

Design Optimization of Pile Foundations

**UG Project Report submitted in the partial fulfillment for the award of
Bachelor of Technology in Civil Engineering**

By

Rahul Singh

20065080

Under the guidance of

Dr. Vishwajit Anand



**Department of Civil Engineering
INDIAN INSTITUTE OF TECHNOLOGY
(BANARAS HINDU UNIVERSITY)
VARANASI – 221005**

November, 2023

UG PROJECT CERTIFICATE

This is to certify that the thesis entitled **Design Optimization of Pile Foundations** submitted by **Rahul Singh** (Roll No. **20065080**), to the Indian Institute of Technology (Banaras Hindu University), Varanasi, in the partial fulfillment for the award of **Bachelor of Technology in Civil Engineering**, is a bona fide record of work done by him/her under my/our supervision. It is certified that the statement made by the student in his/her declaration is correct to the best of my/our knowledge.

Date of Submission: **22 November 2023**

Dr. Vishwajit Anand

Assistant Professor

Department of Civil Engineering

IIT (BHU), Varanasi – 221005

DECLARATION BY THE CANDIDATE

I, **Rahul Singh**, certify that the work embodied in this thesis is my own bona fide work carried out by me under the supervision of **Dr. Vishwajit Anand**, from **01 January 2023** to **20 November 2023** at the Department of Civil Engineering, Indian Institute of Technology (BHU), Varanasi. The matter embodied in this thesis has not been submitted for the award of any other degree/diploma. I declare that I have faithfully acknowledged and/or cited to the researchers wherever their works have been utilized in this thesis. I further declare that I have not willfully copied any other's work, paragraphs, text, data, results, etc., reported in journals, books, magazines, reports dissertation, thesis, etc., or available at websites.

Date: **22 November 2023**

Place: **Varanasi**


(Rahul Singh)

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported me throughout my B.Tech project.

Firstly, I would like to thank my project guide, **Dr. Vishwajit Anand**, for their invaluable guidance, encouragement, and support throughout the project. Their expertise, patience, and willingness to share their knowledge have been instrumental in shaping this project.

I would also like to thank the HOD of the Civil Engineering Department IIT (BHU), Varanasi **Dr. Sasankasekhar Mandal**, and other faculty members of the civil engineering department, for their valuable suggestions and feedback that helped me to refine my project work.

I am grateful to my friends and classmates, who provided me with their moral support and constructive criticism whenever required. Their encouragement and motivation kept me going throughout the project.

Finally, I would like to express my gratitude to my family members, who provided me with the necessary resources and unwavering support during the project. Their love and encouragement have been the driving force behind my success.

Once again, I extend my heartfelt thanks to all those who have supported me in completing this project successfully.

ABSTRACT

Design optimization of pile foundation is a critical aspect of civil engineering, aimed at enhancing the performance of structures built on weak and compressible soil. This abstract introduces an advanced model for the design optimization of concrete pile foundations, a critical component in various civil engineering applications. In this project, we have used a dataset of 2,00,000 testcases to train the model to predict the most optimized pile foundation section with area of reinforcement for both longitudinal as well as lateral direction.

This report includes the implementation a neural network model using TensorFlow and Keras for the design optimization of concrete pile foundations. The model aims to predict key parameters related to pile design, including the diameter of the pile, area of steel reinforcement, and pitch.

The neural network architecture consists of an input layer, a hidden layer with rectified linear unit (ReLU) activation, a dropout layer for regularization, an additional hidden layer, and output layers for each predicted parameter. The model is compiled using the Adam optimizer and mean squared error loss function. Training is performed on a portion of the dataset, with cross-validation to monitor performance.

After training, the model's predictions on the test set are evaluated using various metrics, including R2 scores, explained variances, and accuracy levels within specified tolerances. The code includes visualizations such as pie charts illustrating the distribution of predictions within tolerance, histograms comparing actual and predicted values, scatter plots for each output parameter, and a bar chart depicting individual R2 scores.

This abstract introduces an advanced artificial neural network model tailored for the design optimization of concrete pile foundations. The model showcases the integration of machine learning techniques to predict critical design parameters, providing a valuable tool for engineers in the construction industry. The evaluation metrics and visualizations presented in the code offer insights into the model's performance and its ability to meet accuracy requirements within specified tolerances.

Design of the pile foundation is done considering the live load, spacing of the pile by referring the code book IS456-2000 (plain and reinforced concrete), code book IS2911:2010(Design and constructions of pile foundations), and Design Aids For Reinforced Concrete to IS: 4564978

In conclusion, this project contributes to the development of a robust and efficient design optimization model for pile foundations, which can lead to significant improvements in the performance and cost-effectiveness of structures built on weak and compressible soils.

CONTENTS

UG PROJECT CERTIFICATE.....	2
DECLARATION BY THE CANDIDATE.....	3
ACKNOWLEDGEMENT.....	4
ABSTRACT.....	5
CONTENTS.....	7
I. INTRODUCTION.....	8
I.1. Foundations Classification.....	8
I.2. Pile Foundation.....	9
I.2. i. Purpose of Pile Foundations.....	10
I.2. ii. Classification of Pile Foundations.....	10
I.3. Static Formula for Calculation of bearing force of soil.....	13
II. LITERATURE REVIEW.....	14
III. METHODOLOGY.....	16
III.1. Structural Design of pile doing manual calculation.....	16
III.2. Dataset Generation.....	21
III.3. Training of ML model.....	26
IV. RESULTS AND DISCUSSIONS.....	33
V. CONCLUSIONS AND FUTURE WORK.....	34
VI. REFERENCES.....	36

I. INTRODUCTION

The function of a Foundation is to transfer the structural loads from a building safely into the ground. The structural loads include the dead, superimposed, and wind loads. To perform the function, the foundation must be properly designed and constructed. Its stability depends upon the behavior under load of the soil on which it resists, and this is affected partly by the design of the foundation and partly by the characteristics of the soil.

It is necessary for the design and construction of the foundation to pay attention to the nature and strength of the materials to be used for the foundations as well as the likely behavior under load of the soils on which the foundation rests.

I.1. Foundations Classifications

On the selection of a suitable foundation system for a building, various factors must be taken into consideration. Among them are soil conditions, load transfer patterns, shape and size of the building, site constraints, underground tunnels and services, environmental issues, etc. There are two basic types of foundations:

(1) Shallow Foundation: Those that transfer loads in bearing close to the surface. They either form individual spread footings or mat foundations, which combine the individual footings to support an entire building or part of it. The two systems may also act in combination with each other, for example, where a service core is seated on a large mat while the columns are founded on pad footings. In shallow foundation, the below criteria is satisfied.

$$D_f \leq B_f$$

where D_f means the depth of the foundation

B_f means the width of the foundation

(2) Deep Foundations: The type of foundations distinguished from shallow foundations by the depth they are embedded into the ground. A deep foundation is a type of foundation that transfers building loads to the earth farther down from the surface than a shallow foundation does to a subsurface layer or a range of depths. There are many reasons a civil engineer would recommend a deep foundation over a shallow

foundation, but some of the common reasons are very large design loads, a poor soil at shallow depth, or site constraints (like property lines). There are different terms used to describe different types of deep foundations including piles, drilled shafts, caissons and piers. In a nut shell, a Foundation is deep if its depth is greater than the width.

$$D_f > B_f$$

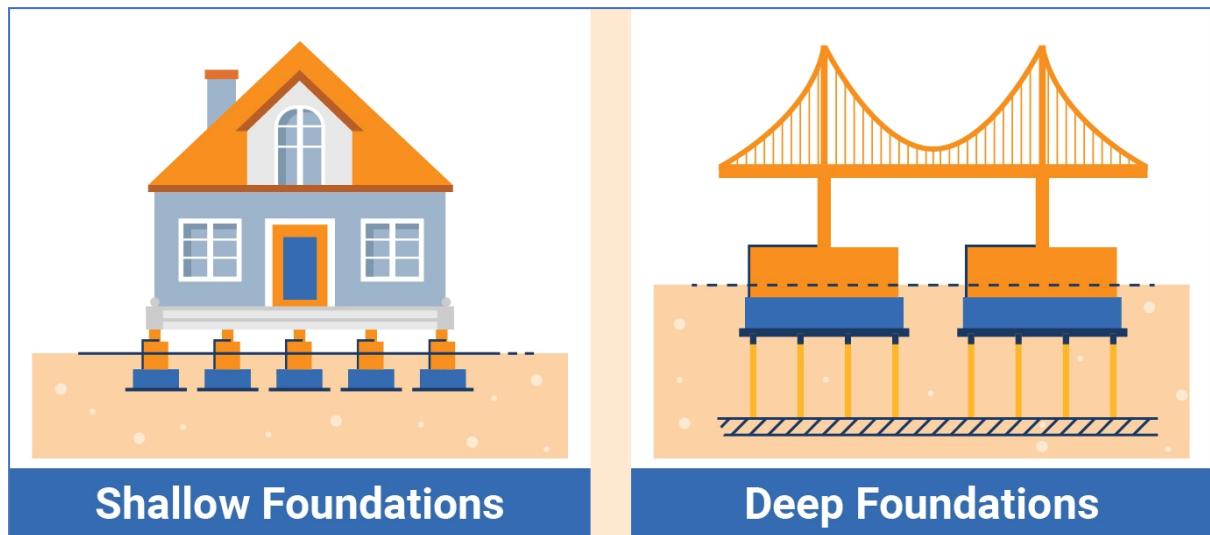


Fig-1. Shallow and Deep foundation diagram

The deep foundations are of three types: -

- > Well foundation
- > Caisson foundation
- > Pile foundation

I.2. Pile Foundation

The word 'pile' is used to describe columns, usually of reinforced concrete, driven into or cast in the ground in order to carry foundation loads to some deep underlying firm stratum or to transmit loads to the subsoil by the friction of their surfaces in contact with the subsoil. The main function of a pile is to transmit loads to lower levels of ground by a combination of friction along their sides and end bearing at the pile point or base as show in the figure below.

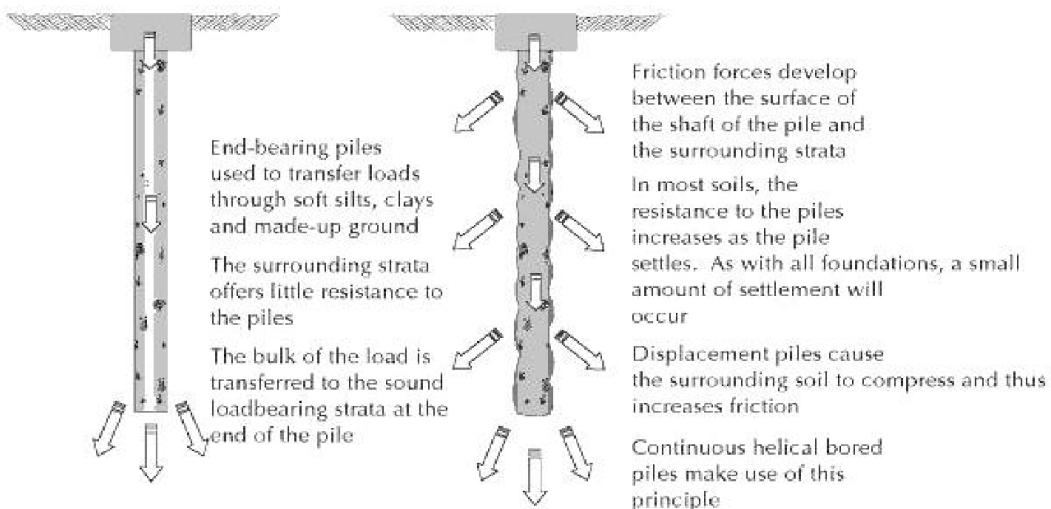


Fig-2 Diagram showing the type of loads acting on a pile

I.2.i. Purpose of Pile Foundation

1. The main purpose of a pile foundation is to transfer the loads into a strong stratum.
2. Generally, pile foundations are used when the bearing capacity of soil is very low and the structural load is heavy.
3. Compressible soil and waterlogged soil is ideal for this type of foundations.
4. Piles are mostly suitable for the foundation of high-rise buildings, bridges, piers, docks etc.

I.2. ii. Classification of Pile Foundation

Depending upon the function, material and method of installation pile foundation can be further classified into following categories;

-> Classification of piles based on load bearing mechanism

The method of load-bearing arrangement is considered here to separate the types of deep foundations. Mainly, there are two methods that we can use to classify the piles.

-> End Bearing Pile: Most types of piles we construct are end-bearing piles. The toe of the pile act to resist the vertical movement of the pile. The penetration resistance of the pile toe is considered the end bearing. Pile is rested on the soil, weather rock, or fresh rock. Depending on the condition of the resting material and toe area, the end-bearing

capacity is varying. The rock produces the highest-end bearing pressure whereas the soil produces the low-end bearing pressure.

The pile end bearing can be calculated as,

$$F_b = f_b A_b$$

F_b – Pile end bearing resistance. A suitable factor of safety shall be applied.

f_b – allowable end bearing, not be obtained from laboratory investigation of soil or rock that rests the pile.

A_b – Pile toe area

Friction Pile: Most of the driven pile types are friction piles. The friction between the soil and the pile creates the penetration resistance of the pile. This is called *friction*. The surface area of the pile and soil overburden, type of soil, etc. are affecting the skin friction of the pile. The skin friction of the pile can be calculated as

$$F_s = f_s A_s$$

F_s – allowable skin friction of the soil/rock

f_s – allowable skin friction stress (from recommendations)

A_s – pile surface area of that section considered

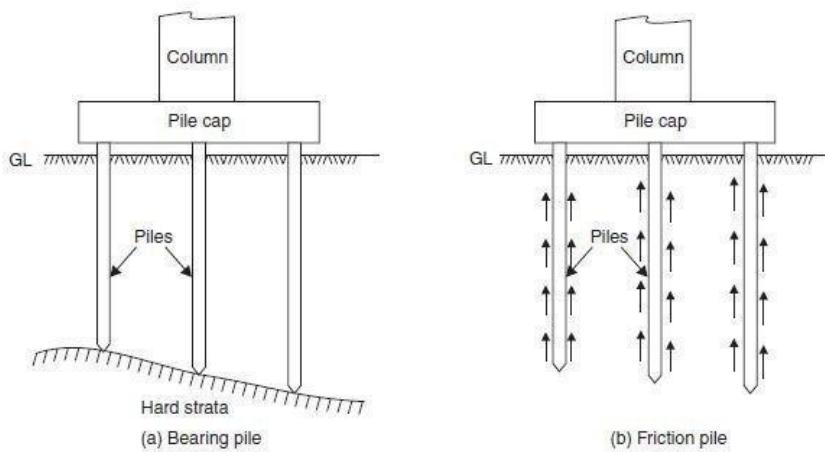


Fig-3. Classification of pile on the basis of load mechanism

Classification of piles based on construction material

-> **Timber pile:** One of the oldest types of piles is timber piles. They were constructed in ancient times. Not only the construction on the land but also the construction of a structure on water, and timber piles are used. Types of wood that have resistance against deterioration timber due to environmental conditions are used for the construction.

-> **Concrete pile:** One of the most frequently used types of pile foundations is concrete piles. It has a higher load-bearing capacity when

compared to the other types of piles as the concrete is very strong in compression. Generally, the concrete pile is constructed as friction piles and end-bearing piles. The precast pile is the friction pile and cast in situ bored piles are classified as end-bearing piles. Further, precast piles are also called driven piles as they cost about the ground and then they later driven into a hard soil layer. Mechanical methods or vibration hammers are used to drive the pile.

-> **Sheet piles:** Sheet piles are not the types of pile foundations used for load-bearing purposes like concrete piles. They are mostly used to resist lateral loads. In shoring construction, permanent sheet pile walls to resist ground movement, etc. are the main uses of sheet piles. The nature of the pile has increased its lateral stiffness. Further, they are tied together by locking each other.

-> **Steel piles:** Steel piles are used on special occasions, and they are not commonly used due to corrosion issues. If the pile is corroded, it will affect structural stability. It is common practice to check the ground condition and groundwater for harmful chemicals that could cause corrosion. Having those will reduce the life span of the structure. Therefore, steel pile construction shall be done with much care.

-> **Composite piles:** Generally, composite piles are a combination of steel and concrete. When we constructed a steel pile covered with concrete it also can be considered a composite pile. However, when we construct the steel pile, we used concrete cover around the pile to protect it against durability issues.

->**Secant piles:** Secant piles are the types of foundations mostly popular in deep basement excavations. When it is required to construct a deep basement without affecting the surrounding structures, we mostly used these types of pile foundations. If the groundwater table is lowered, it will directly affect the structures built on the shallow foundation. In addition, continuous dewatering in a deep excavation is almost impossible without affecting the construction.



I.3. Static Formula for calculation of bearing force of soil

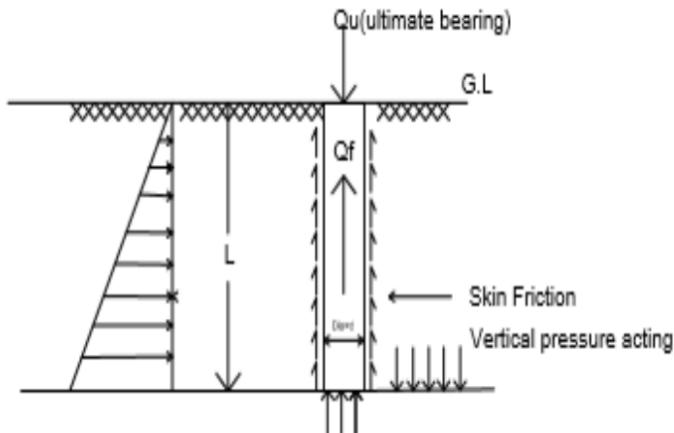


Fig-4. Analysis of pile foundation

$$Qu = Q_b + Q_f$$

$$Q_u = q_b \times A_b + q_{sf} \times A_f$$

where Qu – Ultimate bearing capacity of soil

Q_b – Resistance offered by hard strata

Q_f – Frictional resistance offered by friction

q_b – unit point or toe resistance

A_b – Area of cross-section of pile on which bearing resistance acts

q_{sf} – average skin friction

A_f – surface area of pile upon which the skin friction acts

$$q_b = CN_c + 0.3\gamma BN_y + \sigma'N_q$$

where $\sigma' = \gamma L$

(B is very small, so Neglected)

$$q_b = C + \sigma'N_q$$

$$A_b = \pi d^2 / 4$$

$$Q_f = (\alpha C + \sigma \tan \delta) \pi dl$$

For sandy soil, $C=0$,

$$Q_u = \gamma LN_q \times \pi d^2 / 4 + K\gamma L / 2 \tan \delta \times \pi dl$$

For clay soil, $\phi = 0$

$$N_q = 1; N_c = 9 \text{ (Skempton's theory)}$$

For concrete pile, $\delta = 3/4 \phi$ ($\delta = 0$)

$$Q_u = 9C \times \pi d^2 / 4 + \alpha C \pi dl$$

II. Literature Review

Pile foundations are commonly used to support structures on weak and compressible soil. The design of pile foundations involves the determination of the appropriate size, shape, and number of piles required to support the loads from the structure and to minimize the potential for settlement. Design optimization of pile foundation is an essential aspect of civil engineering that aims to enhance the performance of the foundation system while minimizing the cost and construction time.

In recent years, there has been a significant interest in the development of design optimization models for pile foundations. These models typically employ a combination of analytical and numerical techniques, such as finite element analysis and genetic algorithms, to optimize the design parameters of the pile foundation system. Several studies have focused on the development of such models for different types of structures and soil conditions.

One of the primary challenges in pile foundation design optimization is the consideration of multiple design criteria, such as load capacity, settlement, and constructability. Several researchers have proposed various approaches to address this challenge.

Wang et al. (2020) proposed a multi-objective optimization model that considers the load capacity, settlement, and construction cost of the pile foundation. The model employs the non-dominated sorting genetic algorithm (NSGA-II) to generate a set of Pareto-optimal solutions that balance the competing objectives.

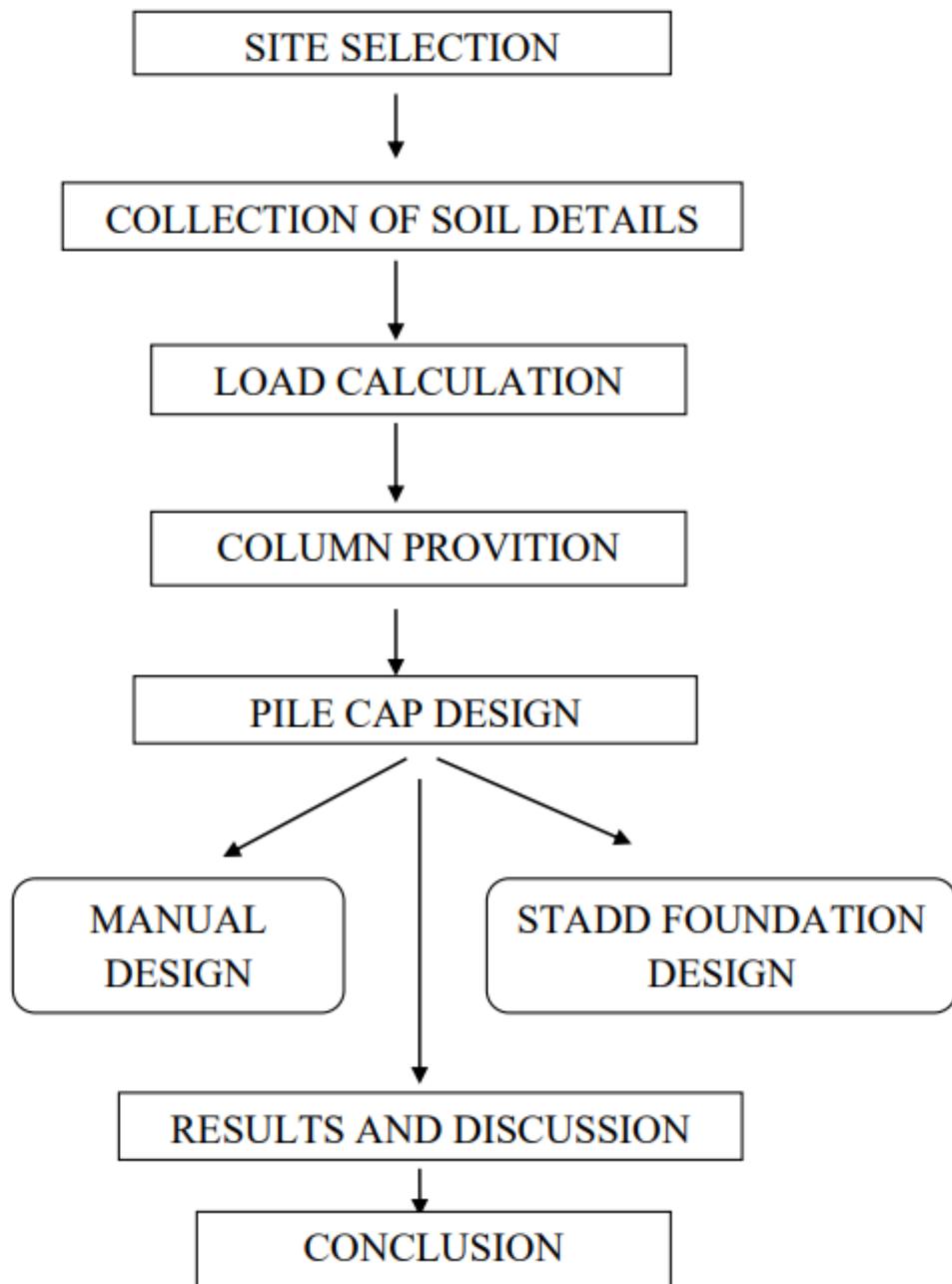
In India, the Bureau of Indian Standards (BIS) provides guidelines for the design of pile foundations in different soil conditions. These guidelines include the use of various methods, such as static load tests, dynamic load tests, and pile driving formulae, to estimate the load capacity and settlement of pile foundations. BIS also recommends the use of safety factors to account for uncertainties in soil properties and loading conditions.

Thakur et al. (2018) proposed a design optimization model for pile foundations using finite element analysis and response surface methodology. The model considers the load capacity, settlement, and

construction cost of the pile foundation and optimizes the design parameters to minimize the total cost.

In conclusion, design optimization of pile foundation is a critical aspect of geotechnical engineering that has attracted significant attention in recent years. The development of robust and efficient design optimization models that consider multiple design criteria and uncertainties can lead to significant improvements in the performance and cost-effectiveness of pile foundation systems.

III. METHODOLOGY



The design of pile foundations has a lot of aspects. In this report, we are mainly going to see the structural design aspects of pile foundation of concrete piles. The structural design of the pile typically involves determining the required cross-sectional dimensions, such as the diameter and wall thickness, area of lateral and longitudinal reinforcement require to ensure that the pile can resist the applied loads

and moments. A single pile can be designed considering it as a RCC column subjected to axial load and biaxial bending.

III.1. Structural design of pile doing manual calculation

The structural design of a pile foundation involves several steps, including determining the required pile size, estimating the load capacity of the pile, and designing the pile to resist the applied loads. These calculations can be done using manual methods, which involve using various equations and formulas to estimate the required parameters.

Manual calculations can be time-consuming and prone to errors, but they are still used in some cases where the design is relatively simple or when there is limited access to computer software. However, the use of software and computer-based tools has become increasingly common in recent years, allowing for faster and more accurate pile design calculations.

Problem: Design a pile under a column transmitting an axial load of 400 kN and moment in x as 120 KNm and moment in y as 130 kNm. The pile is to be driven to a hard stratum available at a depth of 6m.

Solution:

Step-1.

Given

Load: 400 kN

Moment in x = 120 kNm

Moment in y = 130 kNm

Factored Load: $1.5 \times 400 = 600$ kN

Factored Moment in x: $1.5 \times 150 = 180$ kNm

Factored Moment in y: $1.5 \times 130 = 195$ kNm

Use M25 concrete and Fe415 steel

Let the length of the pile above ground, including pile capacity = 0.6 m

Total length of pile = 6.6 m

Let us take the diameter of pile as 600 mm X 600 mm

Slenderness ratio (λ) = L_{eff}/LLD

$$\lambda = (0.8 * 6.6) / 0.6 = 8.8 \leq 12$$

Step-2. Minimum eccentricity

As per IS 456: 2000, clause 25.4, all columns shall be designed for minimum eccentricity, equal to the summation of the unsupported length of column divided by 500 and lateral dimensions divided by 30, subject to a minimum of 20 mm.

$$e_{\min} = \max(l/500 + D/30 \text{ or } 20\text{mm})$$

$$e_{\min x-x} = e_{\min y-y} = \max(6600/500 + 600/30, 20) = 33.2 \text{ mm}$$

Step-3. Minimum Bending Moment

$$M_{x-x} = M_{y-y} = 600 \times \frac{33.2}{1000} = 19.92 \text{ kNm}$$

$$M_{ux} \geq M_{x-x}$$

$$M_{uy} \geq M_{y-y}$$

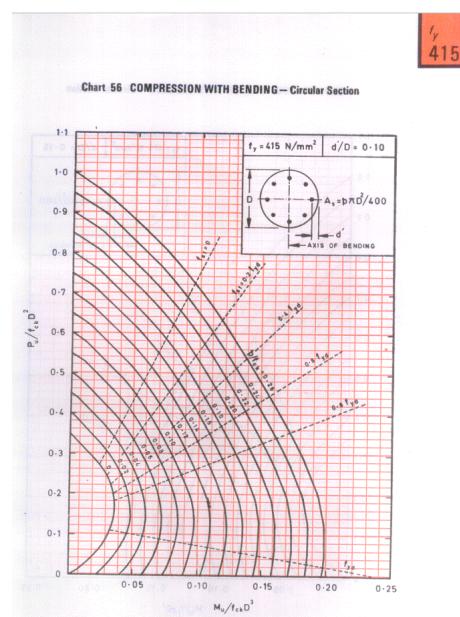
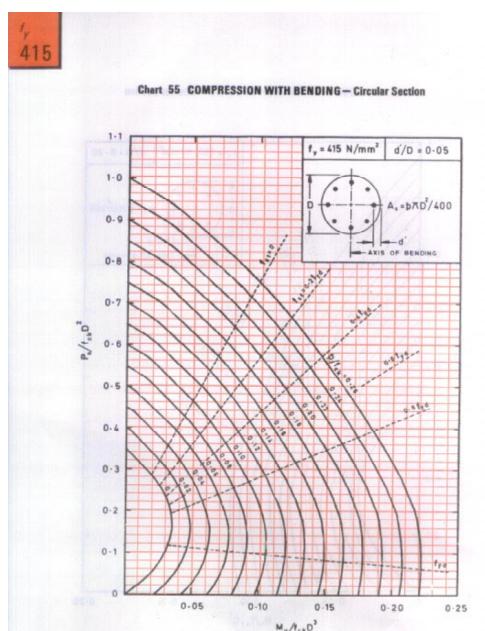
Hence, we have to design it for axial load with biaxial bending moment

Step-4. Finding M_{ux} and M_{uy}

Lets assume an effective cover of 60mm

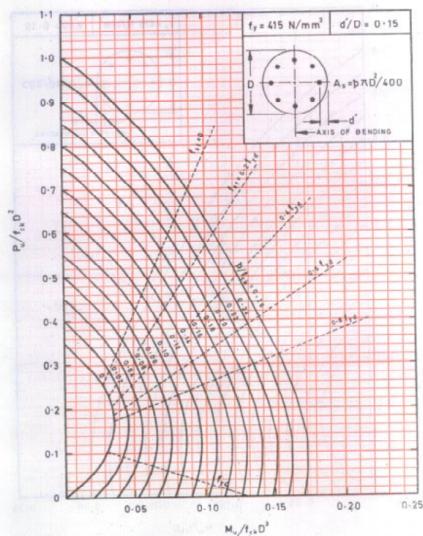
$$\frac{d}{D} = \frac{60}{600} = 0.1$$

Now, we are going to use charts-55,56,57,58 DESIGN AIDS FOR REINFORCED CONCRETE TO IS : 456- 1978



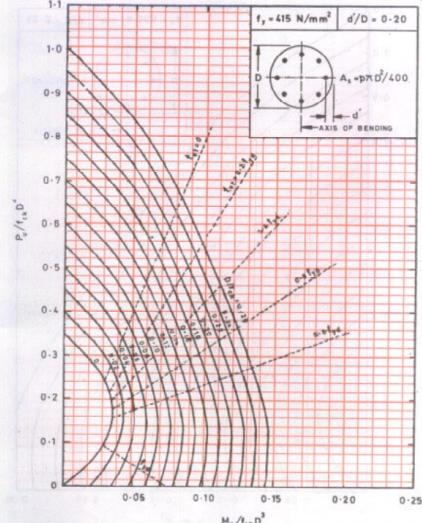
415

Chart 57 COMPRESSION WITH BENDING—Circular Section



415

Chart 58 COMPRESSION WITH BENDING—Circular Section



As $d/D = 0.1$ we are going to use chart - 56

$$y\text{-value} = \frac{P}{f_{ck}kD^3} = (600 \times 10^3) / (25 \times 600^2) = 0.0667$$

Lets assume %p be 1.4%.

$$k\text{-value} = (\%p)/f_{ck} = 1.4/25 = 0.056$$

Now using interpolation x-value = 0.07

$$M_{ux}/f_{ck}D^3 = 0.07$$

$$M_{ux} = 378 \text{ kNm}$$

$$M_{uy} = 378 \text{ kNm}$$

39.6 Members Subjected to Combined Axial Load and Biaxial Bending

The resistance of a member subjected to axial force and biaxial bending shall be obtained on the basis of assumptions given in 39.1 and 39.2 with neutral axis so chosen as to satisfy the equilibrium of load and moments about two axes. Alternatively such members may be designed by the following equation:

$$\left[\frac{M_{ux}}{M_{uxl}} \right]^{\alpha_x} + \left[\frac{M_{uy}}{M_{uyl}} \right]^{\alpha_y} \leq 1.0$$

where

M_{ux}, M_{uy} = moments about x and y axes due to design loads,

M_{uxl}, M_{uyl} = maximum uniaxial moment capacity for an axial load of P_u , bending about x and y axes respectively, and

α_x is related to P/P_{ux}

where $P_{ux} = 0.45 f_{ct} \cdot A_c + 0.75 f_y \cdot A_{sc}$

For values of P_u/P_{ux} = 0.2 to 0.8, the values of α_x vary linearly from 1.0 to 2.0. For values less than 0.2, α_x is 1.0; for values greater than 0.8, α_x is 2.0.

$$A_s = (p^* \pi^* D^2) / 400 = 3958.4 \text{ mm}^2$$

$$P_z = 0.45 f_{ck} A_c + 0.75 f_y A_s$$

$$P_z = 0.45 * 25 * (\pi * 600^2 - 3958.4) + 0.75 * 415 * 3958.4 = 13910.97 \text{ kN}$$

$$P_u/P_z = 600/13910.97 = 0.04$$

Value of $\alpha_n = 1$

Now, lets check for the condition

$$(180/378 + 175/378) = 0.93 \leq 1$$

Hence, design is safe

Now, provide 8 bars of 25mm diameter as main bar

Step-5: Design for pitch

39.4 Compression Members with Helical Reinforcement

The strength of compression members with helical reinforcement satisfying the requirement of 39.4.1 shall be taken as 1.05 times the strength of similar member with lateral ties.

39.4.1 The ratio of the volume of helical reinforcement to the volume of the core shall not be less than $0.36 (A_s/A_c - 1) f_y/f_c$,

where

A_s = gross area of the section,

A_c = area of the core of the helically reinforced column measured to the outside diameter of the helix,

f_c = characteristic compressive strength of the concrete, and

f_y = characteristic strength of the helical reinforcement but not exceeding 415 N/mm^2 .

Lets assume the diameter of spiral reinforcement as 10mm
For pitch the following condition should be satisfied

$$0.36 * (f_{ck}/f_y) * \left(\frac{A_g}{A_c} - 1 \right) \leq \frac{V_h}{V_c}$$

where $A_g = \pi * 600 * 600/4$

$$A_c = \pi * d_c * d_c / 4 \text{ where } d_c = D - 2*d = 600 - 2*60 = 480$$

$$V_c = A_c * 1000$$

$$V_h = \pi * 10 * 10/4 * (1000/p) * \pi * (480 - 10)$$

After solving we get $p \geq 33.62\text{mm}$

d) Helical reinforcement

- 1) *Pitch*—Helical reinforcement shall be of regular formation with the turns of the helix spaced evenly and its ends shall be anchored properly by providing one and a half extra turns of the spiral bar. Where an increased load on the column or the strength of the helical reinforcement is allowed for, the pitch of helical turns shall be not more than 75 mm, nor more than one-sixth of the core diameter of the column, nor less than 25 mm, nor less than three times the diameter of the steel bar forming the helix. In other cases, the requirements of 26.5.3.2 shall be complied with.
- 2) The diameter of the helical reinforcement shall be in accordance with 26.5.3.2 (c) (2).

$$\text{Max pitch} = \min(75\text{mm}, d_c/6) = 75\text{mm}$$

$$\text{Min pitch} = \max(25\text{mm}, 3*10) = 25\text{mm}$$

Hence, taking spiral reinforcement of 10 mm diameter having a pitch of 35mm.

III.2. Dataset generation

In the design process of a pile foundation, accurate and efficient calculations are essential to ensure that the foundation system can safely and effectively support the applied loads.

In the pursuit of advancing engineering solutions in the realm of civil infrastructure, a meticulously crafted dataset of 2 lakh random inputs for pile foundation design has been generated using a sophisticated C++ codebase.

The C++ code intelligently simulates various scenarios and conditions that mimic real-world situations encountered in the construction of pile foundations. The dataset encompasses a comprehensive set of

parameters essential for pile design, meticulously curated to provide a holistic representation of the factors influencing foundation behavior.

This part involves the digitization of charts of SP - 16. Using them to get outputs for randomly generated inputs. The input fields are Load, Moment in x (in kNm), Moment in y (int kNm) and the output fields are Diameter of pile (in m), Area of steel (in mm²), and pitch of spiral reinforcement (in mm).



The screenshot shows a code editor with a dark theme. The left sidebar contains icons for file operations like Open, Save, Find, and Undo. The main area displays a C++ file named 'first.cpp'. The code implements a search algorithm using binary search on sorted vectors. It includes functions for finding lower bounds and calculating x-values based on load and dimensions.

```
101
102 unordered_map<double, vector<pair<double, double>> m1;
103 unordered_map<double, vector<pair<double, double>> m2;
104 unordered_map<double, vector<pair<double, double>> m3;
105 unordered_map<double, vector<pair<double, double>> m4;
106
107 double lowerBound(vector<pair<double, double>> &v, double ab)
108 {
109     int low = 0, high = v.size();
110     int ans = v.size();
111     while (low <= high)
112     {
113         int mid = (low + high) >> 1;
114         if (v[mid].first >= ab)
115         {
116             ans = mid;
117             high = mid - 1;
118         }
119         else
120         {
121             low = mid + 1;
122         }
123     }
124     return ans;
125 }
126
127 double x_value(double d, double D, double load, double p)
128 {
129     double y = (load * 1000 / (25 * D * D));
130     double r = (d / D);
131     if (r <= 0.05)
132     {
133         auto it = lowerBound(m1[p], y);
134         if (it == m1[p].size())
135         {
136             return 1e5;
137         }
138         else
139         {
140             auto x = m1[p].at(it).second * 25 * D * D * D;
141             return x;
142         }
143     }
144     else if (r > 0.05 && r <= 0.10)
145     {
146         auto it = lowerBound(m2[p], y);
147         if (it == m2[p].size())
148         {
149             return 1e5;
150         }
151         else
152         {
153             auto x = m2[p].at(it).second * 25 * D * D * D;
154             return x;
155         }
156     }
157 }
```

```
153     auto x = m2[p].at(it).second * 25 * D * D * D;
154     return x;
155 }
156 }
157 else if (r > 0.10 && r <= 0.15)
158 {
159     auto it = lowerBound(m3[p], y);
160     if (it == m3[p].size())
161     {
162         return 1e5;
163     }
164     else
165     {
166         auto x = m3[p].at(it).second * 25 * D * D * D;
167         return x;
168     }
169 }
170 else
171 {
172     auto it = lowerBound(m4[p], y);
173     if (it == m4[p].size())
174     {
175         return 1e5;
176     }
177     else
178     {
179         auto x = m4[p].at(it).second * 25 * D * D * D;
180         return x;
181     }
182 }
183 }

184 int32_t main()
185 {
186     freopen("input.txt", "r", stdin);
187     freopen("output.txt", "w", stdout);
188
189 // first take input of force, Mx and My
190
191     for (int j = 0; j < 4; j++)
192     {
193         double ab = 0.0;
194         for (int i = 0; i < 14; i++)
195         {
196             if (j == 0)
197             {
198                 for (auto &it : chart_55[i])
199                 {
200                     m1[ab].push_back({it.second, it.first});
201                 }
202             }
203             else if (j == 1)
204         }
```

first.cpp

```

206     for (auto &it : chart_56[i])
207     {
208         m2[ab].push_back({it.second, it.first});
209     }
210 }
211 else if (j == 2)
212 {
213     for (auto &it : chart_57[i])
214     {
215         m3[ab].push_back({it.second, it.first});
216     }
217 }
218 else
219 {
220     for (auto &it : chart_58[i])
221     {
222         m4[ab].push_back({it.second, it.first});
223     }
224 }
225 ab += 0.02;
226 }
227 }
228 for (auto &it : m1)
229 {
230     sort(it.second.begin(), it.second.end());
231 }
232 for (auto &it : m2)
233 {
234     sort(it.second.begin(), it.second.end());
235 }
236 for (auto &it : m3)
237 {
238     sort(it.second.begin(), it.second.end());
239 }
240 for (auto &it : m4)
241 {
242     sort(it.second.begin(), it.second.end());
243 }
244 for (auto &it : m5)
245 {
246     sort(it.second.begin(), it.second.end());
247 }
248 int t;
249 cin >> t;
250 srand(time(NULL));
251 vector<vector<double>> v;
252 while (t--)
253 {
254     double load, moment_in_x, moment_in_y, length;
255     load = rand()%900+1;
256     if(load < 300)
257         load += 300;
258     moment_in_x = rand()%360+1;
259 }

```

Run Testcases

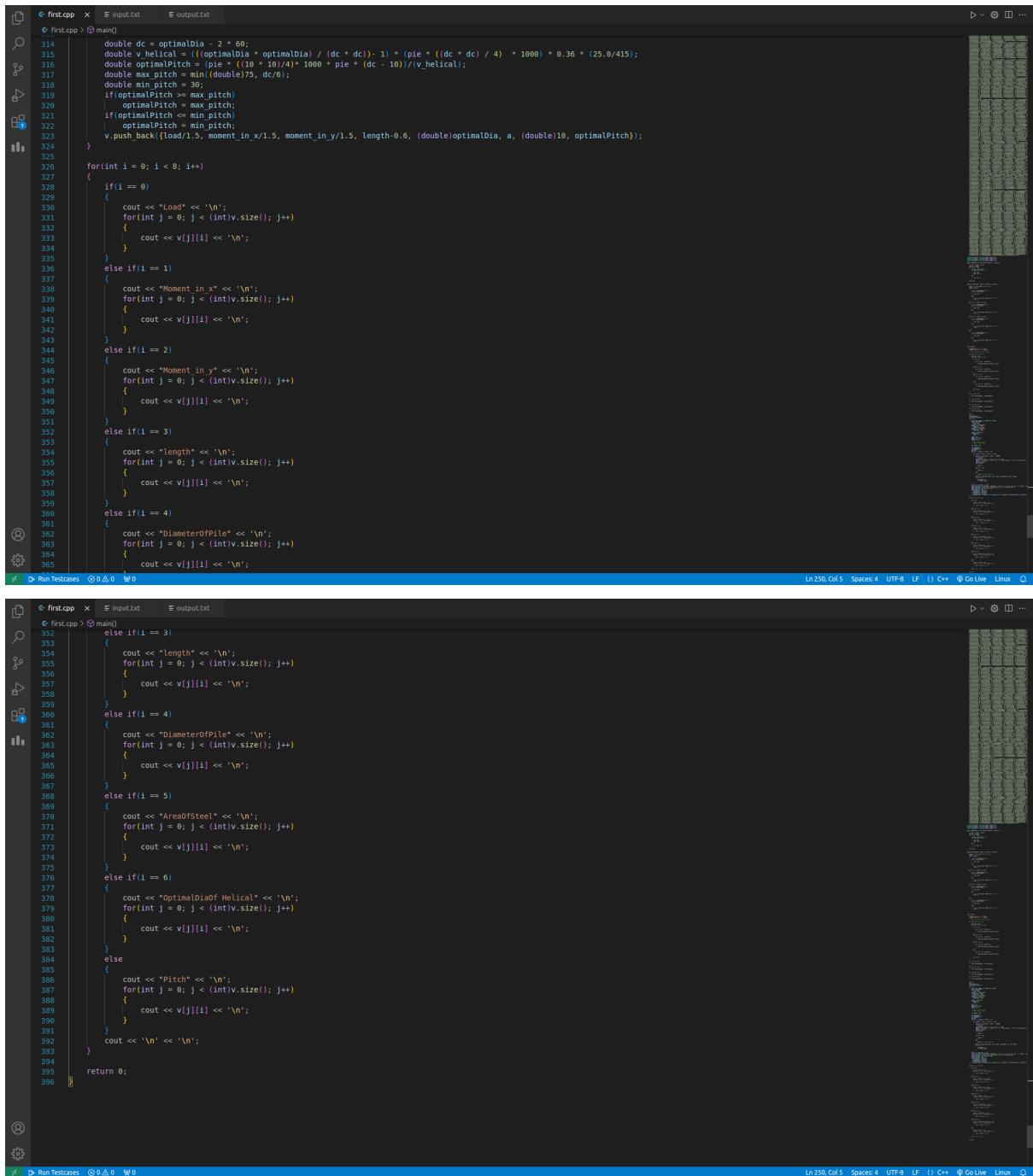
first.cpp

```

261     moment_in_y = rand()%300+1;
262     if(moment_in_y < 100)
263         moment_in_y += 100;
264
265     length = rand()%10+1;
266     if(length < 6)
267         length += 5;
268
269     load += 1.5;
270     moment_in_x += 1.5;
271     moment_in_y += 1.5;
272     length += 0.6;
273
274 // Grade of concrete - M25
275 // Grade of steel - Fe415
276
277     int pileDia = 900;
278
279     int optimalDia();
280     int numberOfBar = 0;
281     int DiaOfBar = 0;
282     double a;
283     for (int i = pileDia; i >= 300; i -= 5)
284     {
285         for (double j = 0.08; j >= 0.04; j -= 0.02)
286         {
287             double ab = x_value(60, i, load, j) / 1000000;
288             if (ab == 1e5)
289                 continue;
290             double area_of_steel = (j * 25 + pie * i * i) / 400;
291             double pz = ((0.45 * 25 + ((pie * i * i) / 4) - area_of_steel)) + 0.75 * 415 * area_of_steel) / 1000;
292             double r = load / pz;
293             double alpha;
294             if (r <= 0.2)
295             {
296                 alpha = 1;
297             }
298             else if (r > 0.8)
299             {
300                 alpha = 2;
301             }
302             else
303             {
304                 alpha = (r - 0.8) / 0.6 + 2;
305             }
306             double val = pow((moment_in_x / ab), alpha) + pow((moment_in_y / ab), alpha);
307             if (val <= 1)
308             {
309                 optimalDia = i;
310                 a = area_of_steel;
311             }
312         }
313     }

```

Run Testcases



```

g++ first.cpp
make
./first
  double dc = optimalDia - 2 * 60;
  double v_helical = ((optimalDia * optimalDia) / (dc * dc)) - 1) * (pie * ((dc * dc) / 4) + 1000) * 0.36 * (25.0/415);
  double optimalPitch = (pie * ((10 * 10)/4) * 1000 * pie * (dc - 10))/v_helical;
  double max_pitch = min((double)75, dc/6);
  double min_pitch = 30;
  if(optimalPitch >= max_pitch)
    optimalPitch = max_pitch;
  if(optimalPitch < min_pitch)
    optimalPitch = min_pitch;
  v.push_back(load/1.5, moment_in_x/1.5, moment_in_y/1.5, length-0.6, (double)optimalDia, a, (double)10, optimalPitch);

for(int i = 0; i < 8; i++)
{
  if(i == 0)
  {
    cout << "Load" << '\n';
    for(int j = 0; j < (int)v.size(); j++)
    {
      cout << v[j][i] << '\n';
    }
  }
  else if(i == 1)
  {
    cout << "Moment_in_x" << '\n';
    for(int j = 0; j < (int)v.size(); j++)
    {
      cout << v[j][i] << '\n';
    }
  }
  else if(i == 2)
  {
    cout << "Moment_in_y" << '\n';
    for(int j = 0; j < (int)v.size(); j++)
    {
      cout << v[j][i] << '\n';
    }
  }
  else if(i == 3)
  {
    cout << "length" << '\n';
    for(int j = 0; j < (int)v.size(); j++)
    {
      cout << v[j][i] << '\n';
    }
  }
  else if(i == 4)
  {
    cout << "DiameterOfPile" << '\n';
    for(int j = 0; j < (int)v.size(); j++)
    {
      cout << v[j][i] << '\n';
    }
  }
}

else if(i == 3)
{
  cout << "length" << '\n';
  for(int j = 0; j < (int)v.size(); j++)
  {
    cout << v[j][i] << '\n';
  }
}
else if(i == 4)
{
  cout << "DiameterOfPile" << '\n';
  for(int j = 0; j < (int)v.size(); j++)
  {
    cout << v[j][i] << '\n';
  }
}
else if(i == 5)
{
  cout << "AreaOfSteel" << '\n';
  for(int j = 0; j < (int)v.size(); j++)
  {
    cout << v[j][i] << '\n';
  }
}
else if(i == 6)
{
  cout << "OptimalDiaof_Helical" << '\n';
  for(int j = 0; j < (int)v.size(); j++)
  {
    cout << v[j][i] << '\n';
  }
}
else
{
  cout << "Pitch" << '\n';
  for(int j = 0; j < (int)v.size(); j++)
  {
    cout << v[j][i] << '\n';
  }
}
cout << '\n' << '\n';

return 0;
}

```

Once the test cases have been generated, the dataset is used to test the design criteria under various conditions. The test results can be analysed to determine if the design criteria are accurate and if any changes need to be made to improve the design process. The dataset can also be used to perform statistical analyses, such as regression analyses, to determine the sensitivity of the design criteria to the input parameters.

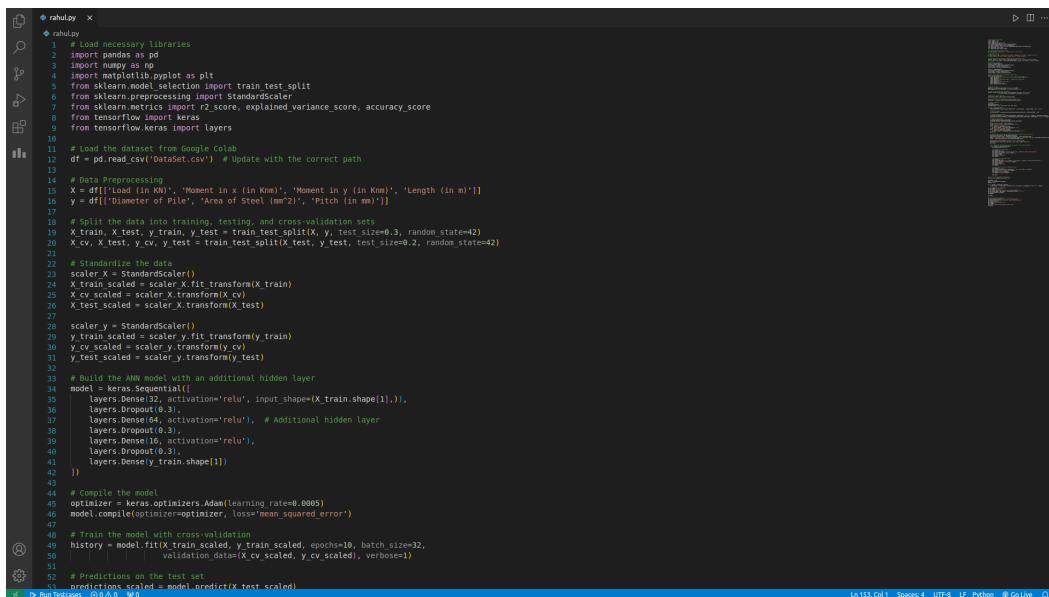
By testing the design criteria using a large dataset of test cases, the accuracy and reliability of the design process can be improved. This can help to ensure that the pile foundation can safely and effectively support the applied loads and provide long-lasting stability to the structure.

After generation of the test cases, it is converted to .csv file to train our ML model.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Load (in KN)	Moment in x (in KNm)	Moment in y (in KNm)	Length (in m)	Diameter of Pile	Area of Steel (mm²)	Pitch (in mm)						
2	559	231	121	6	585	5375.65	52.3112						
3	3535	231	102	6	625	6135.92	52.8812						
4	486	154	249	8	665	6946.45	53.3756						
5	303	180	195	7	660	6842.39	53.3174						
6	420	141	119	8	575	5193.45	52.1551						
7	742	219	251	6	645	6534.01	53.1368						
8	742	283	158	10	685	7370.57	53.5991						
9	398	164	144	8	610	5844.93	52.6772						
10	518	165	102	9	580	5284.16	52.234						
11	747	103	154	8	555	4883.45	51.8236						
12	556	154	297	6	690	7478.56	53.6527						
13	485	191	217	6	670	7051.3	53.4328						
14	531	235	158	7	660	6842.39	53.3174						
15	705	161	174	8	620	6038.14	52.1314						
16	643	132	139	7	576	5193.45	52.1551						
17	447	183	299	6	715	8030.3	53.9087						
18	647	153	151	10	600	5654.97	52.5349						
19	507	156	277	8	685	7370.57	53.5991						
20	561	125	225	10	638	6333.84	53.0112						
21	715	141	252	10	660	6842.39	53.3174						
22	427	162	168	8	625	6135.92	52.8812						
23	474	142	132	8	585	5375.65	52.3112						
24	425	171	160	6	625	6135.92	52.8812						
25	553	210	139	6	635	6333.84	53.0112						
26	562	189	147	9	625	6135.92	52.8812						
27	645	176	183	10	640	6433.98	53.0746						
28	700	159	202	10	645	6433.98	53.0746						
29	799	205	111	10	600	5654.97	52.5349						
30	566	179	159	6	630	6234.49	52.9468						
31	572	190	159	6	635	6333.84	53.0112						
32	480	170	110	9	595	5501.01	52.4618						
33	579	296	269	8	725	8256.5	54.0057						
34	445	113	213	6	620	6038.14	52.8144						
35	388	147	297	7	695	7587.34	53.7055						
36	400	260	153	8	675	7186.44	53.4801						
37	755	164	179	10	625	6135.92	52.8812						
38	394	189	111	10	605	5749.51	52.6067						
39	878	221	160	10	640	6433.98	53.0746						
40	690	277	105	10	640	6433.98	53.0746						
41	833	171	112	9	575	5193.45	52.1551						
42	735	276	169	9	685	7370.57	53.5991						
43	322	117	160	8	595	5561.01	52.4618						
44	367	165	232	6	670	7051.3	53.4328						
45	326	162	192	6	670	7051.3	53.4328						

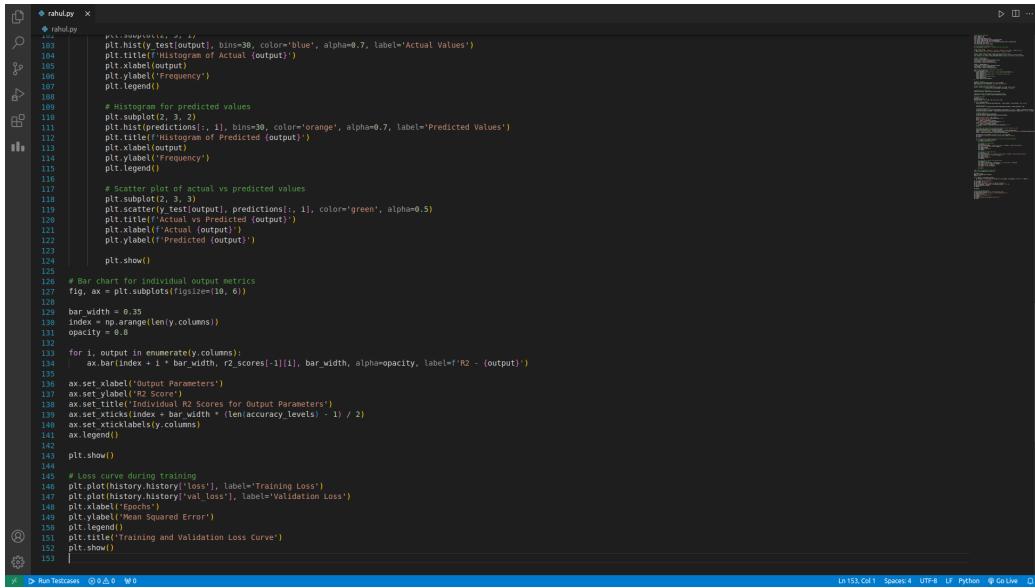
III.3. Training of ML model

The ML model is Artificial Neural Network (ANN) model which uses the TensorFlow and Keras libraries to optimize the design of concrete pile foundations. The objective is to predict key parameters crucial for pile design, including the diameter of the pile, area of steel reinforcement, and pitch. This analysis will delve into the various components of the code, from data preprocessing to model evaluation.



```

1 # Load necessary libraries
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from sklearn.model_selection import train_test_split
6 from sklearn.preprocessing import StandardScaler
7 from sklearn.metrics import r2_score, explained_variance_score, accuracy_score
8 from tensorflow import keras
9 from tensorflow.keras import layers
10
11 # Load the dataset from Google Colab
12 df = pd.read_csv('DataSet.csv') # Update with the correct path
13
14 # Data Preprocessing
15 X = df[['Load (in KN)', 'Moment in x (in Km)', 'Length (in m)']]
16 Y = df[['Diameter of pile', 'Area of Steel (mm²)', 'Pitch (in mm)']]
17
18 # Split the data into training, testing, and cross-validation sets
19 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
20 X_cv, X_test, y_cv, y_test = train_test_split(X_test, y_test, test_size=0.2, random_state=42)
21
22 # Standardize the data
23 scaler_X = StandardScaler()
24 X_train_scaled = scaler_X.fit_transform(X_train)
25 X_cv_scaled = scaler_X.transform(X_cv)
26 X_test_scaled = scaler_X.transform(X_test)
27
28 scaler_y = StandardScaler()
29 y_train_scaled = scaler_y.fit_transform(y_train)
30 y_cv_scaled = scaler_y.transform(y_cv)
31 y_test_scaled = scaler_y.transform(y_test)
32
33 # Build the ANN model with an additional hidden layer
34 model = keras.Sequential([
35     layers.Dense(32, activation='relu'), input_shape=(X_train.shape[1]),),
36     layers.Dropout(0.3), # Additional hidden layer
37     layers.Dense(64, activation='relu'),
38     layers.Dense(10, activation='relu'),
39     layers.Dense(10, activation='relu'),
40     layers.Dropout(0.3),
41     layers.Dense(y_train.shape[1])
42 ])
43
44 # Compile the model
45 optimizer = keras.optimizers.Adam(learning_rate=0.0005)
46 model.compile(optimizer=optimizer, loss='mean_squared_error')
47
48 # Train the model with cross-validation
49 history = model.fit(X_train_scaled, y_train_scaled, epochs=10, batch_size=32,
50                     validation_data=(X_cv_scaled, y_cv_scaled), verbose=1)
51
52 # Predictions on the test set
53 predictions_scaled = model.predict(X_test_scaled)
54
55 # Inverse transform the predictions to the original scale
56 predictions = scaler_y.inverse_transform(predictions_scaled)
57
58 # Evaluate the model
59 r2_scores = []
60 explained_variances = []
61 accuracy_levels = [0.1, 0.08, 0.05, 0.04, 0.03, 0.02]
62
63 for tol in accuracy_levels:
64     within_tolerance = np.all(np.abs(predictions - y_test.values) / y_test.values < tol, axis=1)
65
66     # Overall accuracy
67     accuracy_overall = accuracy_score(np.ones_like(within_tolerance), within_tolerance) * 100
68
69     # Individual output metrics
70     r2_scores_individual = [r2_score(y_test[output], predictions[:, i]) for i, output in enumerate(y.columns)]
71     explained_variances_individual = [explained_variance_score(y_test[output], predictions[:, i]) for i, output in enumerate(y.columns)]
72
73     # Append to the lists
74     r2_scores.append(r2_scores_individual)
75     explained_variances.append(explained_variances_individual)
76
77     print(f'Tolerance Level: ±{tol * 100}%')
78     print(f'Overall Accuracy: {accuracy_overall:.2f}%')
79     print(f'Individual R2 Scores: ')
80     for i, output in enumerate(y.columns):
81         print(f'{output}: {r2_scores_individual[i]:.4f}')
82     print(f'Individual Explained Variances: ')
83     for i, output in enumerate(y.columns):
84         print(f'{output}: {explained_variances_individual[i]:.4f}')
85     print('---')
86
87     # Create the distribution of predictions within tolerance using a pie chart
88     within_tolerance_count = np.sum(within_tolerance)
89     not_within_tolerance_count = len(within_tolerance) - within_tolerance_count
90     labels = [f'Within Tolerance ({within_tolerance_count})', f'Not Within Tolerance ({not_within_tolerance_count})']
91     sizes = [within_tolerance_count, not_within_tolerance_count]
92
93     plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
94     plt.title(f'Distribution of Predictions within ±{tol * 100}% Tolerance')
95     plt.show()
96
97     # Plot histograms for predicted and actual values of each output parameter
98     for i, output in enumerate(y.columns):
99         plt.figure(figsize=(12, 6))
100
101         # Histogram for actual values
102         plt.subplot(2, 3, 1)
103         plt.hist(y[y.columns[i]], bins=30, color='blue', alpha=0.7, label='Actual Values')
104         plt.title(f'Histogram of Actual {output}')
105
106         # Histogram for predicted values
107         plt.subplot(2, 3, 2)
108         plt.hist(predictions[:, i], bins=30, color='red', alpha=0.7, label='Predicted Values')
109         plt.title(f'Histogram of Predicted {output}')
110
111         # Overlay both histograms
112         plt.subplot(2, 3, 3)
113         plt.hist(y[y.columns[i]], bins=30, color='blue', alpha=0.7, label='Actual Values')
114         plt.hist(predictions[:, i], bins=30, color='red', alpha=0.7, label='Predicted Values')
115         plt.title(f'Comparison of Actual vs Predicted {output}')
116
117         plt.tight_layout()
118         plt.show()
119
120         # Scatter plot of actual vs predicted values
121         plt.figure(figsize=(12, 6))
122         plt.scatter(y[y.columns[i]], predictions[:, i])
123         plt.title(f'Actual vs Predicted {output}')
124         plt.xlabel('Actual Value')
125         plt.ylabel('Predicted Value')
126         plt.show()
127
128         # Correlation matrix
129         plt.figure(figsize=(12, 6))
130         sns.heatmap(y.corr(), annot=True, cmap='coolwarm')
131         plt.title('Correlation Matrix')
132         plt.show()
133
134         # Box plot of residuals
135         plt.figure(figsize=(12, 6))
136         sns.boxplot(y[y.columns[i]] - predictions[:, i])
137         plt.title(f'Residuals for {output}')
138         plt.show()
139
140         # Scatter plot of residuals vs predicted values
141         plt.figure(figsize=(12, 6))
142         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
143         plt.title(f'Residuals vs Predicted {output}')
144         plt.xlabel('Predicted Value')
145         plt.ylabel('Residual')
146         plt.show()
147
148         # Scatter plot of residuals vs actual values
149         plt.figure(figsize=(12, 6))
150         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
151         plt.title(f'Residuals vs Actual {output}')
152         plt.xlabel('Actual Value')
153         plt.ylabel('Residual')
154         plt.show()
155
156         # Box plot of residuals vs actual values
157         plt.figure(figsize=(12, 6))
158         sns.boxplot(y[y.columns[i]] - predictions[:, i])
159         plt.title(f'Residuals vs Actual {output}')
160         plt.show()
161
162         # Scatter plot of residuals vs predicted values
163         plt.figure(figsize=(12, 6))
164         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
165         plt.title(f'Residuals vs Predicted {output}')
166         plt.xlabel('Predicted Value')
167         plt.ylabel('Residual')
168         plt.show()
169
170         # Box plot of residuals vs predicted values
171         plt.figure(figsize=(12, 6))
172         sns.boxplot(y[y.columns[i]] - predictions[:, i])
173         plt.title(f'Residuals vs Predicted {output}')
174         plt.show()
175
176         # Scatter plot of residuals vs actual values
177         plt.figure(figsize=(12, 6))
178         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
179         plt.title(f'Residuals vs Actual {output}')
180         plt.xlabel('Actual Value')
181         plt.ylabel('Residual')
182         plt.show()
183
184         # Box plot of residuals vs actual values
185         plt.figure(figsize=(12, 6))
186         sns.boxplot(y[y.columns[i]] - predictions[:, i])
187         plt.title(f'Residuals vs Actual {output}')
188         plt.show()
189
190         # Scatter plot of residuals vs predicted values
191         plt.figure(figsize=(12, 6))
192         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
193         plt.title(f'Residuals vs Predicted {output}')
194         plt.xlabel('Predicted Value')
195         plt.ylabel('Residual')
196         plt.show()
197
198         # Box plot of residuals vs predicted values
199         plt.figure(figsize=(12, 6))
200         sns.boxplot(y[y.columns[i]] - predictions[:, i])
201         plt.title(f'Residuals vs Predicted {output}')
202         plt.show()
203
204         # Scatter plot of residuals vs actual values
205         plt.figure(figsize=(12, 6))
206         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
207         plt.title(f'Residuals vs Actual {output}')
208         plt.xlabel('Actual Value')
209         plt.ylabel('Residual')
210         plt.show()
211
212         # Box plot of residuals vs actual values
213         plt.figure(figsize=(12, 6))
214         sns.boxplot(y[y.columns[i]] - predictions[:, i])
215         plt.title(f'Residuals vs Actual {output}')
216         plt.show()
217
218         # Scatter plot of residuals vs predicted values
219         plt.figure(figsize=(12, 6))
220         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
221         plt.title(f'Residuals vs Predicted {output}')
222         plt.xlabel('Predicted Value')
223         plt.ylabel('Residual')
224         plt.show()
225
226         # Box plot of residuals vs predicted values
227         plt.figure(figsize=(12, 6))
228         sns.boxplot(y[y.columns[i]] - predictions[:, i])
229         plt.title(f'Residuals vs Predicted {output}')
230         plt.show()
231
232         # Scatter plot of residuals vs actual values
233         plt.figure(figsize=(12, 6))
234         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
235         plt.title(f'Residuals vs Actual {output}')
236         plt.xlabel('Actual Value')
237         plt.ylabel('Residual')
238         plt.show()
239
240         # Box plot of residuals vs actual values
241         plt.figure(figsize=(12, 6))
242         sns.boxplot(y[y.columns[i]] - predictions[:, i])
243         plt.title(f'Residuals vs Actual {output}')
244         plt.show()
245
246         # Scatter plot of residuals vs predicted values
247         plt.figure(figsize=(12, 6))
248         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
249         plt.title(f'Residuals vs Predicted {output}')
250         plt.xlabel('Predicted Value')
251         plt.ylabel('Residual')
252         plt.show()
253
254         # Box plot of residuals vs predicted values
255         plt.figure(figsize=(12, 6))
256         sns.boxplot(y[y.columns[i]] - predictions[:, i])
257         plt.title(f'Residuals vs Predicted {output}')
258         plt.show()
259
260         # Scatter plot of residuals vs actual values
261         plt.figure(figsize=(12, 6))
262         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
263         plt.title(f'Residuals vs Actual {output}')
264         plt.xlabel('Actual Value')
265         plt.ylabel('Residual')
266         plt.show()
267
268         # Box plot of residuals vs actual values
269         plt.figure(figsize=(12, 6))
270         sns.boxplot(y[y.columns[i]] - predictions[:, i])
271         plt.title(f'Residuals vs Actual {output}')
272         plt.show()
273
274         # Scatter plot of residuals vs predicted values
275         plt.figure(figsize=(12, 6))
276         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
277         plt.title(f'Residuals vs Predicted {output}')
278         plt.xlabel('Predicted Value')
279         plt.ylabel('Residual')
280         plt.show()
281
282         # Box plot of residuals vs predicted values
283         plt.figure(figsize=(12, 6))
284         sns.boxplot(y[y.columns[i]] - predictions[:, i])
285         plt.title(f'Residuals vs Predicted {output}')
286         plt.show()
287
288         # Scatter plot of residuals vs actual values
289         plt.figure(figsize=(12, 6))
290         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
291         plt.title(f'Residuals vs Actual {output}')
292         plt.xlabel('Actual Value')
293         plt.ylabel('Residual')
294         plt.show()
295
296         # Box plot of residuals vs actual values
297         plt.figure(figsize=(12, 6))
298         sns.boxplot(y[y.columns[i]] - predictions[:, i])
299         plt.title(f'Residuals vs Actual {output}')
300         plt.show()
301
302         # Scatter plot of residuals vs predicted values
303         plt.figure(figsize=(12, 6))
304         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
305         plt.title(f'Residuals vs Predicted {output}')
306         plt.xlabel('Predicted Value')
307         plt.ylabel('Residual')
308         plt.show()
309
310         # Box plot of residuals vs predicted values
311         plt.figure(figsize=(12, 6))
312         sns.boxplot(y[y.columns[i]] - predictions[:, i])
313         plt.title(f'Residuals vs Predicted {output}')
314         plt.show()
315
316         # Scatter plot of residuals vs actual values
317         plt.figure(figsize=(12, 6))
318         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
319         plt.title(f'Residuals vs Actual {output}')
320         plt.xlabel('Actual Value')
321         plt.ylabel('Residual')
322         plt.show()
323
324         # Box plot of residuals vs actual values
325         plt.figure(figsize=(12, 6))
326         sns.boxplot(y[y.columns[i]] - predictions[:, i])
327         plt.title(f'Residuals vs Actual {output}')
328         plt.show()
329
330         # Scatter plot of residuals vs predicted values
331         plt.figure(figsize=(12, 6))
332         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
333         plt.title(f'Residuals vs Predicted {output}')
334         plt.xlabel('Predicted Value')
335         plt.ylabel('Residual')
336         plt.show()
337
338         # Box plot of residuals vs predicted values
339         plt.figure(figsize=(12, 6))
340         sns.boxplot(y[y.columns[i]] - predictions[:, i])
341         plt.title(f'Residuals vs Predicted {output}')
342         plt.show()
343
344         # Scatter plot of residuals vs actual values
345         plt.figure(figsize=(12, 6))
346         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
347         plt.title(f'Residuals vs Actual {output}')
348         plt.xlabel('Actual Value')
349         plt.ylabel('Residual')
350         plt.show()
351
352         # Box plot of residuals vs actual values
353         plt.figure(figsize=(12, 6))
354         sns.boxplot(y[y.columns[i]] - predictions[:, i])
355         plt.title(f'Residuals vs Actual {output}')
356         plt.show()
357
358         # Scatter plot of residuals vs predicted values
359         plt.figure(figsize=(12, 6))
360         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
361         plt.title(f'Residuals vs Predicted {output}')
362         plt.xlabel('Predicted Value')
363         plt.ylabel('Residual')
364         plt.show()
365
366         # Box plot of residuals vs predicted values
367         plt.figure(figsize=(12, 6))
368         sns.boxplot(y[y.columns[i]] - predictions[:, i])
369         plt.title(f'Residuals vs Predicted {output}')
370         plt.show()
371
372         # Scatter plot of residuals vs actual values
373         plt.figure(figsize=(12, 6))
374         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
375         plt.title(f'Residuals vs Actual {output}')
376         plt.xlabel('Actual Value')
377         plt.ylabel('Residual')
378         plt.show()
379
380         # Box plot of residuals vs actual values
381         plt.figure(figsize=(12, 6))
382         sns.boxplot(y[y.columns[i]] - predictions[:, i])
383         plt.title(f'Residuals vs Actual {output}')
384         plt.show()
385
386         # Scatter plot of residuals vs predicted values
387         plt.figure(figsize=(12, 6))
388         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
389         plt.title(f'Residuals vs Predicted {output}')
390         plt.xlabel('Predicted Value')
391         plt.ylabel('Residual')
392         plt.show()
393
394         # Box plot of residuals vs predicted values
395         plt.figure(figsize=(12, 6))
396         sns.boxplot(y[y.columns[i]] - predictions[:, i])
397         plt.title(f'Residuals vs Predicted {output}')
398         plt.show()
399
400         # Scatter plot of residuals vs actual values
401         plt.figure(figsize=(12, 6))
402         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
403         plt.title(f'Residuals vs Actual {output}')
404         plt.xlabel('Actual Value')
405         plt.ylabel('Residual')
406         plt.show()
407
408         # Box plot of residuals vs actual values
409         plt.figure(figsize=(12, 6))
410         sns.boxplot(y[y.columns[i]] - predictions[:, i])
411         plt.title(f'Residuals vs Actual {output}')
412         plt.show()
413
414         # Scatter plot of residuals vs predicted values
415         plt.figure(figsize=(12, 6))
416         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
417         plt.title(f'Residuals vs Predicted {output}')
418         plt.xlabel('Predicted Value')
419         plt.ylabel('Residual')
420         plt.show()
421
422         # Box plot of residuals vs predicted values
423         plt.figure(figsize=(12, 6))
424         sns.boxplot(y[y.columns[i]] - predictions[:, i])
425         plt.title(f'Residuals vs Predicted {output}')
426         plt.show()
427
428         # Scatter plot of residuals vs actual values
429         plt.figure(figsize=(12, 6))
430         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
431         plt.title(f'Residuals vs Actual {output}')
432         plt.xlabel('Actual Value')
433         plt.ylabel('Residual')
434         plt.show()
435
436         # Box plot of residuals vs actual values
437         plt.figure(figsize=(12, 6))
438         sns.boxplot(y[y.columns[i]] - predictions[:, i])
439         plt.title(f'Residuals vs Actual {output}')
440         plt.show()
441
442         # Scatter plot of residuals vs predicted values
443         plt.figure(figsize=(12, 6))
444         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
445         plt.title(f'Residuals vs Predicted {output}')
446         plt.xlabel('Predicted Value')
447         plt.ylabel('Residual')
448         plt.show()
449
450         # Box plot of residuals vs predicted values
451         plt.figure(figsize=(12, 6))
452         sns.boxplot(y[y.columns[i]] - predictions[:, i])
453         plt.title(f'Residuals vs Predicted {output}')
454         plt.show()
455
456         # Scatter plot of residuals vs actual values
457         plt.figure(figsize=(12, 6))
458         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
459         plt.title(f'Residuals vs Actual {output}')
460         plt.xlabel('Actual Value')
461         plt.ylabel('Residual')
462         plt.show()
463
464         # Box plot of residuals vs actual values
465         plt.figure(figsize=(12, 6))
466         sns.boxplot(y[y.columns[i]] - predictions[:, i])
467         plt.title(f'Residuals vs Actual {output}')
468         plt.show()
469
470         # Scatter plot of residuals vs predicted values
471         plt.figure(figsize=(12, 6))
472         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
473         plt.title(f'Residuals vs Predicted {output}')
474         plt.xlabel('Predicted Value')
475         plt.ylabel('Residual')
476         plt.show()
477
478         # Box plot of residuals vs predicted values
479         plt.figure(figsize=(12, 6))
480         sns.boxplot(y[y.columns[i]] - predictions[:, i])
481         plt.title(f'Residuals vs Predicted {output}')
482         plt.show()
483
484         # Scatter plot of residuals vs actual values
485         plt.figure(figsize=(12, 6))
486         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
487         plt.title(f'Residuals vs Actual {output}')
488         plt.xlabel('Actual Value')
489         plt.ylabel('Residual')
490         plt.show()
491
492         # Box plot of residuals vs actual values
493         plt.figure(figsize=(12, 6))
494         sns.boxplot(y[y.columns[i]] - predictions[:, i])
495         plt.title(f'Residuals vs Actual {output}')
496         plt.show()
497
498         # Scatter plot of residuals vs predicted values
499         plt.figure(figsize=(12, 6))
500         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
501         plt.title(f'Residuals vs Predicted {output}')
502         plt.xlabel('Predicted Value')
503         plt.ylabel('Residual')
504         plt.show()
505
506         # Box plot of residuals vs predicted values
507         plt.figure(figsize=(12, 6))
508         sns.boxplot(y[y.columns[i]] - predictions[:, i])
509         plt.title(f'Residuals vs Predicted {output}')
510         plt.show()
511
512         # Scatter plot of residuals vs actual values
513         plt.figure(figsize=(12, 6))
514         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
515         plt.title(f'Residuals vs Actual {output}')
516         plt.xlabel('Actual Value')
517         plt.ylabel('Residual')
518         plt.show()
519
520         # Box plot of residuals vs actual values
521         plt.figure(figsize=(12, 6))
522         sns.boxplot(y[y.columns[i]] - predictions[:, i])
523         plt.title(f'Residuals vs Actual {output}')
524         plt.show()
525
526         # Scatter plot of residuals vs predicted values
527         plt.figure(figsize=(12, 6))
528         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
529         plt.title(f'Residuals vs Predicted {output}')
530         plt.xlabel('Predicted Value')
531         plt.ylabel('Residual')
532         plt.show()
533
534         # Box plot of residuals vs predicted values
535         plt.figure(figsize=(12, 6))
536         sns.boxplot(y[y.columns[i]] - predictions[:, i])
537         plt.title(f'Residuals vs Predicted {output}')
538         plt.show()
539
540         # Scatter plot of residuals vs actual values
541         plt.figure(figsize=(12, 6))
542         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
543         plt.title(f'Residuals vs Actual {output}')
544         plt.xlabel('Actual Value')
545         plt.ylabel('Residual')
546         plt.show()
547
548         # Box plot of residuals vs actual values
549         plt.figure(figsize=(12, 6))
550         sns.boxplot(y[y.columns[i]] - predictions[:, i])
551         plt.title(f'Residuals vs Actual {output}')
552         plt.show()
553
554         # Scatter plot of residuals vs predicted values
555         plt.figure(figsize=(12, 6))
556         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
557         plt.title(f'Residuals vs Predicted {output}')
558         plt.xlabel('Predicted Value')
559         plt.ylabel('Residual')
560         plt.show()
561
562         # Box plot of residuals vs predicted values
563         plt.figure(figsize=(12, 6))
564         sns.boxplot(y[y.columns[i]] - predictions[:, i])
565         plt.title(f'Residuals vs Predicted {output}')
566         plt.show()
567
568         # Scatter plot of residuals vs actual values
569         plt.figure(figsize=(12, 6))
570         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
571         plt.title(f'Residuals vs Actual {output}')
572         plt.xlabel('Actual Value')
573         plt.ylabel('Residual')
574         plt.show()
575
576         # Box plot of residuals vs actual values
577         plt.figure(figsize=(12, 6))
578         sns.boxplot(y[y.columns[i]] - predictions[:, i])
579         plt.title(f'Residuals vs Actual {output}')
580         plt.show()
581
582         # Scatter plot of residuals vs predicted values
583         plt.figure(figsize=(12, 6))
584         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
585         plt.title(f'Residuals vs Predicted {output}')
586         plt.xlabel('Predicted Value')
587         plt.ylabel('Residual')
588         plt.show()
589
590         # Box plot of residuals vs predicted values
591         plt.figure(figsize=(12, 6))
592         sns.boxplot(y[y.columns[i]] - predictions[:, i])
593         plt.title(f'Residuals vs Predicted {output}')
594         plt.show()
595
596         # Scatter plot of residuals vs actual values
597         plt.figure(figsize=(12, 6))
598         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
599         plt.title(f'Residuals vs Actual {output}')
600         plt.xlabel('Actual Value')
601         plt.ylabel('Residual')
602         plt.show()
603
604         # Box plot of residuals vs actual values
605         plt.figure(figsize=(12, 6))
606         sns.boxplot(y[y.columns[i]] - predictions[:, i])
607         plt.title(f'Residuals vs Actual {output}')
608         plt.show()
609
610         # Scatter plot of residuals vs predicted values
611         plt.figure(figsize=(12, 6))
612         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
613         plt.title(f'Residuals vs Predicted {output}')
614         plt.xlabel('Predicted Value')
615         plt.ylabel('Residual')
616         plt.show()
617
618         # Box plot of residuals vs predicted values
619         plt.figure(figsize=(12, 6))
620         sns.boxplot(y[y.columns[i]] - predictions[:, i])
621         plt.title(f'Residuals vs Predicted {output}')
622         plt.show()
623
624         # Scatter plot of residuals vs actual values
625         plt.figure(figsize=(12, 6))
626         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
627         plt.title(f'Residuals vs Actual {output}')
628         plt.xlabel('Actual Value')
629         plt.ylabel('Residual')
630         plt.show()
631
632         # Box plot of residuals vs actual values
633         plt.figure(figsize=(12, 6))
634         sns.boxplot(y[y.columns[i]] - predictions[:, i])
635         plt.title(f'Residuals vs Actual {output}')
636         plt.show()
637
638         # Scatter plot of residuals vs predicted values
639         plt.figure(figsize=(12, 6))
640         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
641         plt.title(f'Residuals vs Predicted {output}')
642         plt.xlabel('Predicted Value')
643         plt.ylabel('Residual')
644         plt.show()
645
646         # Box plot of residuals vs predicted values
647         plt.figure(figsize=(12, 6))
648         sns.boxplot(y[y.columns[i]] - predictions[:, i])
649         plt.title(f'Residuals vs Predicted {output}')
650         plt.show()
651
652         # Scatter plot of residuals vs actual values
653         plt.figure(figsize=(12, 6))
654         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
655         plt.title(f'Residuals vs Actual {output}')
656         plt.xlabel('Actual Value')
657         plt.ylabel('Residual')
658         plt.show()
659
660         # Box plot of residuals vs actual values
661         plt.figure(figsize=(12, 6))
662         sns.boxplot(y[y.columns[i]] - predictions[:, i])
663         plt.title(f'Residuals vs Actual {output}')
664         plt.show()
665
666         # Scatter plot of residuals vs predicted values
667         plt.figure(figsize=(12, 6))
668         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
669         plt.title(f'Residuals vs Predicted {output}')
670         plt.xlabel('Predicted Value')
671         plt.ylabel('Residual')
672         plt.show()
673
674         # Box plot of residuals vs predicted values
675         plt.figure(figsize=(12, 6))
676         sns.boxplot(y[y.columns[i]] - predictions[:, i])
677         plt.title(f'Residuals vs Predicted {output}')
678         plt.show()
679
680         # Scatter plot of residuals vs actual values
681         plt.figure(figsize=(12, 6))
682         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
683         plt.title(f'Residuals vs Actual {output}')
684         plt.xlabel('Actual Value')
685         plt.ylabel('Residual')
686         plt.show()
687
688         # Box plot of residuals vs actual values
689         plt.figure(figsize=(12, 6))
690         sns.boxplot(y[y.columns[i]] - predictions[:, i])
691         plt.title(f'Residuals vs Actual {output}')
692         plt.show()
693
694         # Scatter plot of residuals vs predicted values
695         plt.figure(figsize=(12, 6))
696         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
697         plt.title(f'Residuals vs Predicted {output}')
698         plt.xlabel('Predicted Value')
699         plt.ylabel('Residual')
700         plt.show()
701
702         # Box plot of residuals vs predicted values
703         plt.figure(figsize=(12, 6))
704         sns.boxplot(y[y.columns[i]] - predictions[:, i])
705         plt.title(f'Residuals vs Predicted {output}')
706         plt.show()
707
708         # Scatter plot of residuals vs actual values
709         plt.figure(figsize=(12, 6))
710         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
711         plt.title(f'Residuals vs Actual {output}')
712         plt.xlabel('Actual Value')
713         plt.ylabel('Residual')
714         plt.show()
715
716         # Box plot of residuals vs actual values
717         plt.figure(figsize=(12, 6))
718         sns.boxplot(y[y.columns[i]] - predictions[:, i])
719         plt.title(f'Residuals vs Actual {output}')
720         plt.show()
721
722         # Scatter plot of residuals vs predicted values
723         plt.figure(figsize=(12, 6))
724         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
725         plt.title(f'Residuals vs Predicted {output}')
726         plt.xlabel('Predicted Value')
727         plt.ylabel('Residual')
728         plt.show()
729
730         # Box plot of residuals vs predicted values
731         plt.figure(figsize=(12, 6))
732         sns.boxplot(y[y.columns[i]] - predictions[:, i])
733         plt.title(f'Residuals vs Predicted {output}')
734         plt.show()
735
736         # Scatter plot of residuals vs actual values
737         plt.figure(figsize=(12, 6))
738         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
739         plt.title(f'Residuals vs Actual {output}')
740         plt.xlabel('Actual Value')
741         plt.ylabel('Residual')
742         plt.show()
743
744         # Box plot of residuals vs actual values
745         plt.figure(figsize=(12, 6))
746         sns.boxplot(y[y.columns[i]] - predictions[:, i])
747         plt.title(f'Residuals vs Actual {output}')
748         plt.show()
749
750         # Scatter plot of residuals vs predicted values
751         plt.figure(figsize=(12, 6))
752         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
753         plt.title(f'Residuals vs Predicted {output}')
754         plt.xlabel('Predicted Value')
755         plt.ylabel('Residual')
756         plt.show()
757
758         # Box plot of residuals vs predicted values
759         plt.figure(figsize=(12, 6))
760         sns.boxplot(y[y.columns[i]] - predictions[:, i])
761         plt.title(f'Residuals vs Predicted {output}')
762         plt.show()
763
764         # Scatter plot of residuals vs actual values
765         plt.figure(figsize=(12, 6))
766         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
767         plt.title(f'Residuals vs Actual {output}')
768         plt.xlabel('Actual Value')
769         plt.ylabel('Residual')
770         plt.show()
771
772         # Box plot of residuals vs actual values
773         plt.figure(figsize=(12, 6))
774         sns.boxplot(y[y.columns[i]] - predictions[:, i])
775         plt.title(f'Residuals vs Actual {output}')
776         plt.show()
777
778         # Scatter plot of residuals vs predicted values
779         plt.figure(figsize=(12, 6))
780         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
781         plt.title(f'Residuals vs Predicted {output}')
782         plt.xlabel('Predicted Value')
783         plt.ylabel('Residual')
784         plt.show()
785
786         # Box plot of residuals vs predicted values
787         plt.figure(figsize=(12, 6))
788         sns.boxplot(y[y.columns[i]] - predictions[:, i])
789         plt.title(f'Residuals vs Predicted {output}')
790         plt.show()
791
792         # Scatter plot of residuals vs actual values
793         plt.figure(figsize=(12, 6))
794         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
795         plt.title(f'Residuals vs Actual {output}')
796         plt.xlabel('Actual Value')
797         plt.ylabel('Residual')
798         plt.show()
799
800         # Box plot of residuals vs actual values
801         plt.figure(figsize=(12, 6))
802         sns.boxplot(y[y.columns[i]] - predictions[:, i])
803         plt.title(f'Residuals vs Actual {output}')
804         plt.show()
805
806         # Scatter plot of residuals vs predicted values
807         plt.figure(figsize=(12, 6))
808         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
809         plt.title(f'Residuals vs Predicted {output}')
810         plt.xlabel('Predicted Value')
811         plt.ylabel('Residual')
812         plt.show()
813
814         # Box plot of residuals vs predicted values
815         plt.figure(figsize=(12, 6))
816         sns.boxplot(y[y.columns[i]] - predictions[:, i])
817         plt.title(f'Residuals vs Predicted {output}')
818         plt.show()
819
820         # Scatter plot of residuals vs actual values
821         plt.figure(figsize=(12, 6))
822         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
823         plt.title(f'Residuals vs Actual {output}')
824         plt.xlabel('Actual Value')
825         plt.ylabel('Residual')
826         plt.show()
827
828         # Box plot of residuals vs actual values
829         plt.figure(figsize=(12, 6))
830         sns.boxplot(y[y.columns[i]] - predictions[:, i])
831         plt.title(f'Residuals vs Actual {output}')
832         plt.show()
833
834         # Scatter plot of residuals vs predicted values
835         plt.figure(figsize=(12, 6))
836         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
837         plt.title(f'Residuals vs Predicted {output}')
838         plt.xlabel('Predicted Value')
839         plt.ylabel('Residual')
840         plt.show()
841
842         # Box plot of residuals vs predicted values
843         plt.figure(figsize=(12, 6))
844         sns.boxplot(y[y.columns[i]] - predictions[:, i])
845         plt.title(f'Residuals vs Predicted {output}')
846         plt.show()
847
848         # Scatter plot of residuals vs actual values
849         plt.figure(figsize=(12, 6))
850         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
851         plt.title(f'Residuals vs Actual {output}')
852         plt.xlabel('Actual Value')
853         plt.ylabel('Residual')
854         plt.show()
855
856         # Box plot of residuals vs actual values
857         plt.figure(figsize=(12, 6))
858         sns.boxplot(y[y.columns[i]] - predictions[:, i])
859         plt.title(f'Residuals vs Actual {output}')
860         plt.show()
861
862         # Scatter plot of residuals vs predicted values
863         plt.figure(figsize=(12, 6))
864         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
865         plt.title(f'Residuals vs Predicted {output}')
866         plt.xlabel('Predicted Value')
867         plt.ylabel('Residual')
868         plt.show()
869
870         # Box plot of residuals vs predicted values
871         plt.figure(figsize=(12, 6))
872         sns.boxplot(y[y.columns[i]] - predictions[:, i])
873         plt.title(f'Residuals vs Predicted {output}')
874         plt.show()
875
876         # Scatter plot of residuals vs actual values
877         plt.figure(figsize=(12, 6))
878         plt.scatter(y[y.columns[i]], y[y.columns[i]] - predictions[:, i])
879         plt.title(f'Residuals vs Actual {output}')
880         plt.xlabel('Actual Value')
881         plt.ylabel('Residual')
882         plt.show()
883
884         # Box plot of residuals vs actual values
885         plt.figure(figsize=(12, 6))
886         sns.boxplot(y[y.columns[i]] - predictions[:, i])
887         plt.title(f'Residuals vs Actual {output}')
888         plt.show()
889
890         # Scatter plot of residuals vs predicted values
891         plt.figure(figsize=(12, 6))
892         plt.scatter(predictions[:, i], y[y.columns[i]] - predictions[:, i])
893         plt.title(f'Residuals vs Predicted {output}')
894         plt.xlabel('Predicted Value')
895         plt.ylabel('Residual')
896         plt.show()
897
898         # Box plot of residuals vs predicted values
899         plt.figure(figsize=(12, 6))
900         sns.boxplot(y[y.columns[i]] - predictions[:, i])
901         plt.title(f'Residuals vs Predicted {output}')
902         plt.show()
903
904         # Scatter plot of residuals vs actual values
905         plt.figure(figsize=(12, 6))
906         plt.scatter(y[y.columns[i]], y[y.columns[i]]
```



```

1 # Data visualization
2 import matplotlib.pyplot as plt
3
4 # Histogram of Actual vs Predicted values
5 plt.subplot(2, 3, 1)
6 plt.hist(testoutput, bins=30, color='blue', alpha=0.7, label='Actual Values')
7 plt.title('Histogram of Actual (output)')
8 plt.xlabel('Output')
9 plt.ylabel('Frequency')
10 plt.legend()
11
12 # Histogram for predicted values
13 plt.subplot(2, 3, 2)
14 plt.hist(predictions[:, 1], bins=30, color='orange', alpha=0.7, label='Predicted Values')
15 plt.title('Histogram of Predicted (output)')
16 plt.xlabel('Output')
17 plt.ylabel('Frequency')
18 plt.legend()
19
20 # Scatter plot of actual vs predicted values
21 plt.subplot(2, 3, 3)
22 plt.scatter(testoutput, predictions[:, 1], color='green', alpha=0.5)
23 plt.title('Scatter of Actual (output) vs Predicted (output)')
24 plt.xlabel('Actual (output)')
25 plt.ylabel('Predicted (output)')
26 plt.show()
27
28 # Bar chart for individual output metrics
29 fig = plt.subplots(figsize=(10, 6))
30
31 bar_width = 0.35
32 index = np.arange(len(columns))
33 opacity = 0.8
34
35 for i, output in enumerate(columns):
36     ax = fig.add_subplot(1, len(columns), i+1)
37     r2_scores = r2_scores[:, i]
38     ax.bar(index + i * bar_width, r2_scores, bar_width, alpha=opacity, label=f'R2 - {output}')
39
40     ax.set_xlabel(f'{output} Parameters')
41     ax.set_ylabel('R2 Score')
42     ax.set_title(f'Individual R2 scores for {output} Parameters')
43     ax.set_xticks(index + len(columns) * bar_width / 2 - (len(accuracy_levels) - 1) / 2)
44     ax.set_xticklabels(columns)
45     ax.legend()
46
47 plt.show()
48
49 # Loss curve during training
50 plt.plot(history.history['loss'], label='Training Loss')
51 plt.plot(history.history['val_loss'], label='Validation Loss')
52 plt.xlabel('Epochs')
53 plt.ylabel('Mean Squared Error')
54 plt.title('Training and Validation Loss Curve')
55 plt.legend()
56 plt.show()
57
58

```

Data Preprocessing:

1. Data Loading: The code starts by loading a dataset from a CSV file using the Pandas library. The dataset presumably contains information related to applied loads, moments, lengths, and design parameters of concrete piles.
2. Feature Selection: The input features (`X`) and output targets (`y`) are then defined. Input features include load, moments in x and y directions, and pile length. Output targets consist of diameter of pile, area of steel, and pitch.
3. Data Splitting: The dataset is split into training, testing, and cross-validation sets using the `train_test_split` function from scikit-learn.
4. Standardization: Both input features and output targets are standardized using `StandardScaler` to ensure that each feature contributes proportionally to the model training.

Neural Network Architecture:

1. Model Architecture: The neural network is constructed using Keras Sequential API. It consists of an input layer, a hidden layer with 32 neurons and ReLU activation, a dropout layer for regularization, an

additional hidden layer with 64 neurons, another dropout layer, a third hidden layer with 16 neurons, and a final output layer.

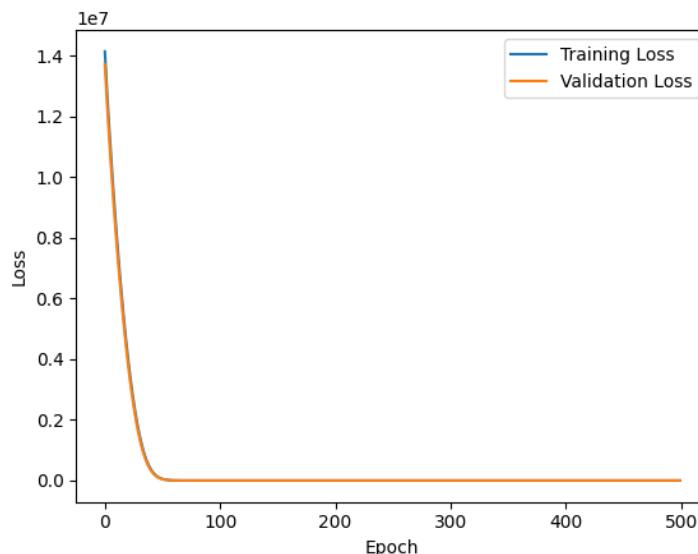
2. Compilation: The model is compiled using the Adam optimizer and mean squared error loss function, suitable for regression tasks.

Model Training:

The model is trained using the training set and validated on the cross-validation set for 500 epochs with a batch size of 32. Training history is stored in the `history` variable.

Model Evaluation:

1. Prediction: The trained model is used to predict the output targets on the test set.
2. Inverse Transformation: Predictions are inverse-transformed to the original scale using the `StandardScaler` to facilitate result interpretation.
3. Evaluation Metrics: The code assesses the model's performance using various metrics, including R2 scores, explained variances, and accuracy levels within specified tolerances.
4. Visualization: Visualizations include pie charts depicting the distribution of predictions within tolerance, histograms comparing actual and predicted values, scatter plots for each output parameter, and a bar chart illustrating individual R2 scores.



Tolerance Level: $\pm 10.0\%$

Overall Accuracy: 99.05%

Individual R2 Scores:

Diameter of Pile: 0.9558

Area of Steel (mm^2): 0.9556

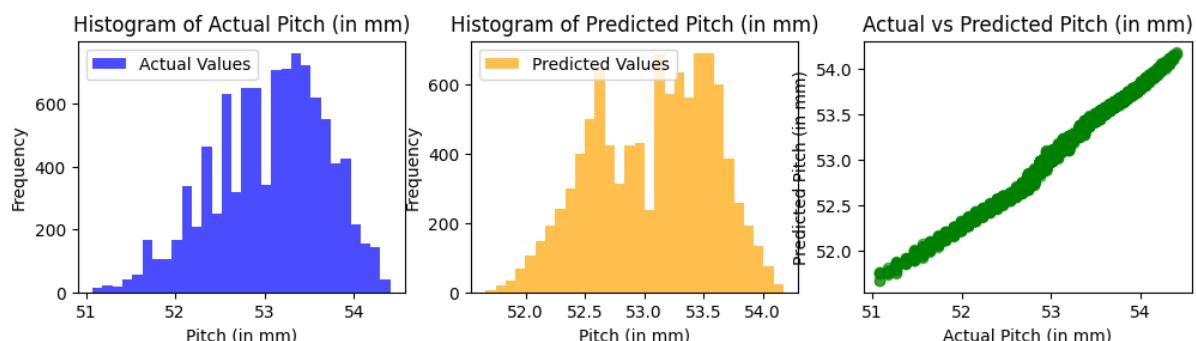
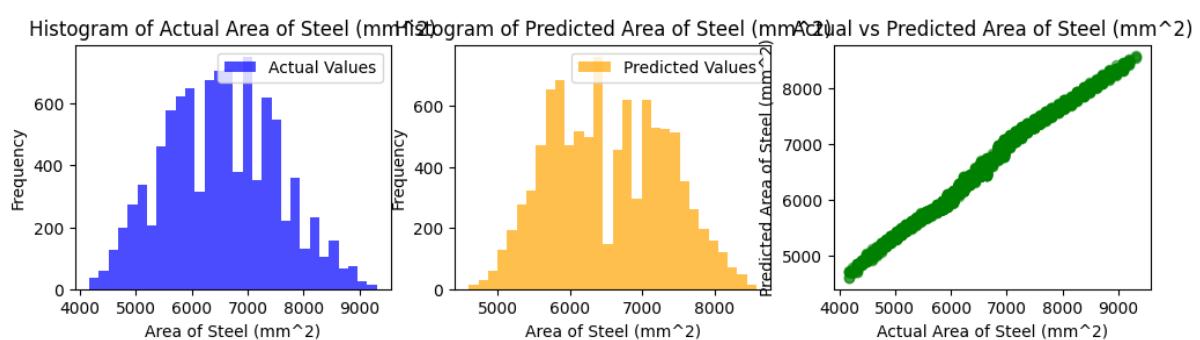
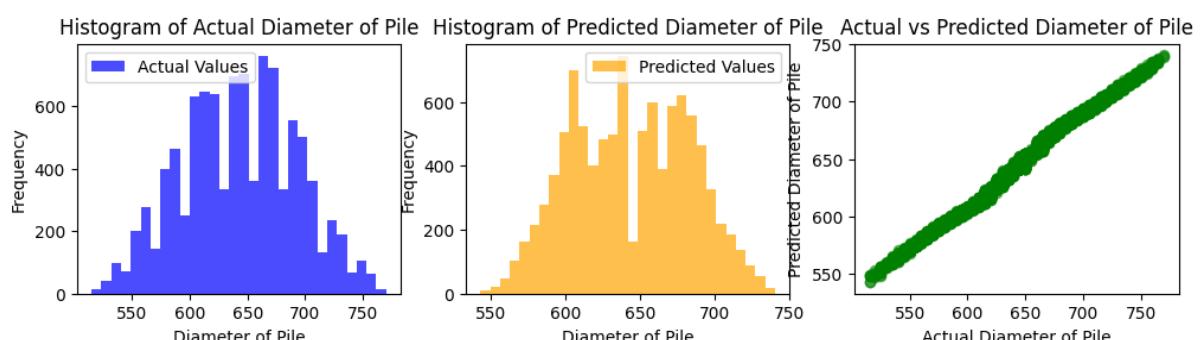
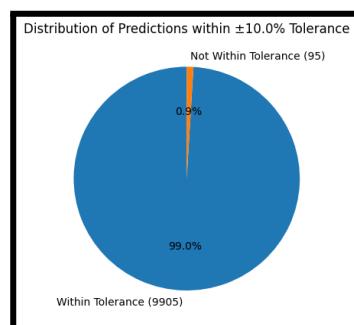
Pitch (in mm): 0.9524

Individual Explained Variances:

Diameter of Pile: 0.9560

Area of Steel (mm^2): 0.9557

Pitch (in mm): 0.9532



Tolerance Level: $\pm 8.0\%$

Overall Accuracy: 96.14%

Individual R2 Scores:

Diameter of Pile: 0.9558

Area of Steel (mm^2): 0.9556

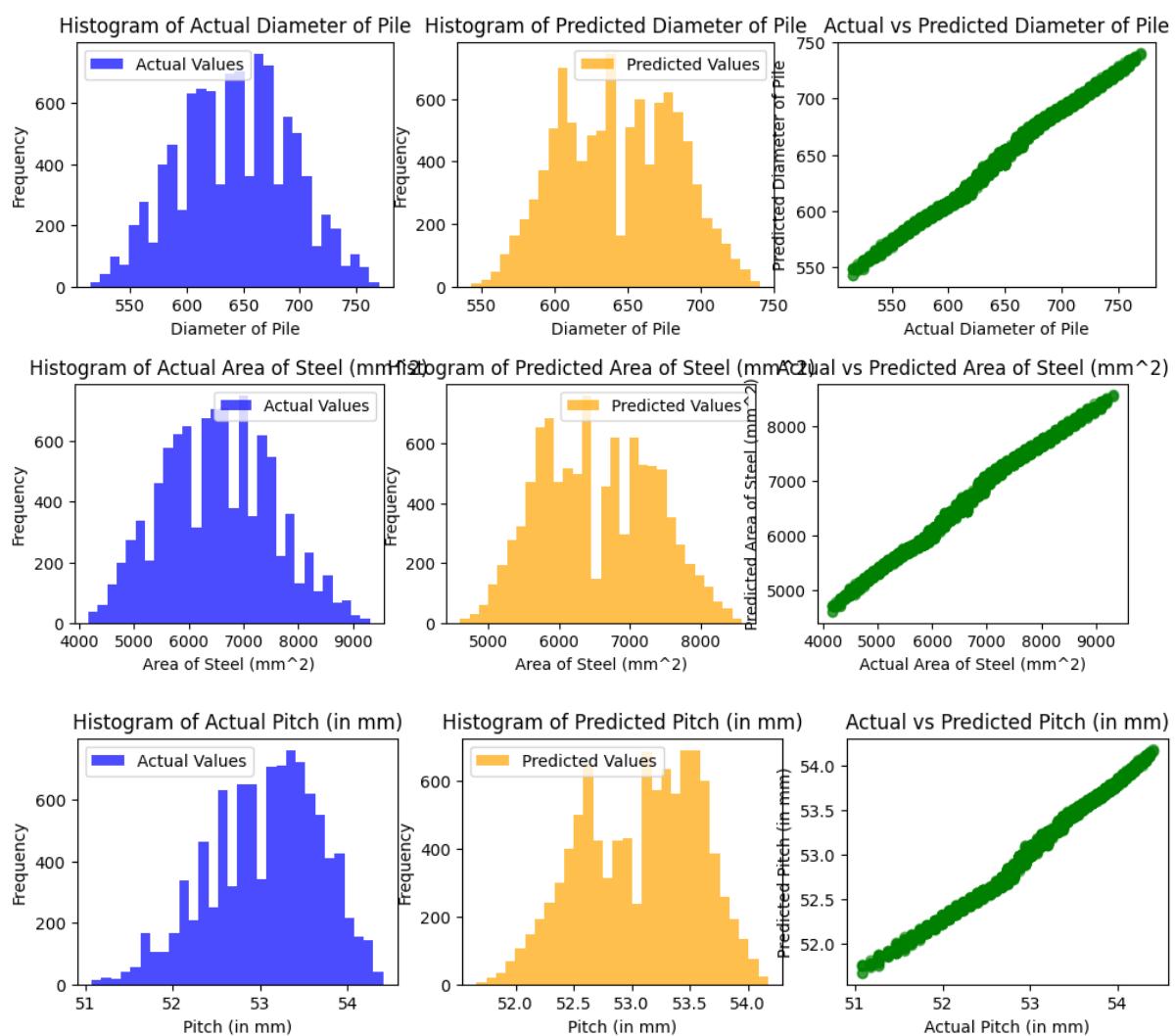
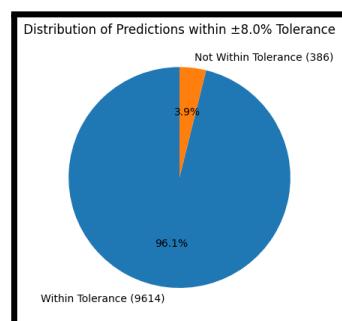
Pitch (in mm): 0.9524

Individual Explained Variances:

Diameter of Pile: 0.9560

Area of Steel (mm^2): 0.9557

Pitch (in mm): 0.9532



Tolerance Level: $\pm 5.0\%$

Overall Accuracy: 83.08%

Individual R2 Scores:

Diameter of Pile: 0.9558

Area of Steel (mm^2): 0.9556

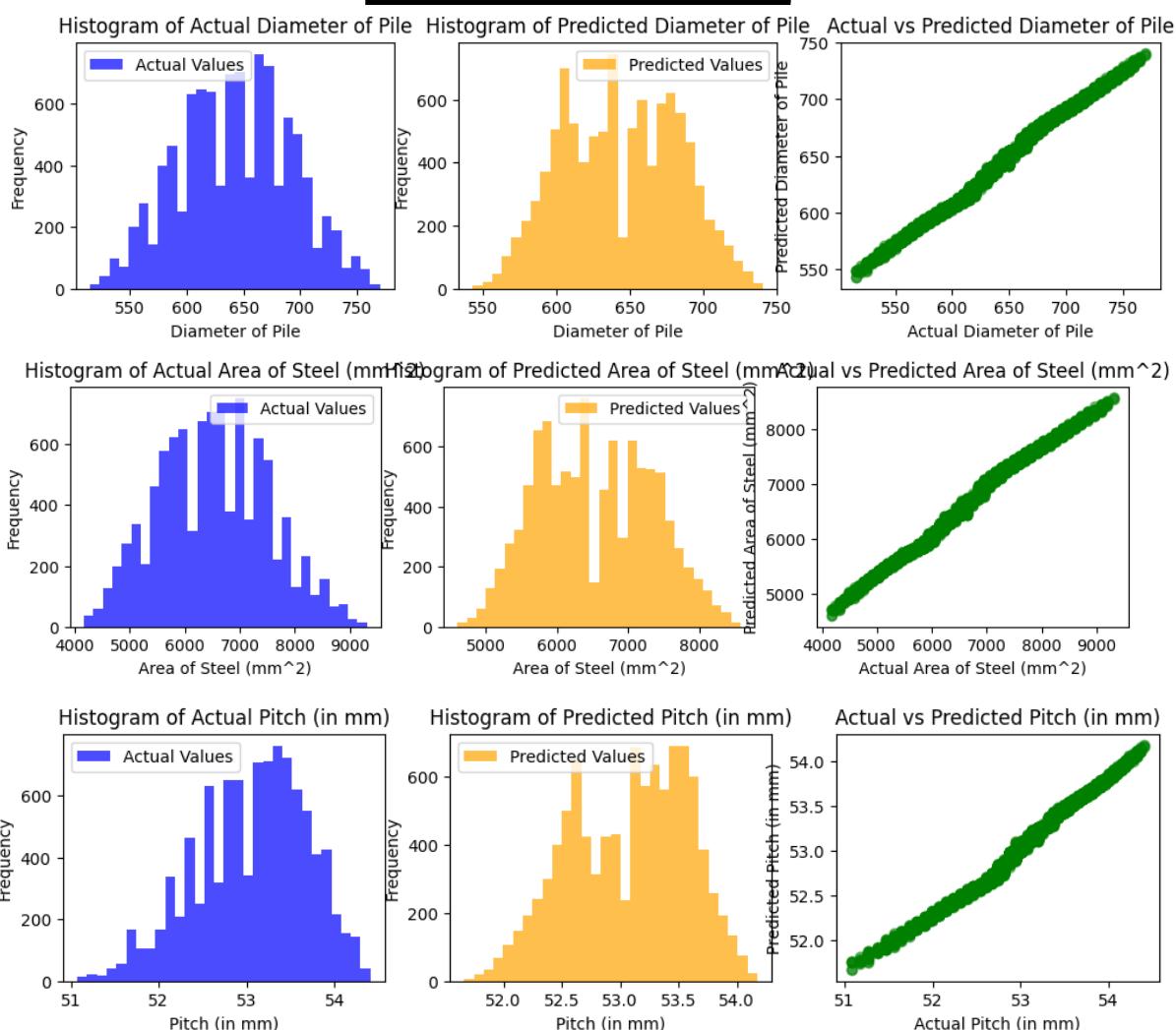
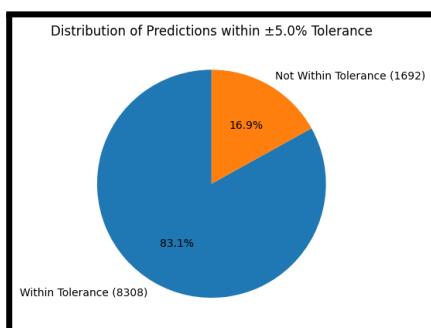
Pitch (in mm): 0.9524

Individual Explained Variances:

Diameter of Pile: 0.9560

Area of Steel (mm^2): 0.9557

Pitch (in mm): 0.9532



Overall Accuracy: 60.81%

Individual R2 Scores:

Diameter of Pile: 0.9558

Area of Steel (mm^2): 0.9556

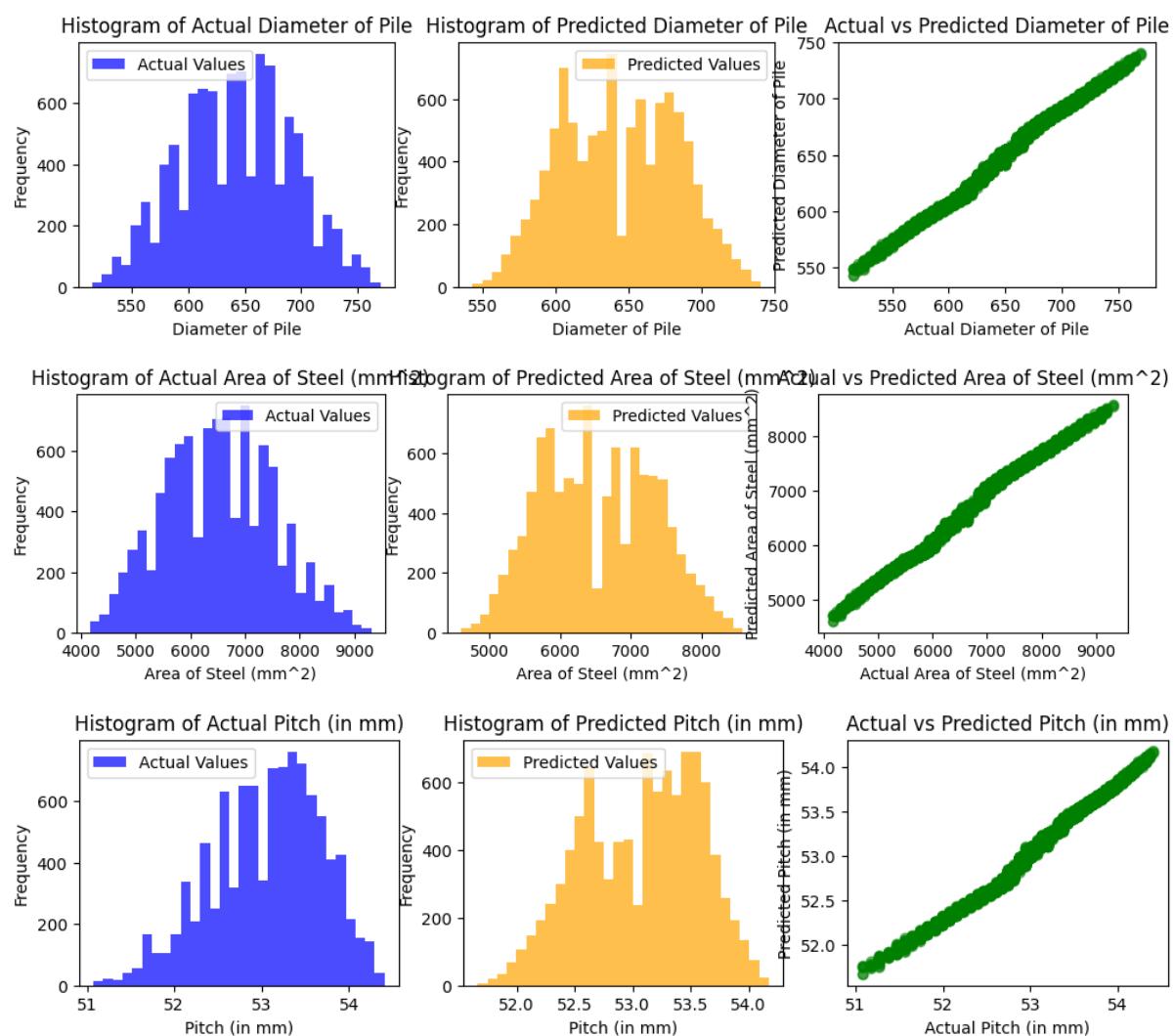
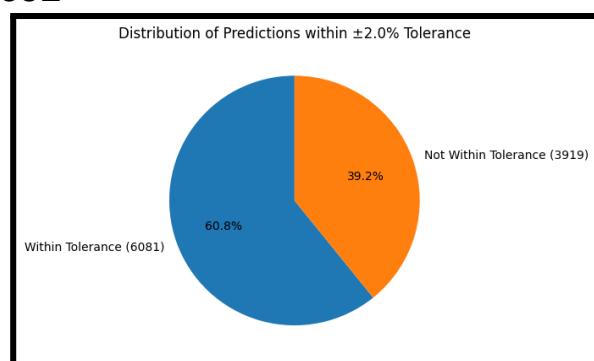
Pitch (in mm): 0.9524

Individual Explained Variances:

Diameter of Pile: 0.9560

Area of Steel (mm^2): 0.9557

Pitch (in mm): 0.9532



IV. RESULTS AND DISCUSSIONS

- ❖ In this study, we have conducted a design optimization of pile foundation for concrete piles. The design was carried out using a combination of manual calculations and computer-based design tools. The design optimization process involved varying the axial load, moments, pile length to achieve the most economical design while meeting the required load capacity.
- ❖ Manual calculation is an important part of the design process for pile foundation. The use of manual calculations also allowed for a deeper understanding of the design process and the factors that influence the design of the pile foundation.
- ❖ We have successfully created the dataset of two lakh test cases for design of pile foundation which is used to train our ML model.
- ❖ The accuracy with +10% is very good but it decreases as we decrease the tolerance to +2%.
- ❖ In conclusion, the design optimization process used in this study was effective in achieving a more economical pile foundation design while maintaining the required load capacity. The results of this study can be useful in informing the design of pile foundations for various types of structures.

V. CONCLUSIONS AND FUTURE WORK

- ❖ The implemented machine learning (ML) model demonstrates its efficacy in optimizing the design of concrete pile foundations. The utilization of an Artificial Neural Network (ANN) facilitates the prediction of critical design parameters, including diameter, steel area, and pitch, showcasing the model's ability to capture complex relationships within the dataset.
- ❖ The evaluation metrics, such as R² scores, explained variances, and accuracy levels within specified tolerances, provide a comprehensive assessment of the model's accuracy and reliability. The model's capacity to predict within defined tolerances underscores its practical utility in real-world scenarios.
- ❖ The visualizations accompanying the analysis, including pie charts, histograms, scatter plots, and bar charts, offer valuable insights into the distribution of predictions, model behavior, and individual output metrics. These visual aids enhance the interpretability of the model's performance.
- ❖ The incorporation of dropout layers in the neural network contributes to model robustness by preventing overfitting. This ensures that the model generalizes well to unseen data, a critical aspect for its application in diverse construction projects.
- ❖ The loss curves during training indicate the convergence of the model, suggesting that the optimization process effectively minimizes the mean squared error. This convergence enhances confidence in the stability and reliability of the trained model.
- ❖ To enhance the model's generalization capabilities, future work could focus on expanding the dataset. Including a more extensive range of scenarios, soil types, and structural configurations can improve the model's adaptability to a broader spectrum of real-world conditions.
- ❖ Further exploration of hyperparameter tuning may optimize the model's architecture and training parameters. This includes adjusting the number of neurons in each layer, dropout rates, and learning rates to identify configurations that improve predictive accuracy.
- ❖ The integration of additional geotechnical data, such as soil properties and subsurface conditions, can refine the model's

accuracy. Incorporating a more detailed understanding of the foundation's interaction with the underlying soil can lead to more accurate predictions.

- ❖ Experimenting with ensemble methods, such as combining predictions from multiple models, could enhance the overall predictive power. Ensemble techniques often mitigate the weaknesses of individual models and improve overall performance.
- ❖ Implementing a system for real-time monitoring and feedback during the construction phase can provide an opportunity for model refinement. The model could be continuously updated based on the feedback obtained from the actual performance of constructed foundations.
- ❖ Integrating the ML model with existing design codes and standards can create a powerful tool for engineers. This involves ensuring that the model's predictions align with industry standards and can be seamlessly integrated into the existing design processes.
- ❖ Exploring the applicability of the model across various construction projects and foundation types can broaden its impact. This includes adapting the model for different types of piles, foundation materials, and structural configurations.

In summary, the presented ML model serves as a promising tool for design optimization of pile foundations. Future efforts can leverage these insights to refine the model's capabilities, expand its applicability, and enhance its integration into the broader landscape of civil engineering practices.

VI. REFERENCES

1. Code

- ❖ IS Code 456:2000 (Plain and Reinforcement Concrete), IS Code 2911(Part-1/Sec-4):2010 (Design and construction of pile foundations) and SP 34:1987(Handbook on Concrete Reinforcement and Detailing).

2. Journal

- ❖ https://nitsri.ac.in/Department/Civil%20Engineering/CGE-202_7_Pile_Foundation_Design_A_Student_Guide.pdf
- ❖ <https://www.youtube.com/watch?v=hzcyUbwgqqQ>
- ❖ https://onlinepubs.trb.org/Onlinepubs/nchrp/nchrp_syn_42.pdf

3. Books

- ❖ Das, B.M. (2010). Principles of Foundation Engineering. Cengage Learning, Boston.
- ❖ Fellenius, B.H. (1984). "The design of pile foundations." Geotechnique, 34(4), 449-489.
- ❖ N. Krishna Raju(2008) "Reinforced Concrete Design".