Project 3: Concrete Strength Prediction

This manuscript ([permalink](#)) was automatically generated from [qinyuz2/project3@b14afff](#) on December 7, 2020.

Authors

- **Qinyu Zhang**
  · ⓖ [qinyuz2](#)
  Department of Civil & Environmental Engineering, University of Illinois at Urbana-Champaign

- **Sonali Srivastava**
  · ⓖ [Sonali1610](#)
  Department of Civil & Environmental Engineering, University of Illinois at Urbana-Champaign

- **Pratyush Kumar**
  · ⓖ [pratyushkr9420](#)
  Department of Civil & Environmental Engineering, University of Illinois at Urbana-Champaign

1. Introduction

The second most consumed substance in the world after water is concrete. Currently, the world produces 4.4 billion of concrete annually. Conventional concrete is a mixture of cement, aggregates (coarse and fine) and water. Many admixtures like mineral and chemical are incorporated in concrete to improve its performance. The main idea is not only to improve its overall performance and durability but also to reduce the emission of carbon dioxide produced by concrete industry itself. Cement production is highly energy intensive process. In 2015, it generated around 2.8 billion of CO^2 (8% of total). Lots of research have been carried out to decrease the percentage of cement in concrete by using different substitutes. Researchers are opting for more environment friendly and sustainable options. This new idea of unconventional concrete requires backup and standards. The conventional method of testing concrete's strength is to cast concrete cylinders or cubes with different mix ratio of its constituents and test them after 7, 14 or 28 days. This method requires significant amount of labor and time. Also, small human error in designing or preparing can lead to drastic change in strength and increase the waiting time.

Recently, researchers are developing models using machine learning and artificial neural network to predict the compressive strength of concrete. This is very useful in predicting the performance using complex non-linear relations.

1.1 Problem Statement

Since conventional methods for checking the performance of concrete are time consuming and prone to human error. This project aims to predict the compressive strength of concrete using different emerging soft computer techniques like Linear Regression, Lasso Regression, Artificial Neural Network, Random Forest, and Decision Tree Regression. A comparison is made using the root mean square error and model with better performance was chosen.

The dataset was collected from UCI Machine Learning Repository. It has total 1030 instances with 9 attributes (8 quantitative input variable and 1 quantitative output variables). The eight independent variables are: cement (kg/m3), blast furnace slag (kg/m3), fly ash (kg/m3), water (kg/m3),

superplasticizer (kg/m3), coarse aggregates (kg/m3), fine aggregates (kg/m3), and age of curing (days). The dependent/output variable is compressive strength in MPa.

1.2 Background

Machine learning (ML) algorithm finds patterns in massive amount of dataset and predicts the output. It provides a system which has ability to learn automatically and improve the performance from experiences. It builds a model based on training dataset and predict the output variable for test dataset. Traditionally, ML is divided into three categories: supervised, unsupervised and reinforcement learning. With the given dataset, we used supervised learning technique because output of trained dataset was provided, and the overall goal was to learn and map inputs to the output.

Models are required for performing machine learning and there are various of models that can have been researched for learning systems. Artificial neural network is one of them. It is based on the behavior of neurons. It consists of a set of interconnected neurons arranged in layers. Typically, there are three types of layers: input layer, hidden layer, and output layer. Each neuron in input layer corresponds to the input variables. The outputs are obtained through neurons in output layer. Hidden layers take the weighted inputs, perform the regression, and produce outputs. Other methods like Linear Regression and Lasso assume linear relationship between input and output variables. To avoid the overfit, regularization is performed. Lasso regularization is one of them which modifies the ordinary least square error by penalizing the regression method. Decision tree method uses tree-like model to deliver the final output. Random decision forest is ensemble learning technique which uses Bootstrap Aggregation. The main idea is to combine multiple decision tree to find the final output.

1.3 Literature Review

Being a highly non-linear materials, modelling concrete's behavior in terms of its strength and durability is a difficult task. Free models – Mathematical models based on experimental data – are generally recommended and have been used widely. Neural network (NN) models for predicting compressive strength of light weight concrete mixtures for 3, 7, 14 and 28 days of curing was used in this article [1]. The data were collected by performing compressive strength test for different concrete mix. Cement, silica fumes, light weight coarse aggregate, light weight fine aggregate and water were used in different proportions to cast concrete cubes (150 mm x 150 mm x 150 mm). Two training processes – back propagation (BP) and cascade correlation (CC) were used. While training the models, minimum mean square error (MSE) corresponds to the most stable system. Back propagation algorithm fine tunes the weights of input neural network based on MSE obtained from previous iteration. However, cascade correlation does not just adjust weights of fixed number of hidden layers. It starts with one hidden layer to train automatically and add new layers one by one by keeping the weights frozen from pervious training. Preprocessing of data set were performed and, 70% and 30% were randomly chosen for training and validation, respectively. The assessments were performed using for indices – Mean absolute error (MAE), root mean square error (RMSE), correlation coefficient R, and coefficient of efficiency Ef. Eight inputs provided in input layers were: sand, water to cement ratio, light weight fine aggregate, light weight coarse aggregate, silica fumes, superplasticizer and curing period and, four outputs were the compressive strength of concrete mix after 3, 7, 14 and 28 days. The parametric study was performed by changing the different number of hidden layers for BP training process. The most stable network was found when there were 14 neurons in one hidden layer. Furthermore, additional hidden layer was added to improve the performance. 14 layers in one hidden layer (based on previous training) and another layer with 6 neurons gave the optimum results. This is similar to the results found by Maier and Dandy [2] that first and second hidden layers should not exceed 3:1 ratio for maximum accuracy. Values of assessment indices are summarized in the following table:

Table: Mean absolute (MAE) and correlation coefficient R for BP (8-14-6-4) and CC model

Method	MAE	R
BP(18-14-6-4)		
Trained data	2.22%	0.972
Testing data	1.987%	0.977
CC		
Trained data	2.22%	0.974
Testing data	1.797%	0.982

This shows that both NN models provided similar results. But it was observed that CC was better than BP (8-14-6-4) model as it learns quickly and determine the size and topology on its own. These analyses have huge potential in predicting the properties of concrete and avoid expensive and time-consuming tests. The problem statement that the paper deals with is the issue of predicting the behavior of High-Performance Concrete (HPC) [3] The paper argues that the problem of difficulty in the prediction of HPC behaviors can be solved by the use of Artificial Neural Networks (ANN). The research describes ANN's a collective of parallel architecture that can be used to solve critical problems by cooperation and interconnection with simple computing elements called neurons. The target behind creating the ANN in the research was to have a model which can efficiently predict the strength of different HPC mixes. The main principle on which neural networks work as back-propagation and gradient descent. The data set used to train the model consisted to experiments involving strength experiments and predictions. The concept was to train the model on the set of experiments which describe material behavior and then create a successful neural network which can then serve as a model to predict strength of HPC mixes in general. There are eight different parameters which the research takes to initialize the network they are: quantities of cement, fly-ash, blast-furnace slag, water, super-plasticizer, and fine aggregate. The model was trained from a dataset complied from 17 different sources and in total 727 records of strength experimentation data was used to train the model. The division of the data was done into four sets A, B, C and D out of which sets of 3 were used to train the model and 1 set was used to test the strength predictive ability of the model. The success of the model is highlighted from the fact that the correlation parameter between strength predictions and variables in a model was much closer to 1 than the standard regression model traditionally used to test the model.

Another article on prediction of the strength of concrete with different types of mineral admixtures was reviewed [4]. The article used the enter techniques in the Statistical Package for Social Science (SPSS) to develop the regression models. A total of six models were developed using different combinations of variables. After F-test, t-test and Durbin–Watson test, Model 4-6 past the validity assessment. When applying the artificial neural network method, the same dataset was used. Seventy percent, fifteen percent and fifteen percent of the data set were used in the learning, validation and testing phases respectively and they used the ANN toolbox (nftool) in Matlab to process computation. The article utilized a back-propagation training algorithm in a two-layer feed-forward network trained using the Levenberg–Marquardt algorithm. In the hidden layer, a nonlinear hyperbolic tangent sigmoid transfer function was applied, and in the output layer, a linear transfer function was applied. To determine the number of neurons in the hidden layer, they compared the predicted results produced by models trained with different numbers of hidden neurons with the desired output. Models were trained through multiple iterations. A total of six models were developed, and the first five proved to be reliable to predict compressive strength of concrete.

Compared to the linear regression method, ANN had advantages of constructing the relationships among input and output parameters automatically, especially for calculating nonlinear functional

relationships. The article verifies the effectiveness of Artificial Neural Networks in predicting concrete strength and is helpful for our own project. However, one weakness is that the training database used in the article is not big enough. If an Artificial Neural Network model with wide applicability was wanted, a larger database is needed.

2. Method

2.1 Exploratory Data Analysis

It is done in order to analyze the data used for the project more comprehensively. In order to do so, the first step is to import the required libraires. And load the training, test and sample data.

Importing required libraries

```python
In [1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

EDA-1

Loading Dataset

Sample, Two dataset (**test** and **train**) have been created. **Train** dataset will be used to train our model and test dataset will be used to check the model.

```python
In [2]:
sample= pd.read_csv("/kaggle/input/cee-498-project3-forecast-strength-of-concrete/sample.csv")
test= pd.read_csv("/kaggle/input/cee-498-project3-forecast-strength-of-concrete/test.csv")
train = pd.read_csv("/kaggle/input/cee-498-project3-forecast-strength-of-concrete/train.csv")
```

EDA-2

Description of train dataset

Exploratory Data Analysis will be carried out on **train** dataset

In [3]:
```
train.describe()
```

Out[3]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Fine Aggregate (component 7)(kg in a m^3 mixture)	Age (day)	Concrete compressive strength(MPa, megapascals)
count	707.000000	707.000000	707.000000	707.000000	707.000000	707.000000	707.000000	707.000000	707.000000
mean	280.970325	76.519250	51.837949	182.347709	6.180854	971.419703	772.372588	47.619519	36.156560
std	105.051424	86.971542	63.004123	21.844600	6.029105	77.425959	81.508478	65.464982	16.687748
min	102.000000	0.000000	0.000000	121.750000	0.000000	801.000000	594.000000	1.000000	4.827711
25%	194.090000	0.000000	0.000000	164.900000	0.000000	932.000000	723.400000	14.000000	24.023412
50%	275.000000	26.000000	0.000000	185.700000	6.350000	967.080000	779.320000	28.000000	34.770275
75%	350.000000	144.100000	116.000000	193.000000	10.300000	1028.400000	824.000000	56.000000	46.070786
max	540.000000	359.400000	200.000000	247.000000	32.200000	1134.300000	992.600000	365.000000	82.599225

EDA-3

Rename Columns Name

In [4]:
```
train1= train.rename(columns={"Cement (component 1)(kg in a m^3 mixture)": "1_cement",
                    "Blast Furnace Slag (component 2)(kg in a m^3 mixture)": "2_BFS",
                "Fly Ash (component 3)(kg in a m^3 mixture)": "3_Flyash",
                "Water  (component 4)(kg in a m^3 mixture)": "4_water",
                "Superplasticizer (component 5)(kg in a m^3 mixture)": "5_SP",
                "Coarse Aggregate  (component 6)(kg in a m^3 mixture)": "6_CA",
                "Fine Aggregate (component 7)(kg in a m^3 mixture)": "7_FA",
                "Age (day)": "Age",
               "Concrete compressive strength(MPa, megapascals) " : "strength" }, errors="raise")
train1.head()
```

Out[4]:

|	1_cement	2_BFS	3_Flyash	4_water	5_SP	6_CA	7_FA	Age	strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
3	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075
4	266.0	114.0	0.0	228.0	0.0	932.0	670.0	90	47.029847

EDA-4

Box Plot

Box plot analysis was performed:

Box plots

```
In [8]:   plt.subplots(figsize=(12, 8))
          ax = sns.boxplot(data=data)
          ax.set_xticklabels(ax.get_xticklabels(), rotation=90);
```
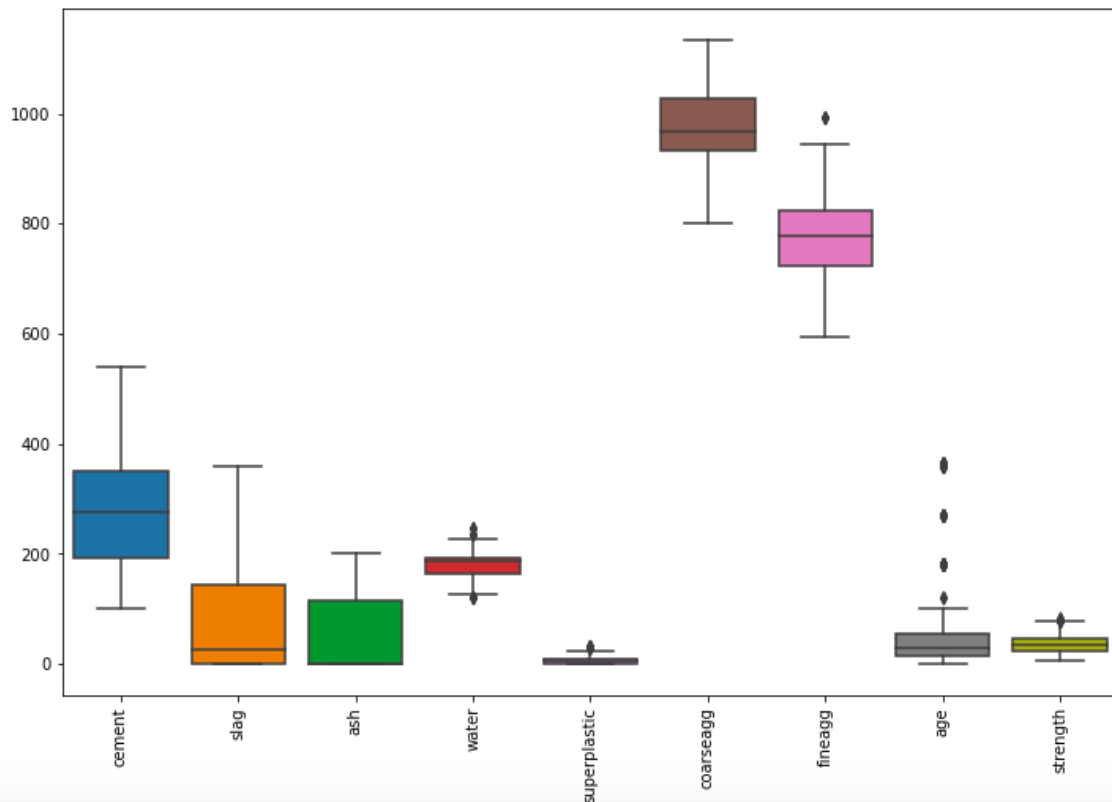

EDA-6

The observations and comments that can be made from the above box plot is as follows:

- The data in cement, slag, ash doesn't appear to have any significant outliers.

- The data from water, superplastic, age, and strength appears to have some outliers, amidst them the age data has a long extension of the plot suggesting the highest amount of outliers amongst all of data columns mentioned.

Distribution

Dsitribution of the variables in the dataset have been plotted to gain better understanding.

Distribution of independent variables

In [9]:
```python
import itertools

cols = [i for i in data.columns if i != 'strength']

fig = plt.figure(figsize=(15, 20))

for i,j in itertools.zip_longest(cols, range(len(cols))):
    plt.subplot(4,2,j+1)
    ax = sns.distplot(data[i],color='orange',rug=True)
    plt.axvline(data[i].mean(),linestyle="dashed",label="mean", color='black')
    plt.legend()
    plt.title(i)
    plt.xlabel("")
```
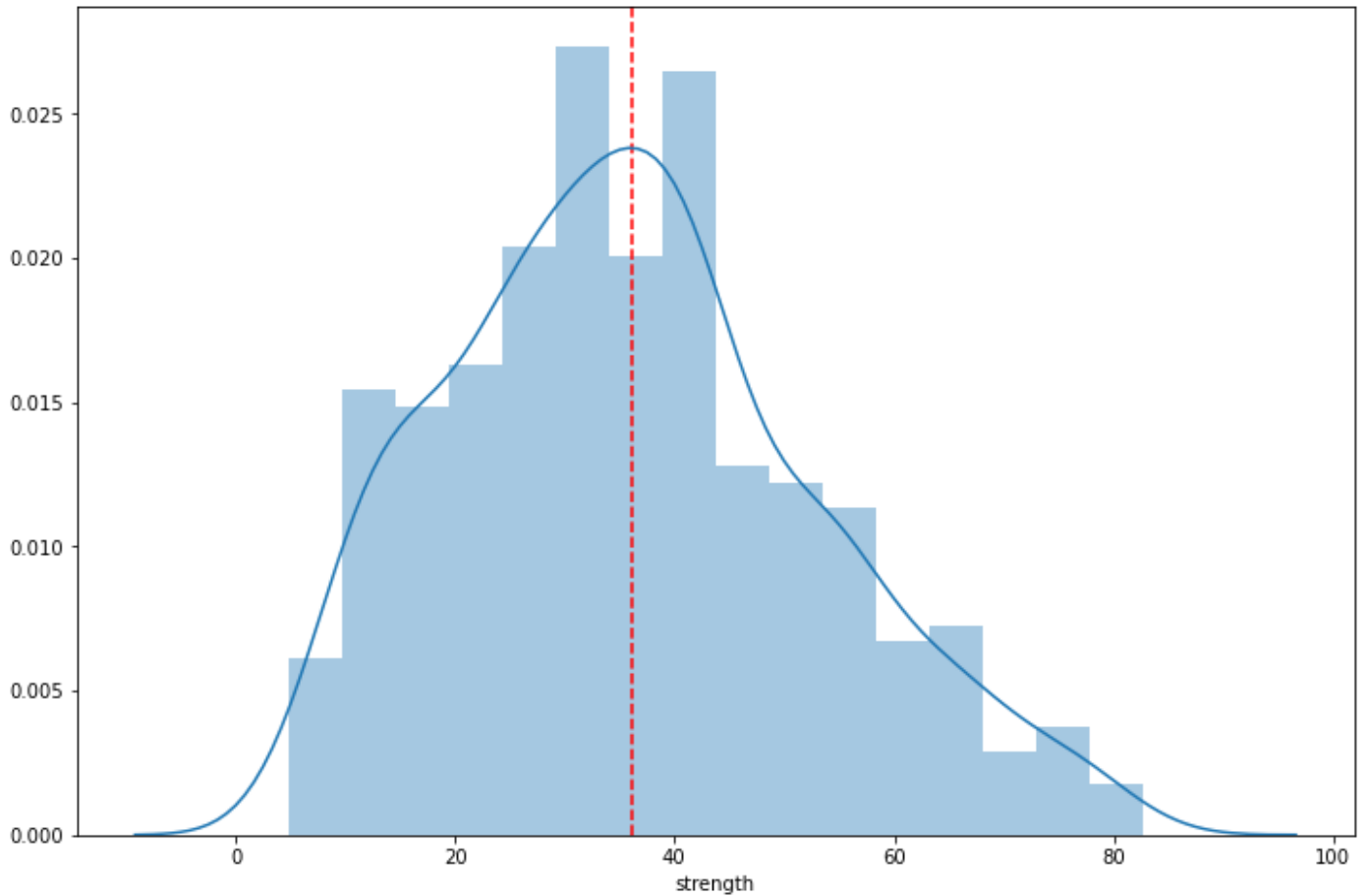
EDA-8

EDA-9

Distribution plot for strength variable

EDA-10

Degree of skewness

```
In [11]:    from scipy.stats import skew
            numerical_features = data.select_dtypes(include=[np.number]).columns
            categorical_features = data.select_dtypes(include=[np.object]).columns
            skew_values = skew(data[numerical_features], nan_policy = 'omit')
            dummy = pd.concat([pd.DataFrame(list(numerical_features), columns=['Features']),
                        pd.DataFrame(list(skew_values), columns=['Skewness degree'])], axis = 1
            )
            dummy.sort_values(by = 'Skewness degree' , ascending = False)
```

Out[11]:

	Features	Skewness degree
7	age	3.153869
4	superplastic	0.951027
1	slag	0.754229
2	ash	0.592097
0	cement	0.500008
8	strength	0.410593
3	water	0.089452
5	coarseagg	0.015180
6	fineagg	-0.269003

EDA-11

The observations and comments that can be made from the distribution plots and skewness degreee data is as follows:

- The strength data is normally distributed.

- Water and cement data seems to be very near to being normally distributed.

- The data from age column as seen from box plot had a lot of outliers which is re-affirmed here with the distribution plot having very lengthy un-symmetrical extension beyond its mean value.

Pair Plot

Pair plot

```
In [12]:
g = sns.PairGrid(data)
g.map_upper(plt.scatter)
g.map_lower(sns.lineplot)
g.map_diag(sns.kdeplot, lw=3, legend=True);
```

EDA-12
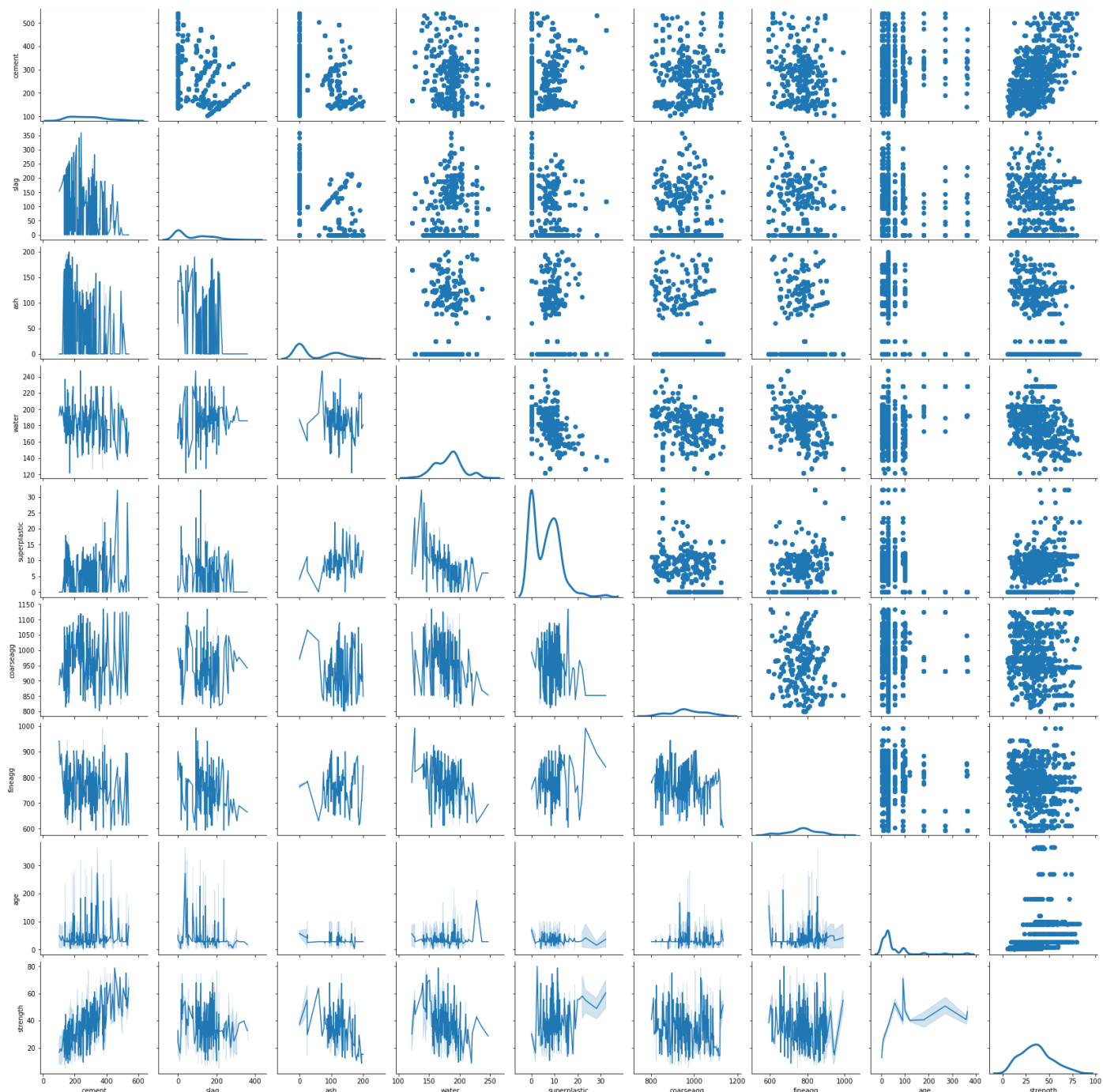
EDA-13

The observations and comments that can be made from the pair plots is as follows:

- There is strong positive correlation between cement and strength which seems theoretically consistent.

- In addition age also has a strong positive correlation to strength.

- Water and strength have a negative correlation which aagain seems theoretically consistent.

- Water and superplastic have a negeative correlation.

- Slag, ash, coarseagg and fineagg are having poor correlation to strength so they aren't the best predctors of strength.
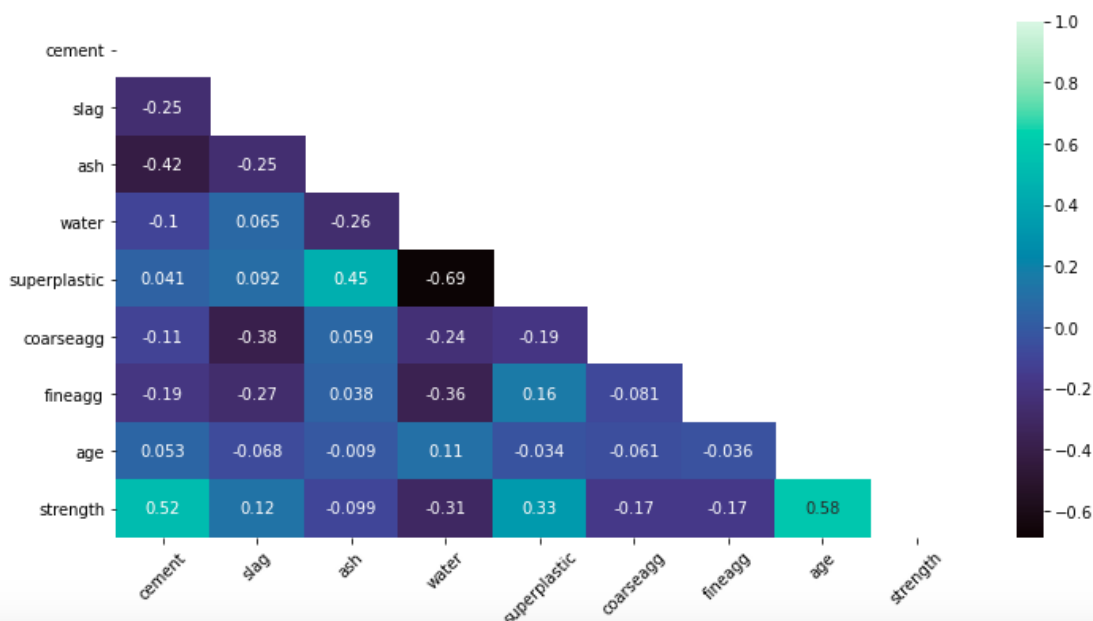
Heat Map

Heat map

```
In [13]:    plt.subplots(figsize=(12, 6))
            corr = data.corr('spearman')

            mask = np.zeros_like(corr)
            mask[np.triu_indices_from(mask)] = True

            ax = sns.heatmap(data=corr, cmap='mako', annot=True, mask=mask)
            ax.set_xticklabels(ax.get_xticklabels(), rotation=45);
```


EDA-14

The observations and comments that can be made from the heat map is as follows:

- Cement and age have strong correlation with strength

- Water and superplastic have strong correlation

- Superplastic has somewhat smaller but a postive correlation with strength

2.2 Independent Variable and Dependent Variable

There are a total of eight independent variables used in the prediction which are cement, blast furnance slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate and age. The dependent variable is the concrete compressive strength, seen in the Table below.

Table: Independent Variable and Dependent Variable ||| |:—-|:———-| |**Independent Variables**|Cement| ||Blast furnance slag| ||Fly ash| ||Water| ||Superplasticizer| ||Coarse aggregate| ||Fine aggregate| ||Age| |**Dependent Variable**|Concrete compressive strength|

2.3 Method Selected

According to exploratory data analysis, what we're working with is tabular data and there's no missing value. Also, all the data is numerical. We use seven different methods.

- Linear Regression
- Lasso Method
- K-nearest Neighbor
- Support Vector Machine
- Neural Network
- Decision Tree Regressor
- Random Forest Regression

Since we don't have the actual concrete strength data in the given test dataset, we are going to split 20% of the train dataset to test models in order to evaluate the model performance. Based on the evaluation, we choose the best method. And then create a model using the whole train dataset to predict the test dataset and submit the result to the Kaggle Competition.
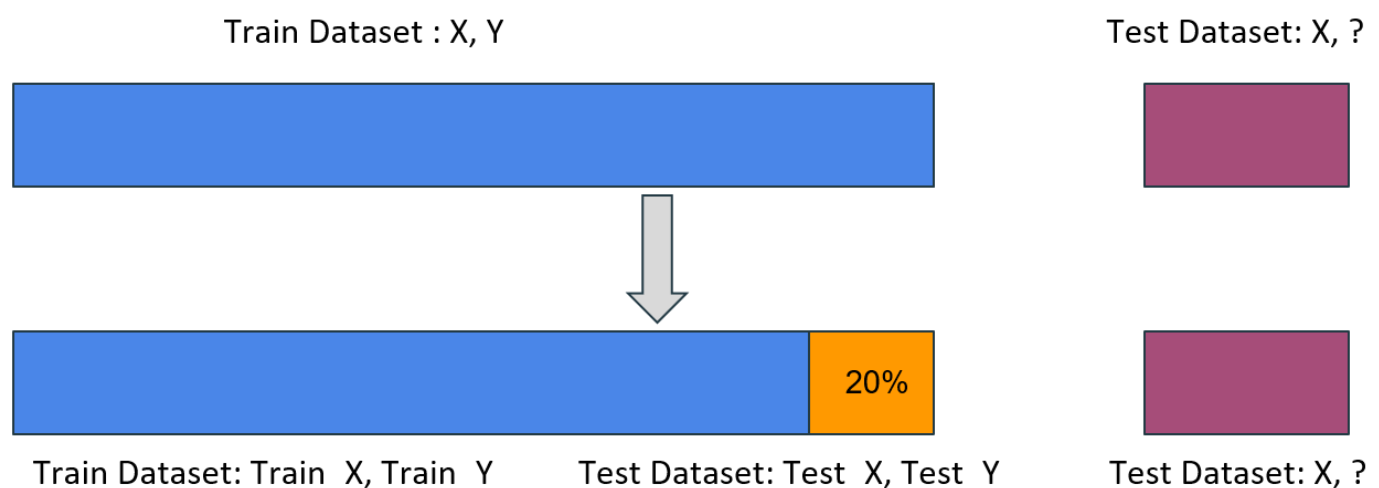

Figure: Split the Train Dataset

3. Discussion

3.1 Model Training and Evaluation

3.1.1 Linear Regression

Linear Regression is the simplest but powerful model. In the previous studies, it was widely used in the prediction of concrete strength. This model assumes a linear relationship between independent and dependent variables.

Code for Linear Regression Model

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(train_x, train_y)
y_pred_lin= lin_reg.predict(test_x)
```

Accuracy of Model is: 0.5177053629131334

Root Mean Squared Error of Model is: 11.420285520195613

3.1.2 Lasso Method

The Lasso is a shrinkage and selection method for linear regression. It minimizes the usual sum of squared errors, with a bound on the sum of the absolute values of the coefficients.

Code for Lasso Model

```
from sklearn.linear_model import Lasso

las = Lasso(alpha=0.1)
model2 = las.fit(train_x, train_y)
predictions2 = las.predict(test_x)
```

Accuracy of Model is: 0.38967572787640603

Root Mean Squared Error of Model is: 11.988560504390488

3.1.3 K-nearest Neighbor

The k-nearest neighbor is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems.

Code for K-nearest Neighbor Model

```
from sklearn.neighbors import KNeighborsRegressor

knn = KNeighborsRegressor()
model3=knn.fit(train_x, train_y)
predictions3 = knn.predict(test_x)
```

Accuracy of Model is: 0.34345271643724284

Root Mean Squared Error of Model is: 12.434253663809674

3.1.4 Support Vector Machine

Support vector machine is a supervised machine learning algorithm used for classification, regression and outlier detection. We use a linear Support Vector Machine model.

Code for Support Vector Machine Model

```python
from sklearn.svm import SVR


svm= SVR(kernel='linear')
model4=svm.fit(train_x, train_y)
predictions4 = svm.predict(test_x)
```

Accuracy of Model is: 0.34808748601553163

Root Mean Squared Error of Model is: 12.390287319386937

3.1.5 Neural Network

Neural Nework is also used in previous studies to predict concrete strength. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data. A neural network consists of input layer, hidden layers and output layer. It's able to learn and model non-linear and complex relationships between independent and dependent variables.

Code for Neural Network Model

```python
import tensorflow as tf

layer_width = 128
l1 = 0.0
l2 = 0.05


model1_split = tf.keras.Sequential()

model1_split.add(tf.keras.layers.Dense(512, activation="relu",
        kernel_regularizer = tf.keras.regularizers.l1_l2(l1=l1, l2=l2)))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(256, activation="relu"))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(128, activation="relu"))
model1_split.add(tf.keras.layers.BatchNormalization())
model1_split.add(tf.keras.layers.Dense(128, activation="relu"))

model1_split.add(tf.keras.layers.Dense(32, activation="relu"))

model1_split.add(tf.keras.layers.Dense(1))

model1_split.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0015),
                loss='mean_squared_error',
                metrics=[tf.keras.metrics.RootMeanSquaredError()]
                )

history_split = model1_split.fit(train_x, train_y, batch_size=1000,
                epochs=500, shuffle=True)
epochs_split = history_split.epoch
hist_split = pd.DataFrame(history_split.history)
rmse_split = hist_split["root_mean_squared_error"]

model1_split.summary()

model1_split_prediction = model1_split.predict(test_x)
```

Accuracy of Model is: 0.8565038038524211

Root Mean Squared Error of Model is: 6.229324285295386

3.1.6 Decision Tree Regressor

Decision-tree algorithm is a kind of supervised learning algorithms. It can be used in classification and regression problems.

Code for Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

dtregressor = DecisionTreeRegressor(random_state = 0, min_samples_split=5)
dtregressor.fit(train_x, train_y)
y_pred_dt= dtregressor.predict(test_x)
```

Accuracy of Model is: 0.8204795438030693

Root Mean Squared Error of Model is: 6.9675118677407

3.1.7 Random Forest Regression

Random Forest Regression is a type of supervised learning algorithms. It constructs multiple decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the prediction accuracy and it controls over-fitting as well.

Code for Hyperparameter Tuning

```python
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

#Use the random grid to search for best hyperparameter
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num =
        10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

# First create the base model to tune
rf_split = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random_split = RandomizedSearchCV(estimator = rf_split,
        param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2,
        random_state=None, n_jobs = -1)
# Fit the random search model
rf_random_split.fit(train_x, train_y)
```

The best parameters for the model is:

{'n_estimators': 1200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto', 'max_depth': 100, 'bootstrap': True}

Code for Random Forest Model

```
# create a model
model2_split = RandomForestRegressor(n_estimators =
        rf_random_split.best_params_['n_estimators'],
                        min_samples_split =
        rf_random_split.best_params_['min_samples_split'],
                        min_samples_leaf =
        rf_random_split.best_params_['min_samples_leaf'],
                        max_features =
        rf_random_split.best_params_['max_features'],
                        max_depth =
        rf_random_split.best_params_['max_depth'],
                        bootstrap =
        rf_random_split.best_params_['bootstrap'])

model2_split.fit(train_x, train_y)

model2_split_prediction = model2_split.predict(test_x)
```

Accuracy of Model is: 0.9053846430868259

Root Mean Squared Error of Model is: 5.058264806966998

3.2 Comparison and Result

The accuracy and root mean squared error are two parameters to evaluate the model performance. The two parameters of all the seven models we used are shown in the following table.

Table: Summary of Model Performance

Method	Accuracy	Root Mean Squared Error
Linear Regression	0.5177053629131334	11.420285520195613
Lasso Method	0.38967572787640603	11.988560504390488
K-nearest Neighbor	0.34345271643724284	12.434253663809674
Support Vector Machine	0.34808748601553163	12.390287319386937
Neural Network	0.8565038038524211	6.229324285295386
Decision Tree Regressor	0.8204795438030693	6.9675118677407
Random Forest Regression	0.9053846430868259	5.058264806966998

As shown in the table, Linear Regression, Lasso Method, K-nearest Neighbor and Support Vector Machine do not perform well in predicting concrete strength. The other three methods: Neural Network, Decision Tree Regressor and Random Forest Regression are relatively better.

The accuracy of Random Forest Regression and Neural Network are in the first and second place, respectively. The comparison of these two methods is as follows.

- When creating a model, Neural Network is more complicated. Setting appropriate values for its parameter such as layer numbers, learning rate, batch size, etc. is extremely improtant to the performace of the model, so that it requires more efforts to find better parameters. Random Forest is much easier to find the best parameters.
- Random Forest is less computationally expensive. It can be trained faster than Neural Network.
- When using Neural Network, we should pay attention to avoiding overfitting. However, Random Forest is less prone to overfitting.

Compared with previous studies, the linear regression and neural network models we used do not show the same level of accuracy. One reason could be that the dataset we used to train the model is not big enough. Since we split 20% of the train dataset into test dataset for evaluating model performance, the total number of data we used for training is 565 rows. In the future study, we are expected to use larger dataset to train models, hoping to get a similar accuracy.

What's more, a key insight of our project is that random forest regression shows a great potential to predict concrete strength, however, it has not been used widely in previous studies. We will further confirm wether random forest regression only shows a great performance in the dataset we used or can be applied to other dataset.

For this project, Random Forest Model works best. We recreate a random forest model using the whole given train dataset and use it to predict the given test dataset.

Code for Hyperparameter Tuning

```
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
        random_grid, n_iter = 100, cv = 3, verbose=2, random_state=None,
        n_jobs = -1)
# Fit the random search model
rf_random.fit(x, y)
```

The best parameters for the model is:

{'n_estimators': 600, 'min_samples_split': 2, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 70, 'bootstrap': True}

Code for recreate a model

```
model2 = RandomForestRegressor(n_estimators =
          rf_random.best_params_['n_estimators'],
                              min_samples_split =
          rf_random.best_params_['min_samples_split'],
                              min_samples_leaf =
          rf_random.best_params_['min_samples_leaf'],
                              max_features =
          rf_random.best_params_['max_features'],
                              max_depth =
          rf_random.best_params_['max_depth'],
                              bootstrap =
          rf_random.best_params_['bootstrap'])

model2.fit(x, y)
```

prediction Code for recreate a model

```
prediction2 = model2.predict(test_df)
prediction2 = pd.DataFrame(prediction2, columns=['Concrete compressive
        strength(MPa, megapascals)'])
prediction2.index.name='index'
prediction2.to_csv('prediction2.csv')
```

After submitting the *prediction2.csv* to Kaggle Competition, it provides a score of 5.50396 which indicates Root Mean Squared Error. We can expect a model accuracy of 85%-90% for such a score.

#	△pub	Team Name	Notebook	Team Members	Score ❓	Entries	Last
1	▲ 1	Qinyu Zhang			5.50396	8	11d
2	▼ 1	Sonali Srivastava			5.71618	5	6d
3	—	Pratyush Kumar62			13.22915	1	6d
📍		sample.csv			40.97523		

Figure: Kaggle Competition Page

4. Conclusion

Being non-linear and uncertain material, performance of concrete varies a lot with different constituents and their ratio. Performing compressive strength test for large set by crushing the cylinders is time consuming and prone to human error. Now, ML and ANN are taking place in mix deisgn because of its high accuracy and speed. Different training methods have been opted. Each method has its own merits and limitations. It is very crucial to understand the dataset first and then optimize the method.

From the Exploratory Data analysis, it is found that Concrete strength has a very strong positive correlation with age and cement content. The correlation states that with the increase in the amount of curing time and cement content the strength of concrete would increase sharply. There is a significant negative correlation between concrete strength and water suggesting that increase in water content within the mix leads to degradation of strength. Prior to training of the model scaling of the dataset is needed and the high amount of outliers in the age variable data was taken into account.

A total of seven methods are used to predict the concrete strength, among which random forest model has a lowest root mean squared error and a highest accuracy. So, we create a random forest model trained by the whole train dataset and use it to predict the given test dataset. The score(root mean squared error) obtained from Kaggle Competition is about 5.5, which indicates the model accuracy is about 85%-90%.

In this study, random forest shows a great potential in predicting concrete strength, however, one limitation is that the dataset we used is not large enough. In the future study, we will use larger datasets, aiming to 1) improve model accuracy and, 2) confirm whether random forest can be applied to other datasets.

References

[1] Alshihri, M. M., Azmy, A. M., & El-Bisy, M. S. (2009). Neural networks for predicting compressive strength of structural light weight concrete. Construction and Building Materials, 23(6), 2214-2219. [https://doi.org/10.1016/j.conbuildmat.2008.12.003] (https://doi.org/10.1016/j.conbuildmat.2008.12.003)

[2] Maier, H. R., & Dandy, G. C. (1998). The effect of internal parameters and geometry on the performance of back-propagation neural networks: an empirical study. Environmental Modelling & Software, 13(2), 193-209. [https://doi.org/10.1016/S1364-8152(98)00020-6] (https://doi.org/10.1016/S1364-8152(98)00020-6)

[3] Yeh, I. C. (1998). Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research, 28(12), 1797-1808. [https://doi.org/10.1016/S0008-8846(98)00165-3] (https://doi.org/10.1016/S0008-8846(98)00165-3)

[4] Atici, U. (2011). Prediction of the strength of mineral admixture concrete using multivariable regression analysis and an artificial neural network. Expert Systems with applications, 38(8), 9609-9618. [https://doi.org/10.1016/j.eswa.2011.01.156] (https://doi.org/10.1016/j.eswa.2011.01.156)