Comparison of Regression Analysis Algorithms

Regression analysis may be defined as a type of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable(s) (predictor). This technique is used for forecasting, time series modelling and finding the cause-effect relationship between the variables.

Regression analysis may be considered a reliable method of identifying the variables that have impact on a topic of interest. In the final part of the study, I tried to determine which factors matter most and which factors can be ignored. When I used only the most important (having the highest correlation to the target) parameters (4 out of 8), the results were still quite satisfactory.

Evaluating the model accuracy is an essential part of the process in evaluating the performance of machine learning models to describe how well the model is performing in its predictions. Evaluation metrics change according to the problem type. The errors represent how much the model is making mistakes in its prediction. The basic concept of accuracy evaluation is to compare the original target with the predicted one according to certain metrics. The below metrics are mainly used to evaluate the prediction error rates and model performance in regression analysis.

- **MAE** (Mean absolute error) represents the difference between the original and predicted values extracted by averaged the absolute difference over the data set.
- **MSE** (Mean Squared Error) represents the difference between the original and predicted values extracted by squared the average difference over the data set.
- **RMSE** (Root Mean Squared Error) is the error rate by the square root of MSE.
- **R-squared** (Coefficient of determination) represents the coefficient of how well the values fit compared to the original values. The value from 0 to 1 interpreted as percentages. The higher the value is, the better the model is.

A more clear definition (at least for me) is: "The coefficient of determination, R^2, is used to analyze how differences in one variable can be explained by a difference in a second variable. More specifically, R-squared gives you the percentage variation in y explained by x-variables." Most often than not (unless there are too many variables), R-squared is used for evaluating the model performance.

A little domain knowledge; more tons of concrete are produced and used than any other technical material. This is due to its low cost and widely available raw materials (water, slag, cement...). Also, cement is the bonding material in the concrete. Concrete strength (target value) is affected by many factors, such as quality of raw materials, water/cement ratio, coarse/fine aggregate ratio, age of concrete etc.

This article involves the application of 5 different machine learning algorithms, with the aim of comparing their performances for regression analysis. The machine learning algorithms that I used for regression are:

1. Linear Regression,
2. Lasso Regression,
3. Ridge Regression,
4. Random Forest Regression,
5. XGBOOST Regression.

I tried to concentrate on the following issues:

- Which regression algorithm provided the best results?
- Is Cross Validation effective in increasing the regression performance?
- Results of determining the important features using only those parameters on the performance of the algorithm.

I used the Kaggle dataset. As explained on the website, the first seven parameters are the addition to the concrete (units in kg in a m^3 mixture and "Age" in days (1 to 365)). There are 8 quantitative input variables, and 1 quantitative output variable (Compressive Strength (in MPa) of the concrete); 1030 instances with no missing data.

Data Cleaning

The first step is to import and clean the data (if needed) using pandas before starting the analysis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import train_test_split, GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

```
df = pd.read_csv('Concrete_Data.csv')
```

```
df.head(3)
```

	cement	slag	flyash	water	superplasticizer	coarseaggregate	fineaggregate	age	csMPa
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.99
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.89
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.27

```
df.shape
```

```
(1030, 9)
```

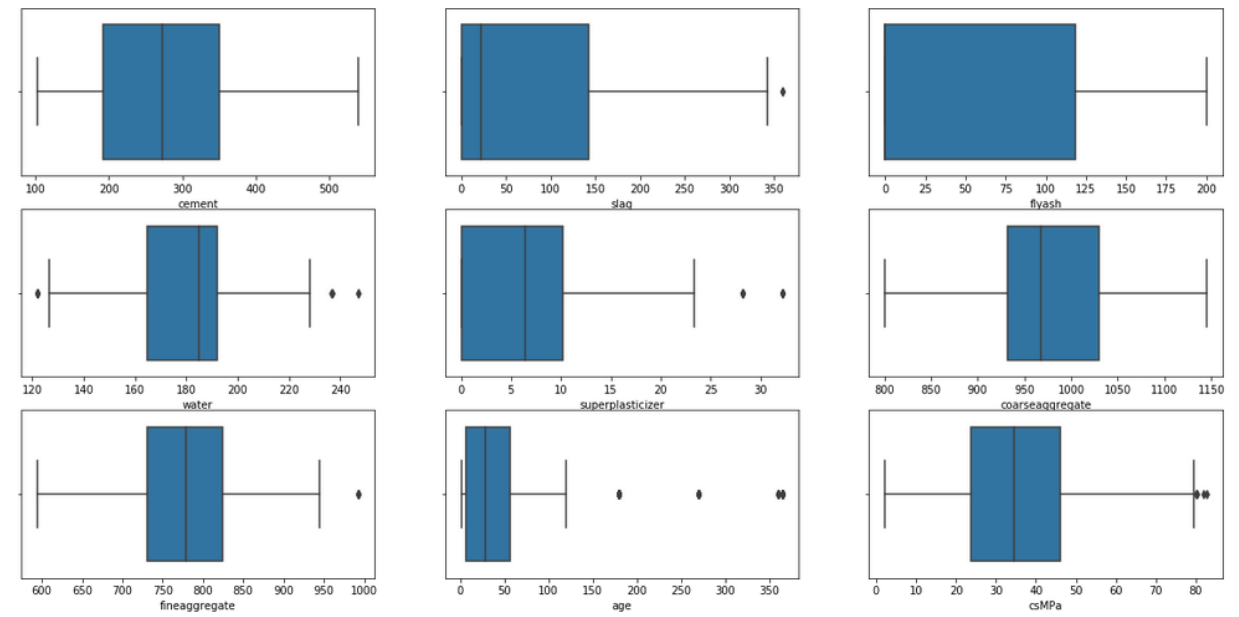
According to the dataset, there were 1030 different prepared concretes and no missing values.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   cement            1030 non-null   float64
 1   slag              1030 non-null   float64
 2   flyash            1030 non-null   float64
 3   water             1030 non-null   float64
 4   superplasticizer  1030 non-null   float64
 5   coarseaggregate   1030 non-null   float64
 6   fineaggregate     1030 non-null   float64
 7   age               1030 non-null   int64  
 8   csMPa             1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

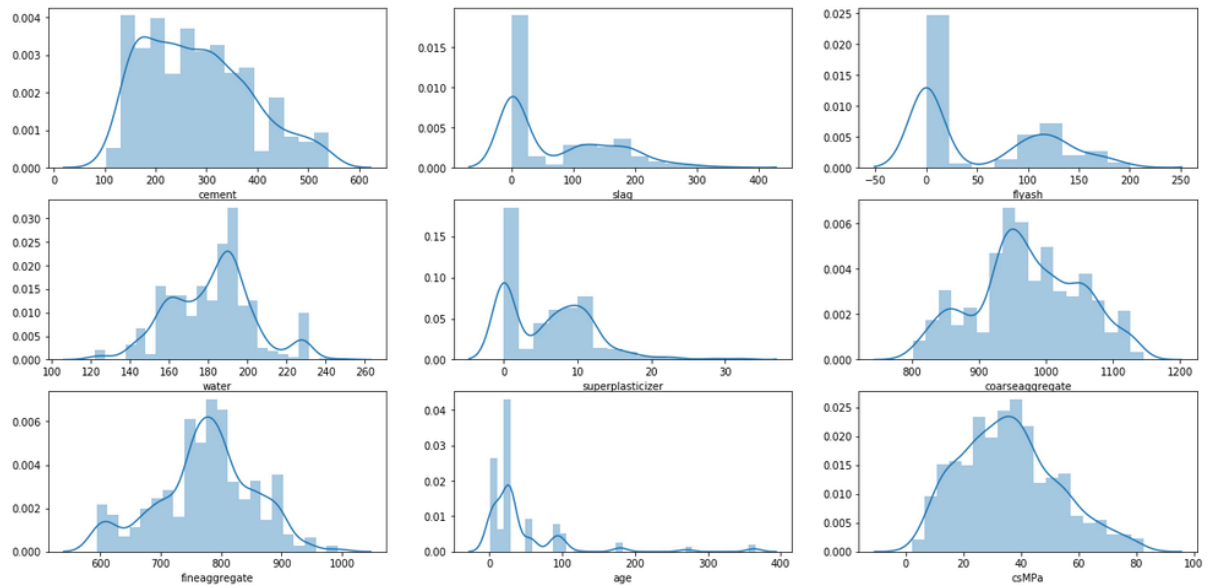
I also checked the outliers, which may cause negative effects on the performance of the analysis. The boxplots below show that there were not many outliers, so I moved to the distribution of the values.

```
fig = plt.figure(figsize=(20,20))
for col in range(len(df.columns)) :
    fig.add_subplot(6,3,col+1)
    sns.boxplot(x=df.iloc[ : , col])
plt.show()
```



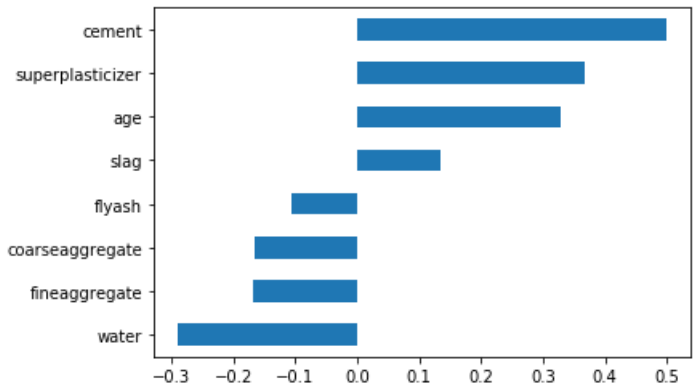
Cement, water, coarse aggregate and fine aggregate have the highest (absolute valued) correlation coefficients and their use in the concrete were close to normal distribution. In many concrete mixtures, slag, fly ash and superplasticizer were not used at all. For the distribution pattern of these three parameters, a reasonable explanation may be that, two distributions were combined in the data, giving the appearance of a bimodal distribution. Also, aging time was kept to a minimum (right-skewed).

```
fig = plt.figure(figsize=(20,20))
for col in range(len(df.columns)) :
    fig.add_subplot(6,3,col+1)
    sns.distplot(df.iloc[ : , col])
plt.show()
```



The dataset is clean (there are no NaNs, Dtypes are correct), so we will directly start by checking the correlation of the parameters with the target value – csMPa (Compressive Strength of concrete). The figure below shows that cement and superplasticizer have the highest positive correlation coefficient values and water has the highest negative correlation coefficient value.

```
: df.corr()['csMPa'].sort_values().head(11)[:-1].plot.barh();
```



I used the below Python function for the prediction of the errors.

```
def eval_metrics(actual, pred):
    rmse = np.sqrt(mean_squared_error(actual, pred))
    mae = mean_absolute_error(actual, pred)
    mse = mean_squared_error(actual, pred)
    score = r2_score(actual, pred)
    return print("r2_score:", score, "\n","mae:", mae, "\n","mse:",mse, "\n","rmse:",rmse)
```

Linear Regression

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

```
lm.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

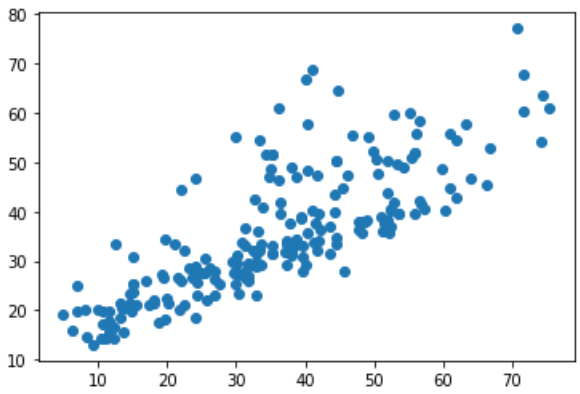
```
y_pred = lm.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.6275531792314851
 mae: 7.745559243921434
 mse: 95.97094009110677
 rmse: 9.796475901624358
```

I started by using linear regression in order to model the relationship between the features and the target variable. MAE and RMSE results were in terms of MPa (unit for the compressive strength) and they tell us that, the difference between the observed and predicted results are ± 7.74 (MAE) and ± 9.79 (RMSE) MPa. R2-score of 0.6275 is not a very high value and shows that even noisy, high-variability data can have a significant trend. The trend indicates that the predictor variable still provides information about the response even though data points fall further from the regression line.

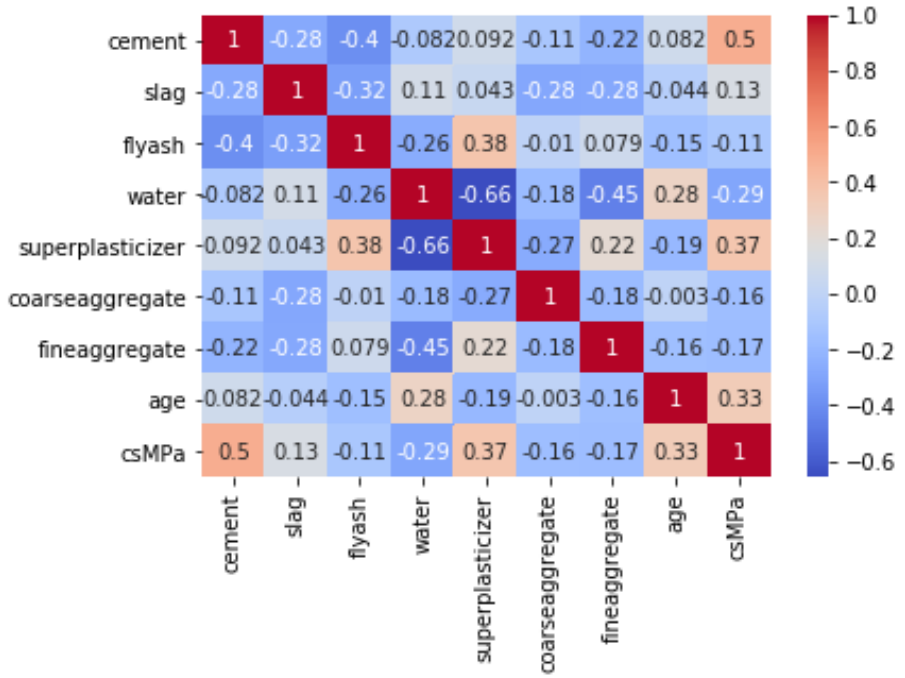
```
plt.scatter(y_test, y_pred);
```



Ridge Regression

Ridge regression is defined as a model tuning method that is used to analyze any data that suffers from multicollinearity. Multicollinearity refers to a situation in which two or more explanatory variables in a multiple regression model are highly linearly related. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values to be far away from the actual values. The heatmap below shows the highest relationship is between water and superplasticizer (- 0.66) and this may be the reason for the low r2_score of the linear regression analysis.

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm');
```



Ridge regression analysis results are given as:

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import RidgeCV
```

```
ridge_model = Ridge(normalize = True)
ridge_model.fit(X_train, y_train)
y_pred = ridge_model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.4543314313295702
 mae: 9.789608304785087
 mse: 140.60618212665062
 rmse: 11.857747767879474
```

Ridge regression uses α (alpha) as the parameter which balances the amount of emphasis given to minimizing RSS vs minimizing sum of square of coefficients. α can take various values. For Ridge Cross Va;idation, I defined an array of 100 possible alpha values between 0.1 and 20; the optimum value turned out to be 0.1. When I fit the model again using 0.1 as the alpha, there was a huge increase in the r2_score – from 0.45 to 0.60.

```
alpha_space = np.linspace(0.1, 20, 100)
```

```
ridgecv = RidgeCV(alphas = alpha_space, normalize = True, cv = 5)
ridgecv.fit(X_train, y_train)
```

```
ridgecv.alpha_
```

```
0.1
```

```
ridge_model = Ridge(alpha = 0.1, normalize = True)
ridge_model.fit(X_train, y_train)
y_pred = ridge_model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.608947460896801
 mae: 8.169587838624908
 mse: 100.76520380898606
 rmse: 10.038187277042905
```

Lasso Regression

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. Like Ridge regression, Lasso is well-suited for models showing high levels of multicollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination. As mentioned in the Ridge Regression section, there is multicollinearity in this data and using Lasso Regression may improve the results.

Lasso Cross Validation analysis results for the optimum alpha value of 0.1 and 0.0001 are given as:

```
lasso_model = Lasso(alpha = 0.1, normalize = True)
lasso_model.fit(X_train, y_train)
y_pred = lasso_model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.4429558286421522
 mae: 9.937719612147946
 mse: 143.53741209865504
 rmse: 11.980709999772762
```

```
lasso_model = Lasso(alpha = 0.01, normalize = True)
lasso_model.fit(X_train, y_train)
y_pred = lasso_model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.6183269275527674
 mae: 7.971717501231669
 mse: 98.34833197029273
 rmse: 9.91707275209236
```

```
lasso_model = Lasso(alpha = 0.0001, normalize = True)
lasso_model.fit(X_train, y_train)
y_pred = lasso_model.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.627561159308472
 mae: 7.74753860480038
 mse: 95.96888380965201
 rmse: 9.796370951002826
```

The jump in the r2_score is very high from 0.44 to 0.62, when the alpha value becomes almost zero. This shows us that, it is impossible to get an improvement in the r2_score with the Lasso regression model.

Random Forest Analysis

Wikipedia definition is: “Random forests or random decision forests are an ensemble learning method (multiple learning algorithms) for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.”

Random Forest analysis results display a huge increase in the r2_score:

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf_reg = RandomForestRegressor()
```

```
rf_reg.fit(X_train, y_train)
```

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

```
y_pred = rf_reg.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.8782516579464815
 mae: 3.8526910968562196
 mse: 31.37173467422852
 rmse: 5.601047640774762
```

Cross Validation mean score is almost 0.90:

```
cv_scores = cross_val_score(estimator = rf_reg, X = X_train, y = y_train, cv =3)
cv_scores.mean()
```

```
0.8995289849597103
```

Next step was the evaluation of the importance of the features on the analysis:

```
rf_reg.feature_importances_
```

```
array([0.3357392 , 0.07996596, 0.01729716, 0.12362899, 0.04759985,
       0.02981436, 0.03470658, 0.33124789])
```

```
importance = pd.DataFrame({'importance':rf_reg.feature_importances_},
                          index = X_train.columns)
```

```
importance.sort_values(by="importance", ascending = False).head(10)
```

	importance
cement	0.335739
age	0.331248
water	0.123629
slag	0.079966
superplasticizer	0.047600
fineaggregate	0.034707
coarseaggregate	0.029814
flyash	0.017297

I will consider the first 4 features (cement, age, water and slag) for further analysis. After getting new dataframes including only the important features and splitting the dataset, the results are as follows:

```
new_list = ['cement', 'age', 'water', 'slag']
```

```
X = df[new_list]
y = df['csMPa']
X = pd.get_dummies(X)
```

```
X.shape
```

```
(1030, 4)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
rf_reg = RandomForestRegressor()
```

```
rf_reg.fit(X_train, y_train)
```

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

```
y_pred = rf_reg.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.8712903374599394
 mae: 4.0740675572122065
 mse: 33.16550611786815
 rmse: 5.7589500881556654
```

What I got here is, I used only 4 features (instead of 8)- only the ones considered as the most important and the drop of r2_score was from 0.8995 to 0.8712. In our case, there were only 1030 data, but while working with huge datasets, working with less features will save a lot of time, with a little loss of accuracy.

XGBoost Analysis

XGBoost (Extreme Gradient Boosting) is an optimized distributed gradient boosting library and it is used for supervised ML problems. XGBoost belongs to a family of boosting algorithms that convert weak learners into strong learners.

I used XGBoost and got the best results as shown:

```
model_xg = xgboost.XGBRegressor()
model_xg.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

```
y_pred = model_xg.predict(X_test)
```

```
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.9300780437301508
 mae: 2.841180106690786
 mse: 18.41601985920957
 rmse: 4.291389036105858
```

Once again, I used the important features function and selected the 4 most important parameters. The drop in r2_score was infinitesimal – from 0.9300 to 0.9234:

```
new_list = ['cement', 'age', 'water', 'slag']
```

```
X=df[new_list]
y=df['csMPa']
X=pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
model = xgboost.XGBRegressor()
model.fit(X_train, y_train)
```

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1,
             objective='reg:squarederror', random_state=0, reg_alpha=0,
             reg_lambda=1, scale_pos_weight=1, subsample=1, tree_method='exact',
             validate_parameters=1, verbosity=None)
```

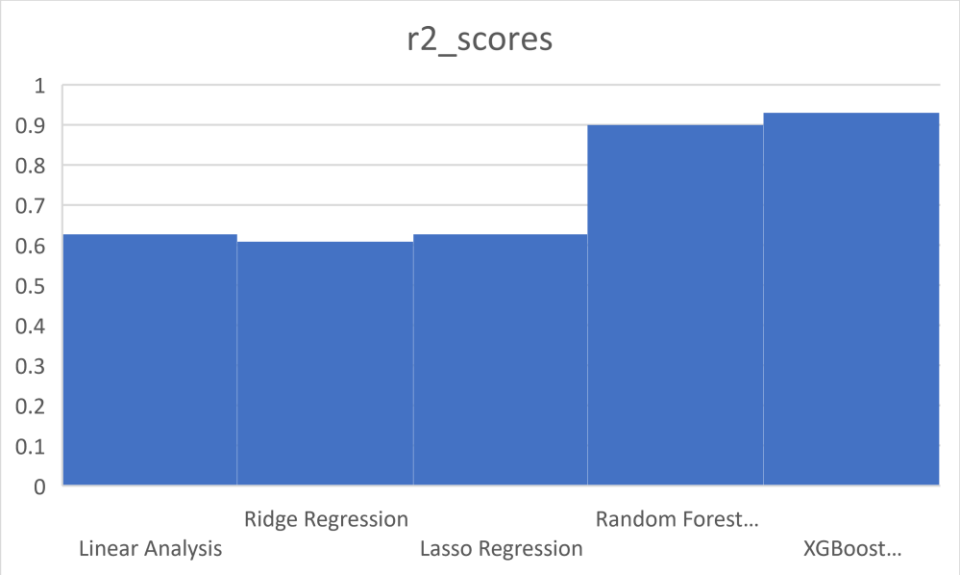
```
y_pred = model.predict(X_test)
eval_metrics(y_test, y_pred)
```

```
r2_score: 0.9234763522315677
 mae: 2.918025233213184
 mse: 19.71837590231892
 rmse: 4.440537794267595
```

Conclusion

In this article, I used 5 regression algorithms and some key facts associated with each technique. The analysis provides evidence that:

- XGBoost algorithm provided the best results, followed by the Random Forest algorithm.
- Linear, Lasso and Ridge regression algorithms' results were quite close.
- After cross validation, there was a huge increase in the accuracy of the Lasso and Ridge regressions.
- Using only the most important 4 features (out of 8) resulted in a little decrease in the accuracy of the results, with a high potential in saving time for large datasets.



You can access to this article and similar ones here.

Photo by 준영 박 on Unsplash