tesile-model

April 19, 2024

1 The tensile Model

1.0.1 Libraries

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

1.0.2 Loading the data

[2]:
```python
data = pd.read_csv('Processed_concrete_data.csv')
data.head()
```

[2]:
```
   Samples  Cement amount (g)  Water (g) Type of cement  \
0        1                701        285   WHITE CEMENT
1        2                701        500            OPC
2        3                701        388      KP Silver
3        4                701        112      KP Silver
4        5                701        140            OPC

   Average Aggregate size (mm)  Aggregate(Coarse)(g)  Aggregate(SAND)(g)  \
0                           20                  2828                1414
1                           20                  2828                1414
2                           10                  2828                1414
3                           20                  2828                1414
4                           20                  2828                1414

   Curing Duration (days)      Admixtures  Load at Fracture (N)  \
0                       7  Air-Entraining                 80534
1                       7             NaN                 73328
2                       7             NaN                 55743
3                       7  Air-Entraining                 94747
4                       7  Air-Entraining                 55627

   Tensile Strength (MPa)
0                2.563477
1                2.334103
2                1.774355
```

```
3                 3.015891
4                 1.770662
```

1.0.3 DATA PROCESSING

[3]:
```python
# Summary OF MY DATA
data.describe(include="all").T
```

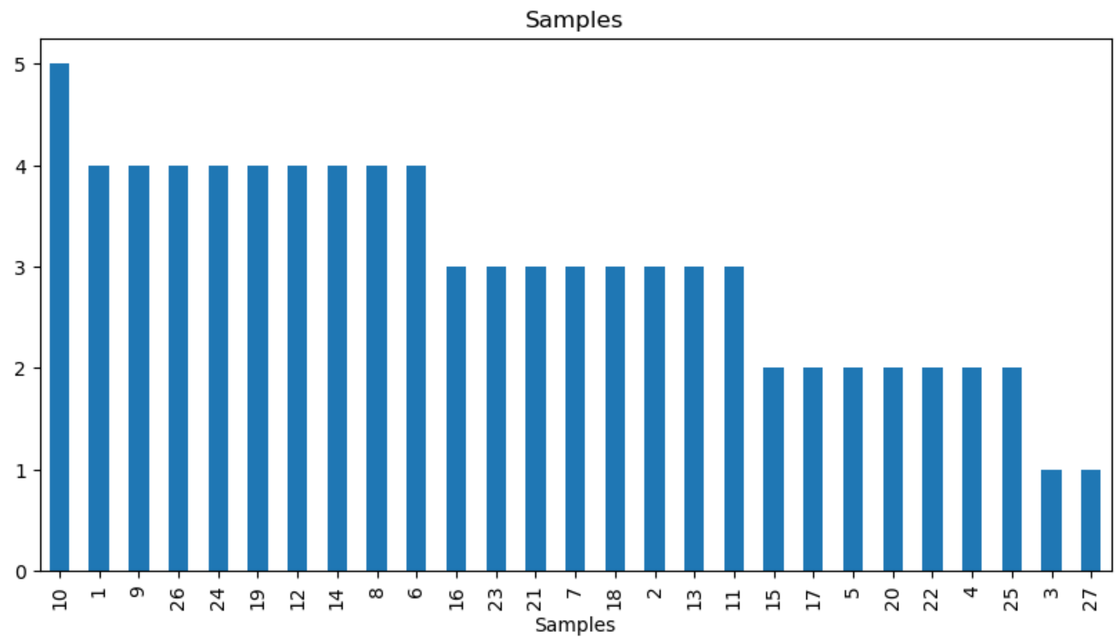
[3]:
```
                             count unique            top freq          mean  \
Samples                       81.0    NaN            NaN  NaN     13.641975
Cement amount (g)             81.0    NaN            NaN  NaN         701.0
Water (g)                     81.0    NaN            NaN  NaN    293.407407
Type of cement                  81      3            OPC   29           NaN
Average Aggregate size (mm)   81.0    NaN            NaN  NaN     11.481481
Aggregate(Coarse)(g)          81.0    NaN            NaN  NaN        2828.0
Aggregate(SAND)(g)            81.0    NaN            NaN  NaN        1414.0
Curing Duration (days)        81.0    NaN            NaN  NaN           7.0
Admixtures                      54      1 Air-Entraining   54           NaN
Load at Fracture (N)          81.0    NaN            NaN  NaN  79986.518519
Tensile Strength (MPa)        81.0    NaN            NaN  NaN       2.54605

                                      std      min       25%       50%  \
Samples                          7.508843      1.0       8.0      13.0
Cement amount (g)                     0.0    701.0     701.0     701.0
Water (g)                      134.055192    111.0     140.0     326.0
Type of cement                        NaN      NaN       NaN       NaN
Average Aggregate size (mm)      6.095308      5.0       5.0      10.0
Aggregate(Coarse)(g)                  0.0   2828.0    2828.0    2828.0
Aggregate(SAND)(g)                    0.0   1414.0    1414.0    1414.0
Curing Duration (days)                0.0      7.0       7.0       7.0
Admixtures                            NaN      NaN       NaN       NaN
Load at Fracture (N)         15222.361135  51371.0   67070.0   80534.0
Tensile Strength (MPa)           0.484543  1.63519  2.134904  2.563477

                                  75%       max
Samples                          20.0      27.0
Cement amount (g)               701.0     701.0
Water (g)                       423.0     500.0
Type of cement                    NaN       NaN
Average Aggregate size (mm)      20.0      20.0
Aggregate(Coarse)(g)           2828.0    2828.0
Aggregate(SAND)(g)             1414.0    1414.0
Curing Duration (days)            7.0       7.0
Admixtures                        NaN       NaN
Load at Fracture (N)          97881.0  100605.0
Tensile Strength (MPa)       3.115649  3.202357
```

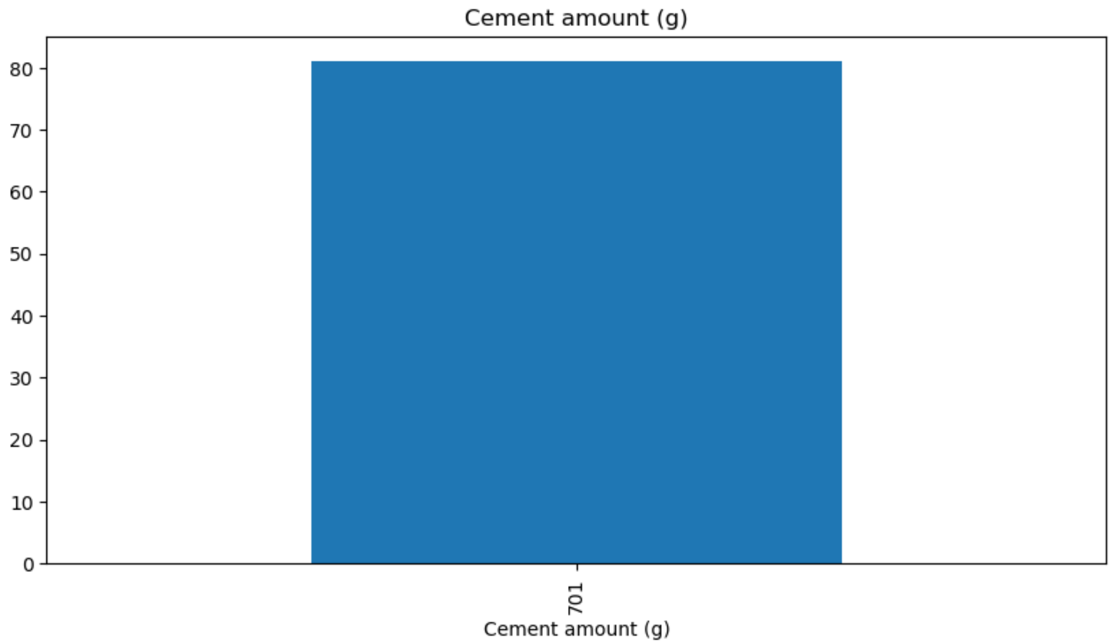
[4]:
```
data.isnull().sum()
```

[4]:
```
Samples                        0
Cement amount (g)              0
Water (g)                      0
Type of cement                 0
Average Aggregate size (mm)    0
Aggregate(Coarse)(g)           0
Aggregate(SAND)(g)             0
Curing Duration (days)         0
Admixtures                    27
Load at Fracture (N)           0
Tensile Strength (MPa)         0
dtype: int64
```

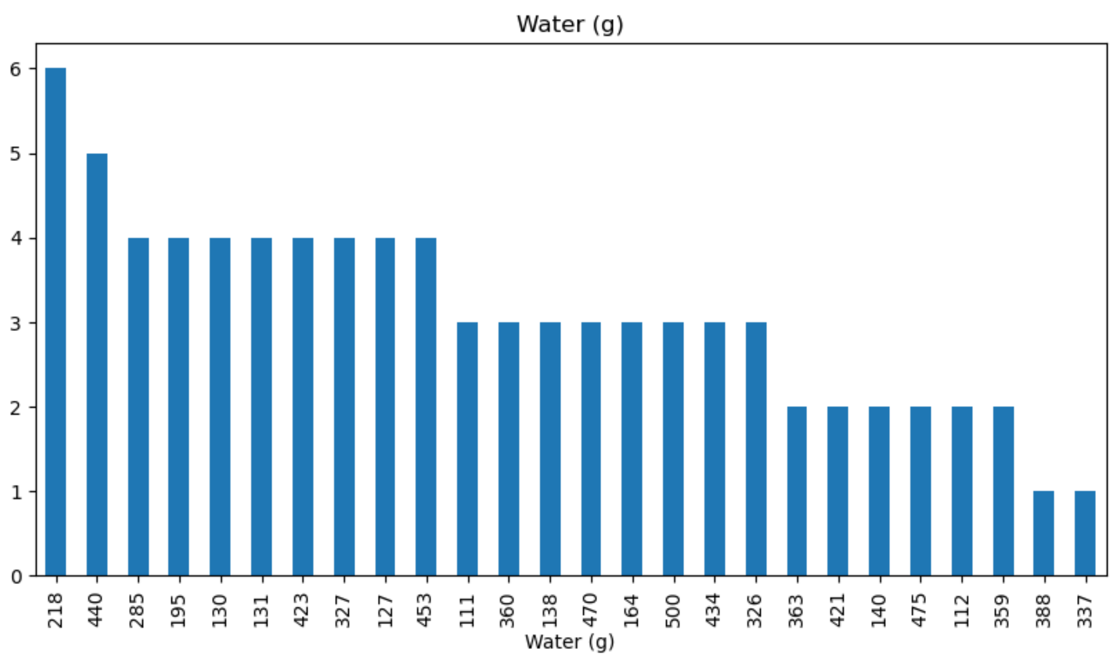
Check if data is normaly distributed

[5]:
```
# Create a bar graph for each column
for column in data.columns:
   plt.figure(figsize=(10, 5))  # Adjust as needed
   data[column].value_counts().plot(kind='bar')
   plt.title(column)
   plt.show()
```

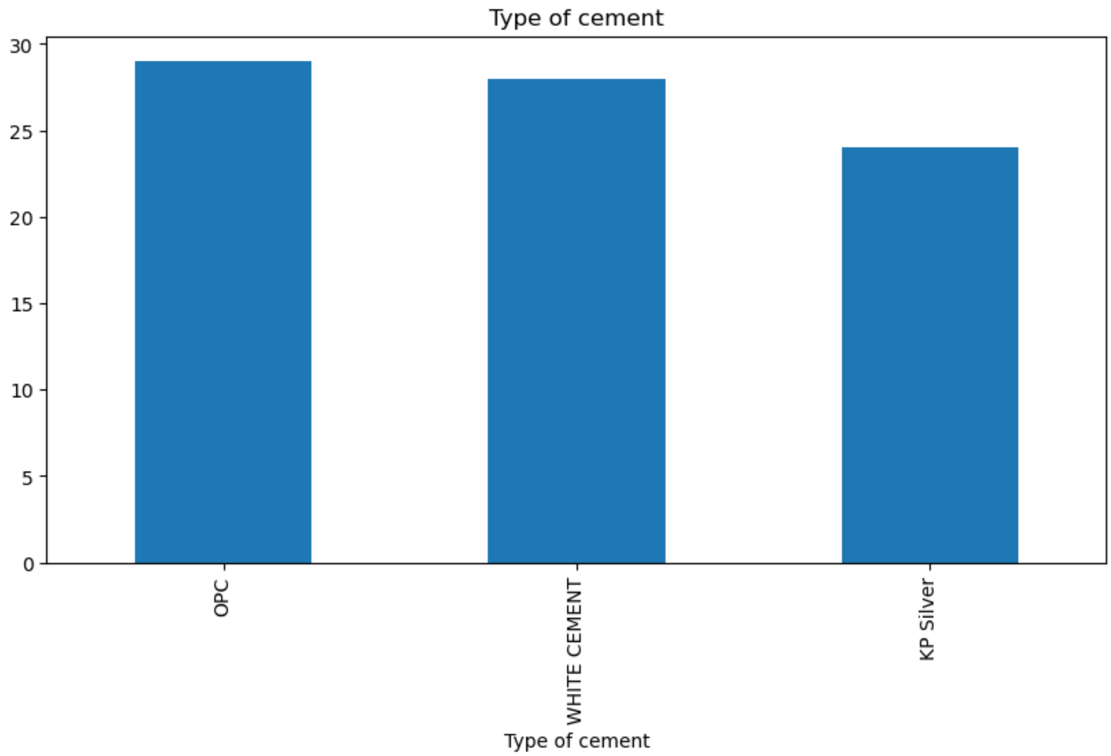




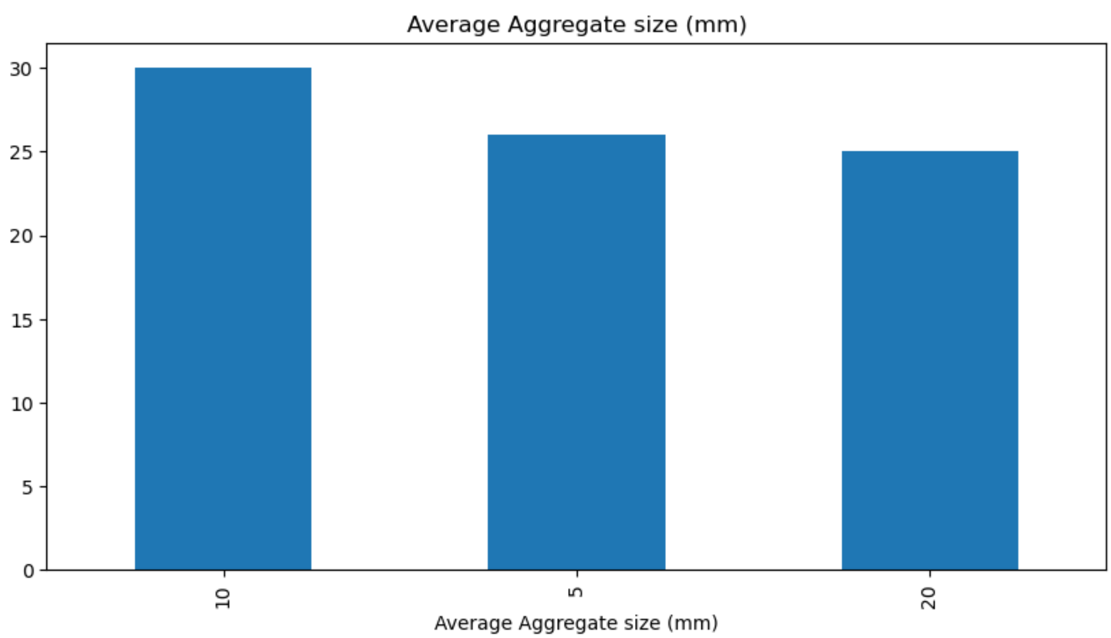
Cement amount (g)
80
70
60
50
40
30
20
10
0
701
Cement amount (g)



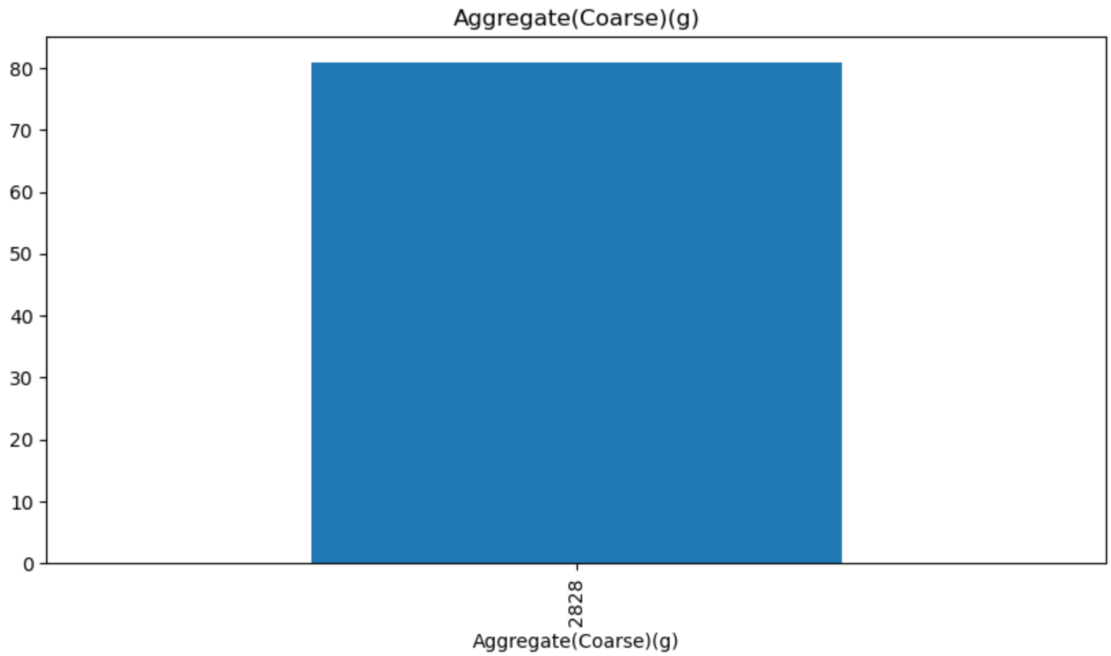
Water (g)
6
5
4
3
2
1
0
218
440
285
195
130
131
423
327
127
453
111
360
138
470
164
500
434
326
363
421
140
475
112
359
388
337
Water (g)

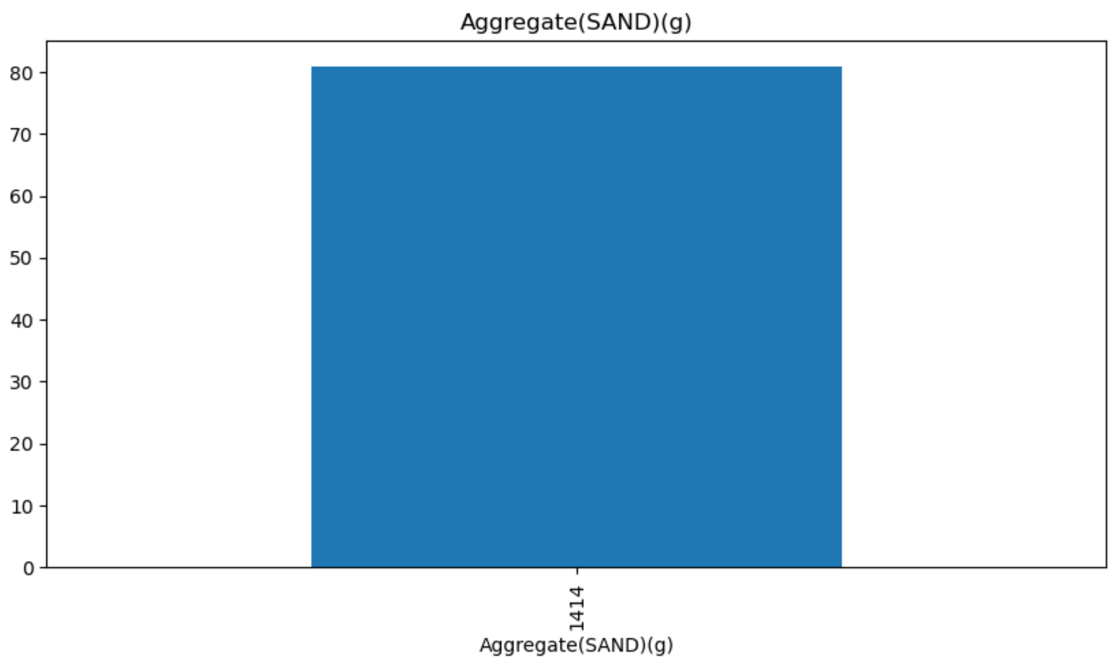


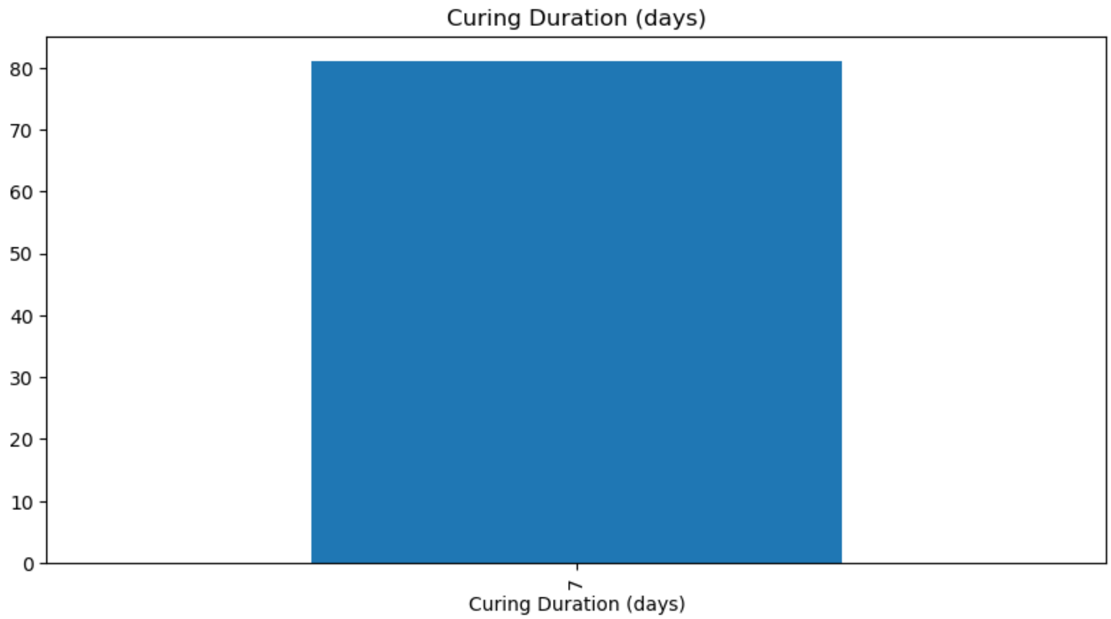
Type of cement
30
25
20
15
10
5
0
OPC
WHITE CEMENT
KP Silver
Type of cement



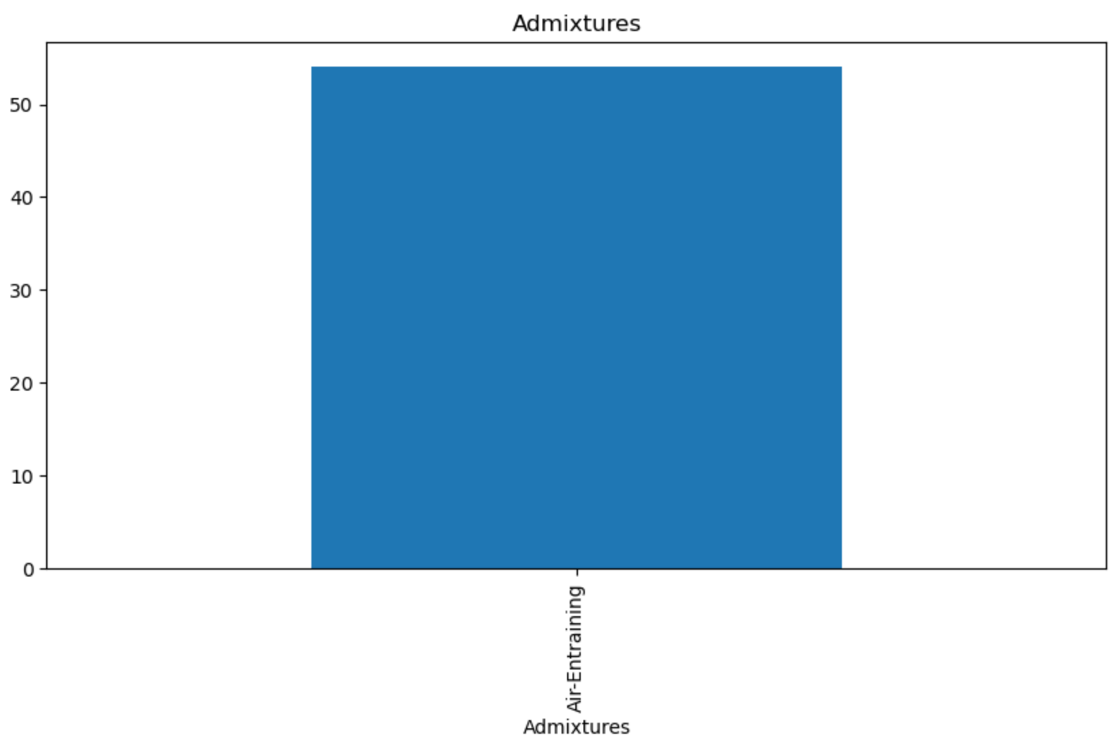
Average Aggregate size (mm)
30
25
20
15
10
5
0
10
5
20
Average Aggregate size (mm)


Aggregate(Coarse)(g)
80
70
60
50
40
30
20
10
0
2828
Aggregate(Coarse)(g)

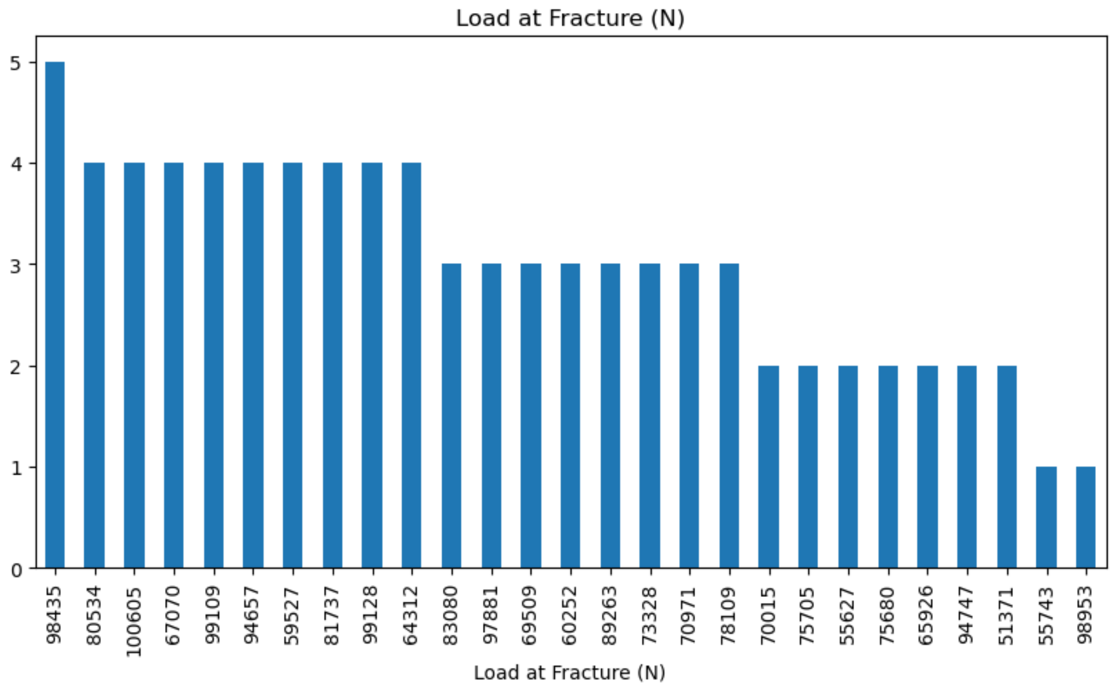

Aggregate(SAND)(g)
80
70
60
50
40
30
20
10
0
1414
Aggregate(SAND)(g)

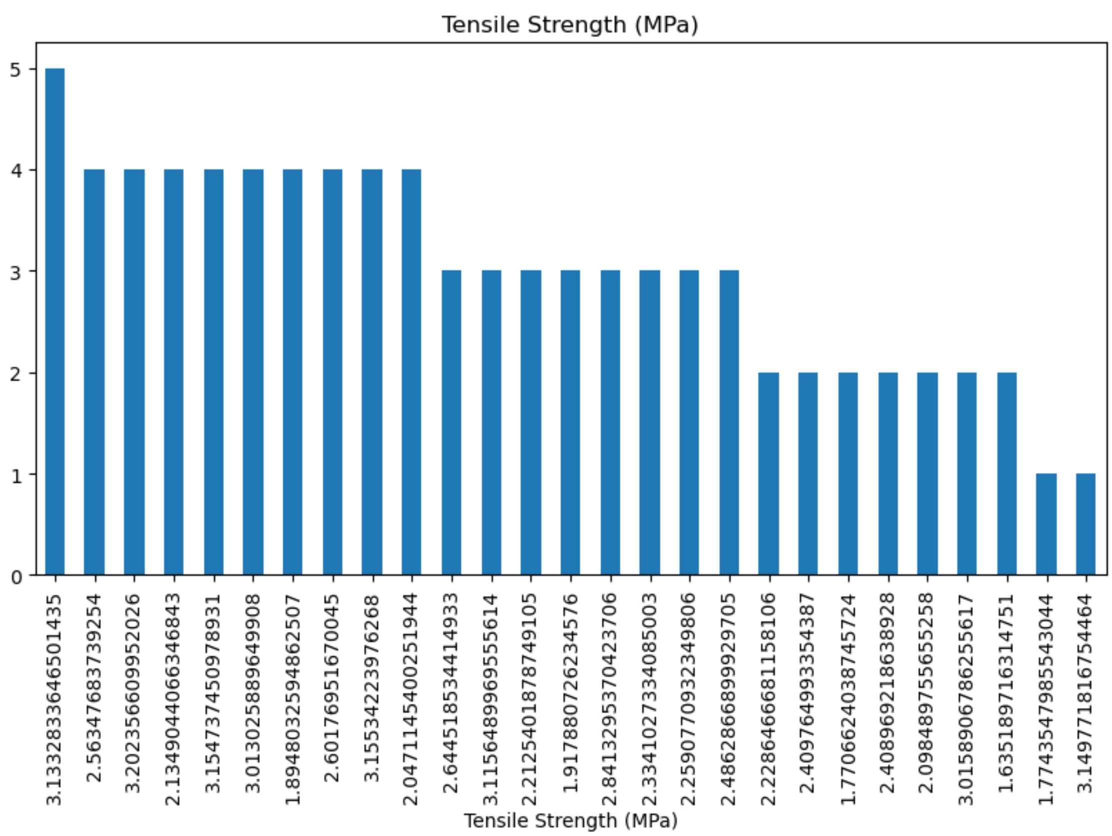


Curing Duration (days)
80
70
60
50
40
30
20
10
0
7
Curing Duration (days)



Admixtures
50
40
30
20
10
0
Air-Entraining
Admixtures


Load at Fracture (N)
98435
80534
100605
67070
99109
94657
59527
81737
99128
64312
83080
97881
69509
60252
89263
73328
70971
78109
70015
75705
55627
75680
65926
94747
51371
55743
98953
Load at Fracture (N)


Tensile Strength (MPa)
Tensile Strength (MPa)

1.0.4 Converting the Null values in Admixtures.

If no admixture was used WRITE IT to No Admixture

[6]:
```
data['Admixtures'] = data['Admixtures'].fillna('No Admixture')
data.isnull().sum()
```

[6]:
```
Samples                       0
Cement amount (g)             0
Water (g)                     0
Type of cement                0
Average Aggregate size (mm)   0
Aggregate(Coarse)(g)          0
Aggregate(SAND)(g)            0
Curing Duration (days)        0
Admixtures                    0
Load at Fracture (N)          0
Tensile Strength (MPa)        0
dtype: int64
```

1.0.5 FEATURE SELECTION

I will do this by removing the columns that I do not need for my model

Like the Samples, Load at Fracture (N)

[7]:
```
data = data.drop(["Samples", "Load at Fracture (N)"], axis=1)
data.head()
```

[7]:
```
   Cement amount (g)  Water (g) Type of cement  Average Aggregate size (mm)  \
0                701        285   WHITE CEMENT                           20
1                701        500            OPC                           20
2                701        388      KP Silver                           10
3                701        112      KP Silver                           20
4                701        140            OPC                           20

   Aggregate(Coarse)(g)  Aggregate(SAND)(g)  Curing Duration (days)  \
0                  2828                1414                       7
1                  2828                1414                       7
2                  2828                1414                       7
3                  2828                1414                       7
4                  2828                1414                       7

       Admixtures  Tensile Strength (MPa)
0  Air-Entraining                2.563477
1    No Admixture                2.334103
2    No Admixture                1.774355
```

```
3   Air-Entraining                 3.015891
4   Air-Entraining                 1.770662
```

1.0.6 ENCODING MY COLUMNS

[8]:
```
data.dtypes
```

[8]:
```
Cement amount (g)                 int64
Water (g)                         int64
Type of cement                   object
Average Aggregate size (mm)       int64
Aggregate(Coarse)(g)              int64
Aggregate(SAND)(g)                int64
Curing Duration (days)            int64
Admixtures                       object
Tensile Strength (MPa)          float64
dtype: object
```

1.1 Encoding

[9]:
```
data['Type of cement'] = data['Type of cement'].replace({"OPC": 0, "KP Silver":␣
↪1, "WHITE CEMENT": 2})
data['Admixtures'] = data['Admixtures'].replace({"Air-Entraining": 0, "No␣
↪Admixture": 1})
data.head(3)
```

[9]:
```
   Cement amount (g)  Water (g)  Type of cement  Average Aggregate size (mm)  \
0                701        285               2                           20
1                701        500               0                           20
2                701        388               1                           10

   Aggregate(Coarse)(g)  Aggregate(SAND)(g)  Curing Duration (days)  \
0                  2828                1414                       7
1                  2828                1414                       7
2                  2828                1414                       7

   Admixtures  Tensile Strength (MPa)
0           0                2.563477
1           1                2.334103
2           1                1.774355
```

The data is NOW PERFECT FOR Training my model

2 THE PYTORCH MODEL

[10]:
```
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from torch import nn, optim
import pandas as pd
```

[11]:
```
from sklearn.preprocessing import StandardScaler
# Assuming df is your DataFrame and that it's already been preprocessed
X = data.drop('Tensile Strength (MPa)', axis=1).values
y = data['Tensile Strength (MPa)'].values

# Initialize a scaler
scaler = StandardScaler()

# Fit on the features and transform
normalized_features = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(normalized_features, y,␣
↪test_size=0.2, random_state=42)

# Convert the data to PyTorch tensors
X_train = torch.FloatTensor(X_train)
y_train = torch.FloatTensor(y_train)
X_test = torch.FloatTensor(X_test)
y_test = torch.FloatTensor(y_test)
```

2.1 The Model

[12]:
```
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn, optim
# custom PyTorch model class
class TensileStrength(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(TensileStrength, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```
[13]: # INITIALIZING THE MODEL
model = TensileStrength(X_train.shape[1], 10, 1)
```

2.1.1 Model Achitecture

```
[14]: from torchsummary import summary
summary(model, X_train.shape)
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 64, 10]              90
              ReLU-2               [-1, 64, 10]               0
            Linear-3                [-1, 64, 1]              11
================================================================
Total params: 101
Trainable params: 101
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 0.00
Estimated Total Size (MB): 0.01
----------------------------------------------------------------
```

2.1.2 LOSS FUNCTION AND MODEL OPTIMIZATION

```
[15]: criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), weight_decay=1e-5)
```

2.2 Training

```
[16]: num_epochs = 1000
losses = []  # to store the loss values
streak = 0  # to count the streak of constant loss
# Training loop
for epoch in range(num_epochs):
    # Forward pass
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    # Backward and optimize
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    # Save the loss
    losses.append(loss.item())
    # Check if the loss is constant
```

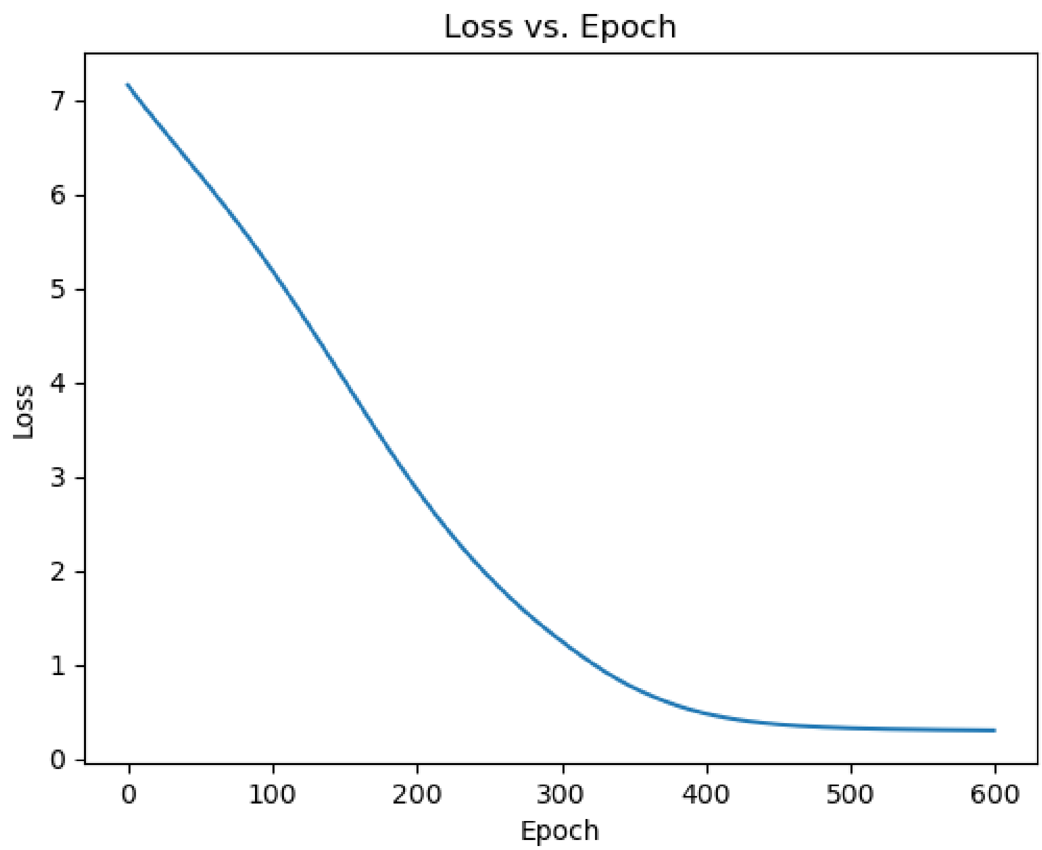
```
    if len(losses) > 1 and losses[-1] == losses[-2]:
        streak += 1
    else:
        streak = 0
    # Stop training if the loss is constant for 10 epochs
    if streak >= 20:
        print('Loss is constant for 10 epochs, stopping training')
        break
    if (epoch+1) % 100 == 0:
        print ('Epoch [{}/{}], Loss: {:.4f}'.format(epoch+1, num_epochs, loss.
↪item()))
```

```
c:\Users\Administrator\anaconda3\Lib\site-packages\torch\nn\modules\loss.py:535:
UserWarning: Using a target size (torch.Size([64])) that is different to the
input size (torch.Size([64, 1])). This will likely lead to incorrect results due
to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
Epoch [100/600], Loss: 5.2074
Epoch [200/600], Loss: 2.8805
Epoch [300/600], Loss: 1.2594
Epoch [400/600], Loss: 0.4812
Epoch [500/600], Loss: 0.3235
Epoch [600/600], Loss: 0.2996
```

2.2.1 Loss Vs Epochs

[17]:
```
# After the training loop
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs. Epoch')
plt.show()
```



2.2.2 Model evaluation

[18]:
```
model.eval()
with torch.no_grad():
    y_pred = model(X_test)
    ss_res = torch.sum((y_test - y_pred) ** 2)
    ss_tot = torch.sum((y_test - torch.mean(y_test)) ** 2)
    r2_score = 1 - ss_res/ss_tot
    print(f"R-squared: {r2_score.item()}")
```

```
R-squared: -22.863006591796875
```

[19]:
```
model.eval()
with torch.no_grad():
    y_pred = model(X_test)
    mse = torch.mean((y_test - y_pred) ** 2)
print(f"Mean Squared Error: {mse.item() * 100} %")
```

```
Mean Squared Error: 31.357023119926453 %
```

2.2.3 Save the model

[21]:
```
# Save the model
torch.save(model, 'Model/copytensile_strength_model.pth')
```

[ ]:
```

```