Concrete_Compressive_Strength___Prediction

December 9, 2022

1 Problem Statement

- **To build a regression model to predict the concrete compressive strength based on the different features**

1.1 Data Description

- Given is the variable name, variable type, the measurement unit and a brief description
- The concrete compressive strength is the regression problem. The order of this listing corresponds to the order of numerals along the rows of the database.

Name	Data Type	Measurement	Description
Cement (component 1)	quantitative	kg in a m3 mixture	Input Variable
Blast Furnace Slag (component 2)	quantitative	kg in a m3 mixture	Input Variable
Fly Ash (component 3)	quantitative	kg in a m3 mixture	Input Variable
Water (component 4)	quantitative	kg in a m3 mixture	Input Variable

Name	Data Type	Measurement	Description
Superplasticizer (component 5)	quantitative	kg in a m3 mixture	Input Variable
Coarse Aggregate (component 6)	quantitative	kg in a m3 mixture	Input Variable
Fine Aggregate (component 7)	quantitative	kg in a m3 mixture	Input Variable
Age	quantitative	Day (1~365)	Input Variable
Concrete compressive strength	quantitative	Mpa	Output Variable

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import scipy.stats as stats
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

2 Data Ingestion

Dataset Link : * https://archive.ics.uci.edu/ml/machine-learning-databases/concrete/compressive/

```
[3]: df = pd.read_csv('/content/drive/MyDrive/FSDS_Job_Gurantee/Projects/Cement␣
     ↪Strength Prediction/New Project/Concrete_Dataset.csv')
```

```
[4]: df
```

```
[4]:       Cement (component 1)(kg in a m^3 mixture)  \
     0                                         540.0
     1                                         540.0
     2                                         332.5
     3                                         332.5
     4                                         198.6
     …                                           …
     1025                                      276.4
     1026                                      322.2
     1027                                      148.5
     1028                                      159.1
     1029                                      260.9

           Blast Furnace Slag (component 2)(kg in a m^3 mixture)  \
     0                                                       0.0
     1                                                       0.0
     2                                                     142.5
     3                                                     142.5
     4                                                     132.4
     …                                                       …
     1025                                                  116.0
     1026                                                    0.0
     1027                                                  139.4
     1028                                                  186.7
     1029                                                  100.5

           Fly Ash (component 3)(kg in a m^3 mixture)  \
     0                                            0.0
     1                                            0.0
     2                                            0.0
     3                                            0.0
     4                                            0.0
     …                                          …
     1025                                          90.3
     1026                                         115.6
     1027                                         108.6
     1028                                           0.0
```

```
1029                                       78.3

      Water  (component 4)(kg in a m^3 mixture)  \
0                                         162.0
1                                         162.0
2                                         228.0
3                                         228.0
4                                         192.0
…                                           …
1025                                      179.6
1026                                      196.0
1027                                      192.7
1028                                      175.6
1029                                      200.6

      Superplasticizer (component 5)(kg in a m^3 mixture)  \
0                                                   2.5
1                                                   2.5
2                                                   0.0
3                                                   0.0
4                                                   0.0
…                                                   …
1025                                                8.9
1026                                               10.4
1027                                                6.1
1028                                               11.3
1029                                                8.6

      Coarse Aggregate  (component 6)(kg in a m^3 mixture)  \
0                                                1040.0
1                                                1055.0
2                                                 932.0
3                                                 932.0
4                                                 978.4
…                                                   …
1025                                              870.1
1026                                              817.9
1027                                              892.4
1028                                              989.6
1029                                              864.5

      Fine Aggregate (component 7)(kg in a m^3 mixture)  Age (day)  \
0                                                 676.0         28
1                                                 676.0         28
2                                                 594.0        270
3                                                 594.0        365
4                                                 825.5        360
```

```
…                                                  …        …
1025                                               768.3         28
1026                                               813.4         28
1027                                               780.0         28
1028                                               788.9         28
1029                                               761.5         28

      Concrete compressive strength(MPa, megapascals)
0                                                 79.99
1                                                 61.89
2                                                 40.27
3                                                 41.05
4                                                 44.30
…                                                 …
1025                                              44.28
1026                                              31.18
1027                                              23.70
1028                                              32.77
1029                                              32.40

[1030 rows x 9 columns]
```

Rename Columns

[5]:
```
df.rename(columns = {
df.columns[0] : 'Cement',
df.columns[1] : 'Blast_Furnace_Slag',
df.columns[2] : 'Fly_Ash',
df.columns[3] : 'Water',
df.columns[4] : 'Superplasticizer',
df.columns[5] : 'Coarse_Aggregate',
df.columns[6] : 'Fine_Aggregate',
df.columns[7] : 'Age(day)',
df.columns[8] : 'Concrete_Compressive_Strength'
}, inplace = True)
```

[6]:
```
df.info()
```

```
<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: 72.5 KB
```

[7]:
```
df.describe().T
```

[7]:
```
                                count        mean         std     min  \
Cement                         1030.0  281.167864  104.506364  102.00
Blast_Furnace_Slag             1030.0   73.895825   86.279342    0.00
Fly_Ash                        1030.0   54.188350   63.997004    0.00
Water                          1030.0  181.567282   21.354219  121.80
Superplasticizer               1030.0    6.204660    5.973841    0.00
Coarse_Aggregate               1030.0  972.918932   77.753954  801.00
Fine_Aggregate                 1030.0  773.580485   80.175980  594.00
Age(day)                       1030.0   45.662136   63.169912    1.00
Concrete_Compressive_Strength  1030.0   35.817961   16.705742    2.33

                                   25%      50%       75%     max
Cement                         192.375  272.900   350.000   540.0
Blast_Furnace_Slag               0.000   22.000   142.950   359.4
Fly_Ash                          0.000    0.000   118.300   200.1
Water                          164.900  185.000   192.000   247.0
Superplasticizer                 0.000    6.400    10.200    32.2
Coarse_Aggregate               932.000  968.000  1029.400  1145.0
Fine_Aggregate                 730.950  779.500   824.000   992.6
Age(day)                         7.000   28.000    56.000   365.0
Concrete_Compressive_Strength   23.710   34.445    46.135    82.6
```

[8]:
```
df.isnull().sum()
```

[8]:
```
Cement                           0
Blast_Furnace_Slag               0
Fly_Ash                          0
Water                            0
Superplasticizer                 0
Coarse_Aggregate                 0
Fine_Aggregate                   0
Age(day)                         0
Concrete_Compressive_Strength    0
dtype: int64
```

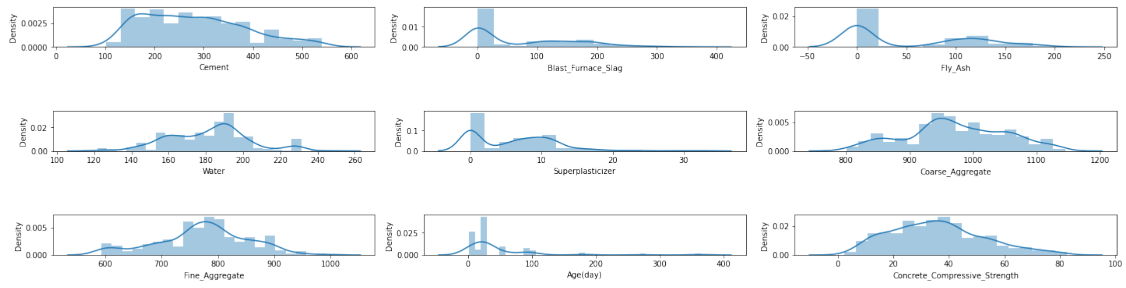
[9]:
```
df.duplicated().sum()
```

[9]:
```
25
```

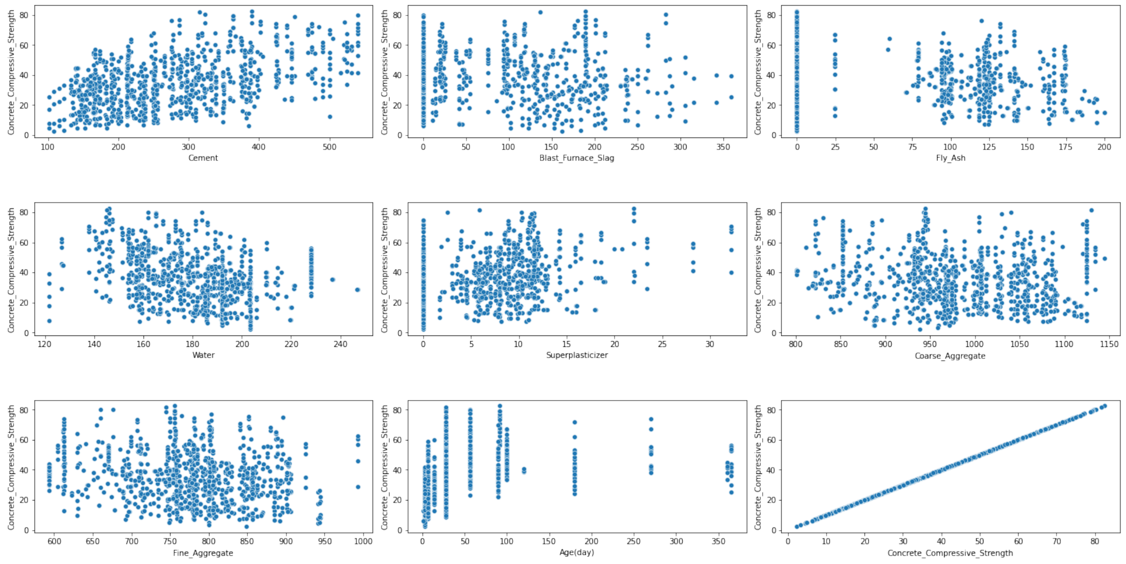
[10]:
```
df.nunique()
```

[10]:
```
Cement                           278
Blast_Furnace_Slag               185
Fly_Ash                          156
Water                            195
Superplasticizer                 111
Coarse_Aggregate                 284
Fine_Aggregate                   302
Age(day)                          14
Concrete_Compressive_Strength    845
dtype: int64
```

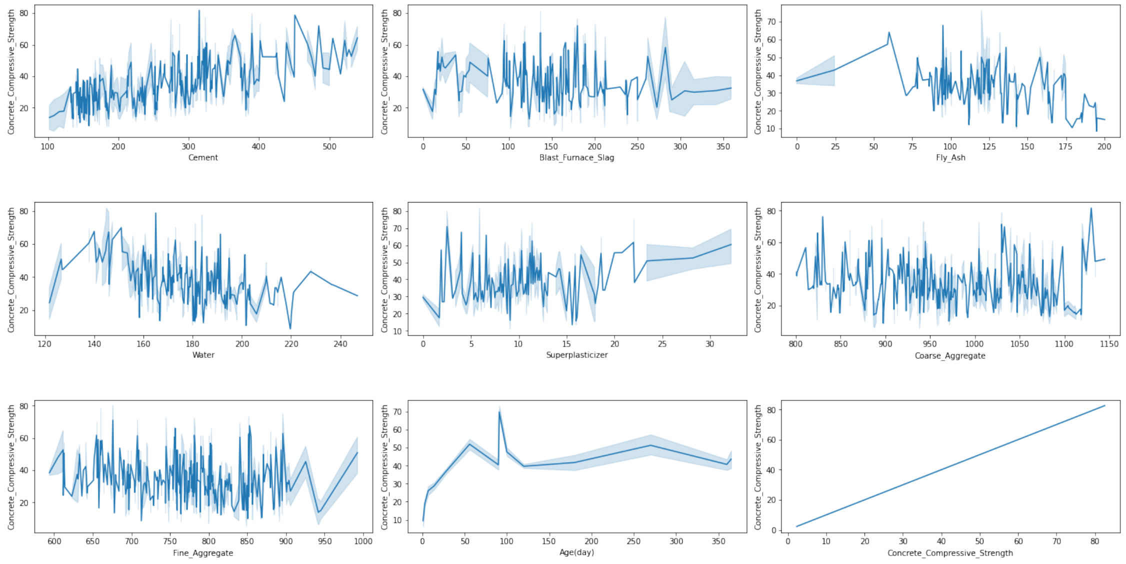
[11]:
```
fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,5))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.distplot(value, ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



[12]:
```
fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.scatterplot(df[col], df['Concrete_Compressive_Strength'], ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

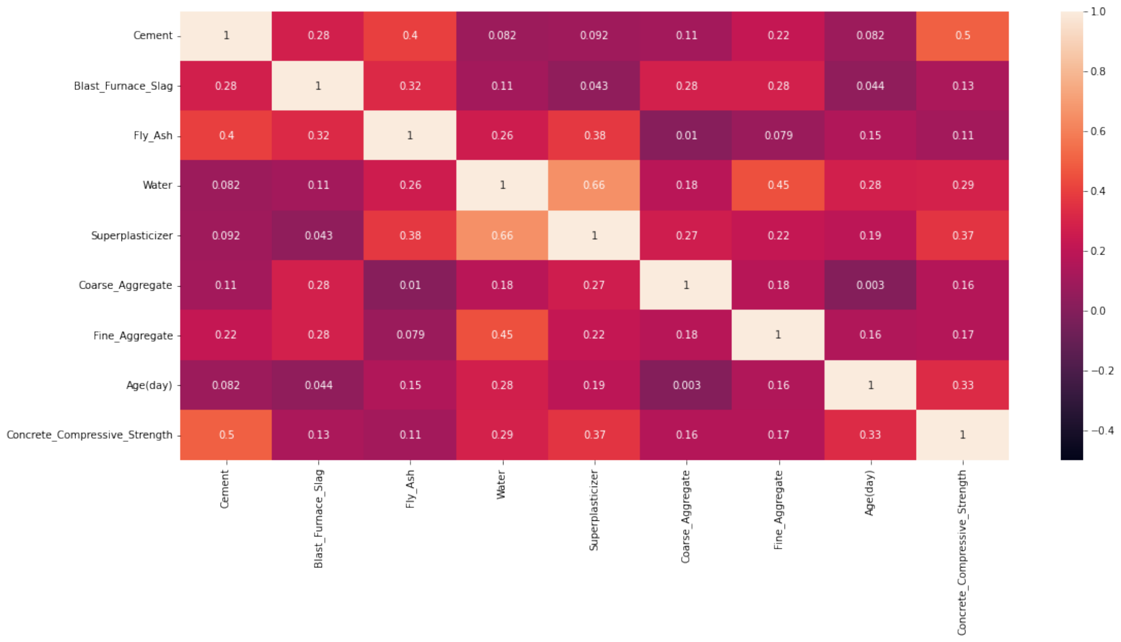


[13]:
```
fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.lineplot(df[col], df['Concrete_Compressive_Strength'], ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



[14]:
```
plt.figure(figsize = (18,8))
sns.heatmap(df.corr().abs(), vmin = -0.5,vmax = 1,annot=True)
```

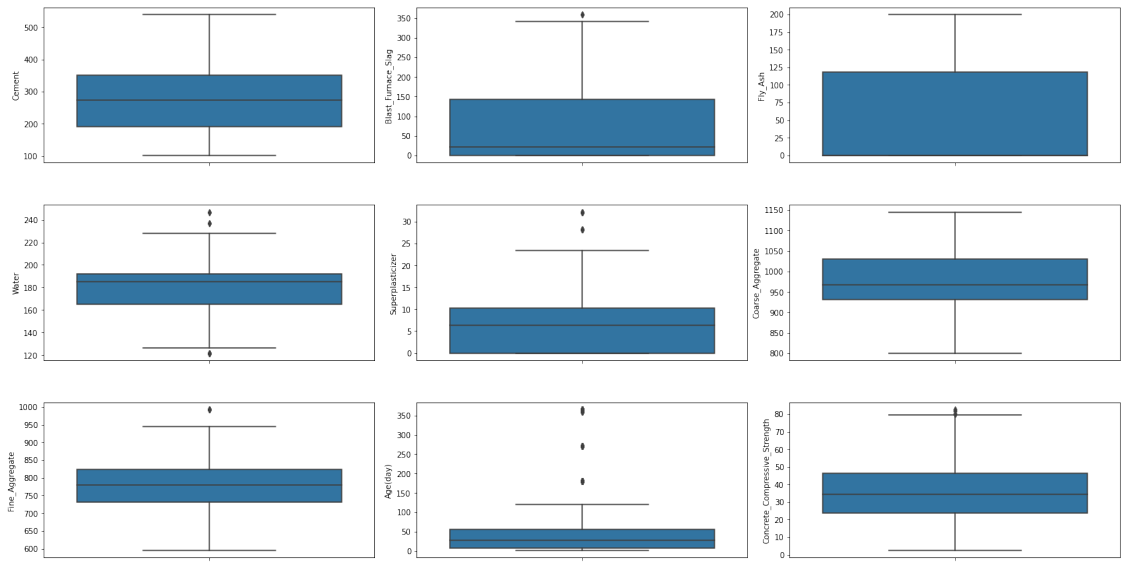
```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7fce924b8bb0>
```



	Cement	Blast_Furnace_Slag	Fly_Ash	Water	Superplasticizer	Coarse_Aggregate	Fine_Aggregate	Age(day)	Concrete_Compressive_Strength
Cement	1	0.28	0.4	0.082	0.092	0.11	0.22	0.082	0.5
Blast_Furnace_Slag	0.28	1	0.32	0.11	0.043	0.28	0.28	0.044	0.13
Fly_Ash	0.4	0.32	1	0.26	0.38	0.01	0.079	0.15	0.11
Water	0.082	0.11	0.26	1	0.66	0.18	0.45	0.28	0.29
Superplasticizer	0.092	0.043	0.38	0.66	1	0.27	0.22	0.19	0.37
Coarse_Aggregate	0.11	0.28	0.01	0.18	0.27	1	0.18	0.003	0.16
Fine_Aggregate	0.22	0.28	0.079	0.45	0.22	0.18	1	0.16	0.17
Age(day)	0.082	0.044	0.15	0.28	0.19	0.003	0.16	1	0.33
Concrete_Compressive_Strength	0.5	0.13	0.11	0.29	0.37	0.16	0.17	0.33	1

3 Outlier Handling

```
[15]: fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.boxplot(y = col, data = df, ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



[16]:
```python
def out1(x):

    Q3 = np.nanpercentile(x , 75)
    Q1 = np.nanpercentile(x , 25)
    IQR =  Q3 - Q1
    UL = (IQR * 1.5 + Q3)
    LL = (Q1 - IQR * 1.5)
    out = [1 if (a > UL) | (a < LL) else 0 for a in x]
    return(out)

# check #outliers in each variable
print(df.apply(out1).apply(sum))

# Function to Replace outlier with LL / UL

def out_impute(x):

    Q3 = np.nanpercentile(x , 75)
    Q1 = np.nanpercentile(x , 25)
    IQR =  Q3 - Q1
    UL = (IQR * 1.5 + Q3)
    LL = (Q1 - IQR * 1.5)
    xnew = [x.mean() if (a<=LL) else x.mean() if (a>=UL) else a for a in x]
    return(xnew)

df = df.apply(out_impute) # Create new data with inputed values
```

```
Cement                           0
Blast_Furnace_Slag               2
```

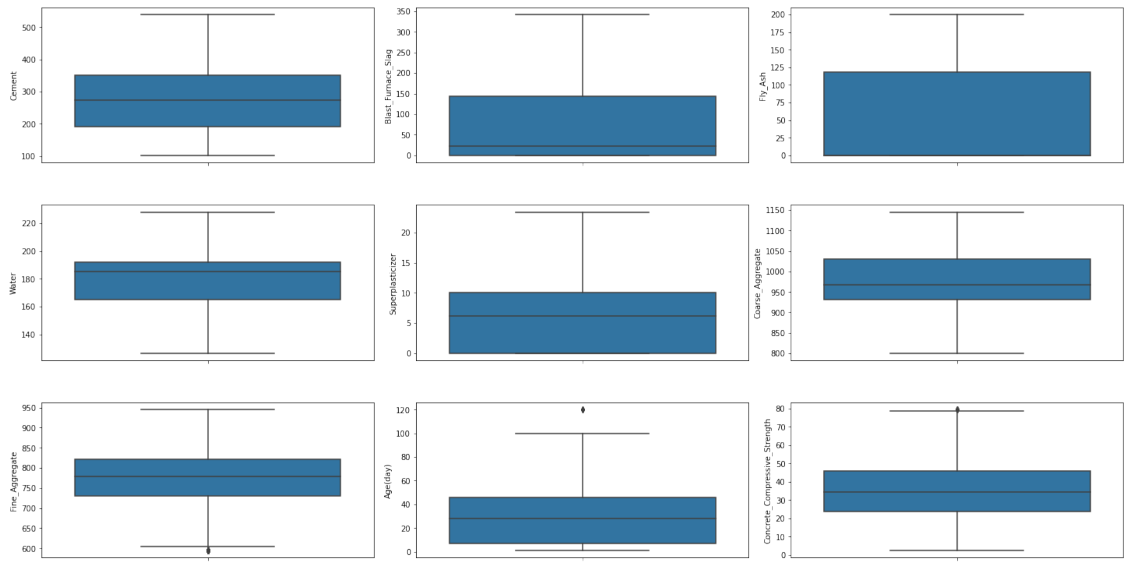
```
Fly_Ash                          0
Water                            9
Superplasticizer                10
Coarse_Aggregate                 0
Fine_Aggregate                   5
Age(day)                        59
Concrete_Compressive_Strength    4
dtype: int64
```

[17]:
```
df
```

[17]:
```
      Cement  Blast_Furnace_Slag  Fly_Ash  Water  Superplasticizer  \
0      540.0                 0.0      0.0  162.0               2.5
1      540.0                 0.0      0.0  162.0               2.5
2      332.5               142.5      0.0  228.0               0.0
3      332.5               142.5      0.0  228.0               0.0
4      198.6               132.4      0.0  192.0               0.0
…        …                   …        …      …                 …
1025   276.4               116.0     90.3  179.6               8.9
1026   322.2                 0.0    115.6  196.0              10.4
1027   148.5               139.4    108.6  192.7               6.1
1028   159.1               186.7      0.0  175.6              11.3
1029   260.9               100.5     78.3  200.6               8.6

      Coarse_Aggregate  Fine_Aggregate   Age(day)  \
0               1040.0           676.0  28.000000
1               1055.0           676.0  28.000000
2                932.0           594.0  45.662136
3                932.0           594.0  45.662136
4                978.4           825.5  45.662136
…                  …              …         …
1025             870.1           768.3  28.000000
1026             817.9           813.4  28.000000
1027             892.4           780.0  28.000000
1028             989.6           788.9  28.000000
1029             864.5           761.5  28.000000

      Concrete_Compressive_Strength
0                         35.817961
1                         61.890000
2                         40.270000
3                         41.050000
4                         44.300000
…                            …
1025                      44.280000
1026                      31.180000
1027                      23.700000
```

```
1028                         32.770000
1029                         32.400000

[1030 rows x 9 columns]
```

[18]:
```
fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.boxplot(y = col, data = df, ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



4 Iteration Checks

[19]:
```
"""Can be Deleted"""
# df.drop(['Coarse_Aggregate','Fine_Aggregate'],axis=1, inplace=True)
```

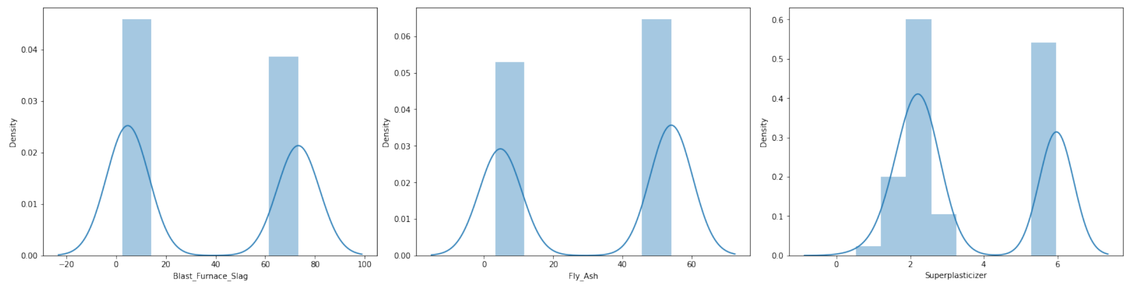
[19]:
```
'Can be Deleted'
```

5 Transformation

Log: Log transformation helps reducing skewness when you have skewed data.

[20]:
```
for col in ['Blast_Furnace_Slag', 'Fly_Ash', 'Superplasticizer']:
  mean_value = df[col].mean()
  df[col] = np.log(df[col])
  df[col] = df[col].mask(np.isinf(df[col])).fillna(mean_value)
# sns.distplot(df['Blast_Furnace_Slag'])
```

[21]:
```
fig, ax = plt.subplots(ncols=3, nrows=1, figsize=(20,5))
index = 0
ax = ax.flatten()
for col in ['Blast_Furnace_Slag', 'Fly_Ash', 'Superplasticizer']:
  sns.distplot(df[col], ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



6 Jarque Bera Test - Normality Check

- The Jarque-Bera Test is a test to determine if a set of data values follows the normal distribution based on the data's skewness and kurtosis
- $JB = n/2[S^2 + 1/4(K - 3)^2]$
- Where
  - n = the number of values for the data
  - S is the sample skewness (how much the data leans away from the mean)
  - K is the sample kurtosis (how thick the tails of the distribution are)
- The test statistic result will always be greater than or equal to zero since:

1. The sample skewness in the test statistic equation is always squared, meaning S^2 is always positive or zero.
2. The sample kurtosis is always positive or zero since the numerator is raised to the 4th power and the denominator is squared.
3. The difference between the sample kurtosis and 3 is squared, meaning this term of the test statistic equation is always positive or zero.
4. The sum of two terms   0 will also be greater than or equal to zero.

- We know if our data follows a normal distribution if the test statistic is close to zero and the p-value is larger than our standard 0.05. The p-value relates to a null hypothesis that the data is following a normal distribution. If the test statistic is large and the p-value is less than 0.05, the data does not follow a normal distribution.

[22]:
```
from scipy.stats import jarque_bera
for col in df.columns:
  print(f"{col} : {jarque_bera(df[col])[1]}")
```

```
Cement : 6.220579606974752e-13
Blast_Furnace_Slag : 0.0
Fly_Ash : 0.0
Water : 0.44805110686953276
Superplasticizer : 0.0
Coarse_Aggregate : 0.00036585771571151504
Fine_Aggregate : 3.5654006097618485e-05
Age(day) : 0.0
Concrete_Compressive_Strength : 1.6120992951673685e-07
```

Observation: * Except water every features has normal distribution beacause p-value is less than 0.05

7 Segregating Independent and Dependent Variable

[23]:
```
X = df.iloc[ : , :-1]
y = df.iloc[ : , -1:]
```

7.0.1 Checking Siginificant features based on P-values

- **H0: Column/Feature does not affect concrete strength**
- **H1: Column/Feature affects concrete strength**
- So, if a column shows p-value <=0.05 then we reject the null hypothesis and say that 'Column/Feature affects medical expenses.'
- We don't have to actually calculate p-values for each and every column. We can simply use OLS from statsmodels.api which basically helps to fit linear regression model and also lets us know what the p-values are.

[25]:
```
import statsmodels.api as sm
X_check = sm.add_constant(X)
model = sm.OLS(y,X_check)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
================================================================================
=========
Dep. Variable:     Concrete_Compressive_Strength   R-squared:
0.727
Model:                                       OLS   Adj. R-squared:
0.725
Method:                            Least Squares   F-statistic:
339.7
Date:                           Fri, 09 Dec 2022   Prob (F-statistic):
1.46e-281
Time:                                   20:31:44   Log-Likelihood:
-3677.8
No. Observations:                           1030   AIC:
7374.
Df Residuals:                               1021   BIC:
7418.
Df Model:                                      8
Covariance Type:                       nonrobust
================================================================================
======
                         coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------
const              117.3157     10.926     10.737      0.000      95.875
138.756
Cement               0.0679      0.004     18.798      0.000       0.061
0.075
Blast_Furnace_Slag  -0.0854      0.011     -7.675      0.000      -0.107
-0.064
Fly_Ash              0.0480      0.017      2.779      0.006       0.014
0.082
Water               -0.2503      0.021    -11.970      0.000      -0.291
-0.209
Superplasticizer    -1.4645      0.237     -6.189      0.000      -1.929
-1.000
Coarse_Aggregate    -0.0258      0.005     -5.185      0.000      -0.036
-0.016
Fine_Aggregate      -0.0440      0.005     -8.363      0.000      -0.054
-0.034
Age(day)             0.3153      0.010     32.663      0.000       0.296
0.334
==============================================================================
Omnibus:                       26.362   Durbin-Watson:                   1.353
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               39.152
Skew:                           0.242   Prob(JB):                     3.15e-09
Kurtosis:                       3.824   Cond. No.                     5.24e+04
```

```
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 5.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

8 Train Test Split

[26]:
```
from sklearn.model_selection import train_test_split
```

[27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

9 Scaling

[28]:
```
from sklearn.preprocessing import RobustScaler # In case of outlier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

[29]:
```
scaler = MinMaxScaler()
```

[30]:
```
X_train_Scaled = scaler.fit_transform(X_train)
```

[31]:
```
X_test_Scaled = scaler.transform(X_test)
```

10 Multicollinearliy Check

[32]:
```
### VIF Check
X_train = pd.DataFrame(X_train)
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
 ↪shape[1])]
for i,j in zip(df.columns,vif):
  print(f"{i} : {j}")
```

```
Cement : 9.236023302973546
Blast_Furnace_Slag : 2.693939762131527
Fly_Ash : 6.657528031015882
Water : 64.46210482974766
Superplasticizer : 11.52022422231611
```

```
Coarse_Aggregate : 87.9532452428649
Fine_Aggregate : 56.65132233337929
Age(day) : 2.468636766841692
```

[33]:
```
"""We are not removing correlated variable because we these are important␣
↪features for our result"""
### Remove var with high VIF one by one in while loop
# X_train = X_train2
# while (max(vif) > 5):
#     indx = vif.index(max(vif)) #Get the index of variable with highest VIF
#     print(X_train.columns)
#     X_train.drop(X_train.columns[indx],axis = 1, inplace = True)
#     vif = [variance_inflation_factor(X_train.values, i) for i in␣
↪range(X_train.shape[1])]
# print(vif)
```

[33]:
```
'We are not removing correlated variable because we these are important features
for our result'
```

11 Model Creation

[34]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xg
from sklearn.ensemble import AdaBoostRegressor
from sklearn.model_selection import GridSearchCV
```

[35]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

[36]:
```
'''Hyperparameters of Ridge Regression'''
Ridge_parameters = {
    'alpha': [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    }

'''Hyperparameters of Lasso Regression'''
Lasso_parameters = {
     'alpha' : [0.001, 0.01, 0.1, 1, 10, 100, 1000]
    }

'''Hyperparameters of Elastic Net Regression'''
Elastic_Net_parameters = {
    'alpha' : [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'l1_ratio' : [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
    }

'''Hyperparameters of Decision Tree Regressor'''
DTC_parameters = {
    'max_depth': np.random.randint(5,10,5),
    'max_features': ['auto', 'sqrt', 'log2'],
    'ccp_alpha': [0.1, .01, .001],
    'criterion' :['squared_error']
    }

'''Hyperparameters of Random Forest Regressor'''
RF_parameters = {
    'n_estimators': [300,500,600,650,700],
    'max_depth': np.random.randint(5,10,5),
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion' :['squared_error']
    }

'''Hyperparameters of Support Vector Regressor'''
SVR_parameters = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'kernel': ['rbf']
    }

'''Hyperparameters of Gradient Boosted Regressor'''
GBR_parameters = {
    'min_samples_split':range(5,20,5),
    'max_depth':range(5,10,2)
  }

'''Hyperparameters of XG Boost Regressor'''
XG_parameters = {
    'min_child_weight': [1, 5, 10],
    'gamma': [0.5, 1, 1.5, 2, 5],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'max_depth': [3, 4, 5]
  }
```

```
# '''Hyperparameters of AdaBoost Classifier'''
AB_parameters = {
    'base_estimator__max_depth':[i for i in range(2,11,2)],
    'base_estimator__min_samples_leaf':[5,10],
    'n_estimators':[10,50,250,1000],
     'learning_rate':[0.01,0.1]
  }

'''All Models'''
models = {
1 : LinearRegression(),
2 : Ridge(alpha=1),
3 : GridSearchCV(estimator = Ridge(alpha=1), param_grid = Ridge_parameters,␣
 ↪scoring='r2', verbose=1, n_jobs=-1),
4 : Lasso(alpha=1),
5 : GridSearchCV(estimator = Lasso(alpha=1), param_grid = Lasso_parameters,␣
 ↪scoring='r2', verbose=1, n_jobs=-1),
6 : ElasticNet(alpha=1, l1_ratio=0.5),
7 : GridSearchCV(estimator = ElasticNet(), param_grid = Elastic_Net_parameters,␣
 ↪scoring='r2', verbose=1, n_jobs=-1),
8 : DecisionTreeRegressor(),
9 : GridSearchCV(estimator = DecisionTreeRegressor(), param_grid =␣
 ↪DTC_parameters, scoring='r2', verbose=1, n_jobs=-1),
10 : RandomForestRegressor(),
11 : GridSearchCV(estimator = RandomForestRegressor(), param_grid =␣
 ↪RF_parameters, scoring='r2', verbose=1, n_jobs=-1),
12 : SVR(),
13 : GridSearchCV(estimator = SVR(), param_grid = SVR_parameters, scoring='r2',␣
 ↪verbose=1, n_jobs=-1),
14 : GradientBoostingRegressor(),
15 : GridSearchCV(estimator = GradientBoostingRegressor(), param_grid =␣
 ↪GBR_parameters, scoring='r2', verbose=1, n_jobs=-1),
16 : xg.XGBRegressor(objective ='reg:linear'),
17 : GridSearchCV(estimator = xg.XGBRegressor(objective ='reg:linear'),␣
 ↪param_grid = XG_parameters, scoring='r2', verbose=1, n_jobs=-1),
18 : AdaBoostRegressor(),
19 : GridSearchCV(estimator = AdaBoostRegressor(), param_grid = AB_parameters,␣
 ↪scoring='r2', verbose=1, n_jobs=-1)
}
```

[37]:
```
map_keys = list(models.keys())
```

[38]:
```
# Get model name using id from linear_model_collection
def get_model_building_technique_name(num):
  if num == 1:
    return 'Linear Regression'
  if num == 2:
    return 'Ridge Regression'
  if num == 3:
    return 'Ridge Regression Tuned'
  if num == 4:
    return 'Lasso Regression'
  if num == 5:
    return 'Lasso Regression Tuned'
  if num == 6:
    return 'Elastic Net Regression'
  if num == 7:
    return 'Elastic Net Regression Tuned'
  if num == 8:
    return 'Decision Trees Regressor'
  if num == 9 :
    return 'Decison Trees Regressor Tuned'
  if num == 10:
    return 'Random Forest Regressor'
  if num == 11:
    return 'Random Forest Regressor Tuned'
  if num == 12:
    return 'Support Vector Regressor'
  if num == 13:
    return 'Support Vector Regressor Tuned'
  if num == 14:
    return 'Gradient Boosting regressor'
  if num == 15:
    return 'Gradient Boosting regressor Tuned'
  if num == 16:
    return 'XG Boost Regressor'
  if num == 17:
    return 'XG Boost Regressor Tuned'
  if num == 18:
    return 'AdaBoost Regressor '
  if num == 19:
    return 'AdaBoost Regressor Tuned '
  return ''
```

[39]:
```
"""Function to calculate all evaluation metrics"""
def evaluation(test,pred, test_set):
    R2_score = r2_score(test, pred)
    adj_R2_score = 1-((1-R2_score)*(len(test)-1)/(len(test)-test_set.
 ↪shape[1]-1))
    MSE = round(mean_squared_error(test, pred),2)
    RMSE = round(np.sqrt(MSE),2)
    MAE = round(mean_absolute_error(test, pred),2)
    return round(R2_score,2),round(adj_R2_score,2),MSE,RMSE,MAE
```

```
[40]: results = [];
for key_index in range(len(map_keys)):
  key = map_keys[key_index]
  try:
    if key in [1,2,3,4,5,6,7,12,13]:
      print(key)
      model = models[key]
      model.fit(X_train, y_train)

      y_pred = model.predict(X_test)
      y_pred_train = model.predict(X_train)

      '''Test Accuracy'''
      r2_score_test, adj_r2_score_test, MSE_test, RMSE_test, MAE_test = ␣
 ↪evaluation(y_test, y_pred, X_test)

      '''Train Accuracy'''
      r2_score_train, adj_r2_score_train, MSE_train, RMSE_train, MAE_train = ␣
 ↪evaluation(y_train, y_pred_train, X_train)

      results.append({
          'Model Name' : get_model_building_technique_name(key),
          'Trained Model' : model,
          'R2_Score_Test' : r2_score_test,
          'Adj_R2_Score_Test' : adj_r2_score_test,
          'MSE_Test' : MSE_test,
          'RMSE_Test' : RMSE_test,
          'MAE_Test' : MAE_test,
          'R2_Score_Train' : r2_score_train,
          'Adj_R2_Score_Train' : adj_r2_score_train,
          'MSE_Train' : MSE_train,
          'RMSE_Train' : RMSE_train,
          'MAE_Train' : MAE_train
          })

    if key in [8,9,10,11,14,15,16,17,18,19]:
      key = map_keys[key_index]
      model = models[key]
      print(key)
      model.fit(X_train_Scaled, y_train)
      y_pred = model.predict(X_test_Scaled)
      y_pred_train = model.predict(X_train_Scaled)

      '''Test Accuracy'''
      r2_Score_test, adj_r2_score_test, MSE_test, RMSE_test, MAE_test = ␣
 ↪evaluation(y_test, y_pred, X_test)
```

```
        '''Train Accuracy'''
        r2_Score_train, adj_r2_score_train, MSE_train, RMSE_train, MAE_train = ␣
↪evaluation(y_train, y_pred_train, X_train)

        results.append({
            'Model Name' : get_model_building_technique_name(key),
            'Trained Model' : model,
            'R2_Score_Test' : r2_score_test,
            'Adj_R2_Score_Test' : adj_r2_score_test,
            'MSE_Test' : MSE_test,
            'RMSE_Test' : RMSE_test,
            'MAE_Test' : MAE_test,
            'R2_Score_Train' : r2_score_train,
            'Adj_R2_Score_Train' : adj_r2_score_train,
            'MSE_Train' : MSE_train,
            'RMSE_Train' : RMSE_train,
            'MAE_Train' : MAE_train
            })
    except Exception as e:
      print(e)

    del model
```

```
1
2
3
Fitting 5 folds for each of 7 candidates, totalling 35 fits
4
5
Fitting 5 folds for each of 7 candidates, totalling 35 fits
6
7
Fitting 5 folds for each of 77 candidates, totalling 385 fits
8
9
Fitting 5 folds for each of 45 candidates, totalling 225 fits
10
11
Fitting 5 folds for each of 30 candidates, totalling 150 fits
12
13
Fitting 5 folds for each of 25 candidates, totalling 125 fits
14
15
Fitting 5 folds for each of 9 candidates, totalling 45 fits
16
[20:33:51] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
```

```
is now deprecated in favor of reg:squarederror.
17
Fitting 5 folds for each of 405 candidates, totalling 2025 fits
[20:35:23] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear
is now deprecated in favor of reg:squarederror.
18
19
Fitting 5 folds for each of 80 candidates, totalling 400 fits
'NoneType' object has no attribute 'set_params'
```

[54]:
```
result_df = pd.DataFrame(results)
```

[42]:
```
result_df_test = result_df.iloc[: , [0,2,3,4,5,6]]
result_df_test
```

[42]:
```
                          Model Name  R2_Score_Test  Adj_R2_Score_Test  \
0                  Linear Regression           0.71               0.70
1                   Ridge Regression           0.71               0.70
2             Ridge Regression Tuned           0.71               0.70
3                   Lasso Regression           0.72               0.71
4             Lasso Regression Tuned           0.71               0.70
5             Elastic Net Regression           0.72               0.71
6       Elastic Net Regression Tuned           0.71               0.71
7           Decision Trees Regressor           0.71               0.82
8      Decison Trees Regressor Tuned           0.71               0.81
9            Random Forest Regressor           0.71               0.87
10     Random Forest Regressor Tuned           0.71               0.84
11          Support Vector Regressor           0.22               0.20
12    Support Vector Regressor Tuned           0.85               0.85
13       Gradient Boosting regressor           0.85               0.87
14 Gradient Boosting regressor Tuned           0.85               0.90
15                XG Boost Regressor           0.85               0.87
16          XG Boost Regressor Tuned           0.85               0.90
17                AdaBoost Regressor           0.85               0.74

    MSE_Test  RMSE_Test  MAE_Test
0      76.11       8.72      6.74
1      76.11       8.72      6.74
2      76.04       8.72      6.74
3      73.88       8.60      6.71
4      76.08       8.72      6.74
5      73.86       8.59      6.71
6      75.64       8.70      6.73
7      47.05       6.86      4.39
8      47.54       6.89      5.06
9      32.12       5.67      3.99
10     39.94       6.32      4.74
```

```
11    206.55       14.37      11.66
12     39.27        6.27       4.29
13     32.68        5.72       4.11
14     25.35        5.03       3.45
15     33.50        5.79       4.17
16     25.15        5.01       3.44
17     66.82        8.17       6.69
```

[43]:
```
result_df_train = result_df.iloc[: , [0,7,8,9,10,11]]
result_df_train
```

[43]:
```
                          Model Name  R2_Score_Train  Adj_R2_Score_Train  \
0                  Linear Regression            0.73                0.73
1                   Ridge Regression            0.73                0.73
2             Ridge Regression Tuned            0.73                0.73
3                   Lasso Regression            0.73                0.72
4             Lasso Regression Tuned            0.73                0.73
5             Elastic Net Regression            0.73                0.72
6       Elastic Net Regression Tuned            0.73                0.73
7            Decision Trees Regressor            0.73                1.00
8       Decison Trees Regressor Tuned            0.73                0.93
9            Random Forest Regressor            0.73                0.98
10      Random Forest Regressor Tuned            0.73                0.94
11           Support Vector Regressor            0.22                0.21
12     Support Vector Regressor Tuned            0.94                0.94
13         Gradient Boosting regressor            0.94                0.93
14   Gradient Boosting regressor Tuned            0.94                0.97
15                 XG Boost Regressor            0.94                0.93
16           XG Boost Regressor Tuned            0.94                0.97
17                 AdaBoost Regressor            0.94                0.78

    MSE_Train  RMSE_Train  MAE_Train
0       74.12        8.61       6.55
1       74.12        8.61       6.55
2       74.12        8.61       6.55
3       74.87        8.65       6.61
4       74.12        8.61       6.55
5       74.95        8.66       6.61
6       74.15        8.61       6.55
7        1.07        1.03       0.14
8       19.78        4.45       3.31
9        5.67        2.38       1.52
10      16.84        4.10       3.06
11     212.70       14.58      11.71
12      17.24        4.15       1.99
13      17.95        4.24       3.07
14       6.90        2.63       1.71
```

```
15       19.37         4.40        3.15
16        9.08         3.01        1.99
17       60.27         7.76        6.43
```

12 Save and Load Best Model

[44]:
```
Best_Model_Name = result_df['Trained␣
↪Model'][result_df[result_df['Adj_R2_Score_Test'] ==␣
↪max(result_df['Adj_R2_Score_Test'])]['Trained Model'].index[0]]
Best_Model_Index = result_df['Trained␣
↪Model'][result_df[result_df['Adj_R2_Score_Test'] ==␣
↪max(result_df['Adj_R2_Score_Test'])]['Trained Model'].index].index[0]
import pickle
""" Save the model to disk """
filename = 'Concrete_Compressive_Strength_Prediction.sav'
pickle.dump(Best_Model_Name, open(filename, 'wb'))
"""Load the model from disk"""
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model.fit(X_train, y_train)
y_pred = loaded_model.predict(X_test)
```

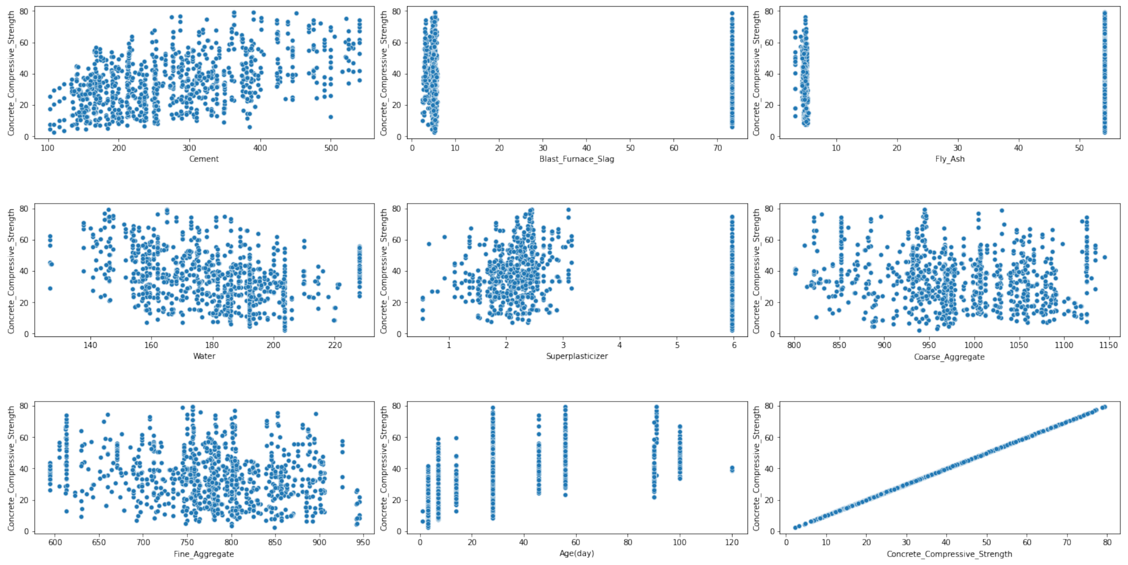
```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
```

13 Linear Regression Assumptions

[45]:
```
from sklearn.linear_model import LinearRegression
LR_Model = LinearRegression()
LR_Model.fit(X_train, y_train)
y_pred = LR_Model.predict(X_test)
```

13.1 1. Linear Relationship

[46]:
```
fig, ax = plt.subplots(ncols=3, nrows=3, figsize=(20,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
  sns.scatterplot(df[col], df['Concrete_Compressive_Strength'], ax=ax[index])
  index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```

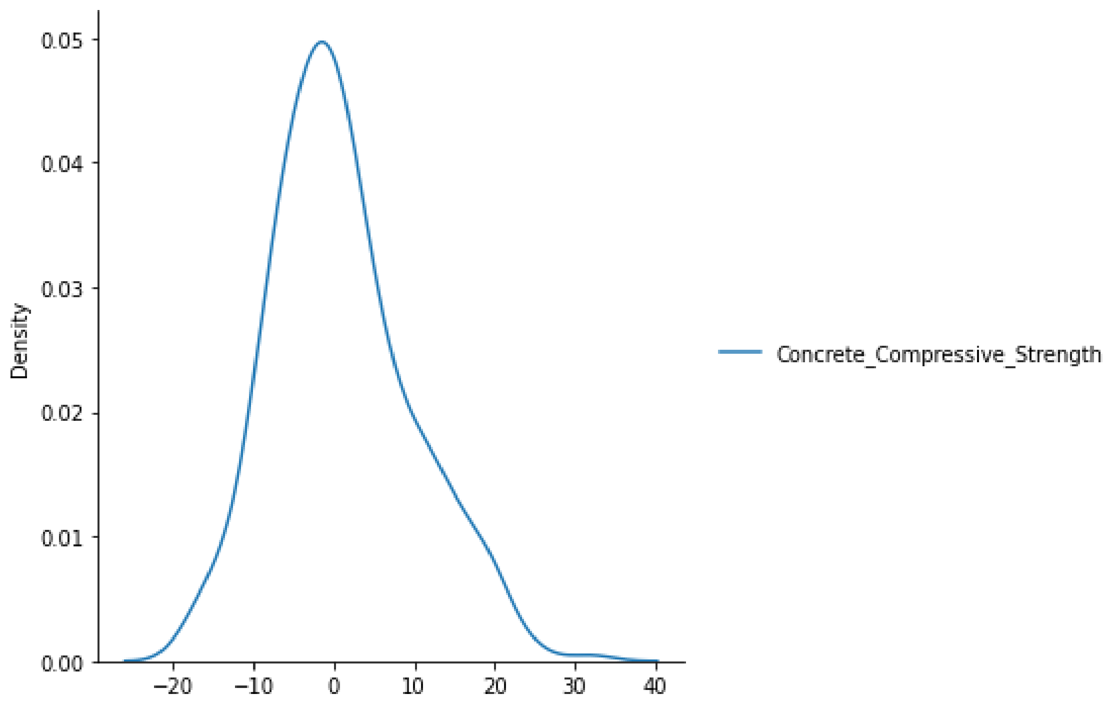


13.2 2. Normality

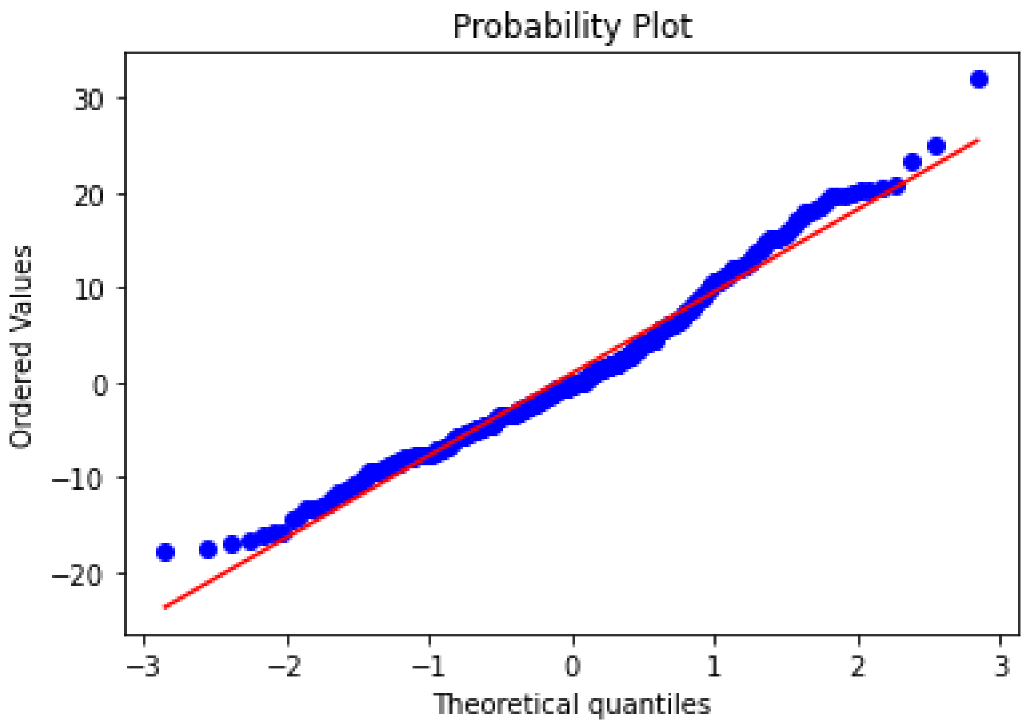
- Residuals should be normally distributed

[47]:
```
residuals = y_test - y_pred
sns.displot(residuals,kind='kde')
```

[47]:
```
<seaborn.axisgrid.FacetGrid at 0x7fce96d1eaf0>
```



[48]:
```
# QQ Plot
import scipy as sp
fig, ax = plt.subplots(figsize=(6,4))
sp.stats.probplot(residuals['Concrete_Compressive_Strength'], plot=ax, fit=True)
plt.show()
```



13.3 3. Multicollinearity

[49]:
```
### VIF Check
X_train = pd.DataFrame(X_train)
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = [variance_inflation_factor(X_train.values, i) for i in range(X_train.
 ↪shape[1])]
for i,j in zip(df.columns,vif):
  print(f"{i} : {j}")
```

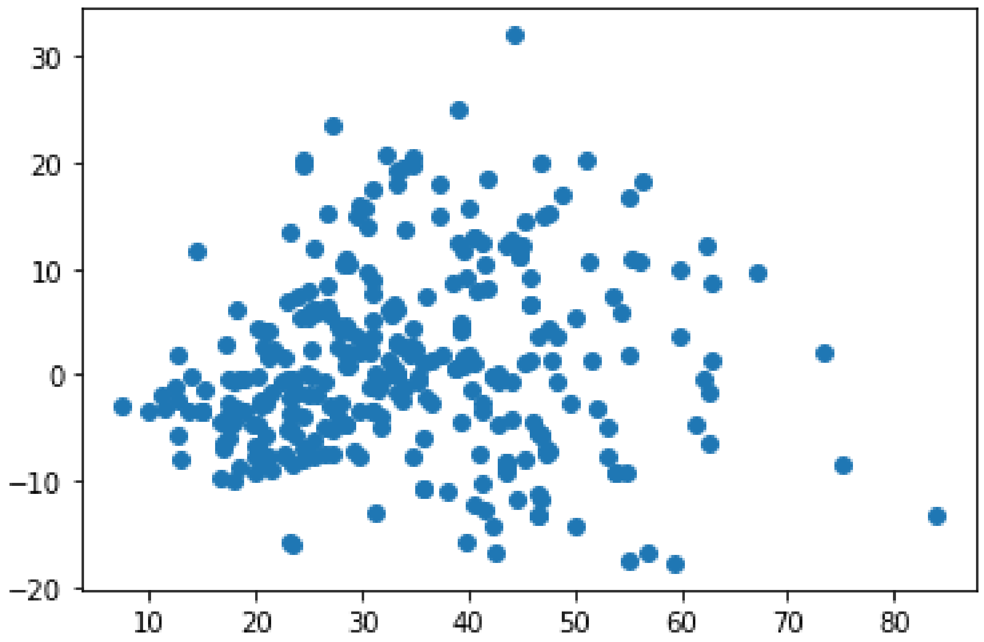
```
Cement : 9.236023302973546
Blast_Furnace_Slag : 2.693939762131527
Fly_Ash : 6.657528031015882
Water : 64.46210482974766
Superplasticizer : 11.52022422231611
Coarse_Aggregate : 87.9532452428649
Fine_Aggregate : 56.65132233337929
Age(day) : 2.468636766841692
```

Observation: * There is multicollineairty in few variables but we decided not to remove because these are important for the concrete compressive strength prediction

13.4 4. Homoscedasticity

[50]:
```
plt.scatter(y_pred,residuals)
```

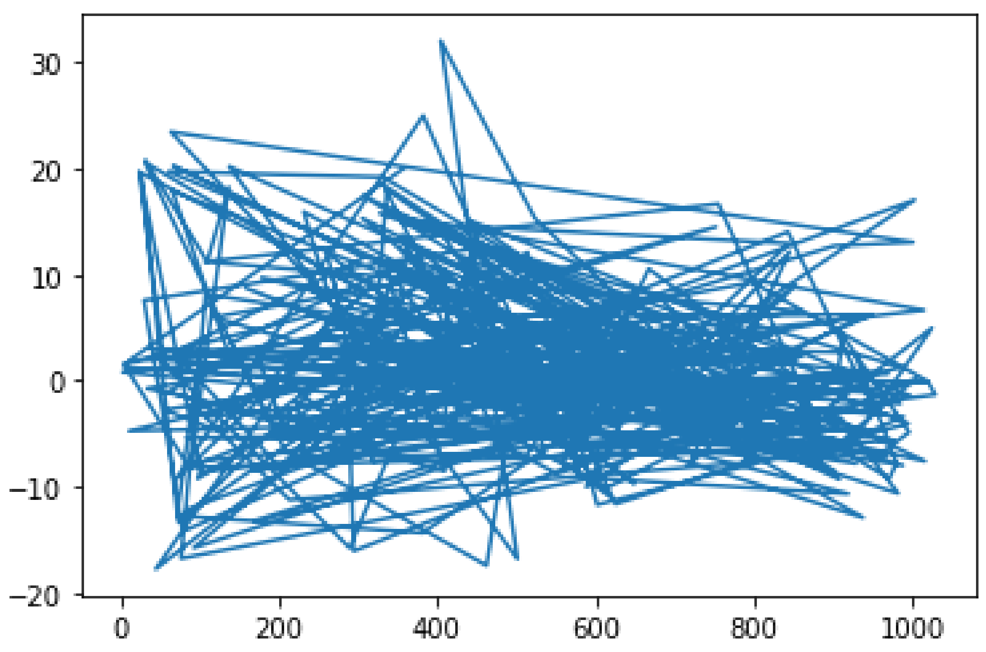
[50]:
```
<matplotlib.collections.PathCollection at 0x7fce96dd58b0>
```



13.5 5. Autocorrelation

[51]:
```
plt.plot(residuals)
```

[51]:
```
[<matplotlib.lines.Line2D at 0x7fce96d99bb0>]
```



13.5.1 Durbin Watson Test

- A test developed by statisticians professor James Durbin and Geoffrey Stuart Watson is used to detect autocorrelation in residuals from the Regression analysis
- **Autocorrelation:**
- Autocorrelation represents the degree of similarity between a given time series and a lagged version of itself over successive time intervals. Autocorrelation measures the relationship between a variable's current value and its past values
- **Assumptions of Durbin-Watson d Test**
- The errors are normally distributed with a mean value of 0
- The errors are stationary
- Null and Alternate Hypothesis of Durbin-Watson d Test
- Null Hypothesis: First order autocorrelation does not exist
- Alternate Hypothesis: First order autocorrelation exists
- The test statistic is approximately equal to 2*(1-r) where r is the sample autocorrelation of the residuals. Thus, the test statistic will always be between 0 and 4 with the following interpretation:
- A test statistic of 2 indicates no serial correlation.
- The closer the test statistics is to 0, the more evidence of positive serial correlation.

- The closer the test statistics is to 4, the more evidence of negative serial correlation.

```
[52]: from statsmodels.stats.stattools import durbin_watson
      gfg = durbin_watson(residuals)
      print(gfg)
```

```
[2.04853471]
```

Observation: * There is very liitle or no auto correlation present between the residuals

14 THE END