# **CHAPTER THREE**

# **3.0 METHODOLOGY**

## **3.1 GENERAL.**

The research methods used in this study are covered in detail in this chapter, with a strong emphasis on laboratory testing as the primary information source. These methods include the design of the experiment, the preparation and testing of the concrete samples, and the analysis of the test results. The data obtained from these tests is then used to train and validate the predictive model. This comprehensive approach ensures the reliability and validity of the findings, providing a solid foundation for future research in this field.

## **3.2 RESEARCH DESIGN.**

The research design of this study is a systematic and comprehensive approach to developing a tensile predictive model for concrete. It begins with data collection and processing, followed by feature selection, model development, evaluation, and training. The model is then optimized using a specific loss function, further refined, and saved for future use.  This design ensures that each phase contributes to the overall success of the project, providing a solid foundation for future research in this field.

## **3.3 MATERIAL SOURCING AND PREPARATION.**

### **3.2.1 Material Sourcing**

The materials used in this study were carefully sourced to ensure the quality and reliability of the results. The primary material used was cement that meet the EN 197-1 requirements, and three different types were utilized:

1. **OPC (Ordinary Portland Cement) 32.5 N:** This type of cement was sourced from the Dedan Kimathi Civil Engineering Lab. OPC is known for its general use in construction when there is no exposure to sulphates in the soil or groundwater. The ‘32.5 N’ refers to the strength of the cement, which means it can withstand a pressure of 32.5 Newton per square millimeter after 28 days of curing.
2. **KP Silver CEM II/B-P/ 32.N:** This cement was purchased from a hardware store in Nyeri town. It is a type of Portland Pozzolana Cement known for properties such as low heat of hydration, which makes it suitable for mass concreting. It also reduces the leaching of calcium hydroxide, thereby increasing the longevity and durability of the concrete structures.
3. **White Portland Cement CEM 52.5N:** Also sourced from a hardware store in Nyeri town, this cement is similar to ordinary gray Portland cement in all aspects except for its high degree of whiteness. The ‘52.5 N’ indicates that it has a high strength class, making it suitable for use in architectural and decorative concrete.

These cements were chosen for their unique properties, which contribute to the tensile strength of the concrete. By using these different types of cement, we aim to develop a more accurate and robust tensile predictive model for concrete.

### **3.2.2 Aggregate Sourcing**

In this study, we used three types of aggregate sizes, which were sieved in the laboratory. The aggregates used met EN 12620 requirements. The aggregates were sourced from the school’s laboratory, ensuring their quality and consistency. The aggregates were sieved to obtain three distinct sizes: 5mm, 10mm, and 20mm. These sizes were chosen to provide a range of granularities, which can significantly affect the concrete’s workability, strength, and durability.

In addition to the coarse aggregates, we also used fine aggregate, specifically sand. The sand was measured in specific proportions for the concrete mix, as shown in the mix proportion table. Fine aggregate, like sand, fills the spaces between the coarse aggregate and binds with cement to form a homogenous mixture. This contributes to the overall strength and durability of the concrete.

The careful sourcing and selection of these materials are crucial in our study as they directly impact the tensile strength of the concrete, which our predictive model aims to forecast. By using these different types of aggregates and sand, we aim to develop a more accurate and robust tensile predictive model for concrete. This approach also allows us to understand the role of different aggregate sizes and types in the concrete’s tensile strength.

### **3.2.3 Water**

Water was obtained from Dedan Kimathi Civil Engineering laboratory premises. Tap water was used as both mixing water and curing water. Following EN 1008. For Concrete to function at its best, the water needs to be pure, consistently high-quality, and at the proper temperature. There is no chance that the cement hydration process will be compromised, which could have a detrimental impact on the concrete's durability and strength. It is also essential to have clean, drinkable water that is devoid of pollutants and excessive salts. Uniformity in mix design is maintained and predictable fresh and hardened qualities are supported by consistent quality in terms of chemical composition and pH levels.

### **3.2.4 Admixture Sourcing**

In this study, we used an air-entraining admixture known as Sika® AIR, which was sourced from Rhombus Concrete, a construction company located at Rhombus HQ, Tara Road off Kiambu Road, Nairobi.

**Sika® AIR** is an organic aqueous solution used for the air entrainment of concrete. It meets the requirements of ASTM C260/AASHTO M 154, ensuring its quality and effectiveness. Ther recommended dosage is by the manufacturer is (7-400 mL / 100 kg) of cement

The use of Sika® AIR provides several benefits to the concrete mixture:

1. **Enhanced Durability:** Sika® AIR enhances the durability of the concrete by providing greater resistance to freeze-thaw cycles. This is particularly important in climates where concrete structures are exposed to freezing and thawing conditions, which can cause damage over time.
2. **Controllability of Air Content:** The admixture allows for precise control over the air content in the concrete. This is crucial as the amount of air in the concrete can significantly impact its workability and strength.
3. **Improved Air-Void System:** Sika® AIR improves the air-void system in the concrete. A well-distributed air-void system can improve the concrete’s resistance to cracking and scaling caused by freeze-thaw cycles.
4. **Reduced Capillary Channels:** The admixture reduces the size and number of capillary channels in the concrete. This can decrease water absorption and increase the concrete’s resistance to weathering and chemical attack.
5. **No Added Chlorides:** Sika® AIR does not contain intentionally added chlorides. This is important as chlorides can lead to corrosion of steel reinforcement in concrete structures.

The careful sourcing and use of this admixture are crucial in our study as it directly impacts the tensile strength of the concrete, which our predictive model aims to forecast. By using Sika® AIR, we aim to develop a more accurate and robust tensile predictive model for concrete. This approach also allows us to understand the role of air-entraining admixtures in the concrete’s tensile strength.

## **3.3 CONCRETE MIX DESIGN AND PROPORTIONS.**

The concrete mix design and proportions for each of the cylinder samples (100mm by 200mm) we created are as follows:

| **Cement Type** | **Admixture** | **Cement-Water Ratio** | **Aggregate Size** |
| --- | --- | --- | --- |
| OPC | Air-Entraining | 0.45 | 5mm |
| KP Silver | -- | 0.50 | 10mm |
| Quick Setting | -- | 0.60 | 20mm |

## **3.4 Data Collection**

### **3.4.1 Sample Preparation**

The sample preparation process was carried out over three days, producing nine samples each day for a total of 27 samples. The samples were prepared using different combinations of materials to ensure a comprehensive study. The combinations were determined based on the number of options for each factor: 3 Cement Types, 1 Admixture, 3 Cement-Water Ratios, and 3 Aggregate Sizes. This resulted in a total of 27 combinations (3 x 1 x 3 x 3).

Here’s a detailed step-by-step procedure of how one sample was prepared:

1. **Select Materials:** Choosing the type of cement, admixture, cement-water ratio, and aggregate size based on the specific combination for the sample.
2. **Measure Materials:** Measure the quantities of cement, aggregate, and water based on the chosen cement-water ratio and the 1:2:4 mix ratio for cement, sand, and coarse aggregate. The following procedure was used to determine the quantity of the cement, aggregate and water:

Step by step for one of the mold, the OPC mold of Air-Entraining, Cement-Water Ratio 0.45,Aggregate Size 5mm.

1. **Calculate the Volume of the Cylinder**
   * The formula for the volume of a cylinder is

V=πr2h

where r is the radius and h is the height.

* + Given a mold cylinder with a diameter of 100mm and height of 200mm, the radius r is half the diameter, so r is 50mm. The height h is 200mm.
  + Substituting these values into the formula gives us the volume of the concrete needed:

V=π(50mm)2(200mm)

* + Converting the dimensions from millimeters to meters (since 1m = 1000mm), we get:

V=π(0.05m)2(0.2m)=0.00157m3

* + This is the volume of the concrete needed in cubic meters.

1. **Determine the Weight of the Cement Needed**
   * Once we have the volume of the concrete, we can determine the weight of the cement needed based on the concrete mix ratio and the specific gravity of the cement.
   * Assuming the specific gravity of the cement is around 3.15 g/cm³ (a typical value), we can calculate the weight of the cement needed for our volume of concrete.
   * In the 1:2:4 mix ratio, the total ratio is 1+2+4 = 7. So, the proportion of cement in the mix is  *.*
   * Therefore, the weight of cement needed is:

Weight of cement = Volume of concrete × Proportion of cement in the mix × Specific gravity of cement × 1000Weight of cement=Volume of concrete×Proportion of cement in the mix×Specific gravity of cement×1000

* + Substituting the known values:
  + The factor of 1000 is used to convert from cubic meters to kilograms, since 1 cubic meter of water weighs 1000 kg.

1. **Calculate the Weights of the Sand and Coarse Aggregate**
   * Once we have the weight of the cement, we can calculate the weights of the sand and coarse aggregate using the mix ratio. For the sand, it would be 2x kg, and for the coarse aggregate, it would be 4x kg.
   * So, for the sand:
   * And for the coarse aggregate:

So, for one cylinder, you would need approximately 0.707 kg of cement, 1.414 kg of sand, and 2.828 kg of coarse aggregate. This process helps you determine the exact amount of materials needed for your concrete mix, ensuring efficiency and accuracy in the making of the samples.

1. **Mix Dry Materials:** Mix the cement and aggregates together in a mixing container until a uniform mixture is achieved.
2. **Add Water:** Gradually add water to the dry mix while continuing to mix. Ensure that the water is distributed evenly throughout the mix.
3. **Add Admixture:** If an admixture is used in the combination, add it to the mix according to the manufacturer’s instructions.
4. **Mix Concrete:** Continue mixing until a uniform, workable consistency is achieved. The concrete should be workable enough to be placed into the mold but stiff enough to hold its shape.
5. **Place in Mold:** Place the fresh concrete into the mold cylinder (100mm by 200mm). Ensure the concrete is fully compacted into the mold to avoid air pockets.

This procedure was repeated for each of the 27 samples, with the specific combination of materials (cement type, admixture, cement-water ratio, and aggregate size) being varied for each sample. This methodical approach to sample preparation ensures that the study covers a wide range of concrete compositions, providing a robust and comprehensive dataset for developing the tensile predictive model. It also allows for the examination of the effects of different cement types, cement-water ratios, and aggregate sizes on the tensile strength of the concrete.

### **3.4.2 Sample Curing**

After we prepared the samples, we allowed them to set for 24 hours. This initial setting time is crucial as it lets the concrete harden and gain enough strength to be handled without causing any damage or deformation to the sample.

Following this initial setting period, we placed the samples in a water basin for curing, which lasted for 7 days. This method of curing, known as immersion curing, is one of the most effective ways to cure concrete. It ensures that the concrete retains the necessary moisture, which is vital for the cement hydration process.

We chose a 7-day curing period, a standard duration in concrete testing. This period allows for a significant portion of the cement hydration process to occur, leading to a substantial development of concrete strength. However, it’s worth noting that concrete continues to gain strength beyond this 7-day period, and longer curing periods can lead to higher strength.

This curing process is essential as it directly impacts the strength, durability, and overall performance of the concrete. By ensuring that the concrete achieves its potential strength and durability, we prepared our samples effectively for the subsequent tensile strength testing.

### **3.4.3 Tensile Strength Testing**

Tensile strength is a critical mechanical property of concrete. However, accurately measuring it can be challenging due to the brittle nature of concrete under tension. There are three commonly used test methods for determining the tensile strength of concrete, each with its own limitations:

1. **Direct Tension Test:** This method can be difficult to set up, especially when aligning the equipment, and lacks standard test specifications.
2. **Flexural Strength Test (ASTM C78):** The results of this test can significantly differ from the actual tensile strength due to methodological issues.
3. **Splitting Tensile Strength Test (ASTM C496 and BS EN 12390-3:2003):** Similar to the flexural strength test, the results can show significant differences from the actual tensile strength.

In this study, we used the Splitting Tensile Strength Test, conducted according to the standards of ASTM C496.

**Materials Needed:**

* Cured concrete sample
* ASTM C496 Machine
* Ruler

**Procedure:**

1. **Machine Preparation:** We started by ensuring that the ASTM C496 machine was clean and ready for use. This step is crucial to prevent any contamination or interference that could affect the test results.
2. **Sample Positioning:** We then carefully placed one of our cured concrete samples in the machine. Proper alignment was ensured so that the load applied would be uniformly distributed across the entire cross-section of the sample.
3. **Load Application:** We started the machine and began to apply the load. The load was increased gradually at a steady rate to avoid any sudden shocks or stresses that could cause premature failure of the sample.
4. **Test Monitoring:** Throughout the test, we kept a close watch on the sample and the machine. We looked out for any signs of failure in the sample, such as cracking or deformation. The test continued until the sample failed, which was typically indicated by a crack or break in the sample.
5. **Results Recording:** Once the sample had failed, we recorded the maximum load that was applied to the sample. This is the load at which the sample could no longer withstand the tension and failed.
6. **Test Repetition:** We repeated this procedure for each of the remaining samples, ensuring that the results for each one were accurately recorded.

The data was recorded in Table shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Samples | Cement amount  (g)  (integer) | Water  (g)  (integer) | Type of cement  (string) | Average Aggregate size  (integer) | Aggregate  (Coarse)  Amount  (g) | Aggregate  (SAND)  Amount  (g) | Curing Duration  (days)  (integer) | Admixtures  (string) | Load at Fracture  (max Load)  (N)  (integer) |
| 1 |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |

## **3.5 DATA PREPROCESSING.**

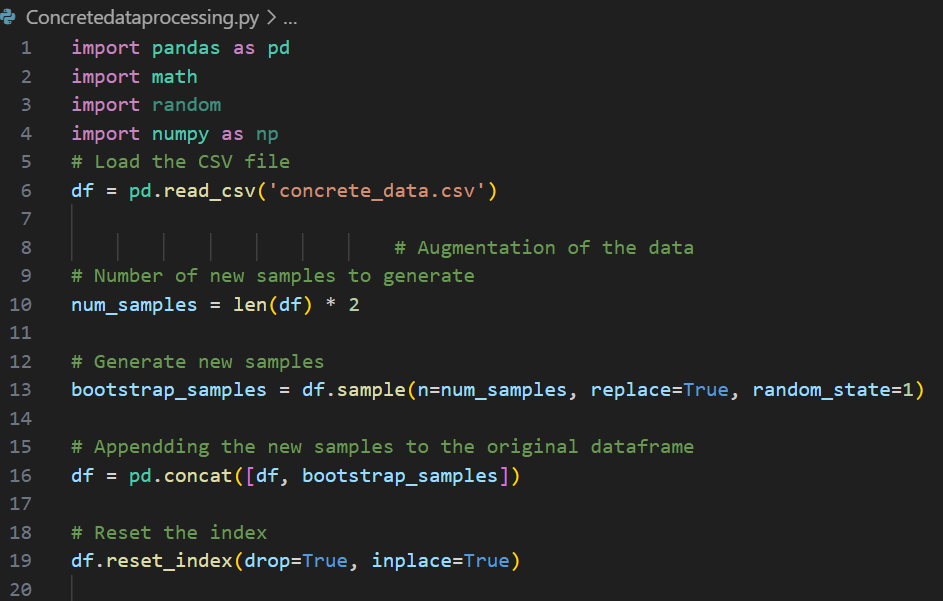
In this section, we detail the processes we undertook in preparing the data for Model development. This involved a series of steps to ensure the data was accurate and ready for further processing.

### **3.5.1 Data Augmentation.**

Data augmentation is a strategy that enables us to significantly increase the diversity of data available for training models, without actually collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.

In our case, we performed data augmentation to increase the size of our dataset and also to avoid bias. The function we used is called **bootstrap sampling.**

Bootstrap sampling is a method that involves drawing random samples from a dataset with replacement. It’s a widely used technique in statistics to create numerous resampled versions of our data. We used bootstrap sampling as a form of data augmentation to increase the size of our dataset and reduce bias.



In the code above, we first load our dataset from a CSV file. We then determine the number of new samples to generate, which is twice the size of our original dataset. We generate these new samples using the sample function from pandas, which allows us to randomly sample items from our dataset. The replace =True parameter indicates that we are doing bootstrap sampling, i.e., sampling with replacement. This means some samples may be selected more than once, which is what we want for data augmentation.

After generating the new samples, we append them to our original dataframe to get a new dataframe that’s three times the size of our original dataset. Finally, we reset the index of our dataframe to ensure it’s in proper order. This augmented dataset is now ready for further processing and model development.

### **3.5.2 Tensile Strength Calculation**

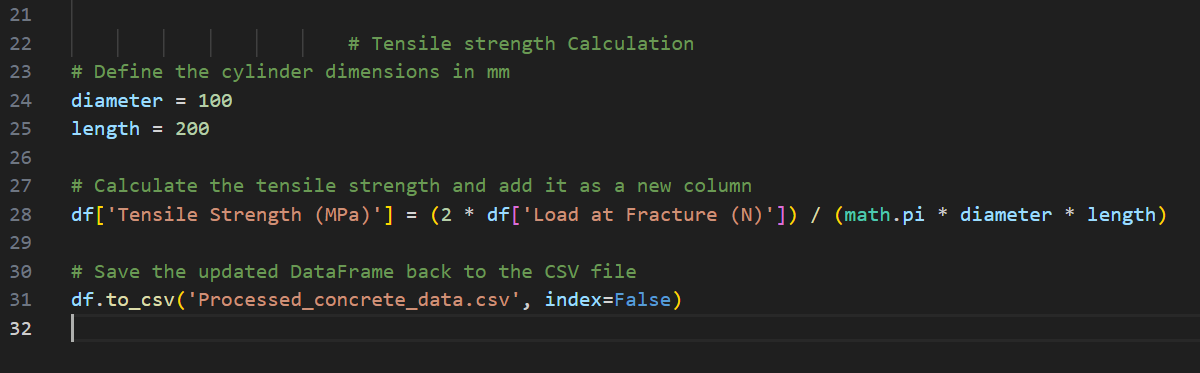
One of the key steps in our data preprocessing was the calculation of the tensile strength of the concrete samples. Tensile strength is a crucial property of concrete that significantly influences its performance.

To calculate the tensile strength, we used the following formula:

This formula allowed us to compute the tensile strength based on the maximum load that each sample could withstand, and the dimensions of the sample (diameter and length).

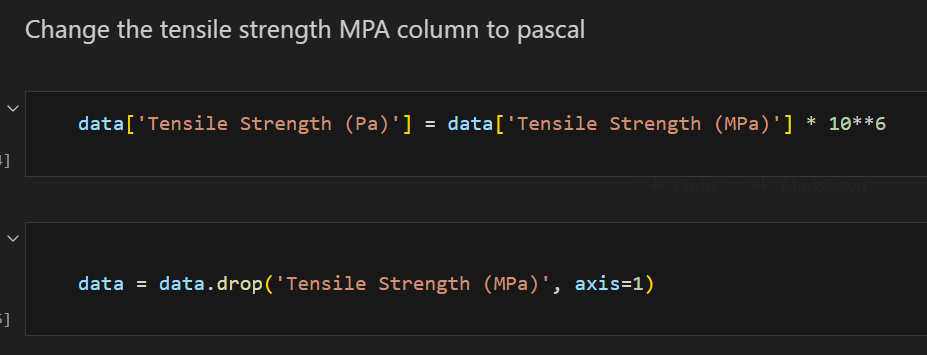
To facilitate the calculation process and handle the large volume of data, we utilized Python programming. Python, being a powerful and flexible programming language, made it easier for us to perform these calculations efficiently and accurately.

Here’s a snippet of the Python code we used for the tensile strength calculation:



This code reads the data from a CSV file, calculates the tensile strength for each sample using the formula, and adds these calculations as a new column in the data frame. This streamlined our data preprocessing and ensured that our data was ready for the subsequent stages of our analysis and saved in the Processed\_concrete\_data.csv file.

We performed a unit conversion on the “Tensile Strength” column in our dataset. The original values were in megapascals (MPa), but we needed them in pascals (Pa) for our analysis.

Here’s how we did it:

In the code above, we created a new column ‘Tensile Strength (Pa)’ in our dataframe. We populated this column by taking each value in the ‘Tensile Strength (MPa)’ column and multiplying it by (10^6). This is because 1 MPa is equal to (10^6) pascals.

After we had our new column with the tensile strength values in pascals, we no longer needed the original ‘Tensile Strength (MPa)’ column. So, we removed it from our dataframe. we used the drop function from pandas to remove the ‘Tensile Strength (MPa)’ column from our dataframe. The axis=1 parameter indicates that we are dropping a column (not a row).

## **3.6 DATA ENGINEERING.**

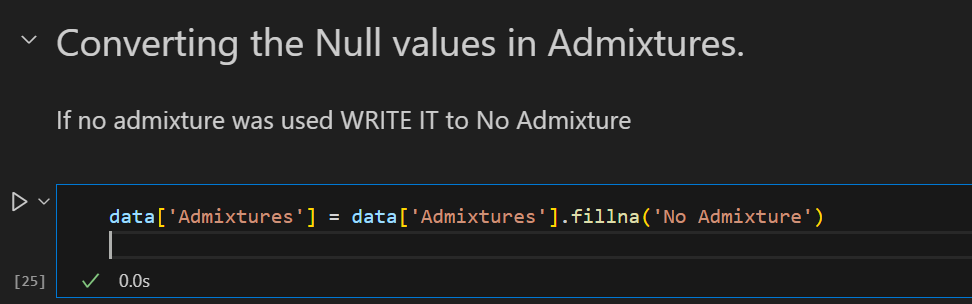
In this section, we focus on preparing our dataset for further analysis and modeling. One of the key steps in this process is handling missing values.

**Handling Missing Values in the ‘Admixtures’ Column**

We observed that the ‘Admixtures’ column in our dataset contains some missing values (NaN). To handle these, we decided to replace all NaN values with the term ‘No Admixture’. This approach is beneficial for a couple of reasons:

1. It allows us to retain these records instead of discarding them, which would lead to loss of information.
2. It simplifies the subsequent encoding process by treating ‘No Admixture’ as another category within the ‘Admixtures’ column.

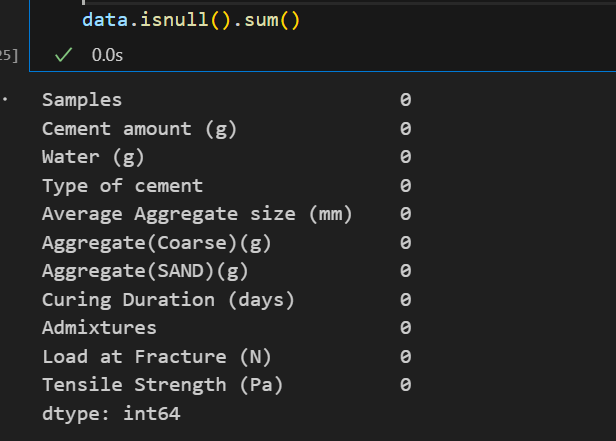
Here’s the Python code that accomplishes this:



**Checking for Other Missing Values**

After handling the missing values in the ‘Admixtures’ column, we proceed to check if there are any other missing values in the entire dataset. This is an important step to ensure the cleanliness and integrity of our data.

Here’s the Python code that checks for other missing values:



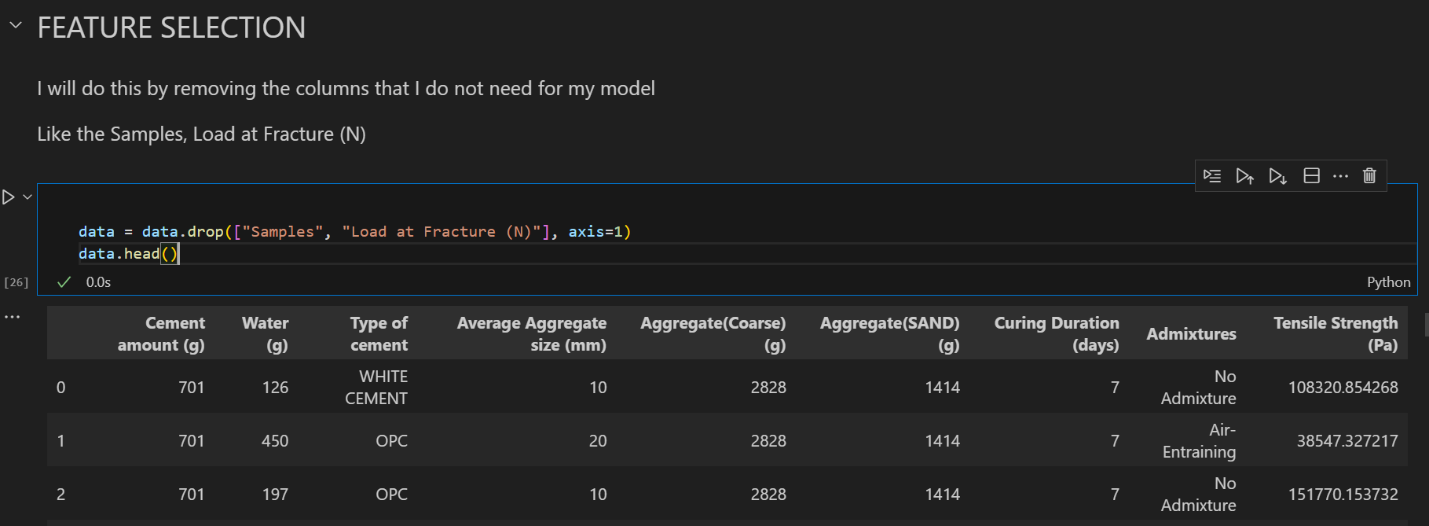
## **3.7 FEATURE SELECTION.**

In this section, we focus on the process of feature selection, which is a crucial step in building our predictive model for tensile strength.

Feature selection involves choosing the most relevant variables (or ‘features’) from our dataset that contribute to the predictive power of our model. The goal is to improve the model’s performance by reducing overfitting, improving accuracy, and reducing training time.

In our case, we identified certain columns in our dataset that do not significantly contribute to the prediction of tensile strength. These columns are “Samples” and “Load at Fracture (N)”. By eliminating these features from our dataset, we aim to increase the efficiency of our model and enhance its predictive accuracy.

Here’s the Python code that performs this feature selection:



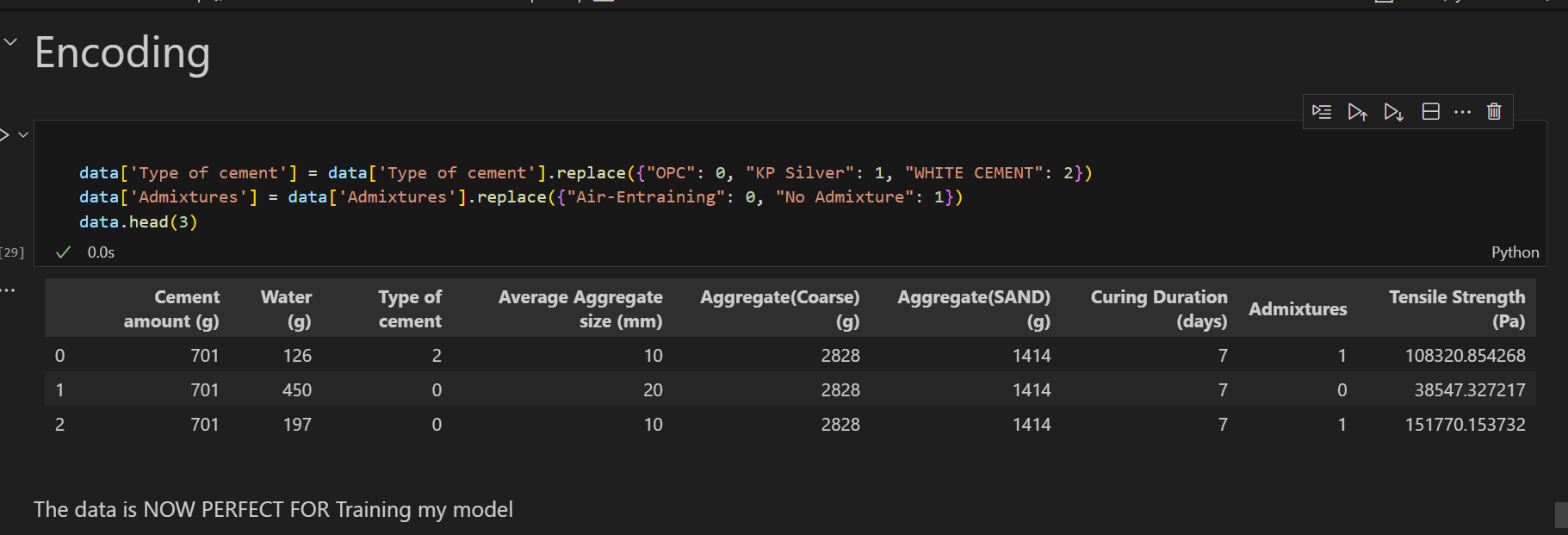
This code removes the “Samples” and “Load at Fracture (N)” columns from our dataset. The data.head() function is then used to display the first few rows of the updated dataset, allowing us to confirm that the specified columns have been successfully dropped.

## **3.8 ENCODING.**

In this section, we focus on encoding our data. Encoding is a process of converting categorical data into numerical data. This step is crucial because machine learning models, including PyTorch models, require numerical inputs and cannot process categorical (string) inputs directly.

We have two categorical variables in our dataset: ‘Type of cement’ and ‘Admixtures’. We use a simple form of encoding known as label encoding, where each unique category value is assigned a unique integer.

Here’s the Python code that performs this encoding:



In the ‘Type of cement’ column, ‘OPC’ is replaced with 0, ‘KP Silver’ with 1, and ‘WHITE CEMENT’ with 2. In the ‘Admixtures’ column, ‘Air-Entraining’ is replaced with 0, and ‘No Admixture’ with 1.

After encoding, we can confirm the changes by displaying the first few rows of the updated dataset using data.head(). This ensures that our dataset is now ready for training with PyTorch.

## **3.9 SPLITING AND NORMALIZING**

In this section, we focus on two important steps in preparing our data for modeling: splitting and normalizing.

**Data Splitting**

We split our dataset into a training set and a testing set. The training set is used to train our model, while the testing set is used to validate the model’s performance. We use an 80-20 split, meaning 80% of the data is used for training and 20% is used for testing. This allows us to train our model while simultaneously validating its performance.

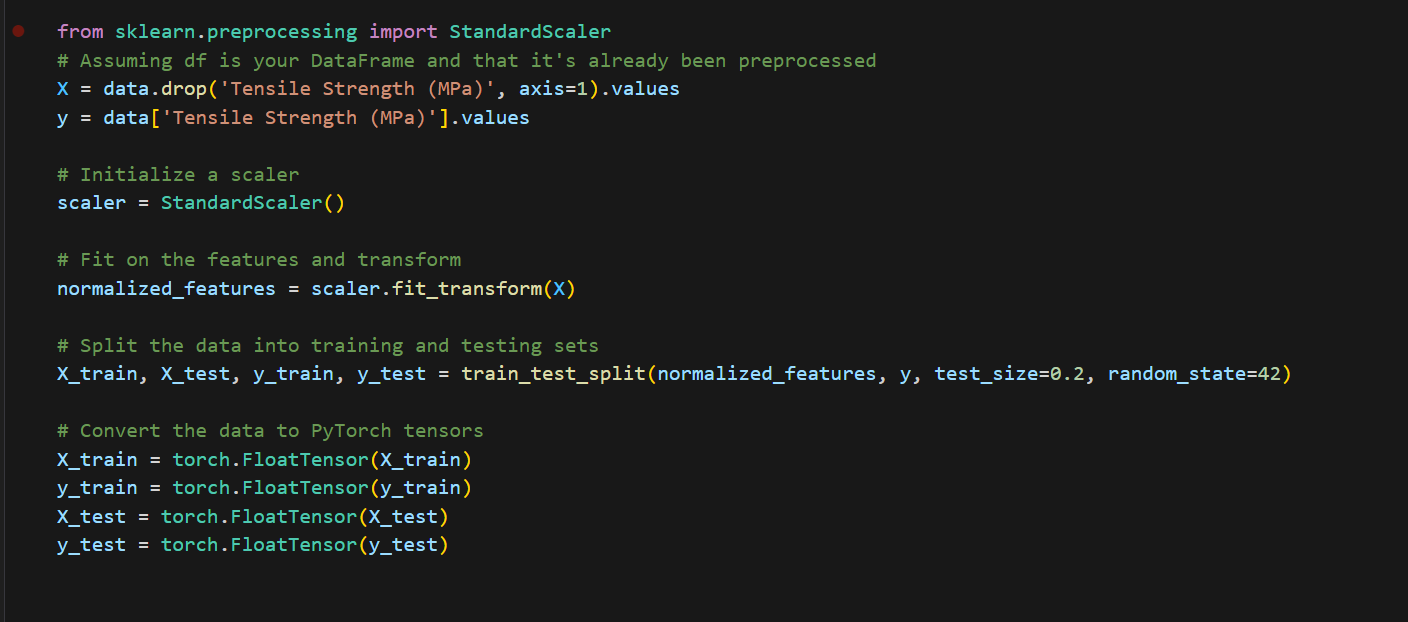
**Data Normalization**

Before training our model, we normalize our features. Normalization is a scaling technique that adjusts the values of different features to a similar scale. This is important because features might be measured in different units, and we want to ensure that no particular feature dominates others due to its scale. We use the StandardScaler from sklearn.preprocessing to perform this normalization.

**Conversion to PyTorch Tensors**

Finally, we convert our data to PyTorch tensors. Tensors are a generalization of matrices and are used in PyTorch to formulate and solve neural networks. Converting our data to tensors ensures compatibility with PyTorch and prepares our data for the training process.

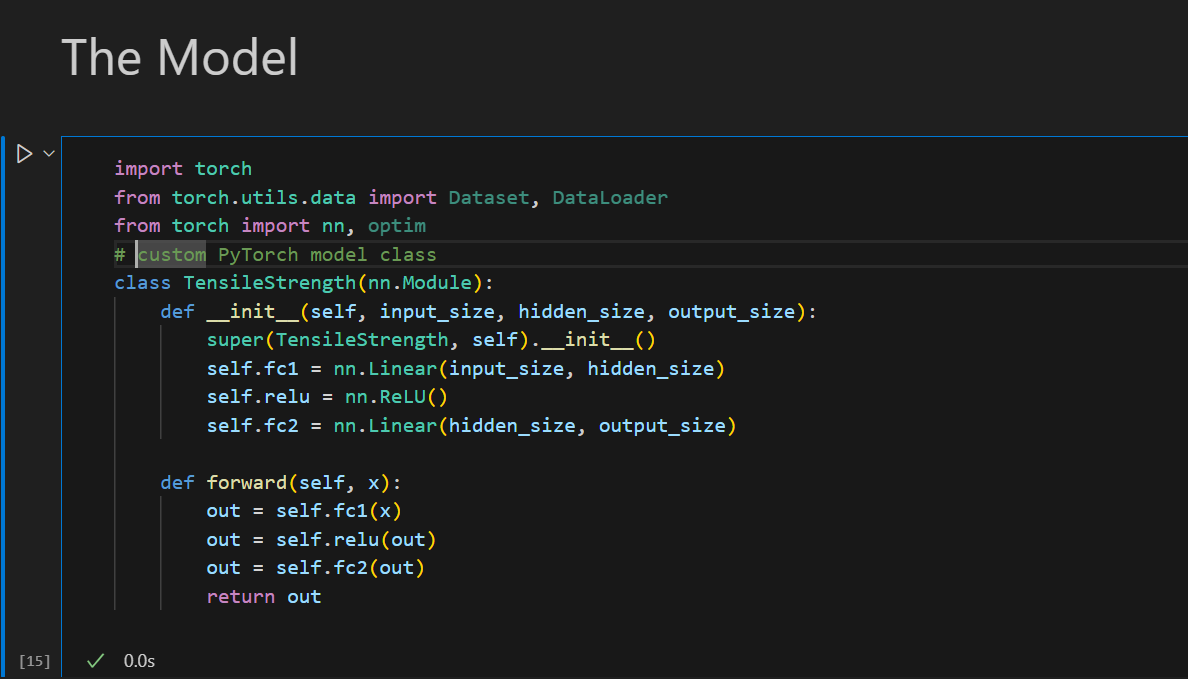
Here’s the Python code that performs these steps:



## **3.10 MODEL DEVELOPMENT**

In this section, we focus on developing our predictive model using the PyTorch library. PyTorch is a popular open-source machine learning library that allows us to build and train neural network models.

Here’s a breakdown of the code:



We define a custom PyTorch model class called TensileStrength. This class inherits from nn.Module, which is the base class for all neural network modules in PyTorch.

The TensileStrength class has the following components:

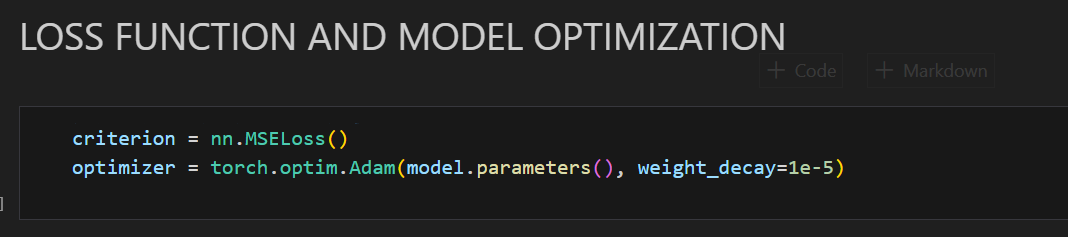
1. **Initialization (\_\_init\_\_ method)**: This method initializes the layers of the neural network. We have two fully connected (nn.Linear) layers and a ReLU activation function (nn.ReLU).
   * nn.Linear(input\_size, hidden\_size): This creates the first fully connected layer (fc1) that connects the input layer to the hidden layer. The number of nodes in the input layer is specified by input\_size, and the number of nodes in the hidden layer is specified by hidden\_size.
   * nn.ReLU(): This creates a ReLU (Rectified Linear Unit) activation function that will be applied after the first fully connected layer. The ReLU function is commonly used in neural networks to introduce non-linearity.
   * nn.Linear(hidden\_size, output\_size): This creates the second fully connected layer (fc2) that connects the hidden layer to the output layer. The number of nodes in the hidden layer is specified by hidden\_size, and the number of nodes in the output layer is specified by output\_size.
2. **Forward Propagation (forward method)**: This method defines the forward propagation of the neural network, which is how the data flows through the network. The data (x) is passed through the first fully connected layer (self.fc1(x)), then through the ReLU activation function (self.relu(out)), and finally through the second fully connected layer (self.fc2(out)).

This model structure is a simple feed-forward neural network (also known as a multi-layer perceptron), which is a good starting point for many regression tasks. Depending on the complexity of the task and the data, the model architecture can be further expanded by adding more layers or changing the type of layers.

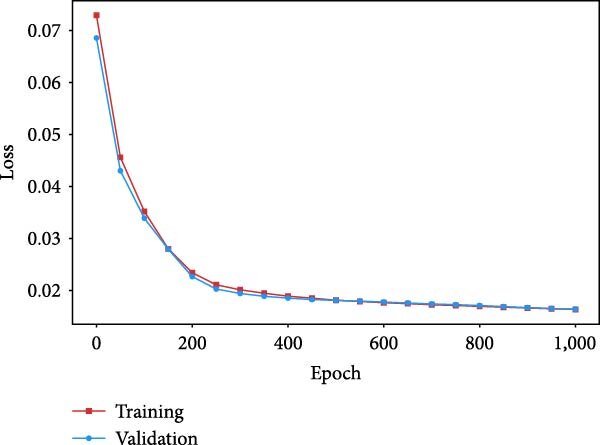
## **3.11 LOSS FUNCTION AND OPTIMIZATION**

In this section, we focus on defining the loss function and the optimization method for our model. These are crucial components in training a neural network.

Here’s a breakdown of the code:

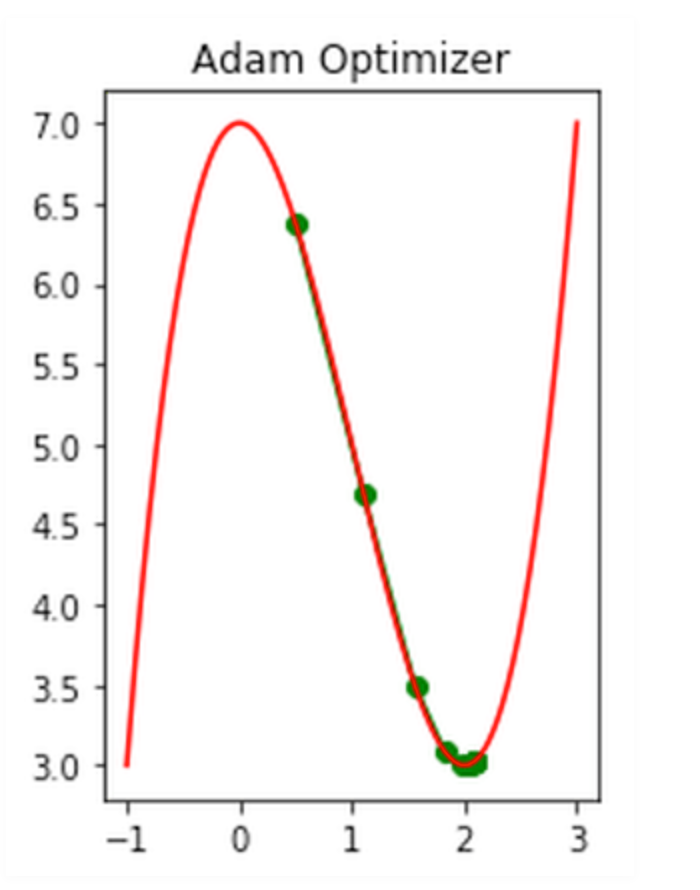


1. **Loss Function (**criterion**)**: The loss function measures how well our model is performing by comparing the model’s predictions with the actual values. In this case, we’re using Mean Squared Error (MSE) loss, which is commonly used for regression problems. MSE calculates the average squared difference between the predicted and actual values, with lower values indicating better model performance.



The red curve represents the MSE loss between the real proportions and the predicted proportions for the training set in each epoch of training, while the blue curve represents the MSE loss between the real proportions and the predicted proportions for the validation set in each epoch of training.

1. **Optimizer**: The optimizer is an algorithm used to adjust the parameters of our model to minimize the loss function. Here, we’re using the Adam optimizer, a popular choice due to its efficiency and low memory requirement. The weight\_decay parameter is used for regularization, which helps prevent overfitting by penalizing complex models.



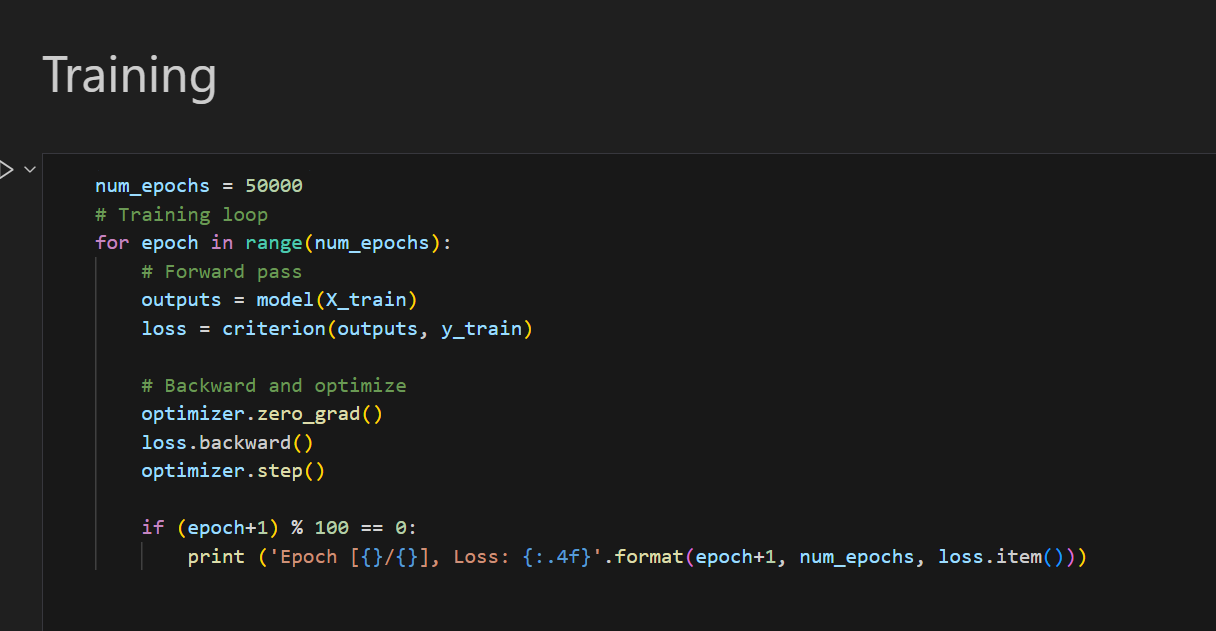
The model.parameters() in the optimizer initialization is a call to the parameters of our model (weights and biases), which the optimizer needs to know so it can update them during training.

In summary, the loss function and optimizer work together during the training process to adjust the model’s parameters in a way that minimizes the loss, thereby improving the model’s accuracy. The loss function provides a measure of the model’s performance, and the optimizer uses this measure to adjust the model’s parameters. This process is repeated for a number of iterations (or ‘epochs’) until the model’s performance is satisfactory.

## **3.12 TRAINING**

In this section, we focus on training our model. Training is the process where our model learns from the data. It involves running the data through the model (forward pass), calculating the loss, and then updating the model parameters (backward pass and optimization). This process is repeated for a certain number of iterations known as epochs.

Here’s a breakdown of the code:

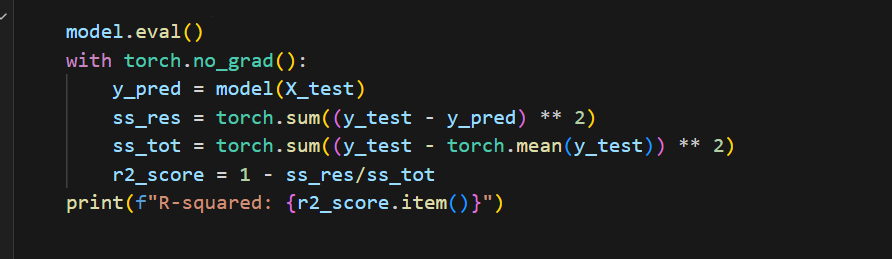


1. **Number of Epochs**: We set the number of epochs to 50000. This means that the entire dataset will be passed forward and backward through the neural network 50000 times.
2. **Training Loop**: We start a loop that will iterate for the number of epochs.
3. **Forward Pass**: In the forward pass, we pass our input data (X\_train) through the model. The model returns some output which is stored in the outputs variable.
4. **Calculate Loss**: We then calculate the loss using the criterion we defined earlier (MSE Loss). The loss is calculated by comparing the model’s output (outputs) with the actual target values (y\_train).
5. **Backward Pass and Optimization**:
   * We first reset all gradients to zero using optimizer.zero\_grad(). This is because PyTorch accumulates gradients, and we don’t want to mix up gradients between epochs.
   * We then perform a backward pass using loss.backward(). This computes the gradient of the loss with respect to the model parameters.
   * Finally, we call optimizer.step() to perform an optimization step. This updates the model parameters (weights and biases) using the gradients computed in the backward pass.
6. **Print Loss**: Every 100 epochs, we print out the epoch number and the current loss. This helps us monitor the training process and ensure that the loss is decreasing, indicating that our model is learning.

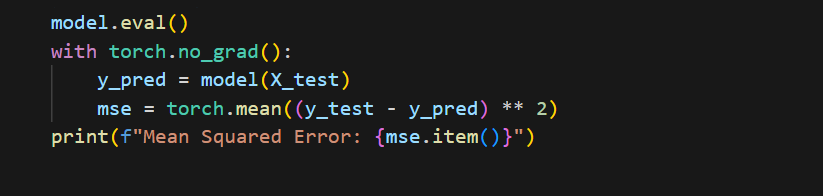
By performing these steps, we train our model to fit our data, adjusting its parameters to minimize the loss and improve its predictive performance.

## **3.13 EVALUATION**

In this section, we focus on evaluating the performance of our trained model. We use two metrics for this purpose: R-squared and Mean Squared Error (MSE).



1. **R-squared**: This is a statistical measure that represents the proportion of the variance for a dependent variable that’s explained by an independent variable or variables in a regression model. The higher the R-squared, the better the model fits your data. Here, ss\_res is the sum of squares of residuals, and ss\_tot is the total sum of squares. The R-squared is then calculated as 1 - ss\_res/ss\_tot.

]

**2. Mean Squared Error (MSE)**: This is the average of the squared differences between the predicted and actual values. It’s a popular metric for regression problems. The lower the MSE, the better the model’s performance.

In both cases, we first set our model to evaluation mode using model.eval(). Then, we use torch.no\_grad() to tell PyTorch that we do not want to perform back-propagation, which reduces memory usage and speeds up computation. We then use our model to predict the output for our test data (model(X\_test)), and compare these predictions (y\_pred) to the actual values (y\_test) to compute R-squared and MSE. The results are then printed out. This gives us a quantitative measure of how well our model is performing.

## **3.14 MODEL SAVING.**

In this section, we focus on saving our trained model. Saving a model is important because it allows us to reuse the model later without having to retrain it. This can save a lot of time and computational resources.

Here’s a breakdown of the code:



The torch.save() function is used to save the model. This function takes two arguments:

1. The model we want to save (model in this case).
2. The path where the model should be saved ('Model/copytensile\_strength\_model.pth' in this case).

The .pth extension is commonly used for PyTorch models.

After running this code, the trained model is saved to the specified path. You can load this model later using the torch.load() function and use it for making predictions, without having to retrain the model. This is particularly useful when model training is time-consuming or computationally expensive. This will save the entire model, which includes both the architecture and the learned parameters (weights and biases).

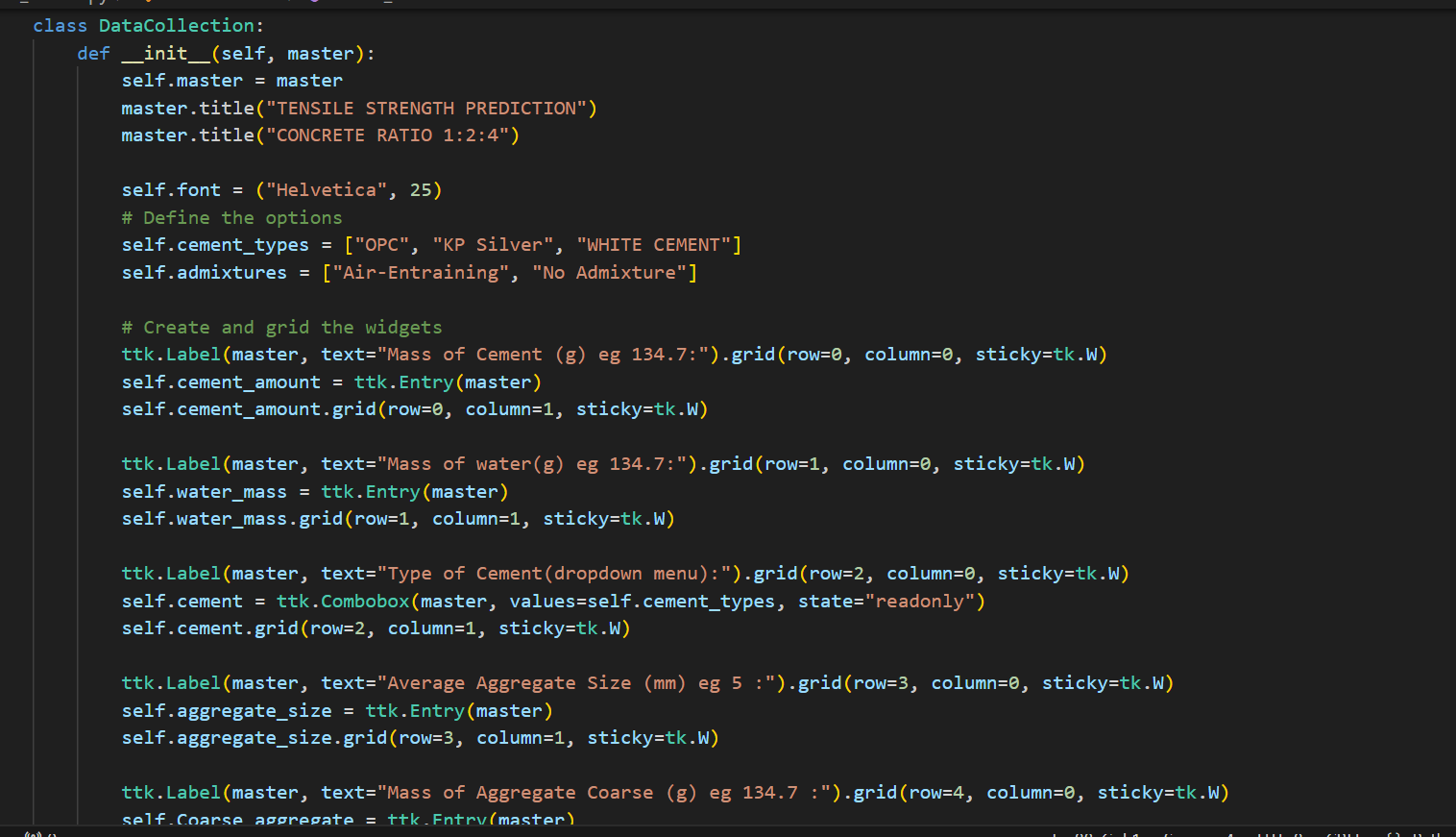
## **3.15 MODEL PREDICTION.**

In this section, we focus on making predictions with our trained model. The prediction process is divided into two main parts: data collection and model prediction.

### **3.15.1 Data Collection**

The first part involves collecting the data that we want to make predictions on. To make this process user-friendly, we create a Graphical User Interface (GUI) using the tkinter library in Python. This GUI allows users to input the data in a clean, professional, and pleasant manner.

Here’s a breakdown of the code:



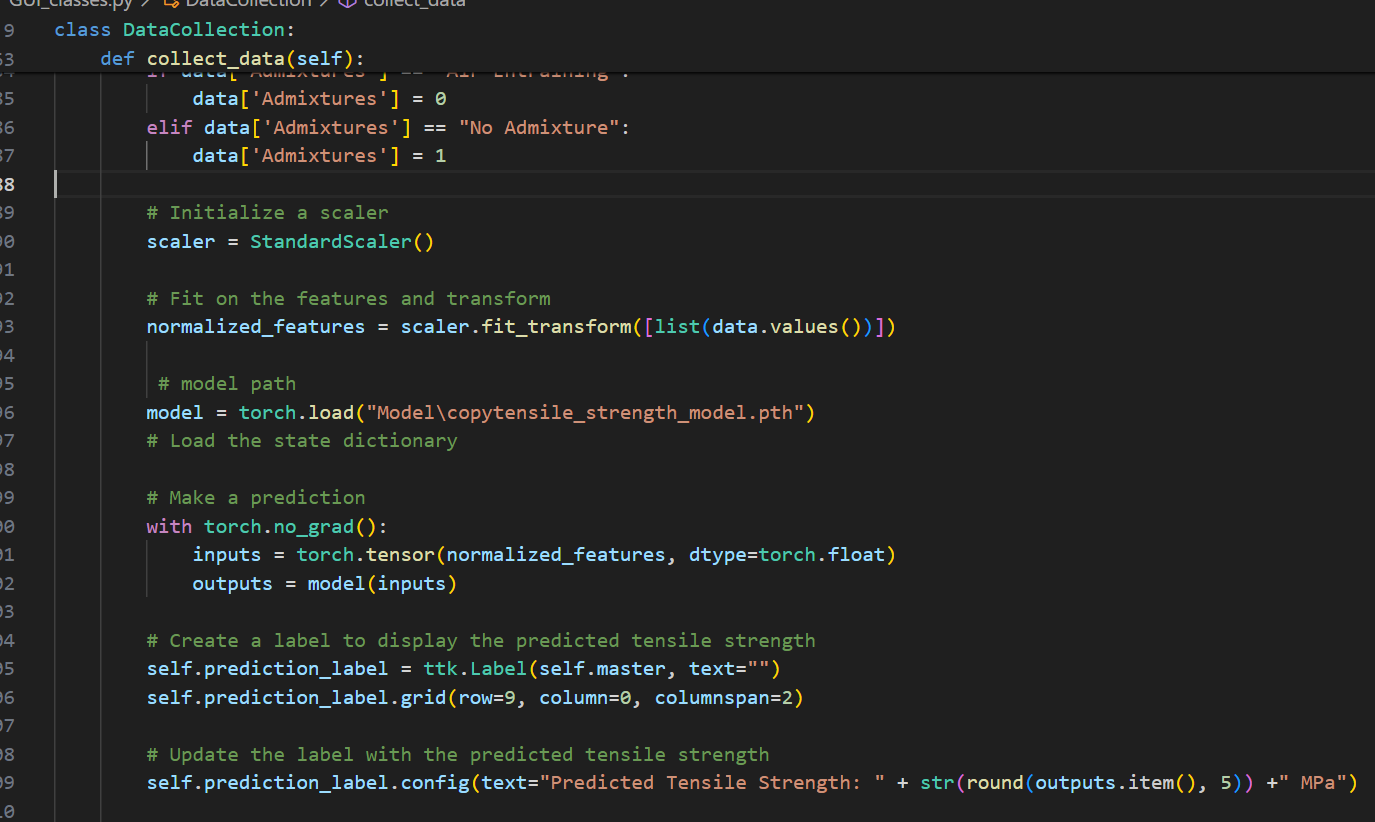
We define a class DataCollection that creates a GUI for data input. The GUI includes fields for entering various parameters like ‘Cement amount (g)’, ‘Water (g)’, ‘Type of cement’, etc. Each field is created using ttk.Entry for text input or ttk.Combobox for dropdown selection.

When the ‘Submit’ button is clicked, the collect\_data method is called. This method collects the data from all the input fields and stores it in a dictionary called data. This data can then be used for making predictions with our model.

### **3.15.2 Predictions**

In this section, we focus on making predictions with our trained model. The process involves data processing, normalization, loading the trained model, making a prediction, and displaying the predicted tensile strength on the GUI.

Here’s a breakdown of the code:



1. **Data Processing**: The categorical variables ‘Type of cement’ and ‘Admixtures’ are converted into numerical form. This is necessary because machine learning models require numerical inputs.
2. **Normalization**: The data is then normalized using the StandardScaler from sklearn.preprocessing. Normalization is a scaling technique that adjusts the values of different features to a similar scale. This is important because features might be measured in different units, and we want to ensure that no particular feature dominates others due to its scale.
3. **Load the Trained Model**: The trained model is loaded from the specified path using torch.load().
4. **Make a Prediction**: The normalized data is converted into a PyTorch tensor and passed through the model to get the predicted tensile strength.
5. **Display the Prediction**: A label is created on the GUI to display the predicted tensile strength. The label is updated with the predicted value.
6. **Print the Data and Prediction**: The processed data and the predicted tensile strength are printed to the console. This is optional and can be useful for debugging or logging purposes.

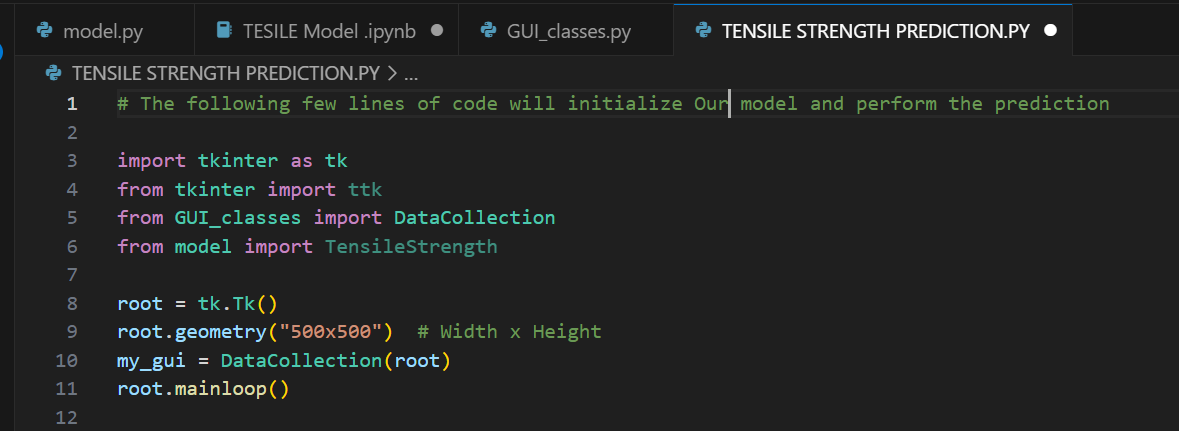
It’s crucial to note that all data preprocessing for the prediction phase is performed in the same manner as during the training phase. This consistency is vital to ensure that the input data for prediction aligns with what the model expects based on its training. Any discrepancies in data preprocessing between the training and prediction phases could lead to confusion for the model and potentially result in inaccurate predictions. Therefore, maintaining uniformity in data preprocessing is a key aspect of reliable model performance.

In summary, this process allows us to collect data in a user-friendly way, process and normalize the data, make predictions using our trained model, and display the predictions in a clear and understandable manner.

### **3.15.3 Making Predictions**

In this section, we focus on making predictions with our trained model. To facilitate this process, we utilize a Graphical User Interface (GUI) for data input and prediction display. The GUI is designed to provide a user-friendly experience, ensuring a smooth prediction process.

Here’s a breakdown of the code:



1. **Import Necessary Libraries and Modules**: We import the necessary libraries (tkinter and ttk) and modules (DataCollection and TensileStrength) for creating the GUI and making predictions.
2. **Initialize Tkinter Window**: We create a Tkinter window (root) and set its size to 500x500 pixels using root.geometry("500x500").
3. **Initialize Data Collection GUI**: We create an instance of the DataCollection class (my\_gui), passing the root window as an argument. This initializes the data collection GUI on our window.
4. **Start Tkinter Event Loop**: Finally, we start the Tkinter event loop using root.mainloop(). This starts the GUI and waits for user interaction.

By running this code, we initialize our data collection GUI and are ready to collect data, make predictions using our trained model, and display the predictions in a clear and understandable manner. The GUI provides a clean and professional interface for users to interact with the model and obtain predictions. It simplifies the prediction process and enhances the user experience.

# **CHAPTER FOUR**

# **4.0 RESULTS, ANALYSIS AND DISCUSSION**

In this section, we delve into the outcomes of our study, providing a comprehensive examination and interpretation of the results. We aim to link our findings to our initial objectives and hypotheses, discussing their implications in a broader context.

## **4.1 Introduction.**

The purpose of this introductory subsection is to set the stage for the detailed discussion that follows. We will revisit the objectives of our study, briefly summarize the methods employed, and provide an overview of the main results. This will facilitate a deeper understanding as we proceed with a thorough analysis and discussion of our findings, their significance, and their potential impact. We will also address any limitations encountered during the study and propose directions for future research.

## **4.2 Data Collection.**

During the data collection phase of our study, we encountered several challenges. One of the primary obstacles was the lack of funding to purchase more admixtures, which limited the amount of data we could collect. Despite this constraint, we managed to augment our dataset using a technique known as bootstrapping.

Bootstrapping is a statistical method that involves resampling a dataset with replacement. It allowed us to effectively double our data from 27 samples to 81 samples. This method was particularly efficient for our data because it allowed us to maximize the use of our limited resources and generate a larger and more diverse dataset for our model to learn from.

However, it’s important to note that while bootstrapping can be a useful tool in certain situations, it’s not always the most efficient method for acquiring data. It essentially creates synthetic data, which may not capture the full complexity and variability of real-world data. Therefore, the results obtained from a model trained on bootstrapped data should be interpreted with caution.

Another limiting factor in our data collection process was time. The process of preparing the samples, conducting the experiments, and collecting the data was time-consuming. This further constrained the amount of data we could collect.

Despite these challenges, we were able to collect a dataset that included variables such as ‘Cement amount (g)’, ‘Water (g)’, ‘Type of cement’, ‘Average Aggregate size (mm)’, ‘Curing Duration (days)’, ‘Admixtures’, and ‘Load at Fracture (N)’. This data served as the foundation for our subsequent analysis and model development.

## **4.2 Data processing.**

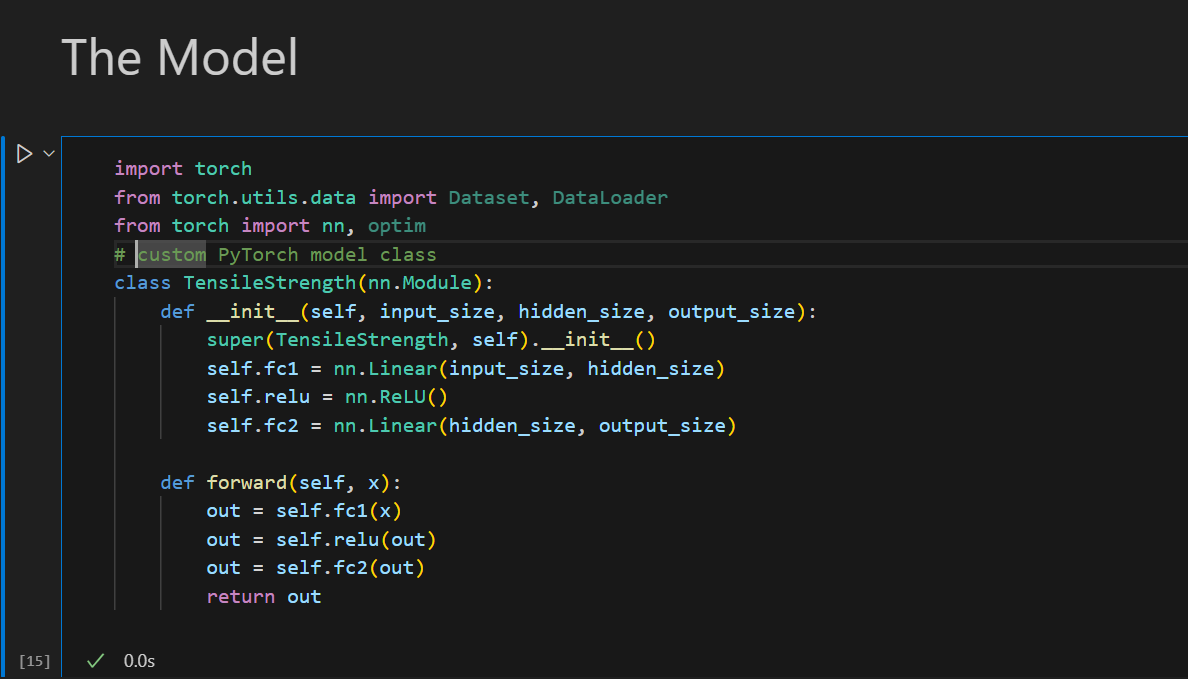
In this section, we focused on preparing our dataset for model development. This involved several key steps:

1. **Handling Missing Values**: During the data collection phase, we encountered a lack of sufficient admixtures to use in sample preparation. This resulted in missing values in our dataset. To handle these missing values, we replaced them with the term ‘No Admixture’. This approach allowed us to retain these records instead of discarding them, which would lead to loss of information.
2. **Encoding**: After handling the missing values, we proceeded to encode our data. Encoding is a process of converting categorical data into numerical data. This step is crucial because machine learning models require numerical inputs. In our case, we converted the ‘No Admixture’ and ‘Air-Entraining’ categories in the ‘Admixtures’ column into numerical form.
3. **Normalization**: Finally, we normalized our data using the StandardScaler method from the sklearn.preprocessing library in Python. Normalization is a technique used to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. In our case, we scaled the data to a range between 0 and 1. Normalization is important when the dataset contains features that are on different scales and can help improve the performance of the machine learning model.

By performing these steps, we ensured that our dataset was clean, properly formatted, and ready for model development. These preprocessing steps are crucial in any machine learning project and can significantly impact the performance of the final model.

## **4.3 Model Development.**

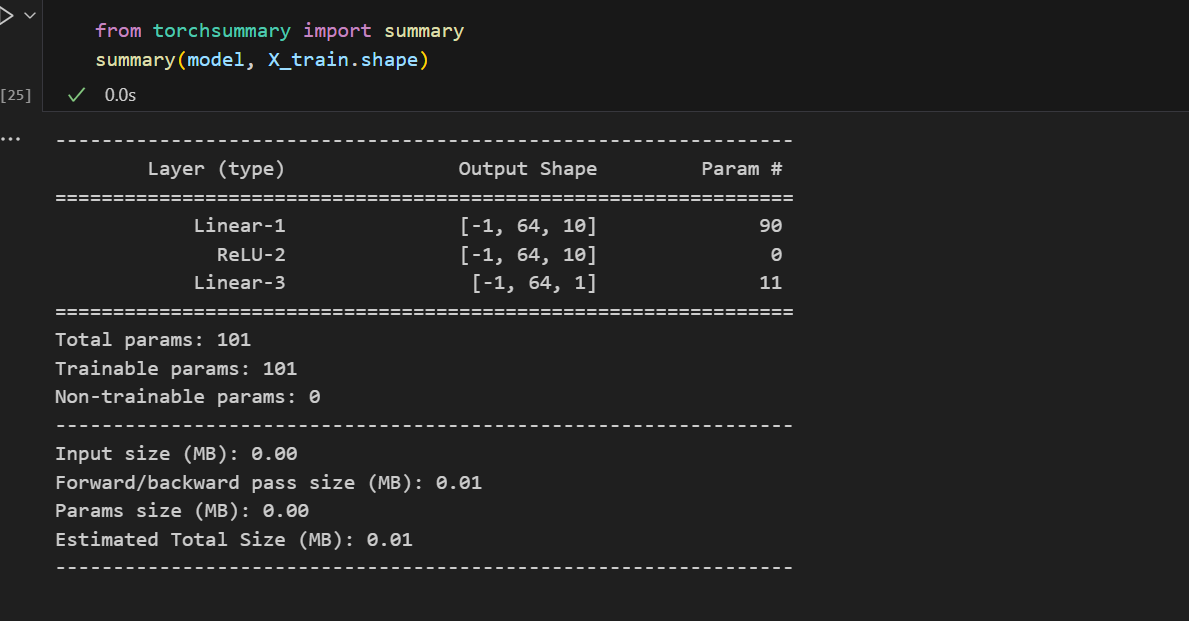
In this section, we built a simple model to predict tensile strength using PyTorch, a popular library for building machine learning models. Let’s break down the code and the model architecture in a way that’s easy to understand.



Think of our model as a little brain that’s trying to learn how to predict tensile strength. This brain has different parts (layers) that work together to make these predictions.

1. **First Layer (fc1)**: This is the first part of the brain where the initial learning happens. It takes in the input data (like the amount of cement, water, type of cement, etc.) and transforms it into a hidden (internal) representation that the model can learn from.
2. **ReLU**: This part of the brain decides which features are important for making predictions. It takes the output from the first layer and applies a function (ReLU stands for Rectified Linear Unit) that essentially keeps the positive values and discards the negative ones.
3. **Second Layer (fc2)**: This is the final part of the brain that takes the output from the ReLU and transforms it into the final prediction of tensile strength.

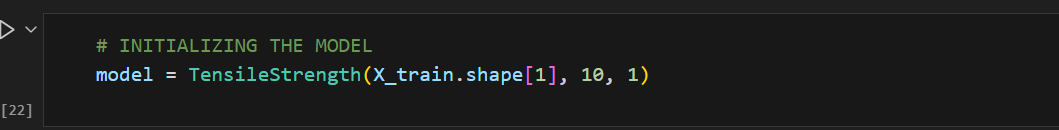
The model summary shows the architecture of our model:



* The first layer (Linear-1) takes in 64 samples each with 10 features and transforms them.
* The ReLU (ReLU-2) applies the ReLU function to the output of the first layer.
* The second layer (Linear-3) takes the output from the ReLU and transforms it into the final prediction.

The model has a total of 101 parameters (weights and biases) that it can learn from.

Finally, we initialize our model with this line of code:



This creates an instance of our TensileStrength model, ready to be trained on our data (X\_train) and make predictions. The numbers inside the parentheses tell the model how many inputs it should expect, how many hidden units to have, and how many outputs to produce. In this case, it expects as many inputs as there are features in X\_train, it has 10 hidden units, and it produces 1 output (the predicted tensile strength).

And that’s how we developed our model! It’s like a little brain that learns from data to make predictions about tensile strength.

## **4.4 Model Training.**

**APPENDIX III EXPERIMENTAL DATA**

Data collected.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Samples | Cement amount (g) | Water (g) | Type of cement | Average Aggregate size (mm) | Aggregate (Coarse)  (g) | Aggregate (SAND) (g) | Curing Duration (days) | Admixtures | Load at Fracture (N) | |
| 1 | 701 | 285 | WHITE CEMENT | 20 | 2828 | 1414 | 7 | Air-Entraining | 80534 |
| 2 | 701 | 500 | OPC | 20 | 2828 | 1414 | 7 |  | 73328 |
| 3 | 701 | 388 | KP Silver | 10 | 2828 | 1414 | 7 |  | 55743 |
| 4 | 701 | 112 | KP Silver | 20 | 2828 | 1414 | 7 | Air-Entraining | 94747 |