House Prices Prediction using TensorFlow

Dataset

> This dataset contains house sale prices for King County, which includes Seattle. It includes homes sold between May 2014 and May 2015. 21 columns. (features) 21597 rows.

o Feature Columns

- `id` : Unique ID for each home sold
- `date` : Date of the home sale
- `price` : Price of each home sold
- `bedrooms` : Number of bedrooms
- `bathrooms` : Number of bathrooms, where .5 accounts for a room with a toilet but no shower
- `sqft_living` : Square footage of the apartments interior living space
- `sqft_lot` : Square footage of the land space
- `floors` : Number of floors
- `waterfront` : - A dummy variable for whether the apartment was - overlooking the waterfront or not
- `view` : An index from 0 to 4 of how good the view of the property was
- `condition` : - An index from 1 to 5 on the condition of the apartment,
- `grade` : An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.
- `sqft_above` : The square footage of the interior housing space that is above ground level
- `sqft_basement` : The square footage of the interior housing space that is below ground level
- `yr_built` : The year the house was initially built
- `yr_renovated` : The year of the house's last renovation
- `zipcode` : What zipcode area the house is in
- `lat` : Lattitude
- `long` : Longitude
- `sqft_living15` : The square footage of interior housing - living - space for the nearest 15 neighbors
- `sqft_lot15` : The square footage of the land lots of the nearest 15 neighbors

Mounting Google Drives

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

```
Mounted at /content/drive
```

Importing Necessary Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style="whitegrid")
import plotly.express as px

import  warnings
warnings.filterwarnings('ignore')
```

Loading and Reading Dataset

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

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180.0	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170.0	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770.0	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050.0	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680.0	0	1987

5 rows × 21 columns

```
df.shape
```

```
(21613, 21)
```

```
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     21611 non-null  float64
 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(6), int64(14), object(1)
memory usage: 3.5+ MB
```

Checking Missing Value

```
null_percentage = df.isnull().sum()*100/len(df)
null_df = pd.DataFrame(null_percentage[null_percentage > 0], columns=["Percentage Missing"])
null_df
```

	Percentage Missing
sqft_above	0.009254

Missing Value Imputation

```
df.sqft_above.fillna(df.sqft_above.median(), inplace = True)
```

```
df.isnull().sum()*100/len(df)
```

```
id               0.0
date             0.0
price            0.0
bedrooms         0.0
bathrooms        0.0
sqft_living      0.0
sqft_lot         0.0
floors           0.0
waterfront       0.0
view             0.0
condition        0.0
grade            0.0
sqft_above       0.0
sqft_basement    0.0
yr_built         0.0
yr_renovated     0.0
zipcode          0.0
lat              0.0
long             0.0
sqft_living15    0.0
sqft_lot15       0.0
dtype: float64
```

```
cat_col=[]
for col in df.columns:
  print(col,'  : ',df[col].nunique())
  #print()
  if df[col].nunique()<15:
    cat_col.append(col)
print('==='*30)
print('categorical_col :', cat_col)
print('==='*30)
```

```
for col in cat_col:
  print(col, ' : ',df[col].unique().tolist())
```

```
id   :  21436
date   :  372
price   :  4028
bedrooms   :  13
bathrooms   :  30
sqft_living   :  1038
sqft_lot   :  9782
floors   :  6
waterfront   :  2
view   :  5
condition   :  5
grade   :  12
sqft_above   :  946
sqft_basement   :  306
yr_built   :  116
yr_renovated   :  70
zipcode   :  70
lat   :  5034
long   :  752
sqft_living15   :  777
sqft_lot15   :  8689
==========================================================================================
categorical_col : ['bedrooms', 'floors', 'waterfront', 'view', 'condition', 'grade']
==========================================================================================
bedrooms  :  [3, 2, 4, 5, 1, 6, 7, 0, 8, 9, 11, 10, 33]
floors  :  [1.0, 2.0, 1.5, 3.0, 2.5, 3.5]
waterfront  :  [0, 1]
view  :  [0, 3, 4, 2, 1]
condition  :  [3, 5, 4, 1, 2]
grade  :  [7, 6, 8, 11, 9, 5, 10, 12, 4, 3, 13, 1]
```

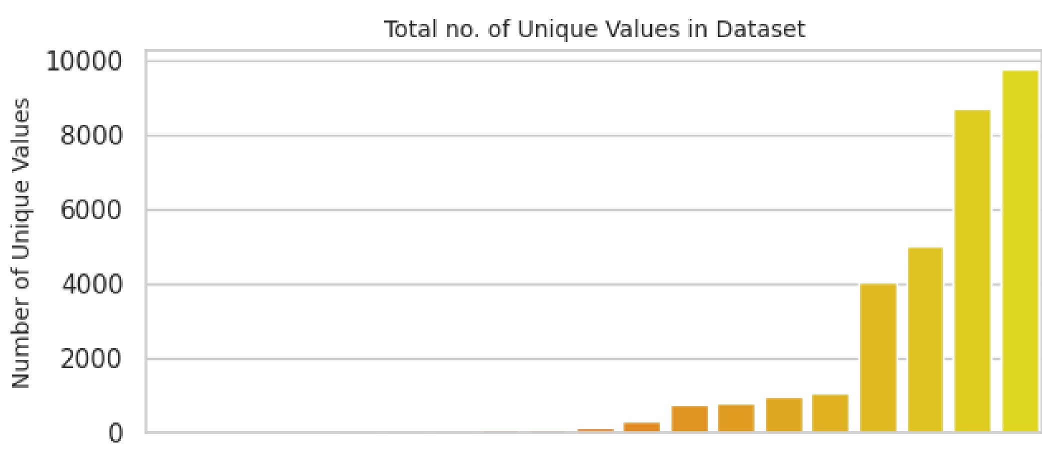
```
df.describe(exclude = 'object').T
```

	count	mean	std	min	25%	50%	75%	max
id	21613.0	4.580302e+09	2.876566e+09	1.000102e+06	2.123049e+09	3.904930e+09	7.308900e+09	9.900000e+09
price	21613.0	5.400881e+05	3.671272e+05	7.500000e+04	3.219500e+05	4.500000e+05	6.450000e+05	7.700000e+06
bedrooms	21613.0	3.370842e+00	9.300618e-01	0.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.300000e+01
bathrooms	21613.0	2.114757e+00	7.701632e-01	0.000000e+00	1.750000e+00	2.250000e+00	2.500000e+00	8.000000e+00
sqft_living	21613.0	2.079900e+03	9.184409e+02	2.900000e+02	1.427000e+03	1.910000e+03	2.550000e+03	1.354000e+04
sqft_lot	21613.0	1.510697e+04	4.142051e+04	5.200000e+02	5.040000e+03	7.618000e+03	1.068800e+04	1.651359e+06
floors	21613.0	1.494309e+00	5.399889e-01	1.000000e+00	1.000000e+00	1.500000e+00	2.000000e+00	3.500000e+00
waterfront	21613.0	7.541757e-03	8.651720e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
view	21613.0	2.343034e-01	7.663176e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
condition	21613.0	3.409430e+00	6.507430e-01	1.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	5.000000e+00
grade	21613.0	7.656873e+00	1.175459e+00	1.000000e+00	7.000000e+00	7.000000e+00	8.000000e+00	1.300000e+01
sqft_above	21613.0	1.788375e+03	8.280928e+02	2.900000e+02	1.190000e+03	1.560000e+03	2.210000e+03	9.410000e+03
sqft_basement	21613.0	2.915090e+02	4.425750e+02	0.000000e+00	0.000000e+00	0.000000e+00	5.600000e+02	4.820000e+03
yr_built	21613.0	1.971005e+03	2.937341e+01	1.900000e+03	1.951000e+03	1.975000e+03	1.997000e+03	2.015000e+03
yr_renovated	21613.0	8.440226e+01	4.016792e+02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	2.015000e+03
zipcode	21613.0	9.807794e+04	5.350503e+01	9.800100e+04	9.803300e+04	9.806500e+04	9.811800e+04	9.819900e+04
lat	21613.0	4.756005e+01	1.385637e-01	4.715590e+01	4.747100e+01	4.757180e+01	4.767800e+01	4.777760e+01
long	21613.0	-1.222139e+02	1.408283e-01	-1.225190e+02	-1.223280e+02	-1.222300e+02	-1.221250e+02	-1.213150e+02
sqft_living15	21613.0	1.986552e+03	6.853913e+02	3.990000e+02	1.490000e+03	1.840000e+03	2.360000e+03	6.210000e+03
sqft_lot15	21613.0	1.276846e+04	2.730418e+04	6.510000e+02	5.100000e+03	7.620000e+03	1.008300e+04	8.712000e+05

Exploratory Data Analysis (EDA)

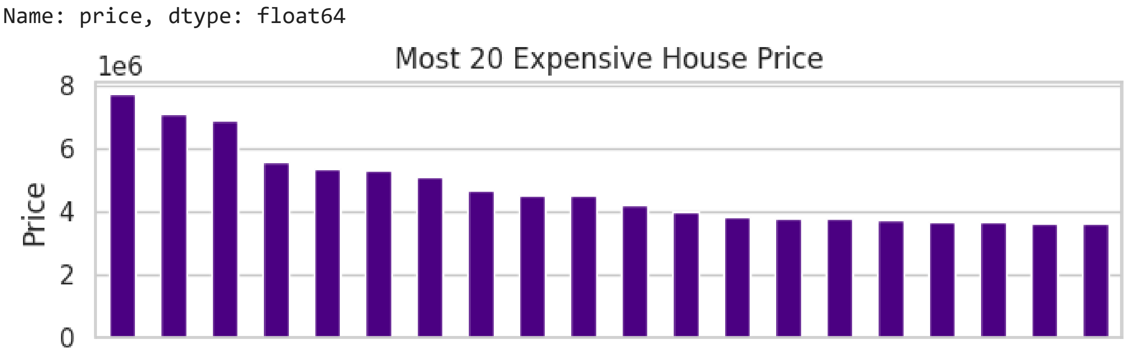
```
# Count the number of unique values in each column
def check_unquie_count(df):
    unique_counts = df.nunique().sort_values()
    print('=='*30)
    print('  '*10, 'Total no. of Unique Values')
    print('=='*30)
    print(unique_counts)
    print('=='*30)
# Create a bar plot or count plot of unique values

    plt.figure(figsize=(7, 3))
    sns.barplot(x=unique_counts.index, y=unique_counts.sort_values(),palette='autumn' )

    plt.xticks(rotation=80, fontsize= 10)
    plt.yticks(  )
    plt.xlabel('Columns',fontsize=10)
    plt.ylabel('Number of Unique Values', fontsize=10)
    plt.title('Total no. of Unique Values in Dataset', fontsize=10)
    plt.show()

check_unquie_count(df.iloc[:,2:])
```

```
=============================================================
                    Total no. of Unique Values
=============================================================
waterfront          2
view                5
condition           5
floors              6
grade              12
bedrooms           13
bathrooms          30
yr_renovated       70
zipcode            70
yr_built          116
sqft_basement     306
long              752
sqft_living15     777
sqft_above        946
sqft_living      1038
price            4028
lat              5034
sqft_lot15       8689
sqft_lot         9782
dtype: int64
=============================================================
```



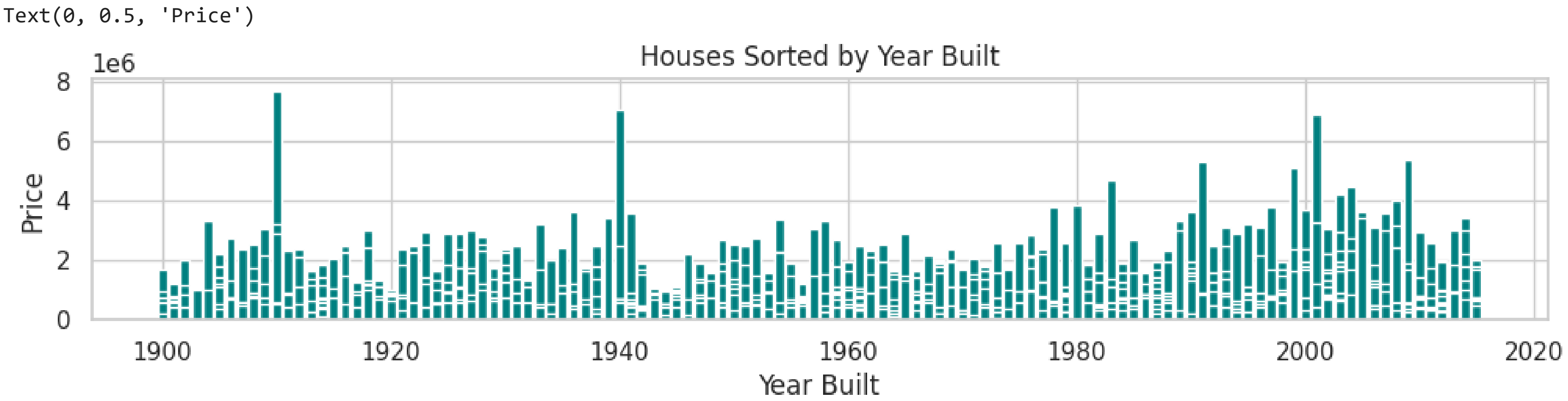

```
price_sorted = df['price'].sort_values(ascending=False)
plt.figure(figsize =(8,2))
price_sorted.head(20).plot(kind='bar', xticks=[], color ='indigo')
plt.ylabel('Price')
plt.title('Most 20 Expensive House Price')
price_sorted.head(10)
```

```
7252    7700000.0
3914    7062500.0
9254    6885000.0
4411    5570000.0
1448    5350000.0
1315    5300000.0
1164    5110800.0
8092    4668000.0
2626    4500000.0
8638    4489000.0
Name: price, dtype: float64
```

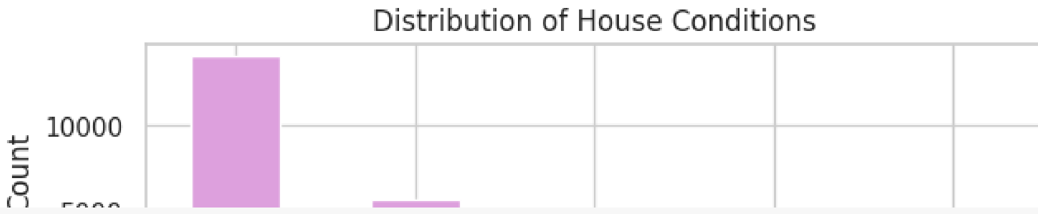



```
df_sorted = df.sort_values(by='yr_built')
plt.figure(figsize =(12,2))
plt.bar(df_sorted['yr_built'], df_sorted['price'], color='teal')
plt.title('Houses Sorted by Year Built')
plt.xlabel('Year Built')
plt.ylabel('Price')
```

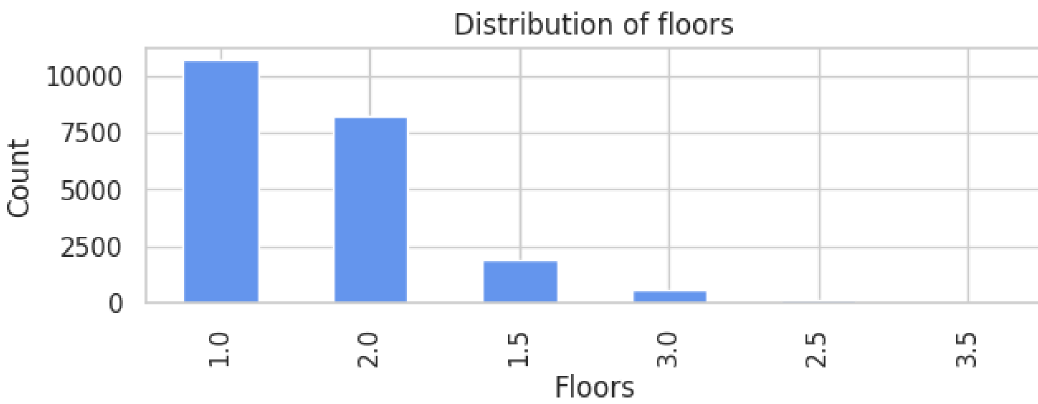
```
Text(0, 0.5, 'Price')
```



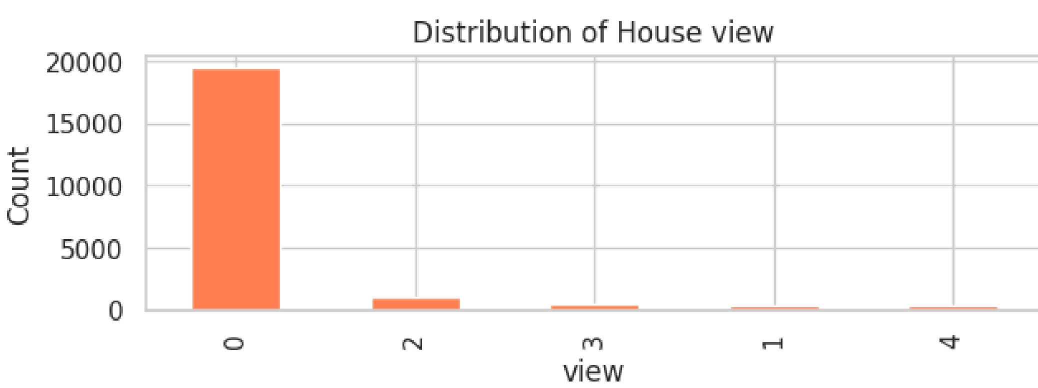

```
plt.figure(figsize=(7,2))
df['condition'].value_counts().plot(kind='bar', color ='plum')
plt.xlabel('Condition')
plt.ylabel('Count')
plt.title('Distribution of House Conditions')
plt.show()
```



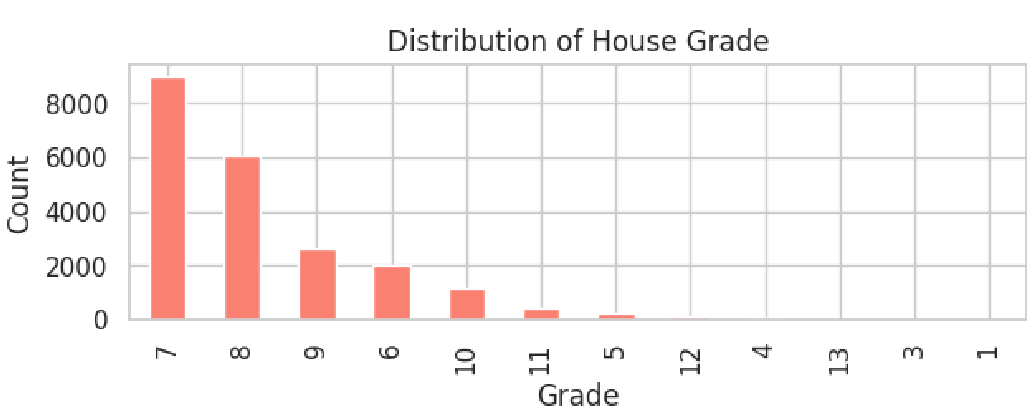

```
plt.figure(figsize=(7,2))
df['floors'].value_counts().plot(kind='bar', color ='cornflowerblue')
plt.xlabel('Floors')
plt.ylabel('Count')
plt.title('Distribution of floors')
plt.show()
```




```
plt.figure(figsize=(7,2))
df['view'].value_counts().plot(kind='bar', color ='coral')
plt.xlabel('view')
plt.ylabel('Count')
plt.title('Distribution of House view')
plt.show()
```



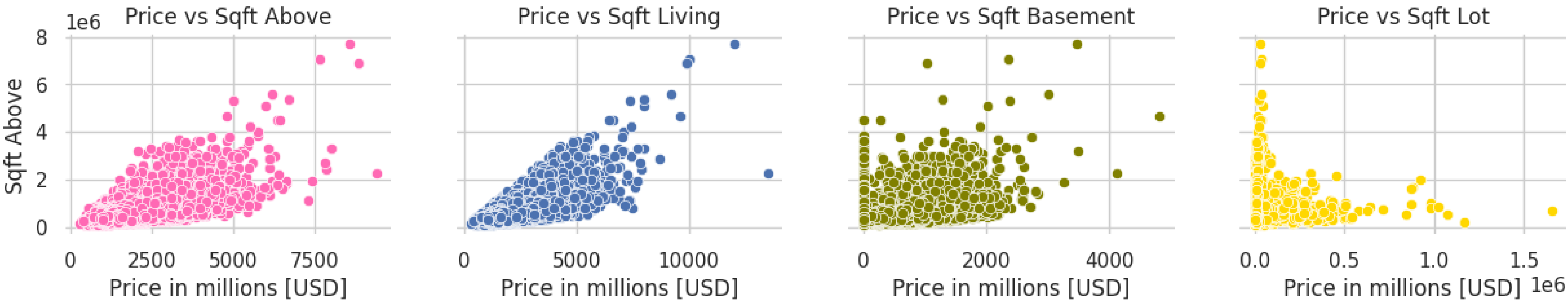

```
plt.figure(figsize=(7,2))
df['grade'].value_counts().plot(kind='bar', color ='salmon')
plt.xlabel('Grade')
plt.ylabel('Count')
plt.title('Distribution of House Grade')
plt.show()
```



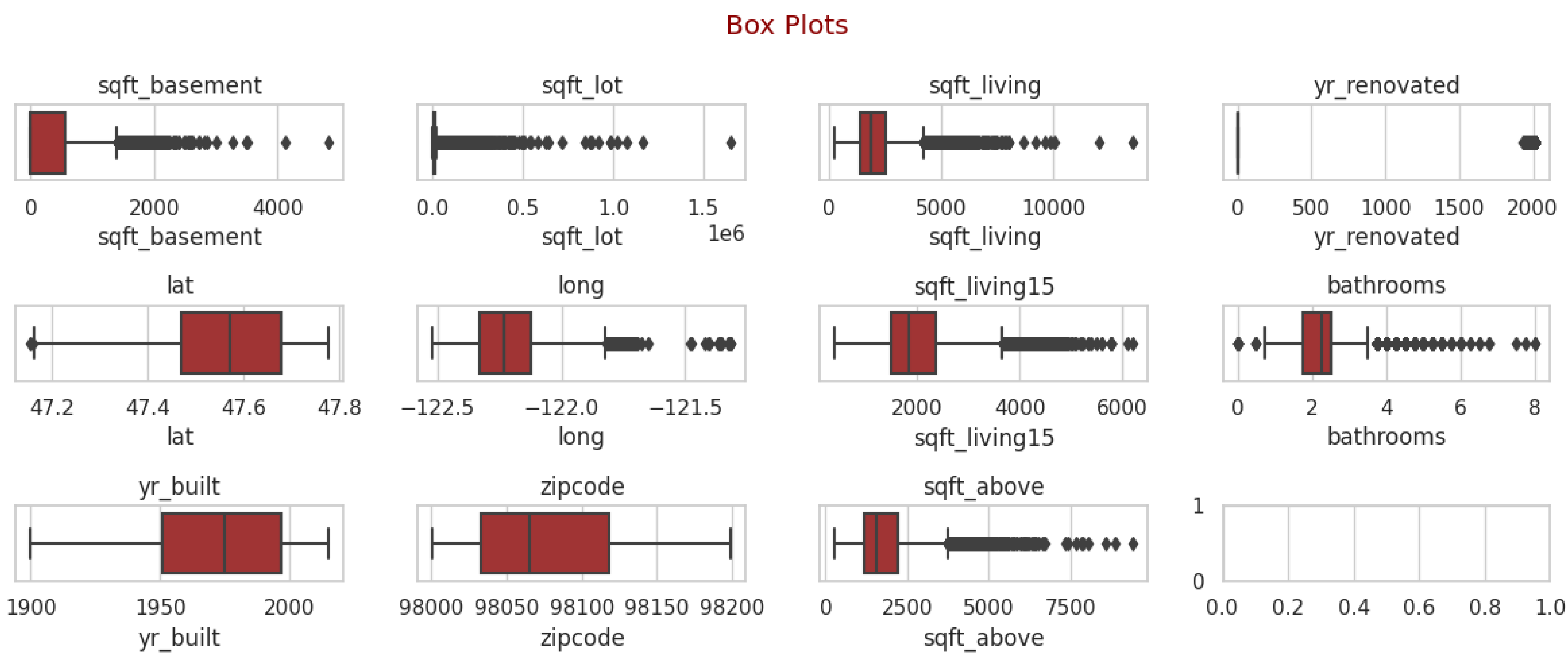

```
df.groupby(['grade','price'])['price'].count()
```

```
grade  price
1      142000.0     1
3      75000.0      1
       262000.0     1
       280000.0     1
4      80000.0      1
                   ..
13     3200000.0    1
       3800000.0    2
       5570000.0    1
       6885000.0    1
       7700000.0    1
Name: price, Length: 6693, dtype: int64
```

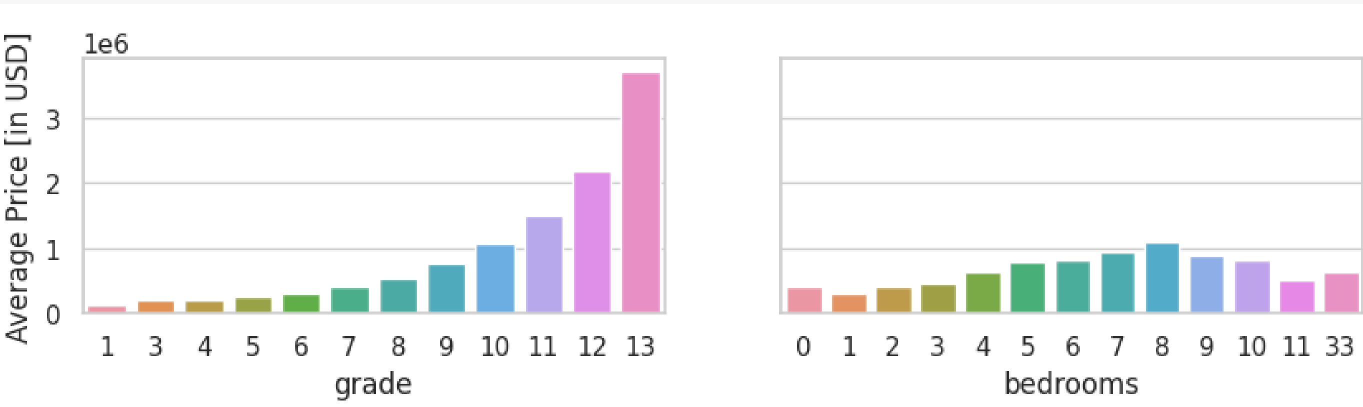
```
f, axes = plt.subplots(1, 4,figsize=(15,2),sharey= True)
sns.scatterplot(y='price',x='sqft_above', data=df, ax=axes[0], color ='hotpink')
sns.scatterplot(y='price',x='sqft_living', data=df, ax=axes[1])
sns.scatterplot(y='price',x='sqft_basement', data=df, ax=axes[2], color ='olive')
sns.scatterplot(y='price',x='sqft_lot', data=df, ax=axes[3],color ='gold')
sns.despine(bottom=True, left=True)
axes[0].set(xlabel='Price in millions [USD]', ylabel='Sqft Above', title='Price vs Sqft Above')
axes[1].set(xlabel='Price in millions [USD]', ylabel='Sqft Living', title='Price vs Sqft Living')
axes[2].set(xlabel='Price in millions [USD]', ylabel='Sqft Basement', title='Price vs Sqft Basement')
axes[3].set(xlabel='Price in millions [USD]', ylabel='Sqft Lot', title='Price vs Sqft Lot')
# axes[1].yaxis.set_label_position("right")
# axes[1].yaxis.tick_right()
plt.show()
```




```
fig, axes = plt.subplots(3, 4, figsize=(12, 5))
axes = axes.flatten()
columns_to_plot= list(set(df.iloc[:,2:-1].columns) - set(cat_col))
columns_to_plot.remove('price')
plt.suptitle('Box Plots', color ='darkred')
for i, col in enumerate(columns_to_plot):
    sns.boxplot(data=df, x=col, ax=axes[i], color='firebrick')
    axes[i].set_title(f'{col}')
plt.tight_layout()
plt.show()
```

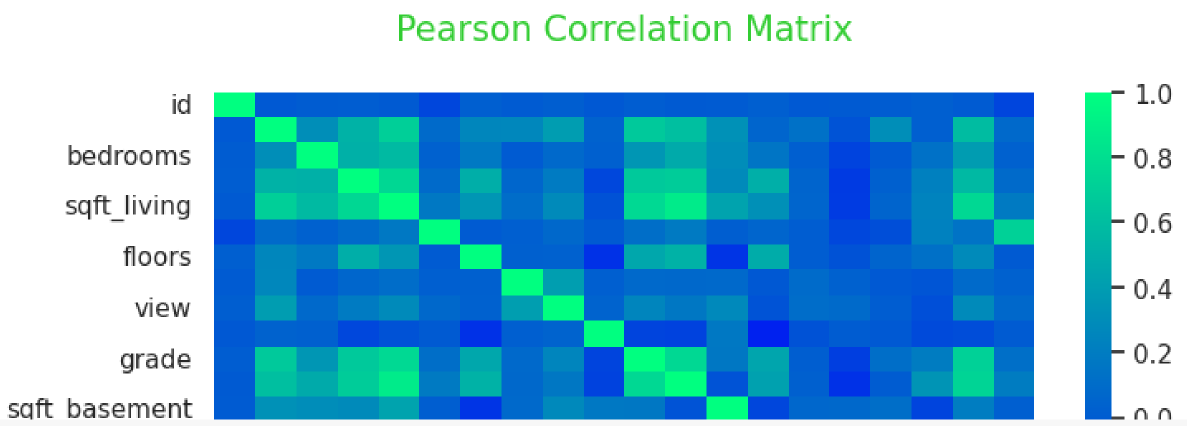



```
price_grade =df.groupby('grade')['price'].mean()
price_bedroom =df.groupby('bedrooms')['price'].mean()

f, axes = plt.subplots(1, 2,figsize=(10,2),sharey= True)

sns.barplot(x=price_grade.index, y=price_grade.values,ax=axes[0])
axes[0].set(ylabel='Average Price [in USD]')
sns.barplot(x=price_bedroom.index, y=price_bedroom.values,ax=axes[1])

plt.show()
```



Finding Correlation

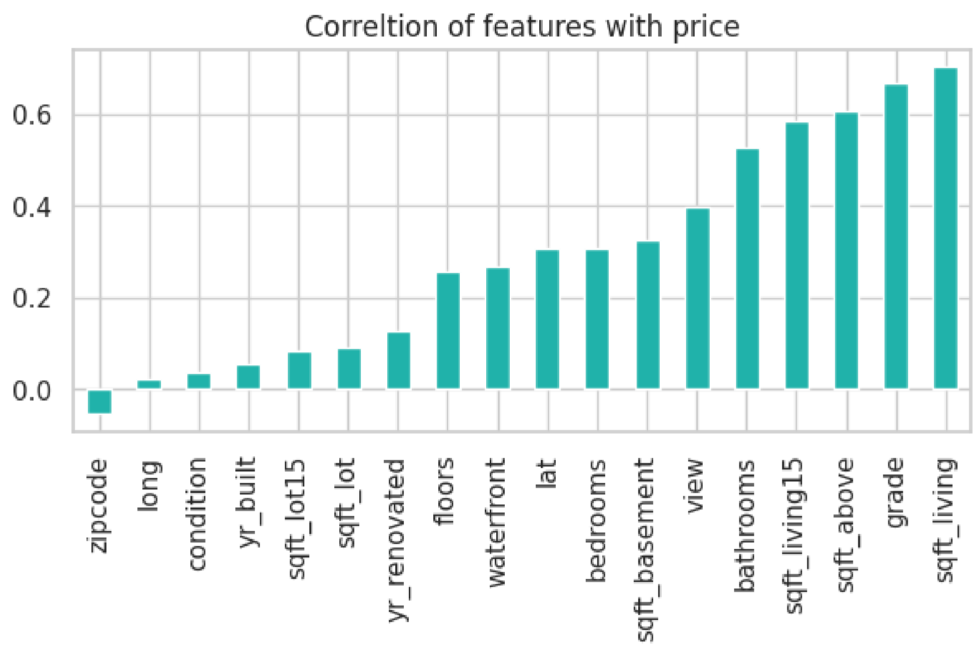
```
sns.set(style="whitegrid", font_scale=1)

plt.figure(figsize=(8,4))
plt.title('Pearson Correlation Matrix\n',fontsize=15, color='limegreen')
sns.heatmap(df.corr(),cmap="winter", )
plt.show()
```




```
len(df.iloc[:,1:].corr()['price'])
```

    19

```
df.iloc[:,1:].corr()['price'].sort_values().head(18).plot(kind='bar', color ='lightseagreen',figsize=(7,3))
plt.title('Correltion of features with price')
plt.show()
```




```
# dropping irrelevant columns
df.drop(['id'], axis=1, inplace= True)
df = df.drop('date',axis=1)
```

```
df.shape
```

    (21613, 19)

```
# splitting into target and features
x = df.drop('price', axis=1)
y = df['price']
```

```
print('xshape :', x.shape)
print('yshape :', y.shape)
```

    xshape : (21613, 18)
    yshape : (21613,)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
```

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state= 12)
print('x_train shape :', x_train.shape)
print('x_test shape :', x_test.shape)
print('y_train shape :', y_train.shape)
print('y_test shape :', y_test.shape)
```

    x_train shape : (17290, 18)
    x_test shape : (4323, 18)
    y_train shape : (17290,)
    y_test shape : (4323,)

Feature Scaling

-**"If you are not doing Feature scaling, then it is a crime!!"**

- Scaling features can improve the performance of DNNs.
- Feature scaling helps in achieving faster convergence and finding a better minimum.

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Importing Tensorflow package

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.initializers import HeNormal
```

```
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.losses import MeanSquaredError
```

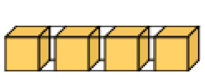
```
! pip install visualkeras
import visualkeras
```

```
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.23.5)
Collecting aggdraw>=1.3.11 (from visualkeras)
  Downloading aggdraw-1.3.16-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.0/993.0 kB 6.3 MB/s eta 0:00:00
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.16 visualkeras-0.0.2
```

Approach No.1 (3 hidden layers)

```
#==================================================================
ann0 = Sequential()
# 1st hidden layer
ann0.add(Dense(units= 32, activation ='relu',kernel_initializer=HeNormal(), input_dim=18))
# 2nd hidden layer
ann0.add(Dense(units=8,activation ='relu'))
# 3rd hidden layer
ann0.add(Dense(units=8,activation ='relu'))
# output
ann0.add(Dense(units=1,activation ='linear'))
#=========================================================
ann0.compile(optimizer ='Adam', loss=MeanSquaredError())
#=========================================================

ann0.summary()
print()
print('Model Structure')
print()
visualkeras.layered_view(ann0)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 32)                608       
                                                                 
 dense_1 (Dense)             (None, 8)                 264       
                                                                 
 dense_2 (Dense)             (None, 8)                 72        
                                                                 
 dense_3 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 953
Trainable params: 953
Non-trainable params: 0
_________________________________________________________________

Model Structure
```

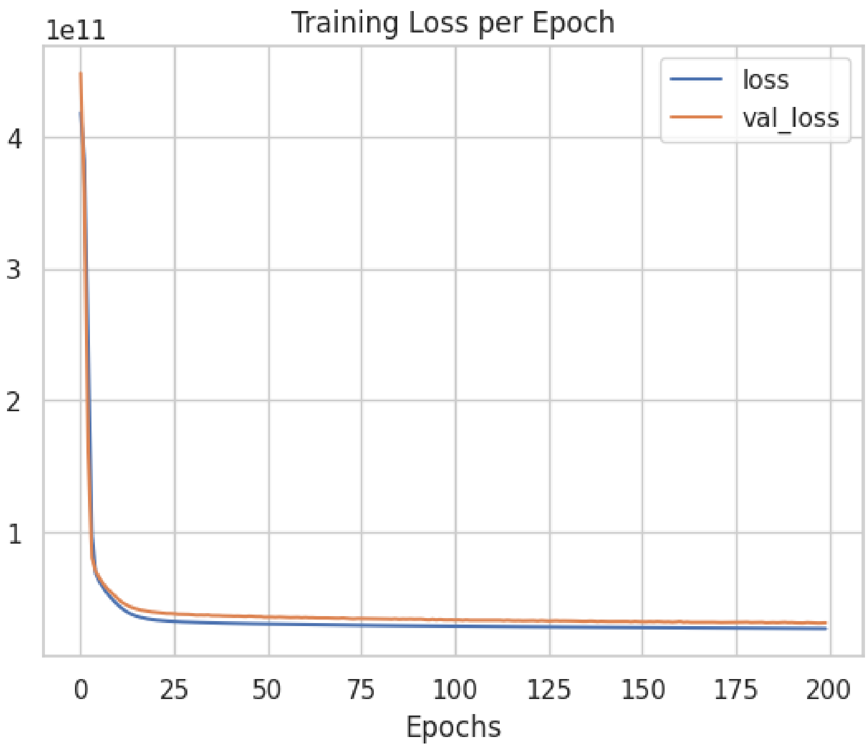


```
ann0.fit(x=x_train,y=y_train,
         validation_data=(x_test,y_test),
         epochs=200)
```

```
541/541 [==============================] - 2s 3ms/step - loss: 26679257088.0000 - val_loss: 30882029568.0000
Epoch 193/200
541/541 [==============================] - 1s 2ms/step - loss: 26647250944.0000 - val_loss: 30866327552.0000
Epoch 194/200
541/541 [==============================] - 1s 2ms/step - loss: 26638968832.0000 - val_loss: 30773739520.0000
Epoch 195/200
541/541 [==============================] - 1s 2ms/step - loss: 26627043328.0000 - val_loss: 31127758848.0000
Epoch 196/200
541/541 [==============================] - 1s 2ms/step - loss: 26610448384.0000 - val_loss: 30999367680.0000
Epoch 197/200
541/541 [==============================] - 1s 2ms/step - loss: 26617874432.0000 - val_loss: 30758598656.0000
Epoch 198/200
541/541 [==============================] - 1s 2ms/step - loss: 26586468352.0000 - val_loss: 30692970496.0000
Epoch 199/200
541/541 [==============================] - 1s 2ms/step - loss: 26588260352.0000 - val_loss: 30866393088.0000
Epoch 200/200
541/541 [==============================] - 2s 3ms/step - loss: 26539020288.0000 - val_loss: 30919233536.0000
<keras.callbacks.History at 0x7a8167b729b0>
```

```
losses_0 = pd.DataFrame(ann0.history.history)

plt.figure(figsize=(15,5))
losses_0.plot()
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
plt.show()
```

```
<Figure size 1500x500 with 0 Axes>
```




```
# Getting Accuracy
y_pred_0= ann0.predict(x_test)
r20 = r2_score(y_test, y_pred_0)
print('==='*25)
print('R2score :', r20)
print('==='*25)
```

```
136/136 [==============================] - 3s 15ms/step
===========================================================================
R2score : 0.7987721829687866
===========================================================================
```

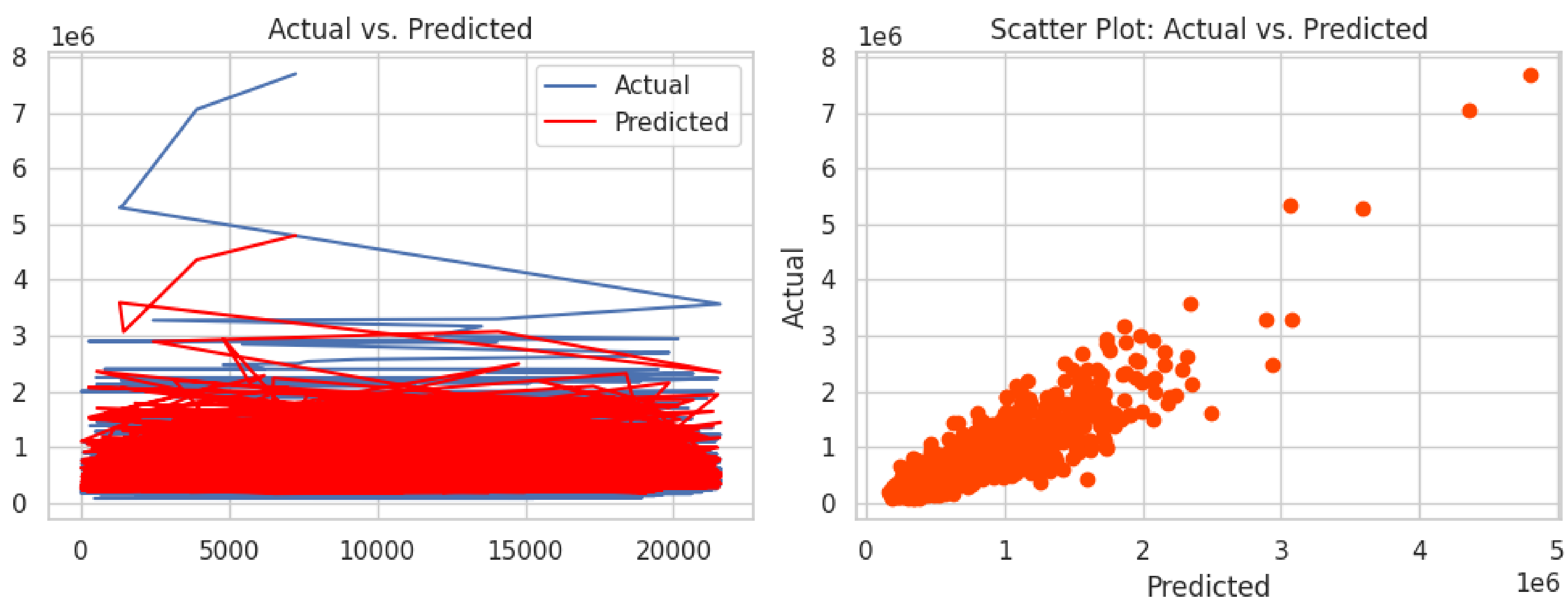
```
rounded_y_pred_0 = np.round(y_pred_0).astype(int)
compare_0 = pd.DataFrame({'Predicted': rounded_y_pred_0.flatten(), 'Actual': y_test.astype(int)})

# compare the actaual and predicted value of house price]
compare_0[['Actual', 'Predicted']]
```

	Actual	Predicted
2019	275000	310484
3435	279000	293984
15940	200500	252449
9811	750000	828574
18665	395000	332086
...	...	...
3390	579000	388695
6801	599000	733503
4775	248500	332596
10634	645000	520178
1529	810000	703473

4323 rows × 2 columns

```
fig, axes =plt.subplots(1, 2, figsize=(10, 4))
sorted_actual_0 = compare_0['Actual'].sort_values()
sorted_predicted_0 = compare_0.loc[sorted_actual_0.index, 'Predicted']

axes[0].plot(sorted_actual_0, label='Actual')
axes[0].plot(sorted_predicted_0, color='red', label='Predicted')
axes[0].set_title('Actual vs. Predicted')
axes[0].legend()
```

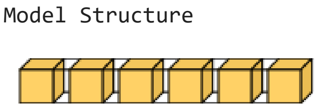
```
axes[1].scatter(compare_0['Predicted'], compare_0['Actual'], color='orangered')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Scatter Plot: Actual vs. Predicted')

plt.tight_layout()
plt.show()
```



Approach no. 2 (5 hidden layers)

```
#================================================================
ann = Sequential()
# 1st hidden layer
ann.add(Dense(units= 128, activation ='relu',kernel_initializer=HeNormal(), input_dim=18))
# 2nd hidden layer
ann.add(Dense(units=128,activation ='relu'))
# 3rd hidden layer
ann.add(Dense(units=64,activation ='relu'))
# 4th hidden layer
ann.add(Dense(units=32,activation ='relu'))
# 5th hidden layer
ann.add(Dense(units=32,activation ='relu'))
# output
ann.add(Dense(units=1,activation ='linear'))
#=======================================================
# compiling
ann.compile(optimizer ='Adam', loss=MeanSquaredError())
#=======================================================
# summarising the model
print(ann.summary())
print()
print('Model Structure')
print()
# visualising the model
visualkeras.layered_view(ann)
```

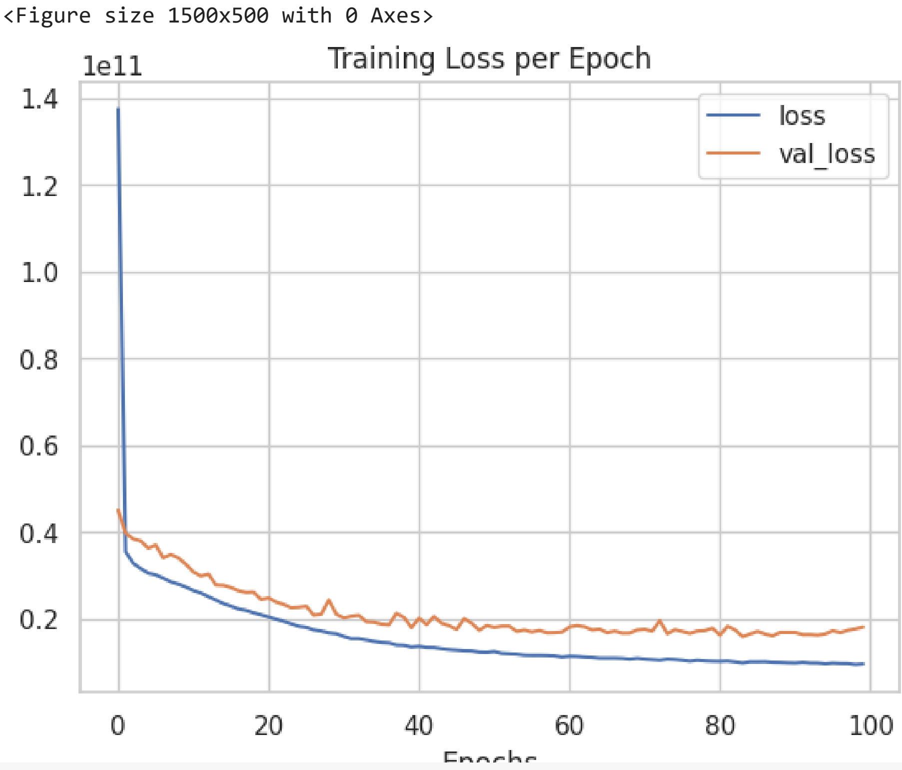
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 128)               2432

 dense_5 (Dense)             (None, 128)               16512

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 32)                1056

 dense_9 (Dense)             (None, 1)                 33

=================================================================
Total params: 30,369
Trainable params: 30,369
Non-trainable params: 0
_________________________________________________________________
None

Model Structure
```



```
ann.fit(x=x_train,y=y_train,
          validation_data=(x_test,y_test),
          epochs=200)
#=====================================================
losses = pd.DataFrame(ann.history.history)

plt.figure(figsize=(15,5))
losses.plot()
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
plt.show()
```

```
Epoch 1/100
541/541 [==============================] - 6s 7ms/step - loss: 137422356480.0000 - val_loss: 45048426496.0000
Epoch 2/100
541/541 [==============================] - 4s 8ms/step - loss: 35435663360.0000 - val_loss: 39709827072.0000
Epoch 3/100
541/541 [==============================] - 5s 10ms/step - loss: 32818038784.0000 - val_loss: 38407929856.0000
Epoch 4/100
541/541 [==============================] - 6s 10ms/step - loss: 31582148608.0000 - val_loss: 37942251520.0000
Epoch 5/100
541/541 [==============================] - 3s 6ms/step - loss: 30530150400.0000 - val_loss: 36273152000.0000
Epoch 6/100
541/541 [==============================] - 4s 7ms/step - loss: 30146813952.0000 - val_loss: 37010280448.0000
Epoch 7/100
541/541 [==============================] - 6s 10ms/step - loss: 29367007232.0000 - val_loss: 34034704384.0000
Epoch 8/100
541/541 [==============================] - 6s 11ms/step - loss: 28514248704.0000 - val_loss: 34839142400.0000
Epoch 9/100
541/541 [==============================] - 5s 8ms/step - loss: 28003907584.0000 - val_loss: 34044037120.0000
Epoch 10/100
541/541 [==============================] - 4s 7ms/step - loss: 27322345472.0000 - val_loss: 32567685120.0000
Epoch 11/100
541/541 [==============================] - 2s 3ms/step - loss: 26528776192.0000 - val_loss: 30828027904.0000
Epoch 12/100
541/541 [==============================] - 2s 4ms/step - loss: 25949997056.0000 - val_loss: 29891565568.0000
Epoch 13/100
541/541 [==============================] - 3s 5ms/step - loss: 25115578368.0000 - val_loss: 30327013376.0000
Epoch 14/100
541/541 [==============================] - 3s 5ms/step - loss: 24328355840.0000 - val_loss: 27803961344.0000
Epoch 15/100
541/541 [==============================] - 2s 4ms/step - loss: 23498936320.0000 - val_loss: 27714248704.0000
Epoch 16/100
541/541 [==============================] - 2s 3ms/step - loss: 22925819904.0000 - val_loss: 27221757952.0000
Epoch 17/100
541/541 [==============================] - 2s 3ms/step - loss: 22279489536.0000 - val_loss: 26485792768.0000
Epoch 18/100
541/541 [==============================] - 2s 3ms/step - loss: 21948305408.0000 - val_loss: 26082635776.0000
Epoch 19/100
541/541 [==============================] - 2s 3ms/step - loss: 21395134464.0000 - val_loss: 26162401280.0000
Epoch 20/100
541/541 [==============================] - 2s 3ms/step - loss: 20932915200.0000 - val_loss: 24475191296.0000
Epoch 21/100
541/541 [==============================] - 2s 4ms/step - loss: 20423815168.0000 - val_loss: 24807718912.0000
Epoch 22/100
541/541 [==============================] - 3s 5ms/step - loss: 19917975552.0000 - val_loss: 23873480704.0000
Epoch 23/100
541/541 [==============================] - 3s 5ms/step - loss: 19421628416.0000 - val_loss: 23310516224.0000
Epoch 24/100
541/541 [==============================] - 3s 5ms/step - loss: 18851102720.0000 - val_loss: 22562816000.0000
Epoch 25/100
541/541 [==============================] - 2s 3ms/step - loss: 18288281600.0000 - val_loss: 22665576448.0000
Epoch 26/100
541/541 [==============================] - 2s 3ms/step - loss: 18034251776.0000 - val_loss: 22899099648.0000
Epoch 27/100
541/541 [==============================] - 2s 3ms/step - loss: 17459703808.0000 - val_loss: 20882376704.0000
Epoch 28/100
541/541 [==============================] - 2s 3ms/step - loss: 17198807040.0000 - val_loss: 21064613888.0000
Epoch 29/100
541/541 [==============================] - 2s 3ms/step - loss: 16767243264.0000 - val_loss: 24328790016.0000
Epoch 30/100
541/541 [==============================] - 2s 3ms/step - loss: 16571196416.0000 - val_loss: 21088917504.0000
Epoch 31/100
541/541 [==============================] - 3s 5ms/step - loss: 15934827520.0000 - val_loss: 20198803456.0000
Epoch 32/100
541/541 [==============================] - 3s 5ms/step - loss: 15435724800.0000 - val_loss: 20617209856.0000
Epoch 33/100
541/541 [==============================] - 3s 5ms/step - loss: 15436709888.0000 - val_loss: 20827373568.0000
Epoch 34/100
541/541 [==============================] - 2s 4ms/step - loss: 15135209472.0000 - val_loss: 19312666624.0000
Epoch 35/100
541/541 [==============================] - 2s 3ms/step - loss: 14795478016.0000 - val_loss: 19226179584.0000
Epoch 36/100
541/541 [==============================] - 2s 3ms/step - loss: 14548568064.0000 - val_loss: 18761496576.0000
Epoch 37/100
541/541 [==============================] - 2s 3ms/step - loss: 14440711168.0000 - val_loss: 18630230016.0000
Epoch 38/100
541/541 [==============================] - 2s 3ms/step - loss: 13970788352.0000 - val_loss: 21269239808.0000
Epoch 39/100
541/541 [==============================] - 2s 3ms/step - loss: 13899895808.0000 - val_loss: 20319809536.0000
Epoch 40/100
541/541 [==============================] - 3s 5ms/step - loss: 13510715392.0000 - val_loss: 17974192128.0000
Epoch 41/100
541/541 [==============================] - 3s 5ms/step - loss: 13665927168.0000 - val_loss: 20156143616.0000
Epoch 42/100
541/541 [==============================] - 3s 5ms/step - loss: 13398671360.0000 - val_loss: 18656714752.0000
Epoch 43/100
541/541 [==============================] - 2s 4ms/step - loss: 13377505280.0000 - val_loss: 20544737280.0000
Epoch 44/100
541/541 [==============================] - 2s 3ms/step - loss: 13108082688.0000 - val_loss: 18965215232.0000
Epoch 45/100
541/541 [==============================] - 2s 4ms/step - loss: 12920199168.0000 - val_loss: 18408980480.0000
Epoch 46/100
541/541 [==============================] - 2s 3ms/step - loss: 12795317248.0000 - val_loss: 17565196288.0000
Epoch 47/100
541/541 [==============================] - 2s 3ms/step - loss: 12659366912.0000 - val_loss: 20040972288.0000
Epoch 48/100
541/541 [==============================] - 2s 4ms/step - loss: 12618765312.0000 - val_loss: 18986090496.0000
Epoch 49/100
541/541 [==============================] - 3s 5ms/step - loss: 12329232384.0000 - val_loss: 17372323840.0000
Epoch 50/100
541/541 [==============================] - 3s 6ms/step - loss: 12283699200.0000 - val_loss: 18466672640.0000
Epoch 51/100
541/541 [==============================] - 3s 5ms/step - loss: 12466487296.0000 - val_loss: 18025895936.0000
Epoch 52/100
541/541 [==============================] - 2s 3ms/step - loss: 12041658368.0000 - val_loss: 18308919296.0000
Epoch 53/100
541/541 [==============================] - 2s 3ms/step - loss: 11955538944.0000 - val_loss: 18312867840.0000
Epoch 54/100
541/541 [==============================] - 2s 3ms/step - loss: 11864729600.0000 - val_loss: 17101800448.0000
Epoch 55/100
541/541 [==============================] - 2s 3ms/step - loss: 11640690688.0000 - val_loss: 17420795904.0000
Epoch 56/100
541/541 [==============================] - 2s 3ms/step - loss: 11602987008.0000 - val_loss: 16983826432.0000
Epoch 57/100
```

```
541/541 [==============================] - 3s 5ms/step - loss: 11609467904.0000 - val_loss: 17335601152.0000
Epoch 58/100
541/541 [==============================] - 3s 6ms/step - loss: 11577352192.0000 - val_loss: 16785148928.0000
Epoch 59/100
541/541 [==============================] - 3s 5ms/step - loss: 11493363712.0000 - val_loss: 16820323328.0000
Epoch 60/100
541/541 [==============================] - 3s 5ms/step - loss: 11209494528.0000 - val_loss: 16954498048.0000
Epoch 61/100
541/541 [==============================] - 2s 3ms/step - loss: 11400751104.0000 - val_loss: 18143444992.0000
Epoch 62/100
541/541 [==============================] - 2s 3ms/step - loss: 11346578432.0000 - val_loss: 18448732160.0000
Epoch 63/100
541/541 [==============================] - 2s 3ms/step - loss: 11200144384.0000 - val_loss: 18166804480.0000
Epoch 64/100
541/541 [==============================] - 2s 3ms/step - loss: 11121937408.0000 - val_loss: 17460979712.0000
Epoch 65/100
541/541 [==============================] - 2s 3ms/step - loss: 10951224320.0000 - val_loss: 17621917696.0000
Epoch 66/100
541/541 [==============================] - 2s 3ms/step - loss: 10944583680.0000 - val_loss: 16806412288.0000
Epoch 67/100
541/541 [==============================] - 2s 5ms/step - loss: 10951102464.0000 - val_loss: 17173984256.0000
Epoch 68/100
541/541 [==============================] - 3s 5ms/step - loss: 10892199936.0000 - val_loss: 16709868544.0000
Epoch 69/100
541/541 [==============================] - 3s 5ms/step - loss: 10717848576.0000 - val_loss: 16726793216.0000
Epoch 70/100
541/541 [==============================] - 2s 4ms/step - loss: 10909222912.0000 - val_loss: 17446057984.0000
Epoch 71/100
541/541 [==============================] - 2s 4ms/step - loss: 10726544384.0000 - val_loss: 17604122624.0000
Epoch 72/100
541/541 [==============================] - 2s 3ms/step - loss: 10616995840.0000 - val_loss: 17164590080.0000
Epoch 73/100
541/541 [==============================] - 2s 3ms/step - loss: 10487380992.0000 - val_loss: 19558537216.0000
Epoch 74/100
541/541 [==============================] - 2s 3ms/step - loss: 10735369216.0000 - val_loss: 16557796352.0000
Epoch 75/100
541/541 [==============================] - 2s 3ms/step - loss: 10648229888.0000 - val_loss: 17493682176.0000
Epoch 76/100
541/541 [==============================] - 3s 5ms/step - loss: 10478521344.0000 - val_loss: 17085480960.0000
Epoch 77/100
541/541 [==============================] - 3s 6ms/step - loss: 10254431232.0000 - val_loss: 16646784000.0000
Epoch 78/100
541/541 [==============================] - 3s 5ms/step - loss: 10470346752.0000 - val_loss: 17185970176.0000
Epoch 79/100
541/541 [==============================] - 2s 3ms/step - loss: 10323322880.0000 - val_loss: 17295822848.0000
Epoch 80/100
541/541 [==============================] - 2s 3ms/step - loss: 10246901760.0000 - val_loss: 17844168704.0000
Epoch 81/100
541/541 [==============================] - 2s 3ms/step - loss: 10216067072.0000 - val_loss: 16271827968.0000
Epoch 82/100
541/541 [==============================] - 2s 3ms/step - loss: 10290741248.0000 - val_loss: 18282672128.0000
Epoch 83/100
541/541 [==============================] - 2s 3ms/step - loss: 10077717504.0000 - val_loss: 17468139520.0000
Epoch 84/100
541/541 [==============================] - 5s 10ms/step - loss: 9846413312.0000 - val_loss: 15910862848.0000
Epoch 85/100
541/541 [==============================] - 7s 14ms/step - loss: 10104541184.0000 - val_loss: 16514131968.0000
Epoch 86/100
541/541 [==============================] - 4s 7ms/step - loss: 10097469440.0000 - val_loss: 17093722112.0000
Epoch 87/100
541/541 [==============================] - 5s 9ms/step - loss: 10123370496.0000 - val_loss: 16491248640.0000
Epoch 88/100
541/541 [==============================] - 4s 7ms/step - loss: 9981187072.0000 - val_loss: 16090681344.0000
Epoch 89/100
541/541 [==============================] - 5s 10ms/step - loss: 9943211008.0000 - val_loss: 16870624256.0000
Epoch 90/100
541/541 [==============================] - 5s 10ms/step - loss: 9865867264.0000 - val_loss: 16871925760.0000
Epoch 91/100
541/541 [==============================] - 3s 6ms/step - loss: 9821626368.0000 - val_loss: 16858118144.0000
Epoch 92/100
541/541 [==============================] - 4s 7ms/step - loss: 9965225984.0000 - val_loss: 16370526208.0000
Epoch 93/100
541/541 [==============================] - 6s 11ms/step - loss: 9813862400.0000 - val_loss: 16376179712.0000
Epoch 94/100
541/541 [==============================] - 6s 12ms/step - loss: 9795236864.0000 - val_loss: 16257963008.0000
Epoch 95/100
541/541 [==============================] - 3s 5ms/step - loss: 9618856960.0000 - val_loss: 16521830400.0000
Epoch 96/100
541/541 [==============================] - 3s 5ms/step - loss: 9771654144.0000 - val_loss: 17254709248.0000
Epoch 97/100
541/541 [==============================] - 3s 6ms/step - loss: 9682592768.0000 - val_loss: 16820824064.0000
Epoch 98/100
541/541 [==============================] - 4s 7ms/step - loss: 9672051712.0000 - val_loss: 17366171648.0000
Epoch 99/100
541/541 [==============================] - 3s 5ms/step - loss: 9451364352.0000 - val_loss: 17685532672.0000
Epoch 100/100
541/541 [==============================] - 3s 5ms/step - loss: 9587271680.0000 - val_loss: 18098513920.0000
<Figure size 1500x500 with 0 Axes>
```



```
# getting accuracy
y_pred= ann.predict(x_test)
r2 = r2_score(y_test, y_pred)
print('==='*25)
print('R2score :', r2)
print('==='*25)
```

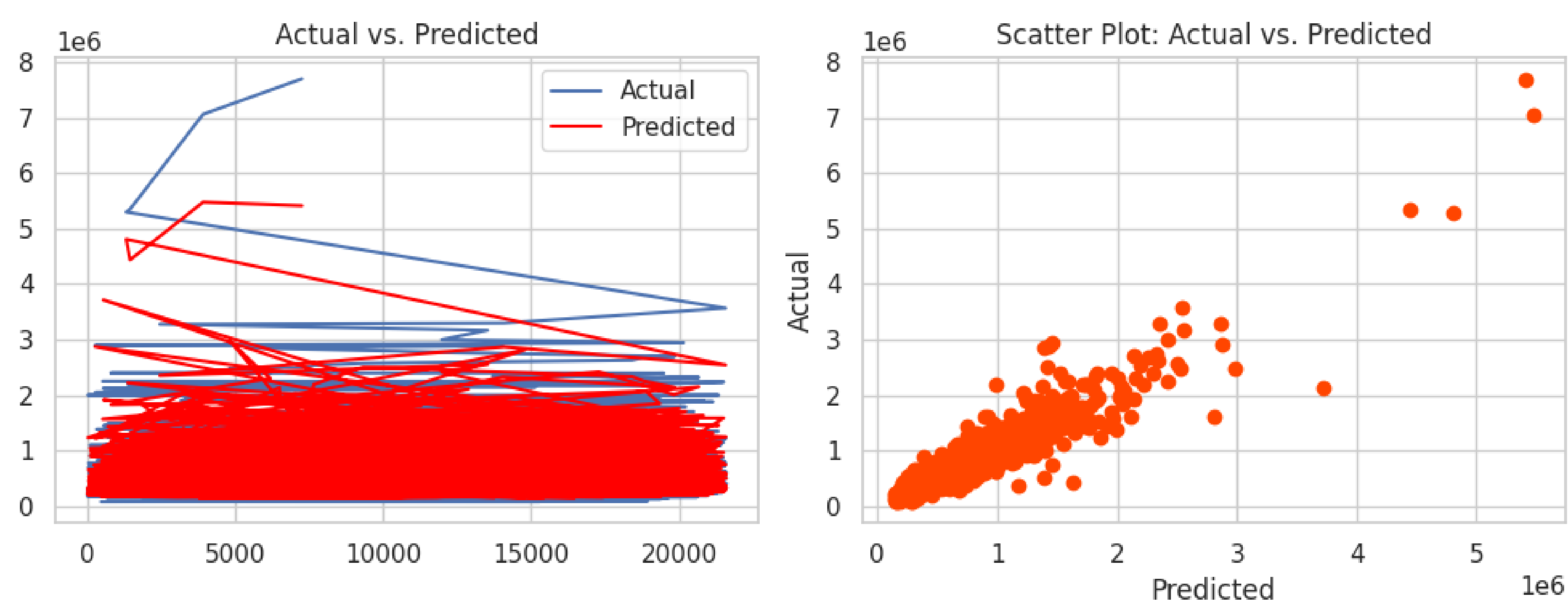
```
136/136 [==============================] - 0s 2ms/step
===========================================================================
R2score : 0.8822117170571872
===========================================================================
```

```
rounded_y_pred = np.round(y_pred).astype(int)
compare = pd.DataFrame({'Predicted': rounded_y_pred.flatten(), 'Actual': y_test.astype(int)})

# compare the actaual and predicted value of house price]
compare[['Actual', 'Predicted']]
```

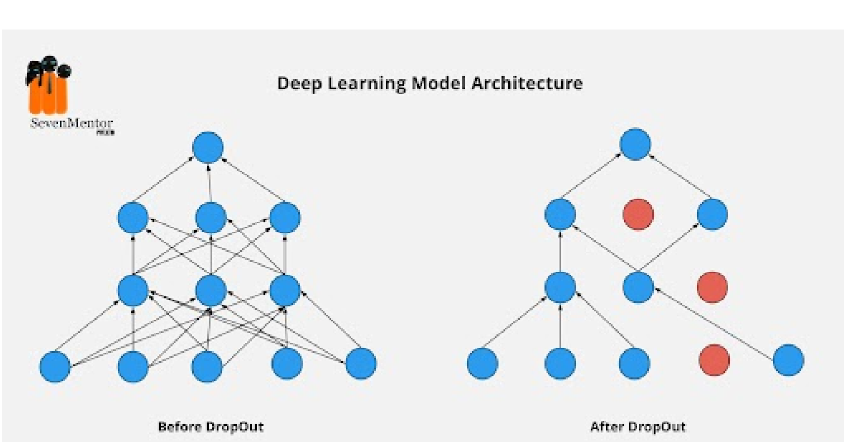
	Actual	Predicted
2019	275000	272542
3435	279000	267426
15940	200500	237488
9811	750000	660498
18665	395000	484496
...	...	...
3390	579000	475822
6801	599000	578025
4775	248500	263556
10634	645000	559371
1529	810000	737901

4323 rows × 2 columns

```
fig, axes =plt.subplots(1, 2, figsize=(10, 4))
sorted_actual = compare['Actual'].sort_values()
sorted_predicted = compare.loc[sorted_actual.index, 'Predicted']

axes[0].plot(sorted_actual, label='Actual')
axes[0].plot(sorted_predicted, color='red', label='Predicted')
axes[0].set_title('Actual vs. Predicted')
axes[0].legend()


axes[1].scatter(compare['Predicted'], compare['Actual'], color='orangered')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Scatter Plot: Actual vs. Predicted')

plt.tight_layout()
plt.show()
```



Approach 3 (with Dropout)



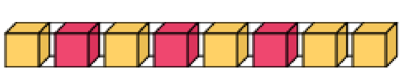
- **Prevents overfitting in Deep Neural Network(DNN)**
- **It is a regularisation tecnique commonly used in DNN.**

- **It invloves randomly "dropping out" (removing / deactivating/ setting to zero) a fraction of neurons in hidden layer during training iteration**

> ***Why overfitting?***
>
> > - complex model with mnay parameters relative to amount of training data fit noise. Instead of learning underlying pattern in the data an overfit model may memorize the training data.

```
ann1 = Sequential()
ann1.add(Dense(128, activation ='relu',kernel_initializer=HeNormal(), input_dim=18))
ann1.add(Dropout(0.1))
ann1.add(Dense(64,activation ='relu'))
ann1.add(Dropout(0.1))
ann1.add(Dense(32,activation ='relu'))
ann1.add(Dropout(0.1))
ann1.add(Dense(8,activation ='relu'))
ann1.add(Dense(1,activation ='linear'))
ann1.compile(optimizer ='Adam', loss=MeanSquaredError())

print(ann1.summary())
print()
print('Model Structure')
print()
visualkeras.layered_view(ann1)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_10 (Dense)            (None, 128)               2432

 dropout (Dropout)           (None, 128)               0

 dense_11 (Dense)            (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_12 (Dense)            (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0

 dense_13 (Dense)            (None, 8)                 264

 dense_14 (Dense)            (None, 1)                 9

=================================================================
Total params: 13,041
Trainable params: 13,041
Non-trainable params: 0
_________________________________________________________________
None

Model Structure
```



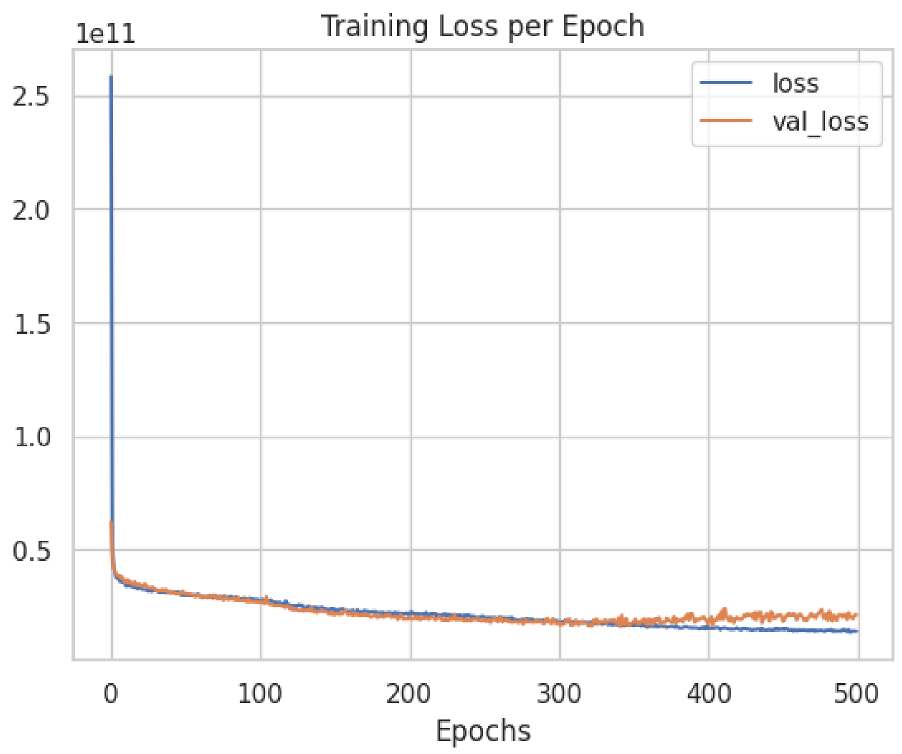
```
ann1.fit(x=x_train,y=y_train,
          validation_data=(x_test,y_test),
          epochs=500)
```

```
Epoch 1/500
541/541 [==============================] - 4s 3ms/step - loss: 258280587264.0000 - val_loss: 61996773376.0000
Epoch 2/500
541/541 [==============================] - 2s 3ms/step - loss: 50945904640.0000 - val_loss: 46065025024.0000
Epoch 3/500
541/541 [==============================] - 2s 3ms/step - loss: 41705193472.0000 - val_loss: 41612169216.0000
Epoch 4/500
541/541 [==============================] - 3s 5ms/step - loss: 38412623872.0000 - val_loss: 39578267648.0000
Epoch 5/500
541/541 [==============================] - 2s 4ms/step - loss: 37317177344.0000 - val_loss: 38652461056.0000
Epoch 6/500
541/541 [==============================] - 3s 5ms/step - loss: 37013217280.0000 - val_loss: 38129004544.0000
Epoch 7/500
541/541 [==============================] - 3s 5ms/step - loss: 35843518464.0000 - val_loss: 38434119680.0000
Epoch 8/500
541/541 [==============================] - 2s 3ms/step - loss: 36073848832.0000 - val_loss: 37060186112.0000
Epoch 9/500
541/541 [==============================] - 2s 3ms/step - loss: 35602231296.0000 - val_loss: 37414379520.0000
Epoch 10/500
541/541 [==============================] - 2s 3ms/step - loss: 35090673664.0000 - val_loss: 35747831808.0000
Epoch 11/500
541/541 [==============================] - 2s 3ms/step - loss: 33823569920.0000 - val_loss: 35405221888.0000
Epoch 12/500
541/541 [==============================] - 2s 3ms/step - loss: 33815752704.0000 - val_loss: 36676558848.0000
Epoch 13/500
541/541 [==============================] - 2s 4ms/step - loss: 34358325248.0000 - val_loss: 35211681792.0000
Epoch 14/500
541/541 [==============================] - 2s 4ms/step - loss: 33719277568.0000 - val_loss: 35795722240.0000
Epoch 15/500
541/541 [==============================] - 2s 4ms/step - loss: 34144247808.0000 - val_loss: 34928713728.0000
Epoch 16/500
541/541 [==============================] - 3s 5ms/step - loss: 33380073472.0000 - val_loss: 34344423424.0000
Epoch 17/500
541/541 [==============================] - 2s 3ms/step - loss: 33126289408.0000 - val_loss: 35691261952.0000
Epoch 18/500
541/541 [==============================] - 2s 3ms/step - loss: 33244624896.0000 - val_loss: 34155714560.0000
Epoch 19/500
541/541 [==============================] - 2s 3ms/step - loss: 32573523968.0000 - val_loss: 34605056000.0000
Epoch 20/500
541/541 [==============================] - 2s 4ms/step - loss: 33108140032.0000 - val_loss: 34769440768.0000
Epoch 21/500
541/541 [==============================] - 2s 3ms/step - loss: 32934549504.0000 - val_loss: 34010429440.0000
Epoch 22/500
```

```
541/541 [==============================] - 2s 3ms/step - loss: 32346161152.0000 - val_loss: 33490900992.0000
Epoch 23/500
541/541 [==============================] - 2s 4ms/step - loss: 32363284480.0000 - val_loss: 34596589568.0000
Epoch 24/500
541/541 [==============================] - 2s 4ms/step - loss: 32206825472.0000 - val_loss: 34834882560.0000
Epoch 25/500
541/541 [==============================] - 3s 6ms/step - loss: 32178552832.0000 - val_loss: 33315573760.0000
Epoch 26/500
541/541 [==============================] - 4s 8ms/step - loss: 31616219136.0000 - val_loss: 33180260352.0000
Epoch 27/500
541/541 [==============================] - 3s 5ms/step - loss: 32336459776.0000 - val_loss: 33041207296.0000
Epoch 28/500
541/541 [==============================] - 2s 3ms/step - loss: 31491287040.0000 - val_loss: 32965572608.0000
Epoch 29/500
541/541 [==============================] - 2s 4ms/step - loss: 31758157824.0000 - val_loss: 33315549184.0000
```

```
losses1 = pd.DataFrame(ann1.history.history)

plt.figure(figsize=(15,5))
losses1.plot()
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
plt.show()
```

<Figure size 1500x500 with 0 Axes>




```
y_pred1= ann1.predict(x_test)
r21 = r2_score(y_test, y_pred1)
print('==='*25)
print('R2score :', r21)
print('==='*25)
rounded_y_pred1 = np.round(y_pred1).astype(int)
compare1 = pd.DataFrame({'Predicted': rounded_y_pred1.flatten(), 'Actual': y_test.astype(int)})

# compare the actaual and predicted value of house price]
compare1[['Actual', 'Predicted']]
```

```
136/136 [==============================] - 0s 1ms/step
===========================================================================
R2score : 0.8622404985233487
===========================================================================
```

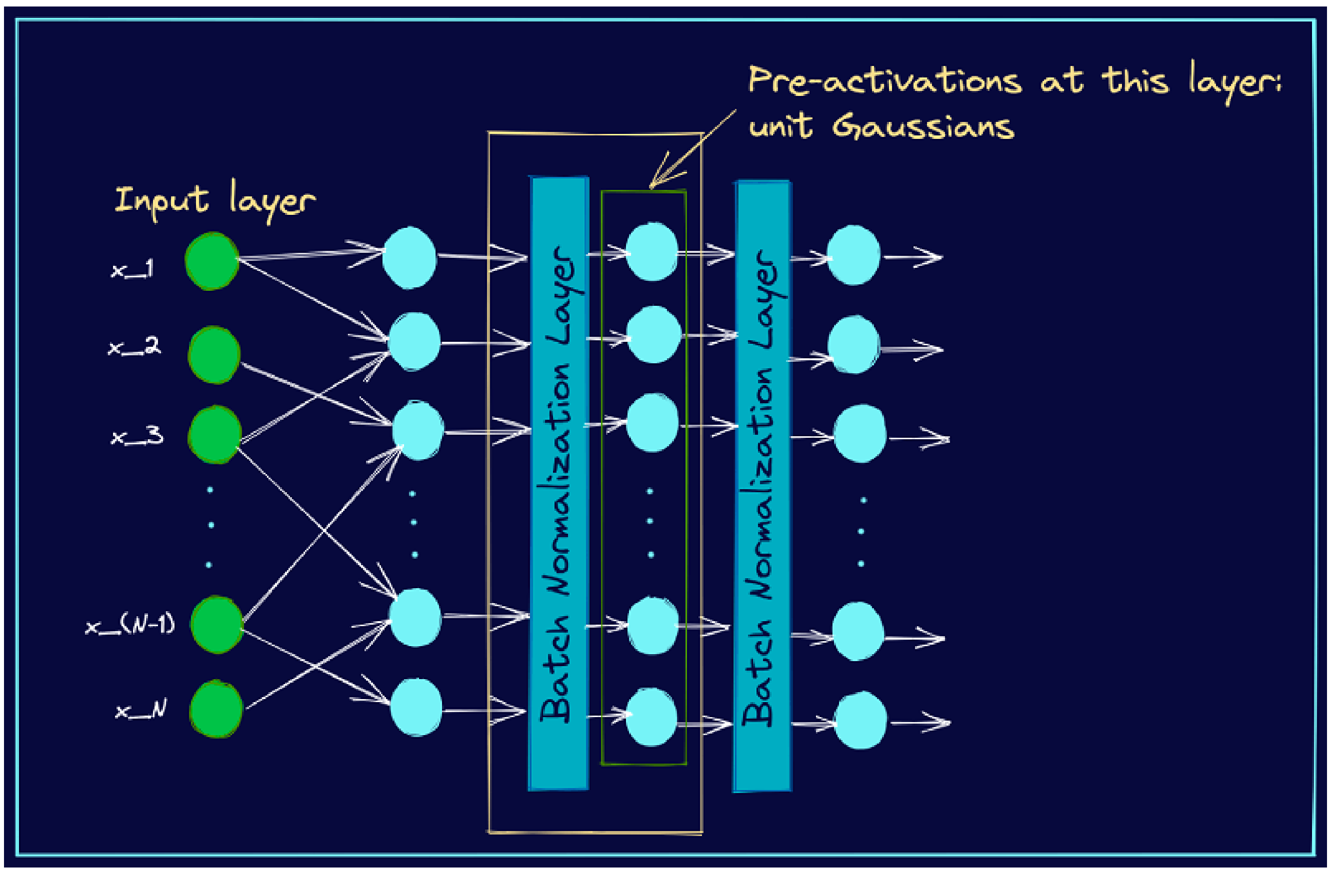
	Actual	Predicted
2019	275000	272418
3435	279000	195378
15940	200500	206928
9811	750000	662184
18665	395000	468833
...	...	...
3390	579000	459410
6801	599000	610682
4775	248500	235530
10634	645000	478633
1529	810000	616569

4323 rows × 2 columns

```
fig, axes =plt.subplots(1, 2, figsize=(10, 4))
sorted_actual1 = compare1['Actual'].sort_values()
sorted_predicted1 = compare1.loc[sorted_actual1.index, 'Predicted']

axes[0].plot(sorted_actual1, label='Actual')
axes[0].plot(sorted_predicted1, color='red', label='Predicted')
axes[0].set_title('Actual vs. Predicted')
axes[0].legend()


axes[1].scatter(compare1['Predicted'], compare1['Actual'], color='orangered')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Scatter Plot: Actual vs. Predicted')
```

```
plt.tight_layout()
plt.show()
```

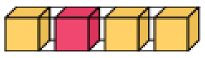
Approach 4 (with Batch Normalization, 3 hidden layers)




```
ann2 = Sequential()
ann2.add(Dense(32, activation ='relu',kernel_initializer=HeNormal(), input_dim=18))
ann2.add(BatchNormalization())
ann2.add(Dense(8,activation ='relu'))
ann2.add(Dense(1,activation ='linear'))
ann2.compile(optimizer ='Adam', loss=MeanSquaredError())

print(ann2.summary())
ann2.fit(x=x_train,y=y_train,
         validation_data=(x_test,y_test),
         epochs=100)
```

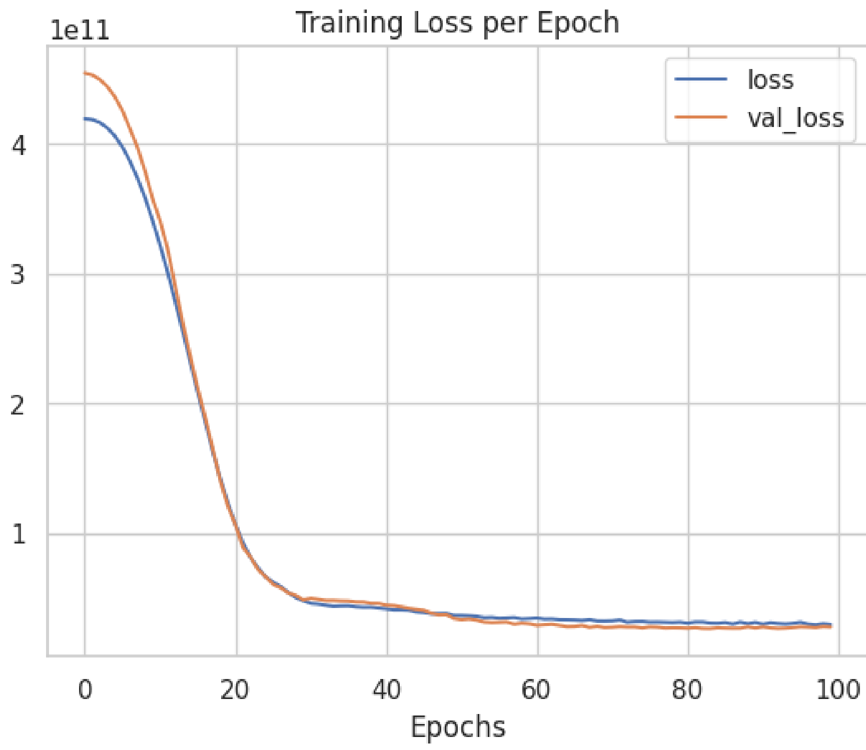
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 32)                608

 batch_normalization (BatchN  (None, 32)               128
 ormalization)

 dense_16 (Dense)            (None, 8)                 264

 dense_17 (Dense)            (None, 1)                 9

=================================================================
Total params: 1,009
Trainable params: 945
Non-trainable params: 64
_________________________________________________________________
None
Epoch 1/100
541/541 [==============================] - 3s 3ms/step - loss: 419352739840.0000 - val_loss: 454445563904.0000
Epoch 2/100
541/541 [==============================] - 1s 3ms/step - loss: 418620866560.0000 - val_loss: 453034016768.0000
Epoch 3/100
541/541 [==============================] - 2s 3ms/step - loss: 416380518400.0000 - val_loss: 449502216192.0000
Epoch 4/100
541/541 [==============================] - 1s 3ms/step - loss: 412239200256.0000 - val_loss: 443997192192.0000
Epoch 5/100
541/541 [==============================] - 2s 4ms/step - loss: 405950857216.0000 - val_loss: 436134051840.0000
Epoch 6/100
541/541 [==============================] - 2s 4ms/step - loss: 397423706112.0000 - val_loss: 425369796608.0000
Epoch 7/100
541/541 [==============================] - 2s 5ms/step - loss: 386535555072.0000 - val_loss: 411542847488.0000
Epoch 8/100
541/541 [==============================] - 2s 3ms/step - loss: 373458337792.0000 - val_loss: 397098254336.0000
Epoch 9/100
541/541 [==============================] - 1s 3ms/step - loss: 358319652864.0000 - val_loss: 378808238080.0000
Epoch 10/100
541/541 [==============================] - 1s 2ms/step - loss: 340893597696.0000 - val_loss: 357776916480.0000
Epoch 11/100
541/541 [==============================] - 1s 3ms/step - loss: 321768685568.0000 - val_loss: 340597112832.0000
Epoch 12/100
541/541 [==============================] - 2s 3ms/step - loss: 300743786496.0000 - val_loss: 318668865536.0000
Epoch 13/100
541/541 [==============================] - 1s 3ms/step - loss: 278652354560.0000 - val_loss: 290626437120.0000
Epoch 14/100
541/541 [==============================] - 1s 3ms/step - loss: 255509872640.0000 - val_loss: 262091390976.0000
Epoch 15/100
541/541 [==============================] - 2s 4ms/step - loss: 232219787264.0000 - val_loss: 236704645120.0000
Epoch 16/100
541/541 [==============================] - 2s 4ms/step - loss: 208415784960.0000 - val_loss: 211481870336.0000
Epoch 17/100
541/541 [==============================] - 2s 4ms/step - loss: 185596641280.0000 - val_loss: 188840755200.0000
Epoch 18/100
541/541 [==============================] - 2s 4ms/step - loss: 162917744640.0000 - val_loss: 165027037184.0000
Epoch 19/100
541/541 [==============================] - 2s 3ms/step - loss: 141924483072.0000 - val_loss: 140228116480.0000
Epoch 20/100
```

```
visualkeras.layered_view(ann2)
```



```
losses2 = pd.DataFrame(ann2.history.history)

plt.figure(figsize=(15,5))
losses2.plot()
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
plt.show()
```

```
<Figure size 1500x500 with 0 Axes>
```




```
y_pred2= ann2.predict(x_test)
r22 = r2_score(y_test, y_pred2)
print('R2score :', r22)
```

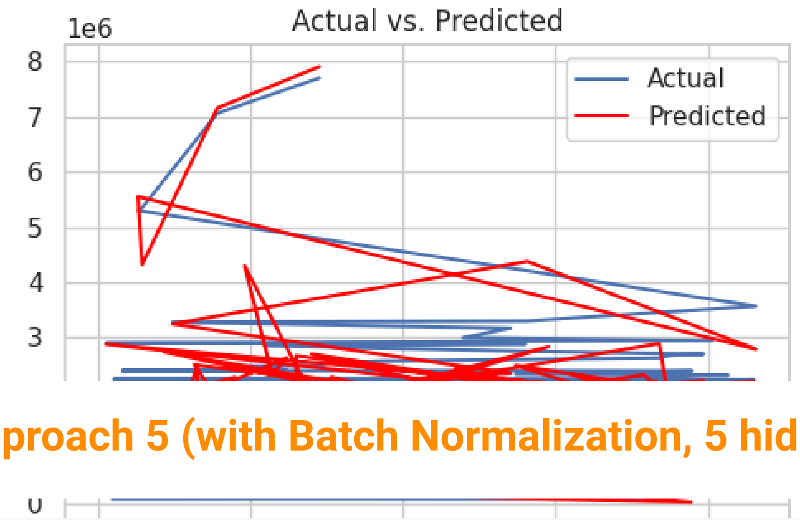
```
136/136 [==============================] - 0s 1ms/step
R2score : 0.8191929593400169
```

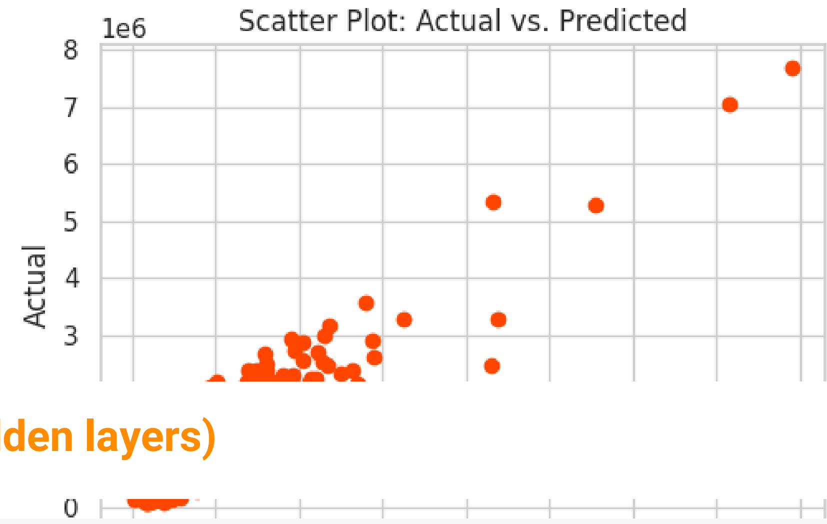
```
rounded_y_pred2 = np.round(y_pred2).astype(int)
compare2 = pd.DataFrame({'Predicted': rounded_y_pred2.flatten(), 'Actual': y_test.astype(int)})

# compare the actaual and predicted value of house price]
compare2[['Actual', 'Predicted']]
```

	Actual	Predicted
2019	275000	317996
3435	279000	221683
15940	200500	268975
9811	750000	773049
18665	395000	447211
...	...	...
3390	579000	451732
6801	599000	689243
4775	248500	350419
10634	645000	494852
1529	810000	705244

4323 rows × 2 columns

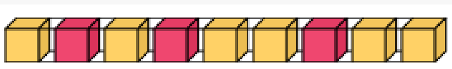
```
fig, axes =plt.subplots(1, 2, figsize=(10, 4))
sorted_actual2 = compare2['Actual'].sort_values()
sorted_predicted2 = compare2.loc[sorted_actual2.index, 'Predicted']

axes[0].plot(sorted_actual2, label='Actual')
axes[0].plot(sorted_predicted2, color='red', label='Predicted')
axes[0].set_title('Actual vs. Predicted')
axes[0].legend()


axes[1].scatter(compare2['Predicted'], compare2['Actual'], color='orangered')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Scatter Plot: Actual vs. Predicted')

plt.tight_layout()
plt.show()
```





Approach 5 (with Batch Normalization, 5 hidden layers)

```
ann3 = Sequential()
#1st hidden layer
ann3.add(Dense(128, activation ='relu',kernel_initializer=HeNormal(), input_dim=18))
ann3.add(BatchNormalization())
# 2nd hidden layer
ann3.add(Dense(units=128,activation ='relu'))
ann3.add(BatchNormalization())
# 3rd hidden layer
ann3.add(Dense(units=64,activation ='relu'))

# 4th hidden layer
ann3.add(Dense(units=32,activation ='relu'))
ann3.add(BatchNormalization())
# 5th hidden layer
ann3.add(Dense(units=32,activation ='relu'))

#outer layer
ann3.add(Dense(1,activation ='linear'))
#compile
ann3.compile(optimizer ='Adam', loss=MeanSquaredError())
visualkeras.layered_view(ann3)
```

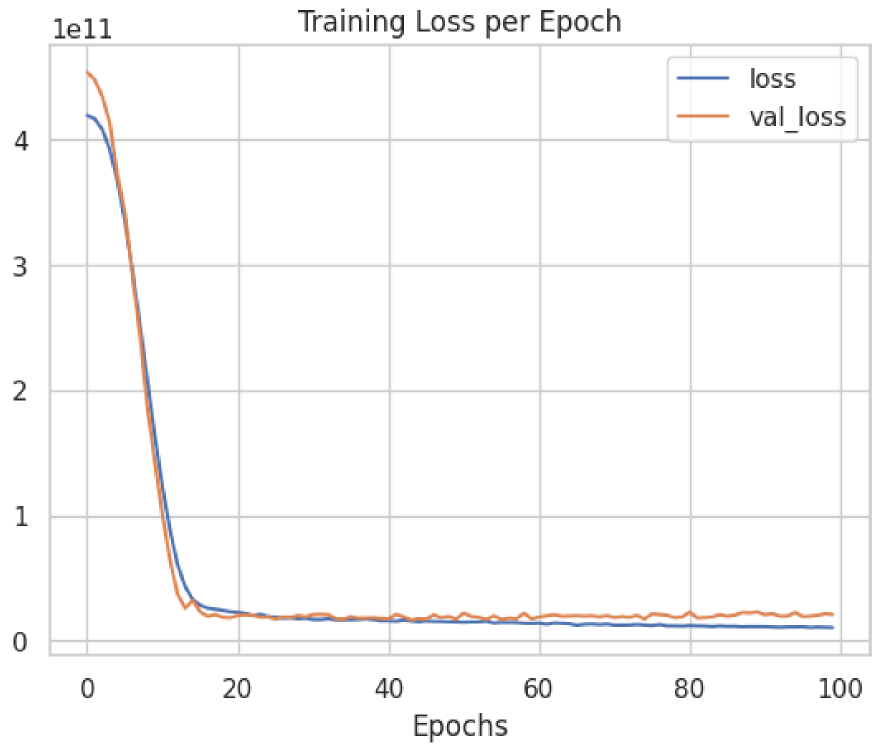


```
print(ann3.summary())
ann3.fit(x=x_train,y=y_train,
          validation_data=(x_test,y_test),
          epochs=100)
```

```
541/541 [==============================] - 2s 4ms/step - loss: 12380891136.0000 - val_loss: 19273816064.0000
Epoch 73/100
541/541 [==============================] - 2s 4ms/step - loss: 12466055168.0000 - val_loss: 18604345344.0000
Epoch 74/100
541/541 [==============================] - 2s 4ms/step - loss: 12942096384.0000 - val_loss: 20186857472.0000
Epoch 75/100
541/541 [==============================] - 2s 4ms/step - loss: 12423649280.0000 - val_loss: 17408585728.0000
Epoch 76/100
541/541 [==============================] - 3s 6ms/step - loss: 12047362048.0000 - val_loss: 21376083968.0000
Epoch 77/100
541/541 [==============================] - 4s 7ms/step - loss: 12860864512.0000 - val_loss: 20869578752.0000
Epoch 78/100
541/541 [==============================] - 3s 5ms/step - loss: 11758569472.0000 - val_loss: 20191617024.0000
Epoch 79/100
541/541 [==============================] - 2s 4ms/step - loss: 11792039936.0000 - val_loss: 18497036288.0000
Epoch 80/100
541/541 [==============================] - 2s 4ms/step - loss: 11608773632.0000 - val_loss: 18980857856.0000
Epoch 81/100
541/541 [==============================] - 2s 4ms/step - loss: 12030330880.0000 - val_loss: 22857017344.0000
Epoch 82/100
541/541 [==============================] - 2s 4ms/step - loss: 11789021184.0000 - val_loss: 18244261888.0000
Epoch 83/100
541/541 [==============================] - 3s 6ms/step - loss: 11636788224.0000 - val_loss: 18527973376.0000
Epoch 84/100
541/541 [==============================] - 4s 7ms/step - loss: 11153427456.0000 - val_loss: 18955468800.0000
Epoch 85/100
541/541 [==============================] - 2s 4ms/step - loss: 11723549696.0000 - val_loss: 20716679168.0000
Epoch 86/100
541/541 [==============================] - 3s 5ms/step - loss: 11357239296.0000 - val_loss: 19482662912.0000
Epoch 87/100
541/541 [==============================] - 2s 5ms/step - loss: 11446361088.0000 - val_loss: 20311250944.0000
Epoch 88/100
541/541 [==============================] - 2s 4ms/step - loss: 11017885696.0000 - val_loss: 22530199552.0000
Epoch 89/100
541/541 [==============================] - 3s 6ms/step - loss: 11231837184.0000 - val_loss: 22209329152.0000
Epoch 90/100
541/541 [==============================] - 3s 6ms/step - loss: 11151793152.0000 - val_loss: 22986944512.0000
Epoch 91/100
541/541 [==============================] - 3s 5ms/step - loss: 11142683648.0000 - val_loss: 20741728256.0000
Epoch 92/100
541/541 [==============================] - 2s 4ms/step - loss: 10811203584.0000 - val_loss: 21585635328.0000
Epoch 93/100
541/541 [==============================] - 2s 4ms/step - loss: 10632462336.0000 - val_loss: 19528208384.0000
Epoch 94/100
541/541 [==============================] - 2s 5ms/step - loss: 10891987968.0000 - val_loss: 19685392384.0000
Epoch 95/100
541/541 [==============================] - 3s 5ms/step - loss: 10914329600.0000 - val_loss: 22582704128.0000
Epoch 96/100
541/541 [==============================] - 3s 6ms/step - loss: 11041880064.0000 - val_loss: 19225022464.0000
Epoch 97/100
541/541 [==============================] - 4s 8ms/step - loss: 10488804352.0000 - val_loss: 19571937280.0000
Epoch 98/100
541/541 [==============================] - 2s 4ms/step - loss: 10791462912.0000 - val_loss: 20206596096.0000
Epoch 99/100
541/541 [==============================] - 2s 4ms/step - loss: 10578880512.0000 - val_loss: 21587417088.0000
Epoch 100/100
541/541 [==============================] - 2s 4ms/step - loss: 10428310528.0000 - val_loss: 20967694336.0000
<keras.callbacks.History at 0x7a81527c7670>
```

```
losses3 = pd.DataFrame(ann3.history.history)

plt.figure(figsize=(15,5))
losses3.plot()
plt.xlabel('Epochs')
plt.ylabel('')
plt.title('Training Loss per Epoch')
plt.show()
```

```
<Figure size 1500x500 with 0 Axes>
```




```
y_pred3= ann3.predict(x_test)
r23 = r2_score(y_test, y_pred3)
print('R2score :', r23)
```

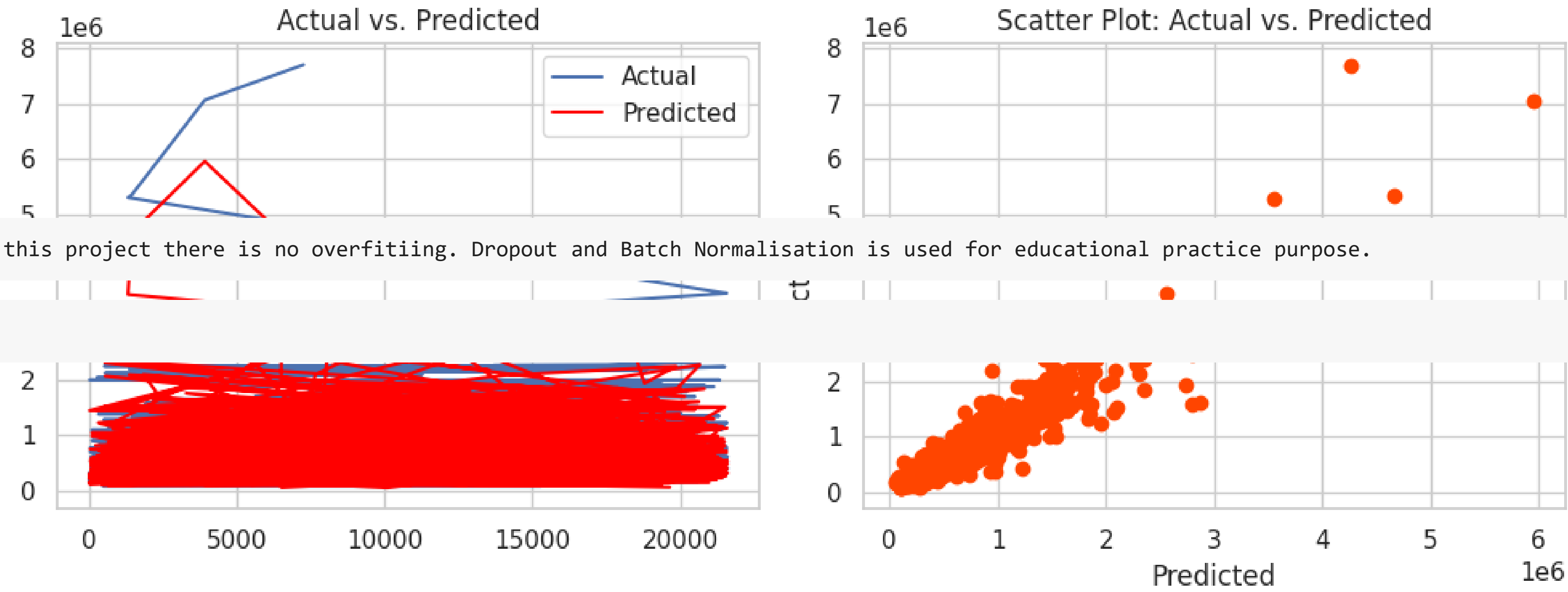
```
136/136 [==============================] - 0s 2ms/step
R2score : 0.8635385131751712
```

```
rounded_y_pred3 = np.round(y_pred3).astype(int)
compare3 = pd.DataFrame({'Predicted': rounded_y_pred3.flatten(), 'Actual': y_test.astype(int)})

# compare the actaual and predicted value of house price]
compare3[['Actual', 'Predicted']]
```

	Actual	Predicted
2019	275000	259962
3435	279000	188666
15940	200500	193134
9811	750000	695535
18665	395000	510177
...	...	...
3390	579000	456655
6801	599000	646255
4775	248500	207810
10634	645000	492408
1529	810000	683417

4323 rows × 2 columns

```
fig, axes =plt.subplots(1, 2, figsize=(10, 4))
sorted_actual3 = compare3['Actual'].sort_values()
sorted_predicted3 = compare3.loc[sorted_actual3.index, 'Predicted']

axes[0].plot(sorted_actual3, label='Actual')
axes[0].plot(sorted_predicted3, color='red', label='Predicted')
axes[0].set_title('Actual vs. Predicted')
axes[0].legend()


axes[1].scatter(compare3['Predicted'], compare3['Actual'], color='orangered')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Scatter Plot: Actual vs. Predicted')

plt.tight_layout()
plt.show()
```




```
# In this project there is no overfitiing. Dropout and Batch Normalisation is used for educational practice purpose.
```

0s completed at 1:14 AM