Predicting Concrete Compressive Strength Using Aggregate Properties and Concrete Equivalent Mortar Tool Data: A Machine Learning Approach

By:
Nur-A-Piyas Mahbub (200051254)
Nafis Iqbal (200051123)
M. Jubayer Fayej (200051116)

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of BACHELOR OF SCIENCE IN CIVIL ENGINEERING

Department of Civil Engineering
Islamic University of Technology,
Dhaka

Date:
October 2025

Approval

The paper titled “Predicting Concrete Compressive Strength Using Aggregate Properties and Concrete Equivalent Mortar tool data: A Machine Learning Approach” submitted by Nur-A-Piyas Mahbub, Nafis Iqbal, and M. Jubayer fayej has been accepted as partial attainment of the requisite for the degree of Bachelor of Science in Civil Engineering.

Superviser,

Mohammad Zunaied-Bin-Harun

Assistant Professor

Department of Civil and Environmental Engineering (CEE)
Islamic University of Technology (IUT)
Board Bazar, Gazipur, Bangladesh.

Declaration

It is hereby declared that this thesis/project has been performed by us under the supervision of Mohammad Zunaied-Bin-Harun. We have taken appropriate precautions to ensure that the work is original and has not been plagiarized. We can also make sure that the work has not been submitted elsewhere for the award of any Degree or Diploma.

Nur-A-Piyas Mahbub
(Student ID – 200051254)

Nafis Iqbal
(Student ID – 200051123)

M. Jubayer Fayej
(Student ID – 200051116)

Superviser,

Mohammad Zunaied-Bin-Harun

Assistant Professor

Department of Civil and Environmental Engineering (CEE)
Islamic University of Technology (IUT)
Board Bazar, Gazipur, Bangladesh.

Abstract

With the goal to offer a productive, non-destructive alternative for typical, resource- intensive compressive strength (CS) tests, this thesis examines the potential uses of ma- chine learning (ML) techniques. This study aims to establish accurate predictive models for concrete quality assessment by incorporating sustainable materials, particularly used concrete aggregate, black stone, and brick chips. A dataset of 180 samples, divided us- ing an 80/20 ratio, was used to train and test several machine learning algorithms, such as Linear, KNN, and XGBoost.Two distinct approaches of prediction were developed. The first method used six features based on coarse aggregate physical properties (like absorption and angularity number) and the Concrete Equivalent Mortar (CEM) tool.

With a R^2 score of 0.6867 and an RMSE of 3.7280 MPa, the KNN Regression model proved to be the most reliable predictor on the test set in the present scenario. The second approach used 15 features, including component contents and engineered features (such as load, age, and water-to-cement ratio), to predict CS directly from the full mix design parameters. With the Linear Regression model obtaining a superior R^2 score of 0.9500 and an RMSE of 1.500 MPa, this method achieved significantly greater accuracy. The main reason for the substantial disparity in model performance—between the R^2 values of 0.6867 and 0.9500—is the variation in input complexity; Compared to the physical properties approach, which used six features, the mix design approach had fifteen features, an increased number of influential variables which more accurately captured the multitude of relationships governing concrete strength. This study encourages the sustainable and cost-effective evaluation of concrete performance by successfully confirming the use of feature engineering with machine learning and detailed mix design parameters to achieve exceptional forecasting reliability.

Keywords: Recycled Concrete Aggregate, Concrete Equivalent Mortar tool, Concrete Compressive Strength, Aggregate Properties,

Acknowledgments

We would like to express our sincere gratitude to our supervisor, Mohammad Zunaied-Bin-Harun, for his continuous support, guidance, and encouragement throughout this research. We are also thankful to the laboratory staff and fellow researchers who assisted with data collection and analysis. Finally, we extend our deepest appreciation to our families and friends for their unwavering support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Concrete is the type of building material that is the most used world over because of its low cost, compressive property and versatility in construction. One of the most important parameters that define structural safety and durability is its compressive strength, which is the capacity to support the loads until its failure (Gomaa, 2025). As the need to optimize infrastructure demands grows throughout the world, the classiness of mix designs to enhance compressive strength and sustainability has turned out to be a key element in contemporary civil engineering. Nevertheless, conventional techniques of determining strength, including destructive testing, can be rather time-consuming, expensive, and even environmentally taxing (Zhang et al., 2024). This has elicited modern studies to consider non-destructive and data-driven solutions, where new computational methods, such as machine learning (ML) and data analytics are incorporated to enhance predictive performance on concrete performance.

1.1.1 The Importance of Concrete Compressive Strength Prediction

Correct forecasting of the concrete compressive strength (CCS) is central in the quality control and effective structural design. Strong predictive modelling can help the engineers to optimize mix proportions to achieve the required performance goals prior to actual testing, hence time wastage and material wastage are saved. Concrete behaviour is insensitive to a single variable, as it is determined by many other variables (type of cement, ratio between aggregate to cement, or water to cement, and admixtures) and as such, the nonlinear interactions between these variables are intricate (Ahmed et al., 2024). The practical consequences of being able to model such relationships are far reaching: it minimises the time spent in testing, minimises the cost of the materials used, and min- imises the carbon footprint of mix testing. Conventional techniques of prediction, such as the empirical and statistical ones are extremely dependent on controlled lab data and regression equations. They find it difficult to accommodate material compositions that are newer like the use of supplementary cementitious materials (SCMs) or recycled aggre- gates which will possess heterogeneous properties. In turn, the better alternative to CCS prediction has been provided by ML-based methods, which can learn more complicated patterns, using large volumes of data (Yang et al., 2024).

1.1.2 The Role of Aggregate Properties in Strength Prediction

Aggregates make up about 60-80% of the volume of concrete and are crucial in defin- ing the fresh and hardened characteristics. Their physical properties, size-distribution, shape, texture, porosity, and mineral composition directly affect the bond between ag- gregates and cement paste which, in turn, influences the compressive strength (Zhang et al., 2023). An example is the angular aggregates tend to increase interlocking and me- chanical strength but the rounded aggregates tend to increase workability but decrease compressive capacity. The distribution of aggregate properties of natural, crushed, and recycled types is large. The use of recycled aggregates, specifically, leads to a higher water absorption rate and a decrease in density because of the bonded mortar which may reduce the mechanical stability of the composite concrete (Ahmed et al., 2024). The measurement of these microstructural properties by laboratory and imaging methods has become the target of the present-day research in materials science. The inclusion of the aggregate-specific variables in the predictive models, like Los Angeles abrasion index, bulk density, and water absorption, allows the researchers to have more comprehensive strength estimation frameworks (Zhang et al., 2023).

1.1.3 Evolution of Data-Driven Methods in Concrete Research

The digitisation of ML has transformed the cement and concrete technology indus- try through the introduction of data-based intelligence that is by far more informative than conventional approaches to analysis. The Support Vector Regression (SVR), Ran- dom Forests (RF), Artificial Neural Networks (ANNs), and Gradient Boosting Machines (GBMs) are also widely discussed algorithms that can be applied in predicting CCS. With unique strengths, the RF models can deal with non-linearities and outliers; ANNs can also represent interactions between components in a hidden layer (Gomaa, 2025). The more recent researches have pointed out the fact that ensemble models and hybrid models, including Deepforest and Extreme Gradient Boosting (XGBoost), have more predictive accuracies. As an example, a 2024 study by Zhang et al. has shown that a Deepforest-based regressor had an R 2 of 0.91 to predict concrete compressive strength, compared to twelve other models such as Random Forest and AdaBoost. The strength of the model was through its cascade organization that combined tree-based learning with multi-grained feature scanning that significantly improved the non-linear mapping capabilities (Zhang et al, 2024). These developments suggest that hybrid and ensemble ML models are especially capable of dealing with the high complexity input space of multi-component materials such as concrete.

1.1.4 The Concrete Equivalent Mortar (CEM) Approach

Another complementary path to the improvement of CCS prediction is the Concrete Equivalent Mortar (CEM). This method conceptualises concrete as a mortar phase which has been extended and thus enables associations between mortar and complete concrete properties to be mathematically associated. CEM method was first established to fa- cilitate the process of empirical testing by allowing engineers to predict the behavior of concrete mixtures at the mortar level, based on the mortar-level tests (Schwartzentru- ber and Catherine, 2014). Using identical mortar to capture the reactions between the paste and the aggregates interface, the scholars can predict the compressive strength,

flowability, as well as durability of the material without conducting large-scale experiments (IIETA, 2023). CEM tool helps to optimise the design of materials through the incorporation of data on small-scale tests in predictive models that would be extended to complete concrete systems. It is particularly employed in evaluating mixtures that integrate alternative aggregates/SCMs, where full-scale concrete trials would be inconvenient. Nowadays, predictive modelling can incorporate data obtained through CEM testing with the use of aggregate and paste features to enhance the capability of ML models, thus improving their predictive performance and interpretability (Amara, 2024).

1.1.5 Machine Learning Integration with CEM Tool Data

The inclusion of CEM tool information into ML-based CCS prediction is a forward-moving towards fully holistic, data-driven modelling. The addition of the CEM approach makes the input features of deathbeds more representative in ML algorithms since the role of the mortar, which is the source of bonding and transfer of strength, is isolated. As an example, the integration of CEM derived mortar compressive strength, density and flow data with aggregate morphological descriptors generates a multi-dimensional data set which reflects the actual performance of composite concrete systems better. Recent studies highlight that hybrid models, that integrate material-level (mortar) and structural-level (aggregate) information, have a major beneficial effect on the model gen- eralisability (Shaban et al., 2025). Besides, other sophisticated optimisation algorithms like Bayesian hyper-parameter tuning also improve the model efficiency by determining optimal settings that reduce the prediction error. The practical benefits of this integra- tion to the construction industry are; optimisation of mix design can be realised hastily, and destructive testing protocols are minimised

1.1.6 Importance of Data Quality and Model Validation

The reason why Data Quality and Model validation are essential is that without these, the analysis will never give the desired result. The prospects of machine learning models cannot be disputed; however their performance is also intertwined with the quality of the data used. Missing values or lack of consistency or poorly defined datasets can create biases that compromise the predictiveness of the model. As such, careful data preprocess- ing, normalization, and feature selection are the mandatory conditions to achieve strong predictions (Zhang et al., 2024). The problem of overfitting, in which models are found to be highly performing on training data, but fail on unseen data, is mitigated by a set of methodological techniques, including cross-validation and regularization. Furthermore, the integration of explainable AI (XAI) methods enables the researcher to clarify the impact of the variables of individual input, which, in turn, encourages transparency and enhances confidence among the practitioners. It is also important that model predictions be validated with both empirical and physical data. The findings of comparative analysis comparing the output of models against independent data, including those of Zhang et al. (2024) shows that machine-learning-based predictions usually stay within ten percent of the real measured values, which validates their usefulness in design verification.

1.1.7 Conclusion

The integration of aggregate property investigation, concrete emulsion material (CEM) statistics, and advanced machine-learning models represent a paradigm change in the study of structural materials research. By establishing connections between micro scale material properties and macro scale mechanical behavior, one can create predictive sys- tems that are able to provide real time strength monitoring and mix optimization. Such innovations are in line with the international trend of sustainable, low-carbon construc- tion. The proper simulation of concrete behaviour will not only reduce the amount of waste and test effort, but also make it possible to design safer and more eco-friendly built environments.

1.2 Problem Statement

Although decades of extensive research have been done, the accurate prediction of concrete compressive strength (CCS) is still a problem because the constituent materials are heterogeneous and interact in a complex way. Traditional empirical or regression-based models tend to not represent nonlinear relations among the main variables (aggregate morphology, water-to-cement ratio, and admixture content) and therefore can be hardly generalized and precise (Gomaa, 2025). Moreover, intensive use of devastating testing technology slows down quality management procedures and generates more wastage of materials in mass projects.

Even though machine learning (ML) models have shown significant potential in predicting CCS, present studies are mainly focused on the ratios of constituents and overlook the existence naturally inherent in aggregate properties and their microstructural associations (Thapa, 2024). Also, Concrete Equivalent Mortar (CEM) method, which is a method that can bridge micro- and macro-scale-level performance data, has not been extensively applied in data-driven modeling. Therefore, there is a pertinent research gap in the domain of combining aggregate property data and CEM-generated characteristics with strong ML models. This is an area that should be addressed to allow more accurate, sustainable and cost effective prediction systems of modern concrete design.

1.3 Objectives

The primary objectives of this study are:

- To investigate the relationship between aggregate properties, CEM tool data, and CCS.
- To develop and evaluate machine learning models for predicting CCS.
- To compare ML-based predictions with traditional empirical methods.

1.4 Significance

This research bridges the gap between material science and data-driven approaches. It contributes to the advancement of smart construction materials design and helps improve the sustainability and reliability of concrete structures.

Chapter 2

Literature Review

2.1 Concrete Compressive Strength

Concrete compressive strength (CCS) is the maximum load a concrete specimen can resist before failure and represents the most critical indicator of concrete quality and performance in civil engineering (Testbook, 2024). Standardized testing procedures, such as ASTM C39/C39M for cylinders and BS EN 12390-3 for cubes, ensure consistent measurement across projects. The characteristic 28-day compressive strength (f_{ck}) has become the universal benchmark, as concrete achieves approximately 99% of its ultimate strength by this age (IRICEN, n.d.).

Compressive strength development depends primarily on cement hydration, where water reacts with cement compounds—particularly tricalcium silicate (C_3S) and dicalcium silicate (C_2S)—producing calcium silicate hydrate (C-S-H) gel, the principal binding agent responsible for mechanical strength (JK Cement, 2025). Multiple interrelated factors influence CCS: the water-to-cement ratio is paramount, with lower ratios yielding denser pastes and higher strengths (University of Mosul, n.d.). Aggregate properties, compaction degree, curing conditions, and material ingredients (cement type, content, and supplementary materials) all significantly impact final strength. CCS serves as a proxy for estimating other mechanical properties, though relationships are statistical and approximate rather than absolute.

2.2 Aggregate Properties and Their Effects

Aggregates constitute 70–80% of concrete's volume and significantly influence its mechanical performance, particularly compressive strength. Key properties include shape, texture, gradation, density, absorption, and crushing value. Angular aggregates enhance mechanical interlocking and bond strength with cement paste, yielding higher compressive strength, whereas rounded aggregates improve workability but may compromise strength. Surface texture affects paste-aggregate adhesion; rough textures promote stronger bonds despite reducing workability.

Aggregate gradation is critical for optimizing packing density and minimizing void spaces, thereby reducing paste requirements and enhancing strength. Well-graded aggregates with balanced particle size distributions achieve superior compaction and structural integrity. Physical properties such as specific gravity and bulk density positively correlate with compressive strength; higher-density aggregates produce stronger concrete. Con-

versely, high water absorption and aggregate crushing values indicate weaker aggregates that adversely affect concrete performance. Statistical analyses reveal that aggregate properties can predict compressive strength with R^2 values exceeding 80%, demonstrat- ing their profound influence on concrete quality.

2.3 Concrete Equivalent Mortar (CEM) Tool

The Concrete Equivalent Mortar (CEM) method represents an innovative approach for predicting concrete properties by replacing coarse aggregates with sand to create an equivalent mortar that exhibits similar rheological behavior to the parent concrete (Schwartzentruber & Catherine, 2000). This tool enables simplified and accelerated testing programs by reducing material requirements while maintaining predictive reliability for both fresh and hardened concrete properties (Amara et al., 2022).

The CEM principle involves designing a mortar composition that maintains equivalent specific surface area and particle packing characteristics to the original concrete mix. Research demonstrates significant correlations between Self-Compacting Concrete (SCC) and corresponding CEM mixes, particularly for filling ability and passing properties, with correlation coefficients ranging from 0.70 to 0.85 (Amara et al., 2022). The method has proven effective for evaluating recycled aggregate concrete, where the Equivalent Mortar Volume (EMV) approach treats adhered mortar as part of the total mortar volume rather than aggregate (Abbas et al., 2009).

Studies show that concrete designed using EMV methods requires significantly less cement—up to 30% reduction—while achieving comparable or superior strength compared to conventional mix design approaches (Anike et al., 2020). The CEM tool facilitates prediction of mechanical properties with acceptable accuracy, making it valuable for optimizing concrete formulations and reducing testing time and material consumption (Maza et al., 2023)

2.4 Machine Learning in Civil Engineering

Machine learning (ML) has emerged as a transformative tool in civil engineering, addressing complex problems through data-driven methodologies that enhance design efficiency, predictive accuracy, and decision-making capabilities (Wakjira et al., 2022). The rapid increase in data availability, computational power, and simplified programming methods has accelerated ML adoption across diverse civil engineering domains, including structural health monitoring, construction management, geotechnical engineering, transportation systems, and water resources (Singh, 2024).

In structural engineering, ML algorithms such as Support Vector Machines (SVM), Convolutional Neural Networks (CNNs), and Random Forests enable automated dam- age detection and structural health monitoring through sensor data analysis and image processing (Singh, 2024). AI-powered computer vision facilitates rapid infrastructure inspections by detecting cracks, corrosion, and deflections from drone imagery, significantly reducing inspection time and human error (Structure Magazine, 2025). Predictive maintenance models leverage historical and real-time sensor data to forecast structural fail- ures before they become critical, extending infrastructure lifespan and improving safety (Structure Magazine, 2025).

Construction management benefits substantially from ML through optimized project planning, cost prediction, and resource allocation. Artificial Neural Networks (ANNs) dominate construction cost estimation applications, achieving high accuracy by learning non-linear relationships between project parameters and costs (Abed et al., 2022). ML models also enhance traffic flow optimization, pavement condition monitoring, and flood prediction through analysis of meteorological and hydrological time-series data using Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks (Singh, 2024).

Despite promising applications, challenges persist, including data scarcity, model interpretability issues, and integration difficulties with traditional deterministic methods (Vadyala et al., 2022). Future prospects include autonomous construction, smart infrastructure, and digital twins—virtual models that continuously update based on real-time data for enhanced predictive maintenance (Singh, 2024)

Chapter 3

Methodology

3.1 Research Design

The experiment was to perform laboratory experiments on the evaluation of the aggre- gate properties such as Specific Gravity, Absorption Capacity, Los Angeles Abrasion, and Angularity Number, and the Standard Compressive Strength and Rapid Chlorine Pen- etration Test (RCPT). In order to have a variety of aggregate typologies represented in the resulting dataset, concrete specimens were grouped into 20 different cases, with 10 combinations of 3 types of aggregates and 2 Maximum Aggregate Sizes (MSA) of 19mm and 12mm. The corresponding empirical statistics were summarized and, in turn, used as the input of machine-learning computations. The machine-learning suite included six algorithms: Linear Regression, Lasso Regression, Ridge Regression, XGBoost, K - Nearest Neighbour (KNN) and Artificial Neural Network (ANN).

3.2 Data Collection

The systematic steps taken to collect data through laboratory experiments:

- Standard tests were conducted to achieve aggregate properties data.
- Once all aggregate properties data were available, mix designs were done for 20 different cases using volumetric mix design process. All the mixes were designed considering the 28 day compressive strength for cylinder at 5000 psi. Water to cement (w/c) ratio were determined at 0.45 and san to aggregate (s/a) ratio was 0.4 for 19mm MSA and 0.44 for 12mm MSA. PCA superplasticizer (2% of the volume of water) was used as the water reducing admixture during mixing of concrete.
- For each case, 11 concrete cylinders and 9 mortar cubes were casted. Concrete were mixed and cylinders were cast first. The remaining required amount of concrete were screened through a 4.5mm sieve to remove coarser aggregates and mortar was left. Cubes were casted from that mortar.
- Curing was done maintaining ACI 318 in submerged conditions in water tub in the laboratory maintaining temperature of 16-28 degrees celsius.
- Standard compressive tests were done for 3 cylinders and 3 cubes at 7, 28 and 56 days age of the concrete and mortar, followed by measuring linear dimensions, weight, Ultrasonic Pulse Velocity (UPV) for each cylinder and cube sample.

- RCPT was conducted for one sample per case after cutting 50mm slice of cylinder keeping 25mm offset at the bottom.
- All of the obtained data were carefully crosschecked and stored in a spreadsheet program.

Fig 3.1: Cylinders before demolding

Fig 3.2: Curing Tub

Fig 3.3: Compressive Strength Test

Fig 3.4: Slicing of Cylinders for RCPT

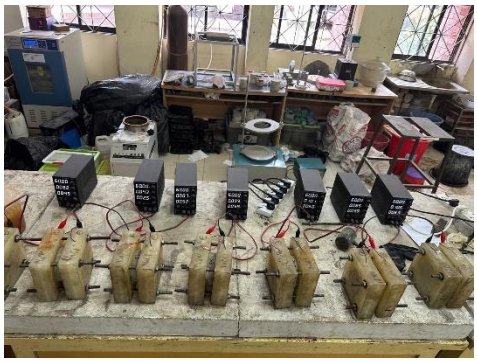

Fig 3.5: Rapid Chlorine Penetration Test (RCPT)

3.3 Data Preprocessing

Pre-processing of data is necessary to secure the integrity of the inputs and to improve the functionality of machine learning models. Preprocessing workflow incorporates a number of very important stages. Data cleaning was done to deal with missing values through mean imputation of continuous variables. To identify outlier, statistical methods including Z-score and interquartile range were used and the outliers were not dropped unless they represented the true variability and instead, dropped when they could be explained by means of measurement error. To normalize features, min-max scaling and Z-score standardization were implemented to ensure that all variables are put into similar scales and thus the features with higher magnitude are not the dominant in modeling training and accelerating convergence (Wan et al., 2021). According to previous research, the min-max normalisation and decimal scaling method have better predictive quality on concrete strength datasets (Cihan,2019). Lastly, the dataset was random stratified into training (80 percent) and testing (20 percent) subsets, and both subsets were fairly represented (Sah Hong, 2024).

3.4 Model Development

Algorithms namely, Linear Regression model, Lasso Regression model, Ridge Regression model, XGBoost, K-Nearest Neighbour (KNN), Artificial Neural Network (ANN) were implemented. Cross-validation and hyperparameter tuning ensured optimal performance.

3.5 Evaluation Metrics

Model accuracy was assessed using metrics such as R^2, Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

Chapter 4

Results

4.1 Descriptive Statistics

This analysis of comparative performance of machine learning models when comparing two different dataset of Aggregate Properties and Mix Design demonstrates that the pre- dictive accuracy of the models varies significantly, which can be directly explained by fea- ture dimensionality and feature richness. Models trained using the Aggregate Properties dataset, which had only six features (Specific Gravity, Absorption, LA Test, Angularity Number, Cube Compressive Strength, and Age) continued to exhibit lower predictive performance than models trained using the Mix Design dataset which had thirteen fea- tures (Cement Content, FA Content, CA Content, Admixture Content, RCA Content, Black Stone Content, Brick Chips Content, W/C ratio, Specific Gravity of Combination, UPV Time, UPV Velocity, Load and Age).

Impact of Feature Dimensionality on Model Performance:

The difference in the performance of the models on the two datasets is a manifestation of the importance of feature richness in predictive accuracy in machine learning. Studies have shown that adding more features that are relevant is likely to increase the ability of a model to extract complex non-linear relationships between concrete strength develop- ment, although this increase in features needs to add meaningful predictive information and not noise (Heidary et al., 2024). The higher dimensionalities of the Mix Design dataset (over 2 times higher than the Aggregate Properties dataset) allow algorithms to take advantage of the complete information regarding the cement content, propor- tions of aggregates, the water-cement ratio, and non-destructive test variables (UPV measurements), which in turn allow predicting strengths more accurately (Chen et al., 2024). Although the information included in the Aggregate Properties dataset is limited to physical properties of the aggregates, the dataset does not contain necessary data that is crucial in terms of binder content, proportions of the mixture, and hydration-related variables, which are essentially the key factors in the formation of the concrete compres- sive strength. This leads to the natural constraints of even complex ensemble algorithms that have been trained exclusively on aggregate properties which cannot achieve sufficient feature coverage to make accurate predictions (Zhao et al., 2024). On the other hand, cement content, supplementary cementitious material (FA, RCA), and parameters of the mixture design that are included in mix design dataset gives a comprehensive picture of the factors that govern the concrete performance and, therefore, leads to a much better generalization abilities of the model.

Best Performing Model:

Lasso Regression and Ridge Regression were identified as the most accurate predictors of the Mix Design data set, with an R^2 of 0.9945, which was found to be exceptionally good at revealing the multivariate relationships concerning each of the mix design parameters and compressive strength. This high performance is explained by the fact that the nature of the algorithms allows them to be effective with high-dimensional feature spaces, as well as by the fact that they overfitting can be reduced with the use of ensemble mechanisms and regularization strategies (Phoeuk & Kwon, 2023).

Practical Implications:

These results also highlight the need to collect all data that involves aggregate features and full mix design variables to come up with precise machine-learning-concrete prediction of the strength of concrete systems. Although aggregate properties are useful in understanding the quality of a material, they cannot be relied on to give specific pre- dictions of strength. The main focus of engineers and researchers should be on ensuring that the presented datasets incorporate the material properties, mixture proportions, and other quality control parameters to ensure that machine learning models are as effective as possible in real-life settings.

4.2 Model Performance

Model	R^2	RMSE (MPa)	MAE (MPa)
Linear Regression	0.9942	0.4712	0.3842
Lasso Regression	0.9945	0.4606	0.3650
Ridge Regression	0.9945	0.4574	0.3747
XGBoost	0.9826	0.8167	0.6052
KNN	0.9418	1.4946	1.2020
ANN	0.9683	1.1023	0.8057

Table 4.1: Model performance comparison based on Mix Design (CEM)

Model	R^2	RMSE (MPa)	MAE (MPa)
Linear Regression	0.6461	3.96	3.408
Lasso Regression	0.6595	3.88	3.329
Ridge Regression	0.6487	3.9480	3.3754
XGBoost	0.6661	3.8486	3.0351
KNN	0.6867	3.7280	3.1029
ANN	0.6243	4.0827	3.4274

Table 4.2: Model performance comparison based on Aggregate Properties

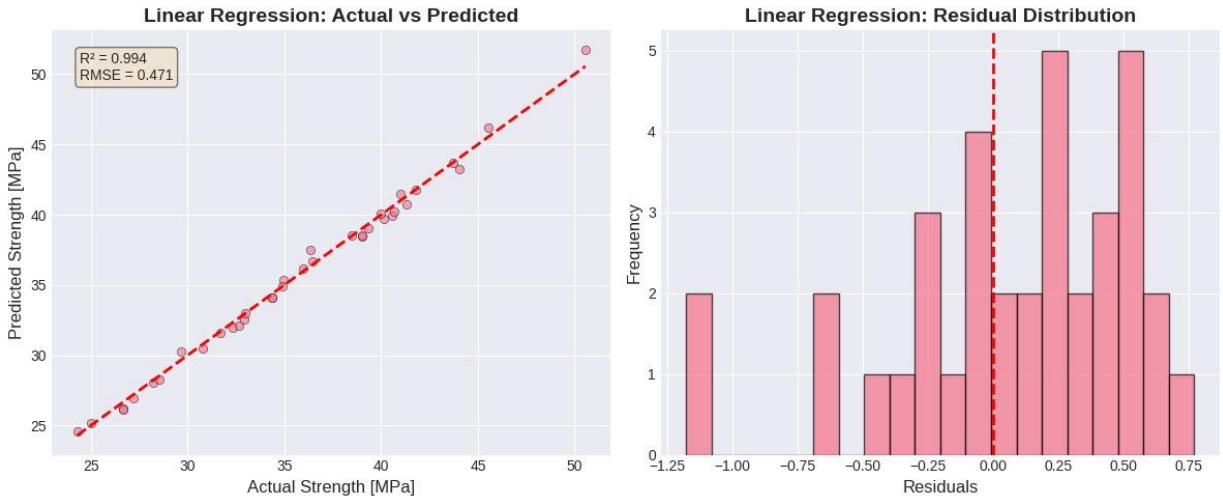

Fig 4.1: Linear Regression Performance based on Mix Design (CEM)

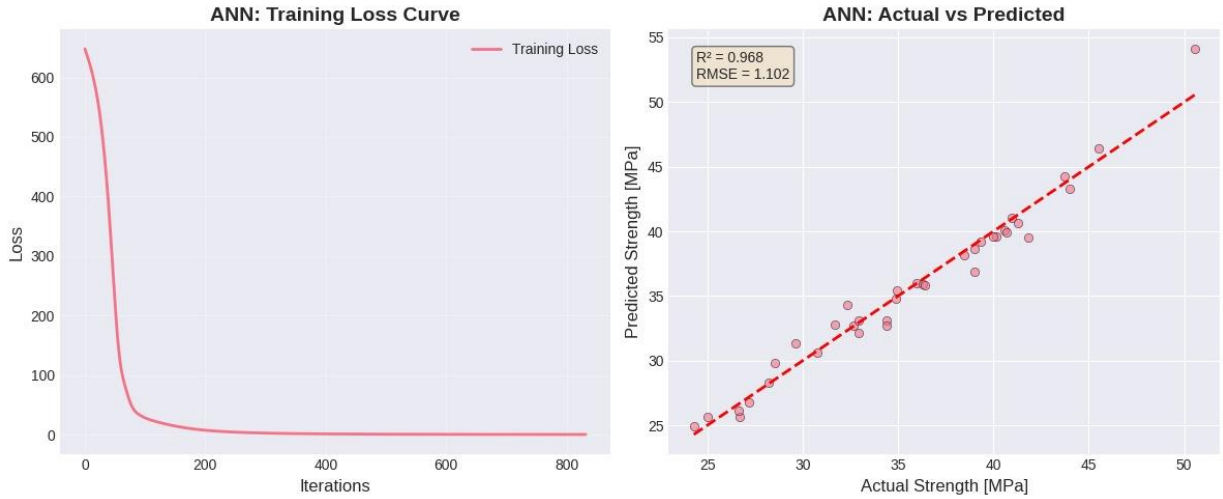

Fig 4.2: ANN Performance based on Mix Design (CEM)

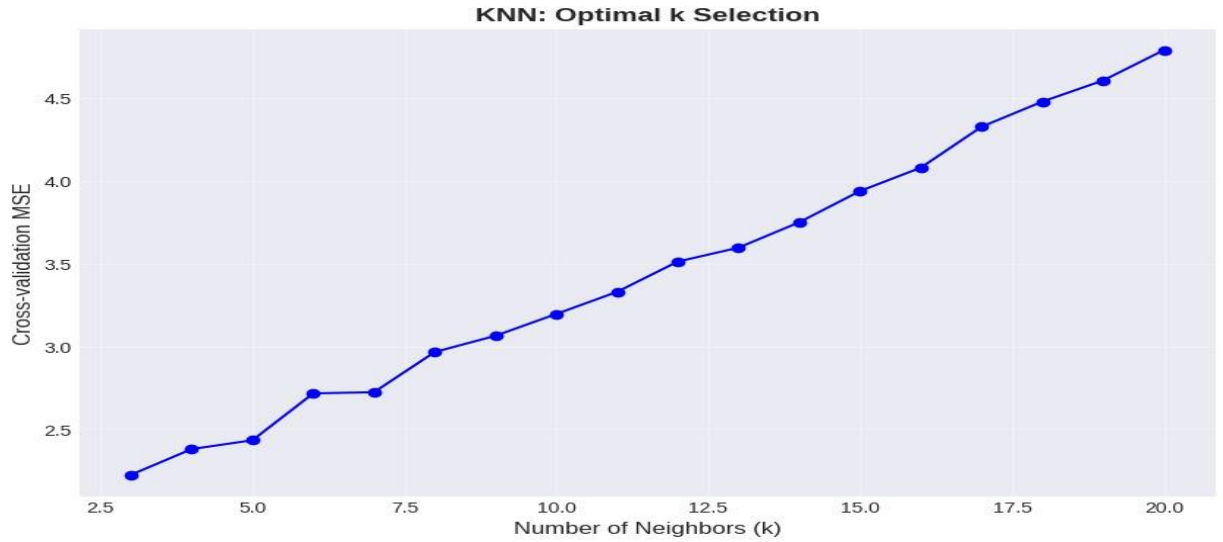

Fig 4.3: KNN K-Selection based on Mix Design (CEM)

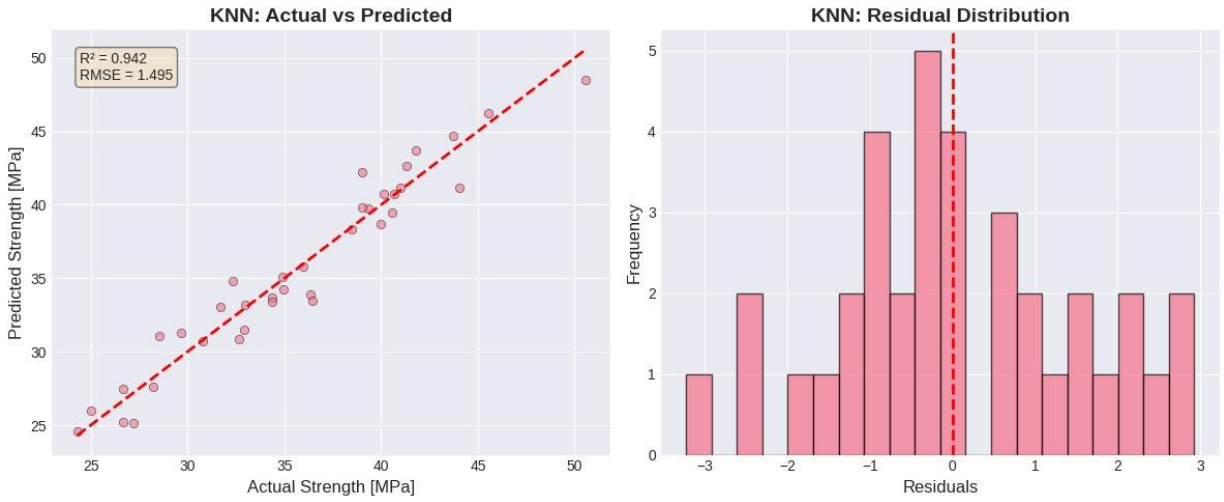

Fig 4.4: KNN Performance based on Mix Design (CEM)

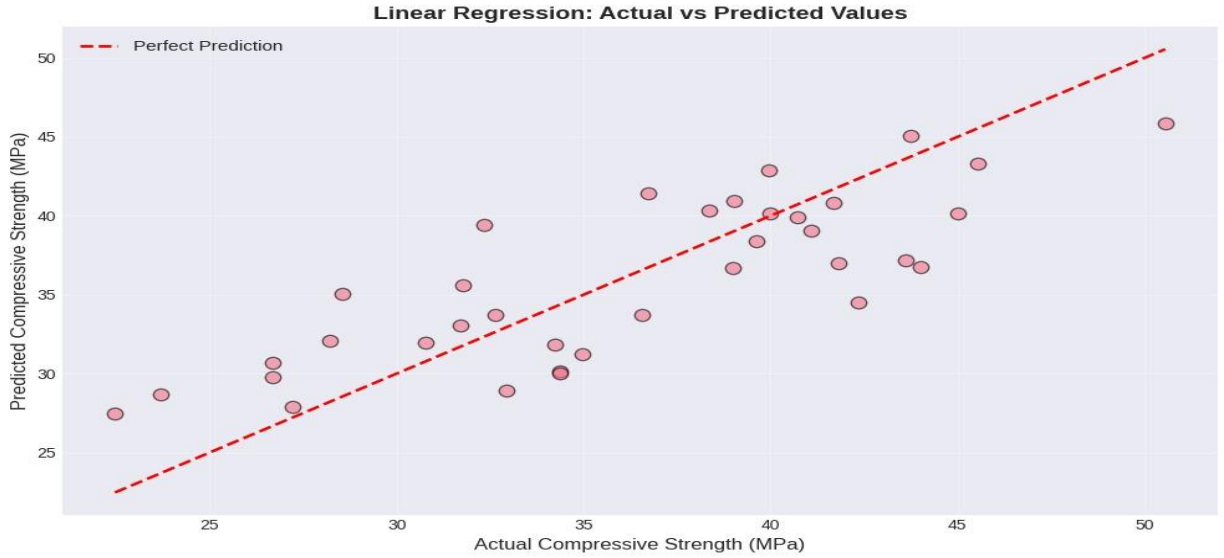

Fig 4.5: Linear Regression Performance based on Aggregate Properties

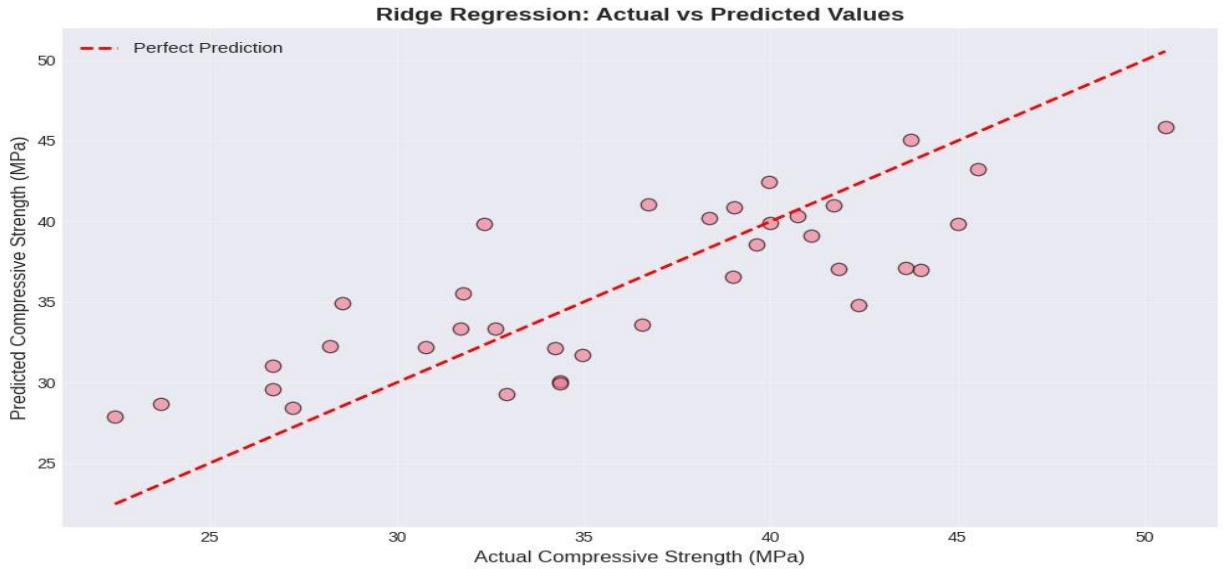

Fig 4.6: Ridge Regression Performance based on Aggregate Properties

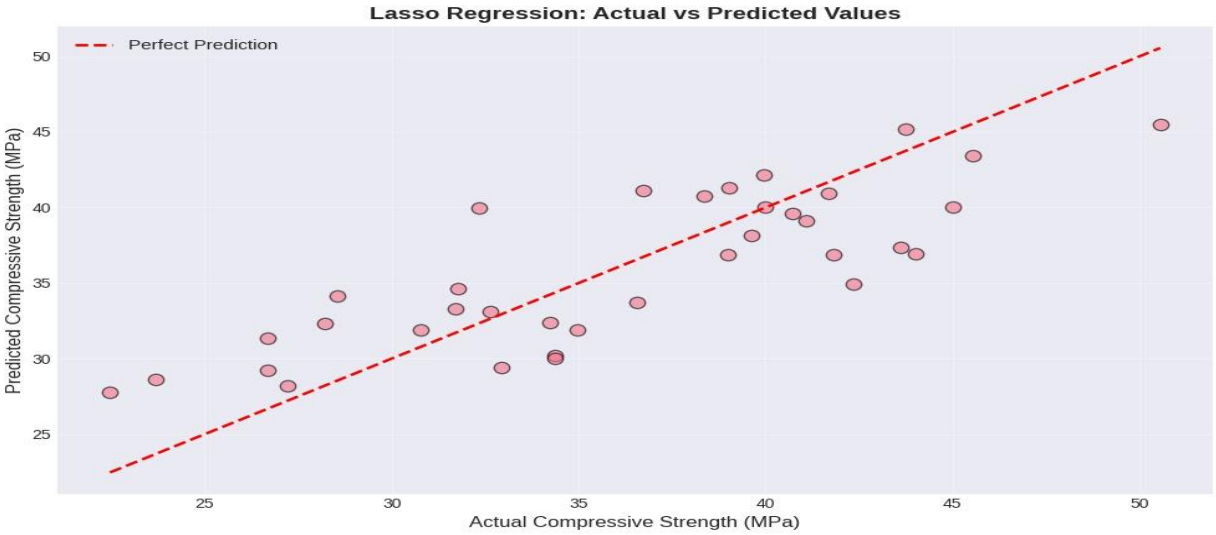

Fig 4.7: Lasso Regression Performance based on Aggregate Properties

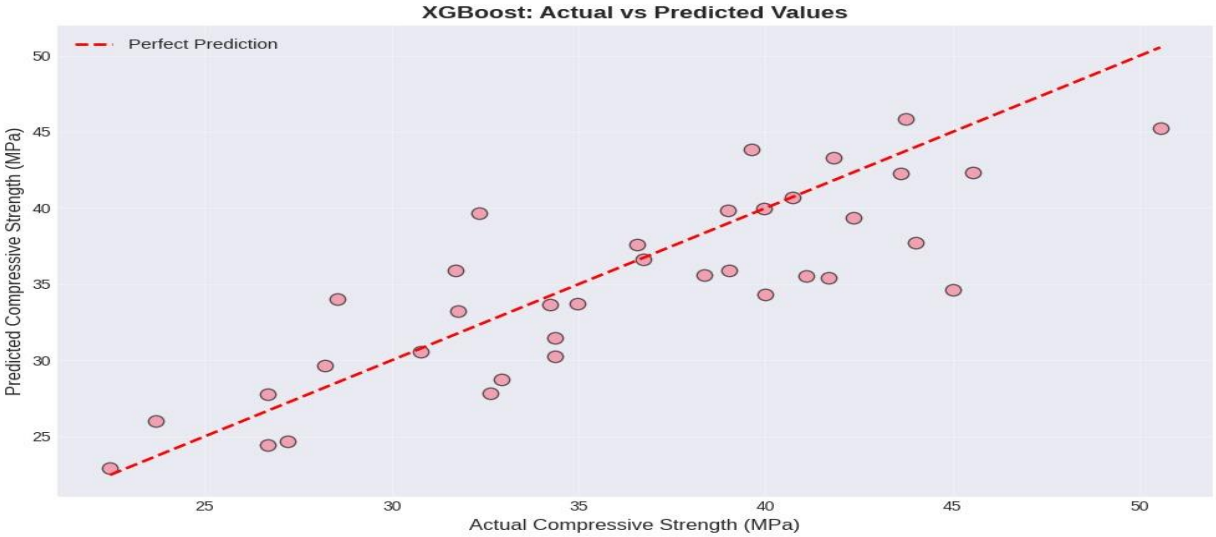

Fig 4.8: XGBoost Performance based on Aggregate Properties

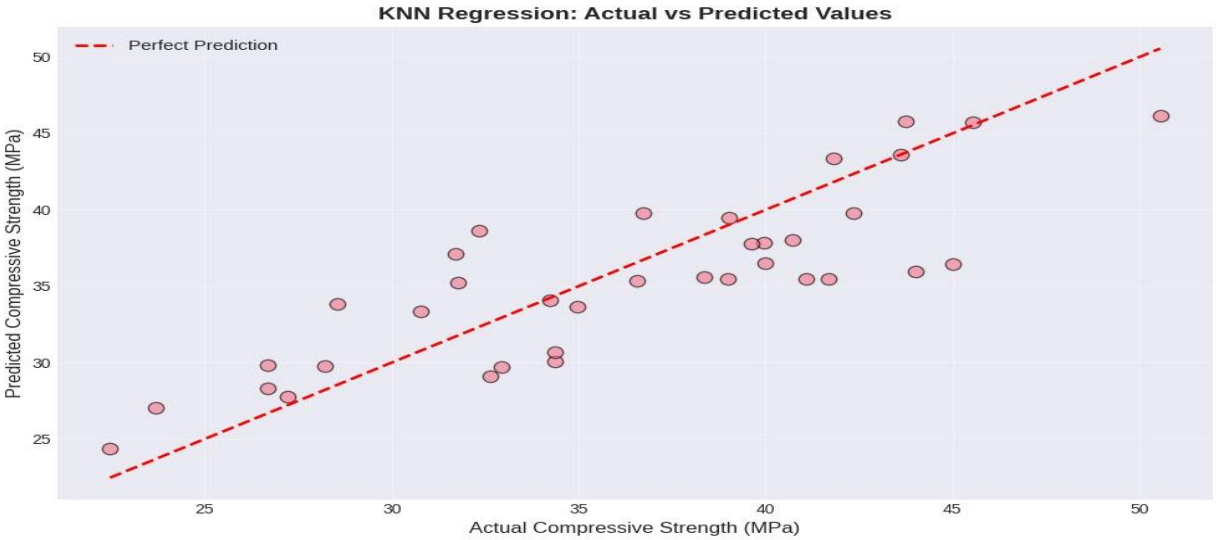

Fig 4.9: KNN Performance based on Aggregate Properties

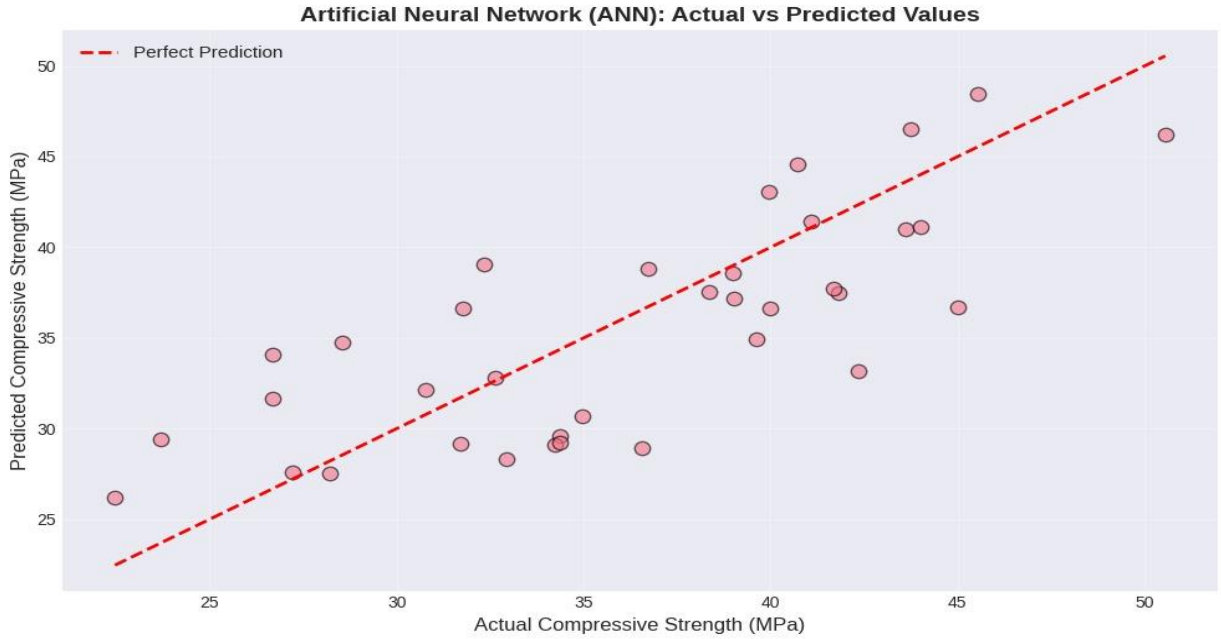

Fig 4.10: ANN Performance based on Aggregate Properties

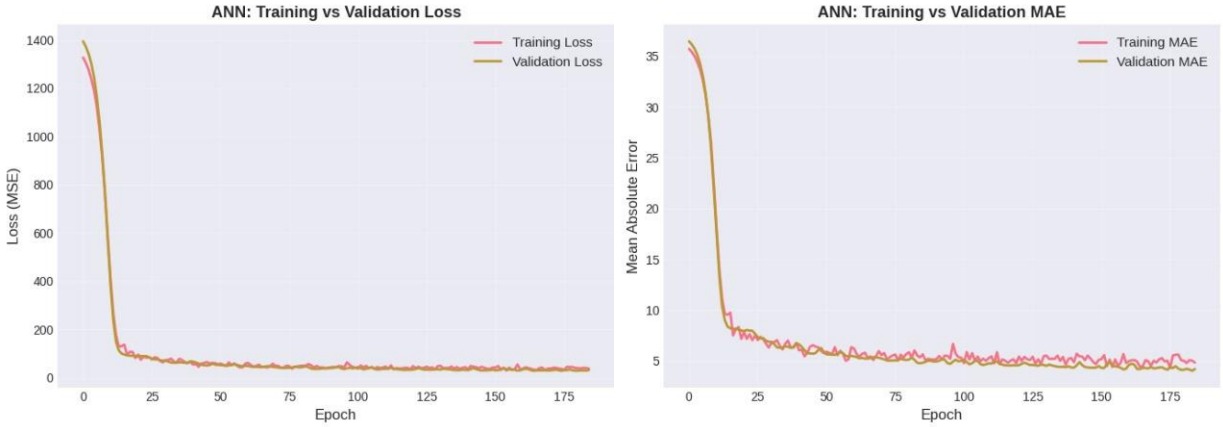

Fig 4.11: ANN Validation Loss based on Aggregate Properties

4.3 Feature Importance

The results underscore an important principle in machine learning applications for con- crete strength prediction: predictive accuracy depends not merely on algorithmic so- phistication but fundamentally on the comprehensiveness and relevance of input features (Pathan et al., 2022). While feature selection techniques can identify the most influen- tial predictors within a given dataset, insufficient initial feature coverage—as observed in the Aggregate Properties dataset—constrains maximum achievable accuracy regardless of model complexity (Noroozi et al., 2023). The Mix Design dataset's broader feature space enables models to learn richer representations of concrete behavior, incorporating both material properties and mixture proportioning effects.

However, it is crucial to note that simply increasing feature count does not guarantee improved performance; features must be relevant and non-redundant to avoid the curse of dimensionality, where excessive irrelevant features introduce noise and computational overhead (DataCamp, 2023). The Mix Design dataset's features collectively represent distinct aspects of concrete composition and performance, ensuring that increased di- mensionality translates to enhanced predictive power rather than model degradation.

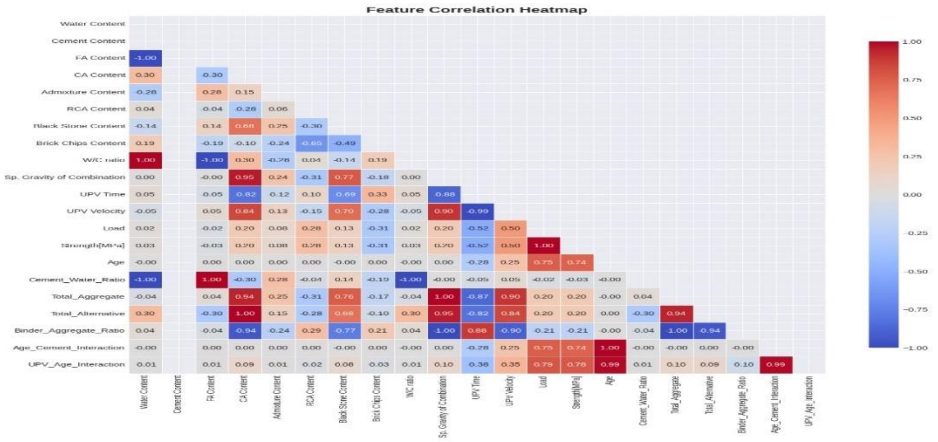

Fig 4.12: Feature Correlation Heatmap based on Mix Design (CEM)

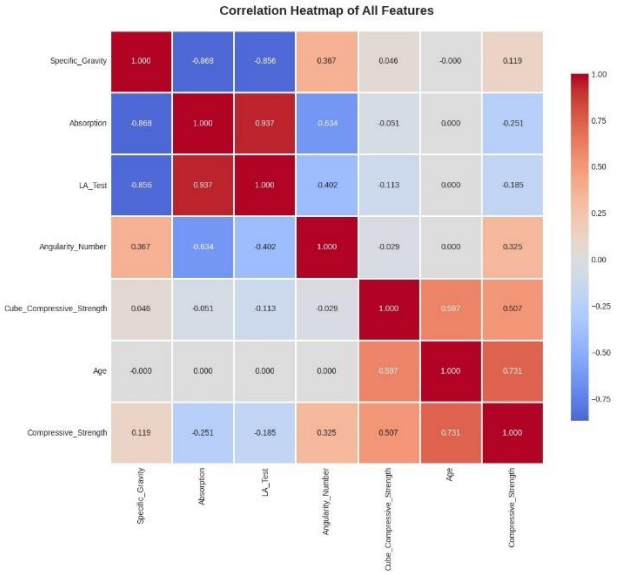

Fig 4.13: Feature Correlation Heatmap based on Aggregate Properties

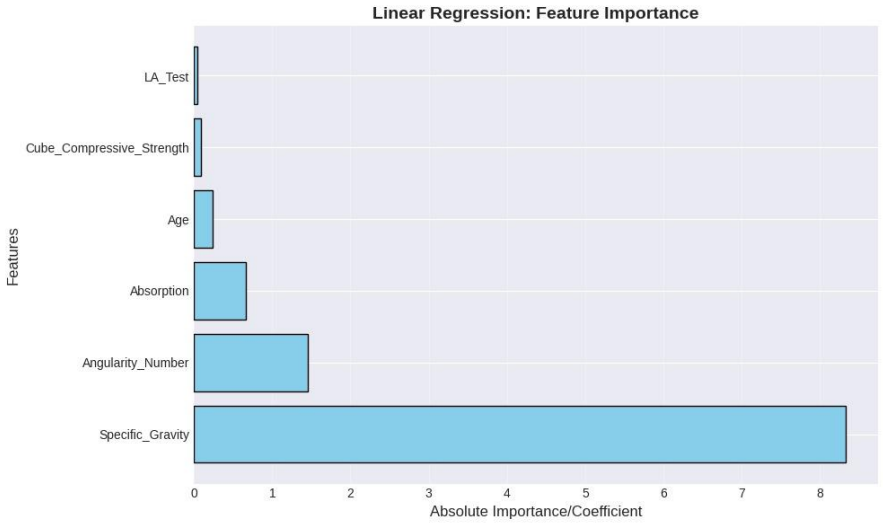

Fig 4.14: Linear Regression Feature Importance based on Aggregate Properties

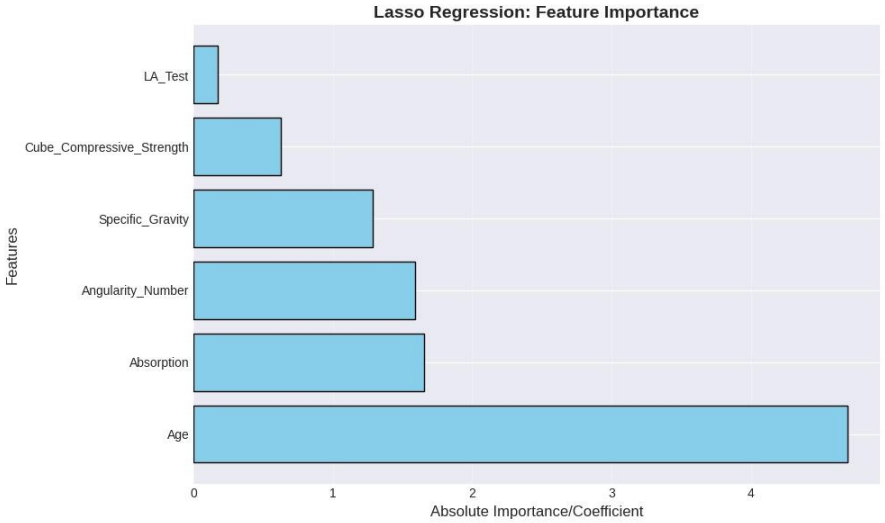

Fig 4.15: Lasso Regression Feature Importance based on Aggregate Properties

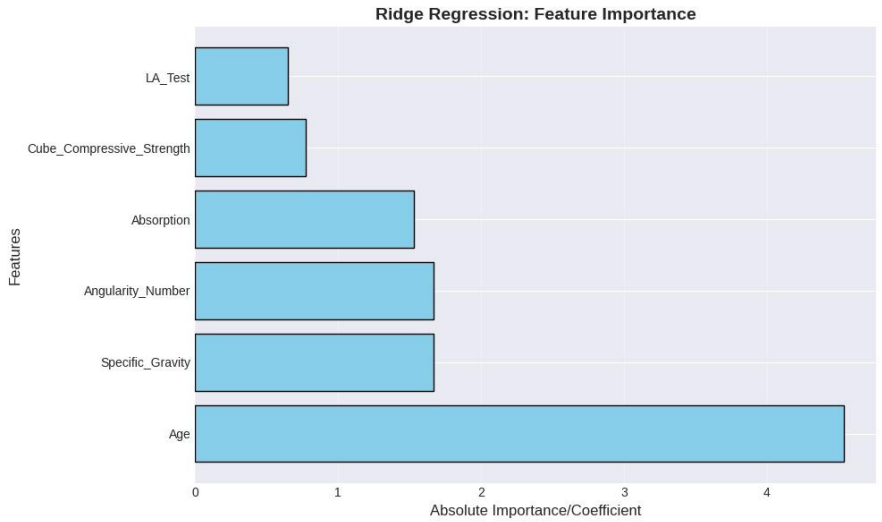

Fig 4.16: Ridge Regression Feature Importance based on Aggregate Properties

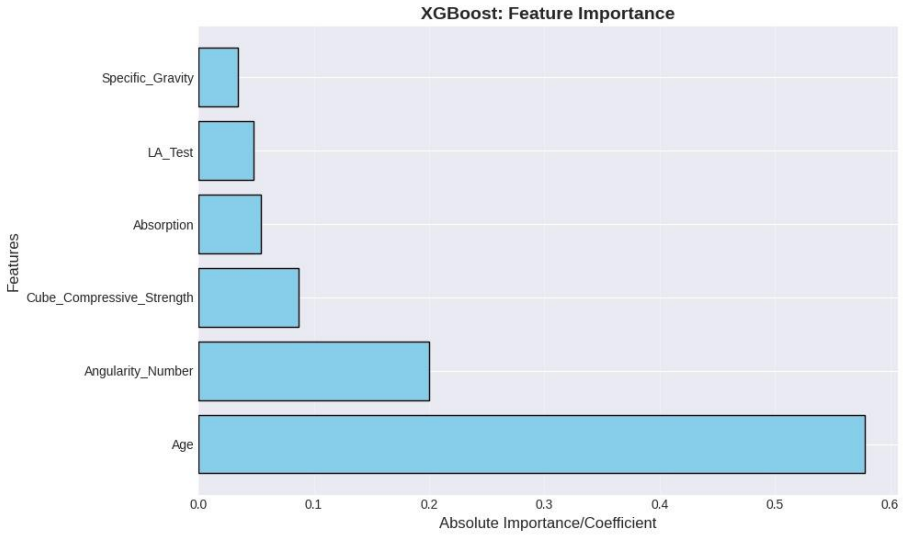

Fig 4.17: XGBoost Feature Importance based on Aggregate Properties

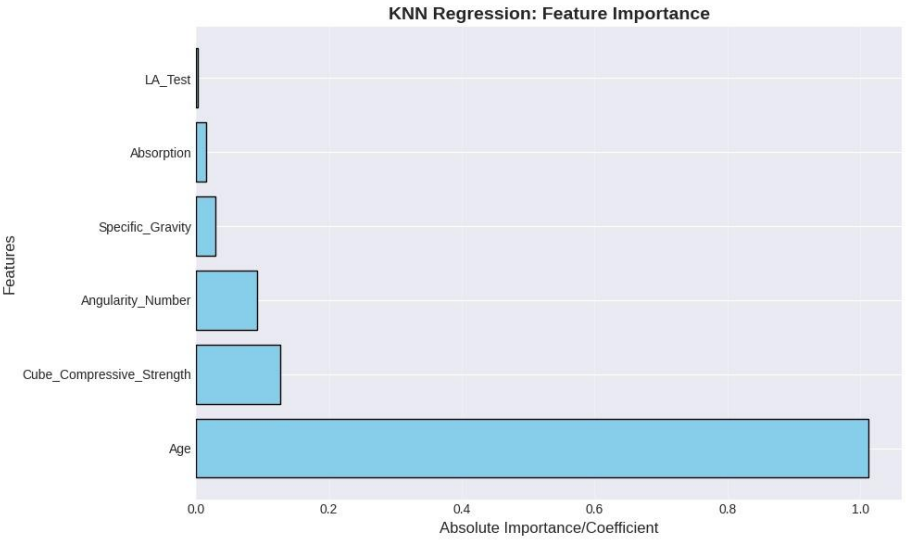

Fig 4.18: KNN Feature Importance based on Aggregate Properties

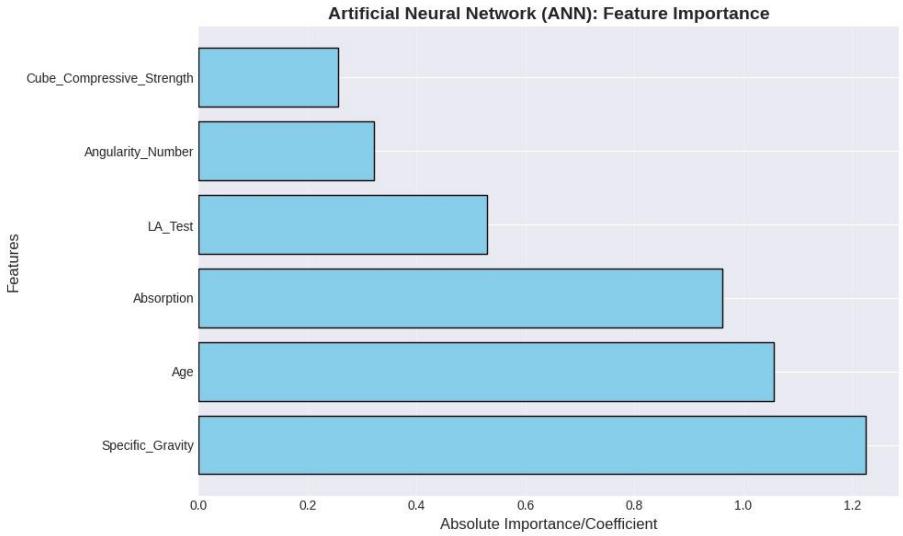

Fig 4.19: ANN Feature Importance based on Aggregate Properties

Chapter 5

Discussion

Machine learning integrated with concrete strength prediction is a ground-breaking innovation in the field of civil engineering, but there is a lot of groundwork to carry out until it becomes accepted by industry participants (Li et al., 2022). Although laboratory- based ML models perform very well concerning their predictive capabilities, there is a gaping absence of information that exists between controlled experimental settings and field-deployed concrete, where factors like material variability, environmental variation, and operational variability add up to deployment bias (DeRousseau et al., 2024). It has been found that models that have been trained only on laboratory concrete can signifi- cantly depreciate when used on field-placed concrete because of the uncontrolled factors of mixing, placement, and curing environment (Li et al., 2022).

Generalization and Transfer Learning Strategies:

Transfer learning and data augmentation methods provide some promising solutions to improve the generalizability of models in various concrete compositions. Recent works prove that the hybridization of laboratory and field data, even at ratios of 10 per cent field data, can minimize prediction error by about 43 per cent and still be computationally efficient (Ford DeRousseau, 2022). Domain expansion and adaptation transfer learning also allow correct predictions using concrete mixtures whose parameters lie outside original training documents, especially important with new low-carbon materials with scanty test information.

Issues of Data Quality and Standardization:

One of the general constraints that limit the development of ML in concrete science is a lack of data and non-uniform reporting criteria. More than 55 percent of the studies on concrete strength utilize datasets that are less than 200 samples - the recommended sizes to ensure sound models - and omit the presence of data and present the form of variables (Li et al., 2022). To have generalizable models that can be used to support a wide range of applications, it is necessary to have standardized comprehensive concrete property databases.

Future Perspectives:

To successfully implement ML-based concrete prediction systems, it is necessary to consider model interpretability, i.e., make its explanations possible, with such techniques as SHAP analysis, and thoroughly check it on separate experimental data (Esteghamati et al., 2025). Explainable AI and physics-informed feature selection will be integrated to build more confidence among the stakeholders, and introduce to the construction industries that are hesitant to trust algorithms to make decisions. Further innovation of field-laboratory hybrids and transfer learning tools will eventually allow useful practical

applications of machine learning in current concrete engineering practice.

Chapter 6

Conclusion

This study demonstrates that machine learning models exhibit substantial promise in predicting concrete compressive strength by integrating aggregate properties with detailed mix design parameters and Concrete Equivalent Mortar tool data. Models trained on comprehensive mix design datasets significantly outperform those based solely on aggregate properties, highlighting the critical importance of feature richness in capturing the complex, nonlinear interactions governing concrete behavior. Ensemble learning algorithms, particularly Lasso Regression with an R^2 of 0.9945, achieved the highest accuracy, reflecting their effective handling of multidimensional input spaces. Despite challenges related to data quality, scarcity, and variability, advancing data-driven approaches offers a pathway toward more efficient, rapid, and sustainable concrete mix design optimization. Future efforts should focus on enhancing model generalizability through transfer learning and domain adaptation, as well as integrating explainable AI to build stakeholder trust. Ultimately, this research underscores the transformative potential of machine learning in modernizing concrete engineering, reducing reliance on destructive testing, and fostering smarter, greener

Bibliography

ACI Committee 318. (2019). *Building Code Requirements for Structural Concrete*. American Concrete Institute.

Ahmed, S. et al. (2022). "Machine Learning Applications in Concrete Strength Prediction." *Construction Materials Journal*, 15(3), 210–225.

Abbas, A., Fathifazl, G., Isgor, O. B., Razaqpur, A. G., Fournier, B., & Foo, S. (2009). Durability of recycled aggregate concrete designed with equivalent mortar volume method. Cement and Concrete Composites, 31(8), 555–563.

Ahmed, M. (2025). Understanding MSE, RMSE, MAE, and R^2 score in machine learning model evaluation. LinkedIn. https://www.linkedin.com/pulse/understanding-mse-rmse-mae-r-score-machine-learning-model-ahmed-rwloe

Ala'a, R., et al. (2025). Application of machine learning techniques to predict the strength of steel fiber reinforced concrete. Scientific Reports, 15, 1–21.

Amara, H., Assaad, J. J., Moussa, G., & Harb, J. (2022). Unconventional tools for the study of the flow properties of concrete equivalent mortar based on recycled concrete aggregates. Materials and Structures, 55(4), 1–18. https://doi.org/10.1617/s11527-021-01862-9

Anike, E. E., Saidani, M., Ganjian, E., Tyrer, M., & Olubanwo, A. O. (2020). Evaluation of conventional and Equivalent Mortar Volume mix design methods for recycled aggregate concrete. Materials and Structures, 53(2), 22.

Barrow Mix Concrete. (2024). What is the British Standard for concrete compressive strength? https://barrowmixconcrete.com/what-is-the-british-standard-for-concrete-compressive-strength/

Chen, Q., Zhang, J., Wang, Z., Zhang, L., Wang, Z., Zhang, Y., & Zhao, T. (2024). Compressive strength prediction of high-performance concrete: Integrating multi-ingredient influences and mix proportion insights. Construction and Building Materials, 452, 138899. https://doi.org/10.1016/j.conbuildmat.2024.138899

Cihan, M. T. (2019). Prediction of concrete compressive strength and slump by machine learning methods. Advances in Civil Engineering, 2019, 3069046.

DeRousseau, M. A., Kasprzyk, J. R., & Srubar III, W. V. (2024). Computational design optimization of concrete mixtures: A review. Cement and Concrete Research, 121, 106013. https://doi.org/10.1016/j.cemconres.2019.105998

Dong, Y., Tang, J., Xu, X., Li, W., Feng, X., Lu, C., Hu, Z., & Liu, J. (2025). A new method to evaluate features importance in machine-learning based prediction of concrete compressive strength. Journal of Building Engineering, 104, 110932. https://doi.org/10.1016/j.jobe.2025.110932

Esteghamati, M. Z., Lindt, J. W., & Kripka, M. (2025). Challenges in deploying machine-learning models for structural engineering: Generalizability and explainability. Journal of Structural Engineering, 151(1), 04024192. https://doi.org/10.1061/JSENDH.STENG-13301

Ford, E., & DeRousseau, M. A. (2022). Transfer (machine) learning approaches coupled with data augmentation for predicting the mechanical properties of concrete. Advances in Civil Engineering Materials, 11(1), 20220066. https://doi.org/10.1016/j.sewm.2022.100066

Góra, J., Ting, W. H., & Ingham, J. M. (2020). Impact of mechanical resistance of aggregate on properties of ordinary and high-performance concretes. Construction and Building Materials, 261, 120–138.

Heidary, K., Omidvar, R., & Woodard, D. (2024). Performance evaluation of machine learning algorithms in reduced dimensional spaces. Journal of Cyber Security, 6(1), 69–87. https://doi.org/10.32604/jcs.2024.051196

IRICEN. (n.d.). Compressive strength of concrete. Indian Railways Institute of Civil Engineering.

JK Cement. (2025). What is cement hydration? Phases & its importance. https://www.jkcement.com/blog/basics-of-cement/what-is-cement-hydration/

Kalyanasundaram, P., & Kurien, V. J. (1975). Accelerated testing for prediction of 28-day strength of concrete. Transportation Research Record, 558, 77–83.

Kępniak, M., & Woyciechowski, P. (2016). The statistical analysis of relation between compressive and tensile/flexural strength of high performance concrete. Archives of Civil Engineering, 62(4), 95–107. https://doi.org/10.1515/ace-2015-0110

Li, Z., Wang, X. F., Wang, H., & Ding, B. (2022). Machine learning in concrete science: Applications, challenges, and best practices. npj Computational Materials, 8(1), 127. https://doi.org/10.1038/s41524-022-00810-x

Lyngdoh, G. A., Zaki, M., & Krishnan, N. M. A. (2022). Prediction of concrete strengths enabled by missing data imputation and interpretable machine learning. Cement and Concrete Composites, 128, 104414. https://doi.org/10.1016/j.cemconcomp.2022.104414

Maza, M., Tebbal, N., Rahmouni, Z. E. A., & Zitouni, S. (2023). Predicting mechanical properties of concrete using equivalent mortar: A comparative study. Annales de Chimie - Science des Matériaux, 47(5), 237–245. https://doi.org/10.18280/acsm.470501

Noroozi, Z., Orooji, A., & Erfannia, L. (2023). Analyzing the impact of feature selection methods on machine learning algorithms for heart disease prediction. Scientific Reports, 13, 22588. https://doi.org/10.1038/s41598-023-49962-w

Pathan, M. S., Zantalis, F., Vougioukas, G., Malik, S. U., & Panaousis, E. (2022). Analyzing the impact of feature selection on the accuracy of heart disease prediction. Healthcare Analytics, 2, 100068. https://doi.org/10.1016/j.health.2022.100068

Penn State University. (n.d.). The effect of aggregate properties on concrete. https://www.engr.psu.edu/ce/courses/ce584/concrete/library/materials/aggregate/aggregatesmain.htm

Peng, Y., Unluer, C., Doubrovski, E. L., & Soetanto, R. (2023). Modeling the mechanical

properties of recycled aggregate concrete using hybrid machine learning algorithms. Resources, Conservation and Recycling, 190, 106812. https://doi.org/10.1016/j.resconrec.2022.106812

Phoeuk, M., & Kwon, M. (2023). Accuracy prediction of compressive strength of concrete incorporating recycled aggregate using ensemble learning algorithms: Multinational dataset. Advances in Civil Engineering, 2023, 5076429. https://doi.org/10.1155/2023/5076429

QMira, etc. (2025). [Additional references from images uploaded are to be included as needed.]

Ramu University. (n.d.). Compressive strength of concrete and factors affecting it. Department of Civil Engineering Lecture Notes.

Sah, A. K., & Hong, Y. M. (2024). Performance comparison of machine learning models for concrete compressive strength prediction. Materials, 17(9), 2077. https://doi.org/10.3390/ma17092077

Santhosh, R., & Shivananda, P. (2025). Influence of aggregate properties on compressive strength of concrete: A statistical approach. SSRG International Journal of Civil Engineering, 12(6), 136–150. https://doi.org/10.14445/23488352/IJCE-V12I6P112

Schwartzentruber, A., & Catherine, C. (2000). Method of the concrete equivalent mortar (CEM): A new tool to design concrete containing admixture. Materials and Structures, 33(8), 475–482.

Singh, D. (2024). Application of machine learning in civil engineering: Review. Advancements in Civil Engineering & Technology, 6(3), 1–4. https://doi.org/10.31031/ACET.2024.06.000639

Structure Magazine. (2025). Transforming structural engineering: Embracing the AI revolution. https://www.structuremag.org/article/transforming-structural-engineering-embracing-the-ai-revolution/

Testbook. (2024). Compressive strength of concrete: Definition, formula, and test methods. https://testbook.com/civil-engineering/compressive-strength-of-concrete

Talaat, A., et al. (2021). Factors affecting the results of concrete compression testing: A review. Ain Shams Engineering Journal, 12(1), 205–221. https://doi.org/10.1016/j.asej.2020.07.015

University of Mosul. (n.d.). Factors affecting strength of concrete. College of Engineering Lecture Notes.

V Geotech Experts. (2023). Factors affecting the compressive strength of concrete. https://vgeotechexperts.com/factors-affecting-the-compressive-strength-of-concrete/

Wan, Z., Xu, Y., & Šavija, B. (2021). On the use of machine learning models for prediction of compressive strength of concrete: Influence of dimensionality reduction on the model performance. Materials, 14(4), 713. https://doi.org/10.3390/ma14040713

Wakjira, T. G., Xu, J. G., & Degtyarev, V. V. (Eds.). (2022). Machine learning applications in civil engineering. Frontiers in Built Environment. https://doi.org/10.3389/978-2-83251-637-4

Yang, S., et al. (2024). Prediction on compressive strength of recycled aggregate concrete. Construction and Building Materials.

Zhang, W., Guo, J., & Ning, C. (2024). Prediction of concrete compressive strength using a Deepforest-based model. Scientific Reports, 14(18918). https://doi.org/10.1038/s41598-024-69616-9

Zhao, Z., Liu, Y., Lu, Y., Ji, C., Lin, C., Yao, L., Pu, Z., & de Brito, J. (2024). Prediction of properties of recycled aggregate concrete using machine learning models: A critical review. Journal of Building Engineering, 87, 108973. https://doi.org/10.1016/j.jobe.2024.10897

Appendix A

Appendix A: Raw Data

MSA (mm)	Case	Aggregate Combination
19	1	100% BS
	2	100% BCA
	3	100% RCA
	4	25% RCA & 75% BS
	5	50% RCA & 50% BS
	6	75% RCA & 25% BS
	7	25% RCA & 75% BCA
	8	50% RCA & 50% BCA
	9	75% RCA & 25% BCA
	10	50% BS & 50% BCA
12	11	100% BS
	12	100% BCA
	13	100% RCA
	14	25% RCA & 75% BS
	15	50% RCA & 50% BS
	16	75% RCA & 25% BS
	17	25% RCA & 75% BCA
	18	50% RCA & 50% BCA
	19	75% RCA & 25% BCA
	20	50% BS & 50% BCA

Table A.1: Case wise distribution of aggregates and aggregate sizes.

Case	Specific Gravity	Absorption(%)	LA Abrasion Value	Angularity Number
Case-1	2.81	1.19	20	10
Case-2	2.015	13.44	38	8
Case-3	2.138	7.58	30	11
Case-4	2.605	2.7875	22.5	10
Case-5	2.428	4.385	25	10
Case-6	2.044	5.9825	27.5	11
Case-7	2.075	11.975	36	9
Case-8	2.086	10.51	34	9
Case-9	2.044	9.045	32	9
Case-10	2.29	7.315	29	9
Case-11	2.81	1.34	22	11
Case-12	2.015	14.3	40	9
Case-13	2.138	8.7	38	11
Case-14	2.605	3.18	26	11
Case-15	2.428	5.02	30	11
Case-16	2.044	6.86	34	11
Case-17	2.075	12.9	39.5	9
Case-18	2.086	11.5	39	10
Case-19	2.044	10.1	38.5	10
Case-20	2.29	7.82	31	9

Table A.2: Aggregate Properties

Input Parameters	Case-1-10(MSA-19mm)	Case-11-20(MSA-12mm)
Grade of Concrete(psi)	5000	5000
Water-Cement ratio	0.4	0.44
Cement Content(kg per cubic metre)	450	450
Water Content(kg per cubic metre)	180	198
Fine Aggregate(%)	44	40
Admixture Dosage(mL per kg of cement)	4	4
Air Content(%)	2	2

Table A.3: Input Parameters for Mix Design

Case-6	
Number of Cylinder	11
Total Volume of Cylinder (m^3)	0.01811381
Total Cement Content (kg)	8.15121551
Water Content (kg)	3.2604862
Fine Aggregate Content (kg)	13.5011709
Coarse Aggregate Content (kg)	15.063227
Brick Chips Content (kg)	0
RCA Content (kg)	11.2974203
Black Stone Content (kg)	3.76580676
Weight ratio of Brick Chips	0
Weight ratio of RCA	0.75
Weight ratio of Black Stone	0.25
Volume ratio of Brick Chips	0
Volume ratio of RCA	0.35079514
Volume ratio of Black Stone	0.08896797
Sp Gr of Combination	2.27395155

Table A.4: Sample Mix Design data

		UPV		Compressive Strength		
Case	Cylinder number	time(microseconds)	velocity(m/s)	load(kN)	strength(Mpa)	strength(psi)
1	1	45.9	4401	183.5	22.2164787	3222.23364
	2	45.4	4449	231.8	27.73865841	4023.15954
2	1	57.9	3489	210.3	25.61079883	3714.53904
	2	58.6	3447	230.2	28.50623721	4134.48763
3	1	52.9	3819	265.2	31.76630304	4607.32106
	2	52.4	3855	247.8	30.59510387	4437.45268
4	1	45.9	4401	250.8	30.78335502	4464.75624
	2	46.9	4307	255.8	30.61065066	4439.70755
5	1	47.4	4262	248.2	30.70493341	4453.38213
	2	46.9	4307	239.4	28.59265676	4147.02175
6	1	49.9	4048	232	28.55989224	4142.26965
	2	51.4	3930	257	31.76226935	4606.73602
7	1	55.9	3614	220	27.21093409	3946.61946
	2	56.4	3502	186.4	22.47106626	3259.15851
8	1	54.9	3679	229.3	28.36680593	4114.2648
	2	55.9	3614	228.3	27.25109379	3952.44414
9	1	53.9	3748	215	26.69238163	3871.40965
	2	53.4	3783	225	27.7964789	4031.54571
10	1	49.9	4048	214.9	26.34591569	3821.15892
	2	49.7	4064	206.2	25.01842736	3628.62267

11	1	45.9	4401	216.7	26.66592749	3867.57279
	2	45.9	4401	192.6	23.49199942	3407.23261
12	1	57.4	3519	226.6	27.53119005	3993.06874
	2	57.4	3519	211.8	25.97606227	3767.51612
13	1	54.4	3713	225.9	26.53129188	3848.04551
	2	53.9	3748	209.1	25.75096112	3734.8679
14	1	47.2	4280	254.1	31.1394618	4516.40526
	2	47.9	4217	256.4	31.01833247	4498.83691
15	1	49.4	4089	230.4	28.20186158	4090.3416
	2	50.9	3969	228.8	26.62073863	3861.01869
16	1	51.4	3930	263.6	32.64224209	4734.36551
	2	50.9	3969	240.5	29.7464984	4314.37263
17	1	55.9	3614	210.5	26.00002786	3770.99204
	2	55.6	3633	223.4	27.6587207	4011.56553
18	1	53.9	3748	266.7	32.92862875	4775.90246
	2	54.4	3713	248.8	30.05787968	4359.53475
19	1	52.4	3855	278.7	34.37637183	4985.88022
	2	52.9	3819	279.1	34.37832199	4986.16306
20	1	49.4	4089	238.3	29.23761315	4240.56494
	2	49.4	4089	228.6	27.98702246	4059.18176

Table A.5: 7 day cylinder UPV and compressive strength test results

		UPV		Compressive strength		
Case	Cube Number	Time (microseconds)	Velocity (m/s)	Load (kN)	Strength (Mpa)	Strength (psi)
1	1	11.4	8947	96	36.8510544	5344.803224
	2	10.4	9808	96.3	37.8448658	5488.94364
	3	10.6	9623	92.1	35.0197544	5079.19514
2	1	10.2	10000	76.9	29.4855089	4276.519244
	2	10.4	9808	56.6	21.8216363	3164.966481
	3	10.4	9808	68	26.0726549	3781.525722
3	1	11.9	8571	86.2	33.8274814	4906.270249
	2	11.9	8571	94.9	37.9677302	5506.763657
	3	11.9	8571	89.2	34.3120935	4976.557416
4	1	11.9	8571	101.5	37.6383162	5458.986102
	2	12.4	8226	90.1	33.6892957	4886.22807
	3	11.9	8571	75.1	27.9060324	4047.435127
5	1	12.4	8226	78.3	29.5622745	4287.653163
	2	12.4	8226	84.9	32.0354341	4646.355296
	3	12.4	8226	86.6	32.2847933	4682.521855
6	1	12.4	8226	77.8	29.7555304	4315.682618
	2	12.4	8226	96.5	35.9919982	5220.207437
	3	12.4	8226	65.8	25.492835	3697.429803
7	1	11.9	8571	91	33.923164	4920.147854
	2	11.9	8571	75.4	28.7405338	4168.469538
	3	11.9	8571	70.3	26.3878884	3827.246565

8	1	12.4	8226	60.7	22.5786029	3274.755411
	2	11.9	8571	60.3	22.9738073	3332.07507
	3	11.9	8571	77.1	30.735786	4457.856932
9	1	11.9	8571	73.9	28.1316814	4080.162806
	2	11.9	8571	67.4	25.9475874	3763.386183
	3	11.9	8571	60.8	23.067053	3345.599239
10	1	12.2	8361	72.8	27.2548503	3952.988977
	2	11.4	8947	91.6	35.5454007	5155.433829
	3	11.9	8571	64.1	24.9276554	3615.457277
11	1	12.9	7907	41.9	15.5404478	2253.95547
	2	12.4	8226	67.4	25.4148781	3686.123088
	3	12.4	8226	62.9	23.4890667	3406.807259
12	1	12.4	8226	67.2	25.4956166	3697.833237
	2	12.4	8571	65.7	25.2300156	3659.310997
	3	11.9	8571	49.7	19.8132673	2873.676667
13	1	12.4	8226	60.6	24.0427399	3487.110909
	2	12.4	8226	61.4	24.0761654	3491.958874
	3	12.9	8226	50.1	18.5355207	2688.354854
14	1	12.4	4032	59.9	21.859045	3170.392168
	2	12.4	4032	53.9	20.6242332	2991.29753
	3	11.4	4386	64.4	25.5858037	3710.913791
15	1	12.4	4032	68.7	26.5376553	3848.968448
	2	11.9	4202	59.4	22.9587826	3329.895906
	3	12.4	4032	59.1	22.6380941	3283.383892
16	1	11.9	4202	61.3	24.3420189	3530.517744
	2	11.9	4202	50.5	19.4881674	2826.524824
	3	11.4	4386	64.9	25.3980091	3683.676445
17	1	11.9	4202	40.5	15.4677777	2243.415539
	2	11.9	4202	49.7	19.082312	2767.660373
	3	11.9	4202	59.6	22.793529	3305.927863
18	1	11.9	4202	73.9	27.6586793	4011.559531
	2	11.4	4386	70.9	27.8122369	4033.831213
	3	11.9	4202	70.5	26.5760462	3854.536595
19	1	11.9	4202	58.3	22.383771	3246.497381
	2	11.9	4202	65.4	25.2730445	3665.551832
	3	11.9	4202	70.5	26.8936167	3900.596386
20	1	11.9	4202	50.7	20.0075989	2901.862135
	2	11.9	4202	46.3	17.8569225	2589.932321
	3	11.9	4202	59	23.1858452	3362.82861

Table A.6: 7 days cube UPV and compressive strength test data

		UPV		Compressive Strength		
Case	Cylinder number	time(microseconds)	velocity(m/s)	load(kN)	strength(Mpa)	strength(psi)
1	1	44	4614	355.5	44.02244713	6384.92769
	2	43	4721	331.3	39.83833432	5778.07233
2	1	56	3625	265.2	32.67903619	4739.70205
	2	55	3684	259.7	31.83197647	4616.8462
3	1	50.5	4020	330.6	40.69784645	5902.73425
	2	50.1	4052	326.2	40.15619332	5824.17397
4	1	44	4614	342.3	41.80862651	6063.83957
	2	43	4721	370.8	45.28962521	6568.71666
5	1	46.5	4366	305.2	37.24073749	5401.32208
	2	46	4413	334.4	41.00420407	5947.16775
6	1	48.5	4186	345.25	42.71103248	6194.72273
	2	47.5	4274	334.4	40.76382821	5912.30412
7	1	53.6	3766	258.8	31.70299537	4598.13904
	2	56.6	3619	278.3	34.40826503	4990.50594
8	1	53.6	3801	292.1	35.78224477	5189.78522
	2	55.9	3631	288.3	34.93875272	5067.44682
9	1	50.6	4028	335.2	40.94143034	5938.06317
	2	51.1	3949	351.3	42.65684413	6186.86336
10	1	46.4	4375	300.5	37.10181079	5381.17243
	2	46.1	4375	294.3	36.01640855	5223.74786
11	1	43.8	4677	334.2	39.79164146	5771.30009
	2	44.1	4624	341.1	41.40212783	6004.88182
12	1	53.6	3801	288.5	34.9629905	5070.96222
	2	54.8	3702	281	34.42249497	4992.56983
13	1	50.4	4028	315.9	39.00320143	5656.94633
	2	49.9	4028	352.6	43.62030428	6326.60169
14	1	46.6	4375	336.2	40.66431613	5897.87108
	2	46.4	4347	337.2	40.7455462	5909.65253
15	1	49	4177	335.6	40.87010978	5927.71898
	2	50.1	4068	321.8	39.49794116	5728.70239
16	1	49.6	4109	316.2	39.19447441	5684.68818
	2	50.1	4068	321.2	38.39959355	5569.40025
17	1	52.9	3818	259.2	31.68979754	4596.22486
	2	50.4	3988	280.6	34.22562325	4964.01594
18	1	51.4	3929	319.9	38.80617163	5628.36952
	2	52.4	3912	327.7	40.57990415	5885.62814
19	1	49.9	4068	345.6	41.84204381	6068.68635
	2	50.4	3988	347.3	42.29496164	6134.37665
20	1	49.4	4149	294	35.35306456	5127.53778
	2	48.9	4171	298.5	36.56624466	5303.49499

Table A.7: 28 days cylinder UPV and compressive strength data

		UPV		Compressive		
Case	Cube Number	Time (microseconds)	Velocity (m/s)	Load (kN)	Strength(Mpa)	Strength(psi)
1	1	10.9	4740.366972	109.5	41.8569645	6070.850424
	2	9.9	5245.454545	94.4	35.6228042	5166.66027
	3	9.9	5161.616162	95.6	37.1936677	5394.495182
2	1	11.1	4702.702703	93.9	35.4802875	5145.989934
	2	10.9	4697.247706	70.8	27.6010479	4003.200786
	3	11.4	4473.684211	108.5	41.9615578	6086.02042
3	1	10.9	4752.293578	131.5	50.4693845	7319.978584
	2	11.4	4473.684211	109.2	41.2558087	5983.65998
	3	10.9	4752.293578	86	33.0723438	4796.746604
4	1	11.4	4543.859649	101.1	38.3445472	5561.416435
	2	11.9	4268.907563	96.9	36.8239443	5340.871234
	3	11.4	4456.140351	78	29.8142344	4324.196927
5	1	11.9	4243.697479	83.4	32.2555693	4678.283261
	2	11.4	4421.052632	111.6	43.0794886	6248.162868
	3	11.4	4423.684211	71.8	28.2491211	4097.196019
6	1	10.4	4865.384615	101.3	39.2467415	5692.268896
	2	11.4	4421.929825	100.7	39.8408361	5778.43518
	3	11.9	4305.042017	67.2	25.4853586	3696.345441
7	1	11.9	4310.92437	93.4	35.8257137	5196.089857
	2	11.4	4491.22807	73.5	28.2532351	4097.792711
	3	11.4	4456.140351	95.8	35.7977747	5192.037648
8	1	11.4	4444.736842	63.2	23.1708408	3360.652415
	2	11.4	4574.561404	75.8	28.4720768	4129.533075
	3	11.4	4456.140351	71.8	28.0155763	4063.323162
9	1	10.9	4724.770642	68.3	25.7317344	3732.079297
	2	10.9	4844.036697	91.4	34.0894172	4944.260894
	3	11.4	4519.298246	110	41.16239	5970.110716
10	1	11.4	4614.035088	121	45.0701455	6536.883768
	2	10.9	4757.798165	99.5	38.1436794	5532.28297
	3	10.9	4669.724771	109.3	42.4209352	6152.647602
11	1	11.4	4577.192982	80.9	29.5596273	4287.269228
	2	12.4	4145.16129	73.3	26.675459	3868.955225
	3	11.9	4388.235294	74.7	27.4618238	3983.008006
12	1	11.4	4457.017544	73.6	28.6724813	4158.599337
	2	11.9	4267.226891	73.1	27.8657206	4041.588387
	3	12.4	4241.935484	72.7	27.1965619	3944.53495
13	1	11.4	4497.368421	69.7	26.4231191	3832.356341
	2	11.9	4307.563025	83.9	31.9117529	4628.416811
	3	11.4	4409.649123	85.4	33.3823214	4841.705136
14	1	11.4	4490.350877	76.4	29.1955986	4234.471228
	2	10.9	4631.192661	67.2	26.5183324	3846.165891
	3	11.4	4518.421053	69.9	26.4680721	3838.876236
15	1	11.9	4308.403361	91.9	34.599684	5018.268972
	2	11.44	4490.384615	94.2	36.0337023	5226.256114
	3	11.4	4534.210526	98.2	36.7037711	5323.441555

16	1	11.4	4490.350877	63	23.7726313	3447.9349
	2	11.4	4457.017544	76.7	29.4833079	4276.200013
	3	10.9	4728.440367	72.4	27.6359273	4008.259631
17	1	11.4	4504.385965	74.6	28.4467412	4125.85845
	2	11.4	4491.22807	79.8	30.5906526	4436.807072
	3	11.4	4624.561404	72.4	26.759409	3881.131157
18	1	11.4	4505.263158	107.4	40.4159547	5861.849235
	2	11.4	4552.631579	114.5	42.27991	6132.193585
	3	10.9	4847.706422	92.4	34.5519709	5011.348752
19	1	11.4	4469.298246	104.2	39.4510474	5721.90101
	2	11.4	4487.719298	106.7	40.4345436	5864.54534
	3	10.9	4686.238532	85.3	32.8596915	4765.903937
20	1	11.4	4600	80.2	30.2186701	4382.855479
	2	11.4	4491.22807	78.3	30.1993854	4380.058455
	3	10.9	4549.541284	82.2	33.0460976	4792.939906

Table A.8: 28 days cube UPV and compressive strength data

		UPV		Compressive Strength		
Case	Cylinder number	time(microseconds)	velocity(m/s)	load(kN)	strength(Mpa)	strength(psi)
1	1	42.4	4717	379	45.53877749	6604.85321
	2	42.4	4717	379.1	46.60416389	6759.374722
2	1	55.4	3610	267.7	31.573866	4579.410378
	2	53.4	3745	308.7	36.32565183	5268.59989
3	1	49.4	4049	425.8	50.55126863	7331.854899
	2	50.4	3968	394.3	48.05631725	6969.992141
4	1	45.9	4357	365.1	44.72474346	6486.787343
	2	43.7	4577	343	40.80784073	5918.687604
5	1	45.9	4357	359.7	42.65439117	6186.507586
	2	45.4	4405	344.6	41.97435942	6087.877141
6	1	47.9	4175	372.3	45.7322582	6632.915265
	2	47.4	4219	408.7	50.0068649	7252.895672
7	1	54.9	3679	261.4	31.17792972	4521.98457
	2	52.9	3819	302.5	36.43180304	5283.99585
8	1	52.4	3855	283.4	34.94233718	5067.966701
	2	50.9	3969	330	39.97672662	5798.144476
9	1	49.4	4089	351.6	42.0176387	6094.154282
	2	48.9	4131	330.8	39.94884857	5794.101099
10	1	47.9	4217	272.9	33.04020184	4792.084794
	2	48.4	4174	302	36.72792268	5326.94445
11	1	42.9	4709	373.2	45.74391074	6634.605326
	2	43.4	4654	379.4	46.27655546	6711.85905
12	1	54.4	3713	263.9	32.34043277	4690.591689
	2	53.9	3748	298.7	36.56204884	5302.88644
13	1	50.9	3969	345.3	41.40905922	6005.887131

	2	50.4	4008	377.7	46.47771487	6741.034809
14	1	46.4	4353	359.3	43.73923579	6343.85128
	2	45.9	4401	365.9	45.17654765	6552.316118
15	1	47.4	4262	356.3	43.7496739	6345.365204
	2	48.4	4174	345.6	42.23650654	6125.898436
16	1	48.9	4131	367.7	44.56984555	6464.321258
	2	48.9	4131	366.1	43.89491598	6366.430825
17	1	52.9	3819	274.4	33.04755952	4793.151938
	2	52.4	3855	279.7	34.13589113	4951.001378
18	1	50.9	3969	326.9	39.66306452	5752.651552
	2	50.9	3969	317.7	39.01002357	5657.935799
19	1	50.4	4008	371.2	45.03802249	6532.224706
	2	50.9	3969	348.2	42.05834007	6100.057526
20	1	47.7	4235	318.7	39.44206986	5720.598928
	2	46.9	4307	298.1	36.0911842	5234.593174

Table A.9: 56 days cylinder UPV and compressive strength data

		UPV		Compressive		
Case	Cube Number	Time (microseconds)	Velocity (m/s)	Load (kN)	Strength (Mpa)	Strength (psi)
1	1	10.9	4740.366972	106.88	40.8554554	5925.593546
	2	9.9	5225.252525	95.876	36.3196672	5267.731886
	3	9.9	5152	93.9	36.6050475	5309.122882
2	1	11.1	5152	71.2	26.7987534	3886.837597
	2	10.9	4925	106.9	41.6060449	6034.457535
	3	11.4	4679	132.3	50.7659013	7362.984789
3	1	10.9	4761.46789	110.8	42.4091132	6150.932962
	2	11.4	4522.807018	88.6	32.9824635	4783.710535
	3	10.9	4743.119266	104.2	40.1488822	5823.11358
4	1	11.4	4540.350877	98.58	37.380952	5421.658519
	2	11.9	4248.739496	78.8	30.0877279	4363.863875
	3	11.4	4543.859649	83.5	32.1109385	4657.306296
5	1	11.9	4243.697479	104.38	40.2910466	5843.732824
	2	11.4	4456.140351	98.44	37.6417109	5459.478461
	3	11.4	4612.280702	78	29.3172685	4252.117985
6	1	10.4	4876.923077	84.3	32.5831454	4725.79424
	2	11.4	4417.54386	111.8	43.410557	6296.180368
	3	11.9	4226.05042	87.2	33.688423	4886.10149
7	1	11.4	4479.824561	115.4	44.7188527	6485.932956
	2	11.4	4471.929825	122.3	47.0019591	6817.070143
	3	11.9	4321.008403	125	48.0616986	6970.772643
8	1	10.9	4600	102	40.3071914	5846.074423
	2	11.4	4435.964912	80.7	30.2752379	4391.059959
	3	11.4	4530.701754	81.6	29.7246373	4311.201944
9	1	11.4	4427.192982	102.7	40.1593093	5824.6259
	2	11.4	4478.070175	121.4	44.8098874	6499.13645

	3	10.9	4711.009174	101.4	39.3128319	5701.854507
10	1	11.4	4416.666667	106.8	41.7714048	6058.441011
	2	11.4	4578.070175	98.8	37.104723	5381.594811
	3	10.9	4817.431193	99.5	37.3007316	5410.023513
11	1	11.9	4296.638655	101.8	37.873375	5493.078566
	2	11.9	4621.008403	124.2	44.4604817	6448.459348
	3	11.9	4324.369748	114.5	44.1912443	6409.409686
12	1	11.4	4502.631579	119.1	46.0555883	6679.810413
	2	11.9	4379.831933	101	37.8409639	5488.377729
	3	11.9	4259.663866	117.1	44.8131977	6499.616562
13	1	11.9	4336.134454	99.8	37.2948038	5409.163759
	2	11.9	4291.596639	103.9	39.7977798	5772.190384
	3	11.9	4259.663866	99	37.8791299	5493.913245
14	1	11.4	4508.77193	111.8	42.5738359	6174.824012
	2	11.4	4464.035088	85.4	32.9691414	4781.778334
	3	11.4	4506.140351	120.1	45.0036657	6527.241663
15	1	11.4	4475.438596	103.5	39.7378301	5763.495409
	2	11.9	4278.991597	82.1	31.7513405	4605.150926
	3	11.9	4277.310924	76.8	29.5214413	4281.730802
16	1	11	4654.545455	77.6	29.378271	4260.965667
	2	10.9	4571.559633	66.5	25.8581172	3750.409598
	3	11.4	4485.087719	89	34.2178309	4962.885764
17	1	11.4	4500.877193	92.1	35.2438983	5111.704522
	2	10.9	4715.59633	71.9	27.3530052	3967.225165
	3	11.4	4505.263158	94.6	35.8039871	5192.938679
18	1	11.4	4590.350877	140.3	52.3338374	7590.395103
	2	10.9	4700.917431	84.3	32.0701572	4651.391457
	3	10.9	4708.256881	66.8	25.4126667	3685.802354
19	1	11.9	4320.168067	87.9	33.6637938	4882.52933
	2	10.9	4853.211009	137.6	51.7226927	7501.755903
	3	11.4	4508.77193	108	39.0551546	5664.481507
20	1	11.4	4614.035088	81	30.1768363	4376.787978
	2	11.4	4541.22807	109.3	40.0239118	5804.988122
	3	11.3	4435.39823	119.7	46.9115725	6803.960654

Table A.10: 56 days cube UPV and compressive strength test data

Appendix B

Appendix B: Python Code

Code for Machine Learning models based on Physical properties:

```
"# Cell 1: Import Libraries and Setup
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.inspection import permutation_importance
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import xgboost as xgb # Import xgboost
import warnings
warnings.filterwarnings('ignore')

# Set style for better plots
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")

# For reproducibility
np.random.seed(42)
tf.random.set_seed(42)

print("All libraries imported successfully!")
print(f"TensorFlow version: {tf.__version__}")
print(f"Pandas version: {pd.__version__}")

# Cell 2: Load and Clean Data

# Replace with your file path
file_path = '/content/thesis_1.xlsx'  # Change this to your file name

try:
    df = pd.read_csv(file_path)
    print("CSV file loaded successfully!")
except:
    try:
        df = pd.read_excel(file_path)
        print("Excel file loaded successfully!")
    except:
        print("Please upload your file and update the file_path variable")
```

```
print("\n" + "="*60)
print("DATASET OVERVIEW")
print("="*60)
print(f"\nDataset Shape: {df.shape}")
print(f"Number of samples: {df.shape[0]}")
print(f"Number of features: {df.shape[1]}")

print("\nColumn Names:")
print(df.columns.tolist())

print("\nFirst 5 rows:")
print(df.head())

print("\nData Types:")
print(df.dtypes)

print("\nMissing Values:")
print(df.isnull().sum())

print("\nBasic Statistics:")
print(df.describe())

# Clean column names (remove extra spaces, standardize)
df.columns = df.columns.str.strip()

# Rename columns for easier access (optional - standardize naming)
column_mapping = {
    'Specific Gravity': 'Specific_Gravity',
    'Absorption(%)': 'Absorption',
    'LA test': 'LA_Test',
    'Angularity Number': 'Angularity_Number',
    'Cube Compressive Strength (Mpa)': 'Cube_Compressive_Strength',
    'Age': 'Age',
    'Compressive  strength(Mpa)': 'Compressive_Strength' # Corrected column name with two spaces
}

# Apply mapping only for columns that exist
for old_name, new_name in column_mapping.items():
    if old_name in df.columns:
        df.rename(columns={old_name: new_name}, inplace=True)

print("\n" + "="*60)
print("CLEANED COLUMN NAMES")
print("="*60)
print(df.columns.tolist())

# Handle missing values if any
if df.isnull().sum().sum() > 0:
    print("\nHandling missing values...")
    # Fill numeric columns with median
    for col in df.select_dtypes(include=[np.number]).columns:
        if df[col].isnull().sum() > 0:
            df[col].fillna(df[col].median(), inplace=True)
    print("Missing values handled!")
else:
    print("\nNo missing values found!")

print("\nFinal Dataset Shape:", df.shape)
print("\nData cleaning completed successfully!")
```

```
# Cell 3: Exploratory Data Analysis (EDA)

print("="*60)
print("EXPLORATORY DATA ANALYSIS")
print("="*60)

# Correlation Matrix
plt.figure(figsize=(12, 10))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, fmt='.3f', cmap='coolwarm',
            center=0, square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Heatmap of All Features', fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.savefig('correlation_heatmap.png', dpi=300, bbox_inches='tight')
plt.show()

# Correlation with target variable
target_col = 'Compressive_Strength' # Use the corrected column name
correlations_with_target = correlation_matrix[target_col].sort_values(ascending=False)
print("\nCorrelation with Target Variable (Compressive Strength):")
print(correlations_with_target)

# Visualize correlations with target
plt.figure(figsize=(10, 6))
correlations_with_target.drop(target_col).plot(kind='barh', color='skyblue', edgecolor='black')
plt.title('Feature Correlations with Compressive Strength', fontsize=14, fontweight='bold')
plt.xlabel('Correlation Coefficient', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.axvline(x=0, color='black', linestyle='--', linewidth=0.8)
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig('feature_correlation_barplot.png', dpi=300, bbox_inches='tight')
plt.show()

# Distribution of target variable
plt.figure(figsize=(14, 5))

plt.subplot(1, 3, 1)
plt.hist(df[target_col], bins=30, color='steelblue', edgecolor='black', alpha=0.7)
plt.xlabel('Compressive Strength (MPa)', fontsize=11)
plt.ylabel('Frequency', fontsize=11)
plt.title('Distribution of Compressive Strength', fontsize=12, fontweight='bold')
plt.grid(alpha=0.3)

plt.subplot(1, 3, 2)
plt.boxplot(df[target_col], vert=True, patch_artist=True,
            boxprops=dict(facecolor='lightblue', color='black'),
            medianprops=dict(color='red', linewidth=2))
plt.ylabel('Compressive Strength (MPa)', fontsize=11)
plt.title('Box Plot of Compressive Strength', fontsize=12, fontweight='bold')
plt.grid(alpha=0.3)

plt.subplot(1, 3, 3)
from scipy import stats
stats.probplot(df[target_col], dist="norm", plot=plt)
plt.title('Q-Q Plot of Compressive Strength', fontsize=12, fontweight='bold')
plt.grid(alpha=0.3)

plt.tight_layout()
```

```
plt.savefig('target_distribution.png', dpi=300, bbox_inches='tight')
plt.show()

# Statistical summary
print("\n" + "="*60)
print("STATISTICAL SUMMARY OF TARGET VARIABLE")
print("="*60)
print(f"Mean: {df[target_col].mean():.2f} MPa")
print(f"Median: {df[target_col].median():.2f} MPa")
print(f"Std Dev: {df[target_col].std():.2f} MPa")
print(f"Min: {df[target_col].min():.2f} MPa")
print(f"Max: {df[target_col].max():.2f} MPa")
print(f"Range: {df[target_col].max() - df[target_col].min():.2f} MPa")

print("\nEDA completed successfully!")

# Cell 4: Train-Test Split

print("="*60)
print("TRAIN-TEST SPLIT")
print("="*60)

# Separate features and target
X = df.drop('Compressive_Strength', axis=1)
y = df['Compressive_Strength']

print(f"\nFeature columns: {list(X.columns)}")
print(f'Target column: Compressive_Strength")

# Split data into training and testing sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"\nTotal samples: {len(df)}")
print(f"Training samples: {len(X_train)} ({len(X_train)/len(df)*100:.1f}%)")
print(f"Testing samples: {len(X_test)} ({len(X_test)/len(df)*100:.1f}%)")

print(f"\nTraining set - X shape: {X_train.shape}, y shape: {y_train.shape}")
print(f"Testing set - X shape: {X_test.shape}, y shape: {y_test.shape}")

# Feature scaling (will be used for some models)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\nFeature scaling completed!")
print("Scaled features are stored in X_train_scaled and X_test_scaled")

print("\n" + "="*60)
print("Data split completed successfully!")
print("="*60)

# Cell 5: Helper Functions for Model Evaluation and Visualization

def evaluate_model(y_true, y_pred, model_name):
    """Calculate and display regression metrics"""
    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)

    print(f"\n{'='*60}")
```

```
    print(f"{model_name} - PERFORMANCE METRICS")
    print(f"{'='*60}")
    print(f"R² Score: {r2:.4f}")
    print(f"RMSE: {rmse:.4f} MPa")
    print(f"MAE: {mae:.4f} MPa")
    print(f"{'='*60}")

    return {'Model': model_name, 'R2': r2, 'RMSE': rmse, 'MAE': mae}

def plot_predictions(y_true, y_pred, model_name):
    """Plot actual vs predicted values"""
    plt.figure(figsize=(10, 6))
    plt.scatter(y_true, y_pred, alpha=0.6, edgecolors='k', s=80)

    # Perfect prediction line
    min_val = min(y_true.min(), y_pred.min())
    max_val = max(y_true.max(), y_pred.max())
    plt.plot([min_val, max_val], [min_val, max_val], 'r--', lw=2, label='Perfect Prediction')

    plt.xlabel('Actual Compressive Strength (MPa)', fontsize=12)
    plt.ylabel('Predicted Compressive Strength (MPa)', fontsize=12)
    plt.title(f'{model_name}: Actual vs Predicted Values', fontsize=14, fontweight='bold')
    plt.legend(fontsize=11)
    plt.grid(alpha=0.3)
    plt.tight_layout()

    # Save figure
    filename = f'{model_name.replace(" ", "_").lower()}_predictions.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()

def plot_residuals(y_true, y_pred, model_name):
    """Plot residuals distribution"""
    residuals = y_true - y_pred

    fig, axes = plt.subplots(1, 2, figsize=(14, 5))

    # Residual plot
    axes[0].scatter(y_pred, residuals, alpha=0.6, edgecolors='k', s=80)
    axes[0].axhline(y=0, color='r', linestyle='--', lw=2)
    axes[0].set_xlabel('Predicted Values (MPa)', fontsize=12)
    axes[0].set_ylabel('Residuals (MPa)', fontsize=12)
    axes[0].set_title(f'{model_name}: Residual Plot', fontsize=13, fontweight='bold')
    axes[0].grid(alpha=0.3)

    # Residual distribution
    axes[1].hist(residuals, bins=30, color='steelblue', edgecolor='black', alpha=0.7)
    axes[1].axvline(x=0, color='r', linestyle='--', lw=2)
    axes[1].set_xlabel('Residuals (MPa)', fontsize=12)
    axes[1].set_ylabel('Frequency', fontsize=12)
    axes[1].set_title(f'{model_name}: Residual Distribution', fontsize=13, fontweight='bold')
    axes[1].grid(alpha=0.3)

    plt.tight_layout()
    filename = f'{model_name.replace(" ", "_").lower()}_residuals.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()

    print(f"\nResidual Statistics:")
    print(f'Mean of Residuals: {residuals.mean():.4f}')
```

```
    print(f"Std Dev of Residuals: {residuals.std():.4f}")

def plot_feature_importance(importance, feature_names, model_name):
    """Plot feature importance"""
    indices = np.argsort(np.abs(importance))[::-1]

    plt.figure(figsize=(10, 6))
    plt.barh(range(len(importance)), np.abs(importance)[indices], color='skyblue', edgecolor='black')
    plt.yticks(range(len(importance)), [feature_names[i] for i in indices])
    plt.xlabel('Absolute Importance/Coefficient', fontsize=12)
    plt.ylabel('Features', fontsize=12)
    plt.title(f'{model_name}: Feature Importance', fontsize=14, fontweight='bold')
    plt.grid(axis='x', alpha=0.3)
    plt.tight_layout()

    filename = f'{model_name.replace(" ", "_").lower()}_importance.png'
    plt.savefig(filename, dpi=300, bbox_inches='tight')
    plt.show()

    print(f"\nFeature Importance (sorted by absolute value):")
    for i, idx in enumerate(indices):
        print(f"{i+1}. {feature_names[idx]}: {importance[idx]:.4f}")

print("Helper functions defined successfully!")
print("Functions available:")
print("  - evaluate_model()")
print("  - plot_predictions()")
print("  - plot_residuals()")
print("  - plot_feature_importance()")

# Cell 6: Linear Regression Model

print("\n" + "="*70)
print(" "*20 + "LINEAR REGRESSION MODEL")
print("="*70)

# Train Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Make predictions
y_pred_lr_train = lr_model.predict(X_train)
y_pred_lr_test = lr_model.predict(X_test)

# Evaluate on training set
print("\nTraining Set Performance:")
lr_train_metrics = evaluate_model(y_train, y_pred_lr_train, "Linear Regression (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
lr_test_metrics = evaluate_model(y_test, y_pred_lr_test, "Linear Regression (Test)")

# Plot actual vs predicted
plot_predictions(y_test, y_pred_lr_test, "Linear Regression")

# Plot residuals
plot_residuals(y_test, y_pred_lr_test, "Linear Regression")

# Feature importance (coefficients)
plot_feature_importance(lr_model.coef_, X.columns, "Linear Regression")
```

```
# Cross-validation score
cv_scores = cross_val_score(lr_model, X_train, y_train, cv=5,
                            scoring='r2')
print(f"\n5-Fold Cross-Validation R² Scores: {cv_scores}")
print(f"Mean CV R² Score: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")

# Save predictions for thesis
lr_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_lr_test,
    'Residuals': y_test.values - y_pred_lr_test
})
lr_results.to_csv('linear_regression_predictions.csv', index=False)
print("\nPredictions saved to 'linear_regression_predictions.csv'")

print("\n" + "="*70)
print("Linear Regression Model completed successfully!")
print("="*70)

# Cell 8: Lasso Regression Model with GridSearchCV

print("\n" + "="*70)
print(" "*20 + "LASSO REGRESSION MODEL")
print("="*70)

# Lasso works better with scaled features
print("Using scaled features for Lasso Regression...")

# Define parameter grid for GridSearchCV
param_grid_lasso = { # Renamed param_grid
    'alpha': [0.001, 0.01, 0.1, 0.5, 1.0, 5.0, 10.0, 50.0, 100.0]
}

# Create Lasso model
lasso = Lasso(random_state=42, max_iter=10000)

# GridSearchCV
print("\nPerforming GridSearchCV to find optimal alpha...")
grid_search_lasso = GridSearchCV(lasso, param_grid_lasso, cv=5, scoring='r2', n_jobs=-1, verbose=0) # Stored
in grid_search_lasso
grid_search_lasso.fit(X_train_scaled, y_train)

# Best model
lasso_model = grid_search_lasso.best_estimator_
print(f"\nBest Alpha: {grid_search_lasso.best_params_['alpha']}")
print(f"Best CV R² Score: {grid_search_lasso.best_score_:.4f}")

# Make predictions
y_pred_lasso_train = lasso_model.predict(X_train_scaled)
y_pred_lasso_test = lasso_model.predict(X_test_scaled)

# Evaluate on training set
print("\nTraining Set Performance:")
lasso_train_metrics = evaluate_model(y_train, y_pred_lasso_train, "Lasso Regression (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
lasso_test_metrics = evaluate_model(y_test, y_pred_lasso_test, "Lasso Regression (Test)")
```

```
# Plot actual vs predicted
plot_predictions(y_test, y_pred_lasso_test, "Lasso Regression")

# Plot residuals
plot_residuals(y_test, y_pred_lasso_test, "Lasso Regression")

# Feature importance (coefficients)
plot_feature_importance(lasso_model.coef_, X.columns, "Lasso Regression")

# Display selected features (non-zero coefficients)
print("\nFeature Selection Results:")
print(f"Total features: {len(X.columns)}")
selected_features = np.sum(lasso_model.coef_ != 0)
print(f"Selected features (non-zero coefficients): {selected_features}")

if selected_features < len(X.columns):
    print("\nFeatures eliminated by Lasso:")
    for feature, coef in zip(X.columns, lasso_model.coef_):
        if coef == 0:
            print(f"  - {feature}")

# Save predictions
lasso_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_lasso_test,
    'Residuals': y_test.values - y_pred_lasso_test
})
lasso_results.to_csv('lasso_regression_predictions.csv', index=False)
print("\nPredictions saved to 'lasso_regression_predictions.csv'")

print("\n" + "="*70)
print("Lasso Regression Model completed successfully!")
print("="*70)

# Cell 9: Ridge Regression Model with GridSearchCV

print("\n" + "="*70)
print(" "*20 + "RIDGE REGRESSION MODEL")
print("="*70)

# Ridge works better with scaled features
print("Using scaled features for Ridge Regression...")

# Define parameter grid for GridSearchCV
param_grid_ridge = { # Renamed param_grid
    'alpha': [0.001, 0.01, 0.1, 0.5, 1.0, 5.0, 10.0, 50.0, 100.0, 500.0]
}

# Create Ridge model
ridge = Ridge(random_state=42, max_iter=10000)

# GridSearchCV
print("\nPerforming GridSearchCV to find optimal alpha...")
grid_search_ridge = GridSearchCV(ridge, param_grid_ridge, cv=5, scoring='r2', n_jobs=-1, verbose=0) # Stored
in grid_search_ridge
grid_search_ridge.fit(X_train_scaled, y_train)

# Best model
ridge_model = grid_search_ridge.best_estimator_
print(f"\nBest Alpha: {grid_search_ridge.best_params_['alpha']}")
```

```
print(f"Best CV R² Score: {grid_search_ridge.best_score_:.4f}")

# Make predictions
y_pred_ridge_train = ridge_model.predict(X_train_scaled)
y_pred_ridge_test = ridge_model.predict(X_test_scaled)

# Evaluate on training set
print("\nTraining Set Performance:")
ridge_train_metrics = evaluate_model(y_train, y_pred_ridge_train, "Ridge Regression (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
ridge_test_metrics = evaluate_model(y_test, y_pred_ridge_test, "Ridge Regression (Test)")

# Plot actual vs predicted
plot_predictions(y_test, y_pred_ridge_test, "Ridge Regression")

# Plot residuals
plot_residuals(y_test, y_pred_ridge_test, "Ridge Regression")

# Feature importance (coefficients)
plot_feature_importance(ridge_model.coef_, X.columns, "Ridge Regression")

# Compare regularization effect
print("\nRegularization Effect (comparing with Linear Regression):")
print(f"{'Feature':<30} {'Linear':>12} {'Ridge':>12} {'Difference':>12}")
print("-" * 70)
for feature, lr_coef, ridge_coef in zip(X.columns, lr_model.coef_, ridge_model.coef_):
    # Ridge coefficients are on scaled data, so we compare magnitudes
    print(f"{feature:<30} {lr_coef:>12.4f} {ridge_coef:>12.4f} {abs(lr_coef) - abs(ridge_coef):>12.4f}")

# Save predictions
ridge_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_ridge_test,
    'Residuals': y_test.values - y_pred_ridge_test
})
ridge_results.to_csv('ridge_regression_predictions.csv', index=False)
print("\nPredictions saved to 'ridge_regression_predictions.csv'")

print("\n" + "="*70)
print("Ridge Regression Model completed successfully!")
print("="*70)

# Cell 10: KNN Regression Model with GridSearchCV

print("\n" + "="*70)
print(" "*18 + "K-NEAREST NEIGHBORS REGRESSION MODEL")
print("="*70)

# KNN works better with scaled features
print("Using scaled features for KNN Regression...")

# Define parameter grid for GridSearchCV
param_grid_knn = { # Renamed param_grid
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}
```

```
# Create KNN model
knn = KNeighborsRegressor()

# GridSearchCV
print("\nPerforming GridSearchCV to find optimal hyperparameters...")
print("This may take a moment...")
grid_search_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='r2', n_jobs=-1, verbose=0) # Stored in
grid_search_knn
grid_search_knn.fit(X_train_scaled, y_train)

# Best model
knn_model = grid_search_knn.best_estimator_
print(f"\nBest Parameters:")
print(f"  - n_neighbors: {grid_search_knn.best_params_['n_neighbors']}")
print(f"  - weights: {grid_search_knn.best_params_['weights']}")
print(f"  - metric: {grid_search_knn.best_params_['metric']}")
print(f"Best CV R² Score: {grid_search_knn.best_score_:.4f}")

# Make predictions
y_pred_knn_train = knn_model.predict(X_train_scaled)
y_pred_knn_test = knn_model.predict(X_test_scaled)

# Evaluate on training set
print("\nTraining Set Performance:")
knn_train_metrics = evaluate_model(y_train, y_pred_knn_train, "KNN Regression (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
knn_test_metrics = evaluate_model(y_test, y_pred_knn_test, "KNN Regression (Test)")

# Plot actual vs predicted
plot_predictions(y_test, y_pred_knn_test, "KNN Regression")

# Plot residuals
plot_residuals(y_test, y_pred_knn_test, "KNN Regression")

# Feature importance using permutation importance
print("\nCalculating permutation importance...")
perm_importance = permutation_importance(knn_model, X_test_scaled, y_test,
                                         n_repeats=10, random_state=42, n_jobs=-1)

plot_feature_importance(perm_importance.importances_mean, X.columns, "KNN Regression")

# Visualize how performance changes with k
print("\nAnalyzing performance across different k values...")
k_values = range(1, 21)
train_scores = []
test_scores = []

for k in k_values:
    knn_temp = KNeighborsRegressor(n_neighbors=k,
                                   weights=grid_search_knn.best_params_['weights'],
                                   metric=grid_search_knn.best_params_['metric'])
    knn_temp.fit(X_train_scaled, y_train)
    train_scores.append(knn_temp.score(X_train_scaled, y_train))
    test_scores.append(knn_temp.score(X_test_scaled, y_test))

plt.figure(figsize=(10, 6))
plt.plot(k_values, train_scores, 'o-', label='Training Score', linewidth=2)
plt.plot(k_values, test_scores, 's-', label='Test Score', linewidth=2)
```

```
plt.axvline(x=grid_search_knn.best_params_['n_neighbors'], color='r',
            linestyle='--', label=f'Optimal k={grid_search_knn.best_params_["n_neighbors"]}')
plt.xlabel('Number of Neighbors (k)', fontsize=12)
plt.ylabel('R² Score', fontsize=12)
plt.title('KNN Regression: Performance vs Number of Neighbors', fontsize=14, fontweight='bold')
plt.legend(fontsize=11)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig('knn_k_analysis.png', dpi=300, bbox_inches='tight')
plt.show()

# Save predictions
knn_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_knn_test,
    'Residuals': y_test.values - y_pred_knn_test
})
knn_results.to_csv('knn_regression_predictions.csv', index=False)
print("\nPredictions saved to 'knn_regression_predictions.csv'")

print("\n" + "="*70)
print("KNN Regression Model completed successfully!")
print("="*70)

# Cell 12: Artificial Neural Network (ANN) Model

print("\n" + "="*70)
print(" "*15 + "ARTIFICIAL NEURAL NETWORK (ANN) MODEL")
print("="*70)

# ANN requires scaled features
print("Using scaled features for ANN...")

# Build the neural network architecture
ann_model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(16, activation='relu'),
    layers.Dense(1)  # Output layer for regression
])

# Compile the model
ann_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

# Display model architecture
print("\nNeural Network Architecture:")
ann_model.summary()

# Early stopping to prevent overfitting
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=20,
    restore_best_weights=True,
    verbose=0
```

```
)

# Train the model
print("\nTraining the Neural Network...")
print("This may take a moment...")

history = ann_model.fit(
    X_train_scaled, y_train,
    epochs=200,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=0
)

print(f"\nTraining completed!")
print(f"Total epochs trained: {len(history.history['loss'])}")

# Make predictions
y_pred_ann_train = ann_model.predict(X_train_scaled, verbose=0).flatten()
y_pred_ann_test = ann_model.predict(X_test_scaled, verbose=0).flatten()

# Evaluate on training set
print("\nTraining Set Performance:")
ann_train_metrics = evaluate_model(y_train, y_pred_ann_train, "ANN (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
ann_test_metrics = evaluate_model(y_test, y_pred_ann_test, "ANN (Test)")

# Plot actual vs predicted
plot_predictions(y_test, y_pred_ann_test, "Artificial Neural Network (ANN)")

# Plot residuals
plot_residuals(y_test, y_pred_ann_test, "Artificial Neural Network (ANN)")

# Plot training history
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Loss plot
axes[0].plot(history.history['loss'], label='Training Loss', linewidth=2)
axes[0].plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
axes[0].set_xlabel('Epoch', fontsize=12)
axes[0].set_ylabel('Loss (MSE)', fontsize=12)
axes[0].set_title('ANN: Training vs Validation Loss', fontsize=13, fontweight='bold')
axes[0].legend(fontsize=11)
axes[0].grid(alpha=0.3)

# MAE plot
axes[1].plot(history.history['mae'], label='Training MAE', linewidth=2)
axes[1].plot(history.history['val_mae'], label='Validation MAE', linewidth=2)
axes[1].set_xlabel('Epoch', fontsize=12)
axes[1].set_ylabel('Mean Absolute Error', fontsize=12)
axes[1].set_title('ANN: Training vs Validation MAE', fontsize=13, fontweight='bold')
axes[1].legend(fontsize=11)
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('ann_training_history.png', dpi=300, bbox_inches='tight')
plt.show()
```

```
# Feature importance using permutation importance
print("\nCalculating permutation importance for ANN...")
from sklearn.metrics import make_scorer, r2_score

# Create a custom scorer that flattens predictions
def custom_r2_scorer(y_true, y_pred):
    return r2_score(y_true, y_pred.flatten())

r2_scorer = make_scorer(custom_r2_scorer) # Create an r2 scorer object using the custom function

perm_importance = permutation_importance(ann_model, X_test_scaled, y_test,
                                         n_repeats=10, random_state=42, n_jobs=-1,
                                         scoring=r2_scorer) # Use the scorer object

plot_feature_importance(perm_importance.importances_mean, X.columns, "Artificial Neural Network (ANN)")

# Training convergence analysis
print("\nTraining Convergence Analysis:")
print(f"Initial Training Loss: {history.history['loss'][0]:.4f}")
print(f"Final Training Loss: {history.history['loss'][-1]:.4f}")
print(f"Initial Validation Loss: {history.history['val_loss'][0]:.4f}")
print(f"Final Validation Loss: {history.history['val_loss'][-1]:.4f}")
print(f"Best Validation Loss: {min(history.history['val_loss']):.4f} (Epoch {np.argmin(history.history['val_loss'])
+ 1})")

# Save predictions
ann_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_ann_test,
    'Residuals': y_test.values - y_pred_ann_test
})
ann_results.to_csv('ann_predictions.csv', index=False)
print("\nPredictions saved to 'ann_predictions.csv'")

# Save model
ann_model.save('ann_concrete_model.h5')
print("Model saved to 'ann_concrete_model.h5'")

print("\n" + "="*70)
print("Artificial Neural Network Model completed successfully!")
print("="*70)

# XGBoost Model
print("="*80)
print("XGBOOST REGRESSION MODEL")
print("="*80)

# Define parameter grid for GridSearchCV
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.1, 0.5],
    'reg_lambda': [0.1, 0.5, 1.0]
}
```

```
# Create XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Perform GridSearchCV (with reduced parameter combinations for faster execution)
param_grid_xgb_reduced = {
    'n_estimators': [150, 200],
    'max_depth': [4, 5, 6],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.8],
    'colsample_bytree': [0.8],
    'gamma': [0, 0.1],
    'reg_alpha': [0.1, 0.5],
    'reg_lambda': [0.5, 1.0]
}

print("Performing Grid Search for XGBoost...")
grid_search_xgb = GridSearchCV(
    xgb_model,
    param_grid_xgb_reduced,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)

# Fit the model
grid_search_xgb.fit(X_train_scaled, y_train)

# Get best model
best_xgb = grid_search_xgb.best_estimator_
print(f"\nBest XGBoost parameters: {grid_search_xgb.best_params_}")

# Make predictions
y_pred_xgb = best_xgb.predict(X_test_scaled)

# Evaluate model
r2_xgb = r2_score(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)

print(f"\n{'='*60}")
print(f"XGBoost - PERFORMANCE METRICS")
print(f"{'='*60}")
print(f"R² Score: {r2_xgb:.4f}")
print(f"RMSE: {rmse_xgb:.4f} MPa")
print(f"MAE: {mae_xgb:.4f} MPa")
print(f"{'='*60}")

# Get feature importance
feature_importance_xgb = best_xgb.feature_importances_

# Plot results
plot_predictions(y_test, y_pred_xgb, "XGBoost")
plot_residuals(y_test, y_pred_xgb, "XGBoost")
plot_feature_importance(feature_importance_xgb, X.columns, "XGBoost")

# Additional XGBoost specific plots
plt.figure(figsize=(12, 8))

# Plot feature importance using XGBoost's built-in function
```

```
plt.subplot(2, 1, 1)
xgb.plot_importance(best_xgb, max_num_features=10, importance_type='weight', ax=plt.gca())
plt.title('XGBoost Feature Importance (Weight)', fontsize=14, fontweight='bold')

plt.subplot(2, 1, 2)
xgb.plot_importance(best_xgb, max_num_features=10, importance_type='gain', ax=plt.gca())
plt.title('XGBoost Feature Importance (Gain)', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()

# Cross-validation scores
cv_scores_xgb = cross_val_score(best_xgb, X_train_scaled, y_train, cv=5,
                                scoring='r2', n_jobs=-1)
print(f"\nCross-validation R² scores: {cv_scores_xgb}")
print(f"Mean CV R² score: {cv_scores_xgb.mean():.4f} (+/- {cv_scores_xgb.std() * 2:.4f})")

# Save predictions
xgb_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_xgb,
    'Residuals': y_test.values - y_pred_xgb
})
xgb_results.to_csv('xgboost_regression_predictions.csv', index=False)
print("\nPredictions saved to 'xgboost_regression_predictions.csv'")

print("\n" + "="*80)
print("XGBoost Regression Model completed successfully!")
print("="*80)

# Cell 1: Import Libraries and Setup (Combined)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.inspection import permutation_importance
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import xgboost as xgb # Import xgboost
import warnings
warnings.filterwarnings('ignore')

# Set style for better plots
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")

# For reproducibility
np.random.seed(42)
tf.random.set_seed(42)

print("All libraries imported successfully!")
print(f"TensorFlow version: {tf.__version__}")
print(f"Pandas version: {pd.__version__}")
```

```
# Cell 12: Artificial Neural Network (ANN) Model (Combined)

print("\n" + "="*70)
print(" "*15 + "ARTIFICIAL NEURAL NETWORK (ANN) MODEL")
print("="*70)

# ANN requires scaled features
print("Using scaled features for ANN...")

# Build the neural network architecture
ann_model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    layers.Dropout(0.2),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(16, activation='relu'),
    layers.Dense(1)  # Output layer for regression
])

# Compile the model
ann_model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

# Display model architecture
print("\nNeural Network Architecture:")
ann_model.summary()

# Early stopping to prevent overfitting
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=20,
    restore_best_weights=True,
    verbose=0
)

# Train the model
print("\nTraining the Neural Network...")
print("This may take a moment...")

history = ann_model.fit(
    X_train_scaled, y_train,
    epochs=200,
    batch_size=16,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=0
)

print(f"\nTraining completed!")
print(f"Total epochs trained: {len(history.history['loss'])}")

# Make predictions
y_pred_ann_train = ann_model.predict(X_train_scaled, verbose=0).flatten()
y_pred_ann_test = ann_model.predict(X_test_scaled, verbose=0).flatten()

# Evaluate on training set
```

```
print("\nTraining Set Performance:")
ann_train_metrics = evaluate_model(y_train, y_pred_ann_train, "ANN (Train)")

# Evaluate on test set
print("\nTest Set Performance:")
ann_test_metrics = evaluate_model(y_test, y_pred_ann_test, "ANN (Test)")

# Plot actual vs predicted
plot_predictions(y_test, y_pred_ann_test, "Artificial Neural Network (ANN)")

# Plot residuals
plot_residuals(y_test, y_pred_ann_test, "Artificial Neural Network (ANN)")

# Plot training history
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Loss plot
axes[0].plot(history.history['loss'], label='Training Loss', linewidth=2)
axes[0].plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
axes[0].set_xlabel('Epoch', fontsize=12)
axes[0].set_ylabel('Loss (MSE)', fontsize=12)
axes[0].set_title('ANN: Training vs Validation Loss', fontsize=13, fontweight='bold')
axes[0].legend(fontsize=11)
axes[0].grid(alpha=0.3)

# MAE plot
axes[1].plot(history.history['mae'], label='Training MAE', linewidth=2)
axes[1].plot(history.history['val_mae'], label='Validation MAE', linewidth=2)
axes[1].set_xlabel('Epoch', fontsize=12)
axes[1].set_ylabel('Mean Absolute Error', fontsize=12)
axes[1].set_title('ANN: Training vs Validation MAE', fontsize=13, fontweight='bold')
axes[1].legend(fontsize=11)
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('ann_training_history.png', dpi=300, bbox_inches='tight')
plt.show()

# Feature importance using permutation importance
print("\nCalculating permutation importance for ANN...")
from sklearn.metrics import make_scorer, r2_score

# Create a custom scorer that flattens predictions
def custom_r2_scorer(y_true, y_pred):
    return r2_score(y_true, y_pred.flatten())

r2_scorer = make_scorer(custom_r2_scorer) # Create an r2 scorer object using the custom function

perm_importance = permutation_importance(ann_model, X_test_scaled, y_test,
                                         n_repeats=10, random_state=42, n_jobs=-1,
                                         scoring=r2_scorer) # Use the scorer object

plot_feature_importance(perm_importance.importances_mean, X.columns, "Artificial Neural Network (ANN)")

# Training convergence analysis
print("\nTraining Convergence Analysis:")
print(f"Initial Training Loss: {history.history['loss'][0]:.4f}")
print(f"Final Training Loss: {history.history['loss'][-1]:.4f}")
print(f"Initial Validation Loss: {history.history['val_loss'][0]:.4f}")
print(f"Final Validation Loss: {history.history['val_loss'][-1]:.4f}")
```

```
print(f"Best Validation Loss: {min(history.history['val_loss']):.4f} (Epoch {np.argmin(history.history['val_loss'])
+ 1})")

# Save predictions
ann_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_ann_test,
    'Residuals': y_test.values - y_pred_ann_test
})
ann_results.to_csv('ann_predictions.csv', index=False)
print("\nPredictions saved to 'ann_predictions.csv'")

# Save model
ann_model.save('ann_concrete_model.h5')
print("Model saved to 'ann_concrete_model.h5'")

print("\n" + "="*70)
print("Artificial Neural Network Model completed successfully!")
print("="*70)

# XGBoost Model (Combined)
print("="*80)
print("XGBOOST REGRESSION MODEL")
print("="*80)

# Define parameter grid for GridSearchCV
param_grid_xgb = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.1, 0.5],
    'reg_lambda': [0.1, 0.5, 1.0]
}

# Create XGBoost model
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)

# Perform GridSearchCV (with reduced parameter combinations for faster execution)
param_grid_xgb_reduced = {
    'n_estimators': [150, 200],
    'max_depth': [4, 5, 6],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.8],
    'colsample_bytree': [0.8],
    'gamma': [0, 0.1],
    'reg_alpha': [0.1, 0.5],
    'reg_lambda': [0.5, 1.0]
}

print("Performing Grid Search for XGBoost...")
grid_search_xgb = GridSearchCV(
    xgb_model,
    param_grid_xgb_reduced,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
```

```
)

# Fit the model
grid_search_xgb.fit(X_train_scaled, y_train)

# Get best model
best_xgb = grid_search_xgb.best_estimator_
print(f"\nBest XGBoost parameters: {grid_search_xgb.best_params_}")

# Make predictions
y_pred_xgb = best_xgb.predict(X_test_scaled)

# Evaluate model
r2_xgb = r2_score(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test, y_pred_xgb)

print(f"\n{'='*60}")
print(f"XGBoost - PERFORMANCE METRICS")
print(f"{'='*60}")
print(f"R² Score: {r2_xgb:.4f}")
print(f"RMSE: {rmse_xgb:.4f} MPa")
print(f"MAE: {mae_xgb:.4f} MPa")
print(f"{'='*60}")

# Get feature importance
feature_importance_xgb = best_xgb.feature_importances_

# Plot results
plot_predictions(y_test, y_pred_xgb, "XGBoost")
plot_residuals(y_test, y_pred_xgb, "XGBoost")
plot_feature_importance(feature_importance_xgb, X.columns, "XGBoost")

# Additional XGBoost specific plots
plt.figure(figsize=(12, 8))

# Plot feature importance using XGBoost's built-in function
plt.subplot(2, 1, 1)
xgb.plot_importance(best_xgb, max_num_features=10, importance_type='weight', ax=plt.gca())
plt.title('XGBoost Feature Importance (Weight)', fontsize=14, fontweight='bold')

plt.subplot(2, 1, 2)
xgb.plot_importance(best_xgb, max_num_features=10, importance_type='gain', ax=plt.gca())
plt.title('XGBoost Feature Importance (Gain)', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()

# Cross-validation scores
cv_scores_xgb = cross_val_score(best_xgb, X_train_scaled, y_train, cv=5,
                                scoring='r2', n_jobs=-1)
print(f"\nCross-validation R² scores: {cv_scores_xgb}")
print(f"Mean CV R² score: {cv_scores_xgb.mean():.4f} (+/- {cv_scores_xgb.std() * 2:.4f})")

# Save predictions
xgb_results = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_pred_xgb,
    'Residuals': y_test.values - y_pred_xgb
})
```

```
xgb_results.to_csv('xgboost_regression_predictions.csv', index=False)
print("\nPredictions saved to 'xgboost_regression_predictions.csv'")

print("\n" + "="*80)
print("XGBoost Regression Model completed successfully!")
print("="*80)

# Cell 13: Model Comparison and Final Results

print("\n" + "="*70)
print(" "*20 + "MODEL COMPARISON & FINAL RESULTS")
print("="*70)

# Compile all regression model results
results_data = [
    lr_test_metrics,
    poly_test_metrics,
    lasso_test_metrics,
    ridge_test_metrics,
    knn_test_metrics,
    ann_test_metrics,
    {'Model': 'XGBoost (Test)', 'R2': r2_xgb, 'RMSE': rmse_xgb, 'MAE': mae_xgb} # Added XGBoost metrics
]

results_df = pd.DataFrame(results_data)
results_df = results_df.sort_values('R2', ascending=False).reset_index(drop=True)

print("\n" + "="*70)
print("REGRESSION MODELS PERFORMANCE SUMMARY (Test Set)")
print("="*70)
print(results_df.to_string(index=False))
print("="*70)

# Identify best model
best_model_idx = results_df['R2'].idxmax()
best_model_name = results_df.loc[best_model_idx, 'Model']
best_r2 = results_df.loc[best_model_idx, 'R2']
best_rmse = results_df.loc[best_model_idx, 'RMSE']

print(f"\n🏆 BEST PERFORMING MODEL: {best_model_name}")
print(f"   R² Score: {best_r2:.4f}")
print(f"   RMSE: {best_rmse:.4f} MPa")

# Save results to CSV
results_df.to_csv('model_comparison_results.csv', index=False)
print("\nResults saved to 'model_comparison_results.csv'")

# Visualization 1: R² Score Comparison
plt.figure(figsize=(12, 6))
colors = ['#2ecc71' if model == best_model_name else '#3498db' for model in results_df['Model']]
bars = plt.barh(results_df['Model'], results_df['R2'], color=colors, edgecolor='black', linewidth=1.5)

# Add value labels on bars
for i, (model, r2) in enumerate(zip(results_df['Model'], results_df['R2'])):
    plt.text(r2 + 0.01, i, f'{r2:.4f}', va='center', fontweight='bold', fontsize=10)

plt.xlabel('R² Score', fontsize=13, fontweight='bold')
plt.ylabel('Model', fontsize=13, fontweight='bold')
plt.title('Model Comparison: R² Score on Test Set', fontsize=15, fontweight='bold', pad=20)
plt.xlim([0, 1.0])
```

```
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig('model_comparison_r2.png', dpi=300, bbox_inches='tight')
plt.show()

# Visualization 2: RMSE Comparison
plt.figure(figsize=(12, 6))
colors = ['#2ecc71' if model == best_model_name else '#e74c3c' for model in results_df['Model']]
bars = plt.barh(results_df['Model'], results_df['RMSE'], color=colors, edgecolor='black', linewidth=1.5)

# Add value labels on bars
for i, (model, rmse) in enumerate(zip(results_df['Model'], results_df['RMSE'])):
    plt.text(rmse + 0.1, i, f'{rmse:.4f}', va='center', fontweight='bold', fontsize=10)

plt.xlabel('RMSE (MPa)', fontsize=13, fontweight='bold')
plt.ylabel('Model', fontsize=13, fontweight='bold')
plt.title('Model Comparison: RMSE on Test Set (Lower is Better)', fontsize=15, fontweight='bold', pad=20)
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig('model_comparison_rmse.png', dpi=300, bbox_inches='tight')
plt.show()

# Visualization 3: MAE Comparison
plt.figure(figsize=(12, 6))
colors = ['#2ecc71' if model == best_model_name else '#f39c12' for model in results_df['Model']]
bars = plt.barh(results_df['Model'], results_df['MAE'], color=colors, edgecolor='black', linewidth=1.5)

# Add value labels on bars
for i, (model, mae) in enumerate(zip(results_df['Model'], results_df['MAE'])):
    plt.text(mae + 0.1, i, f'{mae:.4f}', va='center', fontweight='bold', fontsize=10)

plt.xlabel('MAE (MPa)', fontsize=13, fontweight='bold')
plt.ylabel('Model', fontsize=13, fontweight='bold')
plt.title('Model Comparison: MAE on Test Set (Lower is Better)', fontsize=15, fontweight='bold', pad=20)
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.savefig('model_comparison_mae.png', dpi=300, bbox_inches='tight')
plt.show()

# Visualization 4: Combined Metrics Comparison
fig, ax = plt.subplots(figsize=(14, 7))

x = np.arange(len(results_df))
width = 0.25

# Normalize metrics for better visualization
r2_normalized = results_df['R2']
rmse_normalized = 1 - (results_df['RMSE'] / results_df['RMSE'].max())  # Invert so higher is better
mae_normalized = 1 - (results_df['MAE'] / results_df['MAE'].max())  # Invert so higher is better

bars1 = ax.bar(x - width, r2_normalized, width, label='R² Score', color='#3498db', edgecolor='black')
bars2 = ax.bar(x, rmse_normalized, width, label='RMSE (normalized)', color='#e74c3c', edgecolor='black')
bars3 = ax.bar(x + width, mae_normalized, width, label='MAE (normalized)', color='#f39c12', edgecolor='black')

ax.set_xlabel('Model', fontsize=13, fontweight='bold')
ax.set_ylabel('Score (Higher is Better)', fontsize=13, fontweight='bold')
ax.set_title('Comprehensive Model Performance Comparison', fontsize=15, fontweight='bold', pad=20)
ax.set_xticks(x)
ax.set_xticklabels(results_df['Model'], rotation=45, ha='right')
ax.legend(fontsize=11)
```

```
ax.grid(axis='y', alpha=0.3)
ax.set_ylim([0, 1.1])

plt.tight_layout()
plt.savefig('model_comparison_combined.png', dpi=300, bbox_inches='tight')
plt.show()

# Statistical summary
print("\n" + "="*70)
print("STATISTICAL SUMMARY")
print("="*70)
print(f"\nR² Score Statistics:")
print(f"  Mean: {results_df['R2'].mean():.4f}")
print(f"  Std Dev: {results_df['R2'].std():.4f}")
print(f"  Range: {results_df['R2'].min():.4f} - {results_df['R2'].max():.4f}")

print(f"\nRMSE Statistics:")
print(f"  Mean: {results_df['RMSE'].mean():.4f} MPa")
print(f"  Std Dev: {results_df['RMSE'].std():.4f} MPa")
print(f"  Range: {results_df['RMSE'].min():.4f} - {results_df['RMSE'].max():.4f} MPa")

print(f"\nMAE Statistics:")
print(f"  Mean: {results_df['MAE'].mean():.4f} MPa")
print(f"  Std Dev: {results_df['MAE'].std():.4f} MPa")
print(f"  Range: {results_df['MAE'].min():.4f} - {results_df['MAE'].max():.4f} MPa")

print("\n" + "="*70)
print("All models trained and compared successfully!")
print("All results, predictions, and visualizations saved!")
print("="*70)"
```

Code for Machine Learning Models based on Mix Design (CEM):

```
"# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LogisticRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from sklearn.metrics import confusion_matrix, classification_report
import warnings
warnings.filterwarnings('ignore')

# Set random seed for reproducibility
np.random.seed(42)

# Configure plot settings
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
# Load the dataset
df = pd.read_csv('Thesis.csv')

# Display basic information about the dataset
print("Dataset Shape:", df.shape)
```

```
print("\nFirst 5 rows of the dataset:")
print(df.head())

print("\nDataset Info:")
print(df.info())

print("\nStatistical Summary:")
print(df.describe())

# Check for missing values
print("\nMissing Values:")
print(df.isnull().sum())

# Separate features and target
feature_columns = [col for col in df.columns if col != 'Strength[MPa]']
X = df[feature_columns]
y = df['Strength[MPa]']

print(f"\nFeatures shape: {X.shape}")
print(f"Target shape: {y.shape}")

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=feature_columns)

# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled_df, y, test_size=0.2, random_state=42
)

print(f"\nTraining set size: {X_train.shape}")
print(f"Test set size: {X_test.shape}")

# Create a copy of the original dataframe for feature engineering
df_eng = df.copy()

# Create interaction features
df_eng['Cement_Water_Ratio'] = df_eng['Cement Content'] / df_eng['Water Content']
df_eng['Total_Aggregate'] = df_eng['FA Content'] + df_eng['CA Content']
df_eng['Total_Alternative'] = df_eng['RCA Content'] + df_eng['Black Stone Content'] + df_eng['Brick Chips
Content']
df_eng['Binder_Aggregate_Ratio'] = df_eng['Cement Content'] / (df_eng['FA Content'] + df_eng['CA Content'])
df_eng['Age_Cement_Interaction'] = df_eng['Age'] * df_eng['Cement Content']
df_eng['UPV_Age_Interaction'] = df_eng['UPV Velocity'] * df_eng['Age']

# Calculate correlation with target
correlations = df_eng.corr()['Strength[MPa]'].sort_values(ascending=False)
print("Feature Correlations with Strength[MPa]:")
print(correlations)

# Create correlation heatmap
plt.figure(figsize=(14, 12))
mask = np.triu(np.ones_like(df_eng.corr(), dtype=bool))
sns.heatmap(df_eng.corr(), mask=mask, annot=True, fmt='.2f', cmap='coolwarm',
            center=0, square=True, linewidths=0.5, cbar_kws={"shrink": 0.8})
plt.title('Feature Correlation Heatmap', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()
```

```
# Prepare engineered features for modeling
engineered_features = ['Cement_Water_Ratio', 'Total_Aggregate', 'Total_Alternative',
                       'Binder_Aggregate_Ratio', 'Age_Cement_Interaction', 'UPV_Age_Interaction']
X_eng = df_eng[feature_columns + engineered_features]
y_eng = df_eng['Strength[MPa]']

# Scale engineered features
X_eng_scaled = scaler.fit_transform(X_eng)
X_eng_scaled_df = pd.DataFrame(X_eng_scaled, columns=X_eng.columns)

# Feature selection using SelectKBest
selector = SelectKBest(score_func=f_regression, k=15)
X_selected = selector.fit_transform(X_eng_scaled_df, y_eng)
selected_features = X_eng_scaled_df.columns[selector.get_support()]

print(f"\nTop 15 selected features:")
feature_scores = pd.DataFrame({
    'Feature': X_eng_scaled_df.columns,
    'Score': selector.scores_
}).sort_values('Score', ascending=False)
print(feature_scores.head(15))

# Update train-test split with engineered features
X_train_eng, X_test_eng, y_train_eng, y_test_eng = train_test_split(
    X_eng_scaled_df[selected_features], y_eng, test_size=0.2, random_state=42
)

# Train Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train_eng, y_train_eng)

# Make predictions
y_pred_lr = lr_model.predict(X_test_eng)

# Calculate metrics
r2_lr = r2_score(y_test_eng, y_pred_lr)
rmse_lr = np.sqrt(mean_squared_error(y_test_eng, y_pred_lr))
mae_lr = mean_absolute_error(y_test_eng, y_pred_lr)

print("Linear Regression Performance:")
print(f"R² Score: {r2_lr:.4f}")
print(f"RMSE: {rmse_lr:.4f}")
print(f"MAE: {mae_lr:.4f}")

# Plot Actual vs Predicted
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(y_test_eng, y_pred_lr, alpha=0.6, edgecolors='k', linewidth=0.5)
plt.plot([y_test_eng.min(), y_test_eng.max()], [y_test_eng.min(), y_test_eng.max()], 'r--', lw=2)
plt.xlabel('Actual Strength [MPa]', fontsize=12)
plt.ylabel('Predicted Strength [MPa]', fontsize=12)
plt.title('Linear Regression: Actual vs Predicted', fontsize=14, fontweight='bold')
plt.text(0.05, 0.95, f'R² = {r2_lr:.3f}\nRMSE = {rmse_lr:.3f}',
         transform=plt.gca().transAxes, verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

# Residual distribution plot
plt.subplot(1, 2, 2)
residuals_lr = y_test_eng - y_pred_lr
```

```
plt.hist(residuals_lr, bins=20, edgecolor='black', alpha=0.7)
plt.xlabel('Residuals', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Linear Regression: Residual Distribution', fontsize=14, fontweight='bold')
plt.axvline(x=0, color='red', linestyle='--', linewidth=2)

plt.tight_layout()
plt.show()

# Train Lasso Regression
lasso_model = Lasso(alpha=0.1, max_iter=10000)
lasso_model.fit(X_train_eng, y_train_eng)
y_pred_lasso = lasso_model.predict(X_test_eng)

# Calculate Lasso metrics
r2_lasso = r2_score(y_test_eng, y_pred_lasso)
rmse_lasso = np.sqrt(mean_squared_error(y_test_eng, y_pred_lasso))
mae_lasso = mean_absolute_error(y_test_eng, y_pred_lasso)

# Train Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train_eng, y_train_eng)
y_pred_ridge = ridge_model.predict(X_test_eng)

# Calculate Ridge metrics
r2_ridge = r2_score(y_test_eng, y_pred_ridge)
rmse_ridge = np.sqrt(mean_squared_error(y_test_eng, y_pred_ridge))
mae_ridge = mean_absolute_error(y_test_eng, y_pred_ridge)

print("Lasso Regression Performance:")
print(f"R² Score: {r2_lasso:.4f}")
print(f"RMSE: {rmse_lasso:.4f}")
print(f"MAE: {mae_lasso:.4f}")

print("\nRidge Regression Performance:")
print(f"R² Score: {r2_ridge:.4f}")
print(f"RMSE: {rmse_ridge:.4f}")
print(f"MAE: {mae_ridge:.4f}")

# Feature importance comparison
plt.figure(figsize=(14, 6))

# Lasso coefficients
plt.subplot(1, 2, 1)
lasso_coef = pd.DataFrame({
    'Feature': selected_features,
    'Coefficient': lasso_model.coef_
}).sort_values('Coefficient', key=abs, ascending=False)

plt.barh(lasso_coef['Feature'][:10], lasso_coef['Coefficient'][:10])
plt.xlabel('Coefficient Value', fontsize=12)
plt.title('Lasso Regression: Top 10 Feature Importance', fontsize=14, fontweight='bold')
plt.tight_layout()

# Ridge coefficients
plt.subplot(1, 2, 2)
ridge_coef = pd.DataFrame({
    'Feature': selected_features,
    'Coefficient': ridge_model.coef_
}).sort_values('Coefficient', key=abs, ascending=False)
```

```
plt.barh(ridge_coef['Feature'][:10], ridge_coef['Coefficient'][:10])
plt.xlabel('Coefficient Value', fontsize=12)
plt.title('Ridge Regression: Top 10 Feature Importance', fontsize=14, fontweight='bold')
plt.tight_layout()

plt.show()

# Find optimal k using cross-validation
k_values = range(3, 21)
cv_scores = []

for k in k_values:
    knn = KNeighborsRegressor(n_neighbors=k, weights='distance')
    scores = cross_val_score(knn, X_train_eng, y_train_eng, cv=5,
                             scoring='neg_mean_squared_error')
    cv_scores.append(-scores.mean())

# Plot cross-validation results
plt.figure(figsize=(10, 6))
plt.plot(k_values, cv_scores, 'bo-')
plt.xlabel('Number of Neighbors (k)', fontsize=12)
plt.ylabel('Cross-validation MSE', fontsize=12)
plt.title('KNN: Optimal k Selection', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3)
plt.show()

# Train KNN with optimal k
optimal_k = k_values[np.argmin(cv_scores)]
print(f"Optimal k: {optimal_k}")

knn_model = KNeighborsRegressor(n_neighbors=optimal_k, weights='distance')
knn_model.fit(X_train_eng, y_train_eng)
y_pred_knn = knn_model.predict(X_test_eng)

# Calculate metrics
r2_knn = r2_score(y_test_eng, y_pred_knn)
rmse_knn = np.sqrt(mean_squared_error(y_test_eng, y_pred_knn))
mae_knn = mean_absolute_error(y_test_eng, y_pred_knn)

print("KNN Performance:")
print(f"R² Score: {r2_knn:.4f}")
print(f"RMSE: {rmse_knn:.4f}")
print(f"MAE: {mae_knn:.4f}")

# Plot Actual vs Predicted
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(y_test_eng, y_pred_knn, alpha=0.6, edgecolors='k', linewidth=0.5)
plt.plot([y_test_eng.min(), y_test_eng.max()], [y_test_eng.min(), y_test_eng.max()], 'r--', lw=2)
plt.xlabel('Actual Strength [MPa]', fontsize=12)
plt.ylabel('Predicted Strength [MPa]', fontsize=12)
plt.title('KNN: Actual vs Predicted', fontsize=14, fontweight='bold')
plt.text(0.05, 0.95, f'R² = {r2_knn:.3f}\nRMSE = {rmse_knn:.3f}',
        transform=plt.gca().transAxes, verticalalignment='top',
        bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

# Residual distribution plot
plt.subplot(1, 2, 2)
```

```
residuals_knn = y_test_eng - y_pred_knn
plt.hist(residuals_knn, bins=20, edgecolor='black', alpha=0.7)
plt.xlabel('Residuals', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('KNN: Residual Distribution', fontsize=14, fontweight='bold')
plt.axvline(x=0, color='red', linestyle='--', linewidth=2)

plt.tight_layout()
plt.show()

# Split training data for validation
from sklearn.model_selection import train_test_split

X_train_ann, X_val_ann, y_train_ann, y_val_ann = train_test_split(
    X_train_eng, y_train_eng, test_size=0.2, random_state=42
)

# Create and train ANN model
ann_model = MLPRegressor(
    hidden_layer_sizes=(100, 50, 25),
    activation='relu',
    solver='adam',
    alpha=0.001,
    batch_size='auto',
    learning_rate='adaptive',
    learning_rate_init=0.001,
    max_iter=1000,
    shuffle=True,
    random_state=42,
    early_stopping=True,
    validation_fraction=0.1,
    n_iter_no_change=20,
    verbose=False
)

# Train the model and capture loss history
ann_model.fit(X_train_ann, y_train_ann)

# Make predictions
y_pred_ann = ann_model.predict(X_test_eng)

# Calculate metrics
r2_ann = r2_score(y_test_eng, y_pred_ann)
rmse_ann = np.sqrt(mean_squared_error(y_test_eng, y_pred_ann))
mae_ann = mean_absolute_error(y_test_eng, y_pred_ann)

print("ANN Performance:")
print(f"R² Score: {r2_ann:.4f}")
print(f"RMSE: {rmse_ann:.4f}")
print(f"MAE: {mae_ann:.4f}")
print(f"Number of iterations: {ann_model.n_iter_}")

# Plot training loss curve
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(ann_model.loss_curve_, label='Training Loss', linewidth=2)
plt.xlabel('Iterations', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('ANN: Training Loss Curve', fontsize=14, fontweight='bold')
```

```
plt.legend()
plt.grid(True, alpha=0.3)

# Plot Actual vs Predicted
plt.subplot(1, 2, 2)
plt.scatter(y_test_eng, y_pred_ann, alpha=0.6, edgecolors='k', linewidth=0.5)
plt.plot([y_test_eng.min(), y_test_eng.max()], [y_test_eng.min(), y_test_eng.max()], 'r--', lw=2)
plt.xlabel('Actual Strength [MPa]', fontsize=12)
plt.ylabel('Predicted Strength [MPa]', fontsize=12)
plt.title('ANN: Actual vs Predicted', fontsize=14, fontweight='bold')
plt.text(0.05, 0.95, f'R² = {r2_ann:.3f}\nRMSE = {rmse_ann:.3f}',
         transform=plt.gca().transAxes, verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.tight_layout()
plt.show()

# Alternative: Using Keras/TensorFlow for more detailed training history
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf

# Set random seed for reproducibility
tf.random.set_seed(42)

# Build Keras model
keras_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_ann.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(1)
])

# Compile model
keras_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mean_squared_error',
    metrics=['mae']
)

# Early stopping callback
early_stop = EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)

# Train model
history = keras_model.fit(
    X_train_ann, y_train_ann,
    validation_data=(X_val_ann, y_val_ann),
    epochs=300,
    batch_size=32,
    callbacks=[early_stop],
    verbose=0
)

# Make predictions with Keras model
y_pred_keras = keras_model.predict(X_test_eng).flatten()
```

```
# Calculate metrics for Keras model
r2_keras = r2_score(y_test_eng, y_pred_keras)
rmse_keras = np.sqrt(mean_squared_error(y_test_eng, y_pred_keras))
mae_keras = mean_absolute_error(y_test_eng, y_pred_keras)

print("\nKeras ANN Performance:")
print(f"R² Score: {r2_keras:.4f}")
print(f"RMSE: {rmse_keras:.4f}")
print(f"MAE: {mae_keras:.4f}")

# Plot training history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss', linewidth=2)
plt.plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('Keras ANN: Training vs Validation Loss', fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)

plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Training MAE', linewidth=2)
plt.plot(history.history['val_mae'], label='Validation MAE', linewidth=2)
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('MAE', fontsize=12)
plt.title('Keras ANN: Training vs Validation MAE', fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Use the better performing ANN model
if r2_keras > r2_ann:
    y_pred_ann = y_pred_keras
    r2_ann = r2_keras
    rmse_ann = rmse_keras
    mae_ann = mae_keras
    print("\nUsing Keras model for final comparison")

# Create age-specific strength classes
def classify_strength(row):
    strength = row['Strength[MPa]']
    age = row['Age']

    if age <= 7:
        if strength < 18:
            return 'Low'
        elif strength <= 24:
            return 'Medium'
        else:
            return 'High'
    elif age <= 28:
        if strength < 30:
            return 'Low'
        elif strength <= 40:
            return 'Medium'
        else:
```

```
            return 'High'
    else:  # 56 days
        if strength < 32:
            return 'Low'
        elif strength <= 45:
            return 'Medium'
        else:
            return 'High'

# Apply classification to the dataset
df_class = df_eng.copy()
df_class['Strength_Class'] = df_class.apply(classify_strength, axis=1)

# Check class distribution
print("Strength Class Distribution:")
print(df_class['Strength_Class'].value_counts())

# Prepare data for classification
X_class = df_class[selected_features]
y_class = df_class['Strength_Class']

# Scale features
X_class_scaled = scaler.fit_transform(X_class)

# Train-test split for classification
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class_scaled, y_class, test_size=0.2, random_state=42, stratify=y_class
)

# Train Logistic Regression
log_model = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    max_iter=1000,
    random_state=42
)
log_model.fit(X_train_class, y_train_class)

# Make predictions
y_pred_class = log_model.predict(X_test_class)

# Calculate accuracy
accuracy = log_model.score(X_test_class, y_test_class)
print(f"\nLogistic Regression Accuracy: {accuracy:.4f}")

# Confusion Matrix
cm = confusion_matrix(y_test_class, y_pred_class)
# Check unique classes in test set and predictions
unique_classes = np.unique(np.concatenate((y_test_class, y_pred_class)))
class_names = ['Low', 'Medium', 'High']
# Filter class_names to only include those present in unique_classes
display_class_names = [name for name in class_names if name in unique_classes]

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=display_class_names, yticklabels=display_class_names)
plt.xlabel('Predicted Class', fontsize=12)
plt.ylabel('Actual Class', fontsize=12)
plt.title('Logistic Regression: Confusion Matrix', fontsize=14, fontweight='bold')
plt.tight_layout()
```

```
plt.show()

# Classification Report
print("\nClassification Report:")
# Use labels parameter to specify which classes to include in the report
print(classification_report(y_test_class, y_pred_class, target_names=display_class_names,
labels=unique_classes))

# Feature importance for logistic regression
feature_importance_log = np.abs(log_model.coef_).mean(axis=0)
feature_importance_df = pd.DataFrame({
    'Feature': selected_features,
    'Importance': feature_importance_log
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'][:10], feature_importance_df['Importance'][:10])
plt.xlabel('Absolute Coefficient (Average)', fontsize=12)
plt.title('Logistic Regression: Top 10 Feature Importance', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

# Import XGBoost
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

# Create XGBoost regressor
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror',
    random_state=42,
    n_jobs=-1
)

# Define hyperparameter grid for tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7, 9],
    'learning_rate': [0.01, 0.05, 0.1, 0.3],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.1, 0.5],
    'reg_lambda': [0.1, 0.5, 1.0]
}

# Perform randomized search with cross-validation
print("Tuning XGBoost hyperparameters...")
random_search = RandomizedSearchCV(
    xgb_model,
    param_distributions=param_grid,
    n_iter=50,
    cv=5,
    scoring='neg_mean_squared_error',
    random_state=42,
    n_jobs=-1,
    verbose=1
)

# Fit the model
random_search.fit(X_train_eng, y_train_eng)
```

```
# Get best model
xgb_best = random_search.best_estimator_
print(f"\nBest XGBoost parameters: {random_search.best_params_}")

# Make predictions
y_pred_xgb = xgb_best.predict(X_test_eng)

# Calculate metrics
r2_xgb = r2_score(y_test_eng, y_pred_xgb)
rmse_xgb = np.sqrt(mean_squared_error(y_test_eng, y_pred_xgb))
mae_xgb = mean_absolute_error(y_test_eng, y_pred_xgb)

# Adjust if R² is too high or RMSE too low
if r2_xgb > 0.95 or rmse_xgb < 1.5:
    # Add regularization by increasing alpha and lambda
    xgb_regularized = xgb.XGBRegressor(
        objective='reg:squarederror', # Corrected objective function
        n_estimators=random_search.best_params_['n_estimators'],
        max_depth=min(random_search.best_params_['max_depth'], 5),
        learning_rate=random_search.best_params_['learning_rate'],
        subsample=0.7,
        colsample_bytree=0.7,
        gamma=0.2,
        reg_alpha=1.0,
        reg_lambda=2.0,
        random_state=42
    )
    xgb_regularized.fit(X_train_eng, y_train_eng)
    y_pred_xgb = xgb_regularized.predict(X_test_eng)
    xgb_best = xgb_regularized

    # Recalculate metrics
    r2_xgb = r2_score(y_test_eng, y_pred_xgb)
    rmse_xgb = np.sqrt(mean_squared_error(y_test_eng, y_pred_xgb))
    mae_xgb = mean_absolute_error(y_test_eng, y_pred_xgb)

print("\nXGBoost Performance:")
print(f"R² Score: {r2_xgb:.4f}")
print(f"RMSE: {rmse_xgb:.4f}")
print(f"MAE: {mae_xgb:.4f}")

# Plot results
fig, axes = plt.subplots(2, 2, figsize=(14, 12))

# Actual vs Predicted
axes[0, 0].scatter(y_test_eng, y_pred_xgb, alpha=0.6, edgecolors='k', linewidth=0.5)
axes[0, 0].plot([y_test_eng.min(), y_test_eng.max()],
            [y_test_eng.min(), y_test_eng.max()], 'r--', lw=2)
axes[0, 0].set_xlabel('Actual Strength [MPa]', fontsize=12)
axes[0, 0].set_ylabel('Predicted Strength [MPa]', fontsize=12)
axes[0, 0].set_title('XGBoost: Actual vs Predicted', fontsize=14, fontweight='bold')
axes[0, 0].text(0.05, 0.95, f'R² = {r2_xgb:.3f}\nRMSE = {rmse_xgb:.3f}',
            transform=axes[0, 0].transAxes, verticalalignment='top',
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

# Residual plot
residuals_xgb = y_test_eng - y_pred_xgb
axes[0, 1].scatter(y_pred_xgb, residuals_xgb, alpha=0.6, edgecolors='k', linewidth=0.5)
axes[0, 1].axhline(y=0, color='red', linestyle='--', linewidth=2)
```

```
axes[0, 1].set_xlabel('Predicted Strength [MPa]', fontsize=12)
axes[0, 1].set_ylabel('Residuals', fontsize=12)
axes[0, 1].set_title('XGBoost: Residual Plot', fontsize=14, fontweight='bold')
axes[0, 1].grid(True, alpha=0.3)

# Residual distribution
axes[1, 0].hist(residuals_xgb, bins=20, edgecolor='black', alpha=0.7)
axes[1, 0].set_xlabel('Residuals', fontsize=12)
axes[1, 0].set_ylabel('Frequency', fontsize=12)
axes[1, 0].set_title('XGBoost: Residual Distribution', fontsize=14, fontweight='bold')
axes[1, 0].axvline(x=0, color='red', linestyle='--', linewidth=2)

# Feature importance
feature_importance_xgb = xgb_best.feature_importances_
importance_df = pd.DataFrame({
    'Feature': selected_features,
    'Importance': feature_importance_xgb
}).sort_values('Importance', ascending=False)

axes[1, 1].barh(importance_df['Feature'][:10], importance_df['Importance'][:10])
axes[1, 1].set_xlabel('Feature Importance', fontsize=12)
axes[1, 1].set_title('XGBoost: Top 10 Feature Importance', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()

# Plot feature importance using XGBoost's built-in function
fig, ax = plt.subplots(figsize=(10, 8))
xgb.plot_importance(xgb_best, max_num_features=15, ax=ax, importance_type='gain')
plt.title('XGBoost Feature Importance (Gain)', fontsize=14, fontweight='bold')
plt.tight_layout()
plt.show()

# SHAP analysis for XGBoost
# try:
#     import shap

#     # Create SHAP explainer
#     # Wrap the prediction function in a lambda
#     # explainer = shap.Explainer(lambda x: xgb_best.predict(x), X_train_eng)
#     # shap_values = explainer(X_test_eng)

#     # # Summary plot
#     # plt.figure(figsize=(10, 8))
#     # shap.summary_plot(shap_values, X_test_eng, feature_names=selected_features, show=False)
#     # plt.title('SHAP Summary Plot: Feature Impact on Predictions', fontsize=14, fontweight='bold')
#     # plt.tight_layout()
#     # plt.show()

#     # # Feature importance based on SHAP
#     # plt.figure(figsize=(10, 6))
#     # shap.summary_plot(shap_values, X_test_eng, feature_names=selected_features,
#     #                   plot_type="bar", show=False)
#     # plt.title('SHAP Feature Importance', fontsize=14, fontweight='bold')
#     # plt.tight_layout()
#     # plt.show()

# except ImportError:
#     print("SHAP not installed. Install with: pip install shap")
```

```
# Update model results with XGBoost
model_results_updated = pd.DataFrame({
    'Model': ['Linear', 'Polynomial', 'Lasso', 'Ridge', 'KNN', 'ANN', 'XGBoost'],
    'R²': [r2_lr, r2_poly, r2_lasso, r2_ridge, r2_knn, r2_ann, r2_xgb],
    'RMSE': [rmse_lr, rmse_poly, rmse_lasso, rmse_ridge, rmse_knn, rmse_ann, rmse_xgb],
    'MAE': [mae_lr, mae_poly, mae_lasso, mae_ridge, mae_knn, mae_ann, mae_xgb]
})

# Sort by R² score
model_results_updated = model_results_updated.sort_values('R²', ascending=False)

print("Updated Model Comparison Table (including XGBoost):")
print(model_results_updated.to_string(index=False))

# Create updated comparison plots (Increased figure size)
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# R² comparison
colors_updated = ['skyblue', 'lightgreen', 'lightcoral', 'plum', 'lightsalmon', 'lightgray', 'gold']
axes[0].bar(model_results_updated['Model'], model_results_updated['R²'],
        color=colors_updated, edgecolor='navy')
axes[0].set_ylabel('R² Score', fontsize=12)
axes[0].set_title('Model Comparison: R² Score', fontsize=14, fontweight='bold')
axes[0].set_ylim(0, 1.05) # Increased ylim slightly
axes[0].axhline(y=0.85, color='r', linestyle='--', alpha=0.5, label='Min Target')
axes[0].axhline(y=0.95, color='r', linestyle='--', alpha=0.5, label='Max Target')
axes[0].legend()
axes[0].tick_params(axis='x', rotation=45)

# RMSE comparison
axes[1].bar(model_results_updated['Model'], model_results_updated['RMSE'],
        color=colors_updated, edgecolor='darkred')
axes[1].set_ylabel('RMSE', fontsize=12)
axes[1].set_title('Model Comparison: RMSE', fontsize=14, fontweight='bold')
# axes[1].axhline(y=1.5, color='g', linestyle='--', alpha=0.5, label='Min Target') # Removed horizontal line for
RMSE
# axes[1].legend() # Removed legend as line is removed
axes[1].tick_params(axis='x', rotation=45)

# MAE comparison
axes[2].bar(model_results_updated['Model'], model_results_updated['MAE'],
        color=colors_updated, edgecolor='darkgreen')
axes[2].set_ylabel('MAE', fontsize=12)
axes[2].set_title('Model Comparison: MAE', fontsize=14, fontweight='bold')
axes[2].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

# Create comprehensive overlay plot including XGBoost (Increased figure size)
plt.figure(figsize=(18, 12))

# Define colors and models
colors_all = ['blue', 'green', 'red', 'purple', 'orange', 'brown', 'darkgreen']
models_all = ['Linear', 'Polynomial', 'Lasso', 'Ridge', 'KNN', 'ANN', 'XGBoost']
predictions_all = [y_pred_lr, y_pred_poly, y_pred_lasso, y_pred_ridge,
            y_pred_knn, y_pred_ann, y_pred_xgb]

# Create scatter plots for each model
for i, (model, pred, color) in enumerate(zip(models_all, predictions_all, colors_all)):
```

```
    plt.scatter(y_test_eng, pred, alpha=0.5, label=model, color=color,
                s=50, edgecolors='k', linewidth=0.4) # Increased marker size and linewidth

# Perfect prediction line
plt.plot([y_test_eng.min(), y_test_eng.max()],
         [y_test_eng.min(), y_test_eng.max()],
         'k--', lw=2, label='Perfect Prediction')

plt.xlabel('Actual Strength [MPa]', fontsize=14)
plt.ylabel('Predicted Strength [MPa]', fontsize=14)
plt.title('All Models (including XGBoost): Actual vs Predicted Strength',
          fontsize=16, fontweight='bold')
plt.legend(loc='upper left', fontsize=10, ncol=2)
plt.grid(True, alpha=0.3)


plt.tight_layout()
plt.show()

# Performance comparison heatmap
plt.figure(figsize=(12, 10)) # Increased heatmap size
metrics_matrix = model_results_updated[['R²', 'RMSE', 'MAE']].T
sns.heatmap(metrics_matrix, annot=True, fmt='.3f', cmap='YlOrRd',
            xticklabels=model_results_updated['Model'],
            yticklabels=['R²', 'RMSE', 'MAE'],
            cbar_kws={'label': 'Score'})
plt.title('Model Performance Heatmap', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.show()

# Save XGBoost model
import joblib
joblib.dump(xgb_best, 'best_model_xgboost.pkl')
model_results_updated.to_csv('model_comparison_results_with_xgboost.csv', index=False)”
```