Modelling.ipynb

File Edit View Insert Runtime Tools Help Las...

Comment Share

+ Code + Text

Initializing Editing

```
[ ]  1 import numpy as np
     2 np.random.seed(123)
     3 import pandas as pd
     4 import pandas.testing as tm
     5 import pandas.util.testing as tm
     6 import seaborn as sns
     7 import warnings
     8 warnings.filterwarnings("ignore")
     9 from tqdm import tqdm_notebook
    10 from statsmodels.stats.outliers_influence import variance_inflation_factor
    11 from sklearn.model_selection import train_test_split
    12 from sklearn.decomposition import TruncatedSVD
    13 from sklearn.decomposition import PCA
    14 from sklearn.decomposition import FastICA
    15 from sklearn.ensemble import RandomForestRegressor
    16 from sklearn.model_selection import RandomizedSearchCV
    17 import matplotlib.pyplot as plt
    18 from sklearn.metrics import r2_score
    19 import xgboost as xgb
    20 from sklearn.tree import DecisionTreeRegressor
    21 from sklearn.linear_model import LinearRegression
    22 import statsmodels.api as sm
    23 from sklearn.model_selection import cross_val_score
    24 from sklearn.model_selection import GridSearchCV
    25 from sklearn.linear_model import Ridge
    26 from sklearn.linear_model import Lasso
    27 from sklearn import linear_model
    28 from sklearn import metrics
    29 from sklearn.svm import SVR
    30 import json
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5: FutureWarning: pandas.
  """
```

```
[ ]  1 # loading the train and test dataset
     2 df_1=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
     3 df_2=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Case Study 1/mercedes-b
```

```
[ ]  1 print(df_1.shape)
     2 print(df_2.shape)
```

```
(4209, 378)
(4209, 377)
```

```
[ ]  1 missing_cols = set( df_1.columns ) - set( df_2.columns )
     2 missing_cols
```

```
{'y'}
```

```
[ ]  1 # converting categorical features to numerical features
     2 x_dummies_1=pd.get_dummies(df_1)
     3 x_dummies_1.head()
     4 x_dummies_2=pd.get_dummies(df_2)
     5 x_dummies_2.head()
     6 #https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.get_dummie
```


	ID	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24	X26
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2	3	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

5 rows × 570 columns

```
[6]  1 print(x_dummies_1.shape)
     2 print(x_dummies_2.shape)
```

```
(4209, 565)
(4209, 570)
```

```
[7]  1 missing_cols = set( x_dummies_1.columns ) - set( x_dummies_2.columns )
```

```
[8]  1 missing_cols
```

```
{'X0_aa',
 'X0_ab',
 'X0_ac',
 'X0_q',
 'X2_aa',
 'X2_ar',
 'X2_c',
 'X2_l',
 'X2_o',
 'X5_u',
 'y'}
```

```
[9]  1 y=x_dummies_1['y']
```

```
[10]  1 # align dataframes
      2 x_dummies_1, x_dummies_2= x_dummies_1.align(x_dummies_2, join='inner', axis=1)
```

```
[11]  1 print(x_dummies_1.shape)
      2 print(x_dummies_2.shape)
```

```
(4209, 554)
(4209, 554)
```

```
[12]  1 missing_cols = set( x_dummies_1.columns ) - set( x_dummies_2.columns )
```

```
[13]  1 missing_cols
```

```
set()
```

1: Dropping Unique values

```
[14]  1 # creating a list of columns which have only zeros
      2 zeros=[]
      3 for i,j in x_dummies_1.any().items():#https://pandas.pydata.org/pandas-docs/stabl
      4     if j==False:
      5         zeros.append(i)
```

```
[15]  1 zeros # 'X339' is missing in my code
      2 # we need to drop these columns
```

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

```
[16]  1 x_1_safe = x_dummies_1
      2 x_2_safe = x_dummies_2
```

```
[17]  1 x_dummies_1 = x_dummies_1.drop(zeros, axis=1)
      2 x_dummies_2 = x_dummies_2.drop(zeros, axis=1)
```

```
[18]  1 print(x_2_safe.shape)
      2 x_2_safe = x_2_safe.drop(zeros, axis=1)
```

```
print(x_2_safe.shape)
```

```
(4209, 554)
(4209, 542)
```

```
[19] 1 x_dummies_1['y']=y
```

2: Dropping Outliers

```
[20] 1 x_filtered= x_dummies_1[x_dummies_1['y']>70]#https://www.geeksforgeeks.org/drop-r
     2 x_filtered= x_filtered[x_filtered['y']<150]
     3 x_filtered.describe()
```

	ID	X10	X12	X13	X14	X15	
count	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	419
mean	4209.773724	0.013352	0.074392	0.057940	0.428231	0.000477	
std	2437.897217	0.114792	0.262439	0.233658	0.494881	0.021835	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2096.250000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4224.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	6316.750000	0.000000	0.000000	0.000000	1.000000	0.000000	
max	8417.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 543 columns

```
[21] 1 print(type(x_filtered))
     2 print(x_filtered.shape)
     3 print(type(x_dummies_2))
     4 print(x_dummies_2.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
(4194, 543)
<class 'pandas.core.frame.DataFrame'>
(4209, 542)
```

```
[22] 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, 543)
(4194, 542) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
[23]  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
```

4: Removing multicollinear features using VIF

```
[24]  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(ra
      8
      9     return (vif)
     10
```

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

```
100% 542/542 [47:10<00:00, 5.22s/it]
```

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

	variables	VIF
0	ID	486.868009
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	1.077202
305	X332	1.101729
31	X42	1.125967
267	X288	1.128819
91	X104	1.170178

```
[ ]  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]
```

```
[1.0772021280618025,
 1.1017286619006261,
 1.1259672881764773,
 1.128819140736738,
 1.1701777958231483]
```

```
[ ]  1 vif_dict = dict(zip(lst_variables, lst_vif))
```

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

```
float
```

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

```
False
```

```
[ ]  1 inf_indices=[]
     2 count=0
     3 for k,v in vif_dict.items():
     4     if np.isinf(v):
     5         inf_indices.append(k)
     6         count+=1
     7
```

```
[ ]  1 count
```

```
440
```

```
[ ]  1 len(inf_indices)
```

```
440
```

```python
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
```

```
[ ]  1 inf_indices
```

```
['X113',
 'X1_a',
 'X0_z',
 'X0_y',
 'X0_x',
 'X0_w',
 'X0_v',
 'X0_u',
 'X0_t',
 'X0_s',
 'X0_r',
 'X0_o',
 'X0_n',
 'X0_m',
 'X0_l',
 'X0_k',
 'X0_j',
 'X0_i',
 'X0_h',
 'X0_g',
 'X0_f',
 'X0_e',
 'X0_d',
 'X1_aa',
 'X1_ab',
 'X1_b',
 'X1_c',
 'X2_ac',
 'X2_a',
 'X1_z',
 'X1_y',
 'X1_w',
 'X1_v',
 'X1_u',
 'X1_t',
 'X1_s',
 'X1_r',
 'X1_q',
 'X0_c',
 'X1_p',
 'X1_n',
 'X1_m',
 'X1_l',
 'X1_k',
 'X1_j',
 'X1_i',
 'X1_h',
 'X1_g',
 'X1_f',
 'X1_e',
 'X1_d',
 'X1_o',
 'X0_bc',
```

```
'X0_ba',
'X0_b',
'X374',
'X373',
'X372',
'X371',
```

```
[ ]  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_compl
     3     f.write(json.dumps(inf_indices))
```

```
[25] 1 with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices_compl
     2     checc=json.loads(f.read())
     3
```

```
[26] 1 inf_indices=checc
     2 inf_indices
```

```
['X365',
 'X173',
 'X0_ad',
 'X66',
 'X2_am',
 'X8_f',
 'X167',
 'X5_i',
 'X5_r',
 'X2_r',
 'X316',
 'X341',
 'X5_j',
 'X109',
 'X280',
 'X101',
 'X0_a',
 'X113',
 'X5_y',
 'X323',
 'X135',
 'X5_af',
 'X97',
 'X110',
 'X238',
 'X2_ap',
 'X2_b',
 'X1_b',
 'X6_e',
 'X3_b',
 'X164',
 'X115',
 'X142',
 'X36',
 'X208'
```

```
'X31',
'X5_x',
'X0_ax',
'X120',
'X1_p',
'X145',
'X2_av',
'X1_y',
'X183',
'X80',
'X99',
'X35',
'X2_au',
'X5_ab',
'X308',
'X375',
'X2_m',
'X8_p',
'X122',
'X0_ba',
'X342',
'X2_p',
'X5_n',
'X0_h',
```

[27]
```
print(x_filtered.shape)
x_filtered = x_filtered.drop(inf_indices, axis=1)
x_dummies_2= x_dummies_2.drop(inf_indices, axis=1)
print(x_filtered.shape)
print(x_dummies_2.shape)
```

```
(4194, 542)
(4194, 121)
(4209, 121)
```

4: Adding new features using dimensionality reduction techniques

[28]
```
from sklearn.decomposition import TruncatedSVD
tsvd= TruncatedSVD(n_components=2, random_state=42)
tsvd_train= tsvd.fit_transform(x_filtered)
tsvd_test=  tsvd.transform(x_dummies_2)
```

[29]
```
tsvd_train.shape
```

```
(4194, 2)
```

[30]
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2, random_state=42)
pca_train= pca.fit_transform(x_filtered)
pca_test= pca.transform(x_dummies_2)
```

[31]
```
pca_train.shape
```

```
(4194, 2)
```

[32]
```
from sklearn.decomposition import FastICA
ica=FastICA(n_components=2, random_state=42)
ica_train= ica.fit_transform(x_filtered)
ica_test= ica.transform(x_dummies_2)
```

[33]
```
ica_train.shape
```

```
(4194, 2)
```

[34]
```
for i in range(0, tsvd_train.shape[1]):
    x_filtered['tsvd_'+str(i)]= tsvd_train[:, i]
    x_dummies_2['tsvd_'+str(i)]= tsvd_test[:, i]
    x_filtered['pca_'+str(i)]= pca_train[:, i]
    x_dummies_2['pca_'+str(i)]= pca_test[:, i]
    x_filtered['ica_'+str(i)]= ica_train[:, i]
    x_dummies_2['ica_'+str(i)]= ica_test[:, i]
```

[35]
```
print(x_filtered.shape, x_dummies_2.shape)
```

```
(4194, 127) (4209, 127)
```

5: Adding new features using the top important features

[36]
```
x_filtered['X64 + X218']=x_filtered['X64']+x_filtered['X218']
x_dummies_2['X64 + X218']=x_dummies_2['X64']+x_dummies_2['X218']

x_filtered['X218 + X224 + X273']=x_filtered['X218']+x_filtered['X224'] + x_filter
x_dummies_2['X218 + X224 + X273']=x_dummies_2['X218']+x_dummies_2['X224'] + x_dum

x_filtered['X64 + X224 + X273']=x_filtered['X64']+x_filtered['X224'] + x_filtered
x_dummies_2['X64 + X224 + X273']=x_dummies_2['X64']+x_dummies_2['X224'] + x_dummi


```

[37]
```
x_filtered['X314 + X315']=x_filtered['X314']+x_filtered['X315']
x_dummies_2['X314 + X315']=x_dummies_2['X314']+x_dummies_2['X315']

x_filtered['X314 + X315 + X29']=x_filtered['X314']+x_filtered['X315'] + x_filtere
x_dummies_2['X314 + X315 + X29']=x_dummies_2['X314']+x_dummies_2['X315'] + x_dumm

x_filtered['X314 + X315 + X118']=x_filtered['X314']+x_filtered['X315'] + x_filter
x_dummies_2['X314 + X315 + X118']=x_dummies_2['X314']+x_dummies_2['X315'] + x_dum

x_filtered['X127 + X189']=x_filtered['X127']+x_filtered['X189']
```

```
x_dummies_2['X127 + X189']=x_dummies_2['X127']+x_dummies_2['X189']

x_filtered['X118 + X54']=x_filtered['X118']+x_filtered['X54']
x_dummies_2['X118 + X54']=x_dummies_2['X118']+x_dummies_2['X54']

```

: ==========

6: RandomForestRegressor

[38]
```
from sklearn.ensemble import RandomForestRegressor
# n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf, boot
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in a Random forest
n_estimators=[int(x) for x in np.linspace (start=100, stop=1000, num=10)]

# Number of features to consider at every split
max_features=['auto', 'sqrt']

# Maximun no of levels in a tree
max_depth=[int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)

# minimum number of samples required to split a node
min_samples_split = np.arange(50,250,20)

# Minimum number of samples required at each leaf node
min_samples_leaf = np.arange(5,50,5)
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000], 'max_features':
```

[ ]
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
```

```
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,
# Fit the random search model
rf_random.fit(x_filtered, y)
```

```
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:  3.0min
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed: 14.2min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 28.1min 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,
                                             'min_samples_split': array([ 50,  70,  90, 11(
                                             '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)
```

```
[ ]  rf_random.best_params_
```

```
{'bootstrap': True,
 'max_depth': 70,
 'max_features': 'auto',
 'min_samples_leaf': 40,
 'min_samples_split': 110,
 'n_estimators': 500}
```

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

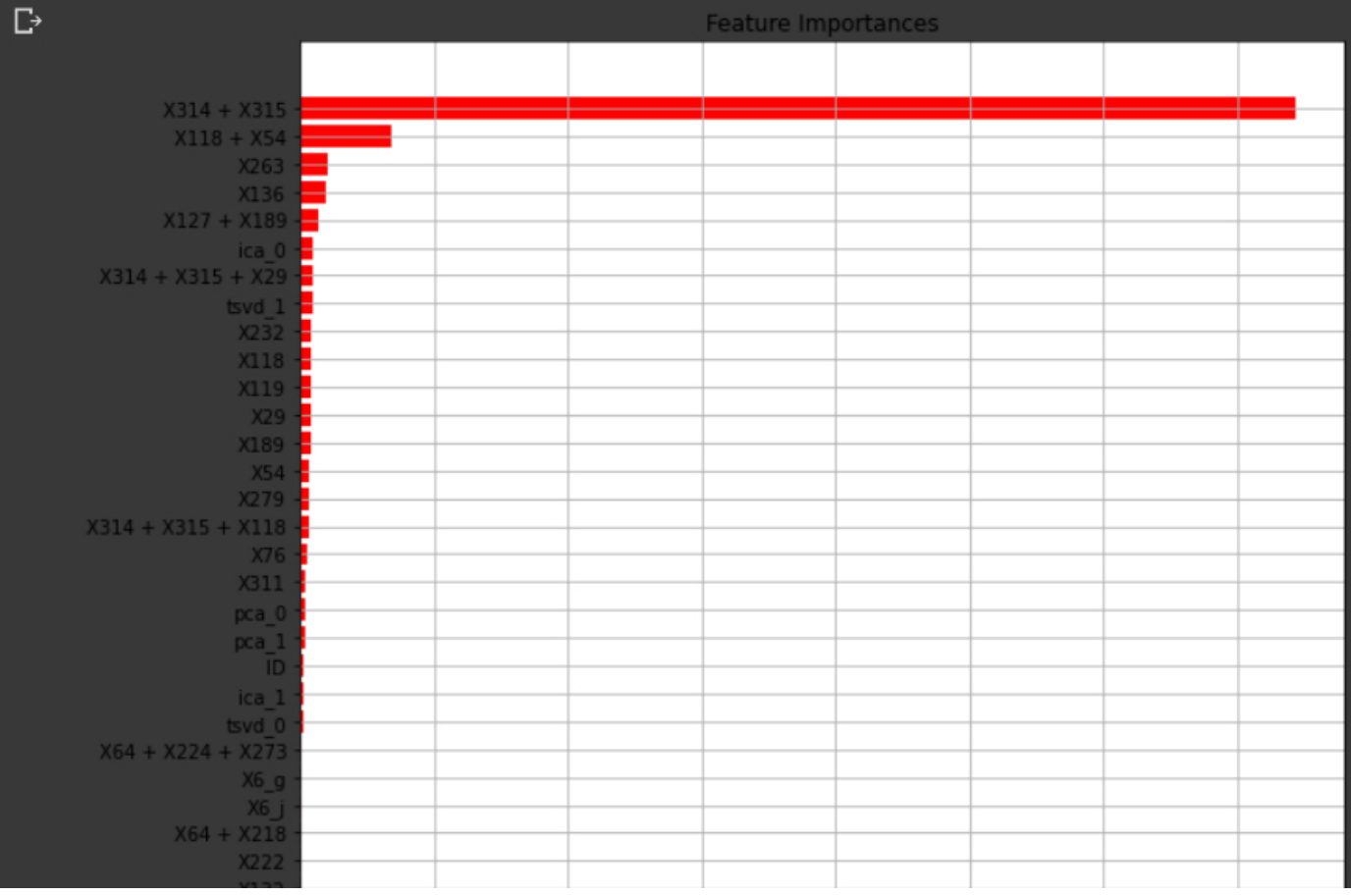
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=70, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=40,
                      min_samples_split=110, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, oob_score=False,
```

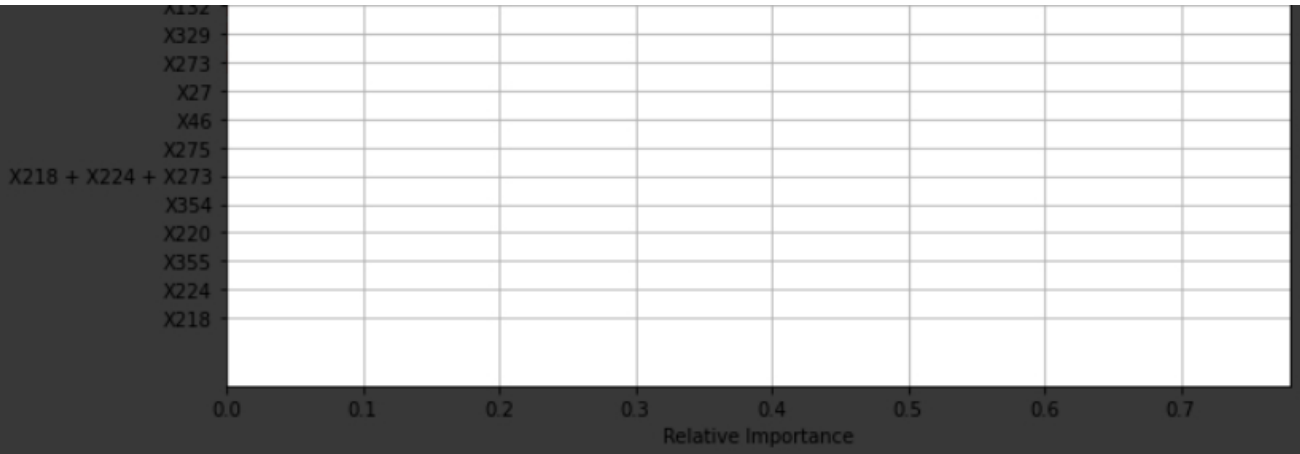
```
random_state=None, verbose=0, warm_start=False)
```

```
[ ]  1 #temp
     2 best_rf= RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
     3                       max_depth=70, max_features='auto', max_leaf_nodes=None,
     4                       max_samples=None, min_impurity_decrease=0.0,
     5                       min_impurity_split=None, min_samples_leaf=40,
     6                       min_samples_split=110, min_weight_fraction_leaf=0.0,
     7                       n_estimators=500, 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_filtered, y)
     3 y_train_pred=best_rf.predict(x_filtered)
     4 y_test_pred=best_rf.predict(x_dummies_2)
```

```
[ ]  1 features = x_filtered.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()
```






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

```
train r2_score 0.6535470835738704
==============================
```

```
[ ]  1 final_df= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID',
```

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

	ID	y
0	1	78.434875
1	2	94.254870
2	3	78.439658
3	4	78.434875
4	5	113.514419

```
[ ]  1 final_df.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/RandomFore
```

7: XGBoostRegressor

```
[ ]  1 import xgboost as xgb
```

```
[ ]  1 xgb_model=xgb.XGBRegressor()
     2 #xgb_model.fit(x_train, y_train)
     3 #y_pred=xgb_model.predict(x_train)
     4 learning_rate=[0.001, 0.01, 0.1, 0.2, 0.3]
     5 n_estimators=[50, 100, 150, 200, 250, 300]
```

```
6 hyperparameters= dict(learning_rate=learning_rate, n_estimators=n_estimators)
7
```

```
[ ]  1 from sklearn.model_selection import RandomizedSearchCV
     2 rsearch = RandomizedSearchCV(xgb_model, hyperparameters, n_iter=10, random_state=
     3 rsearch.fit(x_filtered, y)
     4 print(rsearch.best_params_)
```

```
[09:54:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:54:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:54:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:54:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:26] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:40] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:42] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:49] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:55:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:09] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:12] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:16] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:17] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:19] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[09:56:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
```

```
[09:56:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
{'n_estimators': 300, 'learning_rate': 0.01}
```

```
[ ]  1 # custom implementation of finding optimal hyperparameters has been deleted
```

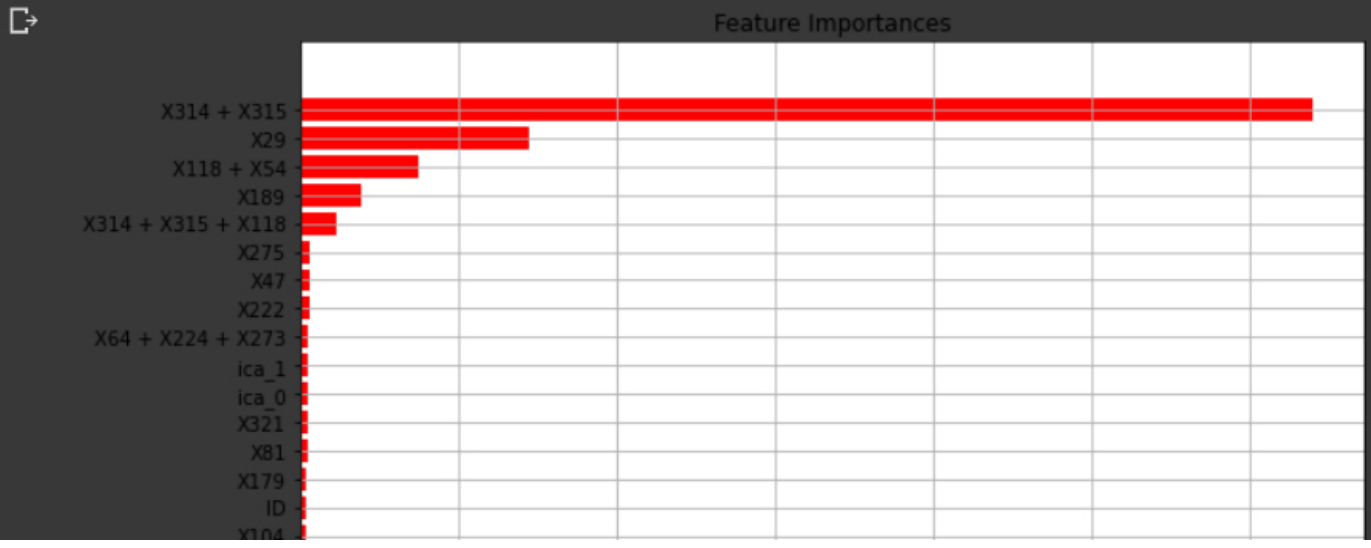
```
[ ]  1 best_xgb=rsearch.best_estimator_
     2 best_xgb
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.01, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=300,
             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_filtered, y)
     3 y_train_pred=best_xgb.predict(x_filtered)
     4 y_test_pred=best_xgb.predict(x_dummies_2)
```

```
[10:01:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
```

```
[ ]  1 import matplotlib.pyplot as plt
     2 features = x_filtered.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, y_train_pred))
     4 print('='*30)
     5 #print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.4664656045758411
==============================
```

```
[ ]  1 final_df_1= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
[ ]  1 final_df_1.head()
```

	ID	y
0	1	77.488136
1	2	93.120033
2	3	75.105789
3	4	75.105789
4	5	106.988533

```
[ ]  1 final_df_1.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/XGBoostR
```

8: 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=4
     3 rsearch.fit(x_filtered, y)
     4 print(rsearch.best_params_)
```

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

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

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

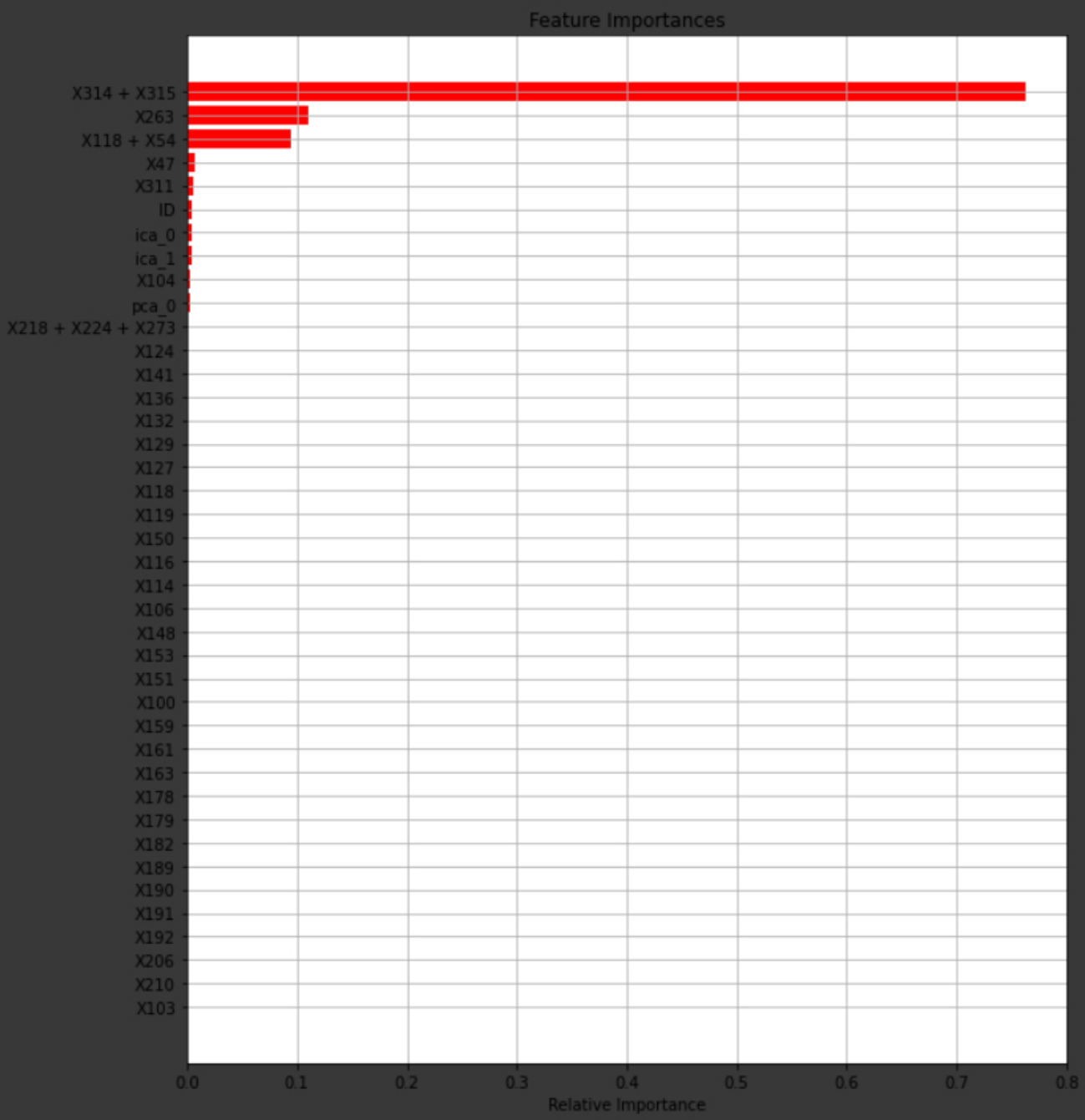
```
[ ]  1 best_dt=rsearch.best_estimator_
     2 best_dt
```

```
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_filtered, y)
     3 y_train_pred=best_dt.predict(x_filtered)
     4 y_test_pred=best_dt.predict(x_dummies_2)
```

```
[ ]  1 import matplotlib.pyplot as plt
     2 features = x_filtered.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, y_train_pred))
     4 print('='*30)
     5 #print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.641140515826669
==============================
```

```
[ ]  1 final_df_2= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
[ ]  1 final_df_2.head()
```

	ID	y
0	1	77.964862
1	2	130.810000
2	3	77.964862
3	4	77.964862
4	5	112.866875

```
[ ]  1 final_df_2.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/Decision
```

9: 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'
     2 gsearch = GridSearchCV(linear_regressor, parameters, scoring='r2')
     3 gsearch.fit(x_filtered, y)
```

```
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=0.001, average=False, early_stopping=False, epsilon=10,
             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='l1', power_t=0.25, random_state=None,
             shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
             warm_start=False)
-1.0219373944520339e+25
```

```
[ ]  1 linear_regressor=SGDRegressor(alpha=0.001, average=False, early_stopping=False, e
     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='l1', power_t=0.25, random_state=None,
     5              shuffle=True, tol=0.001, validation_fraction=0.1, verbose=0,
     6              warm_start=False)
```

```
[ ]  1 r_square= cross_val_score(linear_regressor, x_filtered, y, scoring='r2', cv=5)
     2 mean_r_square=np.mean(r_square)
     3 print(mean_r_square)
```

```
-8.044774132722213e+25
```

```
[ ]  1 linear_regressor.fit(x_filtered, y)
     2 y_train_pred=linear_regressor.predict(x_filtered)
     3 y_test_pred =linear_regressor.predict(x_dummies_2)
```

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

```
train r2_score -2.1440892848120442e+24
==============================
```

```
[ ]  1 final_df_3= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
[ ]  1 final_df_3.head()
```

	ID	y
0	1	3.281617e+13
1	2	3.281565e+13

2	3	3.283771e+13
3	4	3.280254e+13
4	5	3.287232e+13

```
[ ]  1 final_df_3.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/SimpleLi
```

10: 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_filtered, y)
```

```
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': 20}
0.6018030809140671
```

```
[ ]  1 ridge_regressor= Ridge(alpha=20)
     2 ridge_regressor
```

```
Ridge(alpha=20, copy_X=True, fit_intercept=True, max_iter=None, normalize=False,
      random_state=None, solver='auto', tol=0.001)
```

```
[ ]  1 ridge_regressor.fit(x_filtered, y)
     2 y_train_pred=ridge_regressor.predict(x_filtered)
     3 y_test_pred =ridge_regressor.predict(x_dummies_2)
```

```
[ ]  1 # checking r2 score
     2 from sklearn.metrics import r2_score
     3 print('train r2 score', r2_score(y, y_train_pred))
```

```
                  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}
0.6031800961153089
```

```
[ ]  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-pyth
     2 #https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-compl
     3 #https://alfurka.github.io/2018-11-18-grid-search/
     4 lasso_regressor.fit(x_filtered, y)
     5 y_train_pred=lasso_regressor.predict(x_filtered)
     6 y_test_pred =lasso_regressor.predict(x_dummies_2)
```

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

```
train r2_score 0.6234275817168167
==============================
```

```
[ ]  1 final_df_5= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
[ ]  1 final_df_5.head()
```

	ID	y
0	1	79.534046
1	2	96.370153
2	3	80.228208
3	4	77.973903
4	5	111.042708

```
[ ]  1 final_df_5.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/LassoReg
```

12: 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':
     2 gsearch = GridSearchCV(svr, parameters, scoring='r2')
     3 gsearch.fit(x_filtered, y)
```

```
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=10, cache_size=200, coef0=0.0, degree=3, epsilon=10, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
-0.0017361059621410879
```

```
[ ]  1 svr= SVR(C=10, cache_size=200, coef0=0.0, degree=3, epsilon=10, gamma='scale',
     2     kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
     3 svr
```

```
SVR(C=10, cache_size=200, coef0=0.0, degree=3, epsilon=10, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

```
[ ]  1 svr.fit(x_filtered, y)
     2 y_train_pred=svr.predict(x_filtered)
     3 y_test_pred =svr.predict(x_dummies_2)
```

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

```
#print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.0034891052806770295
==============================
```

```
final_df_6= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
final_df_6.head()
```

	ID	y
0	1	100.330852
1	2	100.331628
2	3	100.332691
3	4	100.332910
4	5	100.335622

```
final_df_6.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/SupportV
```

```
1
```

13: AdaBoostRegressor

```
from sklearn.ensemble import AdaBoostRegressor
```

```
abr= AdaBoostRegressor()
abr
```

```
AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',
                  n_estimators=50, random_state=None)
```

```
parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,
gsearch = GridSearchCV(abr, parameters, scoring='r2')
gsearch.fit(x_filtered, y)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=AdaBoostRegressor(base_estimator=None, learning_rate=1.0,
                                         loss='linear', n_estimators=50,
                                         random_state=None),
             iid='deprecated', n_jobs=None,
             param_grid={'learning_rate': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3,
                                           0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
                         'loss': ['linear', 'square', 'exponential'],
                         'n_estimators': [50, 100, 150, 200, 250, 300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
```

```
                  scoring='r2', verbose=0)
```

[42]
```
print(gsearch.best_estimator_)
print(gsearch.best_score_)
```

```
AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='linear',
                  n_estimators=50, random_state=None)
0.6172666188726387
```

[43]
```
abr= AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='linear',
                       n_estimators=50, random_state=None)
abr
```

```
AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='linear',
                  n_estimators=50, random_state=None)
```

[44]
```
abr.fit(x_filtered, y)
y_train_pred=abr.predict(x_filtered)
y_test_pred =abr.predict(x_dummies_2)
```

[45]
```
# checking r2 score
from sklearn.metrics import r2_score
print('train r2_score', r2_score(y, y_train_pred))
print('='*30)
#print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score 0.6288566000103565
==============================
```

[46]
```
final_df_10= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID
```

[47]
```
final_df_10.head()
```

	ID	y
0	1	78.156871
1	2	94.004008
2	3	77.824188
3	4	78.421518
4	5	112.538294

[48]
```
final_df_10.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/CSV fil
```

[ ]
```
1
```

14: Deep Neural Network

14: Deep Neural Network

```
[ ]
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from tensorflow.keras.utils import to_categorical
from numpy import array
import tensorflow as tf
from tensorflow.keras.layers import Dense,Input,Activation
from tensorflow.keras.models import Model
import random as rn
import logging
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import Callback
import keras.backend as k
import tensorflow
```

```
[ ]
import keras.backend as K
from sklearn.metrics import roc_auc_score
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import LearningRateScheduler
import keras
from keras.initializers import RandomNormal,RandomUniform
```

```
[ ]
# custom R2-score metrics for keras backend
#https://www.kaggle.com/c/mercedes-benz-greener-manufacturing/discussion/34019
from keras import backend as K

def r2_keras(y_true, y_pred):
    SS_res =  K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return ( 1 - SS_res/(SS_tot + K.epsilon()) )
```

```
[ ]
filepath="weights-{epoch:02d}-{val_r2_keras:.4f}.hdf5"
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_r2_keras',  verbose=
#https://stackoverflow.com/questions/48971221/keras-modelcheckpoint-monitor-multi
# issue was resolved by Benzion18's comment on oct 10 2019, link is pasted below
#https://github.com/tensorflow/tensorflow/issues/33163
```

```
[ ]
def changeLearningRate(epoch):
    initial_learningrate=0.1
    decay_rate=0.05
    if (1+epoch)%3==0:
        changed=initial_learningrate*decay_rate
```

```
    else:
        changed=initial_learningrate
    return changed
lrschedule=tf.keras.callbacks.LearningRateScheduler(changeLearningRate, verbose=1)
#https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LearningRateSchedu
#https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau
##changing Learning rate on plateau
reduce_lr=tf.keras.callbacks.ReduceLROnPlateau(monitor='val_r2_keras', factor=0.1
```

```
#https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-
#from reference notebook
from tensorflow.keras.callbacks import EarlyStopping
earlystop = EarlyStopping(monitor='val_r2_keras', min_delta=0,mode='auto', patien

```

```
x_filtered_= x_dummies_1[x_dummies_1 ['y']>70]#https://www.geeksforgeeks.org/drop
x_filtered_= x_filtered_[x_filtered_['y']<150]
x_filtered_.describe()
```

	ID	X10	X12	X13	X14	X15	
count	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	4194.000000	419
mean	4209.773724	0.013352	0.074392	0.057940	0.428231	0.000477	
std	2437.897217	0.114792	0.262439	0.233658	0.494881	0.021835	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2096.250000	0.000000	0.000000	0.000000	0.000000	0.000000	
50%	4224.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
75%	6316.750000	0.000000	0.000000	0.000000	1.000000	0.000000	
max	8417.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 543 columns

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

```
(4194, 543)
(4194, 542) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x_filtered_, y_, test_size=0.
```

```
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.
(3355, 542) (3355,) (839, 542) (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, 542) (3355, 1) (839, 542) (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_fr
```

```
1 # input layer
2 input_layer=Input(shape=(X_train.shape[1],))
3 # dense hidden layers                                                52.12 noteb
4 layer1=Dense(512,activation='relu',kernel_initializer='normal')(input_layer)
5 layer2=Dense(512,activation='relu',kernel_initializer='normal')(layer1)
6 layer3=Dense(512,activation='relu',kernel_initializer='normal')(layer2)
7 layer4=Dense(512,activation='relu',kernel_initializer='normal')(layer3)
8 layer5=Dense(512,activation='relu',kernel_initializer='normal')(layer4)
```

```
# output layer
output=Dense(1)(layer5)

#creating a model
model=Model(inputs=input_layer, outputs=output)

#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,
              loss='mse', metrics=[r2_keras])
model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_siz
```

```
Epoch 1/5
14/14 [==============================] - ETA: 0s - loss: 82891.2734 - r2_keras: -557.
Epoch 00001: val_r2_keras improved from -72.92318 to -20.92361, saving model to weight
14/14 [==============================] - 1s 63ms/step - loss: 82891.2734 - r2_keras:
Epoch 2/5
14/14 [==============================] - ETA: 0s - loss: 2870.8547 - r2_keras: -19.53
Epoch 00002: val_r2_keras did not improve from -20.92361
14/14 [==============================] - 1s 51ms/step - loss: 2870.8547 - r2_keras: -
Epoch 3/5
13/14 [==========================>...] - ETA: 0s - loss: 2720.1697 - r2_keras: -18.28
Epoch 00003: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00003: val_r2_keras improved from -20.92361 to -20.56848, saving model to weight
14/14 [==============================] - 1s 53ms/step - loss: 2716.7681 - r2_keras: -
Epoch 00003: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f984c7dfba8>
```

```
print(x_filtered_.shape, x_2_safe.shape)
```

```
(4194, 542) (4209, 542)
```

```
y_train_pred=model.predict(x_filtered_)
y_test_pred =model.predict(x_2_safe)
```

```
print('score on train data : ', model.evaluate(X_train, Y_train, verbose=1))
print('score on test data : ', model.evaluate(X_test, Y_test, verbose=1))
```

```
105/105 [==============================] - 0s 4ms/step - loss: 2620.8586 - r2_keras:
score on train data :  [2620.858642578125, -19.377845764160156]
27/27 [==============================] - 0s 4ms/step - loss: 2930.7612 - r2_keras: -2
score on test data :  [2930.76123046875, -22.221073150634766]
```

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

```
#print('test r2_score', r2_score(y_test, y_test_pred))
```

```
train r2_score -17.651680332663435
==============================
```

```
[ ]  1 final_df_7= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

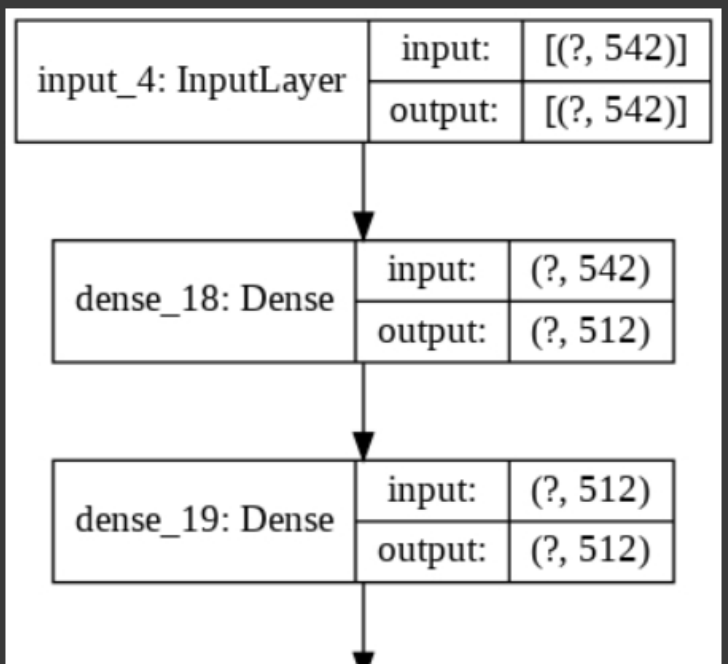
```
[ ]  1 final_df_7.head()
```

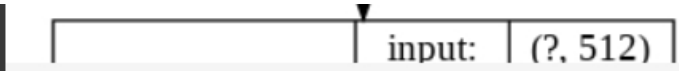
	ID	y
0	1	[56.555984]
1	2	[65.70468]
2	3	[57.262955]
3	4	[56.927643]
4	5	[64.46127]

```
[ ]  1 final_df_7.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/CSV file
```

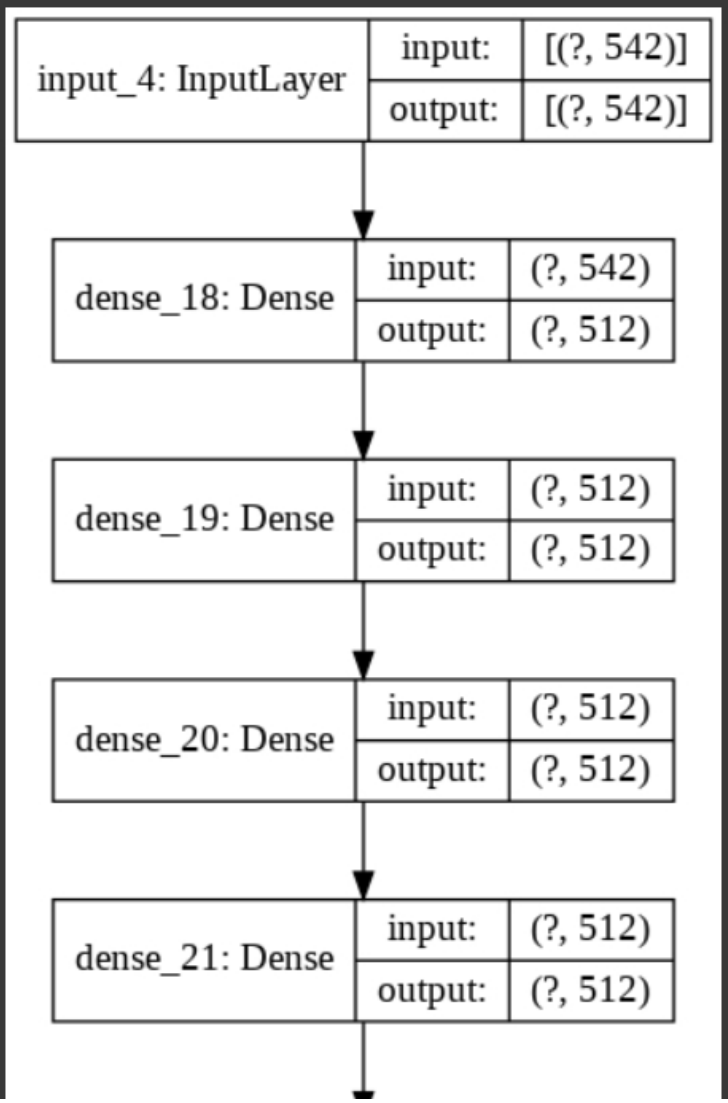
```
[ ]  1 %load_ext tensorboard
     2 import datetime, os
     3 %tensorboard --logdir logs1
     4 ##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf06
```

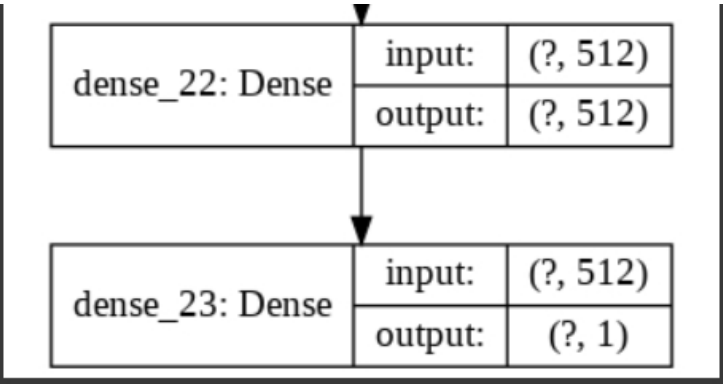
```
[ ]  1 tf.keras.utils.plot_model(
     2     model, to_file='model_1.png', show_shapes=True, show_layer_names=True,
     3     rankdir='TB', expand_nested=False, dpi=96
     4 )
```






```
[ ]  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 )
```





15: CNN

```
1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs2', histogram_fr
```

```
[ ]
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, L
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_
9  drop_a=Dropout(0.1)(conv1_1)
10 conv1_2= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv
11 drop_b=Dropout(0.001)(conv1_2)
12 conv1_3= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv
13 flatten_c1= Flatten(data_format='channels_last',name='Flatten_c1')(conv1_3)
14
15 # adding the dense layers and dropout layers
16 dense_1_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', na
17 dropout_1_ac = Dropout(0.12)(dense_1_ac)
18 dense_2_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', na
19 dropout_2_ac = Dropout(0.11)(dense_2_ac)
20 dense_3_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', na
21 dropout_3_ac = Dropout(0.1)(dense_3_ac)
22 output = Dense(units=1, kernel_initializer='he_normal', name='output')(dropout_3_a
23
24 model= Model(inputs=input, outputs=output)
```

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

```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 542, 1)]          0
_________________________________________________________________
conv1_1 (Conv1D)             (None, 540, 64)           256
```

```
dropout (Dropout)            (None, 540, 64)           0
_________________________________________________________________
conv1_2 (Conv1D)             (None, 538, 64)           12352
_________________________________________________________________
dropout_1 (Dropout)          (None, 538, 64)           0
_________________________________________________________________
conv1_3 (Conv1D)             (None, 536, 64)           12352
_________________________________________________________________
Flatten_c1 (Flatten)         (None, 34304)             0
_________________________________________________________________
dense_1_ac (Dense)           (None, 64)                2195520
_________________________________________________________________
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: 2,228,865
Trainable params: 2,228,865
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,
              loss='mse', metrics=[r2_keras])
validation_data = (X_test, Y_test)
```

```
[ ]
model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_siz
```

```
Epoch 1/5
14/14 [==============================] - ETA: 0s - loss: 4612.5308 - r2_keras: -31.18
Epoch 00001: val_r2_keras improved from -20.56848 to -7.06791, saving model to weight
14/14 [==============================] - 11s 760ms/step - loss: 4612.5308 - r2_keras:
Epoch 2/5
14/14 [==============================] - ETA: 0s - loss: 1131.5754 - r2_keras: -6.897
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras improved from -7.06791 to -1.45880, saving model to weights
14/14 [==============================] - 11s 752ms/step - loss: 1131.5754 - r2_keras:
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f984a748f98>
```

```
[ ]
print(x_filtered_.shape, x_2_safe.shape)
```

```
(4194, 542) (4209, 542)
```

```
[ ]  1 y_train_pred=model.predict(x_filtered_)
     2 y_test_pred =model.predict(x_2_safe)
```

```
[ ]  1 print('score on train data : ', model.evaluate(X_train, Y_train, verbose=1))
     2 print('score on test data : ', model.evaluate(X_test, Y_test, verbose=1))
```

```
105/105 [==============================] - 2s 22ms/step - loss: 383.4030 - r2_keras: -
score on train data :  [383.40301513671875, -1.9262481927871704]
27/27 [==============================] - 1s 22ms/step - loss: 355.8225 - r2_keras: -1.
score on test data :  [355.8224792480469, -1.6556975841522217]
```

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

```
train r2_score -1.6271283295147483
==============================
```

```
[ ]  1 final_df_8= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'.
```

```
[ ]  1 final_df_8.head()
```

	ID	y
0	1	[82.27903]
1	2	[107.11044]
2	3	[93.87634]
3	4	[88.21289]
4	5	[119.45129]

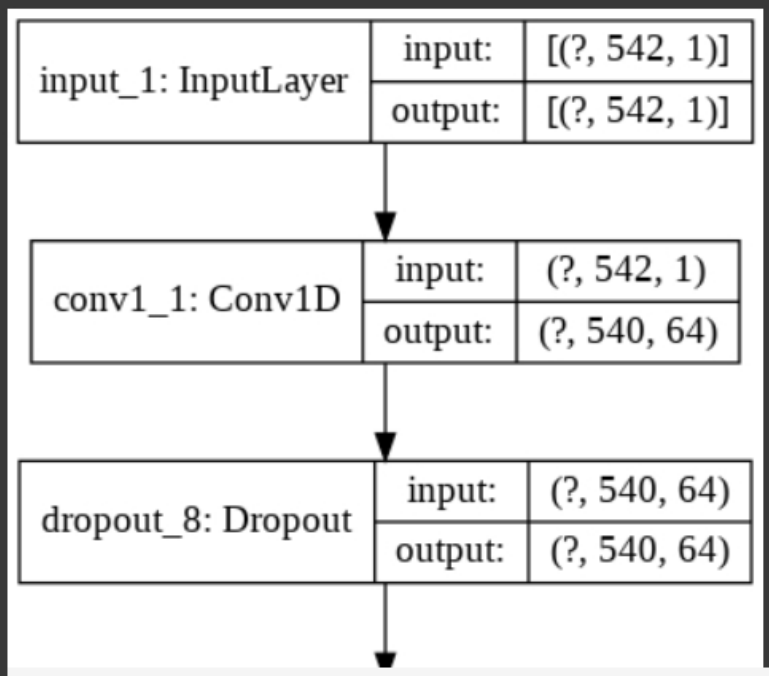
```
[ ]  1 final_df_8.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/CSV file
```

```
[ ]  1 %load_ext tensorboard
     2 import datetime, os
     3 %tensorboard --logdir logs2
     4 ##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf06
```

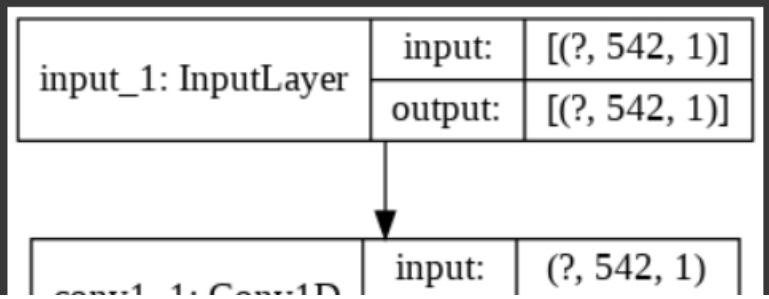
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

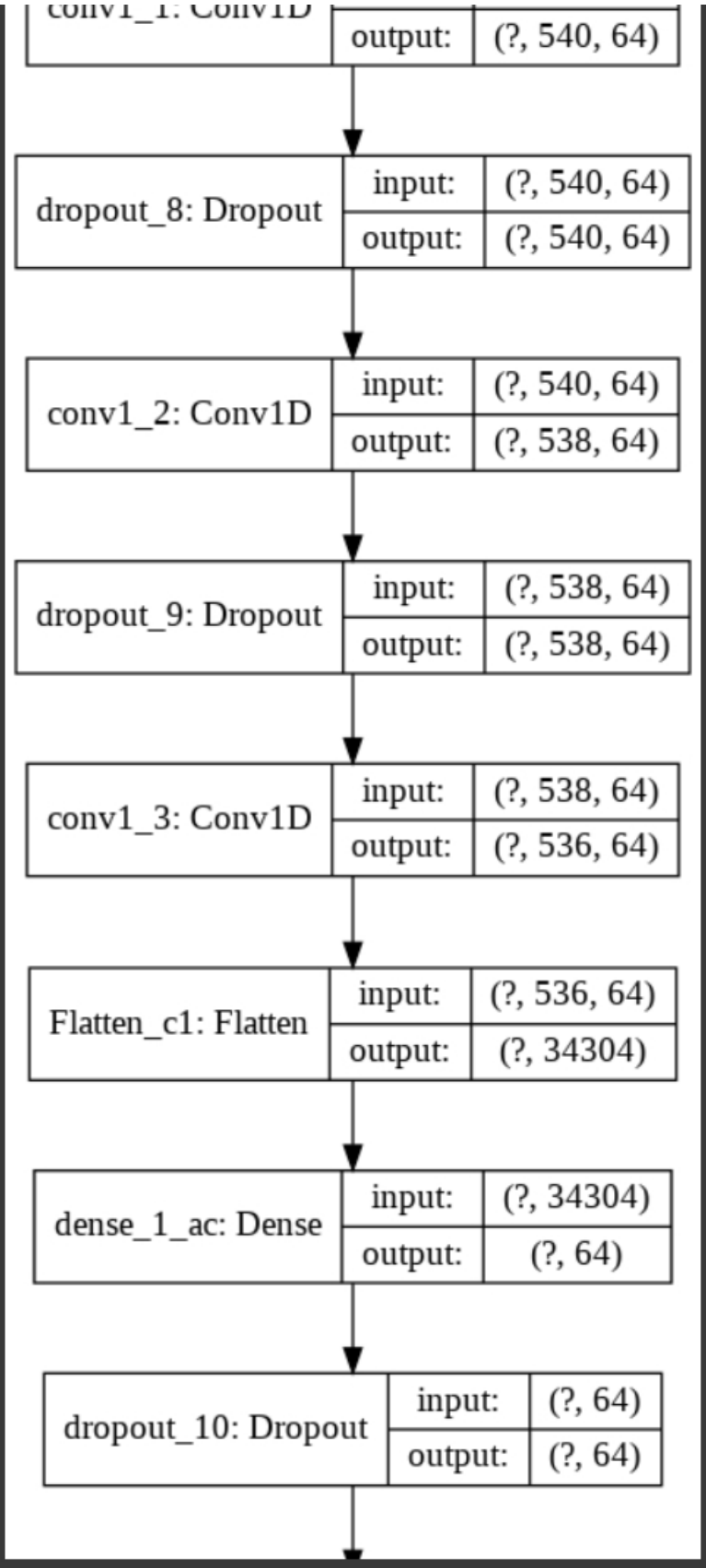
```
[ ]  1 tf.keras.utils.plot_model(
     2     model, to file='model 2.png', show shapes=True, show layer names=True,
```

```
    rankdir='TB', expand_nested=False, dpi=96
)
```



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





16: LSTM

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

```
[ ]  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', na
     6 dropout_1_ac = Dropout(0.6)(dense_1_ac)
     7 dense_2_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', na
     8 dropout_2_ac = Dropout(0.6)(dense_2_ac)
     9 dense_3_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', na
    10 dropout_3_ac = Dropout(0.6)(dense_3_ac)
    11 output_1 = Dense(units=1, kernel_initializer='he_normal', name='output')(dropout_
    12
    13 model= Model(inputs=input_1, outputs=output_1)
```

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

```
Model: "functional_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_lstm (InputLayer)      [(None, 542, 1)]          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                16896
_________________________________________________________________
flatten_1 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_1_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_9 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3_ac (Dense)           (None, 64)                4160
_________________________________________________________________
dropout_10 (Dropout)         (None, 64)                0
_________________________________________________________________
output (Dense)               (None, 1)                 65
=================================================================
Total params: 29,441
Trainable params: 29,441
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]  1 #creating a callback list
     2 callback_list=[tensorboard_callback, earlystop,reduce_lr,checkpoint]
     3
     4 #cross entropy as loss function
     5 model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9,
     6               loss='mse', metrics=[r2_keras])
     7 validation_data = (X_test, Y_test)
```

```
[ ]  1 model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_siz
```

```
Epoch 1/5
14/14 [==============================] - ETA: 0s - loss: 10224.9502 - r2_keras: -72.0(
Epoch 00001: val_r2_keras did not improve from -1.45880
14/14 [==============================] - 14s 983ms/step - loss: 10224.9502 - r2_keras:
Epoch 2/5
14/14 [==============================] - ETA: 0s - loss: 8241.1602 - r2_keras: -56.82
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras did not improve from -1.45880
14/14 [==============================] - 13s 921ms/step - loss: 8241.1602 - r2_keras:
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f9841e6d588>
```

```
[ ]  1 print(x_filtered_.shape, x_2_safe.shape)
```

```
(4194, 542) (4209, 542)
```

```
[ ]  1 y_train_pred=model.predict(x_filtered_)
     2 y_test_pred =model.predict(x_2_safe)
```

```
[ ]  1 print('score on train data : ', model.evaluate(X_train, Y_train, verbose=1))
     2 print('score on test data : ', model.evaluate(X_test, Y_test, verbose=1))
```

```
105/105 [==============================] - 6s 53ms/step - loss: 5898.3643 - r2_keras:
score on train data :  [5898.3642578125, -45.007808685302734]
27/27 [==============================] - 1s 51ms/step - loss: 5770.2280 - r2_keras: -4
score on test data :  [5770.22802734375, -43.762847900390625]
```

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

```
train r2_score -39.82827517405778
==============================
```

```
[ ]  1 final_df_9= pd.DataFrame(list(zip(df_2['ID'].values, y_test_pred)), columns=['ID'
```

```
[ ]  1 final_df_9.head()
```

	ID	y
0	1	[24.74544]
1	2	[24.896534]
2	3	[24.746834]

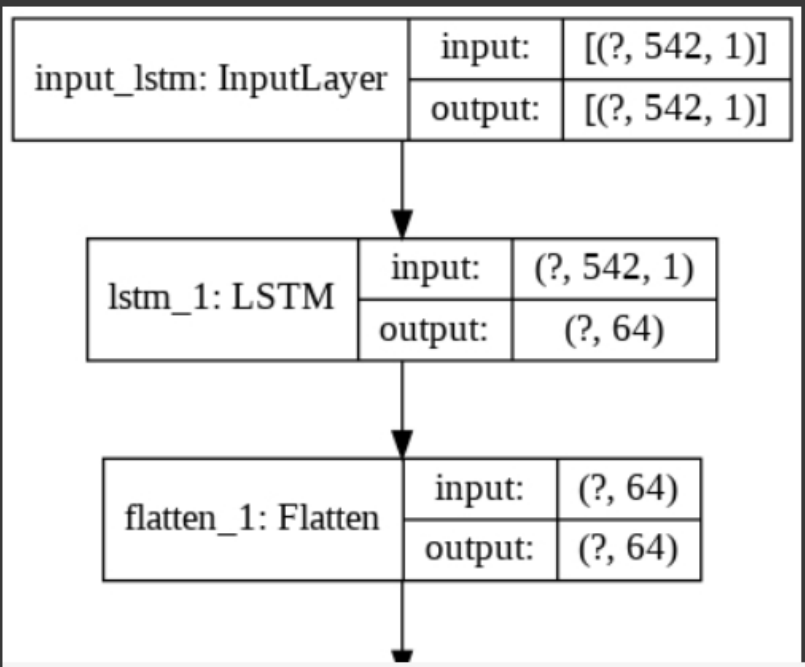
3	4	[24.746859]
4	5	[24.746862]

```
[ ]  1 final_df_9.to_csv(r'/content/drive/My Drive/Colab Notebooks/Case Study 1/CSV file
```

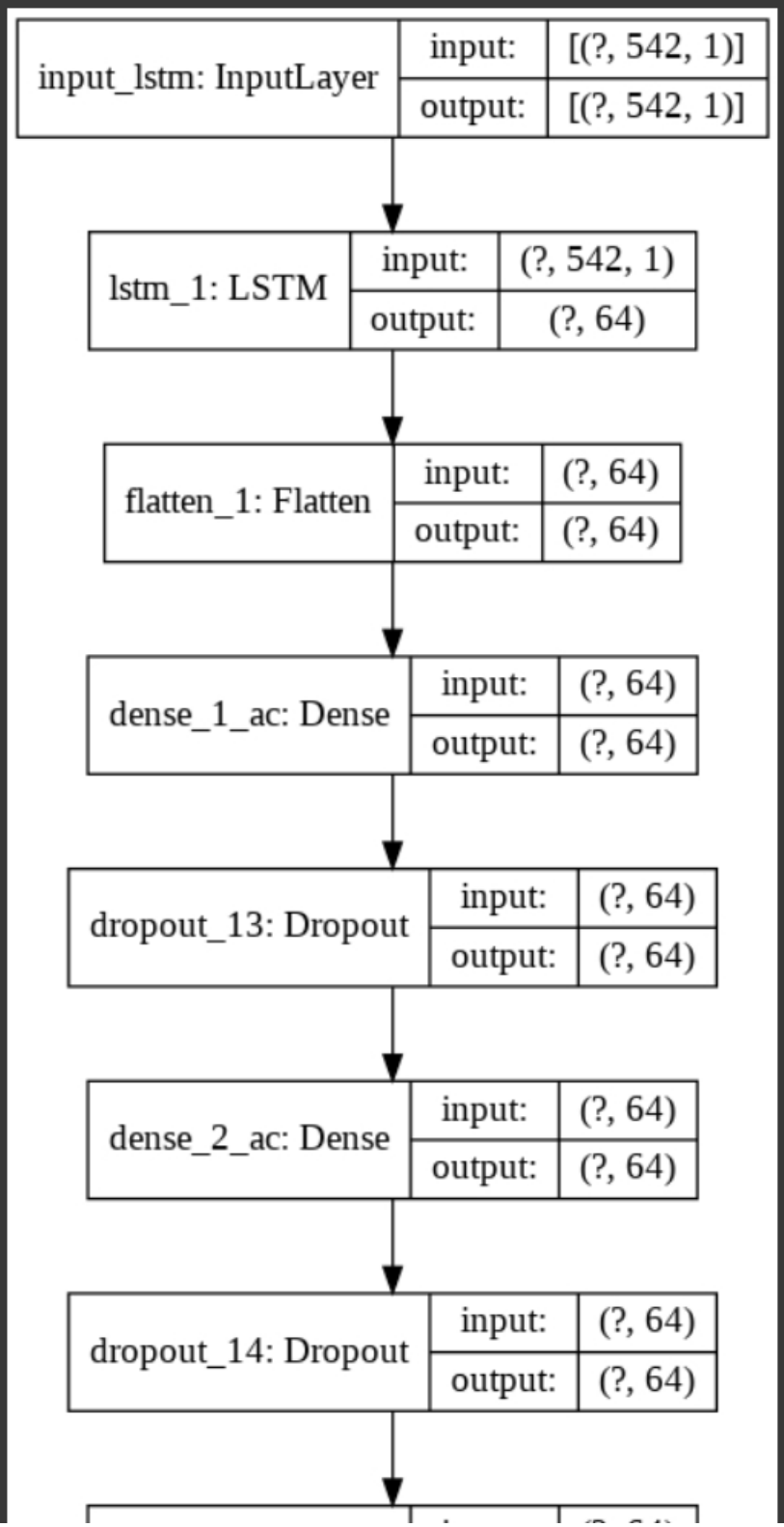
```
[ ]  1 %load_ext tensorboard
     2 import datetime, os
     3 %tensorboard --logdir logs3
     4 ##https://medium.com/@kuanhoong/how-to-use-tensorboard-with-google-colab-43f7cf06
```

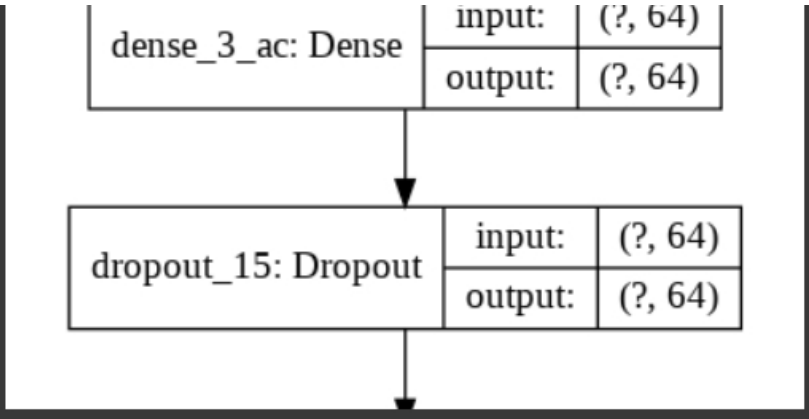
```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6008 (pid 2448), started 0:32:51 ago. (Use '!kill 2448' t
```

```
[ ]  1 tf.keras.utils.plot_model(
     2     model, to_file='model_3.png', show_shapes=True, show_layer_names=True,
     3     rankdir='TB', expand_nested=False, dpi=96
     4 )
```



```
[ ]  1 tf.keras.utils.plot_model(
     2     model, to_file='model_3.png', show_shapes=True, show_layer_names=True,
     3     rankdir='TB', expand_nested=False, dpi=96
     4 )
```





17: 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
```

```
[ ]  1
```