In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import plotly.express as px
warnings.filterwarnings('ignore')
```

In [2]:
```python
data = pd.read_csv("https://raw.githubusercontent.com/rashakil-ds/Public-Datasets/m
```

In [3]:
```python
data.head(5)
```

Out[3]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Ag (co 7
0	540.0	0.0	0.0	162.0	2.5	1040.0	
1	540.0	0.0	0.0	162.0	2.5	1055.0	
2	332.5	142.5	0.0	228.0	0.0	932.0	
3	332.5	142.5	0.0	228.0	0.0	932.0	
4	198.6	132.4	0.0	192.0	0.0	978.4	

Data Preprocessing

Null Values

In [4]:
```python
df = data.copy()
```

In [5]:
```python
df.isna().sum()
```

Out[5]:
```
Cement (component 1)(kg in a m^3 mixture)              0
Blast Furnace Slag (component 2)(kg in a m^3 mixture)   0
Fly Ash (component 3)(kg in a m^3 mixture)              0
Water  (component 4)(kg in a m^3 mixture)               0
Superplasticizer (component 5)(kg in a m^3 mixture)     0
Coarse Aggregate  (component 6)(kg in a m^3 mixture)    0
Fine Aggregate (component 7)(kg in a m^3 mixture)       0
Age (day)                                               0
strength                                                0
dtype: int64
```

Duplicate Values

In [6]:
```
df.duplicated().sum()
```

Out[6]:
```
25
```

In [7]:
```
df.drop_duplicates(inplace = True)
```

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
0
```

Scaling

In [9]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [10]:
```
scaler = MinMaxScaler()
```

In [11]:
```
df.dtypes
```

Out[11]:
```
Cement (component 1)(kg in a m^3 mixture)                float64
Blast Furnace Slag (component 2)(kg in a m^3 mixture)    float64
Fly Ash (component 3)(kg in a m^3 mixture)               float64
Water  (component 4)(kg in a m^3 mixture)                float64
Superplasticizer (component 5)(kg in a m^3 mixture)      float64
Coarse Aggregate  (component 6)(kg in a m^3 mixture)     float64
Fine Aggregate (component 7)(kg in a m^3 mixture)        float64
Age (day)                                                  int64
strength                                                 float64
dtype: object
```

In [12]:
```
for col_name in df.columns:
    if col_name != "strength":
        df[col_name] = scaler.fit_transform(df[[col_name]])
```

In [13]:
```
df.head(5)
```

Out[13]:

	Cement (component 1)(kg in a m^3 mixture)	Blast Furnace Slag (component 2)(kg in a m^3 mixture)	Fly Ash (component 3)(kg in a m^3 mixture)	Water (component 4)(kg in a m^3 mixture)	Superplasticizer (component 5)(kg in a m^3 mixture)	Coarse Aggregate (component 6)(kg in a m^3 mixture)	Ag (co 7
0	1.000000	0.000000	0.0	0.321086	0.07764	0.694767	
1	1.000000	0.000000	0.0	0.321086	0.07764	0.738372	
2	0.526256	0.396494	0.0	0.848243	0.00000	0.380814	
3	0.526256	0.396494	0.0	0.848243	0.00000	0.380814	
4	0.220548	0.368392	0.0	0.560703	0.00000	0.515698	

Data Spliting

In [14]:
```
from sklearn.model_selection import train_test_split
```

In [15]:
```
X = df.drop("strength",axis=1)
```

In [16]:
```
y = df["strength"]
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta
```

Model Training

In [18]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from xgboost import XGBRegressor
```

In [19]:
```
lr = LinearRegression()
rfr = RandomForestRegressor()
xgb = XGBRegressor()
```

In [20]:
```
## Fit
```

In [21]:
```
lr.fit(X_train,y_train)
```

Out[21]:

▾ LinearRegression

LinearRegression()

In [22]:
```python
rfr.fit(X_train,y_train)
```
Out[22]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [23]:
```python
xgb.fit(X_train,y_train)
```
Out[23]:
```
▾ XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=N
one,
             enable_categorical=False, eval_metric=None, feature_types=N
one,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=N
one,
```

In [24]:
```python
## predict
```

In [25]:
```python
predicted_lr_train = lr.predict(X_train)
```

In [26]:
```python
predicted_lr_test = lr.predict(X_test)
```

In [27]:
```python
predicted_rfr_train = rfr.predict(X_train)
```

In [28]:
```python
predicted_rfr_test = rfr.predict(X_test)
```

In [29]:
```python
predicted_xgb_train = xgb.predict(X_train)
```

In [30]:
```python
predicted_xgb_test = xgb.predict(X_test)
```

Scoring

In [31]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

In [32]:
```python
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predicted_lr_test)
print(f'Mean Absolute Error (MAE): {mae}')

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predicted_lr_test)
print(f'Mean Squared Error (MSE): {mse}')

# Root Mean Squared Error (RMSE)
```

```python
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

# R-squared (R2) score
r2 = r2_score(y_test, predicted_lr_test)
print(f'R-squared (R2) score: {r2}')
```

```
Mean Absolute Error (MAE): 8.984068581174695
Mean Squared Error (MSE): 126.20929107403163
Root Mean Squared Error (RMSE): 11.234290857639017
R-squared (R2) score: 0.5609151111173311
```

In [33]:

```python
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predicted_rfr_test)
print(f'Mean Absolute Error (MAE): {mae}')

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predicted_rfr_test)
print(f'Mean Squared Error (MSE): {mse}')

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

# R-squared (R2) score
r2 = r2_score(y_test, predicted_rfr_test)
print(f'R-squared (R2) score: {r2}')
```

```
Mean Absolute Error (MAE): 3.7484239585304335
Mean Squared Error (MSE): 28.80571494417949
Root Mean Squared Error (RMSE): 5.367095578073814
R-squared (R2) score: 0.899784286578143
```

In [34]:

```python
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predicted_xgb_test)
print(f'Mean Absolute Error (MAE): {mae}')

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predicted_xgb_test)
print(f'Mean Squared Error (MSE): {mse}')

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

# R-squared (R2) score
r2 = r2_score(y_test, predicted_xgb_test)
print(f'R-squared (R2) score: {r2}')
```

```
Mean Absolute Error (MAE): 3.1672845013725834
Mean Squared Error (MSE): 23.230438446360438
Root Mean Squared Error (RMSE): 4.819796515036754
R-squared (R2) score: 0.9191807956679459
```

Hyperparameter Tuning

In [35]:
```python
from sklearn.model_selection import GridSearchCV
```

In [36]:
```python
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
}
```

In [37]:
```python
grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=5, scoring='r2'
grid_search.fit(X_train, y_train)
```

Out[37]:
```
GridSearchCV
estimator: XGBRegressor
XGBRegressor
```

In [38]:
```python
best_params = grid_search.best_params_
```

In [39]:
```python
best_xgb_regressor = XGBRegressor(**best_params)
```

In [40]:
```python
best_xgb_regressor.fit(X_train,y_train)
```

Out[40]:
```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=N
one,
             enable_categorical=False, eval_metric=None, feature_types=N
one,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.2, max_bin=No
ne,
```

In [41]:
```python
predicted = best_xgb_regressor.predict(X_test)
```

In [42]:
```python
# Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, predicted)
print(f'Mean Absolute Error (MAE): {mae}')

# Mean Squared Error (MSE)
mse = mean_squared_error(y_test, predicted)
print(f'Mean Squared Error (MSE): {mse}')

# Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f'Root Mean Squared Error (RMSE): {rmse}')

# R-squared (R2) score
```

```python
r2 = r2_score(y_test, predicted)
print(f'R-squared (R2) score: {r2}')
```

```
Mean Absolute Error (MAE): 3.0564892466968256
Mean Squared Error (MSE): 22.040202464981448
Root Mean Squared Error (RMSE): 4.694699400918172
R-squared (R2) score: 0.9233216527251445
```

Evaluation

1. **Linear Regression:**
   - Moderate performance with relatively high MAE, MSE, and RMSE.
   - R2 score suggests that the model explains only 56% of the variance in the concrete strength.
2. **Random Forest Regressor:**
   - Improved performance compared to Linear Regression.
   - Lower MAE, MSE, and RMSE, indicating better predictive accuracy.
   - High R2 score (0.90) indicates a good fit to the data.
3. **Gradient Boosting Regressor (XGBoost):**
   - Further improvement over Random Forest with lower MAE, MSE, and RMSE.
   - High R2 score (0.92) suggests excellent explanatory power.
   - A robust model for concrete strength prediction.
4. **XGBoost After Hyperparameter Tuning:**
   - Slight improvement in MAE, MSE, and RMSE.
   - R2 score increased to 0.92, indicating enhanced model performance.
   - Suggests that hyperparameter tuning refined the model.

Conclusion:

- The Gradient Boosting Regressor (XGBoost) demonstrates the best performance for concrete strength prediction, with or without hyperparameter tuning.
- The hyperparameter tuning of XGBoost leads to marginal improvement, indicating that the initial model configuration was already well-tuned.
- Random Forest also performs well but falls slightly behind XGBoost in predictive accuracy.

In [ ]: