AIM : To apply different models on dataset to get value of mean absolute percentage error less than 5% .

DATASET USED : Automobiles

Data Source (Link): https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data

Objective: To get mean absolute percentage error less than 5% by using different machine and deep learning models.

Description: Collected data from UCI Machine Learning repositories. It is a regression problem containing 205 instances and 26 attributes with both numerical and categorical values. This project contains two major parts. First is, cleaning and preprocessing of data. Use of inbuilt functions (like describe and .isnull) to check if any null value is present in data. Conversion of datatype of attributes using .info() .As we have given many attributes; we need to check how much they are corelated to each other for better understanding of data, which can be done using heatmap. We might have lots of missing values, no need to drop all of them; we can also replace null values using aggregates like mean, median, mode, count etc. Standardising also plays main role in preprocessing of data. Use of scatter plot, box and whiskers plot has been done to analyse data more precisely. To know more about data, we used groupby function to play around with data to get useful information. Second is, use of various machine learning and deep learning models to train data and to choose the one giving more accuracy.

Step_1 : Importing libraries and Dataset

```
import pandas as pd
import numpy as np
```

```
# importing data
#Note :Because the data does not include headers, we can add an argument headers = None inside the read_csv() method, so that pandas will not automatically set the first row as a he

path = r"https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data"
am_df = pd.read_csv(path, header=None)
```

```
am_df
```

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	9.5	114	5400	23	28	16845
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...	141	mpfi	3.78	3.15	8.7	160	5300	19	25	19045
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...	173	mpfi	3.58	2.87	8.8	134	5500	18	23	21485
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...	145	idi	3.01	3.40	23.0	106	4800	26	27	22470

```
## NOTE : Point to Remember : pandas automatically set the header by an integer. To better describe our data we can introduce a header(we create a list "headers" that include all co

headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style","drive-wheels","engine-location","wheel-base", "length","width","height","cur
print("headers\n", headers)
```

```
headers
 ['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length', 'width', 'height
```

```
am_df.columns = headers
```

```
am_df
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	s
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	
			alfa-												
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	

Step_2 : Preprocessing

```
#checking type of data
type(am_df)
```

```
pandas.core.frame.DataFrame
```

```
#checking the datatype for ? using index location
type(am_df.iloc[0,1])
```

```
str
```

```
## NOTE : Relacing all '?' with NaN type
am_df=am_df.replace(['?'],np.nan)
```

```
#The info() method prints information about the DataFrame.
#The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-null values).

am_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       203 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
```

```
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               201 non-null    object
 19  stroke             201 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    object
 22  peak-rpm           203 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              201 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

```
#coverting the object datatype to numeric datatype using the information we get from above code

am_df[['price', 'peak-rpm', 'horsepower', 'stroke', 'bore', 'normalized-losses']] = am_df[['price', 'peak-rpm', 'horsepower', 'stroke', 'bore', 'normalized-losses']].apply(pd.to_num
```

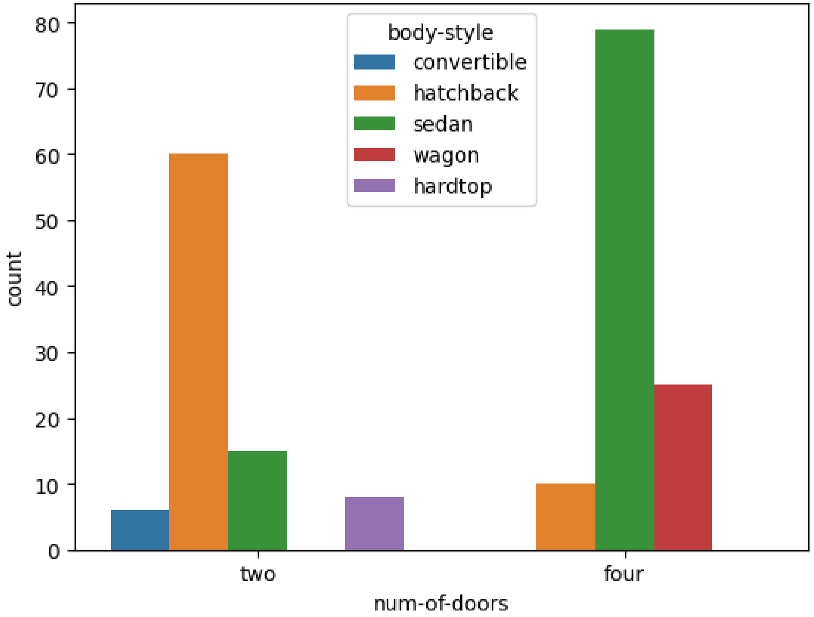
```
am_df.info() #performing this code again to check if dtype for variables has changed or not
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  164 non-null    float64
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       203 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               201 non-null    float64
 19  stroke             201 non-null    float64
 20  compression-ratio  205 non-null    float64
 21  horsepower         203 non-null    float64
 22  peak-rpm           203 non-null    float64
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              201 non-null    float64
```

```
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB
```

```
import seaborn as sns
sns.countplot(data=am_df, x='num-of-doors', hue='body-style')
```

```
<Axes: xlabel='num-of-doors', ylabel='count'>
```




```
am_df['num-of-doors'].mode()
```

```
0    four
Name: num-of-doors, dtype: object
```

```
#Filling misssing values(im columns having num values)

for j in am_df.columns:
  if am_df[j].dtype != 'object':

    am_df[j] = am_df[j].fillna(am_df[j].mean())
```

```
am_df['num-of-doors'].value_counts()
```

```
four    114
two      89
Name: num-of-doors, dtype: int64
```

```
am_df.loc[:,'num-of-doors'].fillna('four', inplace=True)#(filling categorical variable:'num-of-doors' with value'four' in place of null values)
```

```
am_df.isnull().any()#Checking if there is any null value left
```

```
symboling            False
normalized-losses    False
make                 False
fuel-type            False
aspiration           False
num-of-doors         False
body-style           False
drive-wheels         False
engine-location      False
wheel-base           False
length               False
width                False
height               False
curb-weight          False
engine-type          False
num-of-cylinders     False
engine-size          False
fuel-system          False
bore                 False
stroke               False
compression-ratio    False
horsepower           False
peak-rpm             False
city-mpg             False
highway-mpg          False
price                False
dtype: bool
```

Step_2.1: Separating X(input variables) and Y(target/output variable)(i.e 'price')

```
X = am_df.iloc[:, 0:-1]
Y = am_df.iloc[:, -1]
```

```
X_list_num = [j for j in X.columns if X[j].dtype != 'object']     ##List of numeric columns headers  # will be needing this list later on in this project
X_list_cat = [j for j in X.columns if X[j].dtype == 'object']     ##List of category column headers
```

Step_3 : Standardising X

```
#Note : y can not b standardised because this is Linear Regression problem
```

```
# Standardising numerical variables using MinMaxScaler which scales all values in between a range of 0-1
```

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()

#fit_transform() - calculates mean and standard dev of each column - (x-mean)/std
X_numerical_scaled = scaler.fit_transform(X[X_list_num].to_numpy())     ## .to_numpy converts dataframe to numpy array
```

```
X_numerical_scaled
```

```
array([[1.        , 0.29842932, 0.05830904, ..., 0.34693878, 0.22222222,
        0.28947368],
       [1.        , 0.29842932, 0.05830904, ..., 0.34693878, 0.22222222,
        0.28947368],
       [0.6       , 0.29842932, 0.2303207 , ..., 0.34693878, 0.16666667,
        0.26315789],
       ...,
       [0.2       , 0.15706806, 0.65597668, ..., 0.55102041, 0.13888889,
        0.18421053],
       [0.2       , 0.15706806, 0.65597668, ..., 0.26530612, 0.36111111,
        0.28947368],
       [0.2       , 0.15706806, 0.65597668, ..., 0.51020408, 0.16666667,
        0.23684211]])
```

```
X_numerical_scaled.shape, X[X_list_num].shape
```

```
((205, 15), (205, 15))
```

```
#converting numpy array back to dataframe

X_numerical_scaled = pd.DataFrame(X_numerical_scaled, columns = X_list_num)
X_numerical_scaled.head(2)
```

	symboling	normalized-losses	wheel-base	length	width	height	curb-weight	engine-size	bore	stroke	compression-ratio	horsepower
0	1.0	0.298429	0.058309	0.413433	0.316667	0.083333	0.411171	0.260377	0.664286	0.290476	0.125	0.2625

```
#creating dummies for one hot encoding categorical variables

X_categorical_ohe = pd.get_dummies(data = X[X_list_cat], columns = X_list_cat)
```

```
X['fuel-system'].unique()   #checking no. of classes present in column specified in code
```

```
array(['mpfi', '2bbl', 'mfi', '1bbl', 'spfi', '4bbl', 'idi', 'spdi'],
      dtype=object)
```

```
X_categorical_ohe.head()
```

	make_alfa-romero	make_audi	make_bmw	make_chevrolet	make_dodge	make_honda	make_isuzu	make_jaguar	make_mazda	make_mercedes-benz
0	1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0

5 rows × 60 columns

```
X = pd.concat([X_categorical_ohe, X_numerical_scaled,], axis = 1)
```

```
X          #input variables are now finally preprocessed
```

	make_alfa-romero	make_audi	make_bmw	make_chevrolet	make_dodge	make_honda	make_isuzu	make_jaguar	make_mazda	make_mercedes ben
0	1	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	
3	0	1	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	.
200	0	0	0	0	0	0	0	0	0	
201	0	0	0	0	0	0	0	0	0	
202	0	0	0	0	0	0	0	0	0	
203	0	0	0	0	0	0	0	0	0	
204	0	0	0	0	0	0	0	0	0	

205 rows × 75 columns

```
type(X)
```

```
pandas.core.frame.DataFrame
```

```
type(Y)
```

```
pandas.core.series.Series
```

Step_3.1 : converting to numpy array

```
import numpy as np

X = np.array(X)
Y = np.array(Y)
```

```
X.shape
```

```
(205, 75)
```

```
Y.shape
```

```
(205,)
```

Step_4 : Splitting into test and train data

```
from sklearn.model_selection import train_test_split

Xtrain, Xtest, ytrain, ytest = train_test_split(X, Y , test_size=0.2, random_state=42, shuffle=True)
```

```
Xtrain.shape
```

```
(164, 75)
```

```
Xtest.shape
```

```
(41, 75)
```

Step_5 : Building ANN model

```
                                                  #Step_5.1 : Importing models

from keras.models import Sequential
from keras.layers import Dense

am_df_ANN = Sequential()
```

#Step_5.2 : Adding layers

```
'''
Note: sigmoid activation is for  - binary [2 class classification]
      softmax activation is for  - multi  [2+ class classification]
'''

'''
Note: output layer: number of units = number of class/category present in target variable (for  multi  [2+ class classification])
                  : number of units = 1               (for binary [2 class classification])
'''
#hidden layer
am_df_ANN.add(Dense(units=750, activation = 'relu'))
#hidden layer
am_df_ANN.add(Dense(units=300, activation = 'relu'))
#output layer
am_df_ANN.add(Dense(units=1,  activation = 'relu'))
```

#Step_5.3 : Exponential decay, Compile, best mmodel, early stopping

```
from tensorflow.keras.optimizers.schedules import ExponentialDecay
from keras.optimizers import Adam
'''
ExponentialDecay - with iterations reduce the learning rate
'''
initial_learning_rate = 0.001
lr=ExponentialDecay(initial_learning_rate,
    decay_steps=100000,
    decay_rate=0.96,
    staircase=True)

am_df_ANN.compile(loss='mean_squared_error',
                  metrics='mean_absolute_percentage_error',
                  optimizer='adam'
                  )


'''
Note: categorical_crossentropy - loss for multiclass classification
      binary_crossentropy - loss for binary classification
      mean_squared_error - loss for linear regression problems
'''


from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

'''
EarlyStopping- is used when metric doesn't improve for certain epochs we stop training

EarlyStopping(
  monitor="val_accuracy", - metric to monitor
  patience=10, - number of epochs we monitor metric
```

```
  verbose=1, - message will be printed)
'''

es = EarlyStopping(monitor="val_accuracy", patience=10, verbose=1)
mc = ModelCheckpoint(filepath='bestmodel.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

Step_6 : Fit

```
history = am_df_ANN.fit(Xtrain, ytrain, epochs=150)
```

```
Epoch 116/150
6/6 [==============================] - 0s 12ms/step - loss: 6488474.0000 - mean_absolute_percentage_error: 14.5695
Epoch 117/150
6/6 [==============================] - 0s 12ms/step - loss: 6425537.0000 - mean_absolute_percentage_error: 14.5188
Epoch 118/150
6/6 [==============================] - 0s 12ms/step - loss: 6364487.5000 - mean_absolute_percentage_error: 14.4803
Epoch 119/150
6/6 [==============================] - 0s 14ms/step - loss: 6302905.0000 - mean_absolute_percentage_error: 14.4599
Epoch 120/150
6/6 [==============================] - 0s 15ms/step - loss: 6234747.5000 - mean_absolute_percentage_error: 14.3643
Epoch 121/150
6/6 [==============================] - 0s 12ms/step - loss: 6195698.5000 - mean_absolute_percentage_error: 14.2191
Epoch 122/150
6/6 [==============================] - 0s 20ms/step - loss: 6186924.5000 - mean_absolute_percentage_error: 14.1227
Epoch 123/150
6/6 [==============================] - 0s 15ms/step - loss: 6061656.0000 - mean_absolute_percentage_error: 14.0217
Epoch 124/150
```

#Step_6.1 : ANN architecture

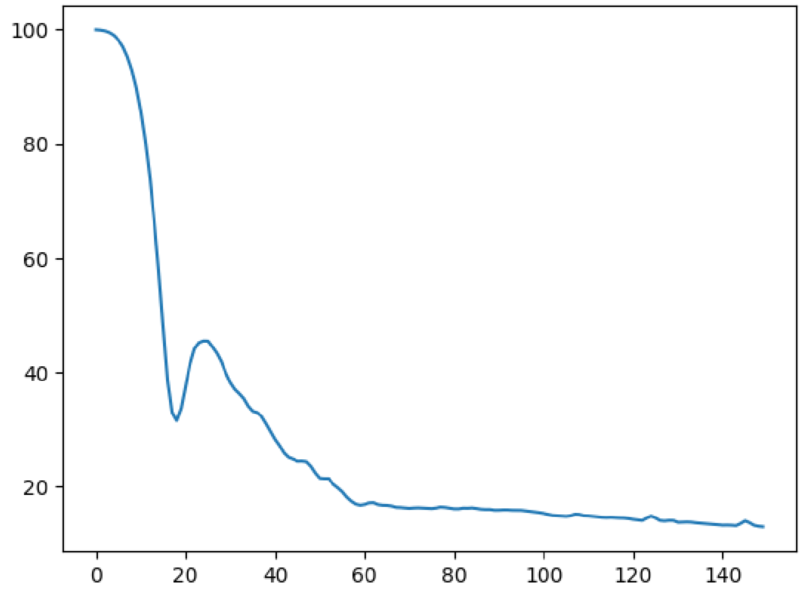
```
am_df_ANN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 750)               57000

 dense_1 (Dense)             (None, 300)               225300

 dense_2 (Dense)             (None, 1)                 301

=================================================================
Total params: 282,601
Trainable params: 282,601
Non-trainable params: 0
_________________________________________________________________
```

#Step_6.2 : Plotting loss and mean absolute percentage error

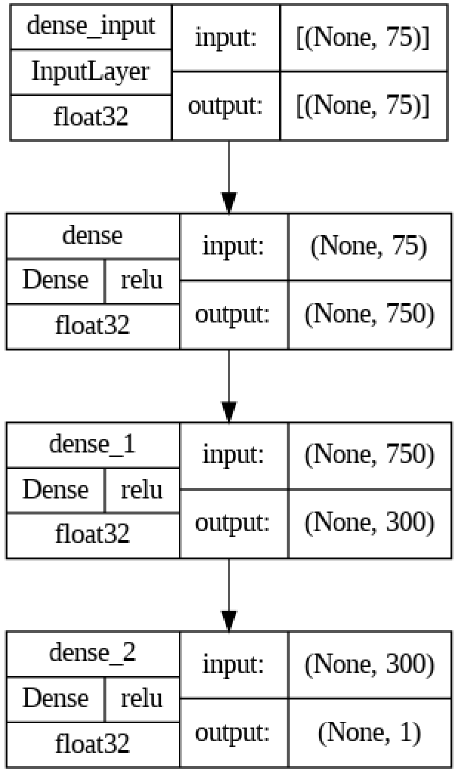
```
import matplotlib.pyplot as plt


plt.plot(history.history['loss'])
plt.show()
```


```
plt.plot(history.history['mean_absolute_percentage_error'])
plt.show()
```



Step_7 : Model plotting

```
#conda install -c anaconda pydot
#conda install -c anaconda graphviz

from tensorflow.keras.utils import plot_model
plot_model(am_df_ANN, show_shapes=True, show_dtype=True, show_layer_activations=True, show_layer_names=True)
```



Step_8 : Prediction of ypred using X

```
ypred = am_df_ANN.predict(Xtest)
```

```
2/2 [==============================] - 0s 11ms/step
```

```
ypred
```

```
ypred.shape
```

```
(41, 1)
```

```
am_df_ANN.evaluate(Xtest, ytest)
```

```
2/2 [==============================] - 0s 8ms/step - loss: 12898391.0000 - mean_absolute_percentage_error: 20.8886
[12898391.0, 20.888553619384766]
```

We can see that ANN model is not precisely built and giving us 20% error but our aim is to get error less than 5%. Therefore we will now be applying Decision Trees model to get better accuracy/less error.

Decision Trees

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
```

Note : Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters. This makes the processing time-consuming and expensive based on the number of hyperparameters involved.

```
#visit this link to know more about hyperparameters of decision tree : https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html#sklearn.tree.Decisio

#hyper-parameters of decision trees
'''
criterion = ['squared_error', 'friedman_mse', 'absolute_error', 'poisson'] #calculating the error of prediction
splitter = ['best','random']
max_depth = [1, 5, 10, None]
min_samples_split = [2, 3, 5, 10, 15]
min_samples_leaf = [1, 2, 3, 5, 10]
min_weight_fraction_leaf = [0, 2, 5, 10]
max_features = [None, 'auto', 'sqrt', 'log2']
max_leaf_nodes = [None, 2, 3, 4, 5]
min_impurity_decrease = [0,1,2,3]
ccp_alpha = [0,1,2,3]

#this will take alot of time ...so we are now reducing some values
'''
criterion = ['friedman_mse', 'absolute_error'] #calculating the error of prediction
splitter = ['best']
max_depth = [1, 5, 10, None]
min_samples_split = [2, 3, 5]
min_samples_leaf = [1, 2, 3]
min_weight_fraction_leaf = [0, 2, 5]
max_features = [None, 'auto', 'sqrt', 'log2']
max_leaf_nodes = [None, 2, 5]
min_impurity_decrease = [0,1,3]
ccp_alpha = [0,1,3]
```

```
param_dict = dict(criterion = criterion,
                  splitter= splitter,
                  max_depth = max_depth,
                  min_samples_split = min_samples_split,
                  min_samples_leaf = min_samples_leaf,
                  min_weight_fraction_leaf = min_weight_fraction_leaf,
                  max_features  = max_features,
                  max_leaf_nodes = max_leaf_nodes,
                  min_impurity_decrease = min_impurity_decrease,
                  ccp_alpha =ccp_alpha)
```

```
obj = DecisionTreeRegressor()
```

```
grid = GridSearchCV(estimator=obj, param_grid=param_dict, scoring = 'neg_root_mean_squared_error', verbose=1)
```

```
grid_result = grid.fit(Xtrain, ytrain)
```

```
finalmodel = grid_result.best_estimator_#it will show us the best value for each hyperparameter which is being chosen by gridsearch
```

```
finalmodel
```

```
DecisionTreeRegressor
DecisionTreeRegressor(ccp_alpha=1, criterion='absolute_error',
                      max_features='sqrt', min_impurity_decrease=3,
                      min_weight_fraction_leaf=0)
```

```
ytestpred = finalmodel.predict(Xtest)
```

```
from sklearn.metrics import mean_absolute_percentage_error
```

```
mean_absolute_percentage_error(ytest, ytestpred)
```

```
0.1519439298166267
```

Therfore; Accuracy = (100-0.15)% = 99.85%

Random Forests

```
from sklearn.ensemble import RandomForestRegressor
```

```
obj1 = RandomForestRegressor()


n_estimators=[2, 5, 10]#how many decision trees we want ?


criterion = ['friedman_mse'] #calculating the error of prediction
max_depth = [1, 5, 10, None]
min_samples_split = [2, 5]
min_samples_leaf = [1, 3]
min_weight_fraction_leaf = [0, 2, 5]
max_features = [None,'sqrt', 'log2']
max_leaf_nodes = [None, 2, 5]
min_impurity_decrease = [0,1,3]
ccp_alpha = [0,1,3]

param_dict = dict(n_estimators=n_estimators,
                  criterion = criterion,
                  max_depth = max_depth,
                  min_samples_split = min_samples_split,
                  min_samples_leaf = min_samples_leaf,
                  min_weight_fraction_leaf = min_weight_fraction_leaf,
                  max_features  = max_features,
                  max_leaf_nodes = max_leaf_nodes,
                  min_impurity_decrease = min_impurity_decrease,
                  ccp_alpha =ccp_alpha)




grid1 = GridSearchCV(estimator=obj1, param_grid=param_dict, scoring = 'neg_root_mean_squared_error', verbose=1)


grid_result1 = grid1.fit(Xtrain, ytrain)
```

```
Fitting 5 folds for each of 11664 candidates, totalling 58320 fits
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
38880 fits failed out of a total of 58320.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
19440 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_forest.py", line 340, in fit
    self._validate_params()
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
```

```
sklearn.utils._param_validation.InvalidParameterError: The 'min_weight_fraction_leaf' parameter of RandomForestRegressor must be a float in the range [0.0, 0.5]. Got 2 instead

--------------------------------------------------------------------------------
19440 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.9/dist-packages/sklearn/ensemble/_forest.py", line 340, in fit
    self._validate_params()
  File "/usr/local/lib/python3.9/dist-packages/sklearn/base.py", line 600, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.9/dist-packages/sklearn/utils/_param_validation.py", line 97, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'min_weight_fraction_leaf' parameter of RandomForestRegressor must be a float in the range [0.0, 0.5]. Got 5 instead

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.9/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non-finite: [-5611.77267003 -4685.95464249 -4840
            nan            nan]
  warnings.warn(
```

```
finalmodel1 = grid_result1.best_estimator_
```

```
finalmodel1
```

```
RandomForestRegressor

RandomForestRegressor(ccp_alpha=1, criterion='friedman_mse', max_depth=10,
                      max_features='log2', min_impurity_decrease=1,
                      min_samples_split=5, min_weight_fraction_leaf=0,
                      n_estimators=10)
```

```
ytestpred1 = finalmodel1.predict(Xtest)
```

```
from sklearn.metrics import mean_absolute_percentage_error

mean_absolute_percentage_error(ytest, ytestpred1)
```

```
0.13359206261878148
```

Therfore; Accuracy = (100-0.13)% = 99.87%

*CONCLUSION :*From all three model applied above i.e Artificial Neural Network(ANN), Decision Trees, Random Forest; we can say that Random Forest with accuracy of 99.85 is best model of all.