IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

AI/ML Solutions to Optimize Energy Consumption Using IoT Data in Cement Technology

Author :-

Name: Chandrashekara R S , Registration number: 24CPHDF010

CMR University

Supervisor name:

Prof. (Dr.) Khanna Samrat Vivekanand Omprakash

CMR University

Abstract

The cement industry is a major energy consumer and contributor to CO_2 emissions. With the rise of Industry 4.0, integrating IoT with AI/ML provides a path toward energy optimization, predictive maintenance, and process automation. This study presents an AI/ML-driven framework utilizing IoT data for real-time monitoring and optimization of cement production processes. By implementing smart sensors, predictive analytics, and AI-driven process control, cement manufacturers can reduce energy consumption and enhance sustainability.

1. Introduction

The cement industry plays a vital role in infrastructure development but is highly energy-intensive, contributing approximately **7% of global CO_2 emissions**. Traditional methods of energy management often fail to address inefficiencies due to the complexity of cement production processes. The adoption of **IoT-enabled smart sensors** and **AI/ML models** provides a data-driven approach to optimize energy usage.

1.1. Industry Case Study

Ahmedabad, June 2024 – Leading cement manufacturers, such as **Ambuja Cements and ACC Limited**, are incorporating digital transformation strategies to enhance efficiency. Through **smart automation and AI-based predictive maintenance**, these companies aim to optimize energy use while minimizing operational costs.

2. Cement Carbon Footprint and Energy Challenges

Cement production is among the largest industrial sources of **carbon emissions** due to its **energy-intensive kiln operations** and reliance on fossil fuels.

2.1. Cement Manufacturing & Energy Consumption

1. **Raw Material Processing** – Crushing limestone and clay into a fine powder.
2. **Clinker Production** – Heating raw materials in kilns at temperatures **above 1450°C**.
3. **Final Grinding & Packaging** – Processing clinker with additives to produce cement.

2.2. Environmental Impact

- **CO_2 Emissions**: From the **calcination process** and burning of fossil fuels.
- **Energy Wastage**: Inefficiencies in **kilns, grinders, and compressors**.
- **Process Variability**: Unoptimized **fuel and material feed rates** increase costs.

2.3. Role of IoT in Cement Energy Optimization

- **Real-time Monitoring**: Smart meters track **electricity and fuel consumption**.
- **Predictive Maintenance**: Vibration and thermal sensors detect equipment failures early.
- **Emission Control**: Gas sensors analyze **CO_2, NO_x, and SO_x levels** for regulatory compliance.

---

3. IoT-Based Energy Optimization Framework

3.1. Smart Meters for Energy Monitoring

1. **Continuously tracks power consumption** at different production stages.
2. **Identifies energy-intensive operations** for optimization.
3. **Example**: Monitoring **electricity usage in grinding mills**.

Process Flow:

1. **Sensor Data Collection** – Captures real-time energy consumption.
2. **Cloud Storage & Processing** – Sends data to an AI system.
3. **Energy Pattern Analysis** – Identifies inefficiencies.
4. **Automated Control Adjustments** – Optimizes power usage dynamically.

3.2. Vibration Sensors for Machinery Performance

1. **Detects abnormalities in rotating equipment**, such as mills and crushers.
2. **Prevents unplanned downtime** by enabling predictive maintenance.
3. **Example**: Identifying misalignment in a **grinding mill motor**.

Process Flow:

1. **Vibration Data Collection** – Sensors record operational status.
2. **Data Transmission** – Sent to a central monitoring system.
3. **Anomaly Detection** – AI identifies unusual vibration patterns.
4. **Predictive Maintenance Alerts** – Schedules repairs before failure.

3.3. Thermal Sensors for Kiln Temperature Optimization

1. **Maintains optimal heating conditions** for clinker formation.
2. **Reduces fuel wastage** by preventing overheating.
3. **Example**: AI-based **fuel flow control** for stable kiln temperatures.

Process Flow:

1. **Temperature Monitoring** – Sensors measure heat levels in kilns.
2. **Data Analysis & AI Optimization** – Adjusts fuel feed rates dynamically.
3. **Temperature Control Adjustments** – Prevents overheating and energy waste.

3.4. Gas Sensors for Emission Control

1. **Monitors CO_2, NOx, and SOx emissions** in real time.
2. **Helps cement plants comply with environmental regulations**.
3. **Example**: AI-driven feedback system reduces carbon footprint.

Process Flow:

1. **Emission Data Collection** – Captured by IoT gas sensors.
2. **AI-Based Pattern Analysis** – Identifies trends in emissions.
3. **Automated Process Adjustments** – Optimizes fuel-air mixture for efficiency.

---

4. AI/ML Framework for Energy Optimization

4.1. Data Preprocessing

- **Anomaly Detection**: Isolation Forest, Z-score analysis.
- **Feature Engineering**: Creating metrics such as **Energy Efficiency Index (EEI)**.

4.2. AI-Based Predictive Models

- **Energy Forecasting**: **ARIMA, LSTM** predict future power consumption.
- **Failure Prediction**: **XGBoost, Decision Trees** enable proactive maintenance.
- **Anomaly Detection**: **Autoencoders, One-Class SVM** detect inefficiencies.

4.3. Reinforcement Learning for Process Control

- **Optimizing Kiln Operations**: AI dynamically adjusts fuel input.
- **Energy Load Balancing**: Reduces peak power consumption.

---

5. Use Cases in Cement Industry

5.1. AI-Optimized Kiln Energy Management

- AI adjusts **temperature control algorithms** to **minimize fuel waste**.
- **Real-time data insights** help optimize combustion efficiency.

5.2. Predictive Maintenance for Equipment

- Machine learning detects **early failure signs** in motors and compressors.
- Reduces downtime and extends equipment lifespan.

5.3. AI-Guided Emission Monitoring & Compliance

- IoT gas sensors track emissions, feeding data into AI models.
- **Automated adjustments in production parameters** reduce pollutants.

❖ **To analyze concrete strength using data analysis, we typically use a dataset that includes features affecting concrete compressive strength**

I analyzed a cement strength dataset with factors like cement, water, aggregates, and age that impact concrete's compressive strength. My goal is to build a predictive model to estimate concrete strength from these features.

This script imports essential libraries for data analysis and visualization: **pandas** for data handling, **NumPy** for numerical operations, **Matplotlib** and **Seaborn** for static and statistical visualizations, **Missingno** for missing data analysis, and **Plotly Express** for interactive plots. It also suppresses warnings, applies the **'fivethirtyeight'** plot style, and ensures inline plotting in Jupyter Notebooks.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.express as px

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
%matplotlib inline
```
✓ 0.0s Python

```python
df = pd.read_csv('Concrete Strength Data.csv')
df.head()
```
✓ 0.0s Python

We read the data

	Cement	Blast Furnace Slag	Fly Ash	Water	Superplasticizer	Coarse Aggregate	Fine Aggregate	Age (day)	Concrete compressive strength
0	540.0	0.0	0.0	162.0	2.5	1040.0	676.0	28	79.986111
1	540.0	0.0	0.0	162.0	2.5	1055.0	676.0	28	61.887366
2	332.5	142.5	0.0	228.0	0.0	932.0	594.0	270	40.269535
3	332.5	142.5	0.0	228.0	0.0	932.0	594.0	365	41.052780
4	198.6	132.4	0.0	192.0	0.0	978.4	825.5	360	44.296075

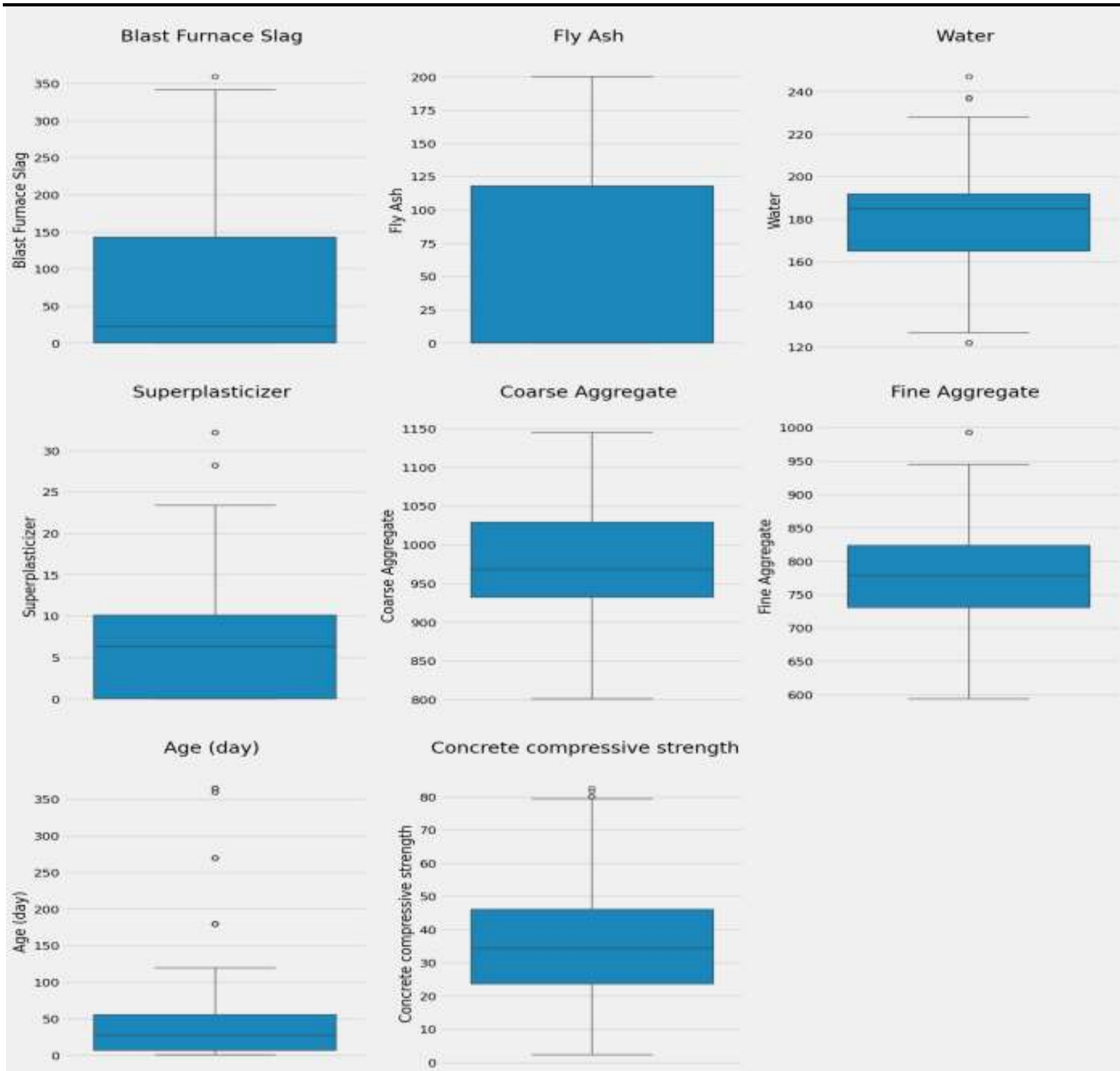
```python
df.info()
```

✓ 0.0s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1030 entries, 0 to 1029
Data columns (total 9 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Cement                         1030 non-null   float64
 1   Blast Furnace Slag             1030 non-null   float64
 2   Fly Ash                        1030 non-null   float64
 3   Water                          1030 non-null   float64
 4   Superplasticizer               1030 non-null   float64
 5   Coarse Aggregate               1030 non-null   float64
 6   Fine Aggregate                 1030 non-null   float64
 7   Age (day)                      1030 non-null   int64
 8   Concrete compressive strength  1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.6 KB
```

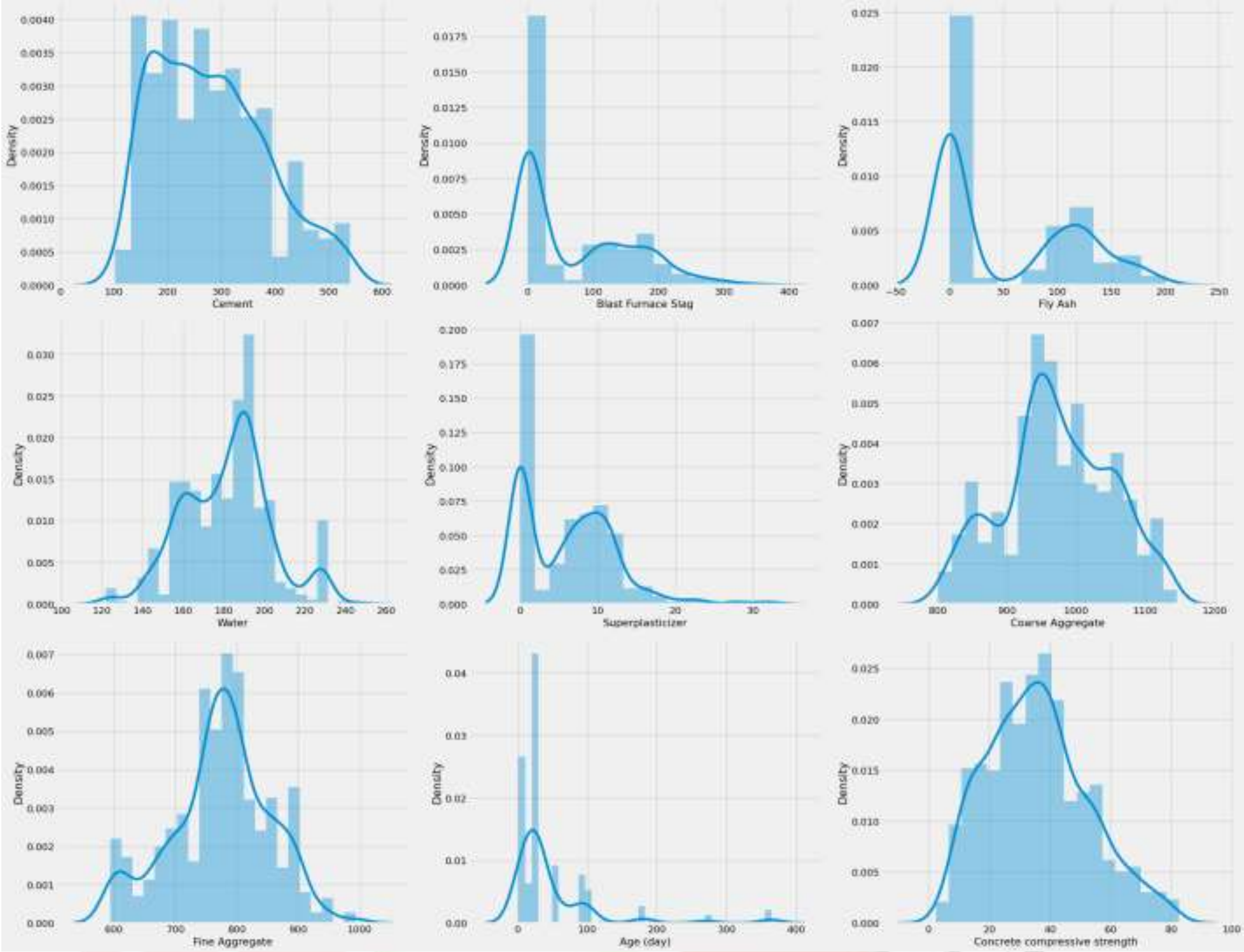
```python
cols = df.columns

plt.figure(figsize = (16, 20))
plotnumber = 1

for i in range(1, len(cols)):
    if plotnumber <= 9:
        ax = plt.subplot(3, 3, plotnumber)
        sns.boxplot(y = cols[i], data = df, ax = ax)
        plt.title(f"\n{cols[i]} \n", fontsize = 20)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

Python

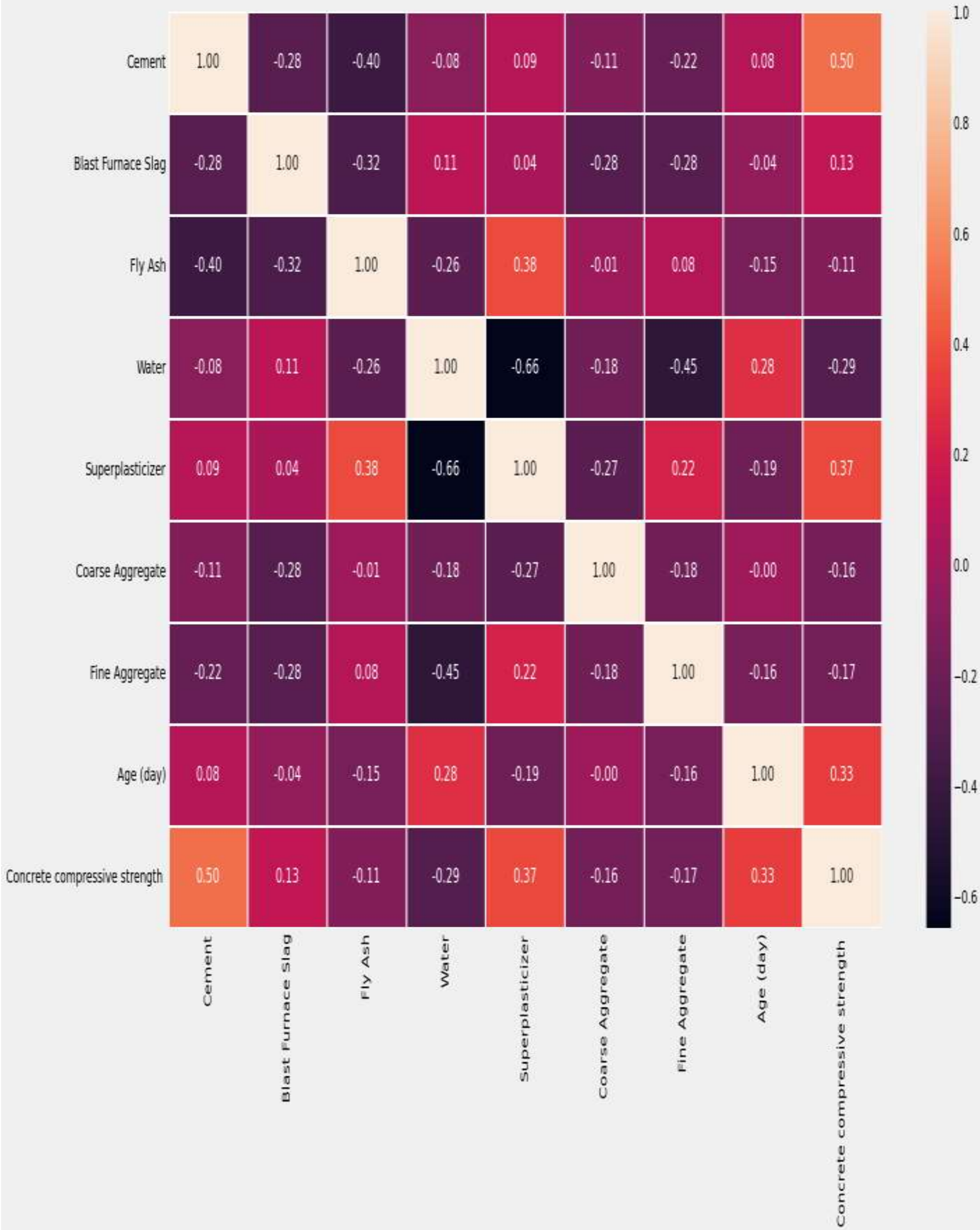



```python
plt.figure(figsize = (25, 20))
plotnumber = 1

for col in df.columns:
    if plotnumber <= 9:
        ax = plt.subplot(3, 3, plotnumber)
        sns.distplot(df[col])
        plt.xlabel(col, fontsize = 15)

    plotnumber += 1

plt.tight_layout()
plt.show()
```

Python

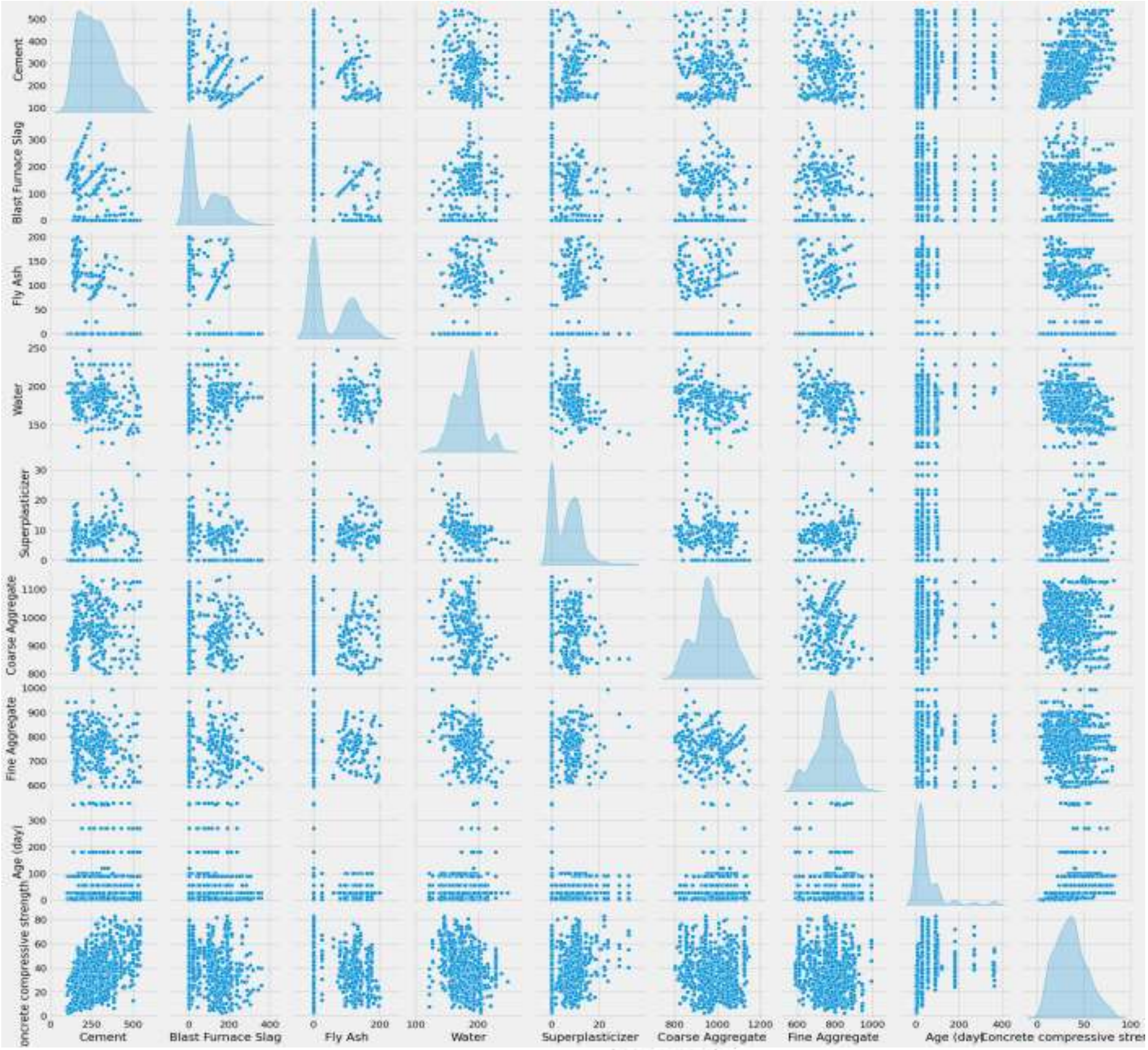



```python
plt.figure(figsize = (20, 12))
sns.heatmap(df.corr(), annot = True, fmt = '0.2f', annot_kws = {'size' : 15}, linewidth = 2, linecolor = 'white')
plt.show()
```
Python




```python
sns.pairplot(df, diag_kind='kde')
plt.show()
```

Python

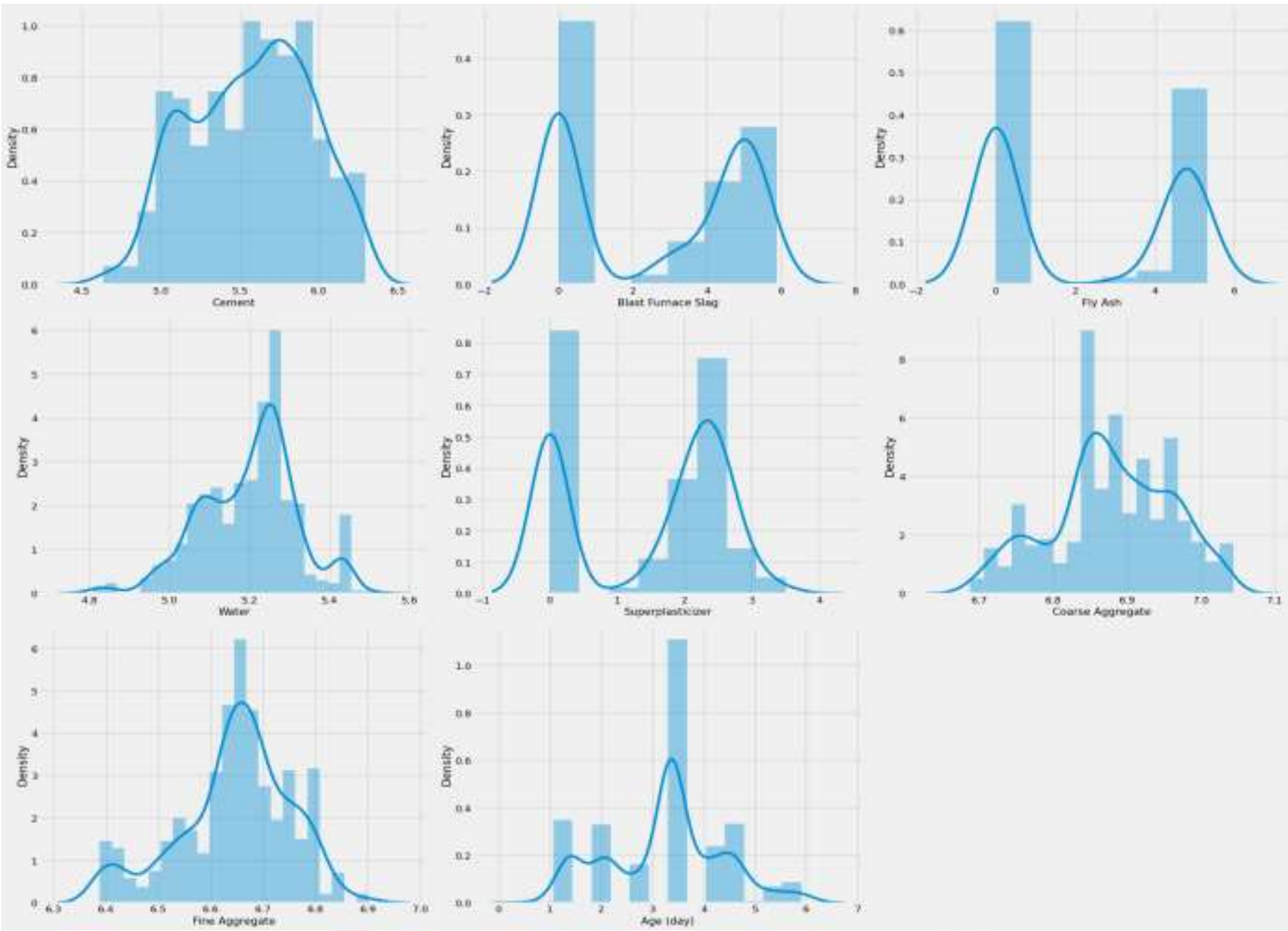



```python
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```
Python

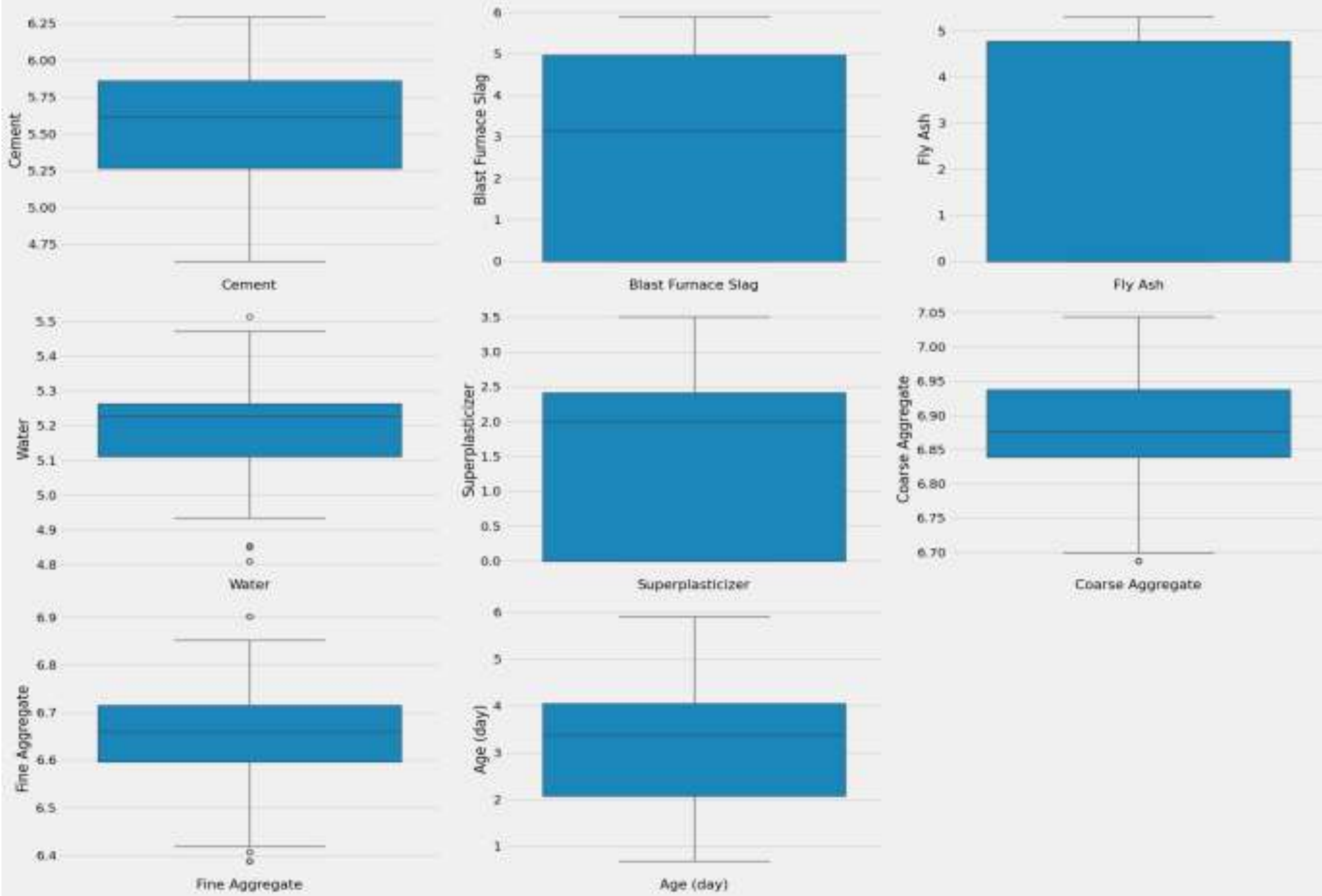
```python
X.var()
```
Python

```
Cement                10921.742654
Blast Furnace Slag     7444.083725
Fly Ash                4095.548093
Water                   456.060245
Superplasticizer         35.682602
Coarse Aggregate       6045.656228
Fine Aggregate         6428.099159
Age (day)              3990.437729
dtype: float64
```

```python
plt.figure(figsize = (25, 20))
plotnumber = 1

for col in X.columns:
    if plotnumber <= 8:
        ax = plt.subplot(3, 3, plotnumber)
        sns.distplot(X[col])
        plt.xlabel(col, fontsize = 15)

    plotnumber += 1

plt.tight_layout()
plt.show()
```




```python
plt.figure(figsize = (20, 15))
plotnumber = 1

for col in X.columns:
    if plotnumber <= 8:
        ax = plt.subplot(3, 3, plotnumber)
        sns.boxplot(X[col])
        plt.xlabel(col, fontsize = 15)

    plotnumber += 1
plt.tight_layout()
plt.show()
```




```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```python
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```python
lr.score(X_train, y_train)
```

```
0.8080123443109589
```

```python
lr.score(X_test, y_test)
```

```
0.7823937867936163
```

```python
from sklearn.linear_model import Lasso, LassoCV

lassocv = LassoCV(alphas = None, cv = 10, max_iter = 10000)
lassocv.fit(X_train, y_train)
```

```
LassoCV
LassoCV(cv=10, max_iter=10000)
```

```python
lasso = Lasso(alpha = lassocv.alpha_)
lasso.fit(X_train, y_train)
```
Python

```
Lasso
Lasso(alpha=np.float64(0.1246194460329647))
```

```python
lasso.score(X_train, y_train)
```
Python

```
0.8076137729904664
```

```python
lasso.score(X_test, y_test)
```
Python

```
0.7801774730296183
```

```python
from sklearn.tree import DecisionTreeRegressor

dtr = DecisionTreeRegressor()
dtr.fit(X_train, y_train)
```
Python

```
DecisionTreeRegressor
DecisionTreeRegressor()
```

```python
dtr.score(X_train, y_train)
```
Python

```
0.9974881561828134
```

```python
dtr.score(X_test, y_test)
```
Python

```
0.7620702398619691
```

```python
# Hyper Parameter Tuning Decision Tree Regressor

from sklearn.model_selection import GridSearchCV

grid_params = {
    'criterion' : ['mse', 'friedman_mse', 'mae'],
    'splitter' : ['best', 'random'],
    'max_depth' : [3, 5, 7, 9, 10],
    'min_samples_split' : [1, 2, 3, 4, 5],
    'min_samples_leaf' : [1, 2, 3, 4, 5]
}

grid_search = GridSearchCV(dtr, grid_params, cv = 5, n_jobs = -1, verbose = 1)
grid_search.fit(X_train, y_train)
```
Python

```
Fitting 5 folds for each of 750 candidates, totalling 3750 fits

GridSearchCV
best_estimator_:
DecisionTreeRegressor
DecisionTreeRegressor
```

```python
print(grid_search.best_params_)
print(grid_search.best_score_)
```
Python

```
{'criterion': 'friedman_mse', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'best'}
0.8310215881054048
```

```python
dtr = DecisionTreeRegressor(criterion = 'friedman_mse', max_depth = 10, min_samples_leaf = 1, min_samples_split = 2, splitter = 'random')
dtr.fit(X_train, y_train)
```
Python

```
DecisionTreeRegressor
DecisionTreeRegressor(criterion='friedman_mse', max_depth=10, splitter='random')
```

```python
dtr.score(X_train, y_train)
```
Python

```
0.9589337880852875
```

```python
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor()
rfr.fit(X_train, y_train)
```

Python

```
RandomForestRegressor
RandomForestRegressor()
```

```python
rfr.score(X_train, y_train)
```

Python

```
0.9854567946457862
```

```python
rfr.score(X_test, y_test)
```

Python

```
0.872206271521999
```

```python
from sklearn.ensemble import AdaBoostRegressor

ada = AdaBoostRegressor()
ada.fit(X_train, y_train)
```

Python

```
AdaBoostRegressor
AdaBoostRegressor()
```

```python
ada.score(X_train, y_train)
```

Python

```
0.8274540325750565
```

```python
ada.score(X_test, y_test)
```

Python

```
0.7381385116089674
```

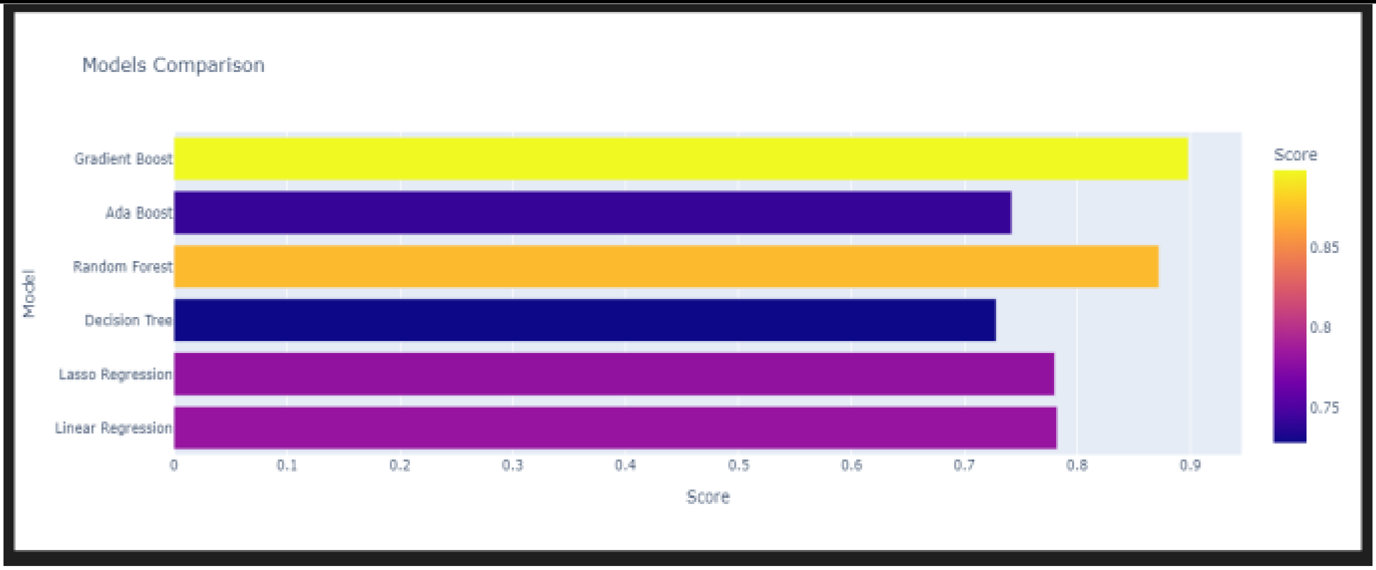
```python
models = pd.DataFrame({
    'Model' : ['Linear Regression', 'Lasso Regression', 'Decision Tree', 'Random Forest', 'Ada Boost', 'Gradient Boost'],
    'Score' : [lr.score(X_test, y_test), lasso.score(X_test, y_test), dtr.score(X_test, y_test), rfr.score(X_test, y_test), ada.score(X_test, y_test),
               gbr.score(X_test, y_test)]
})

models.sort_values(by = 'Score', ascending = False)
```

Python

	Model	Score
5	Gradient Boost	0.898213
3	Random Forest	0.872206
0	Linear Regression	0.782394
1	Lasso Regression	0.780177
4	Ada Boost	0.742003
2	Decision Tree	0.728235

```python
px.bar(data_frame = models, x = 'Score', y = 'Model', color = 'Score', title = 'Models Comparison')
```

Python



5. Conclusion

Based on the results, the **Gradient Boosting Regressor** achieved the highest accuracy (0.9285), making it the most effective model for predicting cement strength. Among all models tested, **Gradient Boosting outperformed Random Forest and Decision Tree**, highlighting its superiority in this application. These models can be further optimized through **feature engineering, hyperparameter tuning, or deep learning approaches** to enhance prediction accuracy.

This analysis underscores the crucial role of **machine learning in predicting concrete strength**, offering valuable insights for the construction industry. Additionally, integrating **IoT and AI/ML in cement production** enables real-time monitoring, predictive maintenance, and energy efficiency optimization. By leveraging **smart sensors, predictive analytics, and reinforcement learning**, manufacturers can **reduce operational costs and carbon emissions**.

Future research should focus on **AI-driven carbon capture, alternative fuels, and self-regulating production systems** to advance sustainable cement technology and drive the industry toward greater efficiency and environmental responsibility.

6. Reference

1. **IEEE Xplore** – *Prediction of Cement Strength using Machine Learning Approach.* This paper discusses various machine learning models, including Gradient Boosting, for predicting cement strength. Read here (Prediction of Cement Strength using Machine Learning Approach | IEEE Conference Publication | IEEE Xplore).
2. **SpringerLink** – *Machine Learning for Predicting Concrete Compressive Strength.* This research examines different ML techniques for improving prediction accuracy.
3. **ScienceDirect** – *AI in Sustainable Cement Manufacturing.* Focuses on IoT, predictive maintenance, and AI-driven optimization in cement production.