.375,
  152568.359375,
  148976.84375,
  145471.25,
  142214.390625,
  139262.59375,
  136820.234375,
  134847.25,
  133275.9375,
  132009.0625,
  130960.6640625,
  130070.3359375,
  129297.109375,
  128653.796875,
  128052.859375,
  127535.6953125,
  127081.3125,
  126654.6796875,
  126264.796875,
  125885.0234375,
  125556.75,
  125224.859375,
  124924.296875,
  124646.9921875,
  124365.4765625,
  124110.3515625,
  123886.25,
  123652.859375,
  123461.5703125,
  123279.09375,
  123082.6953125,
  122911.203125,
  122747.9453125,
  122589.84375,
  122424.90625,
  122275.6953125,
  122131.3984375,
  121983.1171875,
  121861.875,
  121736.421875,
  121592.5625,
  121497.9296875,
  121366.7890625,
  121271.828125,
```

```
#lossdf=pd.DataFrame(model1.history.history)
#lossdf.plot()
pd.DataFrame(history.history).plot(figsize=(10,6))
```

```
<Axes: >
```




```
ypred=model1.predict(X_test)
```

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

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

```
The absolute mean error : 100322.00034218846
The r2_score : 0.7867297810522911
```