FEATURE ENGINEERING, DATA ANALYTICS, VISUALIZATION AND ADVANCED MATHEMATICAL MODELLING TO PREDICT SHRINKAGE LIMIT OF SOIL USING DIFFERENT PHYSICAL PROPERTIES OF SOIL

PRATEEK PAL (prateekpal641@gmail.com)

ABSTRACT:

We have been given the data which contains values for several properties of soil and our objective is to make a model using labels which will predict values of shrinkage limit in future foe the given values of different properties. As, the data have many missing values for several columns and data also contains outliers so the data is highly unstable therefore, the first task here we perform is to fill in the missing values using statical and machine learning models. After the data does not contain any missing values, we analyze our data and try to derive some important conclusions. We also draw some beautiful visualizations to visualize the data closely.

After feature engineering, data analysis and visualization we move on to mathematical modeling and use some models like Lasso, Ridge, ElasticnetCV, Support Vector Regressor, RandomForestRegressor, DecisionTreeRegressor to obtain max accuracy we tune their hyperparameters. After linear models we also design deep neural networks to see if it can provide us with better accuracy and tune it's number of hidden layers, number of hidden units and activation functions to obtain max accuracy for our validation set.

KEY WORDS

Gravel (%), Sand (%), Silt (%), Clay (%), Liquid Limit, LL (%), Plastic Limit, PL (%), Plasticity Index, PI (%), PI = LL - PL, Specific Gravity, Maximum Dry Density, MDD (kN/m3), Optimum Moisture Content, OMC (%), Shrinkage Limit, SL (%), Lasso, Ridge, ElasticnetCV, Support Vector Regressor, RandomForestRegressor, DecisionTreeRegressor, DeepNeuralNetworks, Adam.

INTRODUCTION

The given data is on different physical properties of soil like Gravel (%), Sand (%), Silt (%), Clay (%), Liquid Limit, LL (%), Plastic Limit, PL (%), Plasticity Index, PI (%), PI = LL - PL, Specific Gravity, Maximum Dry Density, MDD (kN/m3), Optimum Moisture Content, OMC (%) and our objective is to use these features and predict the shrinkage limit of soil.

Our objective is to make a mathematical model using present values for physical properties and shrinkage limit to predict values for shrinkage limit in future. We have used several machine learning models here and also a deep neural network too.

As, we have very small amount of data which too have a lot of missing values with so getting good accuracy for our model is too hard though we try to tune our hyperparameters so that we get a good accuracy for our model.

LITERATURE REVIEW

The shrinkage limit is one of the Atterberg limits and is a fundamental geotechnical parameter used for the assessment of the settlement of clay soils due to reduction in water content, yet is rarely tested as part of ground investigation. This paper describes shrinkage limit test results on a variety of soils from Britain and overseas obtained using an improved laboratory testing procedure developed at the British Geological Survey (BGS). The co-relationships with the other Atterberg limits and with density are explored. In particular, the coincidence of the shrinkage limit with the water content at the peak bulk density achieved in the test is examined. The shrinkage behavior for undisturbed and remolded states and a 3-way relationship between water content, density, and suction are demonstrated. Some tropical residual and highly smectite soils show a very wide range of shrinkage behavior, albeit for a small dataset, when compared with the larger dataset of temperate soils tested. Consideration is given to the limitations of the new and existing test methods.

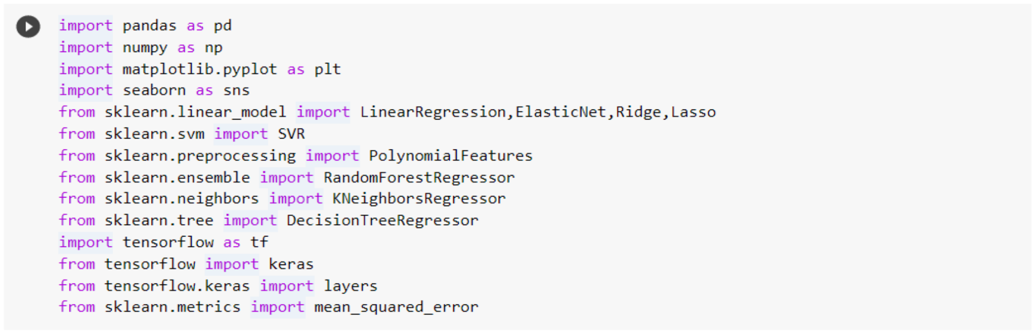
LIBRARIES USED

Two libraries for data manipulation NumPy pandas has been used.

Two libraries for data visualization have also been used.

Several sub libraries of skleran are used for preprocessing, model making and metrics.

A deep learning framework – TensorFlow is also used to implement deep neural network.



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression,ElasticNet,Ridge,Lasso
from sklearn.svm import SVR
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import mean_squared_error
```

ALGORITHMS USED

LASSO -> lasso (least absolute shrinkage and selection operator; also Lasso or LASSO) is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the resulting statistical model.

It performs like a normal linear regression but to add regularization in the algorithm it introduces a regularization term to the loss function which is sum of mod of all the weights multiplies by the coefficient gamma.

$$\mathcal{L}(\mathbb{L}) = \sum_{l=1}^{L} \left(y_l - \mathbf{x}_l^T \mathbf{w} - b\right)^2 + \lambda |\mathbf{w}|$$

RIDGE-> possible solution to the imprecision of least square estimators when linear regression models have some multicollinear (highly correlated) independent variables—by creating a ridge regression estimator (RR).

It is also adds a regularization term to the loss function just like lasso but instead of adding mods of weight multiplied by gamma it adds a term which is addition of square of weights multiplies by the coefficient gamma.

$$\mathcal{L}(\mathbb{L}) = \sum_{l=1}^{L} \left(y_l - \mathbf{x}_l^T \mathbf{w} - b\right)^2 + \lambda ||\mathbf{w}||^2$$

ELASTICNET-> Elastic Net first emerged as a result of critique on the lasso, whose variable selection can be too dependent on data and thus unstable. The solution is to combine the penalties of ridge regression and lasso to get the best of both worlds.

Regularization term in the elastic net is summation of regularization terms in lasso and ridge.

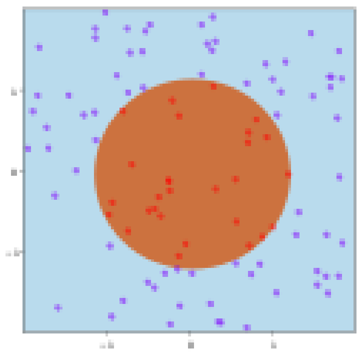
$$\hat{\beta} \equiv \underset{\beta}{\operatorname{argmin}}(\|y - X\beta\|^2 + \lambda_2 \|\beta\|^2 + \lambda_1 \|\beta\|_1).$$

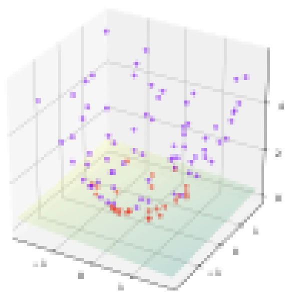
SUPPORT VECTOR MACHINE-> The classifier separates data points using a hyperplane with the largest amount of margin. That's why an SVM classifier is also known as a discriminative classifier. SVM finds an optimal hyperplane which helps in classifying new data points.

It takes input of our data and scales it such that it is able to draw a boundary between relevant data.

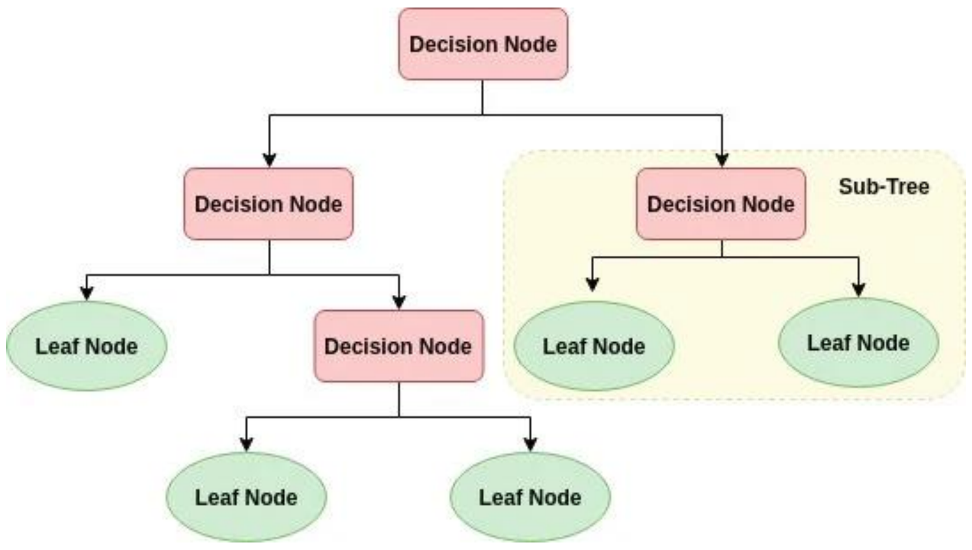
This algorithm is generally used for classification and gives a very good accuracy.

The algorithm discriminates the data on the basis of function given as hyperparameter, the available hyperparameters in scikit learn are – rbf, linear, poly, sigmoid.

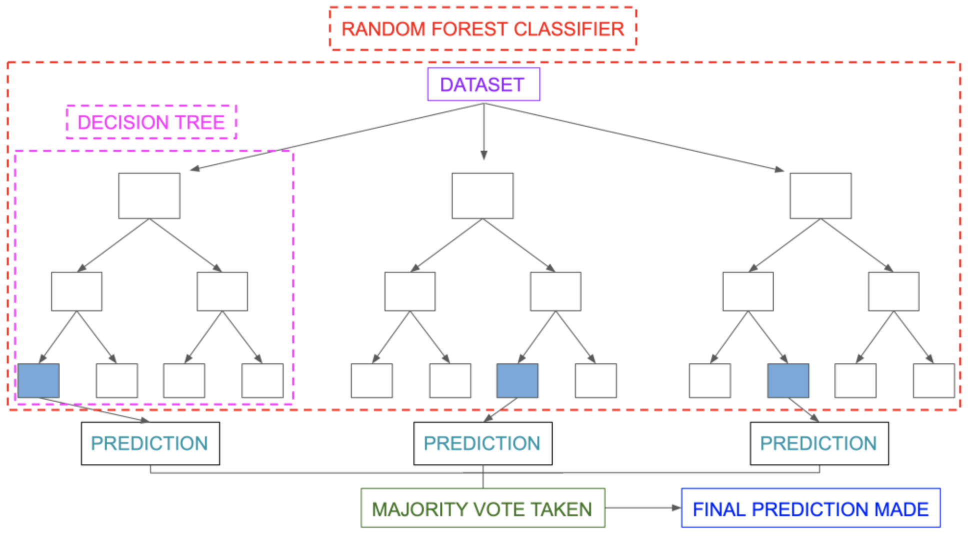




Decision Trees-> A decision tree is a flowchart-like tree structure where an internal node represents a feature(attribute), the branch represents a decision rule, and each leaf node represents the outcome. The topmost node in a decision tree is known as the root node. It learns to partition on the basis of the attribute value. It partitions the tree in a recursive manner call recursive partitioning. This flowchart-like structure helps you in decision-making. It's visualization like a flowchart diagram which easily mimics the human level thinking. That is why decision trees are easy to understand and interpret.



RANDOM FOREST-> A rain forest system relies on various decision trees. Every decision tree consists of decision nodes, leaf nodes, and a root node. The leaf node of each tree is the final output produced by that specific decision tree. The selection of the final output follows the majority-voting system. In this case, the output chosen by the majority of the decision trees becomes the final output of the rain forest system.

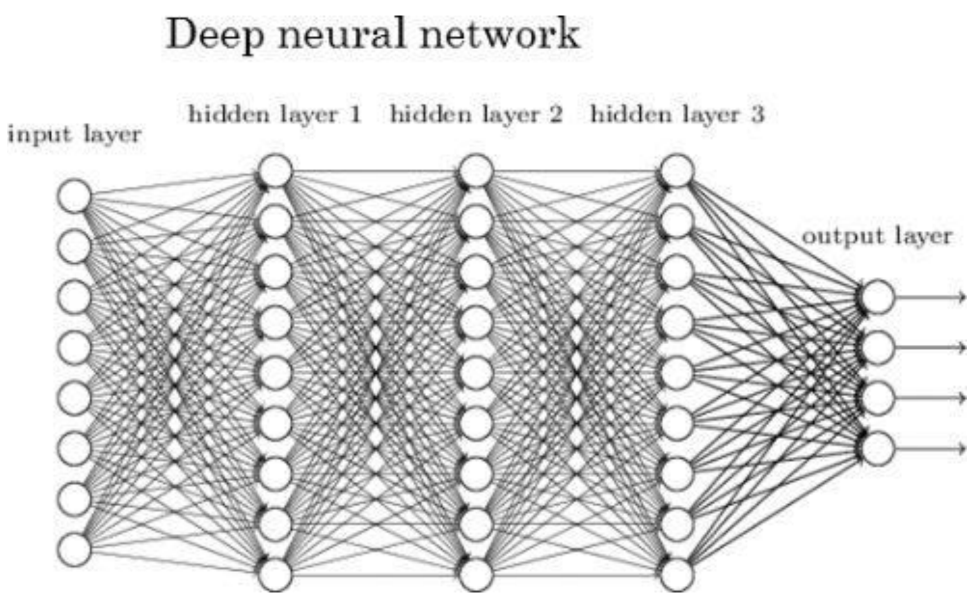


DEEP NEURAL NETWORKS-> Deep neural network represents the type of machine learning when the system uses many layers of nodes to derive high-level functions from input information. It means transforming the data into a more creative and abstract component.

In a deep learning neural network every unit in every layer helps in computing result for next layer and after every iteration loss function is calculated and derivative of loss function with respect to particular weight is subtracted by the weight after it is multiplied by the learning rate.

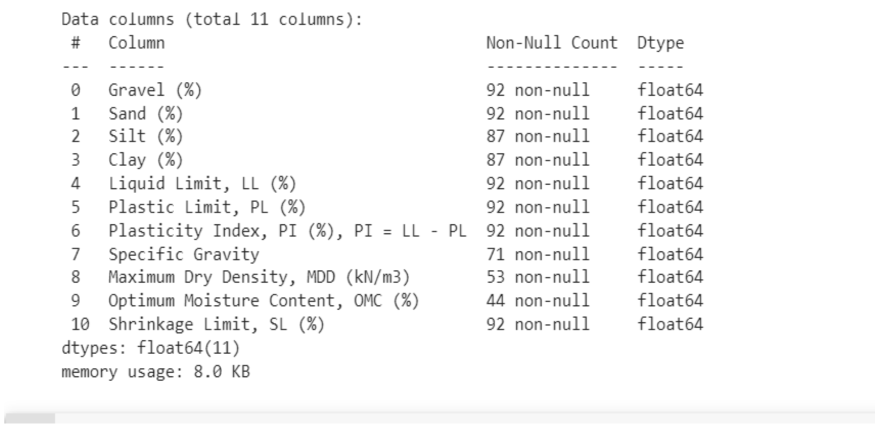
Like this we are able to reach local optimum.

Several algorithms have been designed to optimize deep neural networks like rmsprop and adam which can be applied in any deep learning framework easily.



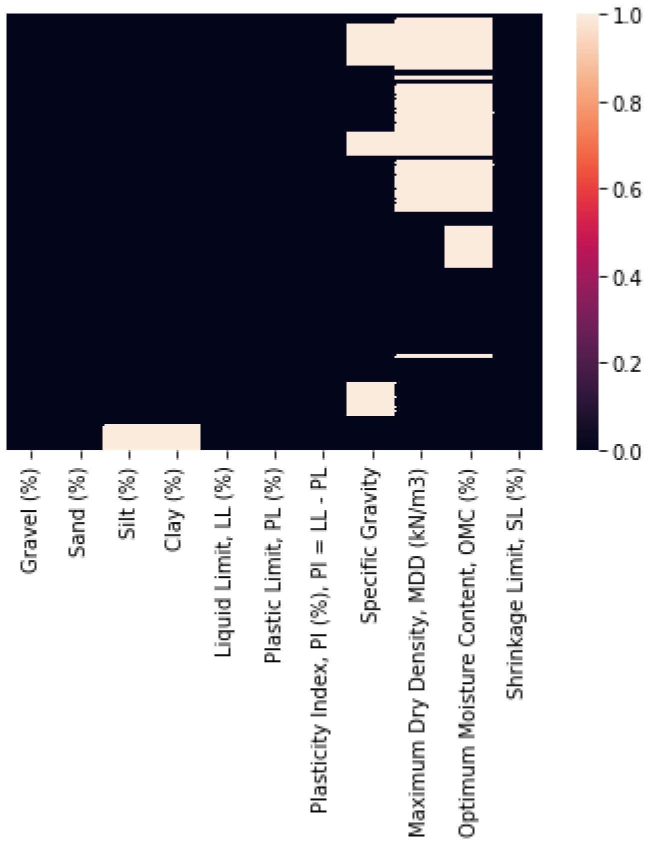
DATA

The given data has 11 columns and 92 rows. All values are of float type and there are missing values in Silt, Clay, Specific Gravity, Max dry Density and Optimal Moisture content.



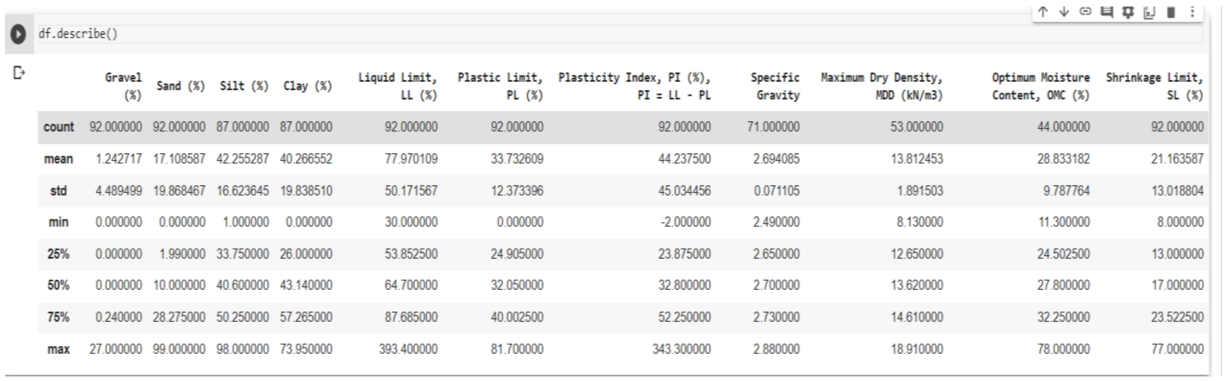
```
Data columns (total 11 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   Gravel (%)                            92 non-null     float64
 1   Sand (%)                              92 non-null     float64
 2   Silt (%)                              87 non-null     float64
 3   Clay (%)                              87 non-null     float64
 4   Liquid Limit, LL (%)                  92 non-null     float64
 5   Plastic Limit, PL (%)                 92 non-null     float64
 6   Plasticity Index, PI (%), PI = LL - PL  92 non-null     float64
 7   Specific Gravity                      71 non-null     float64
 8   Maximum Dry Density, MDD (kN/m3)      53 non-null     float64
 9   Optimum Moisture Content, OMC (%)     44 non-null     float64
 10  Shrinkage Limit, SL (%)               92 non-null     float64
dtypes: float64(11)
memory usage: 8.0 KB
```

As there are many missing values in our data for many columns first, we try to see missing values by plotting a heat map.



The data contains missing values for silt, clay, Specific Gravity, Maximum Dry Density and Optimum Moisture Content.

To fill in these values we check if the values can be filled by some statistical models by seeing some statistical calculations for our data.

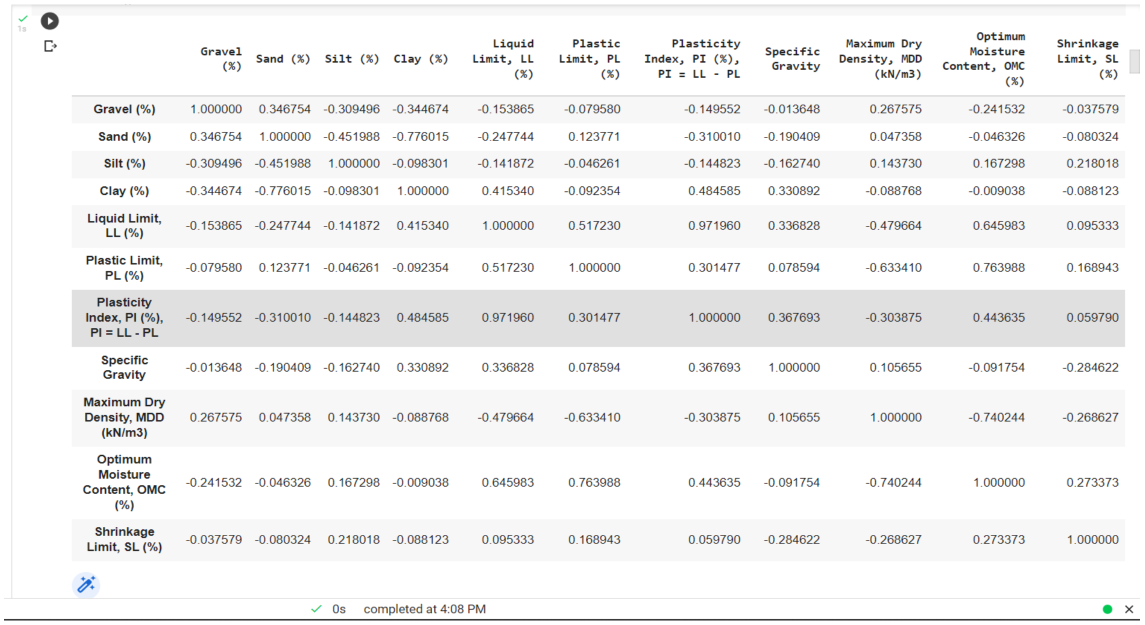


```
df.describe()
```

	Gravel (%)	Sand (%)	Silt (%)	Clay (%)	Liquid Limit, LL (%)	Plastic Limit, PL (%)	Plasticity Index, PI (%), PI = LL - PL	Specific Gravity	Maximum Dry Density, MDD (kN/m3)	Optimum Moisture Content, OMC (%)	Shrinkage Limit, SL (%)
count	92.000000	92.000000	87.000000	87.000000	92.000000	92.000000	92.000000	71.000000	53.000000	44.000000	92.000000
mean	1.242717	17.108587	42.255287	40.266552	77.970109	33.732609	44.237500	2.694085	13.812453	28.833182	21.163587
std	4.489499	19.868467	16.623645	19.838510	50.171567	12.373396	45.034456	0.071105	1.891503	9.787764	13.018804
min	0.000000	0.000000	1.000000	0.000000	30.000000	0.000000	-2.000000	2.490000	8.130000	11.300000	8.000000
25%	0.000000	1.990000	33.750000	26.000000	53.852500	24.905000	23.875000	2.650000	12.650000	24.502500	13.000000
50%	0.000000	10.000000	40.600000	43.140000	64.700000	32.050000	32.800000	2.700000	13.620000	27.800000	17.000000
75%	0.240000	28.275000	50.250000	57.265000	87.685000	40.002500	52.250000	2.730000	14.610000	32.250000	23.522500
max	27.000000	99.000000	98.000000	73.950000	393.400000	81.700000	343.300000	2.880000	18.910000	78.000000	77.000000

Seeing, mean, max and min values for data we conclude that only specific gravity column can be imputed by a statistical model other columns need some other models to impute values.

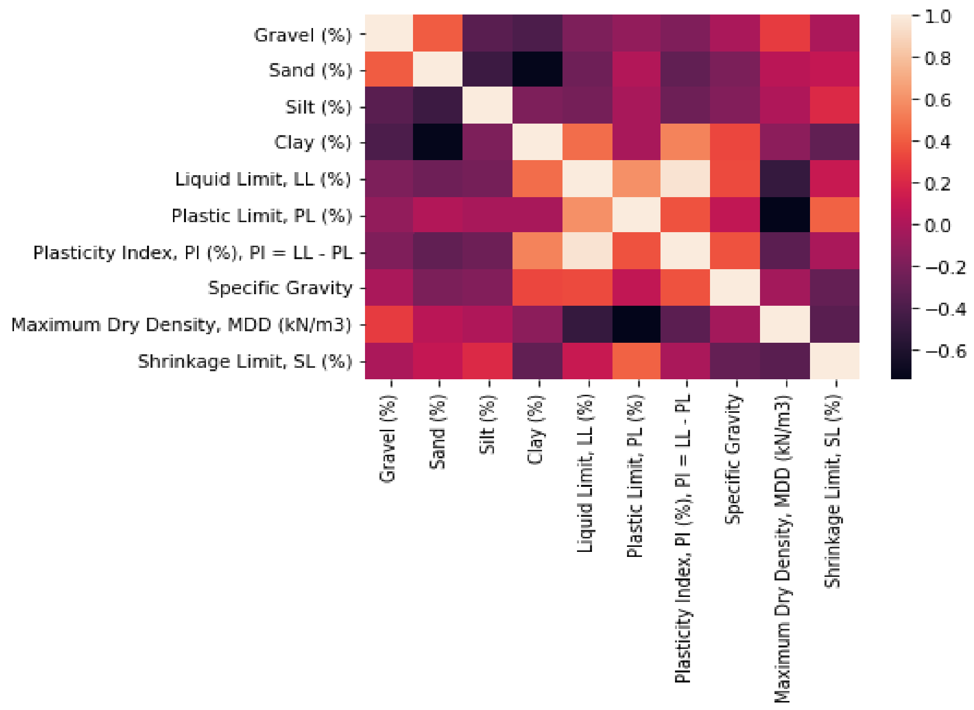
So, for other columns we need to some machine learning models for imputation to make machine learning models we first check the correlation values between the columns and see what columns correlates best with the column which we have to impute.



	Gravel (%)	Sand (%)	Silt (%)	Clay (%)	Liquid Limit, LL (%)	Plastic Limit, PL (%)	Plasticity Index, PI (%), PI = LL - PL	Specific Gravity	Maximum Dry Density, MDD (kN/m3)	Optimum Moisture Content, OMC (%)	Shrinkage Limit, SL (%)
Gravel (%)	1.000000	0.346754	-0.309496	-0.344674	-0.153865	-0.079580	-0.149552	-0.013648	0.267575	-0.241532	-0.037579
Sand (%)	0.346754	1.000000	-0.451988	-0.776015	-0.247744	0.123771	-0.310010	-0.190409	0.047358	-0.046326	-0.080324
Silt (%)	-0.309496	-0.451988	1.000000	-0.098301	-0.141872	-0.046261	-0.144823	-0.162740	0.143730	0.167298	0.218018
Clay (%)	-0.344674	-0.776015	-0.098301	1.000000	0.415340	-0.092354	0.484585	0.330892	-0.088768	-0.009038	-0.088123
Liquid Limit, LL (%)	-0.153865	-0.247744	-0.141872	0.415340	1.000000	0.517230	0.971960	0.336828	-0.479664	0.645983	0.095333
Plastic Limit, PL (%)	-0.079580	0.123771	-0.046261	-0.092354	0.517230	1.000000	0.301477	0.078594	-0.633410	0.763988	0.168943
Plasticity Index, PI (%), PI = LL - PL	-0.149552	-0.310010	-0.144823	0.484585	0.971960	0.301477	1.000000	0.367693	-0.303875	0.443635	0.059790
Specific Gravity	-0.013648	-0.190409	-0.162740	0.330892	0.336828	0.078594	0.367693	1.000000	0.105655	-0.091754	-0.284622
Maximum Dry Density, MDD (kN/m3)	0.267575	0.047358	0.143730	-0.088768	-0.479664	-0.633410	-0.303875	0.105655	1.000000	-0.740244	-0.268627
Optimum Moisture Content, OMC (%)	-0.241532	-0.046326	0.167298	-0.009038	0.645983	0.763988	0.443635	-0.091754	-0.740244	1.000000	0.273373
Shrinkage Limit, SL (%)	-0.037579	-0.080324	0.218018	-0.088123	0.095333	0.168943	0.059790	-0.284622	-0.268627	0.273373	1.000000

✓ 0s completed at 4:08 PM

To visualize our correlation values more easily we plot a heatmap on correlation values.



Visualizing the heatmap we derive some conclusions and finalize that what columns we can use to derive missing values.

-> For the silt column – Gravel and Sand are highly correlated with it.

-> For the Clay columns – Again Gravel and Sand are highly correlated.

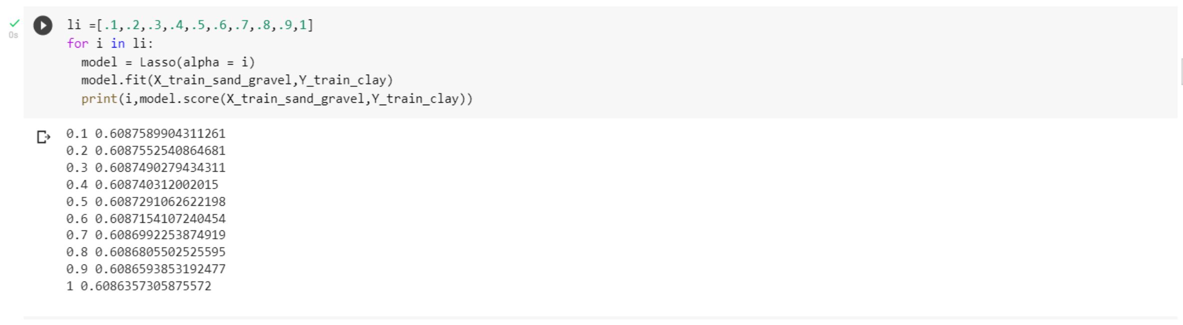
-> For the Maximum Dry Density – Liquid and Plastic Limit are highly are correlated.

-> For the optimum moisture content – As this column is highly correlated with Maximum Dry Density so we can drop it.

First let us build a model to fill in the missing values in clay column using gravel and sand column.

Below are the models with accuracy which they provide with training data.

Lasso Model -> we build Lasso model and check its accuracy for different values of alpha.

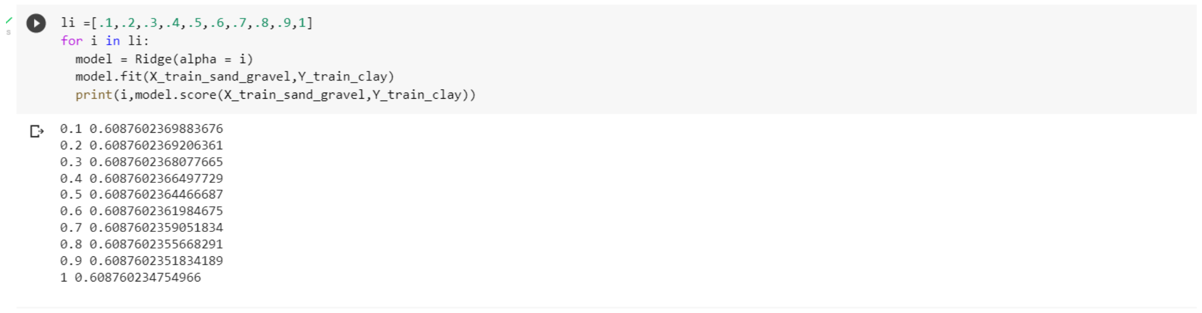


```
li =[.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
for i in li:
  model = Lasso(alpha = i)
  model.fit(X_train_sand_gravel,Y_train_clay)
  print(i,model.score(X_train_sand_gravel,Y_train_clay))
```

```
0.1 0.6087589904311261
0.2 0.6087552540864681
0.3 0.6087490279434311
0.4 0.608740312002015
0.5 0.6087291062622198
0.6 0.6087154107240454
0.7 0.6086992253874919
0.8 0.6086805502525595
0.9 0.6086593853192477
1 0.6086357305875572
```

The accuracy values do not change for different values of alpha as there are only two features.

Ridge Model – Again we use ridge model for different values of alpha and see accuracy on training data.



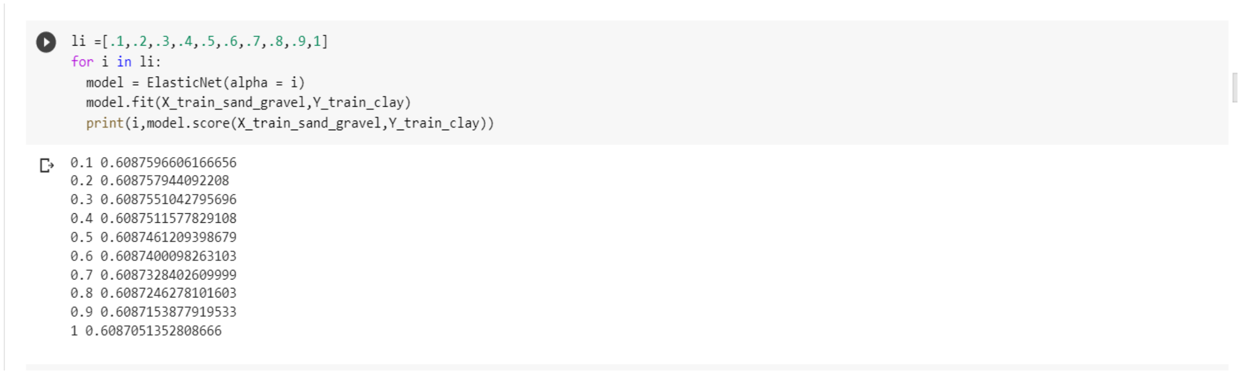
```
li =[.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
for i in li:
  model = Ridge(alpha = i)
  model.fit(X_train_sand_gravel,Y_train_clay)
  print(i,model.score(X_train_sand_gravel,Y_train_clay))
```

```
0.1 0.6087602369883676
0.2 0.6087602369206361
0.3 0.6087602368077665
0.4 0.6087602366497729
0.5 0.6087602364466687
0.6 0.6087602361984675
0.7 0.6087602359051834
0.8 0.6087602355668291
0.9 0.6087602351834189
1 0.608760234754966
```

Again, the accuracy values are same which are near to 60% so we are not satisfied with model and try some different models.

Our next model is a combination of Ridge and Lasso.

Elastic Net -> this model is a combination of ridge and lasso -> here hyperparameter alpha is given as the ratio of coefficients of ridge and lasso.



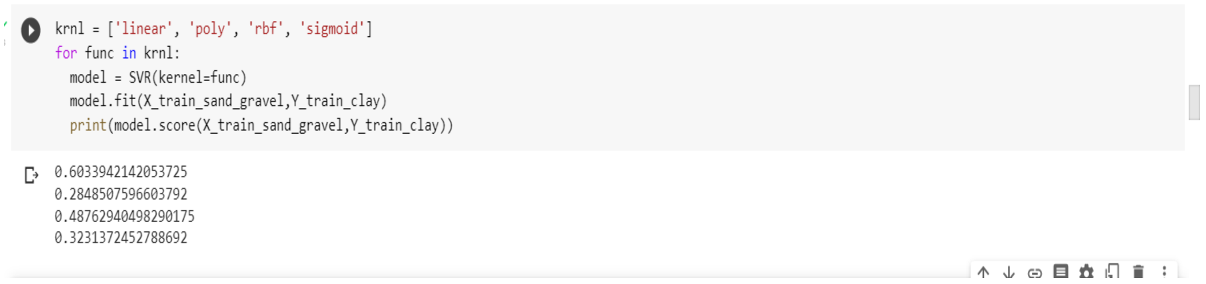
```
li =[.1,.2,.3,.4,.5,.6,.7,.8,.9,1]
for i in li:
  model = ElasticNet(alpha = i)
  model.fit(X_train_sand_gravel,Y_train_clay)
  print(i,model.score(X_train_sand_gravel,Y_train_clay))
```

```
0.1 0.6087596606166656
0.2 0.608757944092208
0.3 0.6087551042795696
0.4 0.6087511577829108
0.5 0.6087461209398679
0.6 0.6087400098263103
0.7 0.6087328402609999
0.8 0.6087246278101603
0.9 0.6087153877919533
1 0.6087051352808666
```

We, see here that linear models don’t perform well.

We use some other models -

Support Vector Regressor -> We use SVR with all the functions and obtain accuracies as listed below.

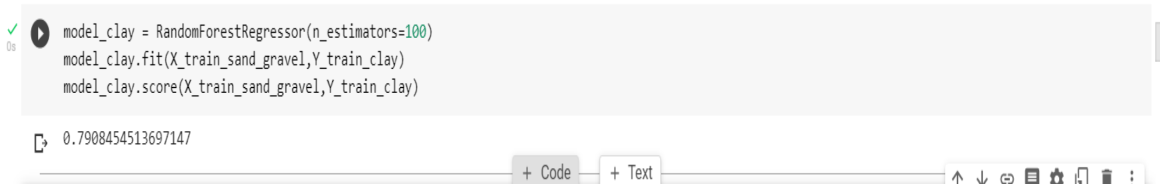


```
krnl = ['linear', 'poly', 'rbf', 'sigmoid']
for func in krnl:
  model = SVR(kernel=func)
  model.fit(X_train_sand_gravel,Y_train_clay)
  print(model.score(X_train_sand_gravel,Y_train_clay))
```

```
0.6033942142053725
0.2848507596603792
0.48762940498290175
0.3231372452788692
```

Support vector machine too perform badly on our data set with linear function giving best accuracy which is same as our linear models.

Last models where we get max accuracy is by RandomForestRegressor.

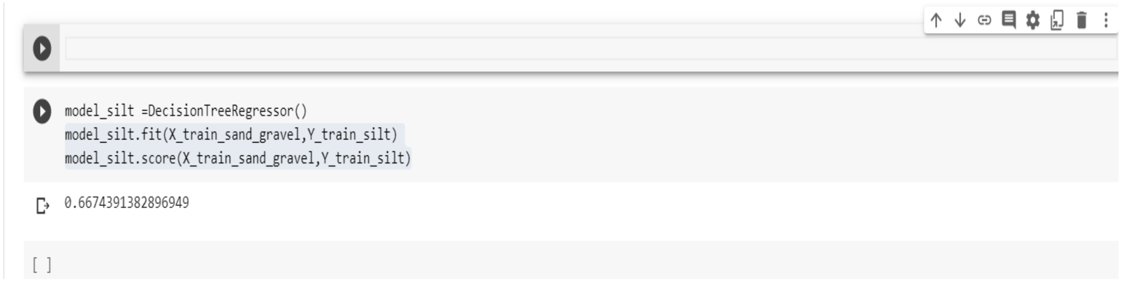


```
model_clay = RandomForestRegressor(n_estimators=100)
model_clay.fit(X_train_sand_gravel,Y_train_clay)
model_clay.score(X_train_sand_gravel,Y_train_clay)
```

```
0.7908454513697147
```

We will use this model to replace values for missing values is clay column.

Similarly, we try all the models for silt column and best model we get is DecisionTreeRegressor .

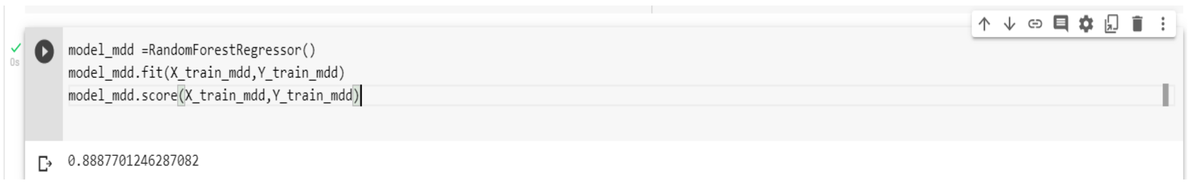


```
model_silt =DecisionTreeRegressor()
model_silt.fit(X_train_sand_gravel,Y_train_silt)
model_silt.score(X_train_sand_gravel,Y_train_silt)
```

```
0.6674391382896949
```

Now, taking plastic limit and liquid limit columns to predict values for maximum dry density column.

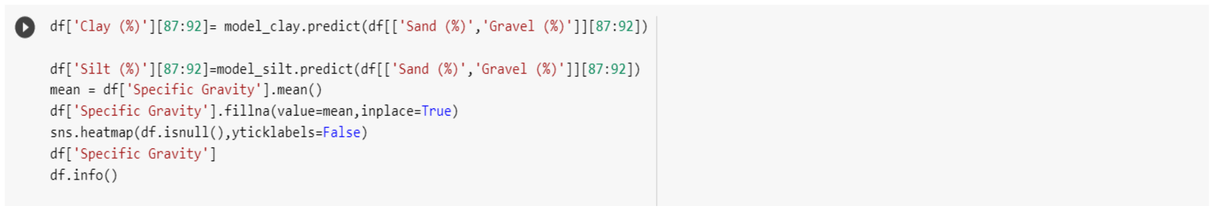
Testing few of our models the best model with 88% accuracy was found to be RandomForestRegressor.


```
model_mdd =RandomForestRegressor()
model_mdd.fit(X_train_mdd,Y_train_mdd)
model_mdd.score(X_train_mdd,Y_train_mdd)
```

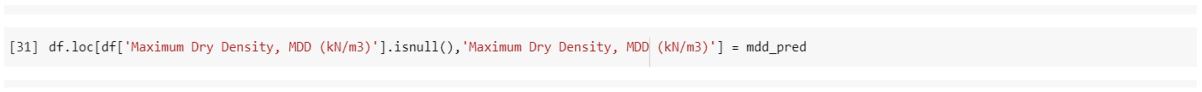
```
0.8887701246287082
```

As, the optimum moisture content and max dry density are highly correlated so we can drop optimum moisture content column as it will not provide any additional information to our model.

Now, finally replacing all the values by models which we made.


```
df['Clay (%)'][87:92]= model_clay.predict(df[['Sand (%)','Gravel (%)']][87:92])

df['Silt (%)'][87:92]=model_silt.predict(df[['Sand (%)','Gravel (%)']][87:92])
mean = df['Specific Gravity'].mean()
df['Specific Gravity'].fillna(value=mean,inplace=True)
sns.heatmap(df.isnull(),yticklabels=False)
df['Specific Gravity']
df.info()
```

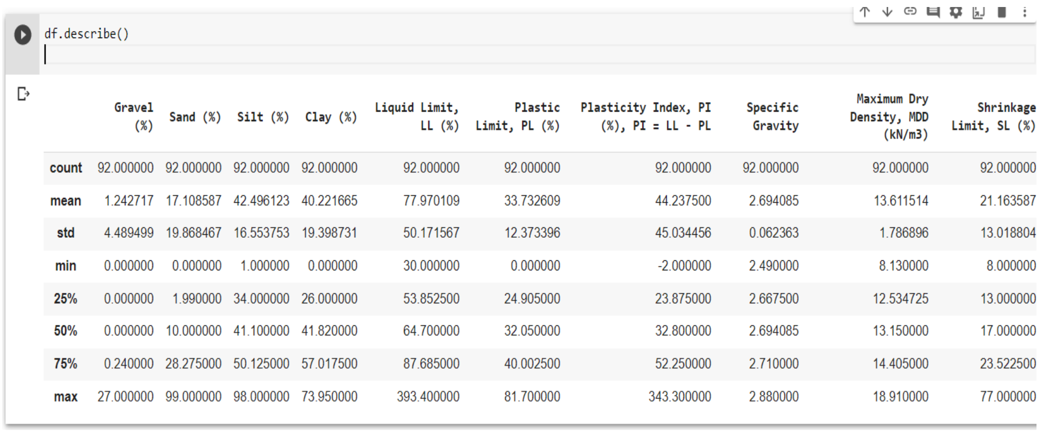

```
[31] df.loc[df['Maximum Dry Density, MDD (kN/m3)'].isnull(),'Maximum Dry Density, MDD (kN/m3)'] = mdd_pred
```

In above cells we have also replaced values in specific gravity column with the mean of values

Also, we have dropped the optimum moisture content column.

Now, our data is finally ready for further visualization and prediction.

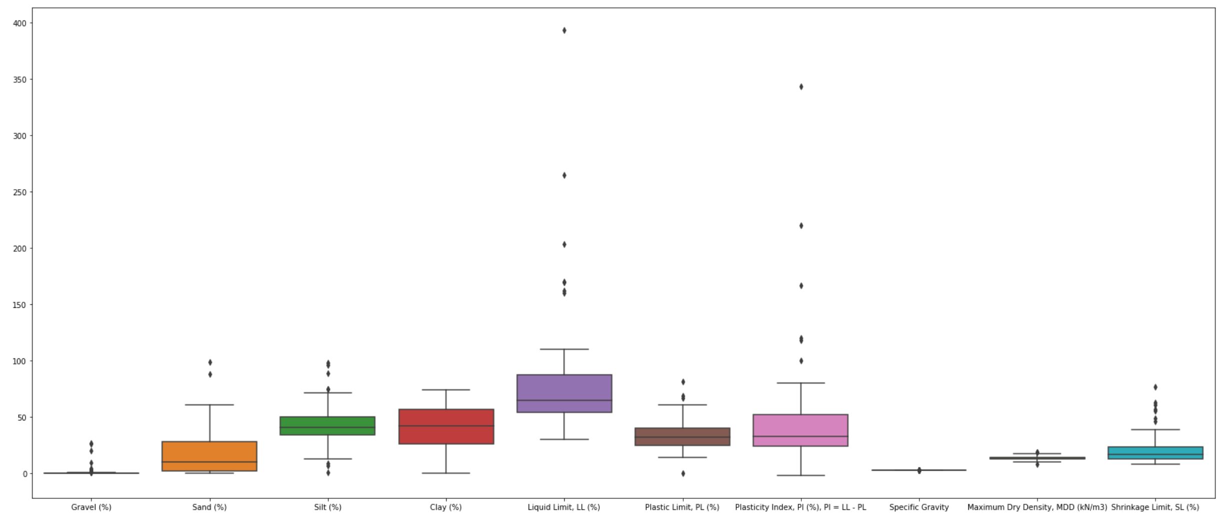
Let's, see some basis statistics for our data.


```
df.describe()
```

	Gravel (%)	Sand (%)	Silt (%)	Clay (%)	Liquid Limit, LL (%)	Plastic Limit, PL (%)	Plasticity Index, PI (%), PI = LL - PL	Specific Gravity	Maximum Dry Density, MDD (kN/m3)	Shrinkage Limit, SL (%)
count	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000	92.000000
mean	1.242717	17.108587	42.496123	40.221665	77.970109	33.732609	44.237500	2.694085	13.611514	21.163587
std	4.489499	19.868467	16.553753	19.398731	50.171567	12.373396	45.034456	0.062363	1.786896	13.018804
min	0.000000	0.000000	1.000000	0.000000	30.000000	0.000000	-2.000000	2.490000	8.130000	8.000000
25%	0.000000	1.990000	34.000000	26.000000	53.852500	24.905000	23.875000	2.667500	12.534725	13.000000
50%	0.000000	10.000000	41.100000	41.820000	64.700000	32.050000	32.800000	2.694085	13.150000	17.000000
75%	0.240000	28.275000	50.125000	57.017500	87.685000	40.002500	52.250000	2.710000	14.405000	23.522500
max	27.000000	99.000000	98.000000	73.950000	393.400000	81.700000	343.300000	2.880000	18.910000	77.000000

VISUALIZATION

To see how values of the data in different columns vary and also visualize their median, max min and outliers we plot a box plot for our data.

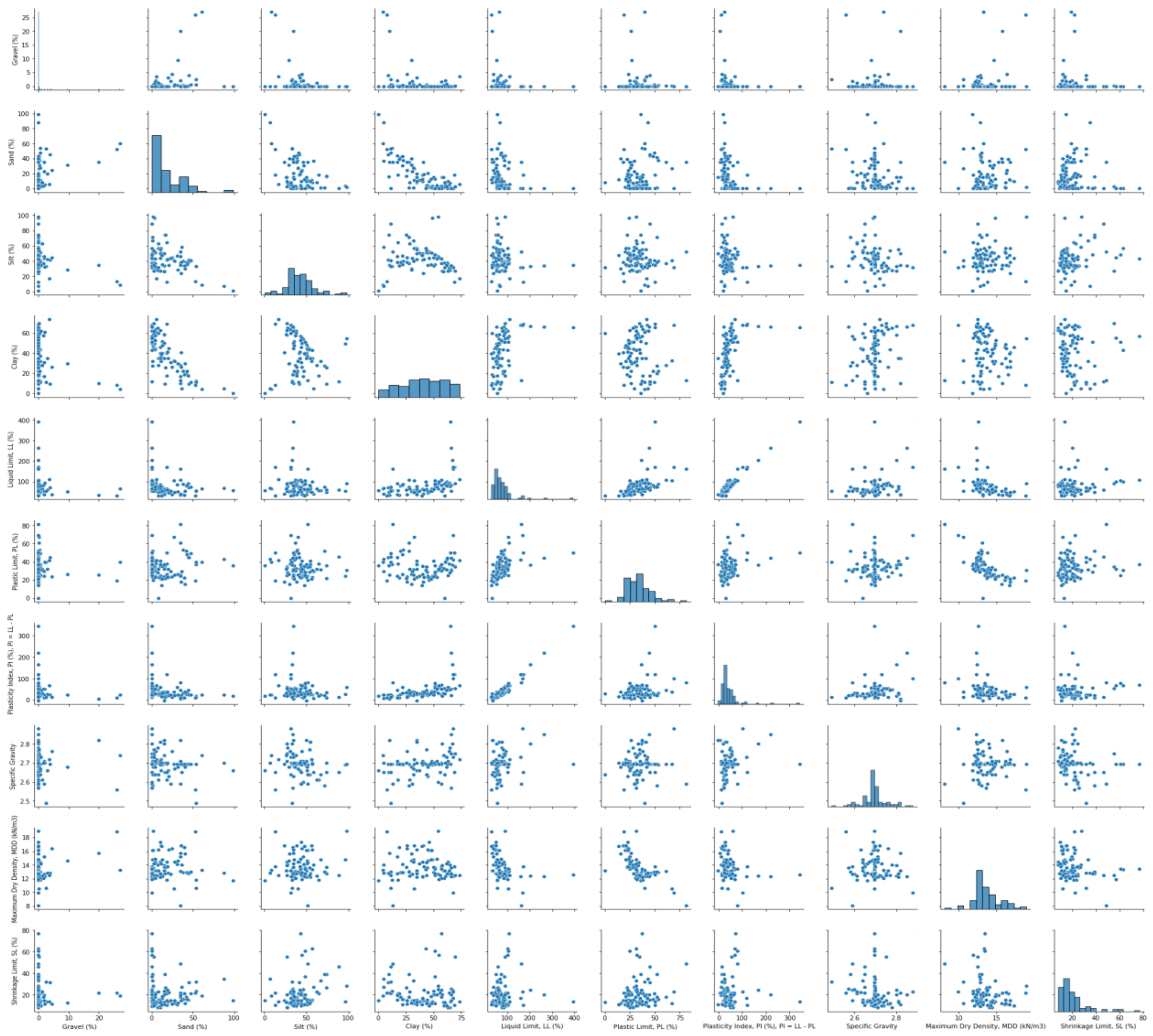


Gravel, maximum dry density and specific gravity columns are spread over a very little range of values.

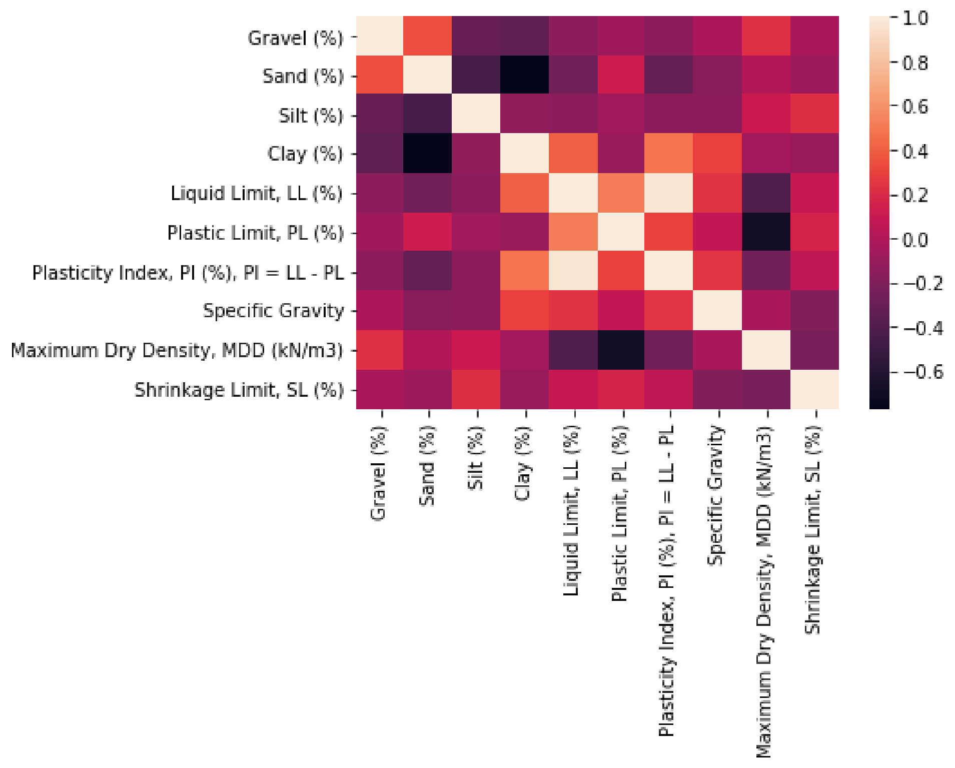
Liquid limit, plasticity index seems to have many outliers.

All other columns are uniformly distributed.

To observe values in a particular column and also to see if there’s a correlation between any two columns, we plot a pair plot among all the columns.

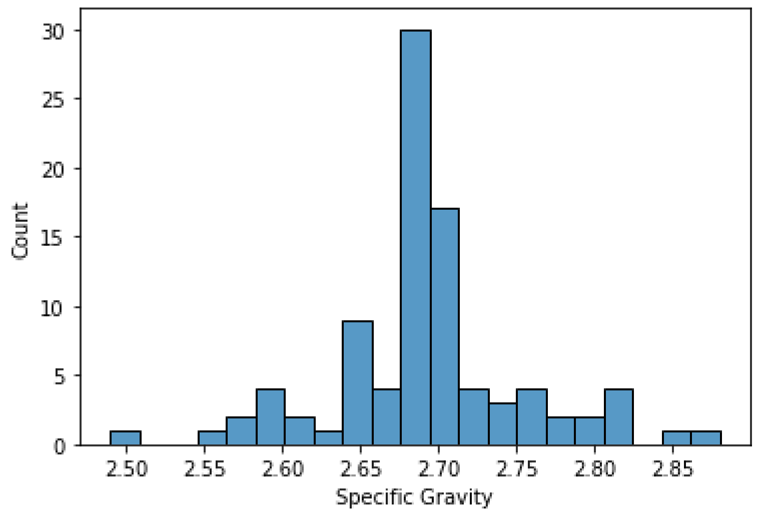


There’s no high correlation between any column and output column to see this more closely we can plot and heatmap of our reformed data.



As we see no column correlates with the output column shrinkage limit therefore it’s hard to create a model with good accuracy.

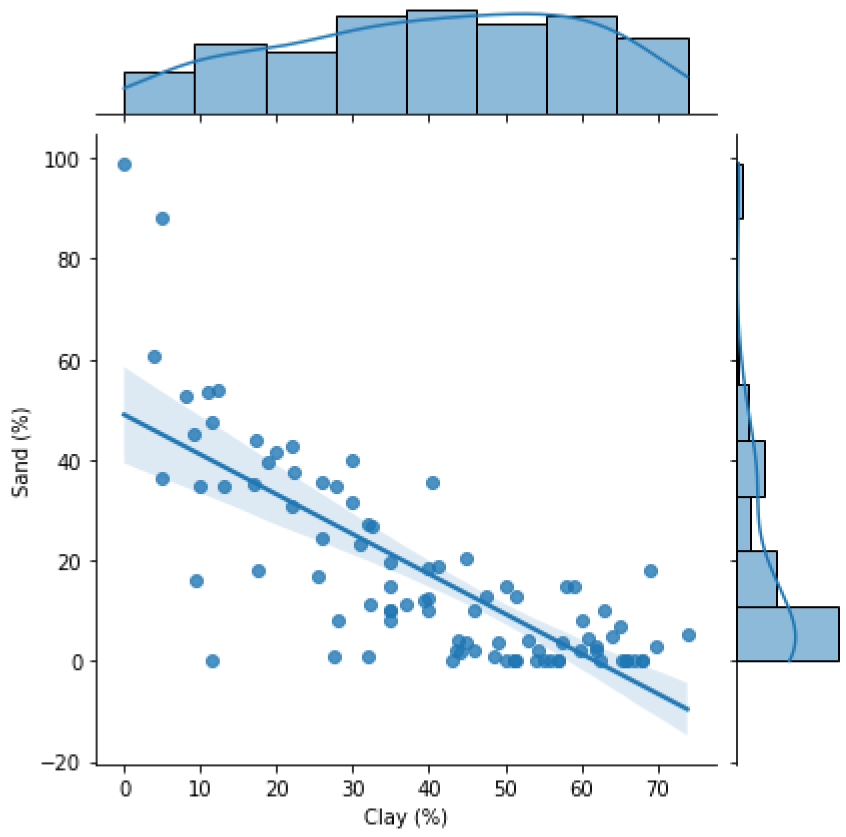
We have replaced missing values in specific gravity column with the mean so let’s see how the values in specific gravity column vary by plotting and histplot .



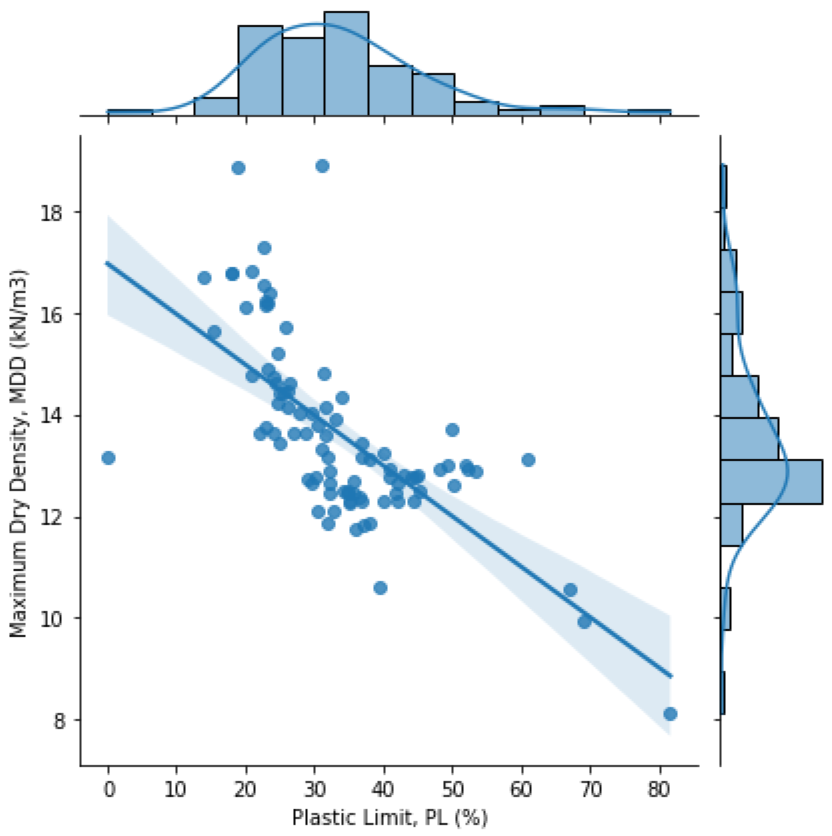
Seeing the graph our decision to replace value with mean was quite good.

During, visualizing pair plot we saw that some columns are correlated so now let’s make joint plot among them to see if we can relate them with a regression plot

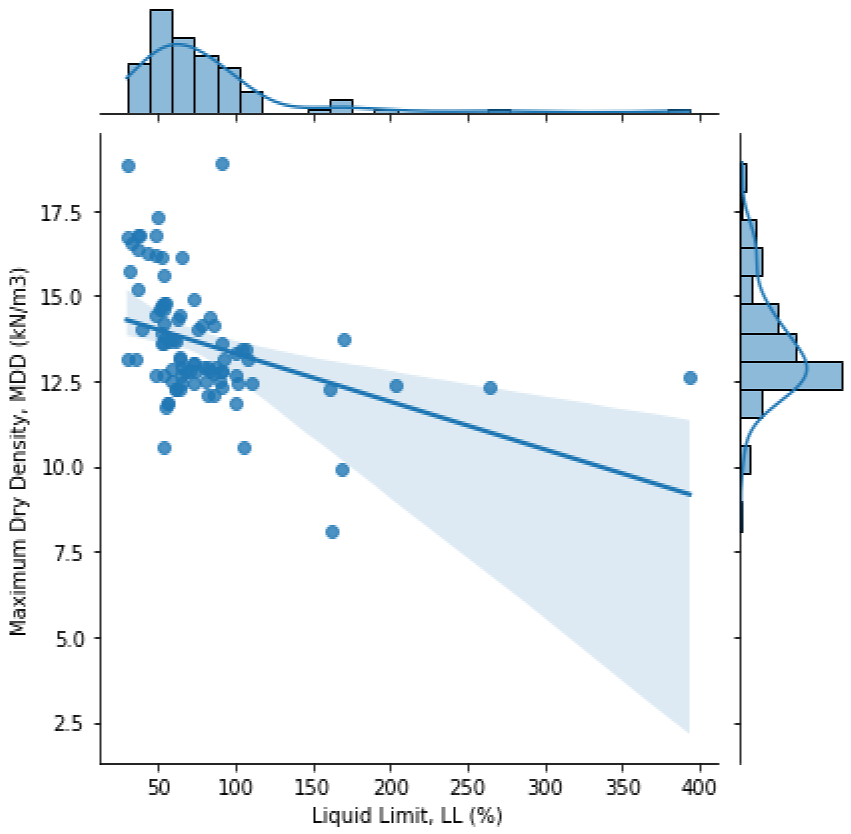
Sand and Clay



Finally, the two columns which we used to predict missing values of max dry density column.



Maximum Dry Density, MDD (kN/m3)
Plastic Limit, PL (%)
8
10
12
14
16
18
0
10
20
30
40
50
60
70
80



METHODOLGY AND MODELS FOR PREDICTION

We have used 5 models to predict the shrinkage limit for our data.

The model with least root mean squared error will be selected at last-

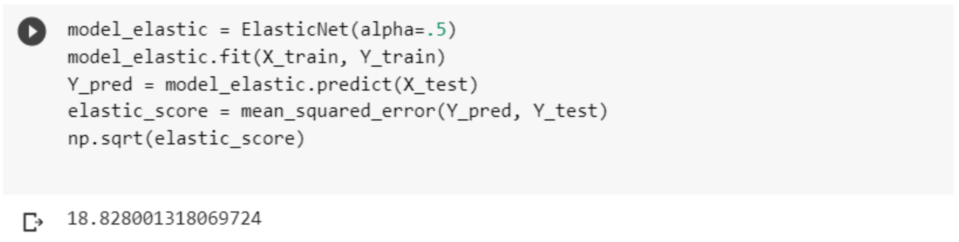
ELASTIC NET

Our first model will be elastic net which will combine linear model and regularization terms of ridge and lasso.

Let’s see what accuracy it provides to us.

We test our model for one value of alpha - .5.

If this will provide lower rmse than we will check for more values of alpha else we will move on to other models for prediction.



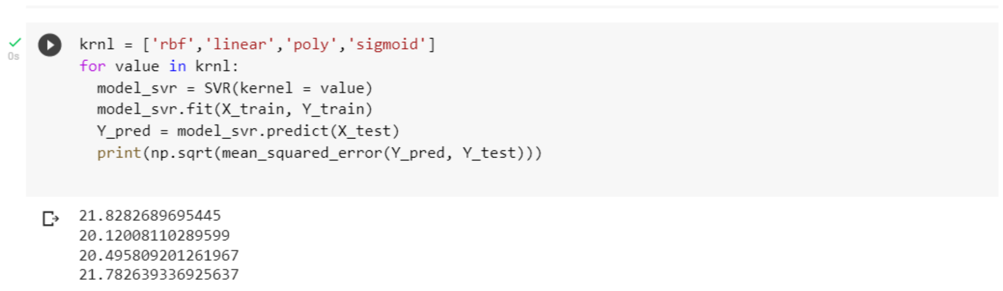
```
model_elastic = ElasticNet(alpha=.5)
model_elastic.fit(X_train, Y_train)
Y_pred = model_elastic.predict(X_test)
elastic_score = mean_squared_error(Y_pred, Y_test)
np.sqrt(elastic_score)
```

```
18.828001318069724
```

The model gives us very high rmse so we try different models instead of tuning it.

SUPPORT VETOR REGRESSOR

We try support vector regression with providing a list of kernel density functions and check which function gives lowest rmse.



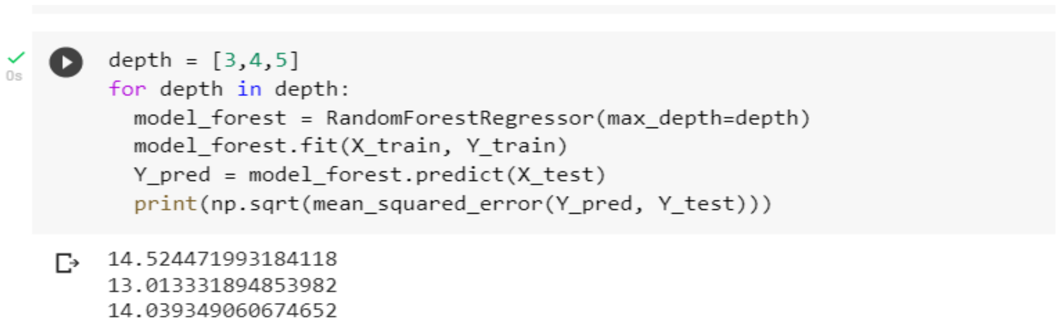
```
krnl = ['rbf','linear','poly','sigmoid']
for value in krnl:
  model_svr = SVR(kernel = value)
  model_svr.fit(X_train, Y_train)
  Y_pred = model_svr.predict(X_test)
  print(np.sqrt(mean_squared_error(Y_pred, Y_test)))
```

```
21.8282689695445
20.12008110289599
20.495809201261967
21.782639336925637
```

The support vector machine performs badly on this regression problem though among all functions linear functions performs somewhat better than others.

RANDOM FOREST REGRESOR

We try random forest regressor model with varying value for max depth for different tress.



```
depth = [3,4,5]
for depth in depth:
  model_forest = RandomForestRegressor(max_depth=depth)
  model_forest.fit(X_train, Y_train)
  Y_pred = model_forest.predict(X_test)
  print(np.sqrt(mean_squared_error(Y_pred, Y_test)))
```

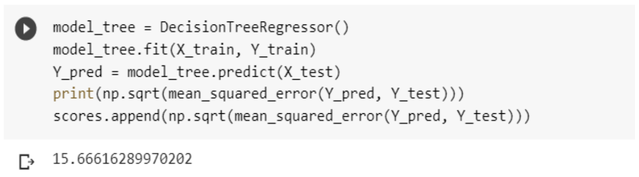
```
14.524471993184118
13.013331894853982
14.039349060674652
```

Model performs better than previous models with max depth as 4 it gives best result.

Let's try some more models to see if we can get more good accuracy.

DECISION TREE REGRESSOR

We build a decision tree regressor with default hyperparameters and if it will perform better than random forest than we will tune it's hyperparameter or else we will move on to next model.



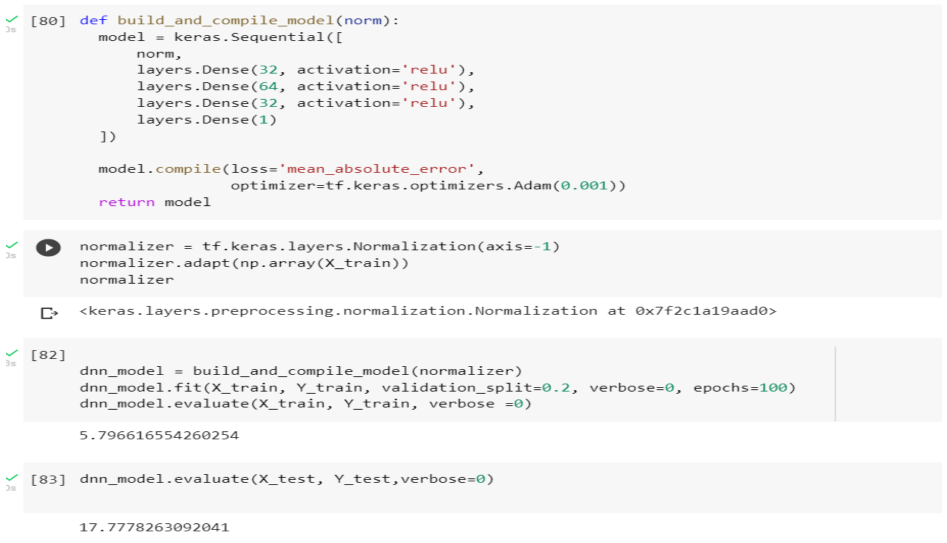
```
model_tree = DecisionTreeRegressor()
model_tree.fit(X_train, Y_train)
Y_pred = model_tree.predict(X_test)
print(np.sqrt(mean_squared_error(Y_pred, Y_test)))
scores.append(np.sqrt(mean_squared_error(Y_pred, Y_test)))
```

```
15.66616289970202
```

It performs better but not better than random forest so we will skip tuning it.

The last model we are going to use here will be deep neural network model with tuning it's layers and units to get lowest rmse.

First we build a model then we make a function that will normalize the input layers every time and finally we will fit our data into the model to get predictions.



```
[80] def build_and_compile_model(norm):
       model = keras.Sequential([
           norm,
           layers.Dense(32, activation='relu'),
           layers.Dense(64, activation='relu'),
           layers.Dense(32, activation='relu'),
           layers.Dense(1)
       ])

       model.compile(loss='mean_absolute_error',
                     optimizer=tf.keras.optimizers.Adam(0.001))
       return model
```

```
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.array(X_train))
normalizer
```

```
<keras.layers.preprocessing.normalization.Normalization at 0x7f2c1a19aad0>
```

```
[82]
dnn_model = build_and_compile_model(normalizer)
dnn_model.fit(X_train, Y_train, validation_split=0.2, verbose=0, epochs=100)
dnn_model.evaluate(X_train, Y_train, verbose =0)
```

```
5.796616554260254
```

```
[83] dnn_model.evaluate(X_test, Y_test,verbose=0)
```

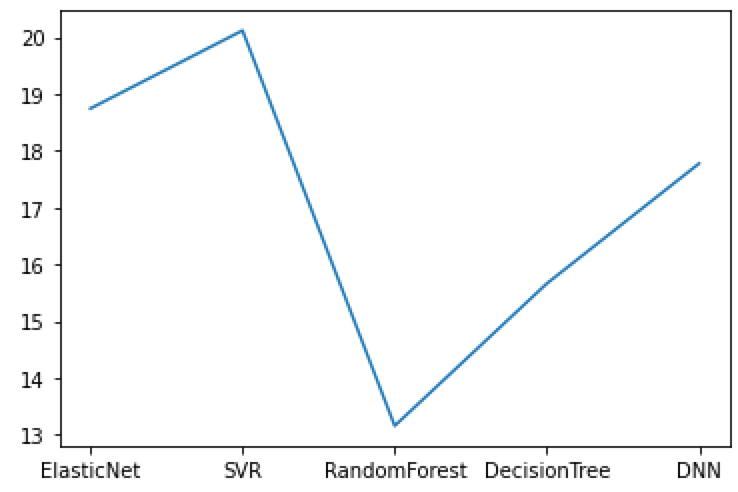
```
17.7778263092041
```

This also does not perform better than random forest.

So, we finalize our model to be random forest.

CONCLUSION

Below is graph to compare rmse given by different models.



As the given data does not correlates with the output column so it was very hard to get an modle which would get good accuracy.

So, we get random forest regressor model with rmse of near 13 as lowest rmse.