



+ Code + Text

✓ RAM Disk

Editing

```
[1] 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.

"""

```
[2] 1 from google.colab import drive
2 drive.mount('/content/drive',force_remount=True)
```

↳ Mounted at /content/drive

- Loading the DataSets

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

```
[4] 1 df_1.columns
```

```
⇒ Index(['ID', 'y', 'X0', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X8',
       ...,
       'X375', 'X376', 'X377', 'X378', 'X379', 'X380', 'X382', 'X383', 'X384',
       'X385'],
      dtype='object', length=378)
```

```
[5] 1 # converting categorical features to numerical features
2 x_dummies=pd.get_dummies(df_1, prefix_sep='_', drop_first=True)
3 x_dummies.head()
4 #https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.get_dummies.html
5 #https://towardsdatascience.com/encoding-categorical-features-21a2651a065c
```

	ID	y	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	X21	X22	X23	X24
0	0	130.81	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0
1	6	88.53	0	0	0	0	0	0	0	0	1	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
3	9	80.62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	13	78.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 557 columns

```
[6] 1 # creating a list of columns which have only zeros
2 zeros=[]
3 for i,j in x_dummies.any().items():#https://pandas.pydata.org/pandas-docs/stable/indexing.html#boolean-indexing
4     if j==False:
5         zeros.append(i)
```

```
[7] 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']
```

```
[8] 1 #https://stackoverflow.com/questions/29294983/how-to-calculate-correlation-between-pandas-dataframes
2 # to delete columns with high correlation
```

- Dropping columns with only zeros

```
[9] 1 x_dummies = x_dummies.drop(zeros, axis=1)
```

- Dropping outliers => considering only IQR

```
[10] 1 x_filtered= x_dummies[x_dummies ['y']>70]#https://www.geeksforgeeks.org/drop-rows-in-pandas-dataframe/
2 x_filtered= x_filtered[x_filtered['y']<150]
3 x_filtered.describe()
```

	ID	y	X10	X12	X13	X14
count	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
std	2437.897217	11.994753	0.114792	0.262439	0.233658	0.494881
min	0.000000	72.110000	0.000000	0.000000	0.000000	0.000000
25%	2096.250000	90.800000	0.000000	0.000000	0.000000	0.000000
50%	4224.000000	99.090000	0.000000	0.000000	0.000000	0.000000
75%	6316.750000	108.967500	0.000000	0.000000	0.000000	1.000000
max	8417.000000	149.630000	1.000000	1.000000	1.000000	1.000000

8 rows × 545 columns

```
[11] 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))
```

```
[12] (4194, 545)
(4194, 544) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
[12] 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)
```

```
[13] list
```

- Finding important features using VIF

```
[13] 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(ran
8
9     return (vif)
10
```

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

↳ 100%  544/544 [11:17<00:00, 1.24s/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']

E [1930 , 1932 , 1933 , 1938 , 1944]

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

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

```
[ ]    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

373

```
[ ]    1 len(inf indices)
```

→ 373

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

[] 1 inf_indices

↳ 'X373',
'X371',
'X370',
'X369',
'X368',
'X115',
'X117',
'X365',
'X364'

```
'X364',
'X363',
'X372',
'X183',
'X335',
'X333',
'X307',
'X306',
'X305',
'X304',
'X302',
'X143',
'X144',
'X299',
'X298',
'X296',
'X145',
'X146',
'X147',
'X152',
'X154',
'X287',
'X286',
'X285',
'X284',
'X308',
'X309',
'X310',
'X311',
'X133',
'X331',
'X134',
'X135',
'X136',
'X326',
'X137',
'X324',
'X323',
'X334',
'X322',
'X320',
'X139',
'X140',
'X317',
'X316',
'X315',
'X314',
'X313',
'X142',
'X138',
'X295']
```

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

```
[14] 1 with open('/content/drive/My Drive/Colab Notebooks/Case Study 1/inf_indices.txt',  
2       checc=json.loads(f.read())  
3
```

```
[15] 1 inf_indices=checc  
2 inf_indices
```

```
↳  'X126',  
  'X0_am',  
  'X0_bc',  
  'X52',  
  'X345',  
  'X0_b',  
  'X202',  
  'X2_ay',  
  'X237',  
  'X0_m',  
  'X125',  
  'X2_e',  
  'X165',  
  'X0_q',  
  'X0_ba',  
  'X360',  
  'X198',  
  'X295',  
  'X378',  
  'X200',  
  'X26',  
  'X31',  
  'X280',  
  'X217',  
  'X1_l',  
  'X227',  
  'X131',  
  'X2_ac',  
  'X115',  
  'X2_ak',  
  'X128',  
  'X374',  
  'X259',  
  'X2_t',  
  'X146',  
  'X0_k',  
  'X0_v',  
  'X254',  
  'X95',  
  'X1_i',  
  'X255',  
  'X1_p',  
  'X1_r',  
  'X364',  
  'X28',  
  'X281',  
  'X167',  
  'X0_f',  
  'X17',  
  'X251',
```

```
'X258',
'X287',
'X307',
'X2_an',
'X380',
'X172',
'X2_af',
'X334',
'X175']
```

```
[16] 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)
```

```
[21] 1 all_columns = list(x_filtered.columns)
2 all_columns
```

```
'X5_ac',
'X5_ad',
'X5_ae',
'X5_af',
'X5_ag',
'X5_ah',
'X5_c',
'X5_d',
'X5_f',
'X5_g',
'X5_i',
'X5_j',
'X5_k',
'X5_l',
'X5_m',
'X5_n',
'X5_o',
'X5_p',
'X5_q',
'X5_r',
'X5_s',
'X5_v',
'X5_w',
'X5_y',
'X6_b',
'X6_c',
'X6_d',
'X6_e',
'X6_f',
'X6_g',
'X6_h',
'X6_i',
'X6_j',
'X6_k',
'X6_l',
'X8_h'.
```

```
'x8_a',
'x8_c',
'x8_d',
'x8_e',
'x8_f',
'x8_g',
'x8_h',
'x8_i',
'x8_j',
'x8_k',
'x8_l',
'x8_m',
'x8_n',
'x8_o',
'x8_p',
'x8_q',
'x8_r',
'x8_s',
'x8_t',
'x8_u',
'x8_v',
'x8_w',
'x8_x',
'x8_y']
```

```
[22] 1 len(all_columns)
```

```
↳ 188
```

```
[40] 1 sample_string=['this contains _ underscore', 'this doesnt']
2 no=[]
3 for i in all_columns:
4     #print(i)
5     jk=False
6     for j in i:
7         if j=='_':
8             jk=True
9     if jk==False:
10        no.append(i)
```

```
[41] 1 no
```

- Splitting data into Train and Test

```
[53] 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,
```

<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 # Maximum 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':
```

```
◀ ▶
```

```
[ ] 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,
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=False),
                                              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, 110]),
                                              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)
```

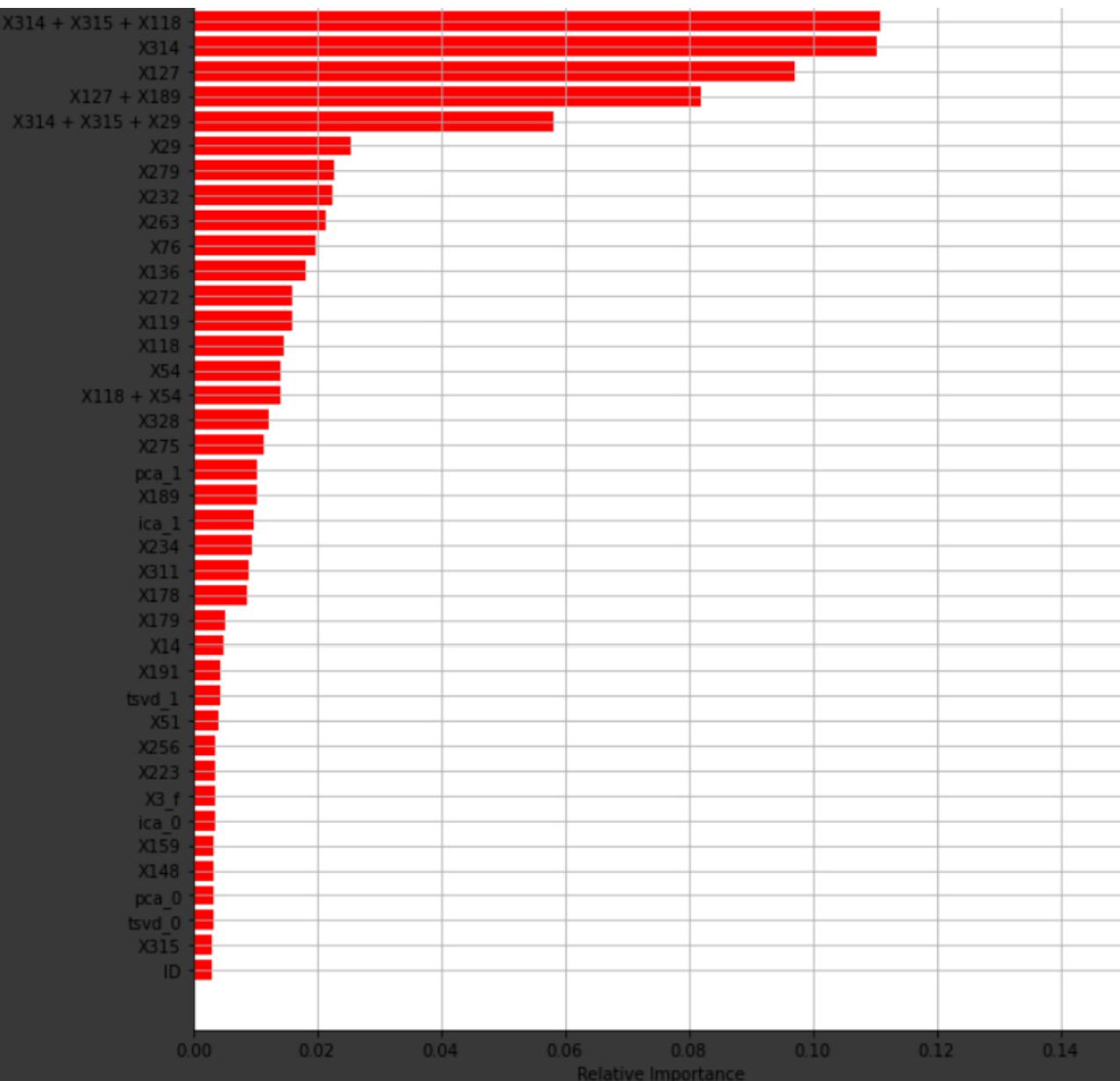
```
[ ] 1 features = x_train.columns
2 importances = best_rf.feature_importances_
3 indices = (np.argsort(importances))[-40:]
4 plt.figure(figsize=(10,12))
5 plt.title('Feature Importances')
6 plt.barh(range(len(indices)), importances[indices], color='r', align='center')
7 plt.yticks(range(len(indices)),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

```
↳
```

Feature Importances

X314 + X315





```
[ ] 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 0.6574681408333823
=====
test r2_score 0.6483743021527534

▼ 2: 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
```

[] [05:33:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

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

[] [05:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:33:55] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:33:59] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:06] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:10] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:45] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:50] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:54] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:56] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:34:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:00] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:02] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:03] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:11] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:22] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:24] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:27] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:29] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:34] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:36] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

[05:35:37] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now

```
[05:35:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:39] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:41] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:43] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:46] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:48] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:53] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
{'n_estimators': 150, 'learning_rate': 0.1}
```

```
[ ] 1 from sklearn.metrics import r2_score
2 train_r2=[]
3 cv_r2=[]
4 dict_cv_r2={}
5 learning_rate=[0.3, 0.2, 0.1, 0.01, 0.001]
6 n_estimators=[50, 100, 150, 200, 250, 300]
7 for i in tqdm_notebook(learning_rate):
8     for j in tqdm_notebook(n_estimators):
9         xb=xgb.XGBRegressor(learning_rate=i,n_estimators=j)
10        xb.fit(x_train,y_train)
11        y_cap_train=xb.predict(x_train)
12        y_cap_cv=xb.predict(x_test)
13        train_r2.append(r2_score(y_train,y_cap_train))
14        k=r2_score(y_test,y_cap_cv)
15        dict_cv_r2[k]=i,j
16        cv_r2.append(r2_score(y_test,y_cap_cv))
17 maximum_auc_score=max(cv_r2)
18 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
[05:35:57] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:35:58] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:01] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:04] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
```

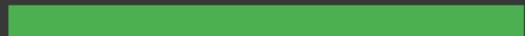
100% [6/6 [00:34<00:00, 5.68s/it]

```
[05:36:13] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:14] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:15] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:18] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:21] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:25] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
```

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
[05:36:31] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:32] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:35] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
[05:36:38] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now
```

```
[05:36:38] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:42] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now
```

100%  6/6 [05:12<00:00, 52.15s/it]

```
[05:36:47] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:48] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:49] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:52] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:55] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:36:59] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now
```

100%  6/6 [04:55<00:00, 49.28s/it]

```
[05:37:04] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:37:05] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:37:07] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:37:09] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:37:12] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now  
[05:37:16] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now
```

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: (0.1, 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)
```

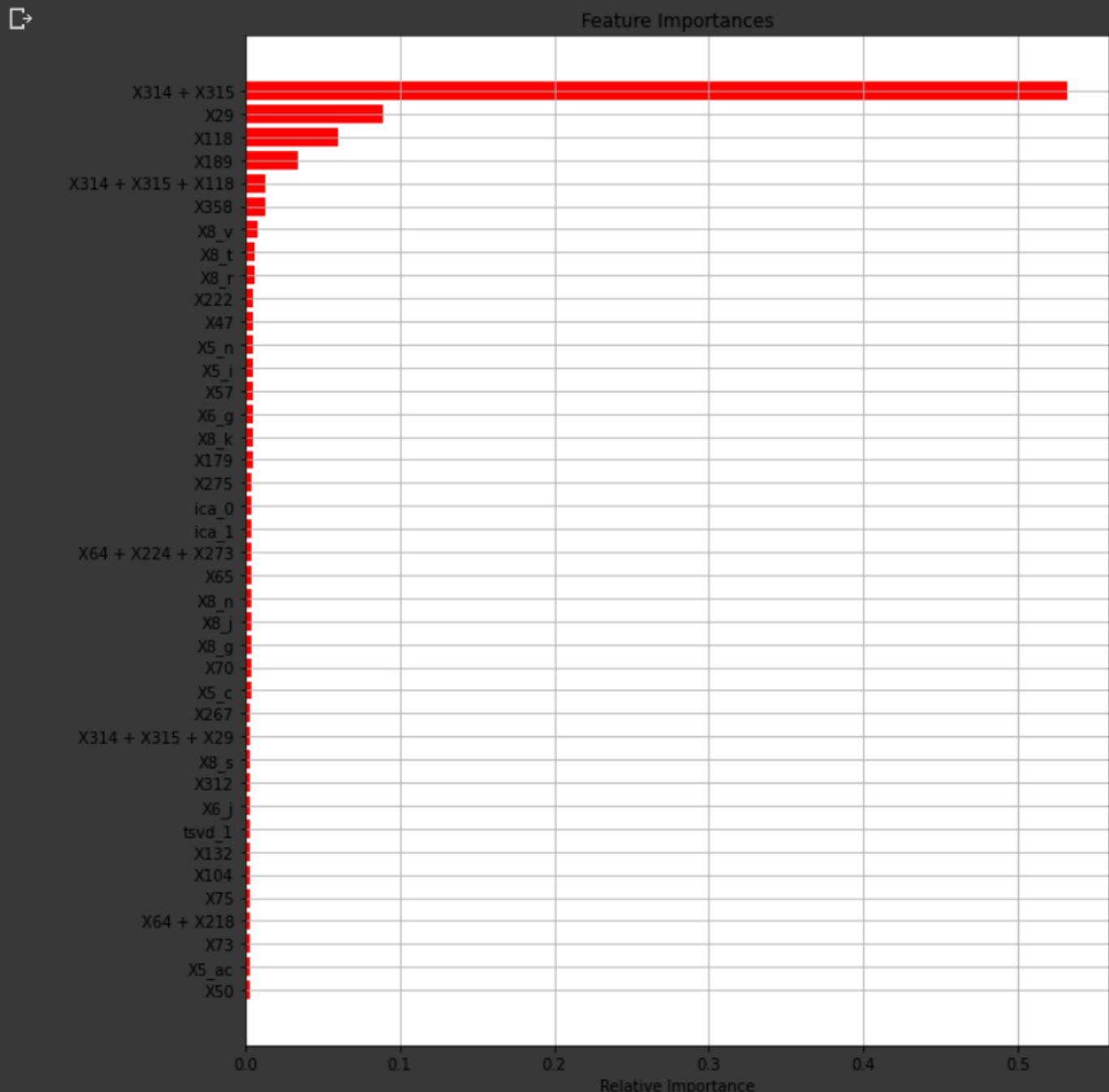
```
↳ [05:37:21] WARNING: /workspace/src/objective/regression_obj.cc:152: reg:linear is now
```

```
[ ] 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))

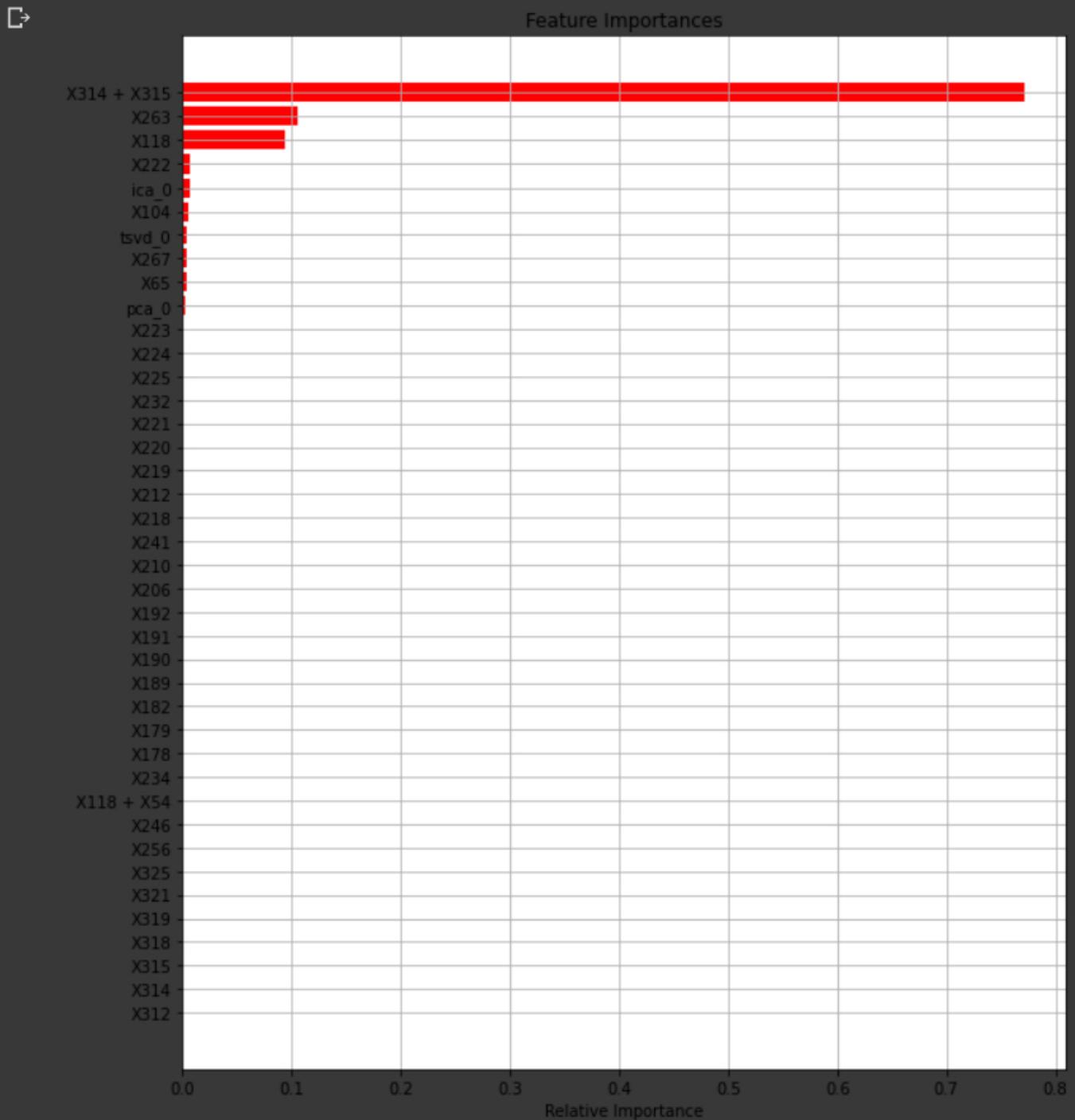
```

```
[ ] 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=42)  
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))
```

```
[4]: 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
```

```
[4]: 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_train, y_train)
```

```
[4]: 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_)
```

```
[4]: 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,
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,
```

```
[ ] 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-1014a2a2a2d0
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
```

```
[ ] 1 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_train, y_train)
```

```
[ ] 1 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_)
```

```
[ ] 1 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
```

```
[ ] 1 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))
```

```
[ ] 1 train r2_score 0.12238421871397831
=====
test r2_score 0.1197384354647586
```

```
[ ] 1
```

▼ 8: AdaBoostRegressor

```
[ ] 1 from sklearn.ensemble import AdaBoostRegressor  
2  
  
[ ] 1 from sklearn.svm import SVR  
2 abr= AdaBoostRegressor()  
3 abr  
  
↳ AdaBoostRegressor(base_estimator=None, learning_rate=1.0, loss='linear',  
                     n_estimators=50, random_state=None)  
  
[ ] 1 parameters = {'learning_rate':[0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6,  
2 gsearch = GridSearchCV(abr, parameters, scoring='r2')  
3 gsearch.fit(x_train, y_train)  
  
↳ 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)  
  
[ ] 1 print(gsearch.best_estimator_)  
2 print(gsearch.best_score_)  
  
↳ AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='exponential',  
                     n_estimators=200, random_state=None)  
0.6106137543959904  
  
[ ] 1 abr= AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='exponential',  
2                               n_estimators=200, random_state=None)  
3 abr  
  
↳ AdaBoostRegressor(base_estimator=None, learning_rate=0.0001, loss='exponential',  
                     n_estimators=200, random_state=None)  
  
[ ] 1 abr.fit(x_train, y_train)  
2 y_train_pred=abr.predict(x_train)  
3 y_test_pred =abr.predict(x_test)  
  
[ ] 1 # checking r2 score  
2 f
```

```
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 0.6134014860036547
=====
test r2_score 0.653160564280501
```

```
[ ] 1
```

▼ 9: Deep Neural Network

```
[44] 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
```

```
[45] 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
```

```
[46] 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()) )
```

```
[47] 1 filepath="weights-{epoch:02d}-{val_r2_keras:.4f}.hdf5"
2 checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_r2_keras', verbose=1)
3 #https://stackoverflow.com/questions/48971221/keras-modelcheckpoint-monitor-multi
4 # issue was resolved by Benzion18's comment on oct 10 2019, link is pasted below
5 #https://github.com/tensorflow/tensorflow/issues/33163
```

```
[48] 1 def changeLearningRate(epoch):
2     initial_learningrate=0.1
3     decay_rate=0.05
4     if (1+epoch)%3==0:
5         changed=initial_learningrate*decay_rate
6     else:
7         changed=initial_learningrate
8     return changed
9 lr_schedule=tf.keras.callbacks.LearningRateScheduler(changeLearningRate, verbose=1)
10 #https://www.tensorflow.org/api\_docs/python/tf/keras/callbacks/LearningRateScheduler
11 #https://www.tensorflow.org/api\_docs/python/tf/keras/callbacks/ReduceLROnPlateau
12 ##changing Learning rate on plateau
13 reduce_lr=tf.keras.callbacks.ReduceLROnPlateau(monitor='val_r2_keras', factor=0.1)
```

```
[49] 1 #https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-1
2 #from reference notebook
3 from tensorflow.keras.callbacks import EarlyStopping
4 earlystop = EarlyStopping(monitor='val_r2_keras', min_delta=0, mode='auto', patience=10)
```

```
[55] 1 x_filtered_= x_dummies[x_dummies ['y']>70]https://www.geeksforgeeks.org/drop-rows-in-pandas-dataframe/
2 x_filtered_= x_filtered_[x_filtered_['y']<150]
3 x_filtered_.describe()
```

	ID	y	X10	X12	X13	X14	
count	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	
std	2437.897217	11.994753	0.114792	0.262439	0.233658	0.494881	
min	0.000000	72.110000	0.000000	0.000000	0.000000	0.000000	
25%	2096.250000	90.800000	0.000000	0.000000	0.000000	0.000000	
50%	4224.000000	99.090000	0.000000	0.000000	0.000000	0.000000	
75%	6316.750000	108.967500	0.000000	0.000000	0.000000	1.000000	
max	8417.000000	149.630000	1.000000	1.000000	1.000000	1.000000	

8 rows × 545 columns

```
[56] 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_))
```

```
[4] (4194, 545)
(4194, 544) (4194,)
<class 'pandas.core.frame.DataFrame'> <class 'pandas.core.series.Series'>
```

```
[57] 1 x_safe=x_filtered_
2 x_safe.head()
3
```

```
[5]   ID X10 X12 X13 X14 X15 X16 X17 X18 X19 X20 X21 X22 X23 X24 X26 X27
  0   0   0   0   1   0   0   0   0   1   0   0   1   0   0   0   0   0   0
  1   6   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   1
  2   7   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   1
  3   9   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
  4  13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
5 rows × 544 columns
```

```
◀ ▶
```

```
[60] 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.)
```

▼ MinMaxScalar and StandardScalar

```
[62] 1 from sklearn.preprocessing import StandardScaler
2 # copy of datasets
3 X_train_stand = X_train.copy()
4 X_test_stand = X_test.copy()
5
```

```
[ ] 1
```

```
[64] 1 all_columns = list(X_train.columns)
2 all_columns
```

```
[65] 1 sample_string=['this contains _ underscore', 'this doesnt']
2 no=[]
3 for i in all_columns:
4     #print(i)
5     jk=False
6     for j in i:
```

```
7         if j=='_':
8             jk=True
9         if jk==False:
10            no.append(i)
11 no[0:5]
```

```
[1] ['ID', 'X10', 'X12', 'X13', 'X14']
```

```
[68] 1 for k in no:#https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-in-python/
2
3     # fit on train data column
4     scale = StandardScaler().fit(X_train[[k]])
5
6     # transform the training data column
7     X_train[k] = scale.transform(X_train[[k]])
8
9     # transform
10    X_test[k] = scale.transform(X_test[[k]])
```

```
[69] 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)
```

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

```
[70] 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)
```

```
[1] (3355, 1) (839, 1)
```

```
[71] 1 print(X_train.shape, Y_train.shape, X_test.shape, Y_test.shape)
```

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

```
[72] 1 X_train
```

```
[1] array([[ 0.51605874, -0.1112284 , -0.28923404, ...,
          0.          , 0.          ],
           [-0.76802942, -0.1112284 , -0.28923404, ...,
          0.          , 0.          ],
           [ 1.5828906 , -0.1112284 , -0.28923404, ...,
          0.          , 0.          ],
           ...,
           [ 0.82270048, -0.1112284 , -0.28923404, ...,
          0.          , 0.          ]])
```

```
          0.        ,  0.        ],
 [ 1.38384246, -0.1112284 , -0.28923404, ... ,  0.        ,
  0.        ,  0.        ],
 [-1.04611748, -0.1112284 , -0.28923404, ... ,  0.        ,
  0.        ,  0.        ]])
```

```
[73] 1 Y_train
```

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

```
[74] 1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs01', histogram_fi
```

```
[75] 1 # input layer
 2 input_layer=Input(shape=(X_train.shape[1],))
 3 # dense hidden layers
 4 layer1=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform)
 5 layer2=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform)
 6 layer3=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform)
 7 layer4=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform)
 8 layer5=Dense(512,activation='relu',kernel_initializer=keras.initializers.he_uniform)
 9 # output layer
10 output=Dense(1)(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,
20                                     loss='mean_squared_error', metrics=[r2_keras])
21 model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=
```

52.12 notebook

```
↳ Epoch 1/5
13/14 [=====>...] - ETA: 0s - loss: 3439.4260 - r2_keras: -23.548
Epoch 00001: val_r2_keras did not improve from -50.54527
14/14 [=====] - 1s 63ms/step - loss: 3414.5684 - r2_keras: -23.548
Epoch 2/5
13/14 [=====>...] - ETA: 0s - loss: 391.0041 - r2_keras: -1.7567
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras did not improve from -50.54527
14/14 [=====] - 1s 51ms/step - loss: 390.3332 - r2_keras: -1.7567
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f4c91f76fd0>
```

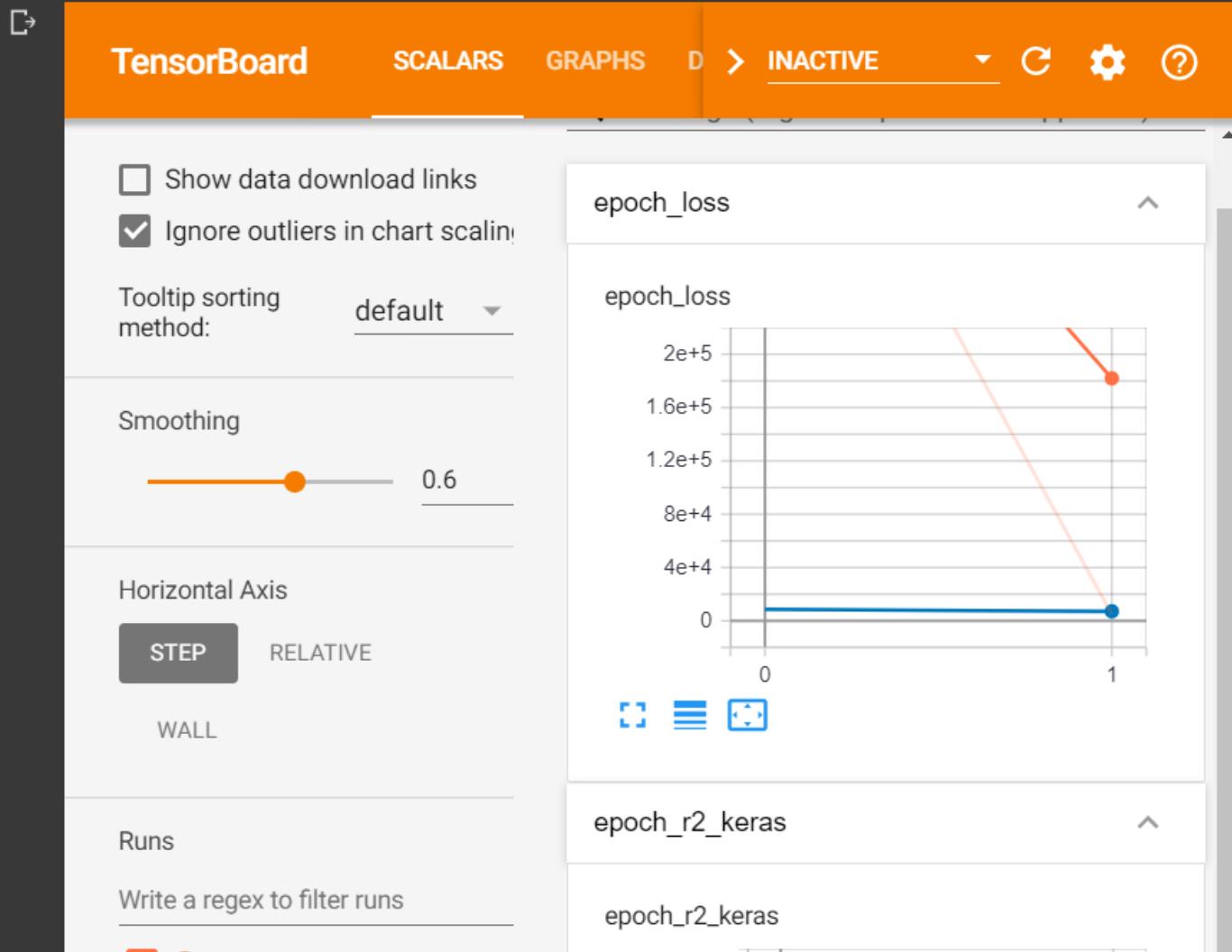
```
[76] 1 #y_train_pred=model.predict(x_train)
      2 y_test_pred =model.predict(x_filtered_)
```

```
[77] 1 final_df_9= pd.DataFrame(list(zip(x_safe['ID'].values, y_test_pred)), columns=[ 'ID', 'y'])
```

```
[78] 1 final_df_9.head()
```

```
   ID          y
0  0  [36.933903]
1  6  [71.677605]
2  7  [39.63203]
3  9  [48.80096]
4 13  [57.53786]
```

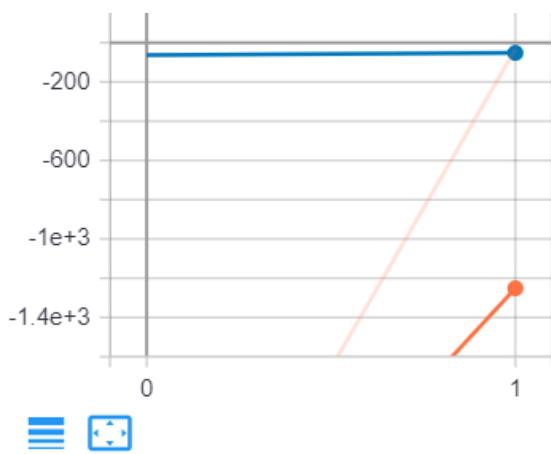
```
[ ] 1 %load_ext tensorboard
2 import datetime, os
3 %tensorboard --logdir logs01
4 ##https://medium.com/@kuangoong/how-to-use-tensorboard-with-google-colab-43f7cf06:
```



train
 validation

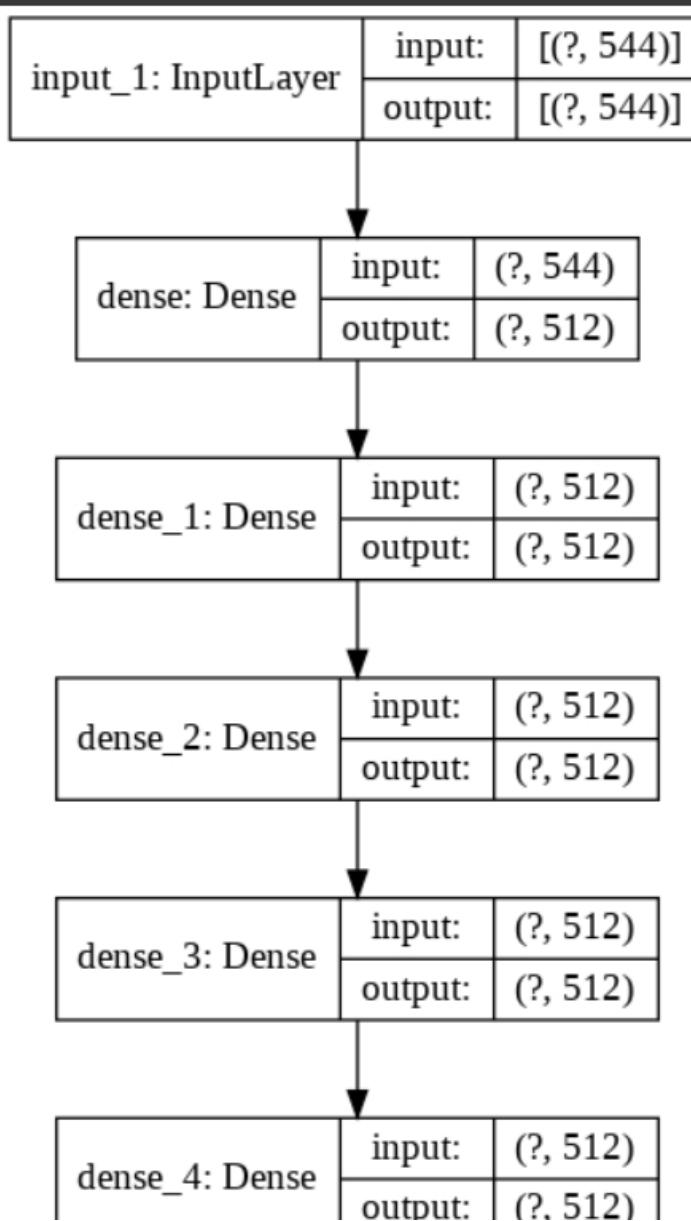
TOGGLE ALL RUNS

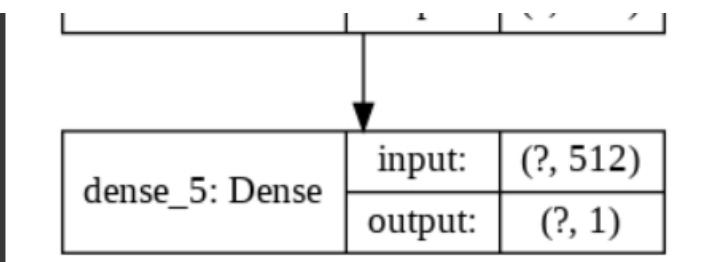
logs01



```
[ ] 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 )
```

↳





▼ 10: CNN

```
[79] 1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs2', histogram_fra
```

```
[80] 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_1')
9 drop_a=Dropout(0.1)(conv1_1)
10 conv1_2= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv1_2')
11 drop_b=Dropout(0.2)(conv1_2)
12 conv1_3= Conv1D(filters=64, kernel_size=3, strides=1, padding='valid', name='conv1_3')
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', name='dense_1_ac')
17 dropout_1_ac = Dropout(0.1)(dense_1_ac)
18 dense_2_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_2_ac')
19 dropout_2_ac = Dropout(0.2)(dense_2_ac)
20 dense_3_ac = Dense(units=64, activation='relu', kernel_initializer='he_normal', name='dense_3_ac')
21 dropout_3_ac = Dropout(0.1)(dense_3_ac)
22 output = Dense(units=1, activation='linear', kernel_initializer='he_normal', name='output')
23
24 model= Model(inputs=input, outputs=output)
```

```
[81] 1 model.summary()
```

↳ Model: "functional_5"

Layer (type)	Output Shape	Param #
<hr/>		
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

```
[82] 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)

[83] 1 model.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=32
2

↳ Epoch 1/5
14/14 [=====] - ETA: 0s - loss: 6151.6675 - r2_keras: -42.209
Epoch 00001: val_r2_keras did not improve from -50.54527
14/14 [=====] - 11s 754ms/step - loss: 6151.6675 - r2_keras: -42.209
Epoch 2/5
14/14 [=====] - ETA: 0s - loss: 1246.5182 - r2_keras: -7.620
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 00002: val_r2_keras improved from -50.54527 to -23.62599, saving model to weight
14/14 [=====] - 10s 743ms/step - loss: 1246.5182 - r2_keras: -7.620
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f4c90c5f080>
```

```
1 #y_train_pred=model.predict(X_train)
2 y_test_pred =model.predict(x_filtered_)
```

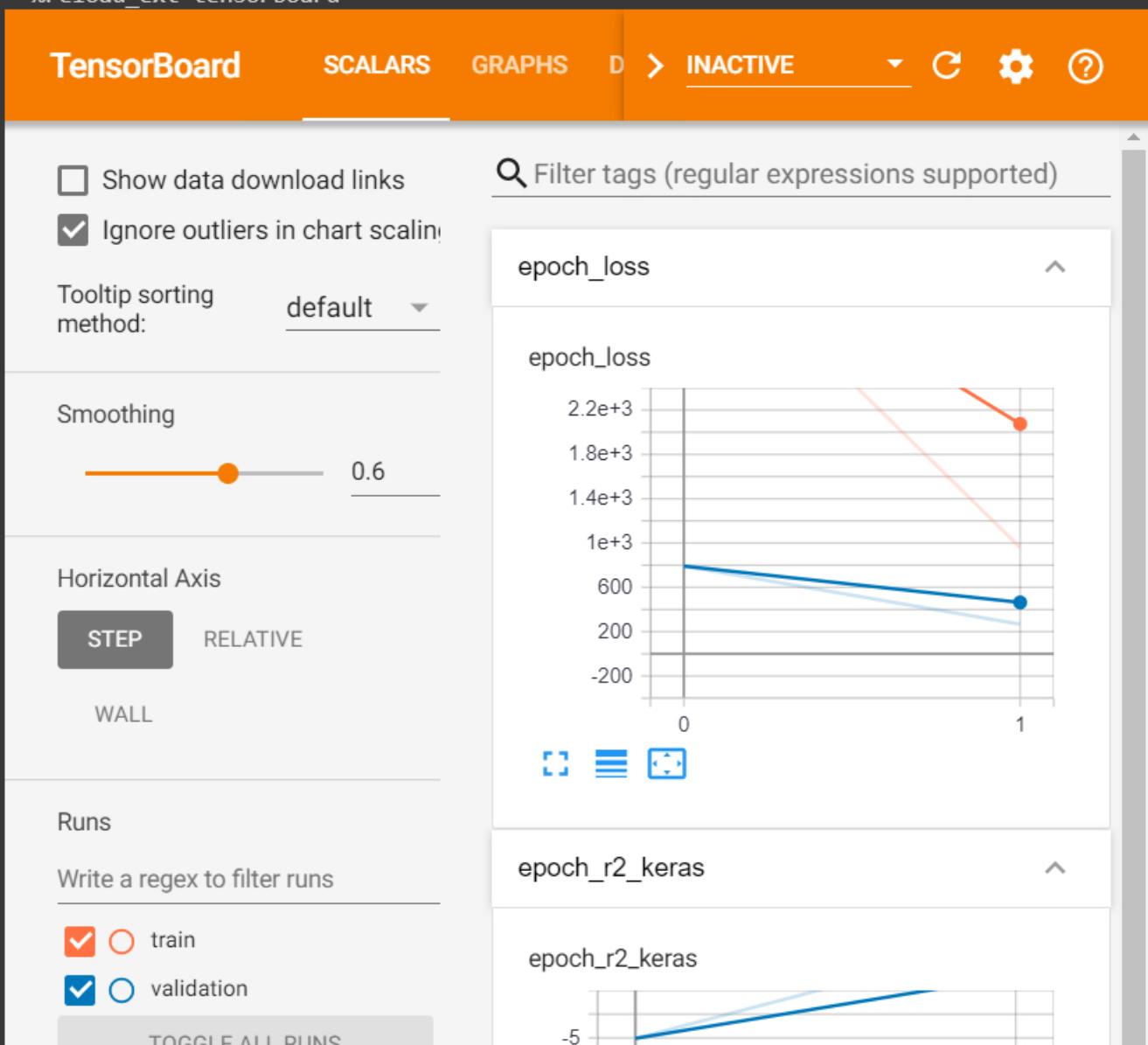
```
[85] 1 final_df = pd.DataFrame(list(zip(x_safe['ID'].values, y_test_pred)), columns=['ID', 'pred'])
```

```
[86] 1 final_df_9.head()
```

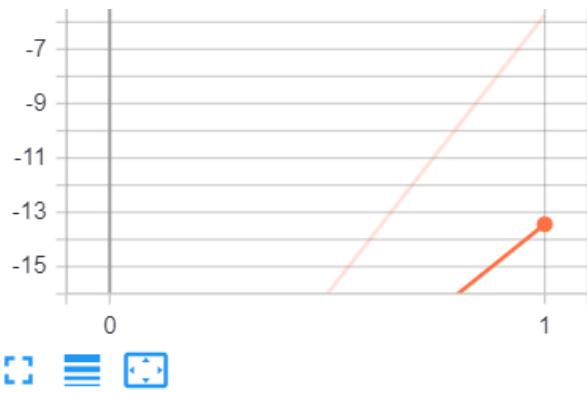
	ID	y
0	0	[44.4179]
1	6	[54.337364]
2	7	[20.183456]
3	9	[18.924988]
4	13	[19.856712]

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

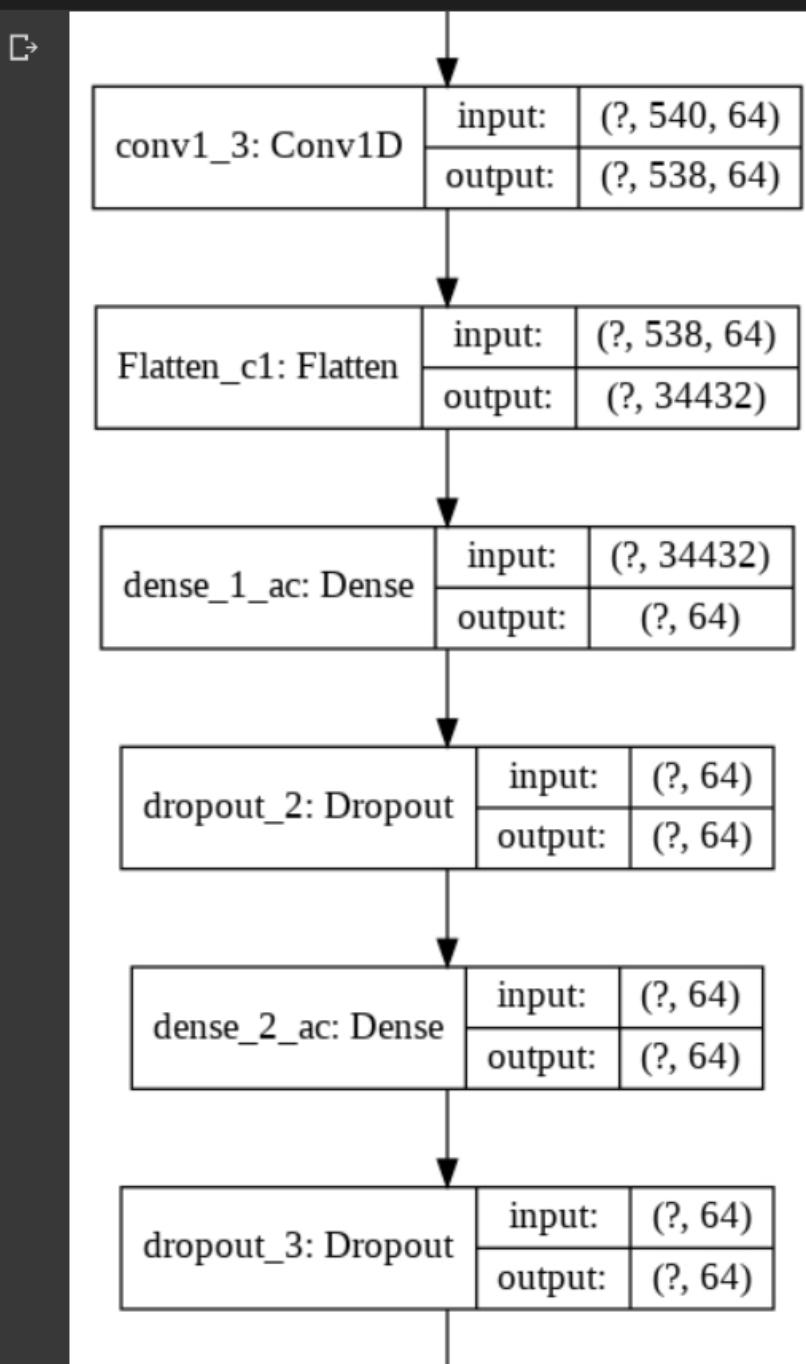
↳ The tensorboard extension is already loaded. To reload it, use:
 %reload_ext tensorboard

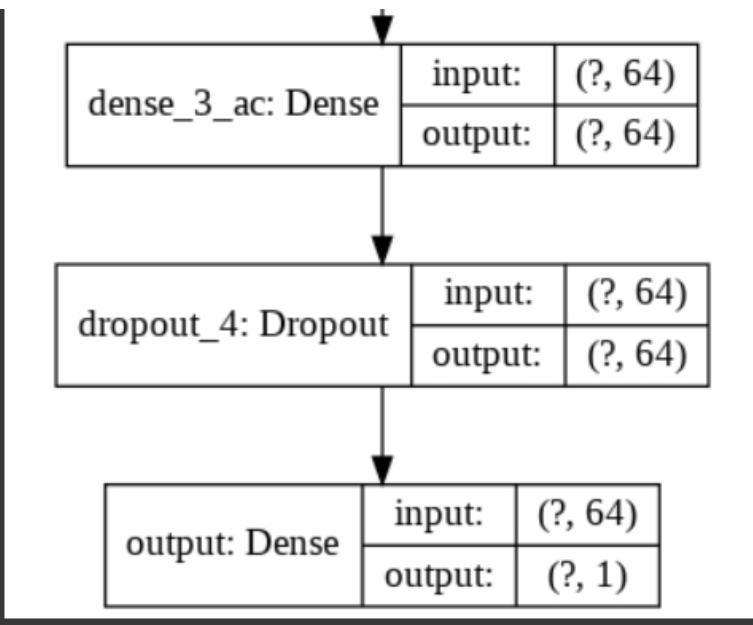


logs2



```
[ ] 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 )
```





▼ 11: LSTM

```
[87] 1 tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir='logs3', histogram_fra
```

```
[88] 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(0.1)(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(0.2)(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(0.1)(dense_3_ac)
11 output_1 = Dense(units=1, activation='linear', kernel_initializer='he_normal', name='output_1')(dropout_3_ac)
12
13 model= Model(inputs=input_1, outputs=output_1)
```

```
[89] 1 model.summary()
```

↳ Model: "functional_7"

Layer (type)	Output Shape	Param #
<hr/>		
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		

```
[90] 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)

[91] 1 el.fit(X_train, Y_train, epochs=5, validation_data=(X_test, Y_test), batch_size=25
```

↳ Epoch 1/5
14/14 [=====] - ETA: 0s - loss: 10181.2266 - r2_keras: -72.13
Epoch 00001: val_r2_keras did not improve from -23.62599
14/14 [=====] - 13s 950ms/step - loss: 10181.2266 - r2_keras: -72.13
Epoch 2/5
14/14 [=====] - ETA: 0s - loss: 7333.7627 - r2_keras: -49.591
Epoch 00002: ReduceLROnPlateau reducing learning rate to 0.0001000000474974513.
Epoch 00002: val_r2_keras did not improve from -23.62599
14/14 [=====] - 13s 904ms/step - loss: 7333.7627 - r2_keras: -49.591
Epoch 00002: early stopping
<tensorflow.python.keras.callbacks.History at 0x7f4c900f4748>

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

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

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

	ID	y
0	0	[37.142464]
1	6	[37.142574]

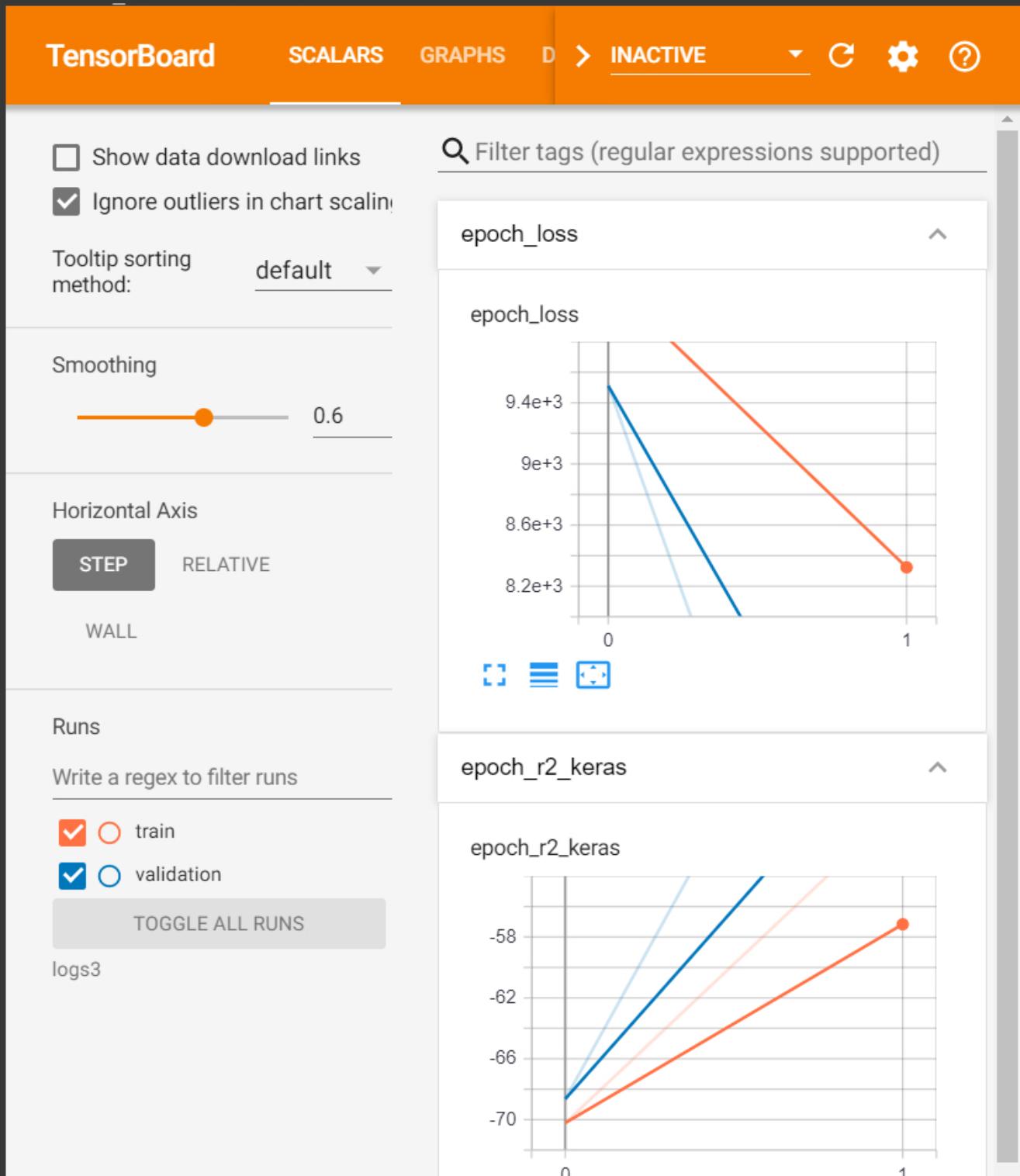
```
2 7 [37.203022]
```

```
3 9 [37.140675]
```

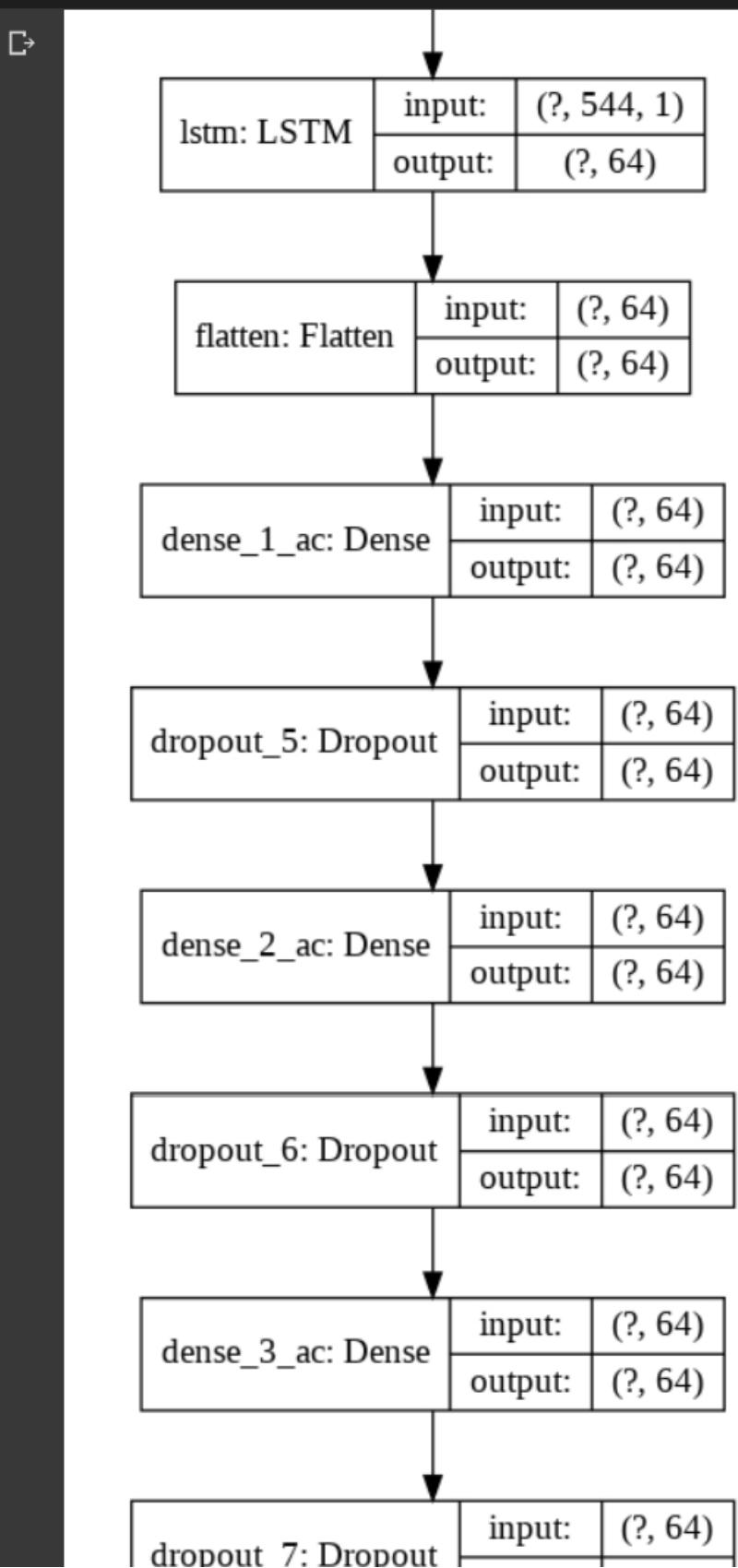
```
4 13 [37.142017]
```

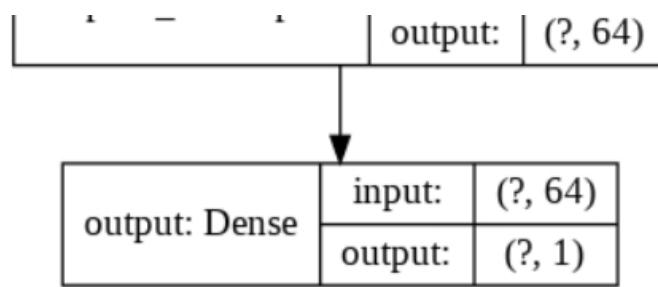
```
[ ] 1 %load_ext tensorboard
2 import datetime, os
3 %tensorboard --logdir logs3
4 ##https://medium.com/@kuanhong/how-to-use-tensorboard-with-google-colab-43f7cf06:
```

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



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





▼ 12: 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)]
```

```
[ ] 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

▼ 13: 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