Creating a Regression Model For Boston Housing Price

```
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import pandas as pd
```

```
boston_house_data=pd.read_csv('https://raw.githubusercontent.com/JayantArsode/Machine_Learn
boston_house_data.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Str
0	1	60	RL	65.0	8450	F
1	2	20	RL	80.0	9600	F
2	3	60	RL	68.0	11250	F
3	4	70	RL	60.0	9550	F
4	5	60	RL	84.0	14260	F

5 rows × 81 columns

Getting Data Description

```
import requests
url = 'https://raw.githubusercontent.com/JayantArsode/Machine_Learning/Machine-Learning/Hou
boston_house_variable_decs = requests.get(url)
print (boston_house_variable_decs.text)
```

```
       NA       No Fence

MiscFeature: Miscellaneous feature not covered in other categories

       Elev     Elevator
       Gar2     2nd Garage (if not described in garage section)
       Othr     Other
       Shed     Shed (over 100 SF)
       TenC     Tennis Court
       NA       None

MiscVal: $Value of miscellaneous feature

MoSold: Month Sold (MM)

YrSold: Year Sold (YYYY)

SaleType: Type of sale

       WD       Warranty Deed - Conventional
       CWD      Warranty Deed - Cash
       VWD      Warranty Deed - VA Loan
       New      Home just constructed and sold
       COD      Court Officer Deed/Estate
       Con      Contract 15% Down payment regular terms
       ConLw    Contract Low Down payment and low interest
       ConLI    Contract Low Interest
       ConLD    Contract Low Down
       Oth      Other

SaleCondition: Condition of sale

       Normal   Normal Sale
       Abnorml  Abnormal Sale -  trade, foreclosure, short sale
       AdjLand  Adjoining Land Purchase
       Alloca   Allocation - two linked properties with separate deeds, typically
       Family   Sale between family members
       Partial  Home was not completed when last assessed (associated with New Hom
```

```
# Looking for variable information
boston_house_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
```

```
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
```

Dropping columns with too much null values and Unwanted Columns

Columns	Not-Null Count
Id	1460 non-null
Alley	91 non-null
FireplaceQu	770 non-null
PoolQC	7 non-null
Fence	281 non-null
MiscFeature	54 non-null

```
boston_house_data.drop(columns=['Id','Alley','FireplaceQu','PoolQC','Fence','MiscFeature'],
boston_house_data.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	LotShape	LandContour	Utilitie
0	60	RL	65.0	8450	Pave	Reg	Lvl	AllPu
1	20	RL	80.0	9600	Pave	Reg	Lvl	AllPu
2	60	RL	68.0	11250	Pave	IR1	Lvl	AllPu
3	70	RL	60.0	9550	Pave	IR1	Lvl	AllPu
4	60	RL	84.0	14260	Pave	IR1	Lvl	AllPu

5 rows × 75 columns

Getting Numerical And Categorical Columns Seprate

```
# Now seprating int, float,categorical columns in data
numerical_columns=boston_house_data.select_dtypes(include=['float64','int64']).columns
categorical_columns=boston_house_data.select_dtypes(include='object').columns
print(f'Numerical columns names: {numerical_columns}\nCategoricla Column Names: {categorica
```

```
Numerical columns names: Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual'
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF',
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold', 'SalePrice'],
      dtype='object')
Categoricla Column Names: Index(['MSZoning', 'Street', 'LotShape', 'LandContour', 'Ut
       'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2',
       'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st',
       'Exterior2nd', 'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation',
       'BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',
       'Heating', 'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual',
       'Functional', 'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond',
       'PavedDrive', 'SaleType', 'SaleCondition'],
      dtype='object')
```

Getting Correlation between Features And Target Variable(SalePrice)

```
boston_house_data[numerical_columns].corr()['SalePrice'][:]
```

```
MSSubClass      -0.084284
LotFrontage      0.351799
LotArea          0.263843
OverallQual      0.790982
OverallCond     -0.077856
YearBuilt        0.522897
YearRemodAdd     0.507101
```

```
MasVnrArea       0.477493
BsmtFinSF1       0.386420
BsmtFinSF2      -0.011378
BsmtUnfSF        0.214479
TotalBsmtSF      0.613581
1stFlrSF         0.605852
2ndFlrSF         0.319334
LowQualFinSF    -0.025606
GrLivArea        0.708624
BsmtFullBath     0.227122
BsmtHalfBath    -0.016844
FullBath         0.560664
HalfBath         0.284108
BedroomAbvGr     0.168213
KitchenAbvGr    -0.135907
TotRmsAbvGrd     0.533723
Fireplaces       0.466929
GarageYrBlt      0.486362
GarageCars       0.640409
GarageArea       0.623431
WoodDeckSF       0.324413
OpenPorchSF      0.315856
EnclosedPorch   -0.128578
3SsnPorch        0.044584
ScreenPorch      0.111447
PoolArea         0.092404
MiscVal         -0.021190
MoSold           0.046432
YrSold          -0.028923
SalePrice        1.000000
Name: SalePrice, dtype: float64
```

From Above We Can Conclude That

Features	Correlation-Coefficient
OverallQual	0.783546
YearBuilt	0.504297
YearRemodAdd	0.501435
MasVnrArea	0.465811
TotalBsmtSF	0.602042
1stFlrSF	0.604714
GrLivArea	0.711706
FullBath	0.569313
Fireplaces	0.445434
GarageYrBlt	0.481730
TotRmsAbvGrd	0.551821
GarageCars	0.640154
GarageArea	0.607535

Are highly corelated to the Target Variable SalePrice

Selecting Highly Correlated Features

```
boston_house_data_selected_columns=['OverallQual','YearBuilt','YearRemodAdd','MasVnrArea','
                                    ,'FullBath','GarageYrBlt','TotRmsAbvGrd','GarageCars','
boston_house_data_selected=boston_house_data[boston_house_data_selected_columns].copy()
boston_house_data_selected.head()
```

	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	TotalBsmtSF	1stFlrSF	GrLivArea
0	7	2003	2003	196.0	856	856	1710
1	6	1976	1976	0.0	1262	1262	1262
2	7	2001	2002	162.0	920	920	1786
3	7	1915	1970	0.0	756	961	1717
4	8	2000	2000	350.0	1145	1145	2198

5 rows × 52 columns

Updating Numerical_Columns

```
# Now seprating int, float,categorical columns in data
numerical_columns=boston_house_data_selected.select_dtypes(include=['float64','int64']).col
print(f'Numerical columns names: {numerical_columns}')
```

```
Numerical columns names: Index(['OverallQual', 'YearBuilt', 'YearRemodAdd', 'MasVnrAr
       '1stFlrSF', 'GrLivArea', 'Fireplaces', 'FullBath', 'GarageYrBlt',
       'TotRmsAbvGrd', 'GarageCars', 'GarageArea', 'SalePrice'],
      dtype='object')
```

Creating Data Preprocessing Function

> Replace Numerical Columns Nan Values With Median

> Impute the Categorial Columns With Most Frequent Startegy

```
def data_preprocesses(data,
                      numerical_columns=numerical_columns,
                      categorical_columns=categorical_columns):
  '''This function will impute the nan values in numerical and categorical columns and retu

  # Imputing the numerical columns
  data[numerical_columns] = data[numerical_columns].fillna(data[numerical_columns].median()

  # Imputing the categorical columns
  data[categorical_columns] = data[categorical_columns].fillna(data[categorical_columns].mc

  return data
```

Preprocessing Boston Housin Data Selected

```
boston_house_data_selected=data_preprocesses(boston_house_data_selected)
```

```
boston_house_data_selected.columns[boston_house_data_selected.isna().any()]
```

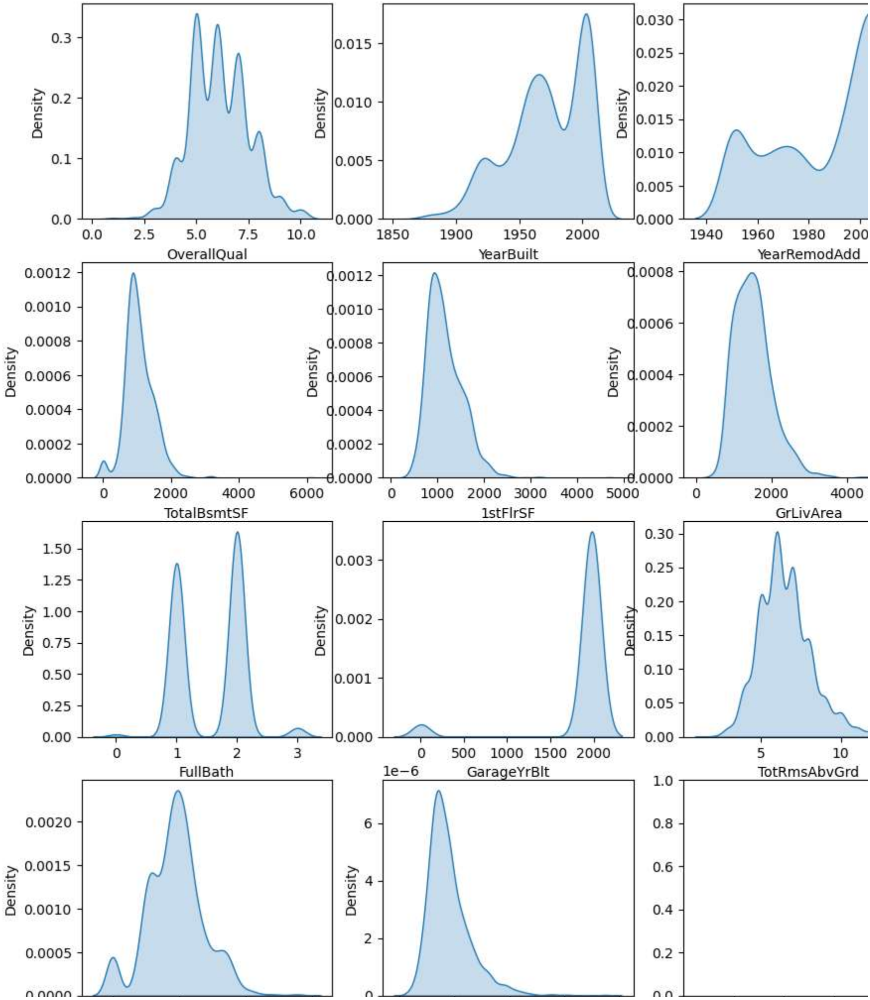
```
Index([], dtype='object')
```

```
boston_house_data_selected.columns
```

```
Index(['OverallQual', 'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'TotalBsmtSF',
       '1stFlrSF', 'GrLivArea', 'Fireplaces', 'FullBath', 'GarageYrBlt',
       'TotRmsAbvGrd', 'GarageCars', 'GarageArea', 'SalePrice', 'MSZoning',
       'Street', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
       'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', 'KitchenQual', 'Functional',
       'GarageType', 'GarageFinish', 'GarageQual', 'GarageCond', 'PavedDrive',
       'SaleType', 'SaleCondition'],
      dtype='object')
```

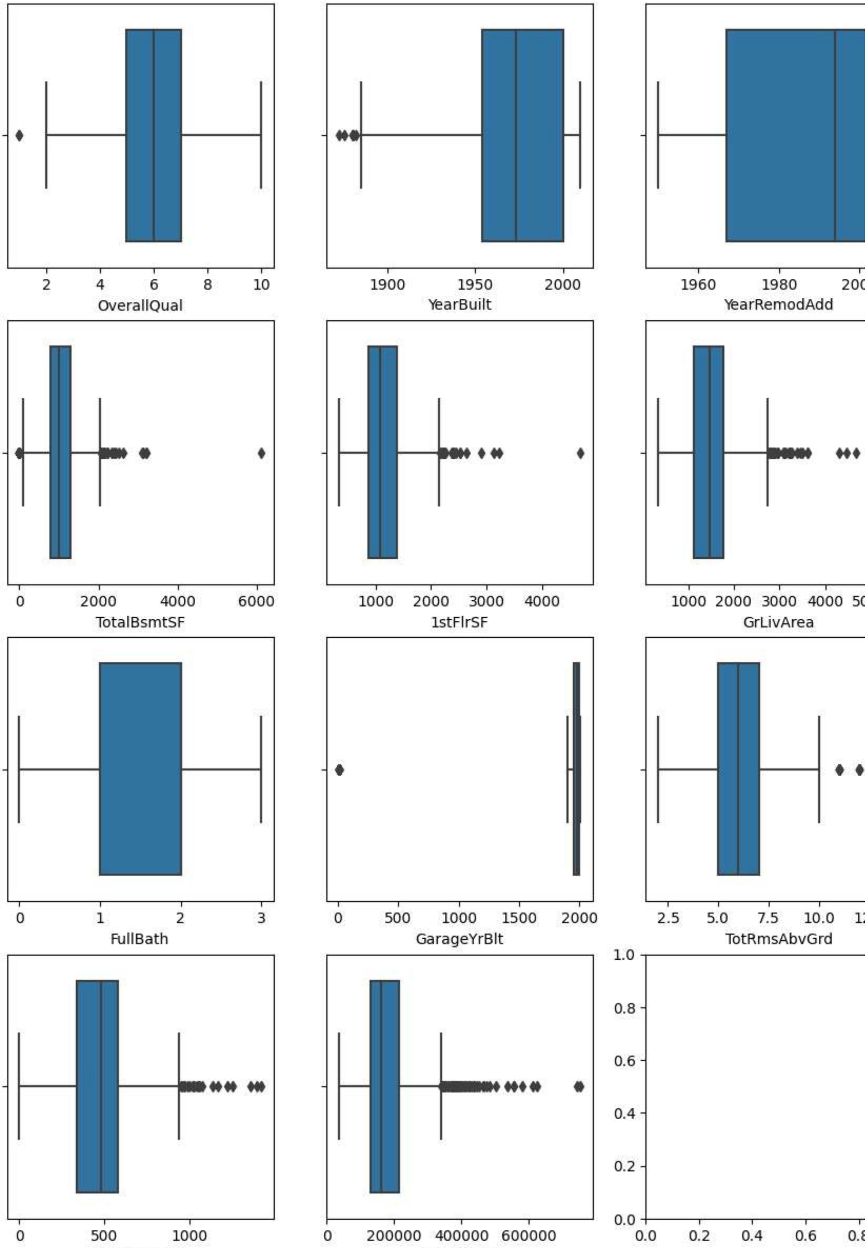
Checking Distribution of All Numerical Columns

```
fig,ax=plt.subplots(4,4,figsize=(15,13))
ax=ax.ravel()
j=0
for i in numerical_columns:
  sns.kdeplot(x=boston_house_data_selected[i],ax=ax[j],fill=True)
  j+=1
plt.show()
```



Checking for outliers using boxplots

```
fig,ax=plt.subplots(4,4,figsize=(15,15))
ax=ax.ravel()
j=0
for i in numerical_columns:
  sns.boxplot(x=boston_house_data_selected[i],ax=ax[j])
  j+=1
plt.show()
```



Creating a Column Transform To Preprocesses data:

> Scale Numerical Columns Using Standard Scaller To Reduce Effect Of Outliers.

> OneHot Encoding Data

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler,OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from mlxtend.preprocessing import DenseTransformer
from sklearn.impute import SimpleImputer
import numpy as np


# Assuming you have separate lists for numerical_columns and categorical_columns
ct = ColumnTransformer([
    ('numerical_scaling', StandardScaler(), numerical_columns[:13]),
    ('OneHotEncoding', OneHotEncoder(handle_unknown='ignore'), categorical_columns)
])

# Dividing dataset in features(X) vs labels(y)

X=boston_house_data_selected.drop(columns=['SalePrice'])
y=boston_house_data_selected['SalePrice']
```

Train Test Spliting Data

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,shuffle=True,random_state=
```

```
X_train.shape,y_train.shape
```

```
((1168, 51), (1168,))
```

```
# Some global Variables
input_shape=124
```

Now using PCA(Principal Component Analysis) To Reduce Dimensions Of Features

```
data_preprocessing_pipeline=Pipeline([
    ('column_transform',ct),
    ('to_dense',DenseTransformer()),
    ('pca',PCA(n_components=124))
])
```

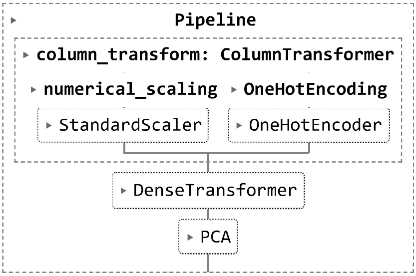
It is used to get number of components to retain for getting 99% variance of data

> Here it is 124 i.e number_components=124

```
data_preprocessing_pipeline.fit(X_train)
# # Calculate the cumulative explained variance ratio
# pca=data_preprocessing_pipeline.named_steps['pca']
# cumulative_variance_ratio = np.cumsum(pca.explained_variance_ratio_)

# # Determine the number of components to retain (e.g., 95% variance)
# desired_variance = 0.99
# num_components = np.argmax(cumulative_variance_ratio >= desired_variance) + 1

# # Print the results
# print(f"Number of components to retain for {desired_variance * 100}% variance: {num_compo
```




```
X_train_norm=data_preprocessing_pipeline.transform(X_train)
X_test_norm=data_preprocessing_pipeline.transform(X_test)
```

```
X_train_norm.shape
```

```
(1168, 124)
```

Creating A Neural Network To Train Model

1. Creating a model with 2 hidden layer and 100 epochs
2. Creating a model with 2 hidden layer more than 100 epochs
3. Creating a model with 3 hidden layer and 100 epochs and more

Model_1

```
# Setting Random Seed
tf.random.set_seed(42)

# 1. Creating Model
price_prediction_1=tf.keras.Sequential(name='price_prediction_1')
price_prediction_1.add(tf.keras.layers.Dense(100,input_shape=[input_shape],activation='relu
price_prediction_1.add(tf.keras.layers.Dense(50,activation='relu',name='layer2',kernel_init
price_prediction_1.add(tf.keras.layers.Dense(1,name='output_layer'))

# 2. Compile Model
```

```
price_prediction_1.compile(loss=tf.keras.losses.mae,
                           optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
                           metrics=['mae'])
# 3. Get model summary
price_prediction_1.summary()
```

```
Model: "price_prediction_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 100)               12500

 layer2 (Dense)              (None, 50)                5050

 output_layer (Dense)        (None, 1)                 51

=================================================================
Total params: 17,601
Trainable params: 17,601
Non-trainable params: 0
_________________________________________________________________
```

```
history_1=price_prediction_1.fit(X_train_norm,y_train,epochs=100,verbose=1)
```

```
Epoch 91/100
37/37 [==============================] - 0s 3ms/step - loss: 14547.0361 - mae: 145
Epoch 92/100
37/37 [==============================] - 0s 3ms/step - loss: 14585.1064 - mae: 145
Epoch 93/100
37/37 [==============================] - 0s 3ms/step - loss: 13767.0273 - mae: 137
Epoch 94/100
37/37 [==============================] - 0s 3ms/step - loss: 13753.8232 - mae: 137
Epoch 95/100
37/37 [==============================] - 0s 3ms/step - loss: 14515.4609 - mae: 145
Epoch 96/100
37/37 [==============================] - 0s 3ms/step - loss: 13817.5723 - mae: 138
Epoch 97/100
37/37 [==============================] - 0s 4ms/step - loss: 14963.0996 - mae: 149
Epoch 98/100
37/37 [==============================] - 0s 3ms/step - loss: 14722.1953 - mae: 147
Epoch 99/100
37/37 [==============================] - 0s 3ms/step - loss: 14107.7061 - mae: 141
Epoch 100/100
37/37 [==============================] - 0s 3ms/step - loss: 15726.8018 - mae: 157
```

Evaluating Model 1

```
price_prediction_1.evaluate(X_test_norm,y_test)
```

```
10/10 [==============================] - 0s 2ms/step - loss: 20038.4805 - mae: 20038.
[20038.48046875, 20038.48046875]
```

Comparing First 20 samples

```
y_pred_1=price_prediction_1.predict(X_test_norm[:20])
print(f' True Price: {y_test[:20]}\nPredicted Price: {tf.squeeze(y_pred_1)}')
```

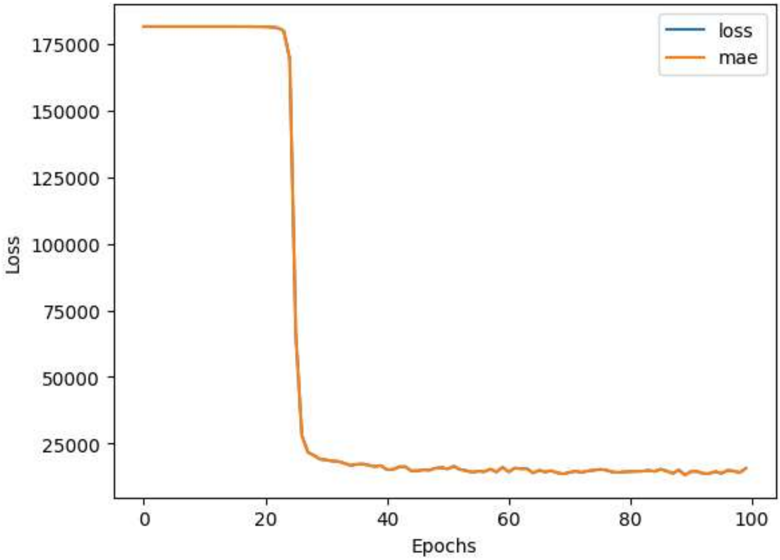
```
1/1 [==============================] - 0s 60ms/step
 True Price: 892     154500
1105    325000
413     115000
522     159000
1036    315500
614      75500
218     311500
1160    146000
649      84500
887     135500
576     145000
1252    130000
1061     81000
567     214000
1108    181000
1113    134500
168     183500
1102    135000
1120    118400
```

```
67      226000
Name: SalePrice, dtype: int64
Predicted Price: [159231.48 344066.2   93816.05 179850.98 371536.47  73792.11 246715.
 155276.42  73804.92 117418.7  142879.56 123843.84 161553.4  232389.88
 169436.89 136351.77 179979.69 140464.47 140203.81 208532.66]
```

Loos Curve

```
pd.DataFrame(history_1.history).plot()
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

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



Model_2

```
# Setting Random Seed
tf.random.set_seed(42)

# 1. Creating Model
price_prediction_2=tf.keras.Sequential(name='price_prediction_2')
price_prediction_2.add(tf.keras.layers.Dense(100,input_shape=[input_shape],activation='relu
price_prediction_2.add(tf.keras.layers.Dense(100,activation='relu',name='layer2'))
price_prediction_2.add(tf.keras.layers.Dense(1,name='output_layer'))

# 2. Compile Model
price_prediction_2.compile(loss=tf.keras.losses.mae,
                           optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
```

```
                           metrics=['mae'])
# 3. Get model summary
price_prediction_2.summary()
```

```
Model: "price_prediction_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 100)               12500

 layer2 (Dense)              (None, 100)               10100

 output_layer (Dense)        (None, 1)                 101

=================================================================
Total params: 22,701
Trainable params: 22,701
Non-trainable params: 0
_________________________________________________________________
```

```
history_2=price_prediction_2.fit(X_train_norm,y_train,epochs=150,verbose=1)
```

```
Epoch 142/150
37/37 [==============================] - 0s 2ms/step - loss: 12188.9473 - mae: 121
Epoch 143/150
37/37 [==============================] - 0s 2ms/step - loss: 12154.6768 - mae: 121
Epoch 144/150
37/37 [==============================] - 0s 2ms/step - loss: 12109.6826 - mae: 121
Epoch 145/150
37/37 [==============================] - 0s 2ms/step - loss: 12094.5029 - mae: 120
Epoch 146/150
37/37 [==============================] - 0s 2ms/step - loss: 12047.1445 - mae: 120
Epoch 147/150
37/37 [==============================] - 0s 2ms/step - loss: 12020.1279 - mae: 120
Epoch 148/150
37/37 [==============================] - 0s 3ms/step - loss: 11971.5410 - mae: 119
Epoch 149/150
37/37 [==============================] - 0s 2ms/step - loss: 11945.9170 - mae: 119
Epoch 150/150
37/37 [==============================] - 0s 2ms/step - loss: 11911.5527 - mae: 119
```

Evaluating Model 2

```
price_prediction_2.evaluate(X_test_norm,y_test)
```

```
10/10 [==============================] - 0s 2ms/step - loss: 19942.9766 - mae: 19942.
[19942.9765625, 19942.9765625]
```

Comparing First 20 Smaples

```
y_pred_2=price_prediction_2.predict(X_test_norm[:20])
print(f' True Price: {y_test[:20]}\nPredicted Price: {tf.squeeze(y_pred_2)}')
```

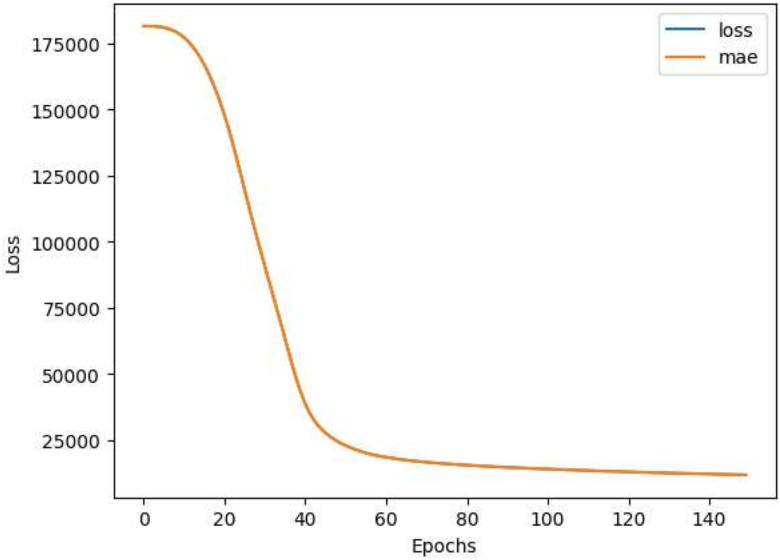
```
1/1 [==============================] - 0s 67ms/step
 True Price: 892     154500
1105    325000
413     115000
522     159000
1036    315500
614      75500
218     311500
1160    146000
649      84500
887     135500
576     145000
1252    130000
1061     81000
567     214000
1108    181000
1113    134500
168     183500
1102    135000
1120    118400
67      226000
Name: SalePrice, dtype: int64
```

```
Predicted Price: [154854.89 328281.7  104548.32 169559.55 347517.5   73153.6  227945.
 159758.98  72184.74 117675.06 141974.12 130546.35 128108.22 220525.31
 171722.19 138631.97 190333.86 131984.3  122605.23 205509.48]
```

Loss Curve

```
pd.DataFrame(history_2.history).plot()
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

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



Model_3

```
# Setting Random Seed
tf.random.set_seed(42)

# 1. Creating Model
price_prediction_3=tf.keras.Sequential(name='price_prediction_3')
price_prediction_3.add(tf.keras.layers.Dense(200,input_shape=[input_shape],activation='relu
price_prediction_3.add(tf.keras.layers.Dense(100,activation='relu',name='layer2',kernel_ini
price_prediction_3.add(tf.keras.layers.Dense(50,activation='relu',name='layer3',kernel_init
price_prediction_3.add(tf.keras.layers.Dense(1,name='output_layer'))

# 2. Compile Model
price_prediction_3.compile(loss=tf.keras.losses.mae,
                           optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
                           metrics=['mae'])
```

```
# 3. Get model summary
price_prediction_3.summary()
```

```
Model: "price_prediction_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 200)               25000

 layer2 (Dense)              (None, 100)               20100

 layer3 (Dense)              (None, 50)                5050

 output_layer (Dense)        (None, 1)                 51

=================================================================
Total params: 50,201
Trainable params: 50,201
Non-trainable params: 0
_________________________________________________________________
```

```
history_3=price_prediction_3.fit(X_train_norm,y_train,epochs=100,verbose=1)
```

```
37/37 [==============================] - 0s 4ms/step - loss: 8054.6230 - mae: 8054
Epoch 92/100
37/37 [==============================] - 0s 4ms/step - loss: 7950.0679 - mae: 7950
Epoch 93/100
37/37 [==============================] - 0s 4ms/step - loss: 7903.8511 - mae: 7903
Epoch 94/100
37/37 [==============================] - 0s 4ms/step - loss: 7867.7622 - mae: 7867
Epoch 95/100
37/37 [==============================] - 0s 4ms/step - loss: 7822.5601 - mae: 7822
Epoch 96/100
37/37 [==============================] - 0s 4ms/step - loss: 7833.5908 - mae: 7833
Epoch 97/100
37/37 [==============================] - 0s 4ms/step - loss: 7702.0952 - mae: 7702
Epoch 98/100
37/37 [==============================] - 0s 4ms/step - loss: 7713.1230 - mae: 7713
Epoch 99/100
37/37 [==============================] - 0s 4ms/step - loss: 7639.4805 - mae: 7639
Epoch 100/100
37/37 [==============================] - 0s 5ms/step - loss: 7580.2363 - mae: 7580
```

```
price_prediction_3.evaluate(X_test_norm,y_test)
```

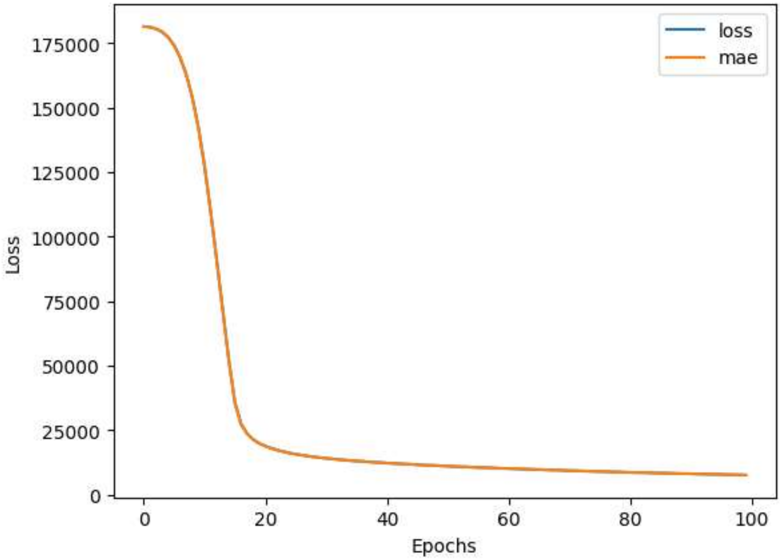
```
10/10 [==============================] - 0s 2ms/step - loss: 18156.6836 - mae: 18156.
[18156.68359375, 18156.68359375]
```

```
y_pred_3=price_prediction_2.predict(X_test_norm[:20])
print(f' True Price: {y_test[:20]}\nPredicted Price: {tf.squeeze(y_pred_3)}')
```

```
1/1 [==============================] - 0s 56ms/step
 True Price: 892     154500
1105    325000
413     115000
522     159000
1036    315500
614      75500
218     311500
1160    146000
649      84500
887     135500
576     145000
1252    130000
1061     81000
567     214000
1108    181000
1113    134500
168     183500
1102    135000
1120    118400
67      226000
Name: SalePrice, dtype: int64
Predicted Price: [154854.89 328281.7  104548.32 169559.55 347517.5   73153.6  227945.
 159758.98  72184.74 117675.06 141974.12 130546.35 128108.22 220525.31
 171722.19 138631.97 190333.86 131984.3  122605.23 205509.48]
```

```
pd.DataFrame(history_3.history).plot()
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

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



Submitting Testing Data To Kaggle

```
test_data=pd.read_csv("https://raw.githubusercontent.com/JayantArsode/Machine_Learning/Mach
test_data.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandCont
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	

5 rows × 80 columns

```
temp_column=boston_house_data_selected_columns.copy()
temp_column.remove('SalePrice')
temp_column.insert(0,'Id')
```

```
test_data=test_data[temp_column]
test_data.head()
```

	Id	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	TotalBsmtSF	1stFlrSF	GrLi
0	1461	5	1961	1961	0.0	882.0	896	
1	1462	6	1958	1958	108.0	1329.0	1329	
2	1463	5	1997	1998	0.0	928.0	928	
3	1464	6	1998	1998	20.0	926.0	926	
4	1465	8	1992	1992	0.0	1280.0	1280	

5 rows × 52 columns

```
test_data_preprocessed=data_preprocesses(test_data.drop(columns=['Id']),numerical_columns[:
test_data_preprocessed.columns[test_data_preprocessed.isna().any()]
```

```
Index([], dtype='object')
```

```
test_data_norm=data_preprocessing_pipeline.transform(test_data_preprocessed)
```

```
test_data['SalePrice']=price_prediction_3.predict(test_data_norm)
test_data.head()
```

```
46/46 [==============================] - 0s 2ms/step
```

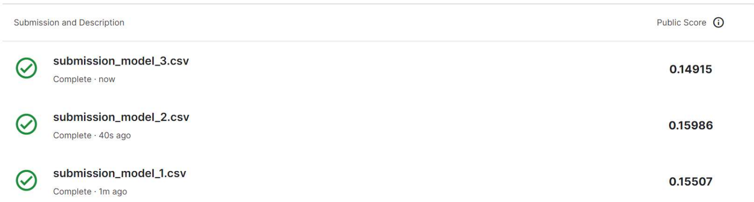
	Id	OverallQual	YearBuilt	YearRemodAdd	MasVnrArea	TotalBsmtSF	1stFlrSF	GrLi
0	1461	5	1961	1961	0.0	882.0	896	
1	1462	6	1958	1958	108.0	1329.0	1329	
2	1463	5	1997	1998	0.0	928.0	928	
3	1464	6	1998	1998	20.0	926.0	926	
4	1465	8	1992	1992	0.0	1280.0	1280	

5 rows × 53 columns

```
submission=pd.DataFrame([])
submission['Id']=test_data['Id']
submission['SalePrice']=test_data['SalePrice']
submission.to_csv('submission_model_3.csv',index=False)
```

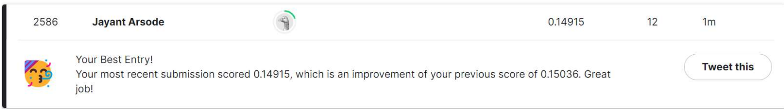
Conclusion:

Scores



Submission and Description	Public Score
submission_model_3.csv Complete · now	0.14915
submission_model_2.csv Complete · 40s ago	0.15986
submission_model_1.csv Complete · 1m ago	0.15507

Ranking



2586	Jayant Arsode	0.14915	12	1m

Your Best Entry!
Your most recent submission scored 0.14915, which is an improvement of your previous score of 0.15036. Great job!

Tweet this

From Above We Can Conclude that `price_prediction_model_3` performed best.

✓ 0s completed at 9:11 AM