Hyperparameter tuned Deep Neural Network (Regression)

Name: Surabhi S

reg no: 20MDT1021

AIM:

- By using Bayesian hyperparameter tuning we should build a regression model(deep neural network)
- Model performance
- Interpretation of models using LIME

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import ensemble
from sklearn import metrics
from sklearn import model_selection
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import sklearn.metrics as metrics
import tensorflow as tf
```

Importing basic and necessary packages

Importing the data

Russian house price preciction data

```
x_train = pd.read_csv('/content/train_data.csv')
y_train = pd.read_csv('/content/ytrain_data.csv')
x_test = pd.read_csv('/content/test_data.csv')
y_test = pd.read_csv('/content/ytest_data.csv')

x_train.drop('Unnamed: 0',axis=1,inplace=True)
x_test.drop('Unnamed: 0',axis=1,inplace=True)
y_train.drop('Unnamed: 0',axis=1,inplace=True)
y_test.drop('Unnamed: 0',axis=1,inplace=True)
#x_train.head(2)
```

saved dataset and then removed house id column from dataset

Preprocessing of data

Since we are going to perform regression, the dataset should have only numarical data. It can be checked with "info()"

```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21329 entries, 0 to 21328
Data columns (total 50 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   full_sq                      21329 non-null  int64  
 1   floor                        21329 non-null  float64
 2   build_year                   21329 non-null  float64
 3   num_room                     21329 non-null  float64
 4   kitch_sq                     21329 non-null  float64
 5   state                        21329 non-null  float64
 6   product_type                 21329 non-null  int64  
 7   raion_popul                  21329 non-null  float64
 8   indust_part                  21329 non-null  float64
 9   sport_objects_raion          21329 non-null  int64  
 10  shopping_centers_raion       21329 non-null  int64  
 11  radiation_raion              21329 non-null  int64  
 12  build_count_block            21329 non-null  float64
 13  build_count_brick            21329 non-null  float64
 14  build_count_monolith         21329 non-null  float64
 15  metro_min_avto               21329 non-null  float64
 16  school_km                    21329 non-null  float64
 17  green_zone_km                21329 non-null  float64
 18  industrial_km                21329 non-null  float64
 19  water_treatment_km           21329 non-null  float64
 20  cemetery_km                  21329 non-null  float64
 21  incineration_km              21329 non-null  float64
 22  ID_railroad_station_avto     21329 non-null  float64
 23  mkad_km                      21329 non-null  float64
 24  ttk_km                       21329 non-null  float64
 25  oil_chemistry_km             21329 non-null  float64
 26  nuclear_reactor_km           21329 non-null  float64
 27  power_transmission_line_km   21329 non-null  float64
 28  market_shop_km               21329 non-null  float64
 29  fitness_km                   21329 non-null  float64
 30  stadium_km                   21329 non-null  float64
 31  basketball_km                21329 non-null  float64
 32  detention_facility_km        21329 non-null  float64
 33  additional_education_km      21329 non-null  float64
 34  big_church_km                21329 non-null  float64
 35  mosque_km                    21329 non-null  float64
 36  theater_km                   21329 non-null  float64
 37  exhibition_km                21329 non-null  float64
 38  catering_km                  21329 non-null  float64
 39  green_part_1000              21329 non-null  float64
 40  cafe_sum_1000_min_price_avg  21329 non-null  float64
 41  cafe_count_1000_price_high   21329 non-null  int64  
```

```
 42  cafe_sum_1500_min_price_avg  21329 non-null  float64
 43  green_part_2000              21329 non-null  float64
 44  cafe_sum_2000_min_price_avg  21329 non-null  float64
 45  mosque_count_3000            21329 non-null  int64
 46  prom_part_5000               21329 non-null  float64
 47  cafe_sum_5000_min_price_avg  21329 non-null  float64
 48  mosque_count_5000            21329 non-null  int64
 49  year                         21329 non-null  int64
dtypes: float64(41), int64(9)
memory usage: 8.1 MB
```

```
y_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21329 entries, 0 to 21328
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   price_doc  21329 non-null  float64
dtypes: float64(1)
memory usage: 166.8 KB
```

```
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9142 entries, 0 to 9141
Data columns (total 50 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   full_sq                      9142 non-null   int64
 1   floor                        9142 non-null   float64
 2   build_year                   9142 non-null   float64
 3   num_room                     9142 non-null   float64
 4   kitch_sq                     9142 non-null   float64
 5   state                        9142 non-null   float64
 6   product_type                 9142 non-null   int64
 7   raion_popul                  9142 non-null   float64
 8   indust_part                  9142 non-null   float64
 9   sport_objects_raion          9142 non-null   int64
 10  shopping_centers_raion       9142 non-null   int64
 11  radiation_raion              9142 non-null   int64
 12  build_count_block            9142 non-null   float64
 13  build_count_brick            9142 non-null   float64
 14  build_count_monolith         9142 non-null   float64
 15  metro_min_avto               9142 non-null   float64
 16  school_km                    9142 non-null   float64
 17  green_zone_km                9142 non-null   float64
 18  industrial_km                9142 non-null   float64
 19  water_treatment_km           9142 non-null   float64
 20  cemetery_km                  9142 non-null   float64
 21  incineration_km              9142 non-null   float64
 22  ID_railroad_station_avto     9142 non-null   float64
 23  mkad_km                      9142 non-null   float64
 24  ttk_km                       9142 non-null   float64
 25  oil_chemistry_km             9142 non-null   float64
 26  nuclear_reactor_km           9142 non-null   float64
 27  power_transmission_line_km   9142 non-null   float64
 28  market_shop_km               9142 non-null   float64
```

```
 29  fitness_km                  9142 non-null   float64
 30  stadium_km                  9142 non-null   float64
 31  basketball_km               9142 non-null   float64
 32  detention_facility_km       9142 non-null   float64
 33  additional_education_km     9142 non-null   float64
 34  big_church_km               9142 non-null   float64
 35  mosque_km                   9142 non-null   float64
 36  theater_km                  9142 non-null   float64
 37  exhibition_km               9142 non-null   float64
 38  catering_km                 9142 non-null   float64
 39  green_part_1000             9142 non-null   float64
 40  cafe_sum_1000_min_price_avg  9142 non-null   float64
 41  cafe_count_1000_price_high   9142 non-null   int64
 42  cafe_sum_1500_min_price_avg  9142 non-null   float64
 43  green_part_2000             9142 non-null   float64
 44  cafe_sum_2000_min_price_avg  9142 non-null   float64
 45  mosque_count_3000           9142 non-null   int64
 46  prom_part_5000              9142 non-null   float64
 47  cafe_sum_5000_min_price_avg  9142 non-null   float64
 48  mosque_count_5000           9142 non-null   int64
 49  year                        9142 non-null   int64
dtypes: float64(41), int64(9)
memory usage: 3.5 MB
```

```
y_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9142 entries, 0 to 9141
Data columns (total 1 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   price_doc  9142 non-null   float64
dtypes: float64(1)
memory usage: 71.5 KB
```

The dataset contains only float and integer values, and there are no missing values

```
print("train set shape:\n",x_train.shape,y_train.shape,"\n test set shape:\n",x_test.shape,y_test.shape)
```

```
train set shape:
 (21329, 50) (21329, 1)
 test set shape:
 (9142, 50) (9142, 1)
```

The dataset contains 50 attributes(columns)

Multicollinearity

```
x_train.corr()
```

	full_sq	floor	build_year	num_room	kitch_sq	state	product_type	raion_popul	indust_part	sport_objects_raion	shopping_centers_raion	radiation_r
full_sq	1.000000	0.077964	-0.002775	0.294462	0.008868	-0.030968	0.089731	-0.034144	-0.049103	0.026406	0.008693	-0.01
floor	0.077964	1.000000	0.000445	-0.007755	-0.007676	-0.085835	0.207234	-0.064959	-0.016169	-0.038687	0.007963	-0.09
build_year	-0.002775	0.000445	1.000000	-0.008960	0.000860	0.317613	-0.005625	0.004193	0.001068	-0.000430	0.002742	0.00
num_room	0.294462	-0.007755	-0.008960	1.000000	0.018603	0.076824	-0.075171	0.064253	-0.029175	0.079500	0.051578	0.03
kitch_sq	0.008868	-0.007676	0.000860	0.018603	1.000000	0.056075	-0.069952	0.038787	0.001382	0.019197	0.006038	0.01
state	-0.030968	-0.085835	0.317613	0.076824	0.056075	1.000000	-0.494085	0.317565	0.041254	0.138604	0.096883	0.15
product_type	0.089731	0.207234	-0.005625	-0.075171	-0.069952	-0.494085	1.000000	-0.654368	-0.111841	-0.307163	-0.201685	-0.34
raion_popul	-0.034144	-0.064959	0.004193	0.064253	0.038787	0.317565	-0.654368	1.000000	0.174910	0.559707	0.506811	0.43
indust_part	-0.049103	-0.016169	0.001068	-0.029175	0.001382	0.041254	-0.111841	0.174910	1.000000	-0.132848	-0.073828	0.04
sport_objects_raion	0.026406	-0.038687	-0.000430	0.079500	0.019197	0.138604	-0.307163	0.559707	-0.132848	1.000000	0.741323	0.44
shopping_centers_raion	0.008693	0.007963	0.002742	0.051578	0.006038	0.096883	-0.201685	0.506811	-0.073828	0.741323	1.000000	0.25
radiation_raion	-0.010936	-0.099087	0.009444	0.030539	0.019082	0.151534	-0.347468	0.432850	0.045655	0.443067	0.252834	1.00
build_count_block	-0.055363	-0.149495	0.013035	-0.002148	0.015782	0.130466	-0.317506	0.285768	-0.057709	0.158781	-0.089177	0.32
build_count_brick	0.019235	-0.088233	-0.004171	0.051430	0.008193	0.010368	-0.101551	0.193789	-0.112954	0.700321	0.402020	0.35
build_count_monolith	0.041147	0.026684	0.002747	0.044800	0.016175	0.076560	-0.100515	0.239957	-0.151673	0.431974	0.230294	0.01
metro_min_avto	0.026584	-0.074963	-0.002268	-0.036755	-0.003971	-0.115033	0.234719	-0.409870	-0.057858	-0.329359	-0.281033	-0.27
school_km	0.044439	-0.078244	-0.002841	-0.031640	-0.022930	-0.179866	0.316169	-0.461990	-0.181312	-0.259943	-0.242318	-0.21
green_zone_km	-0.010497	0.038699	-0.000563	0.003065	-0.002976	-0.024228	0.068248	-0.061542	0.151747	-0.098599	-0.098021	-0.05
industrial_km	0.021899	-0.044562	0.005844	0.030985	-0.000897	0.016437	-0.027794	0.070535	-0.310440	0.239442	0.261867	0.15
water_treatment_km	0.000200	-0.095454	-0.000883	0.036892	-0.000215	0.088632	-0.230208	0.142109	-0.130395	0.222073	0.181854	0.16
cemetery_km	0.013362	0.041744	0.008488	-0.005191	-0.017048	-0.108525	0.177812	-0.092285	0.057209	-0.064541	-0.036981	-0.00
incineration_km	0.028810	-0.044467	0.000031	0.004252	-0.004787	-0.104330	0.188051	-0.265686	-0.341825	0.072074	-0.003669	-0.08
ID_railroad_station_avto	0.025332	-0.048463	0.001968	-0.009671	-0.006485	-0.059124	0.153056	-0.262930	0.100487	-0.138279	-0.208244	-0.06
mkad_km	0.032192	-0.032637	0.003781	-0.000763	-0.006357	-0.110939	0.198656	-0.364799	-0.056959	0.021789	-0.090143	-0.13
ttk_km	0.014627	0.099228	-0.005570	-0.066707	-0.015803	-0.158883	0.391904	-0.424236	-0.110687	-0.531525	-0.327126	-0.38
oil_chemistry_km	0.052454	0.075770	0.000375	-0.004611	-0.018808	-0.158833	0.357343	-0.450006	-0.135027	-0.357485	-0.238776	-0.39
nuclear_reactor_km	0.013428	0.030481	-0.009743	-0.054310	-0.016141	-0.204609	0.420032	-0.536205	-0.221023	-0.341399	-0.314619	-0.30
power_transmission_line_km	0.029462	0.030751	-0.002018	-0.028640	-0.019807	-0.177975	0.331664	-0.435866	-0.122417	-0.176340	-0.179236	-0.23
market_shop_km	0.022367	0.055947	-0.000355	-0.052613	-0.023117	-0.208056	0.439827	-0.602583	-0.158803	-0.413634	-0.390020	-0.30
fitness_km	0.033478	-0.016718	-0.004257	-0.055422	-0.029445	-0.225155	0.418210	-0.576997	-0.080534	-0.403427	-0.333354	-0.27
stadium_km	0.016675	0.091792	-0.001068	-0.061587	-0.014404	-0.183328	0.434055	-0.536757	-0.061444	-0.539395	-0.298585	-0.49
basketball_km	0.029138	0.104470	-0.003931	-0.065088	-0.017020	-0.209579	0.493164	-0.598377	-0.118574	-0.504009	-0.322311	-0.43
detention_facility_km	0.025963	0.042825	-0.001746	-0.052370	-0.019259	-0.183719	0.424955	-0.568665	-0.118105	-0.530582	-0.430129	-0.33

detention_facility_km	0.025963	0.042825	0.001746	-0.052370	-0.019259	-0.183719	0.424955	-0.568665	-0.118105	-0.530582	-0.439129	-0.3
additional_education_km	0.027475	-0.049021	-0.002439	-0.016003	-0.007531	-0.153291	0.294552	-0.516939	-0.156628	-0.307153	-0.315821	-0.19
big_church_km	0.012947	0.016468	0.000448	-0.072849	-0.019882	-0.168430	0.359383	-0.507382	0.031000	-0.573612	-0.523922	-0.35
mosque_km	-0.002377	0.028036	-0.007540	-0.047650	-0.014157	-0.116682	0.218911	-0.255206	0.001633	-0.340575	-0.273382	-0.23
theater_km	0.008989	0.087750	-0.013589	-0.081037	-0.018922	-0.170153	0.388373	-0.385644	-0.059248	-0.478404	-0.296436	-0.37
exhibition_km	0.018891	0.053273	-0.004098	-0.045548	-0.004542	-0.145220	0.353846	-0.493122	-0.068790	-0.469743	-0.383288	-0.39
catering_km	-0.004602	0.019870	-0.010954	-0.074723	-0.015060	-0.185437	0.375180	-0.425858	0.065884	-0.479494	-0.385721	-0.27
green_part_1000	0.027689	0.010953	-0.004959	-0.029533	-0.010864	-0.097995	0.213574	-0.306212	-0.248911	-0.177320	-0.153719	-0.11
cafe_sum_1000_min_price_avg	0.042187	-0.001646	-0.000730	0.019457	-0.010220	-0.009513	0.015121	-0.059424	-0.099965	0.180742	0.107575	0.05
cafe_count_1000_price_high	0.029911	-0.025089	-0.001236	0.052420	-0.002575	-0.008092	-0.009995	0.050255	-0.131799	0.395537	0.307577	0.11
cafe_sum_1500_min_price_avg	0.038265	-0.022782	0.001933	0.026121	-0.005519	0.005970	-0.001706	-0.074344	-0.102875	0.184697	0.094751	0.05
green_part_2000	0.014952	-0.036202	-0.004613	-0.023053	-0.000359	-0.013822	0.049895	-0.160424	-0.359195	-0.137108	-0.135237	-0.05
cafe_sum_2000_min_price_avg	0.028235	-0.046057	0.002985	0.028597	0.002402	0.010298	-0.026180	-0.018227	-0.108204	0.208106	0.123885	0.05
mosque_count_3000	0.019874	-0.001474	0.012396	0.036220	0.004446	0.010232	-0.008443	0.098133	-0.134634	0.402463	0.416857	0.16
prom_part_5000	-0.045225	-0.080128	0.001340	0.023412	0.018210	0.158389	-0.387764	0.463857	0.399741	0.186511	0.169636	0.28
cafe_sum_5000_min_price_avg	0.041648	0.035803	0.000675	-0.001985	-0.023018	-0.155430	0.257431	-0.280020	-0.199616	0.096921	0.022980	-0.08
mosque_count_5000	0.017499	-0.018655	0.017675	0.057165	0.008616	0.065876	-0.099789	0.106573	-0.079275	0.330246	0.298615	0.19
year	0.021287	-0.015183	-0.003466	-0.036782	0.001714	0.007337	0.066804	-0.055802	-0.025115	-0.006462	0.001082	-0.02

the attributes are not correlated, there is no multicollinearity between the attributes. Multicollinearity affect model performance.

Model Building:
Using neural network we will be able to build a model, but what is the ideal numbe of hidden layers to be used, how to fix dense values.To find the optimal value of these for a data, there is method called Hyperparamter tuning.

Parameters: which can be learned by model

Here, the parameter are bias, weight or coeffiecients of attributes

Hyperparameters: to be learned by the model builder (us)

here, they are the number of hidden layers, learning rate, dense

Hyperparameter tunning

tunning will tell us "How deep should the model be:"

```
#!pip install keras_tuner
```

```
from tensorflow import keras
from tensorflow.keras import layers
from keras_tuner.tuners import BayesianOptimization
```

Hyperparameter search space

```
def build_model(hp):
    model = keras.Sequential()
    for i in range(hp.Int('num_layers', 2, 20)):
        model.add(layers.Dense(units=hp.Int('units_' + str(i),
                                            min_value=32,
                                            max_value=512,
                                            step=32),
                               activation='relu'))
    model.add(layers.Dense(1, activation='linear'))
    model.compile(
        optimizer=keras.optimizers.Adam(
            hp.Choice('learning_rate', [1e-2, 1e-3, 1e-4])),
        loss='mean_absolute_error',
        metrics=['mean_absolute_error'])
    return model
```

Here, we are creating a hyperparameter space, by fixing the range for the hyperparameter.

- range of hidden layer:2- 20
- range of dense value: 32 - 512 (32,64,96,.....512)

In hyperparameter tuning there are different methods like gridsearch, randomsearch, Bayesian optimization. Among them Bayesian is best.

why bayesian?

- less memory
- less run time
- based on probability

```
tuner = BayesianOptimization(
    build_model,
    objective='val_mean_absolute_error',
    max_trials=5,
    executions_per_trial=3,
    directory='project',
    project_name='house price')
```

Here, tuner is running over the fixed search space.

With the help of Bayesian optimizer, the ideal number of layers, learning rates are found.

```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 4
num_layers (Int)
{'default': None, 'conditions': [], 'min_value': 2, 'max_value': 20, 'step': 1, 'sampling': None}
units_0 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 32, 'max_value': 512, 'step': 32, 'sampling': None}
learning_rate (Choice)
{'default': 0.01, 'conditions': [], 'values': [0.01, 0.001, 0.0001], 'ordered': True}
```

```
tuner.search(x_train, y_train,
             epochs=5,
             validation_data=(x_test, y_test))
```

```
Trial 5 Complete [00h 02m 19s]
val_mean_absolute_error: 0.3334464331467946

Best val_mean_absolute_error So Far: 0.307867964108785
Total elapsed time: 00h 11m 52s
INFO:tensorflow:Oracle triggered exit
```

The considered dataset is assigned to the tuner serach.

tunner will take this dataset and will run through the search speace, and then will provide different ideal hyperparameter value combination.

```
tuner.results_summary()
```

```
units_7: 32
units_8: 32
units_9: 32
units_10: 32
units_11: 32
units_12: 32
units_13: 32
units_14: 32
units_15: 192
units_16: 32
units_17: 160
units_18: 64
units_19: 384
Score: 0.3334464331467946
Trial summary
Hyperparameters:
num_layers: 20
units_0: 32
units_1: 512
learning_rate: 0.001
units_2: 512
units_3: 512
units_4: 192
units_5: 160
units_6: 192
units_7: 128
units_8: 160
```

```
units_9: 160
units_10: 160
units_11: 192
units_12: 128
units_13: 128
units_14: 128
units_15: 32
units_16: 32
units_17: 32
units_18: 32
units_19: 32
Score: 0.36851834257443744
Trial summary
Hyperparameters:
num_layers: 5
units_0: 352
units_1: 480
learning_rate: 0.0001
units_2: 384
units_3: 64
units_4: 32
Score: 0.38449641068776447
Trial summary
Hyperparameters:
num_layers: 4
units_0: 384
units_1: 192
learning_rate: 0.0001
units_2: 32
units_3: 32
Score: 0.41446494062741596
```

we have got 5 different combination , since we have fixed the epoch as 5.

Deep Neural Network Regression model

```
model = Sequential()
model.add(Dense(51, input_dim=50, kernel_initializer='normal', activation='relu'))
model.add(Dense(420, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
```

```
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 51)                2601

 dense_21 (Dense)            (None, 420)               21840

 dense_22 (Dense)            (None, 32)                13472

 dense_23 (Dense)            (None, 32)                1056

 dense_24 (Dense)            (None, 32)                1056

 dense_25 (Dense)            (None, 32)                1056

 dense_26 (Dense)            (None, 32)                1056

 dense_27 (Dense)            (None, 32)                1056

 dense_28 (Dense)            (None, 32)                1056

 dense_29 (Dense)            (None, 32)                1056

 dense_30 (Dense)            (None, 32)                1056

 dense_31 (Dense)            (None, 32)                1056

 dense_32 (Dense)            (None, 32)                1056

 dense_33 (Dense)            (None, 32)                1056

 dense_34 (Dense)            (None, 32)                1056

 dense_35 (Dense)            (None, 32)                1056

 dense_36 (Dense)            (None, 32)                1056

 dense_37 (Dense)            (None, 32)                1056

 dense_38 (Dense)            (None, 32)                1056

 dense_39 (Dense)            (None, 32)                1056

 dense_40 (Dense)            (None, 32)                1056

 dense_41 (Dense)            (None, 1)                 33

=================================================================
Total params: 56,954
Trainable params: 56,954
```

```
Non-trainable params: 0
```

Model is built over the hyperparamter values obtained from tuning.

Model performance

```
model.compile(loss='mse', optimizer='adam', metrics=['mse','mae','mape',tf.keras.metrics.RootMeanSquaredError()
])
history=model.fit(x_train, y_train, epochs=30, batch_size=150, verbose=1,validation_split=0.2)
predictions = model.predict(x_test)
```

```
Epoch 3/30
114/114 [==============================] - 1s 8ms/step - loss: 9.2269 - mse: 9.2269 - mae: 1.3842 - mape: 8.8958 - root_mean_squared_error: 3.0376 - val_loss: 0.4330 - val_mse: 0.4330 -
Epoch 4/30
114/114 [==============================] - 1s 7ms/step - loss: 17.3447 - mse: 17.3447 - mae: 0.4089 - mape: 2.6597 - root_mean_squared_error: 4.1647 - val_loss: 13.8244 - val_mse: 13.82
Epoch 5/30
114/114 [==============================] - 1s 7ms/step - loss: 73.2929 - mse: 73.2929 - mae: 3.1015 - mape: 19.8793 - root_mean_squared_error: 8.5611 - val_loss: 0.5126 - val_mse: 0.512
Epoch 6/30
114/114 [==============================] - 1s 8ms/step - loss: 2455.2422 - mse: 2455.2422 - mae: 2.0750 - mape: 13.2615 - root_mean_squared_error: 49.5504 - val_loss: 0.9414 - val_mse:
Epoch 7/30
114/114 [==============================] - 1s 7ms/step - loss: 0.4557 - mse: 0.4557 - mae: 0.4866 - mape: 3.1605 - root_mean_squared_error: 0.6750 - val_loss: 0.3152 - val_mse: 0.3152 -
Epoch 8/30
114/114 [==============================] - 1s 8ms/step - loss: 0.3266 - mse: 0.3266 - mae: 0.3976 - mape: 2.5946 - root_mean_squared_error: 0.5715 - val_loss: 0.2864 - val_mse: 0.2864 -
Epoch 9/30
114/114 [==============================] - 1s 7ms/step - loss: 0.3113 - mse: 0.3113 - mae: 0.3771 - mape: 2.4611 - root_mean_squared_error: 0.5579 - val_loss: 0.2813 - val_mse: 0.2813 -
Epoch 10/30
114/114 [==============================] - 1s 7ms/step - loss: 0.3937 - mse: 0.3937 - mae: 0.3755 - mape: 2.4501 - root_mean_squared_error: 0.6275 - val_loss: 0.2783 - val_mse: 0.2783 -
Epoch 11/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2989 - mse: 0.2989 - mae: 0.3710 - mape: 2.4216 - root_mean_squared_error: 0.5467 - val_loss: 0.2765 - val_mse: 0.2765 -
Epoch 12/30
114/114 [==============================] - 1s 7ms/step - loss: 0.3931 - mse: 0.3931 - mae: 0.3765 - mape: 2.4576 - root_mean_squared_error: 0.6269 - val_loss: 0.2708 - val_mse: 0.2708 -
Epoch 13/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2875 - mse: 0.2875 - mae: 0.3634 - mape: 2.3748 - root_mean_squared_error: 0.5362 - val_loss: 0.2690 - val_mse: 0.2690 -
Epoch 14/30
114/114 [==============================] - 1s 7ms/step - loss: 0.3343 - mse: 0.3343 - mae: 0.3798 - mape: 2.4788 - root_mean_squared_error: 0.5782 - val_loss: 0.2617 - val_mse: 0.2617 -
Epoch 15/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2863 - mse: 0.2863 - mae: 0.3611 - mape: 2.3598 - root_mean_squared_error: 0.5350 - val_loss: 0.2648 - val_mse: 0.2648 -
Epoch 16/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2722 - mse: 0.2722 - mae: 0.3502 - mape: 2.2916 - root_mean_squared_error: 0.5218 - val_loss: 0.2609 - val_mse: 0.2609 -
Epoch 17/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2731 - mse: 0.2731 - mae: 0.3523 - mape: 2.3048 - root_mean_squared_error: 0.5226 - val_loss: 0.2561 - val_mse: 0.2561 -
Epoch 18/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2677 - mse: 0.2677 - mae: 0.3444 - mape: 2.2550 - root_mean_squared_error: 0.5174 - val_loss: 0.2557 - val_mse: 0.2557 -
Epoch 19/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2703 - mse: 0.2703 - mae: 0.3489 - mape: 2.2842 - root_mean_squared_error: 0.5199 - val_loss: 0.2558 - val_mse: 0.2558 -
Epoch 20/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2683 - mse: 0.2683 - mae: 0.3461 - mape: 2.2667 - root_mean_squared_error: 0.5179 - val_loss: 0.2908 - val_mse: 0.2908 -
Epoch 21/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2671 - mse: 0.2671 - mae: 0.3457 - mape: 2.2647 - root_mean_squared_error: 0.5169 - val_loss: 0.2532 - val_mse: 0.2532 -
Epoch 22/30
114/114 [==============================] - 1s 7ms/step - loss: 0.2617 - mse: 0.2617 - mae: 0.3382 - mape: 2.2165 - root_mean_squared_error: 0.5116 - val_loss: 0.2558 - val_mse: 0.2558 -
Epoch 23/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2629 - mse: 0.2629 - mae: 0.3398 - mape: 2.2269 - root_mean_squared_error: 0.5128 - val_loss: 0.2474 - val_mse: 0.2474 -

Epoch 24/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2611 - mse: 0.2611 - mae: 0.3380 - mape: 2.2164 - root_mean_squared_error: 0.5110 - val_loss: 0.2545 - val_mse: 0.2545 -
Epoch 25/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2597 - mse: 0.2597 - mae: 0.3367 - mape: 2.2080 - root_mean_squared_error: 0.5096 - val_loss: 0.2518 - val_mse: 0.2518 -
Epoch 26/30
```

```
114/114 [==============================] - 1s 8ms/step - loss: 0.2562 - mse: 0.2562 - mae: 0.3327 - mape: 2.1830 - root_mean_squared_error: 0.5062 - val_loss: 0.2460 - val_mse: 0.2460 -
Epoch 27/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2634 - mse: 0.2634 - mae: 0.3418 - mape: 2.2413 - root_mean_squared_error: 0.5132 - val_loss: 0.2434 - val_mse: 0.2434 -
Epoch 28/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2587 - mse: 0.2587 - mae: 0.3359 - mape: 2.2039 - root_mean_squared_error: 0.5086 - val_loss: 0.2949 - val_mse: 0.2949 -
Epoch 29/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2640 - mse: 0.2640 - mae: 0.3439 - mape: 2.2552 - root_mean_squared_error: 0.5138 - val_loss: 0.2521 - val_mse: 0.2521 -
Epoch 30/30
114/114 [==============================] - 1s 8ms/step - loss: 0.2589 - mse: 0.2589 - mae: 0.3361 - mape: 2.2052 - root_mean_squared_error: 0.5088 - val_loss: 0.2483 - val_mse: 0.2483 -
```

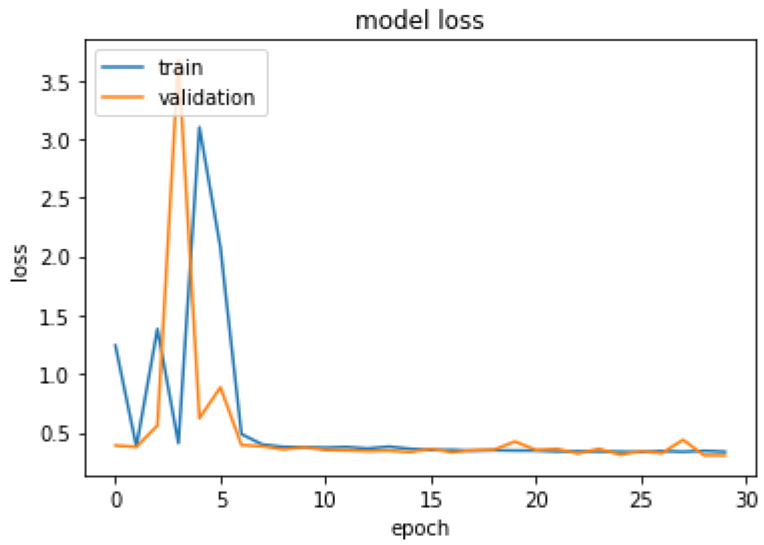
The loss getting less over the iterations

less the loss, high the accuracy of the model

Plots to visualize the model performances

```
print(history.history.keys())
# "Loss"
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

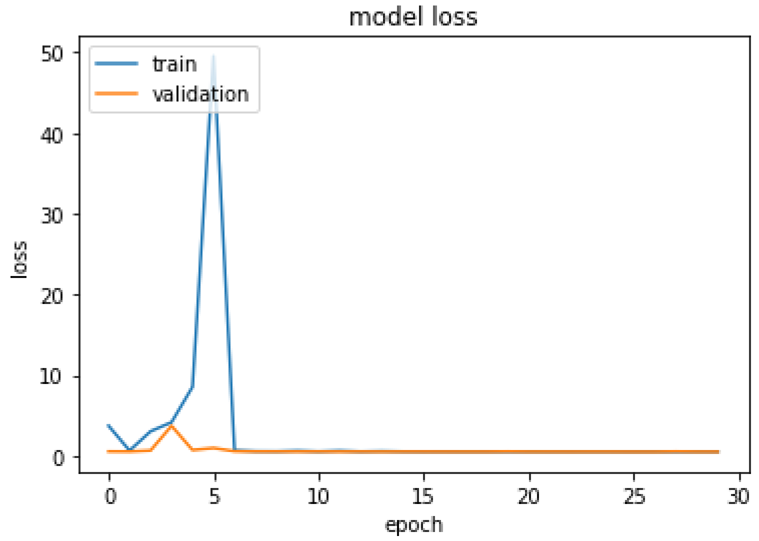
```
dict_keys(['loss', 'mse', 'mae', 'mape', 'root_mean_squared_error', 'val_loss', 'val_mse', 'val_mae', 'val_mape', 'val_root_mean_squared_error'])
```




```
print(history.history.keys())
# "Loss"

plt.plot(history.history['root_mean_squared_error'])
plt.plot(history.history['val_root_mean_squared_error'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'mse', 'mae', 'mape', 'root_mean_squared_error', 'val_loss', 'val_mse', 'val_mae', 'val_mape', 'val_root_mean_squared_error'])
```



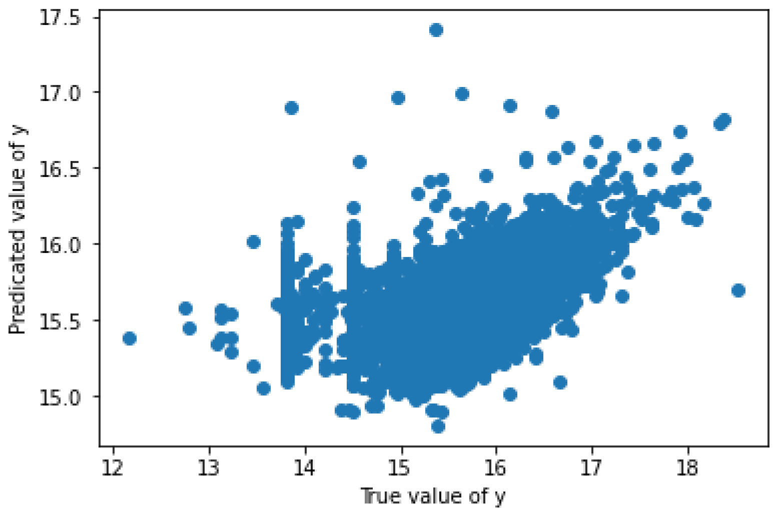
these plots tell us that the train and test data loss keep reducing, then after some iterations they become near to 0.

actual vs predicted

```
y_pred=model.predict(x_test)
y_pred.shape
```

```
(9142, 1)
```

```
fig, ax = plt.subplots()
ax.scatter(y_test,y_pred)
ax.set_xlabel('True value of y')
ax.set_ylabel('Predicated value of y')
plt.show()
```



The points are more uniform, hence we can say the model do well

Model explainability

using LIME - Local Interpretable Model-Agnostic Explanations, we get the features values for which prediction are done correctly, with help of this we can understand the model better.

LIME - VISULAIZE THE MODEL FEATURES

SHAP VS LIME VS Integrated Gradients

- SHAP is most ideal for accuary check,
- since we are not using any backprogation I have not used Integrated gradients

why Lime?

LIME in most ideal the linear function, Since the task is based on regression (multiple linear regression), I preferred LIME

```
#!pip install lime
```

```
x_train = x_train.values
print(x_train.shape)
y_train = y_train.values
x_test = x_test.values
y_test = y_test.values
```

```
(21329, 50)
```

```
from lime import lime_tabular

explainer = lime_tabular.LimeTabularExplainer(x_train, mode="regression" ) #, feature_names= feat)
explainer
```

```
<lime.lime_tabular.LimeTabularExplainer at 0x7f1a826b5e90>
```

```
import random
import warnings

warnings.filterwarnings("ignore")
```

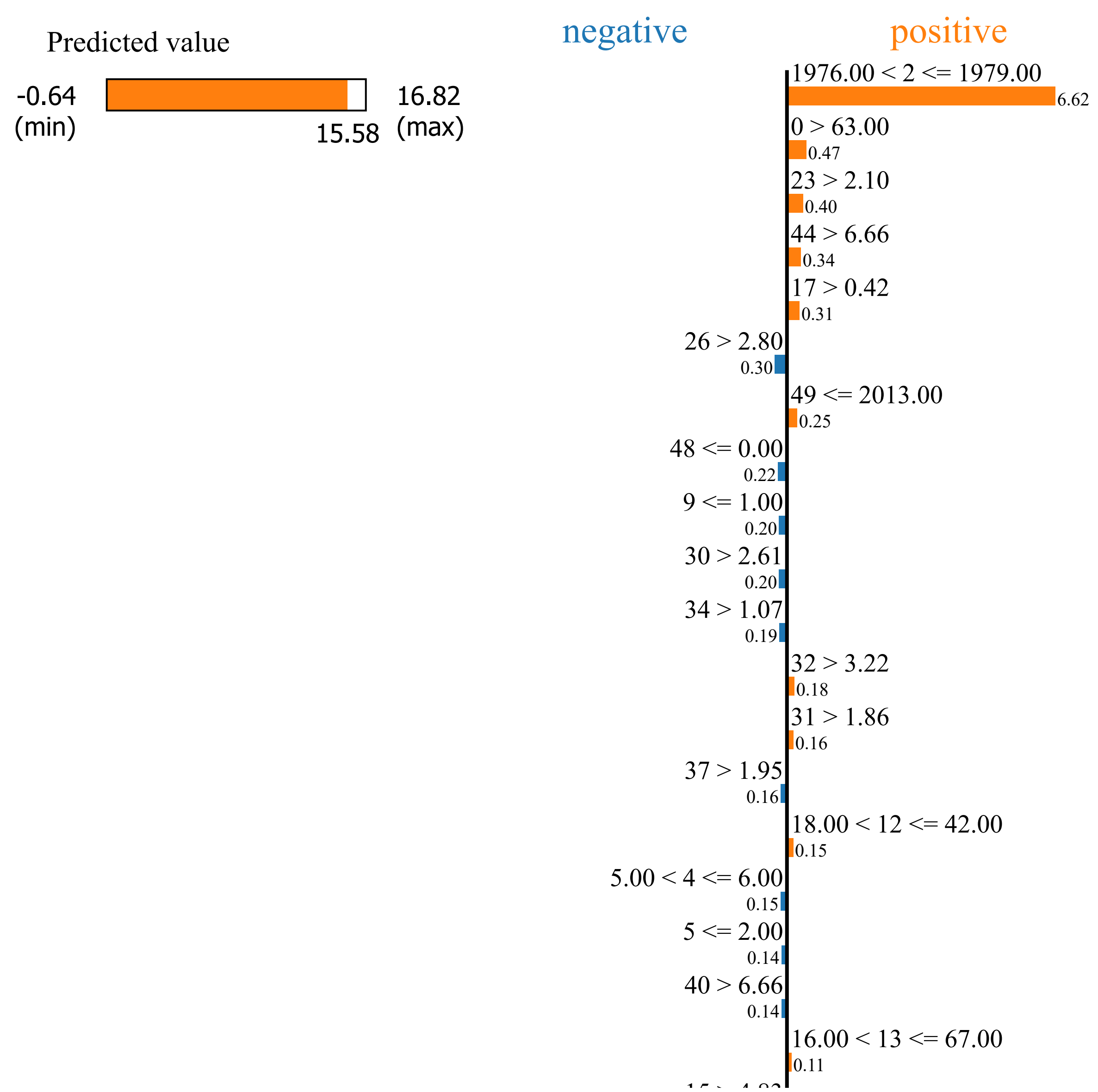
```
idx = random.randint(1, len(x_test)+1)
#idx= random.randint(1, 50)

print("Prediction : ", model.predict(x_test[idx].reshape(1,-1)))
print("Actual :     ", y_test[idx])

explanation = explainer.explain_instance(x_test[idx], model.predict, num_features=50)   #len(boston.feature_names))
explanation
```

```
Prediction :  [[15.584011]]
Actual :      [15.39811629]
```

```
<lime.explanation.Explanation at 0x7f1a8586d110>
```

Here, random datapoint are selected and used for prediction of the target values,where the mode which we built was used.

```
explanation.show_in_notebook()
```



Feature	Value
2	1979.00
0	65.00
23	2.46
44	6.91
17	0.99
26	3.15
49	2012.00
48	0.00
9	0.00
30	2.98
34	2.26
32	3.61

this notebook, tell the range of the attributes whos prediction are done correctly.

here, is the list of it.

```
explanation.as_list()
```

```
[('1976.00 < 2 <= 1979.00', 6.621825012495475),
 ('0 > 63.00', 0.4743536444111954),
 ('23 > 2.10', 0.3950479872420051),
 ('44 > 6.66', 0.3434610997011183),
 ('17 > 0.42', 0.31268020711590844),
 ('26 > 2.80', -0.3006093633897373),
 ('49 <= 2013.00', 0.2509200490928154),
 ('48 <= 0.00', -0.22038975287042353),
 ('9 <= 1.00', -0.19805317993420107),
 ('30 > 2.61', -0.1969112965391686),
 ('34 > 1.07', -0.19124693724708358),
 ('32 > 3.22', 0.18286741930706457),
 ('31 > 1.86', 0.1622347959375496),
 ('37 > 1.95', -0.15545861779047712),
 ('18.00 < 12 <= 42.00', 0.15444838074462905),
 ('5.00 < 4 <= 6.00', -0.1540324024111512),
 ('5 <= 2.00', -0.13669365766403616),
```

```
('40 > 6.66', -0.13551974574869954),
('16.00 < 13 <= 67.00', 0.1093407912321061),
('15 > 4.83', -0.10430203735163246),
('20 <= 1.34', -0.10331252074218564),
('36 > 2.60', -0.09641389235003285),
('18 > 1.04', -0.09601008393760525),
('1.66 < 19 <= 2.34', -0.09494389940592152),
('21 > 2.60', 0.09457271623334977),
('39 <= 6.26', 0.09211457528242557),
('45 <= 0.00', 0.08032649259942205),
('2.87 < 43 <= 3.34', -0.06414121991263118),
('41 <= 0.00', 0.0618405412789239),
('8 <= 0.02', 0.060748701082287714),
('46 <= 1.80', 0.05702856063648218),
('42 > 6.67', -0.05623809087922166),
('11 <= 0.00', -0.0551168195508417),
('25 > 3.15', 0.050498244385575895),
('24 > 2.75', 0.04987335686544497),
('22 > 4.29', 0.04831078408260206),
('38 <= -1.57', -0.04791338610356091),
('3.00 < 14 <= 6.00', 0.04788064444836897),
('27 > 1.59', -0.04408936545298406),
('7 <= 9.99', -0.03380000056931959),
('3 <= 2.00', -0.031097816398652573),
('29 > 1.33', -0.026677976866402345),
('28 > 1.71', 0.02647534559405266),
('47 > 6.70', 0.015573454173805217),
('10 <= 1.00', 0.013728433556126803),
('33 > 1.56', 0.008841745646757548),
('16 > 0.89', 0.006982118257268862),
('0.00 < 6 <= 1.00', -0.002473884664649642),
('35 > 2.31', 0.002230365745808581З),
('6.50 < 1 <= 11.00', 0.0006831372496166846)]
```

[vedio explanation link:] (https://screenrec.com/share/jDgrsR2C3l)

Thank you

0s completed at 2:20 PM