```
import numpy as np
np.random.seed(123)
import pandas as pd
import pandas.testing as tm
import pandas.util.testing as tm
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from tqdm import tqdm_notebook
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
from sklearn.decomposition import FastICA
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import linear_model
from sklearn import metrics
from sklearn.svm import SVR
import json
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: FutureWarning: pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  """
```

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

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

- Loading the DataSets

```
# loading the train and test dataset
df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-benz-greener-manufacturing/train.csv/train.csv')
```

```
# converting categorical features to numerical features
x_dummies=pd.get_dummies(df_1, prefix_sep='_', drop_first=True)
x_dummies.head()
#https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.get_dummies.html
#https://towardsdatascience.com/encoding-categorical-features-21a2651a065c
```

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26	X27	X28	X29	X30	X31	X32	X33	X34	X35	X36	X37	X38	X39	X40	X41	X42	X43	X44	X45
0	0	130.81	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
1	6	88.53	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0
2	7	76.26	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0
3	9	80.62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0
4	13	78.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	0	0

5 rows × 557 columns

```
# creating a list of columns which have only zeros
zeros=[]
for i,j in x_dummies.any().items():#https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.iteritems.html
    if j==False:
        zeros.append(i)
```

```
zeros # 'X339' is missing in my code
# we need to drop these columns
```

```
['X11',
 'X93',
 'X107',
 'X233',
 'X235',
 'X268',
 'X289',
 'X290',
 'X293',
 'X297',
 'X330',
 'X347']
```

```
#https://stackoverflow.com/questions/29294983/how-to-calculate-correlation-between-all-columns-and-remove-highly-correlated-on
# to delete columns with high correlation
```

- Dropping columns with only zeros

```
x_dummies = x_dummies.drop(zeros, axis=1)
```

- Dropping outliers => considering only IQR

```
[ ]  1 x_filtered= x_dummies[x_dummies ['y']>70]#https://www.geeksforgeeks.org/drop-rows-from-the-dataframe-based-on-certain-condition-applied-on-a-column/
     2 x_filtered= x_filtered[x_filtered['y']<150]
     3 x_filtered.describe()
```

	ID	y	X10	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	419
count	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	
mean	4209.773724	100.439938	0.013352	0.074392	0.057940	0.428231	0.000477	0.002623	0.007630	0.007868	0.099666	0.142823	0.002623	0.086791	
std	2437.897217	11.994753	0.114792	0.262439	0.233658	0.494881	0.021835	0.051152	0.087026	0.088365	0.299590	0.349934	0.051152	0.281562	
min	0.000000	72.110000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2096.250000	90.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4224.000000	99.090000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	6316.750000	108.967500	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
max	8417.000000	149.630000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 545 columns

```
[ ]  1 print(x_filtered.shape)
     2 y=x_filtered['y']
     3 x_filtered=x_filtered.drop(['y'], axis=1)
     4 print(x_filtered.shape, y.shape)
     5 print(type(x_filtered), type(y))
```

```
(4194, 545)
(4194, 544) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
[ ]  1 top_20_features= ['ID', 'X14', 'X29', 'X54', 'X76', 'X118', 'X119', 'X127', 'X132',
     2        'X136', 'X189', 'X222', 'X232', 'X263', 'X279', 'X311', 'X314', 'X315',
     3        'X6_g', 'X6_j']
     4 type(top_20_features)
```

```
list
```

- Finding important features using VIF

```
[ ]  1 from tqdm import tqdm_notebook
```

```
[ ]  1 from statsmodels.stats.outliers_influence import variance_inflation_factor
     2 def calc_vif(X):
     3
     4     # Calculationg VIF
     5     vif=pd.DataFrame()
     6     vif['variables']=X.columns
     7     vif['VIF']= [variance_inflation_factor(X.values, i) for i in tqdm_notebook(range(X.shape[1]))]
     8
     9     return (vif)
    10
```

```
[ ]  1 vif_values=calc_vif(x_filtered)
```

```
100% 544/544 [14:15<00:00, 1.57s/it]
```

```
[ ]  1 vif_values.head()
```

	variables	VIF
0	ID	487.143315
1	X10	inf
2	X12	inf
3	X13	inf
4	X14	inf

```
[ ]  1 vif_values=vif_values.sort_values(by='VIF', ascending=True)
```

```
[ ]  1 vif_values.head()
```

	variables	VIF
173	X190	1.077204
305	X332	1.101729
31	X42	1.125967
267	X288	1.128819
91	X104	1.170182

```
[ ]  1 lst_variables= list(vif_values['variables'])
     2 lst_variables[0:5]
```

```
['X190', 'X332', 'X42', 'X288', 'X104']
```

```
[ ]  1 lst_vif= list(vif_values['VIF'])
     2 lst_vif[0:5]
```

```
[1.0772040419357858,
 1.101729119950782,
 1.1259673158700982,
 1.1288191495521718,
 1.1701823277500567]
```

```
vif_dict = dict(zip(lst_variables, lst_vif))
```

```
type(lst_vif[1])
```

```
float
```

```
np.isinf(lst_vif[0])
```

```
False
```

```
inf_indices=[]
count=0
for k,v in vif_dict.items():
    if np.isinf(v):
        inf_indices.append(k)
        count+=1
```

```
count
```

```
373
```

```
len(inf_indices)
```

```
373
```

```
for i in inf_indices:
    print(vif_dict[i])
```

```
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
inf
```

```
inf_indices
```

```
['X184',
 'X1_v',
 'X1_u',
 'X1_t',
 'X1_s',
 'X1_r',
 'X1_q',
 'X1_p',
 'X1_o',
 'X1_n',
 'X1_l',
 'X1_k',
 'X1_j',
 'X1_i',
 'X1_h',
 'X1_g',
 'X1_f',
 'X1_e',
 'X1_d',
```

```
'X1_c',
'X1_m',
'X1_w',
'X1_y',
'X1_z',
'X2_av',
'X2_au',
'X2_at',
'X2_as',
'X2_ar',
'X2_aq',
'X2_ap',
'X2_ao',
'X2_an',
'X2_am',
'X2_al',
'X2_ak',
'X2_ai',
'X2_ah',
'X2_ag',
'X2_af',
'X2_ae',
'X2_ac',
'X2_aa',
'X1_b',
'X1_ab',
'X0_z',
'X2_aw',
'X0_b',
'X0_az',
'X0_ay',
'X0_ax',
'X0_aw',
'X0_au',
'X0_at',
'X0_as',
'X0_aq',
'X0_ap',
'X0_ao',
'X0_am',
```

```
[ ]  1 # removing top_20_features from the features that need to be dropped from VIF
     2 inf_indices=list(set(inf_indices)-set(top_20_features))
```

```
[ ]  1 import json
     2 with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices.txt', 'w') as f:
     3     f.write(json.dumps(inf_indices))
```

```
[ ]  1 with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices.txt', 'r') as f:
     2     checc=json.loads(f.read())
     3
```

```
[ ]  1 inf_indices=checc
     2 inf_indices
```

```
[ ]  1 print(x_filtered.shape)
     2 x_filtered = x_filtered.drop(inf_indices, axis=1)
     3 #x_dummies_2= x_dummies_2.drop(inf_indices, axis=1)
     4 print(x_filtered.shape)
     5 #print(x_dummies_2.shape)
```

```
(4194, 544)
(4194, 188)
```

- Splitting data into Train and Test

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 x_train, x_test, y_train, y_test = train_test_split(x_filtered, y, test_size=0.2, random_state=42)
```

https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be

- From the above link we have learned that apart from SVD we can also add features using PCA, ICA, LDA, LLE
- Adding SVD features

```
[ ]  1 from sklearn.decomposition import TruncatedSVD
     2 tsvd= TruncatedSVD(n_components=2, random_state=42)
     3 tsvd_train= tsvd.fit_transform(x_train)
     4 tsvd_test=  tsvd.transform(x_test)
```

```
[ ]  1 tsvd_train.shape
```

```
(3355, 2)
```

- PCA

```
[ ]  1 from sklearn.decomposition import PCA
     2 pca = PCA(n_components=2, random_state=42)
     3 pca_train= pca.fit_transform(x_train)
     4 pca_test= pca.transform(x_test)
```

```
[ ]  1 pca_train.shape
```

```
(3355, 2)
```

- ICA

```
[ ]  1 from sklearn.decomposition import FastICA
     2 ica=FastICA(n_components=2, random_state=42)
     3 ica_train= ica.fit_transform(x_train)
     4 ica_test= ica.transform(x_test)
```

```
[ ]  1 ica_train.shape
```

```
(3355, 2)
```

- adding all these new features to the dataframe

```
[ ]  1 for i in range(0, tsvd_train.shape[1]):
     2     x_train['tsvd_'+str(i)]= tsvd_train[:, i]
     3     x_test['tsvd_'+str(i)]= tsvd_test[:, i]
     4     x_train['pca_'+str(i)]= pca_train[:, i]
     5     x_test['pca_'+str(i)]= pca_test[:, i]
     6     x_train['ica_'+str(i)]= ica_train[:, i]
     7     x_test['ica_'+str(i)]= ica_test[:, i]
```

- Two way and Three way Feature interaction

```
[ ]  1 x_train['X64 + X218']=x_train['X64']+x_train['X218']
     2 x_test['X64 + X218']=x_test['X64']+x_test['X218']
     3
     4 x_train['X218 + X224 + X273']=x_train['X218']+x_train['X224'] + x_train['X273']
     5 x_test['X218 + X224 + X273']=x_test['X218']+x_test['X224'] + x_test['X273']
     6
     7 x_train['X64 + X224 + X273']=x_train['X64']+x_train['X224'] + x_train['X273']
     8 x_test['X64 + X224 + X273']=x_test['X64']+x_test['X224'] + x_test['X273']
     9
    10
    11 x_train['X314 + X315']=x_train['X314']+x_train['X315']
    12 x_test['X314 + X315']=x_test['X314']+x_test['X315']
    13
    14 x_train['X314 + X315 + X29']=x_train['X314']+x_train['X315'] + x_train['X29']
    15 x_test['X314 + X315 + X29']=x_test['X314']+x_test['X315'] + x_test['X29']
    16
    17 x_train['X314 + X315 + X118']=x_train['X314']+x_train['X315'] + x_train['X118']
    18 x_test['X314 + X315 + X118']=x_test['X314']+x_test['X315'] + x_test['X118']
    19
    20 x_train['X127 + X189']=x_train['X127']+x_train['X189']
    21 x_test['X127 + X189']=x_test['X127']+x_test['X189']
    22
    23 x_train['X118 + X54']=x_train['X118']+x_train['X54']
    24 x_test['X118 + X54']=x_test['X118']+x_test['X54']
    25
```

```
[ ]  1 print(x_train.shape, x_test.shape)
```

```
(3355, 202) (839, 202)
```

1: RandomForestRegressor

```
[ ]  1 from sklearn.ensemble import RandomForestRegressor
     2 # n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, bootstrap
     3 from sklearn.model_selection import RandomizedSearchCV
     4
     5 # Number of trees in a Random forest
     6 n_estimators=[int(x) for x in np.linspace (start=100, stop=1000, num=10)]
     7
     8 # Number of features to consider at every split
     9 max_features=['auto', 'sqrt']
    10
    11 # Maximun no of levels in a tree
    12 max_depth=[int(x) for x in np.linspace(10, 110, num = 11)]
    13 max_depth.append(None)
    14
    15 # minimum number of samples required to split a node
    16 min_samples_split = np.arange(50,250,20)
    17
    18 # Minimum number of samples required at each leaf node
    19 min_samples_leaf = np.arange(5,50,5)
    20 # Method of selecting samples for training each tree
    21 bootstrap = [True, False]
    22
    23 # Create the random grid
    24 random_grid = {'n_estimators': n_estimators,
    25                'max_features': max_features,
    26                'max_depth': max_depth,
    27                'min_samples_split': min_samples_split,
    28                'min_samples_leaf': min_samples_leaf,
    29                'bootstrap': bootstrap}
    30 print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_s
```

```
[ ]  1 # Use the random grid to search for best hyperparameters
     2 # First create the base model to tune
     3 rf = RandomForestRegressor()
     4 # Random search of parameters, using 3 fold cross validation,
     5 # search across 100 different combinations, and use all available cores
     6 rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
     7 # Fit the random search model
     8 rf_random.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 14.9min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 29.4min finished
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   ccp_alpha=0.0,
```

```
                                             criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100,
                                             n_jobs=None, oob_score=Fals...
                                    'max_depth': [10, 20, 30, 40, 50, 60,
                                                  70, 80, 90, 100, 110,
                                                  None],
                                    'max_features': ['auto', 'sqrt'],
                                    'min_samples_leaf': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45]),
                                    'min_samples_split': array([ 50,  70,  90, 110, 130, 150, 170, 190, 210, 230]),
                                    'n_estimators': [100, 200, 300, 400,
                                                     500, 600, 700, 800,
                                                     900, 1000]},
                         pre_dispatch='2*n_jobs', random_state=42, refit=True,
                         return_train_score=False, scoring=None, verbose=2)
```

```
[ ]  1 rf_random.best_params_
```

```
{'bootstrap': False,
 'max_depth': 60,
 'max_features': 'sqrt',
 'min_samples_leaf': 5,
 'min_samples_split': 150,
 'n_estimators': 200}
```

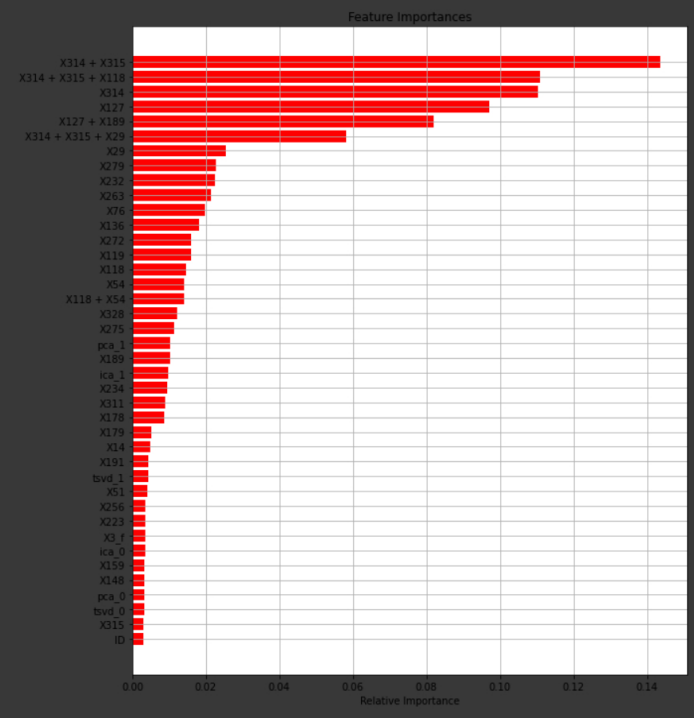
```
[ ]  1 best_rf=rf_random.best_estimator_
     2 best_rf
```

```
RandomForestRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                      max_depth=60, max_features='sqrt', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=5,
                      min_samples_split=150, min_weight_fraction_leaf=0.0,
                      n_estimators=200, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

```
[ ]  1 #temp
     2 best_rf= RandomForestRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
     3                       max_depth=60, max_features='sqrt', max_leaf_nodes=None,
     4                       max_samples=None, min_impurity_decrease=0.0,
     5                       min_impurity_split=None, min_samples_leaf=5,
     6                       min_samples_split=150, min_weight_fraction_leaf=0.0,
     7                       n_estimators=200, n_jobs=None, oob_score=False,
     8                       random_state=None, verbose=0, warm_start=False)
```

```
[ ]  1 # fitting the best model on to the dataset
     2 best_rf.fit(x_train, y_train)
     3 y_train_pred=best_rf.predict(x_train)
     4 y_test_pred=best_rf.predict(x_test)
```

```
[ ]  1 features = x_train.columns
     2 importances = best_rf.feature_importances_
     3 indices = (np.argsort(importances))[-40:]
     4 plt.figure(figsize=(10,12))
     5 plt.title('Feature Importances')
     6 plt.barh(range(len(indices)), importances[indices], color='r', align='center')
     7 plt.yticks(range(len(indices)), [features[i] for i in indices])
     8 plt.xlabel('Relative Importance')
     9 plt.grid()
    10 plt.show()
```



```
# checking r2 score
from sklearn.metrics import r2_score
print('train r2_score', r2_score(y_train, y_train_pred))
print('='*30)
print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.6574681408333823
==============================
test r2_score 0.6483743021527534
```

2: XGBoostRegressor

```
import xgboost as xgb
```

```
xgb_model=xgb.XGBRegressor()
xgb_model.fit(x_train, y_train)
y_pred=xgb_model.predict(x_train)
learning_rate=[0.001, 0.01, 0.1, 0.2, 0.3]
n_estimators=[50, 100, 150, 200, 250, 300]
hyperparameters= dict(learning_rate=learning_rate, n_estimators=n_estimators)

```

```
[05:33:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
from sklearn.model_selection import RandomizedSearchCV
rsearch = RandomizedSearchCV(xgb_model, hyperparameters, n_iter=10, random_state=41)
rsearch.fit(x_train, y_train)
print(rsearch.best_params_)
```

```
[05:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:33:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:33:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:34:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
{'n_estimators': 150, 'learning_rate': 0.1}
```

```
from sklearn.metrics import r2_score
train_r2=[]
cv_r2=[]
dict_cv_r2={}
learning_rate=[0.3, 0.2, 0.1, 0.01, 0.001]
n_estimators=[50, 100, 150, 200, 250, 300]
for i in tqdm_notebook(learning_rate):
    for j in tqdm_notebook(n_estimators):
        xb=xgb.XGBRegressor(learning_rate=i,n_estimators=j)
        xb.fit(x_train,y_train)
        y_cap_train=xb.predict(x_train)
        y_cap_cv=xb.predict(x_test)
        train_r2.append(r2_score(y_train,y_cap_train))
        k=r2_score(y_test,y_cap_cv)
        dict_cv_r2[k]=i,j
        cv_r2.append(r2_score(y_test,y_cap_cv))
maximum_auc_score=max(cv_r2)
print(maximum_auc_score)
```

```
100% 5/5 [06:04<00:00, 72.81s/it]
100% 6/6 [06:04<00:00, 60.67s/it]
[05:35:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:35:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
[05:36:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

100% 6/6 [00:34<00:00, 5.68s/it]
[05:36:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

100% 6/6 [05:29<00:00, 54.98s/it]
[05:36:30] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

100% 6/6 [05:12<00:00, 52.15s/it]
[05:36:47] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:36:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

100% 6/6 [04:55<00:00, 49.28s/it]
[05:37:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:37:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:37:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:37:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:37:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
[05:37:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.

0.6604956264525383
```

```
[ ] 1 print(dict_cv_r2)
    2 dict_cv_r2[maximum_auc_score]
```

```
{0.6402537526049236: (0.3, 50), 0.6252262886435493: (0.3, 100), 0.6157867395052805: (0.3, 150), 0.6022101794445373: (0.3, 200), 0.5986576037823697: (0.3, 250), 0.5920276258487682: (0.3, (0.1, 50)
```

```
[ ] 1
```

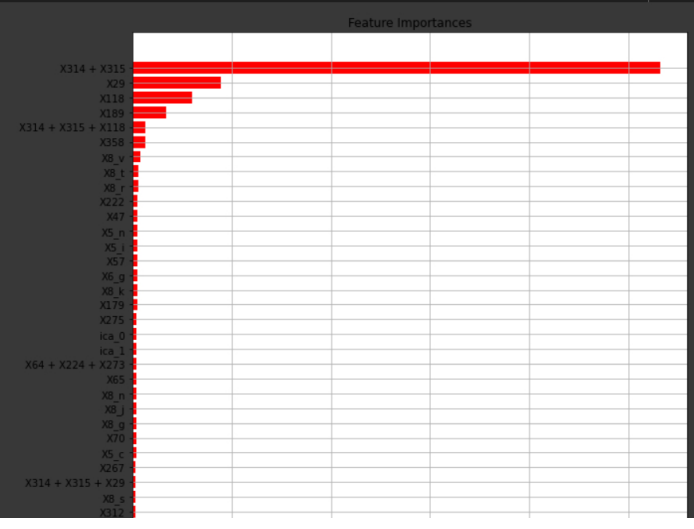
```
[ ] 1 best_xgb=rsearch.best_estimator_
    2 best_xgb
```

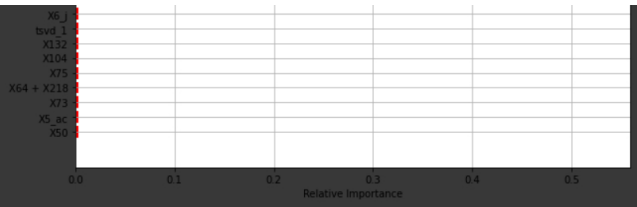
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=150,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

```
[ ] 1 # fitting the best model on to the dataset
    2 best_xgb.fit(x_train, y_train)
    3 y_train_pred=best_xgb.predict(x_train)
    4 y_test_pred=best_xgb.predict(x_test)
```

```
[05:37:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

```
[ ] 1 import matplotlib.pyplot as plt
    2 features = x_train.columns
    3 importances = best_xgb.feature_importances_
    4 indices = (np.argsort(importances))[-40:]
    5 plt.figure(figsize=(10,12))
    6 plt.title('Feature Importances')
    7 plt.barh(range(len(indices)), importances[indices], color='r', align='center')
    8 plt.yticks(range(len(indices)), [features[i] for i in indices])
    9 plt.xlabel('Relative Importance')
   10 plt.grid()
   11 plt.show()
```






```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.6931087582848501
==============================
test r2_score 0.6510758493146287
```

3: DecisionTreeRegressor

```
[ ]  1 from sklearn.tree import DecisionTreeRegressor
     2 dt_model=DecisionTreeRegressor()
     3 dt_model
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
[ ]  1 max_depth=[1, 5, 10, 50]
     2 min_samples_split=[5, 10, 100, 500]
     3 hyperparameters= dict(max_depth=max_depth, min_samples_split=min_samples_split)
     4
```

```
[ ]  1 from sklearn.model_selection import RandomizedSearchCV
     2 rsearch = RandomizedSearchCV(dt_model, hyperparameters, n_iter=10, random_state=41)
     3 rsearch.fit(x_train, y_train)
     4 print(rsearch.best_params_)
```

```
{'min_samples_split': 500, 'max_depth': 5}
```

```
[ ]  1 rsearch.best_params_
```

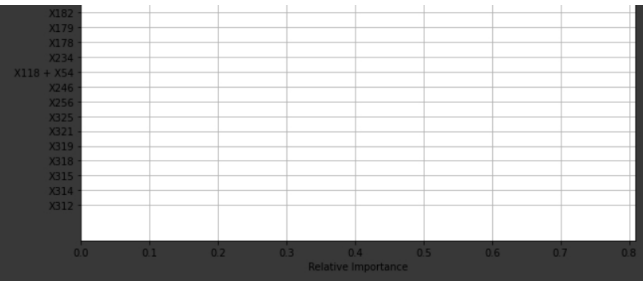
```
{'max_depth': 5, 'min_samples_split': 500}
```

```
[ ]  1 best_dt=rsearch.best_estimator_
     2 best_dt
```

```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=5,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=500,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

```
[ ]  1 # fitting the best model on to the dataset
     2 best_dt.fit(x_train, y_train)
     3 y_train_pred=best_dt.predict(x_train)
     4 y_test_pred=best_dt.predict(x_test)
```

```
[ ]  1 import matplotlib.pyplot as plt
     2 features = x_train.columns
     3 importances = best_dt.feature_importances_
     4 indices = (np.argsort(importances))[-40:]
     5 plt.figure(figsize=(10,12))
     6 plt.title('Feature Importances')
     7 plt.barh(range(len(indices)), importances[indices], color='r', align='center')
     8 plt.yticks(range(len(indices)), [features[i] for i in indices])
     9 plt.xlabel('Relative Importance')
    10 plt.grid()
    11 plt.show()
```




```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.6290528360587266
==============================
test r2_score 0.6554069343566447
```

4: Simple Linear Regressor(SGD)

```
[ ]  1 from sklearn.linear_model import SGDRegressor
     2 linear_regressor= SGDRegressor(loss='squared_loss')
     3 linear_regressor
```

```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
```

```
[ ]  1 parameters = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'eta0':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000],  'epsilon':[0.0001, 0.001, 0.01, 0.1, 1, 10, 1
     2 gsearch = GridSearchCV(linear_regressor, parameters, scoring='r2')
     3 gsearch.fit(x_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SGDRegressor(alpha=0.0001, average=False,
                                    early_stopping=False, epsilon=0.1,
                                    eta0=0.01, fit_intercept=True,
                                    l1_ratio=0.15, learning_rate='invscaling',
                                    loss='squared_loss', max_iter=1000,
                                    n_iter_no_change=5, penalty='l2',
                                    power_t=0.25, random_state=None,
                                    shuffle=True, tol=0.001,
                                    validation_fraction=0.1, verbose=0,
                                    warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                   10000],
                         'epsilon': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                     10000],
                         'eta0': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                  10000],
                         'penalty': ['l1', 'l2']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
[ ]  1 print(gsearch.best_estimator_)
     2 print(gsearch.best_score_)
```

```
SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.01,
             eta0=0.0001, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='squared_loss', max_iter=1000,
             n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
-1.102111593420816e+25
```

```
[ ]  1 linear_regressor=SGDRegressor(alpha=0.0001, average=False, early_stopping=False, epsilon=0.01,
     2              eta0=0.0001, fit_intercept=True, l1_ratio=0.15,
     3              learning_rate='invscaling', loss='squared_loss', max_iter=1000,
     4              n_iter_no_change=5, penalty='l2', power_t=0.25, random_state=None,
     5              shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
     6              warm_start=False)
```

```
[ ]  1 mse= cross_val_score(linear_regressor, x_train, y_train, scoring='r2', cv=5)
     2 mean_mse=np.mean(mse)
     3 print(mean_mse)
```

```
-6.282814753035179e+25
```

```
[ ]  1 linear_regressor.fit(x_train, y_train)
     2 y_train_pred=linear_regressor.predict(x_train)
     3 y_test_pred =linear_regressor.predict(x_test)
```

```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score -3.531115895059146e+25
==============================
test r2_score -3.536211676998593e+25
```

5: Ridge Regressor

```
[ ]  1 from sklearn.linear_model import Ridge
     2 ridge_regressor= Ridge()
     3 ridge_regressor
```

```
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
[ ]  1 parameters= {'alpha': [10,20,  50, 100, 150, 200, 250, 300]}
     2 ridge_grid_scv= GridSearchCV(ridge_regressor, parameters, scoring='r2', cv=5)
     3 ridge_grid_scv.fit(x_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=None, normalize=False, random_state=None,
                             solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [10, 20, 50, 100, 150, 200, 250, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
[ ]  1 print(ridge_grid_scv.best_params_)
     2 print(ridge_grid_scv.best_score_)
```

```
{'alpha': 50}
0.5954015864151923
```

```
[ ]  1 ridge_regressor= Ridge(alpha=50)
     2 ridge_regressor
```

```
Ridge(alpha=50, copy_X=True, fit_intercept=True, max_iter=None, normalize=False,
      random_state=None, solver='auto', tol=0.001)
```

```
[ ]  1 ridge_regressor.fit(x_train, y_train)
     2 y_train_pred=ridge_regressor.predict(x_train)
     3 y_test_pred =ridge_regressor.predict(x_test)
```

```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.622606917016735
==============================
test r2_score 0.6398563894858129
```

6: Lasso Regressor

```
[ ]  1 lasso= Lasso()
     2 lasso
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
[ ]  1 parameters= {'alpha': [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
     2 lasso_grid_scv= GridSearchCV(lasso, parameters, scoring='r2', cv=5)
     3 lasso_grid_scv.fit(x_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True,
                             max_iter=1000, normalize=False, positive=False,
                             precompute=False, random_state=None,
                             selection='cyclic', tol=0.0001, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [1e-15, 1e-10, 1e-08, 0.0001, 0.001, 0.01, 1,
                                   5, 10, 20]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
[ ]  1 print(lasso_grid_scv.best_params_)
     2 print(lasso_grid_scv.best_score_)
```

```
{'alpha': 0.01}
0.596278101998697
```

```
[ ]  1 lasso_regressor= Lasso(alpha=0.01)
     2 lasso_regressor
```

```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
```

```
[ ]  1 #https://towardsdatascience.com/how-to-perform-lasso-and-ridge-regression-in-python-3b3b75541ad8
     2 #https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/
     3 #https://alfurka.github.io/2018-11-18-grid-search/
     4 lasso_regressor.fit(x_train, y_train)
     5 y_train_pred=lasso_regressor.predict(x_train)
     6 y_test_pred =lasso_regressor.predict(x_test)
```

```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.6256643013649388
```

```
==============================
test r2_score 0.6463713208948576
```

7: SupportVectorRegressor

```
[ ]  1 from sklearn.svm import SVR
     2 svr= SVR()
     3 svr
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
[ ]  1 parameters = {'C':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'epsilon':[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]}
     2 gsearch = GridSearchCV(svr, parameters, scoring='r2')
     3 gsearch.fit(x_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='scale', kernel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                               10000],
                         'epsilon': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000,
                                     10000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='r2', verbose=0)
```

```
[ ]  1 print(gsearch.best_estimator_)
     2 print(gsearch.best_score_)
```

```
SVR(C=10000, cache_size=200, coef0=0.0, degree=3, epsilon=1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
0.0932147253722481
```

```
[ ]  1 svr= SVR(C=10000, cache_size=200, coef0=0.0, degree=3, epsilon=1, gamma='scale',
     2     kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
     3 svr
```

```
SVR(C=10000, cache_size=200, coef0=0.0, degree=3, epsilon=1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
[ ]  1 svr.fit(x_train, y_train)
     2 y_train_pred=svr.predict(x_train)
     3 y_test_pred =svr.predict(x_test)
```

```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2_score', r2_score(y_train, y_train_pred))
     4 print('='*30)
     5 print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.12238421871397831
==============================
test r2_score 0.1197384354647586
```

8: Deep Neural Network

```
[ ]  1 from tensorflow.keras.models import Sequential
     2 from tensorflow.keras.layers import Dense
     3 from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
     4 from sklearn.model_selection import cross_val_score
     5 from sklearn.preprocessing import LabelEncoder
     6 from sklearn.model_selection import StratifiedKFold
     7 from sklearn.preprocessing import StandardScaler
     8 from sklearn.pipeline import Pipeline
     9 from tensorflow.keras.utils import to_categorical
    10 from numpy import array
    11 import tensorflow as tf
    12 from tensorflow.keras.layers import Dense,Input,Activation
    13 from tensorflow.keras.models import Model
    14 import random as rn
    15 import logging
    16 from sklearn.metrics import roc_auc_score
    17 from tensorflow.keras.callbacks import Callback
    18 import keras.backend as k
    19 import tensorflow
```

```
[ ]  1 import keras.backend as K
     2 from sklearn.metrics import roc_auc_score
     3 from tensorflow.keras.callbacks import ModelCheckpoint
     4 from tensorflow.keras.callbacks import LearningRateScheduler
     5 import keras
     6 from keras.initializers import RandomNormal,RandomUniform
```

```
[ ]  1 # custom R2-score metrics for keras backend
     2 #https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion/34019
     3 from keras import backend as K
     4
     5 def r2_keras(y_true, y_pred):
     6     SS_res =  K.sum(K.square(y_true - y_pred))
     7     SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
     8     return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

```
[ ]  1 filepath="weights-{epoch:02d}-{val_r2_keras:.4f}.hdf5"
     2 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_r2_keras',  verbose=1, save_best_only=True, mode='max')
     3 #https://stackoverflow.com/questions/48971221/keras-modelcheckpoint-monitor-multiple-values
```

8 rows × 545 columns

```
[ ]  1 print(x_filtered_.shape)
     2 y_=x_filtered_['y']
     3 x_filtered_=x_filtered_.drop(['y'], axis=1)
     4 print(x_filtered_.shape, y_.shape)
     5 print(type(x_filtered_), type(y_))
```

```
(4194, 545)
(4194, 544) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
[ ]  1 from sklearn.model_selection import train_test_split
     2 X_train, X_test, Y_train, Y_test = train_test_split(x_filtered_, y_, test_size=0.2, random_state=42)
```

```
[ ]  1 X_train=X_train.to_numpy()
     2 Y_train=Y_train.to_numpy()
     3 X_test=X_test.to_numpy()
     4 Y_test=Y_test.to_numpy()
     5
     6 print(type(X_train), type(Y_train), type(X_test), type(Y_test))
     7 print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
(3355, 544) (3355,) (839, 544) (839,)
```

```
[ ]  1 Y_train=Y_train.reshape(X_train.shape[0],1)
     2 Y_test=Y_test.reshape(X_test.shape[0],1)
     3 print(Y_train.shape, Y_test.shape)
```

```
(3355, 1) (839, 1)
```

```
[ ]  1 print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

```
(3355, 544) (3355, 1) (839, 544) (839, 1)
```

```
[ ]  1 X_train
```

```
array([[5491,    0,    0, ...,    0,    0,    0],
       [2388,    0,    0, ...,    0,    0,    0],
       [8069,    0,    0, ...,    0,    0,    0],
       ...,
       [6232,    0,    0, ...,    0,    0,    0],
       [7588,    0,    0, ...,    0,    0,    0],
       [1716,    0,    0, ...,    0,    0,    0]])
```

```
[ ]  1 Y_train
```

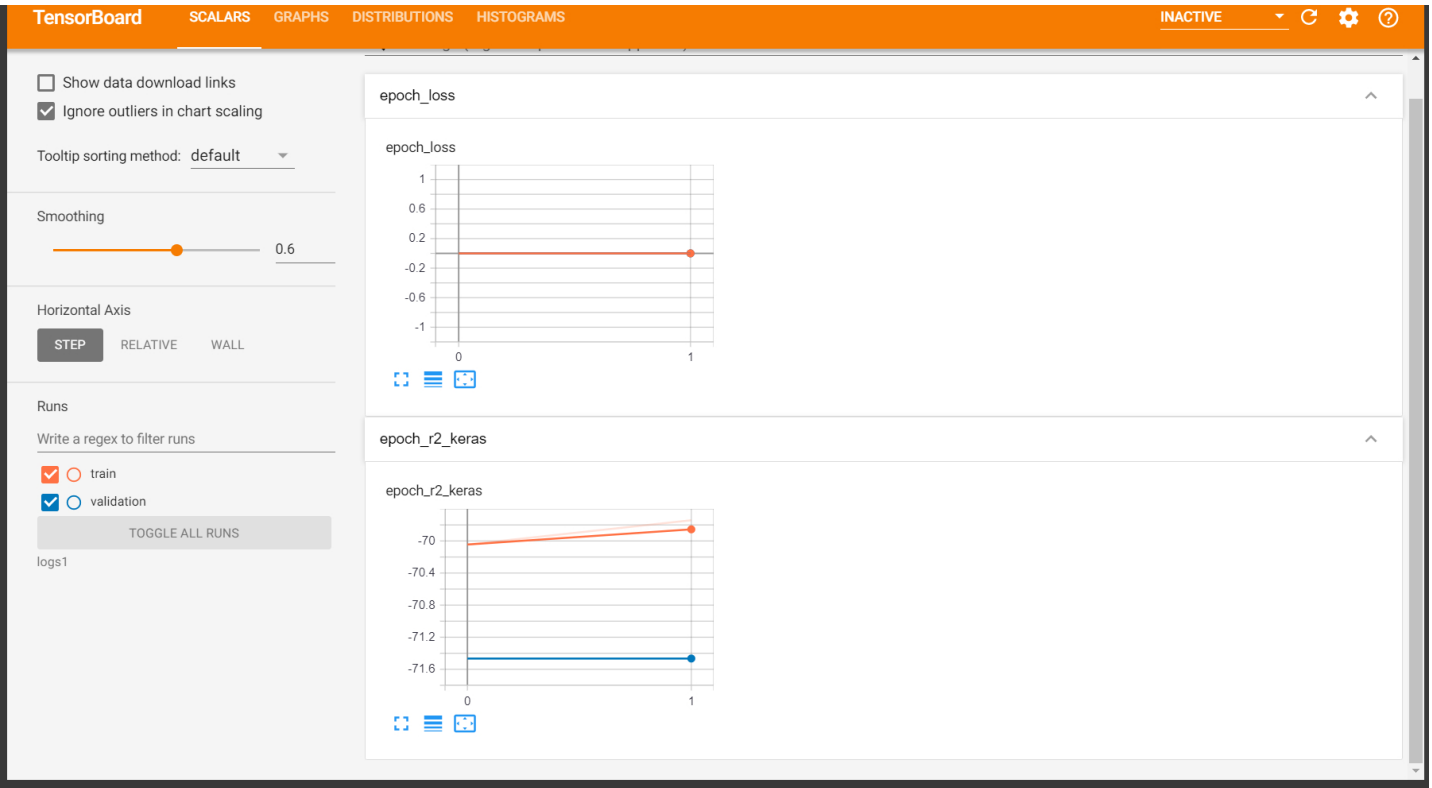
```
array([[111.56],
       [121.27],
       [106.4 ],
       ...,
       [110.  ],
       [ 90.11],
       [ 92.12]])
```

```
[ ]  1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs1', histogram_freq=1)
```

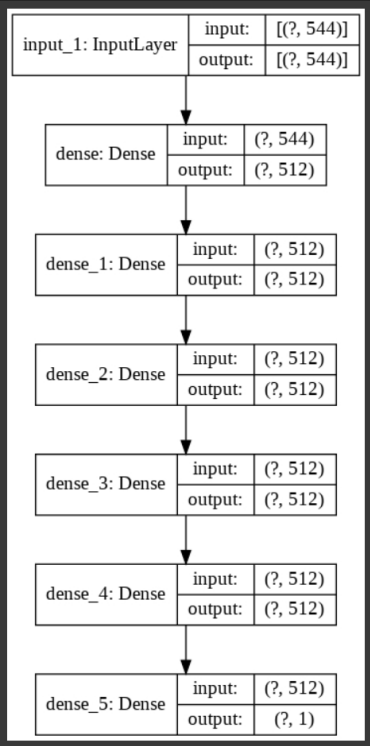
```
[ ]  1 # input layer
     2 input_layer=Input(shape=(X_train.shape[1],))
     3 # dense hidden layers                                    52.12 notebook has the same code randomnormal
     4 layer1=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform(seed=None))(input_layer)
     5 layer2=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform(seed=None))(layer1)
     6 layer3=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform(seed=None))(layer2)
     7 layer4=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform(seed=None))(layer3)
     8 layer5=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform(seed=None))(layer4)
     9 # output layer
    10 output=Dense(1,activation='softmax')(layer5)
    11
    12 #creating a model
    13 model=Model(inputs=input_layer, outputs=output)
    14
    15 #creating a callback list
    16 callback_list=[tensorboard_callback, earlystop,reduce_lr,checkpoint]
    17
    18 #cross entropy as loss function
    19 model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False),
    20               loss='categorical_crossentropy', metrics=[r2_keras])
    21 model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=256,callbacks=callback_list)
```

```
Epoch 1/5
 1/14 [=>............................] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -86.7688WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/summary_ops_
Instructions for updating:
use `tf.profiler.experimental.stop` instead.
13/14 [==========================>...] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -70.3159
Epoch 00001: val_r2_keras improved from -inf to -71.46603, saving model to weights-01--71.4660.hdf5
14/14 [==============================] - 1s 73ms/step - loss: 0.0000e+00 - r2_keras: -70.0445 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 2/5
13/14 [==========================>...] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -70.5522
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras did not improve from -71.46603
14/14 [==============================] - 1s 54ms/step - loss: 0.0000e+00 - r2_keras: -69.7416 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7fcd7988b8d0>
```

```
[ ]  1 %load_ext tensorboard
     2 import datetime, os
     3 %tensorboard --logdir logs1
     4 ##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf061fe4
```




```
[ ] 1 tf.keras.utils.plot_model(
    2     model, to_file='model_1.png', show_shapes=True, show_layer_names=True,
    3     rankdir='TB', expand_nested=False, dpi=96
    4 )
```



9: CNN

```
[ ] 1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs2', histogram_freq=1)
```

```
[ ]  1 # use 1D CNN
     2 from tensorflow.keras.layers import Conv1D
     3 #importing layers from tensorflow
     4 from tensorflow.keras.layers import Dense,concatenate,Activation,Dropout,Input, LSTM, Flatten
     5 from tensorflow.keras.models import Model
     6
     7 input = Input(shape=(X_train.shape[1], 1), name = 'input_1')
     8 conv1_1= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid',name='conv1_1')(input)
     9 drop_a=Dropout(0.6)(conv1_1)
    10 conv1_2= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv1_2')(drop_a)
    11 drop_b=Dropout(0.6)(conv1_2)
```

```
conv1_3= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv1_3')(drop_b)
flatten_c1= Flatten(data_format='channels_last',name='Flatten_c1')(conv1_3)

# adding the dense layers and dropout layers
dense_1_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_1_ac')(flatten_c1)
dropout_1_ac = Dropout(0.6)(dense_1_ac)
dense_2_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_2_ac')(dropout_1_ac)
dropout_2_ac = Dropout(0.6)(dense_2_ac)
dense_3_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_3_ac')(dropout_2_ac)
dropout_3_ac = Dropout(0.6)(dense_3_ac)
output = Dense(units=1, activation='softmax', kernel_initializer='he_normal', name='output')(dropout_3_ac)

model= Model(inputs=input, outputs=output)
```

```
[ ]  model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 544, 1)]          0
_________________________________________________________________
conv1_1 (Conv1D)             (None, 542, 64)           256
_________________________________________________________________
dropout (Dropout)            (None, 542, 64)           0
_________________________________________________________________
conv1_2 (Conv1D)             (None, 540, 64)           12352
_________________________________________________________________
dropout_1 (Dropout)          (None, 540, 64)           0
_________________________________________________________________
conv1_3 (Conv1D)             (None, 538, 64)           12352
_________________________________________________________________
Flatten_c1 (Flatten)         (None, 34432)             0
_________________________________________________________________
dense_1_ac (Dense)           (None, 64)                2203712
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
output (Dense)               (None, 1)                 65
=================================================================
Total params: 2,237,057
Trainable params: 2,237,057
Non-trainable params: 0
_________________________________________________________________
```

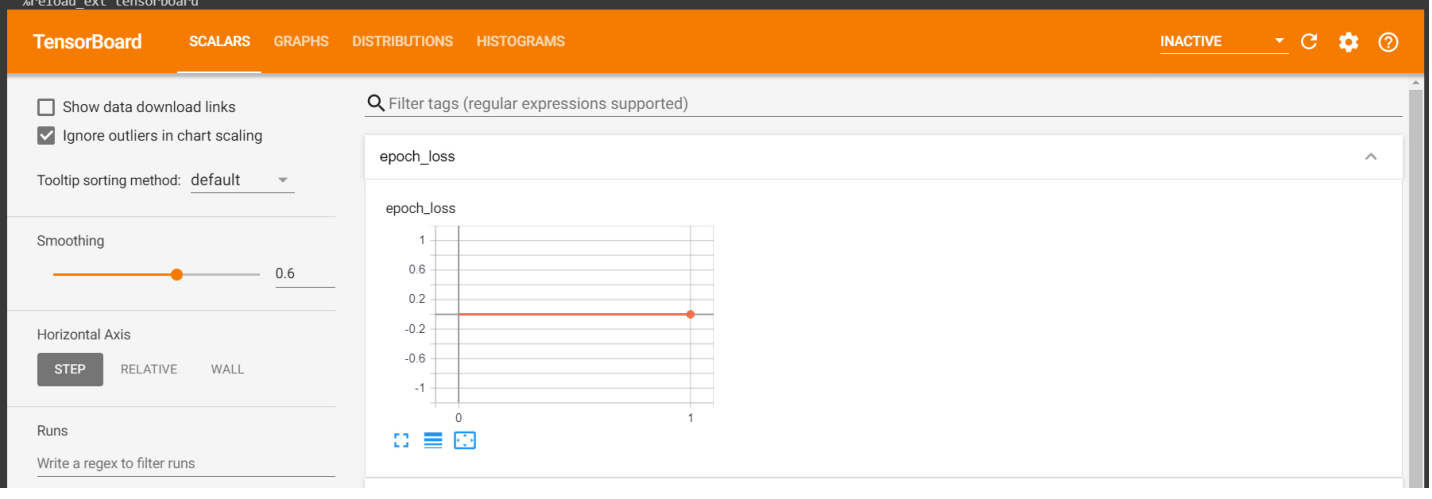
```
[ ]  #creating a callback list
callback_list=[tensorboard_callback, earlystop,reduce_lr,checkpoint]

#cross entropy as loss function
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False),
              loss='categorical_crossentropy', metrics=[r2_keras])
validation_data = (X_test, Y_test)
```

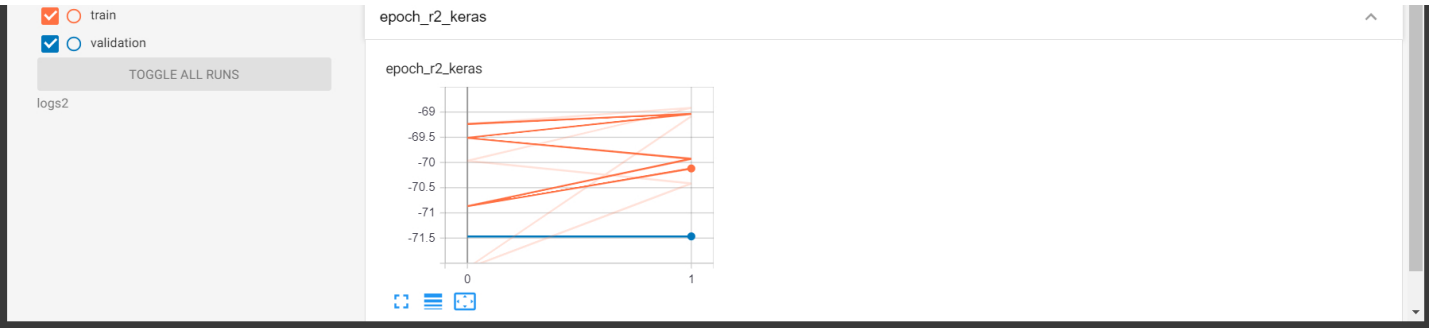
```
[ ]  model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=256,callbacks=callback_list)

```

```
Epoch 1/5
14/14 [==============================] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -72.0947
Epoch 00001: val_r2_keras did not improve from -71.46603
14/14 [==============================] - 11s 766ms/step - loss: 0.0000e+00 - r2_keras: -72.0947 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 2/5
14/14 [==============================] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -69.0840
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras did not improve from -71.46603
14/14 [==============================] - 10s 747ms/step - loss: 0.0000e+00 - r2_keras: -69.0840 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7fcd6ea829b0>
```

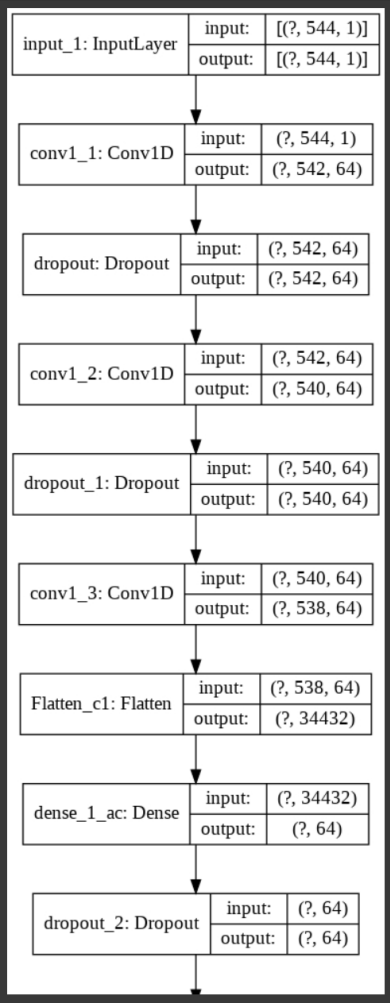
```
[ ]  %load_ext tensorboard
import datetime, os
%tensorboard --logdir logs2
##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf061fe4
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```






```
[ ]  1 tf.keras.utils.plot_model(
     2     model, to_file='model_2.png', show_shapes=True, show_layer_names=True,
     3     rankdir='TB', expand_nested=False, dpi=96
     4 )
```



10: LSTM

```
[ ]  1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs3', histogram_freq=1)
```

```
[ ]  1 input_1 = Input(shape=(X_train.shape[1], 1), name = 'input_lstm')
     2 lstm_1= LSTM(64)(input_1)
     3 flatten_1=Flatten()(lstm_1)
     4 # adding the dense layers and dropout layers
     5 dense_1_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_1_ac')(flatten_1)
     6 dropout_1_ac = Dropout(0.6)(dense_1_ac)
     7 dense_2_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_2_ac')(dropout_1_ac)
     8 dropout_2_ac = Dropout(0.6)(dense_2_ac)
     9 dense_3_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_3_ac')(dropout_2_ac)
    10 dropout_3_ac = Dropout(0.6)(dense_3_ac)
    11 output_1 = Dense(units=1, activation='softmax', kernel_initializer='he_normal', name='output')(dropout_3_ac)
    12
    13 model= Model(inputs=input_1, outputs=output_1)
```

```
[ ]  1 model.summary()
```

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_lstm (InputLayer)      [(None, 544, 1)]          0
```

```
lstm (LSTM)                  (None, 64)                16896
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dense_1_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
output (Dense)               (None, 1)                 65
=================================================================
Total params: 29,441
Trainable params: 29,441
Non-trainable params: 0
_________________________________________________________________
```

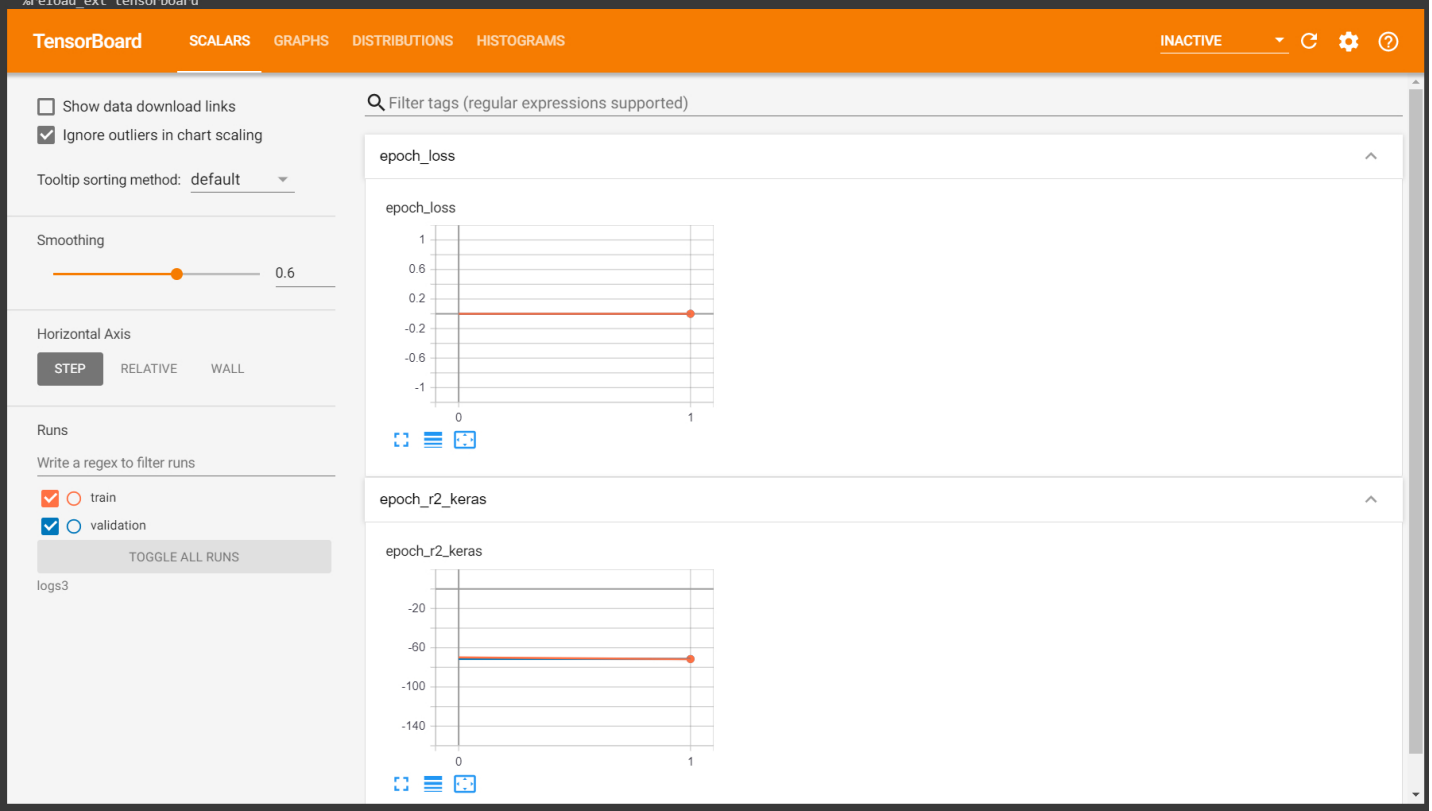
```
[ ]
#creating a callback list
callback_list=[tensorboard_callback, earlystop,reduce_lr,checkpoint]

#cross entropy as loss function
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False),
              loss='categorical_crossentropy', metrics=[r2_keras])
validation_data = (X_test, Y_test)
```

```
[ ]
model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=256,callbacks=callback_list)

```

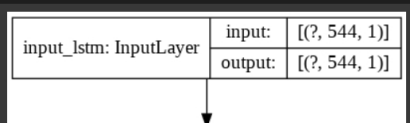
```
Epoch 1/5
14/14 [==============================] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -69.8794
Epoch 00001: val_r2_keras did not improve from -71.46603
14/14 [==============================] - 14s 990ms/step - loss: 0.0000e+00 - r2_keras: -69.8794 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 2/5
14/14 [==============================] - ETA: 0s - loss: 0.0000e+00 - r2_keras: -72.5599
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.0001000000047497451.

Epoch 00002: val_r2_keras did not improve from -71.46603
14/14 [==============================] - 13s 954ms/step - loss: 0.0000e+00 - r2_keras: -72.5599 - val_loss: 0.0000e+00 - val_r2_keras: -71.4660
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7fcd6d689630>
```

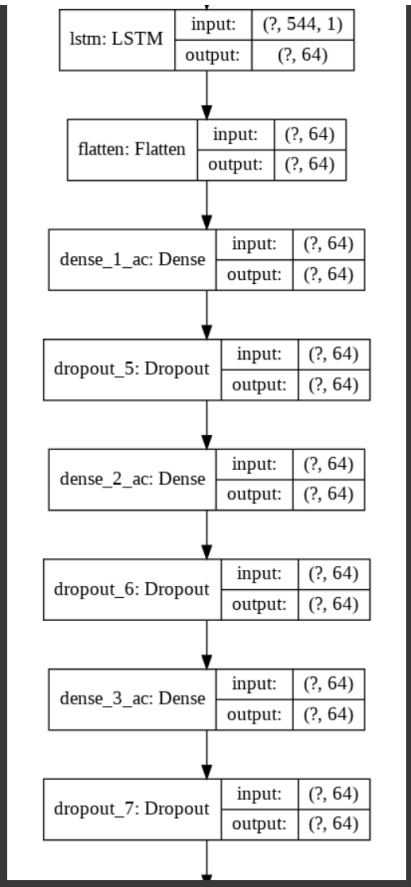
```
[ ]
%load_ext tensorboard
import datetime, os
%tensorboard --logdir logs3
##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf061fe4
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```




```
[ ]
tf.keras.utils.plot_model(
    model, to_file='model_3.png', show_shapes=True, show_layer_names=True,
    rankdir='TB', expand_nested=False, dpi=96
)
```






```
[ ]  1
```

11: Comparision of models

```
[ ]  1 #https://alfurka.github.io/2018-11-18-grid-search/
     2 def test(models, x_train, y_train, x_test, y_test, iterations = 100):
     3     results = {}
     4     for i in models:
     5         r2_train = []
     6         r2_test = []
     7         for j in range(iterations):
     8             r2_test.append(metrics.r2_score(y_test,
     9                                             models[i].fit(x_train,
    10                                                           y_train).predict(x_test)))
    11             r2_train.append(metrics.r2_score(y_train,
    12                                              models[i].fit(x_train,
    13                                                            y_train).predict(x_train)))
    14         results[i] = [np.mean(r2_train), np.mean(r2_test)]
    15     return pd.DataFrame(results)
```

```
[ ]  1 models = {'OLS': linear_model.LinearRegression(),
     2          'Lasso': linear_model.Lasso(),
     3          'Ridge': linear_model.Ridge(),}
```

```
[ ]  1 test(models, x_train, y_train, x_test, y_test)
```

	OLS	Lasso	Ridge
0	0.633391	0.516745	0.632844
1	-89462.866322	0.549245	0.632531

12: Summary

- RandomForestRegressor, XGBoostRegressor, DecisionTreeRegressor, SimpleLinearREgressor, Ridge Regressor, LassoRegressor Model's have been tuned for best Hyperparameters.
- Then they are applied on to the dataset.

Table:

Model Name	Train, Test (r^2 error)
RandomForestRegressor	0.6574681408333823, 0.6483743021527534
XGBoostRegressor	0.6931087582848501, 0.6510758493146287
DecisionTreeRegressor	0.6290528360587266, 0.6554069343566447
SimpleLinearRegressor	0.6333910409456334, -89462.86632170125
RidgeRegressor	0.622606917016735, 0.6398563894858129
LassoRegressor	0.6256643013649388, 0.6463713208948576

- DecisionTreeRegressor performed more than any other model, and SimpleLinearRegressor performed worse than any other model

```
[ ]  1
```