Project Feature Engineering && Hyperparameter Tuning

March 16, 2021

1. Cement (cement) quantitative kg in a m3 mixture Input Variable
2. Blast Furnace Slag (slag) quantitative kg in a m3 mixture Input Variable
3. Fly Ash (ash) quantitative kg in a m3 mixture Input Variable
4. Water(water) quantitative kg in a m3 mixture Input Variable
5. Superplasticizer (superplastic) quantitative kg in a m3 mixture Input Variable
6. Coarse Aggregate (coarseagg) quantitative kg in a m3 mixture Input Variable
7. Fine Aggregate (fineagg) quantitative kg in a m3 mixture Input Variable
8. Age(age) quantitative Day (1~365) Input Variable
9. Concrete compressive strength(strength) quantitative MPa Output Variable

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt

from sklearn import linear_model
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV, RandomizedSearchCV
from sklearn.metrics import  r2_score
from sklearn.ensemble import RandomForestRegressor,BaggingRegressor,␣
↪GradientBoostingRegressor, AdaBoostRegressor, StackingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler,␣
↪StandardScaler, RobustScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.svm import SVR
from sklearn.gaussian_process import GaussianProcessRegressor

import warnings
warnings.filterwarnings('ignore')

sns.set()
```

[2]:
```
df = pd.read_csv("concrete.csv")
df.head()
```

[2]:

	cement	slag	ash	water	superplastic	coarseagg	fineagg	age	\
0	141.3	212.0	0.0	203.5	0.0	971.8	748.5	28	

```
1   168.9   42.2  124.3  158.3          10.8     1080.8    796.2   14
2   250.0    0.0   95.7  187.4           5.5      956.9    861.2   28
3   266.0  114.0    0.0  228.0           0.0      932.0    670.0   28
4   154.8  183.4    0.0  193.3           9.1     1047.4    696.7   28

   strength
0     29.89
1     23.51
2     29.22
3     45.85
4     18.29
```

1 Exploratory Data Quality Report Reflecting the Following:

1.0.1 Data types and description of the independent attributes which should include (name, range of values observed, central values (mean and median), standard deviation and quartiles, analysis of the body of distributions/tails. (2 Marks)

[3]:
```
df.describe().T
```

[3]:
```
                count        mean         std     min      25%      50%  \
cement         1030.0  281.167864  104.506364  102.00  192.375  272.900
slag           1030.0   73.895825   86.279342    0.00    0.000   22.000
ash            1030.0   54.188350   63.997004    0.00    0.000    0.000
water          1030.0  181.567282   21.354219  121.80  164.900  185.000
superplastic   1030.0    6.204660    5.973841    0.00    0.000    6.400
coarseagg      1030.0  972.918932   77.753954  801.00  932.000  968.000
fineagg        1030.0  773.580485   80.175980  594.00  730.950  779.500
age            1030.0   45.662136   63.169912    1.00    7.000   28.000
strength       1030.0   35.817961   16.705742    2.33   23.710   34.445

                   75%     max
cement         350.000   540.0
slag           142.950   359.4
ash            118.300   200.1
water          192.000   247.0
superplastic    10.200    32.2
coarseagg     1029.400  1145.0
fineagg        824.000   992.6
age             56.000   365.0
strength        46.135    82.6
```

[4]:
```
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   slag          1030 non-null   float64
 2   ash           1030 non-null   float64
 3   water         1030 non-null   float64
 4   superplastic  1030 non-null   float64
 5   coarseagg     1030 non-null   float64
 6   fineagg       1030 non-null   float64
 7   age           1030 non-null   int64
 8   strength      1030 non-null   float64
dtypes: float64(8), int64(1)
memory usage: 72.5 KB
```

1.0.2 Insights:

All variables are numerical in nature

The values being numerical in nature allows the computer to process the data using quantative techniques. This is important since the computer struggles at dealing with non-numerical data, especially noticable when trying to plot the data.

1.0.3 Missing values analysis, Outlier detection, Duplicates check, Comment about if the zero values mean the null value here. (4 Marks)

[5]:
```
for col in df.columns:
    print(col + ':')
    print(df[col].unique())
    print(df[col].nunique())
```

```
cement:
[141.3 168.9 250.  266.  154.8 255.  166.8 251.4 296.  155.  151.8 173.
 385.  237.5 167.  213.8 336.  190.7 312.7 229.7 228.  236.  132.  331.
 310.  304.  425.  166.1 339.  475.  145.7 313.  178.  165.  277.2 325.
 194.7 246.8 382.  149.  531.3 387.  193.5 326.  337.9 200.  218.9 234.
 309.9 350.  182.  480.  295.7 233.8 379.5 332.5 237.  238.1 323.7 342.
 388.6 147.8 290.4 500.  284.  218.2 190.3 116.  277.  376.  273.  212.5
 362.6 275.1 139.6 427.5 183.9 318.8 252.  149.5 540.  380.  436.  281.
 151.6 326.5 397.  238.  158.6 302.  192.  155.6 160.  222.4 251.8 213.5
 446.  133.  122.6 290.2 375.  181.4 298.2 162.  262.  213.7 313.3 322.
 173.5 299.8 198.6 286.3 349.  520.  252.1 255.5 172.4 212.1 276.  393.
 230.  389.9 157.  359.  374.  102.  202.  252.3 336.5 315.  159.  231.8
 159.8 164.6 136.4 190.  184.  424.  212.  156.  136.  203.5 254.  220.8
 167.4 144.  108.3 214.9 469.  522.  250.2 439.  322.5 153.  525.  259.9
 236.9 366.  333.  145.9 277.1 166.  143.  181.9 450.1 528.  238.2 186.2
 212.6 491.  152.6 252.5 295.  150.7 249.1 505.  148.1 143.7 289.  298.
 173.8 135.7 225.  305.3 170.3 330.5 304.8 150.  148.  297.8 321.3 134.7
 168.  300.  382.5 321.  339.2 288.  400.  155.2 334.  261.9 485.  356.
 264.5 317.9 288.4 275.  145.4 297.2 280.  451.  313.8 164.  298.1 381.4
```

```
 261.  145.  272.8 260.  153.1 355.  272.6 133.1 143.6 210.7 260.9 295.8
 322.2 405.  401.8 255.3 266.2 307.  152.  160.2 143.8 151.  158.8 139.9
 374.3 287.3 303.6 140.  150.9 135.  146.5 146.  314.  516.  152.7 305.
 148.5 154.  164.2 139.7 325.6 279.8 158.4 265.  500.1 141.9 355.9 318.
 159.1 285.  239.6 297.  321.4 316.1 312.9 153.6 142.  287.  158.  147.
 144.8 276.4]
278
slag:
[212.   42.2   0.  114.  183.4 250.2 184.  178.1 116.  237.5 187.   98.1
 342.1 157.  207.  143.   76.  106.3  99.  172.6 145.  129.8  97.8  75.4
 118.   20.  290.2 189.  200.  156.  142.8  45.2 151.2 142.5  92.  282.8
 133.   38.   97.1 175.1  15.  128.5  54.6 173.  209.4  47.5 122.6 212.5
  97.  236.   17.2 148.9 288.  243.5 128.  183.   24.  210.  183.9 193.5
  93.8 214.  111.  262.2  95.  188.   50.1 132.4 200.9 170.3  13.6 149.
  19.  189.2 153.   11.  137.  209.  250.  161.6 190.   86.   22.  243.
 196.  135.7 147.2 129.9 170.  162.4  53.8 117.2 166.8 177.  148.6 239.
 118.8 100.6  91.7 166.  230.5 190.1 260.  116.8 169.  305.3 272.8  50.
 158.8 124.1  26.  238.7 140.  170.2 134.   93.4 120.  203.5 162.  139.
 133.7 102.  155.5 169.6 112.  237.  137.2 164.2  42.1 117.  192.   17.6
 110.5 175.  121.  180.  129.  141.3 100.  236.8 181.9 101.  136.  210.2
 316.1 100.5 164.  117.6  94.7  98.8 112.3 178.  136.3 259.7 238.2 132.6
 120.5 139.9  17.5 105.  114.6 230.  145.3 144.7 139.4 105.1 174.  161.
 163.9 166.4 206.5 128.9 166.6 186.7 359.4 210.7 160.5 144.2 169.4 167.
 119.  119.7 144.  163.  115. ]
185
ash:
[  0.  124.3  95.7 118.3 143.  138.7 195.   24.5 125.4 118.2 161.  163.3
  77.  118.6 143.6 167.  100.5 125.1  92.   94.  138.  124.1 111.2 122.
  95.6  94.6  71.   94.1  96.2 141.  132.1 123.8 125.2  90.  100.4 121.4
  59.   76.  113.  111.9 137.9 158.  116.   96.7  99.9 174.2  79.  107.
 164.   86.  146.  173.5 119.8  75.6 172.4 121.6  98.8 150.4 125.8 190.
 132.  124.8  98.  128.6 133.  121.9 194.  174.7  78.4  71.5 142.8 148.1
  97.4  91.  123.  175.  185.3  60.  136.6 132.6 159.9 126.  109.   99.6
  87.  182.  106.9 165.7 163.8 120.  128.  143.2  86.1 142.   86.5 126.5
 172.  178.9 117.5 100.  112.6 200.  111.  107.5  78.  119.  150.  106.
  97.  152.   89.6 134.  174.9  78.3 115.6  91.7  87.5 136.  179.  139.
 146.4 187.  106.2 184.  103.3  93.9 183.9 163.  193.  185.   89.3 106.7
 113.2 112.  108.6  81.8 200.1  95.  127.7 160.9 148.   82.  194.9 129.7
 182.1 137.  141.6 127.9 112.3 142.7 130.  103.   89.  133.6 166.   90.3]
156
water:
[203.5 158.3 187.4 228.  193.3 192.  188.5 194.  167.5 186.  185.  181.7
 182.  162.1 178.1 195.2 185.7 179.  168.  153.5 176.5 189.  197.  181.9
 127.  179.9 163.8 160.7 184.  164.  170.2 143.3 183.  141.8 157.  199.
 174.9 190.  158.5 167.8 171.5 197.9 165.6 153.9 203.  247.  186.7 183.8
 157.9 171.2 168.1 200.  175.1 140.8 161.9 191.  214.6 140.  159.3 164.9
 159.5 142.  155.7 175.8 162.  178.  192.9 218.  193.  184.4 167.  180.3
 189.3 146.1 191.8 154.6 196.  126.6 169.6 209.7 202.  195.  175.5 159.2
```

```
 164.8 211.5 144.7 170.  193.8 156.8 180.  195.5 145.9 154.  170.1 206.
 146.3 145.  201.  174.  168.4 181.6 171.6 213.  159.  198.  178.5 151.4
 155.6 137.8 146.  185.8 154.8 181.1 170.6 246.9 191.3 202.5 178.8 160.6
 190.6 173.  159.4 210.  158.  166.7 158.1 214.  191.6 172.3 172.  181.
 194.9 166.6 176.  167.9 221.  201.3 190.5 180.2 121.8 212.  187.  195.4
 171.  177.4 216.  201.7 174.8 165.  169.9 163.6 173.8 198.7 195.7 158.4
 200.6 151.  177.  175.  182.9 147.4 188.6 177.9 203.2 183.2 200.3 190.2
 187.6 186.4 213.5 237.  201.9 221.4 178.9 192.7 181.2 236.7 127.3 172.4
 219.7 173.5 181.4 175.6 163.  182.5 177.6 220.1 190.7 160.  188.  168.3
 220.  180.8 179.6]
195
superplastic:
[ 0.  10.8  5.5  9.1  6.4  9.  18.3  7.   6.7  3.   7.8  8.   6.1  5.
 10.  16.5  4.5  6.   3.4  3.6 11.2  7.9  7.5 12.  28.2 11.6 11.   9.5
 11.3  5.9 22.1  8.2  8.9  4.6 15.9 10.3 12.1  2.2  9.4  8.1 11.9  9.9
  4.   8.7  1.9 14.3 12.6  2.5  5.8 20.8 15.  10.7 12.4  5.3 11.7 23.4
  7.6 11.1 13.9  6.9  8.6  6.5  5.2  8.3  4.1  5.7 22.  10.9 10.1  1.7
 14.2 12.2 10.4  8.5 18.6  9.6 32.2 10.2  6.6 18.8 11.8 13.   3.9  6.3
 18.  15.6 12.8 16.1  9.7  9.8  9.2 15.3  2.  13.1 10.6  3.1 17.9  7.1
 11.4 20.  12.7  7.4  6.2 16.   8.8 10.5 19.  11.5 12.3  8.4  7.2]
111
coarseagg:
[ 971.8 1080.8  956.9  932.  1047.4  889.8  975.6 1028.4 1085.   880.
  944.   946.8  966.   898.  1066.   986.  1090.   999.7 1028.1  955.8
  972.6  867.  1025.   914.   852.1 1058.6  919.   968.   985.8 1000.
 1007.3 1005.6 1061.7 1063.   998.  1086.8 1047.   953.   938.   998.2
  801.   944.7 1145.   978.  1078.7  981.   913.9 1050.  1059.4  936.2
  955.1  947.  1006.4 1134.3  974.   853.   949.9  942.7  965.4  961.2
 1125.   842.  1005.8 1075.7 1088.1  909.8 1003.5  931.  1007.8 1053.6
  959.2 1098.   835.   846.8 1055.  1002.  1043.6  838.4  877.  1104.
  992.   801.1  967.  1119.   953.3  929.8 1022.   824.   967.1 1006.
  948.9 1052.3  949.   958.2 1055.6  879.6  927.4  820.   895.  1065.8
 1046.9  829.  1006.2 1111.   878.2  978.4 1004.6  855.   835.5 1026.6
 1006.3  951.  1057.6  870.   940.6 1118.8 1029.4  935.4  942.   926.1
  887.   987.8 1130.   848.   877.2 1056.4 1049.3 1040.6 1023.3  922.6
  923.   822.  1085.4  847.  1076.2 1001.9  814.  1012.   936.   938.2
 1014.3  896.   977.6  935.   884.9  940.  1053.5  935.7  852.9  824.3
  931.2  882.   827.  1056.   838.1  973.9  859.   946.5  963.4 1012.4
 1124.4  920.  1083.4 1003.8 1001.8  943.  1111.6 1069.  1074.5 1030.
  814.1  924.   878.  1007.2  970.   892.  1113.   924.1  888.  1079.
  913.2  990.   811.   959.4  910.   914.3  839.   819.   878.4  961.
 1058.7 1047.8  946.  1069.2  895.2 1120.   832.6  860.5  907.9  830.
 1022.8  825.  1060.   973.4  925.3  849.   953.2 1104.6 1044.   864.
  833.  1023.  1069.3  941.   931.3  949.4  979.   977.   864.5 1033.
 1091.4  908.   817.9 1088.   953.4  818.   909.7  903.  1100.   828.7
  854.   941.5  991.   971.   858.8  916.  1013.2  904.4  879.   895.5
 1040.   869.   991.2  996.   965.  1074.   860.  1013.   925.   819.2
  869.1  959.  1038.   892.4  904.   845.   849.3  917.   868.6  881.6
```

```
 1075.   866.9  838.   825.1  897.7  882.6  838.9  801.4  861.   989.6
 1031.   941.6 1021.   870.1  867.2  916.6  923.2  967.4  883.  1061.
  970.4 1118.  1049.   979.5]
284
fineagg:
[748.5 796.2 861.2 670.  696.7 945.  692.6 757.7 765.  699.  694.6 856.8
 763.  594.  636.  785.5 817.  804.  822.2 757.6 674.3 749.1 736.  821.
 887.1 780.1 749.  781.  816.8 822.  746.8 900.9 782.5 783.  770.1 901.8
 800.9 739.  780.  893.7 845.  704.3 792.  755.8 660.  825.  794.9 760.
 651.2 770.  780.7 712.2 859.2 852.2 905.9 605.  775.  695.  847.  659.9
 806.2 925.7 828.5 865.  613.  801.  746.6 792.7 802.6 891.9 856.  762.4
 885.  762.  853.  903.6 777.5 806.9 800.  641.  880.4 892.7 676.  689.
 754.3 719.7 868.  774.  815.9 792.5 633.  789.  716.1 697.7 879.  870.3
 899.8 857.2 775.5 712.  795.  800.1 992.6 777.8 744.2 839.2 680.  733.
 785.4 611.8 771.9 710.  793.5 784.  727.6 825.5 803.7 806.  855.  821.4
 724.3 856.4 709.  779.3 768.  785.6 789.3 758.6 781.2 756.7 942.  889.
 745.  669.  867.7 778.5 688.2 734.3 728.9 764.4 623.  750.  799.5 698.
 759.3 863.  744.3 688.7 805.  830.  849.  780.6 840.5 896.  694.1 707.9
 709.5 758.  776.4 684.  781.5 762.9 695.4 756.9 842.6 790.  871.8 809.
 741.4 875.6 827.  665.6 643.  630.  714.3 824.  613.2 720.  764.3 903.8
 683.9 844.  784.3 769.  678.  754.  830.1 722.  805.3 655.  794.  764.
 833.  760.1 943.1 798.9 895.3 721.  757.  631.  802.3 705.2 675.  884.
 778.  655.3 804.9 728.  739.3 666.  717.8 698.5 732.6 828.  790.4 736.6
 829.5 638.  868.7 753.5 782.9 846.  651.  662.  761.  880.  729.  674.8
 697.  871.  893.  704.  762.2 795.3 812.  844.5 689.3 761.5 769.3 829.
 813.4 808.  780.3 852.1 749.3 813.  744.5 820.  850.  869.  709.7 614.
 774.3 772.  850.6 826.8 753.4 730.4 695.9 815.2 722.5 656.  772.2 652.
 730.  872.  778.4 690.2 802.  816.  705.  679.7 612.  759.  734.  655.6
 735.6 741.  712.9 785.3 884.3 737.  788.9 685.  690.  664.3 779.7 759.5
 657.9 643.5 785.  657.  744.  696.  794.2 658.  688.  713.  815.  753.
 811.5 768.3]
302
age:
[ 28  14  90   7  56   3 100  91 180 365 270 360 120   1]
14
strength:
[29.89 23.51 29.22 45.85 18.29 21.86 15.75 36.64 21.65 28.99 36.35  6.94
 27.92 26.26 23.89 49.97 30.08 44.86 15.04 25.1  13.36 21.92 20.42 33.3
 31.74 45.3  49.19 65.2  21.54 33.8  32.04 39.29 23.74 44.52 37.91 39.16
 16.88 47.71 17.54 41.41 24.28 60.32 37.42 23.52 59.2  61.46 17.2  40.68
 49.9  33.4  49.25 39.   37.27 39.3  38.22 20.28 48.67 34.57 35.23 42.35
 20.73 10.38 37.34 54.9  41.05 29.98 28.63 44.3  52.04 49.8  11.41 50.46
 28.1  26.92 45.08 36.94 24.13 19.42 27.42 39.15 28.47 31.02 31.25 43.06
 12.84 32.24 67.57 26.31 35.3  23.8  14.59 55.9  21.5  41.84  4.9  57.23
 71.3  36.25 32.96 61.89 36.8  53.1  11.98 23.85  9.74 14.5  12.18 38.63
 55.65 12.05 27.68 31.35 18.13 12.79 37.36 39.4  29.45 33.36 16.26 39.64
 23.79 45.94 61.07 31.03 33.19 74.7  62.5  27.77 31.88  9.99 25.08 13.22
 56.34 43.57 46.68  6.88 30.65 33.72 30.39 37.72 42.8  26.74 32.53 33.21
```

55.51 41.15 29.59 32.84 33.01 11.47 23.84 56.4 22.35 55.26 31.45 67.7
60.28 17.24 44.42 37.68 35.34 52.42 57.03 48.59 12.47 44.28 48.85 17.58
10.03 40.6 43.38 14.2 24.48 59.49 63.4   7.68 9.85 22.53 32.88 13.71
60.95 44.87 77.3 61.99 81.75 30.88 22.44 26.77 68.1 52.91 32.33 33.95
39.46 25.45 45.84 18.03 24.29 29.07 46.93 59.89 39.49 22.93 52.2 19.52
26.97 66.42 11.96 14.7 13.62 26.94 23.64 24.   25.75 51.72 29.87 35.76
64.9   2.33 18.02 69.3 27.83 50.51 6.47 18.   9.73 24.58 71.7 41.54
26.2 52.43 48.79 79.3   3.32 40.15 12.25 29.23 40.23 34.74 26.86 33.27
20.77 67.11 79.4 42.29 37.8 49.77 51.73 65.91 15.62 61.23 32.72 13.66
36.97 55.16 12.46 40.86 74.5 33.76 55.64 43.58 66.   41.64 43.94 42.23
29.72 51.86 12.37 26.23 39.38 70.7 56.83 26.91 17.6 37.4 42.13 9.13
37.92 51.04 52.12 45.9 38.07 55.55 34.67 15.42 42.22 19.77 24.1 74.17
14.8 13.46 30.22 22.14 51.06 31.84 64.02 31.54 34.4 44.7 17.96 40.39
25.62 32.66 16.89 47.81 31.81 33.42 7.32 19.69 28.3 52.96 76.8 47.78
33.73 45.37 42.03 32.63 46.24 53.52 50.73 51.26 32.1 17.37 37.81 40.93
 7.51 52.5   8.06 24.92 42.62 23.69 33.7 11.17 46.25 54.32 26.32 42.7
17.34 33.69 48.7 31.64 35.85 50.7 25.97 4.83 35.17 32.07 43.89 28.02
25.72 9.01 21.95 47.13 35.96 15.69 32.76 25.46 10.73 49.2 56.14 23.08
56.62 55.5 30.12 39.42 33.09 39.36 37.43 15.52 41.68 4.57 54.6 24.05
31.42 58.61 65.7 10.76 57.22 17.28 57.6 54.28 13.29 13.82 27.74 74.19
42.92 17.22 51.33 6.9 15.82 61.09 24.24 60.29 48.4 53.39 17.84 8.2
38.77 37.44 35.75 31.72 33.04 40.56 55.02 45.7 41.24 37.26 74.99 21.18
38.   50.53 25.37 43.7 44.33 39.23 32.05 71.99 12.45 38.8 66.78 40.87
73.3 21.06 50.94 40.06 42.14 25.12 27.87 39.84 44.14 64.3 33.49 76.24
10.54 30.45 13.54 46.23 36.99 31.97 52.82 49.99 31.65 12.64 22.9 29.
78.8 33.94 39.7   8.37 25.2 25.18 38.46 15.09 51.96 15.44 19.54 40.66
24.39 35.87 51.02 14.54 58.52 26.06 32.4 68.75 33.12 19.2 72.99 50.77
48.99 19.93 24.43 47.97 46.9 50.95 58.78 29.16 6.81 46.2 72.1 15.03
43.73 6.28 47.1 27.66 27.94 47.22 32.9 13.33 47.4 25.56 42.42 25.57
43.5 26.14 18.28 13.12 72.3 50.66 33.66 44.03 23.25 55.25 66.7 40.57
30.57 44.21 61.92 42.55 60.2 56.06 36.56 7.75 33.05 32.25 10.34 74.36
36.45 13.52 28.94 24.9 53.72 13.2 11.36 38.89 21.82 14.14 33.54 69.84
38.6 56.1 19.11 30.23 61.86 59.59 55.2 14.84 38.33 27.04 39.78 40.76
53.3 47.74 43.25 25.51 18.91 31.18 29.93 35.36 54.38 11.85 52.3 57.21
 8.   54.1 14.31 41.72 40.71 29.55 18.75 24.4   8.49 26.4 66.95 12.54
39.94 55.94 12.73 39.27 59.76 24.85 36.84 11.39 9.31 22.72 23.14 11.65
31.9   7.84 25.22 20.08 46.8 39.59 13.4 15.05 35.31 15.07 29.41 24.45
26.15 48.15 47.03 33.96 33.61 32.82 15.57 16.28 38.5   9.87 34.49 11.48
68.3 24.54 9.62 36.44 14.99 9.69 10.35 26.05 37.23 53.77 39.06 43.8
42.64 53.69 34.29 27.34 7.4 21.16 21.78 30.85 53.46 22.49 44.09 33.
10.22 13.09 39.32 50.08 9.45 79.99 14.6 20.59 25.73 69.66 18.2 48.97
56.81 19.35 18.42 21.48 47.28 21.07 50.6 38.02 29.79 15.61 29.65 43.39
21.91 48.28 22.75 19.99 36.96 39.05 33.06 31.12 20.92 30.14 44.4 24.5
71.62 36.3 35.86 4.78 33.56 23.35 32.92 41.37 32.85 35.08 38.2 23.4
20.97 13.18 19.01 14.64 11.58 56.74 23.7 37.17 32.11 38.61 33.02 52.01
 7.72 28.68 59.09 24.66 17.57 46.39 31.27 66.9 24.34 20.87 30.44 29.75
24.07 52.44 67.87 39.09 38.56 56.61 22.32 6.27 25.69 61.24 67.8 24.89
47.82 35.57 58.8 35.1 21.75 28.8 30.28 15.58 34.9 14.4 55.83 15.87

```
 43.01 15.36 22.63 33.31 34.77 56.7  63.53 26.85 37.33 38.7  73.7  53.9
 53.96 21.29 10.09 17.44 39.61 41.1  52.83  8.54 38.41 22.5  41.89 44.13
 48.72 51.43 44.61 57.92 15.53 66.6  41.16 17.95 39.6  34.56 46.64 27.63
 41.93 56.5  32.77 53.58 28.6  62.05 66.1  17.82 23.22 34.2  24.99 39.44
 38.11 21.6  34.68 38.21 30.96 22.95 55.6  15.34 54.77 24.44 33.08 67.31
 59.8  55.45 27.53 52.45 16.5  56.63 29.73 31.38 36.59 41.67 82.6  40.27
 42.33 56.85 25.48 40.2  59.   36.15 17.17 44.39 21.02 21.26 63.14 68.5
 13.57 39.66 14.94 25.61 44.64 39.58 25.89 31.87 41.94 61.8  66.82 32.01
 53.66 21.97 28.24 40.29 25.42 75.5  25.02 52.52 34.24 27.23 39.45 41.2
 37.96 22.84 80.2  27.22 50.24 16.11 59.3  45.71 12.55 29.39 62.94 10.79
  9.56 10.39 41.3  55.06 52.61]
845
```

[6]:
```
df.isnull().sum()
```

[6]:
```
cement          0
slag            0
ash             0
water           0
superplastic    0
coarseagg       0
fineagg         0
age             0
strength        0
dtype: int64
```

[ ]:
```

```

1.0.4 Insights:

There are no NAN values in the dataset.

I believe the 0's in this dataset could represent instances where the specific additive was not used in the mixture.

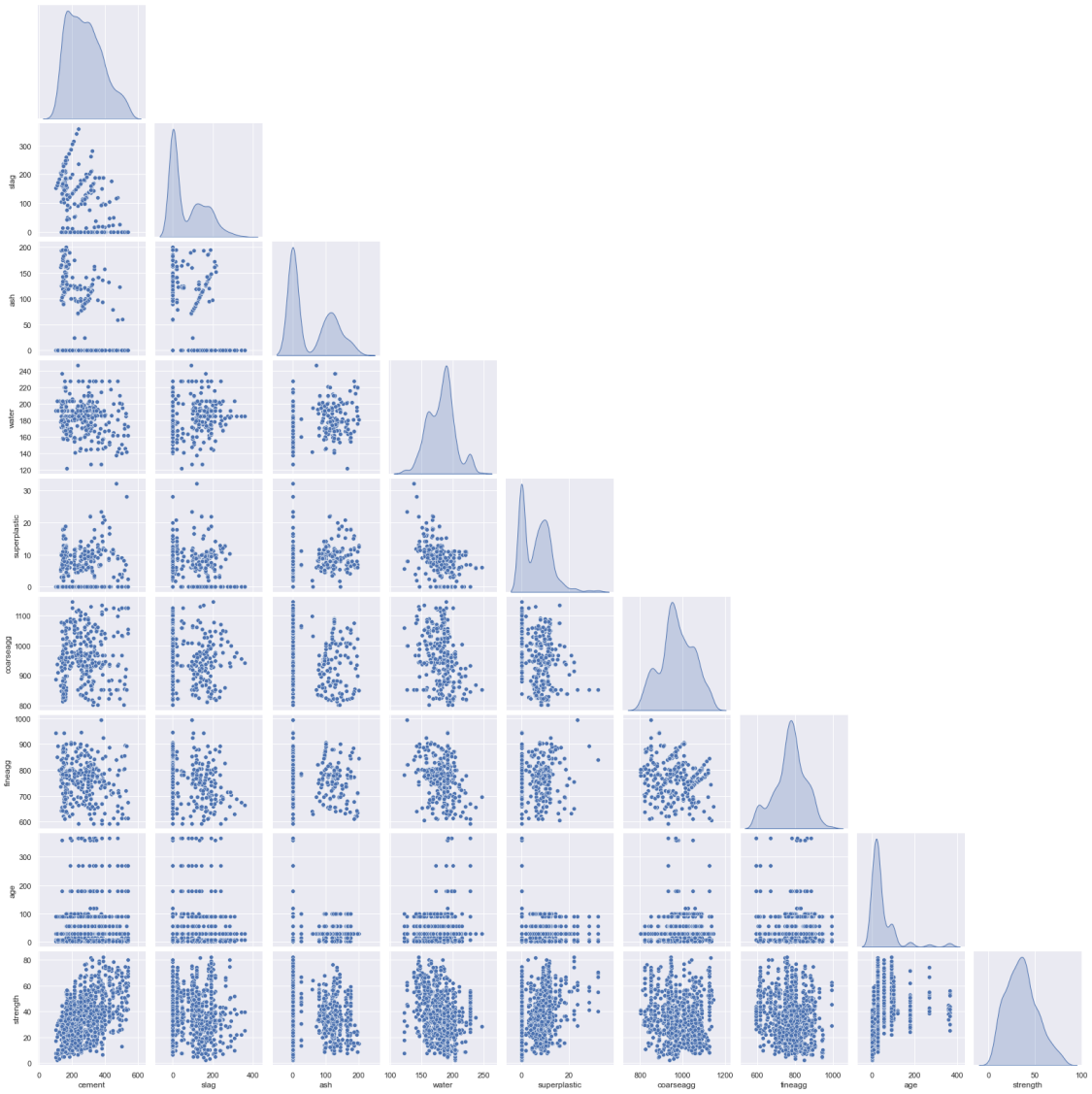
Since concrete is by definition a mixture of broken stone or gravel, sand, cement, and water it should be not need to be explicitly stated that different applications need varying amounts of materials. Therefore, with the exception of cement and water, having 0 values is expected. link to how concrete is made: https://www.cement.org/cement-concrete/how-concrete-is-made#:~:text=A%20properly%20designed%20mixture%20possesses,15%20to%2020%20percent%20water.

2 Bi-variate analysis between the predictor variables and also between the predictor variables and target column.

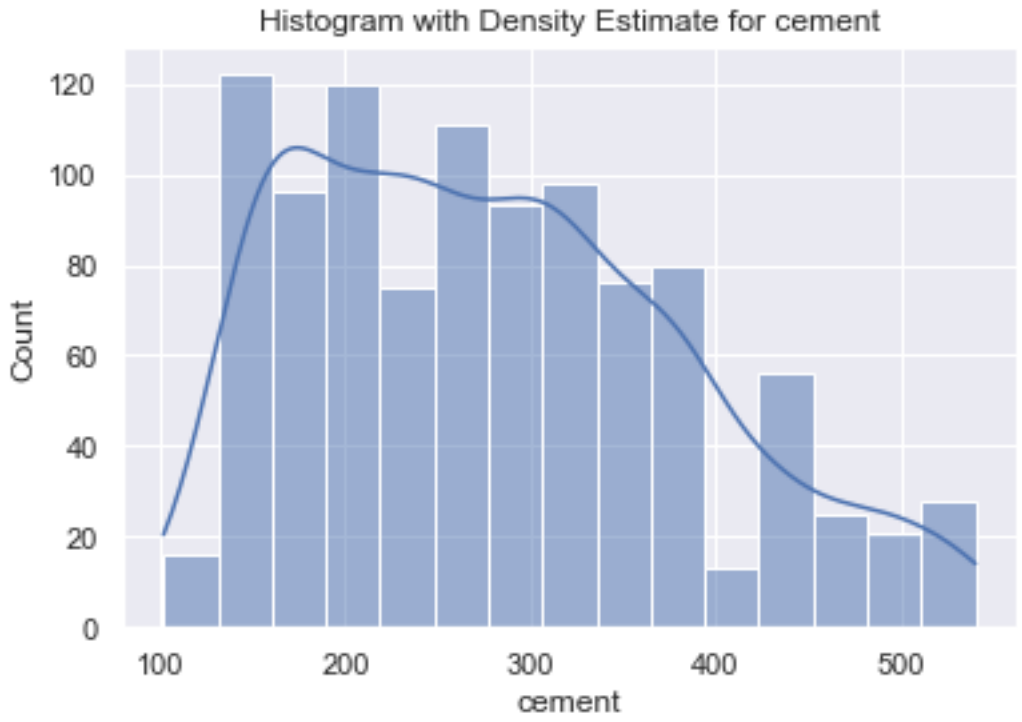
2.0.1 Comment on your findings in terms of their relationship and degree of relation if any. Visualize the analysis using boxplots and pair plots, histograms, or density curves and write the insights/business understanding of the same. (8 marks)

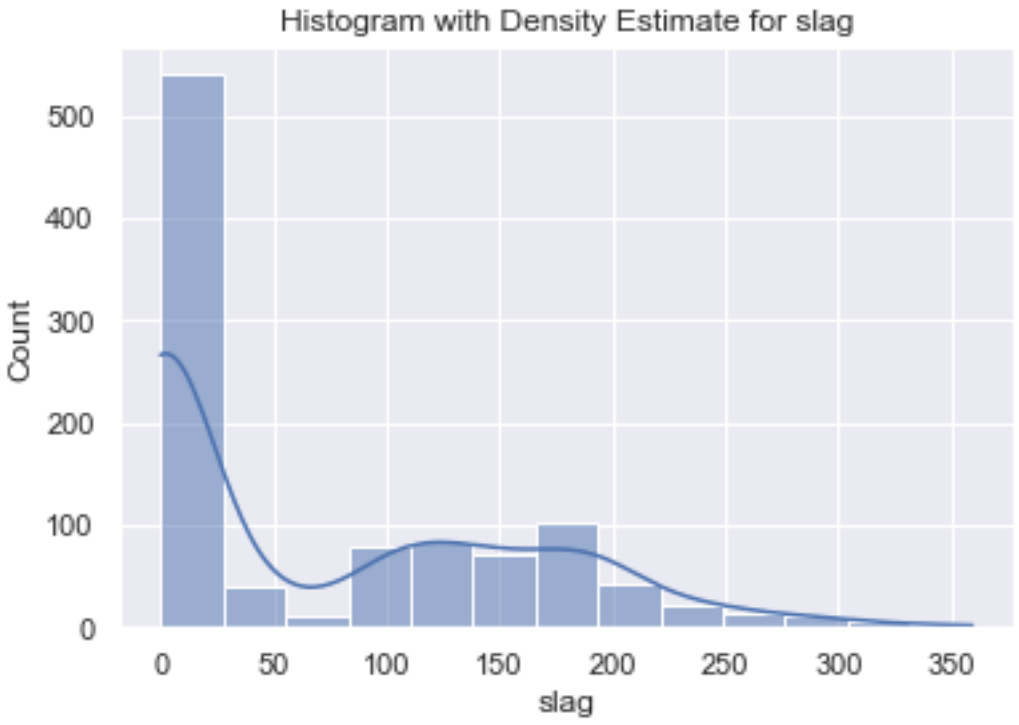
[7]:
```
plt.figure(figsize=(12,12));
sns.pairplot(df, diag_kind="kde", corner=True);
```

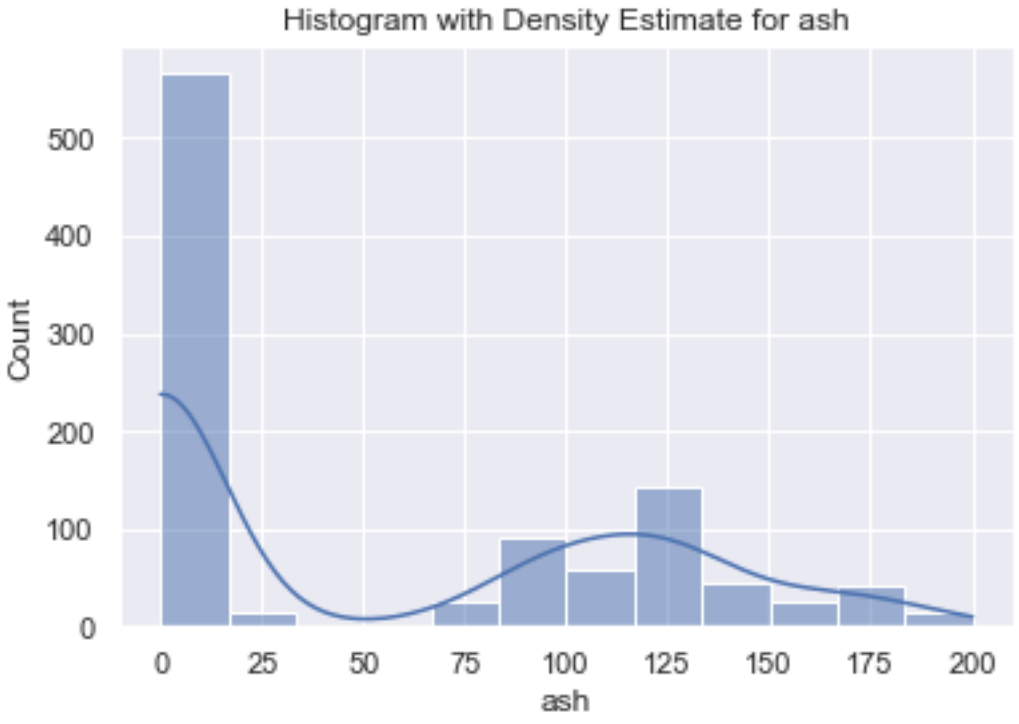
```
<Figure size 864x864 with 0 Axes>
```

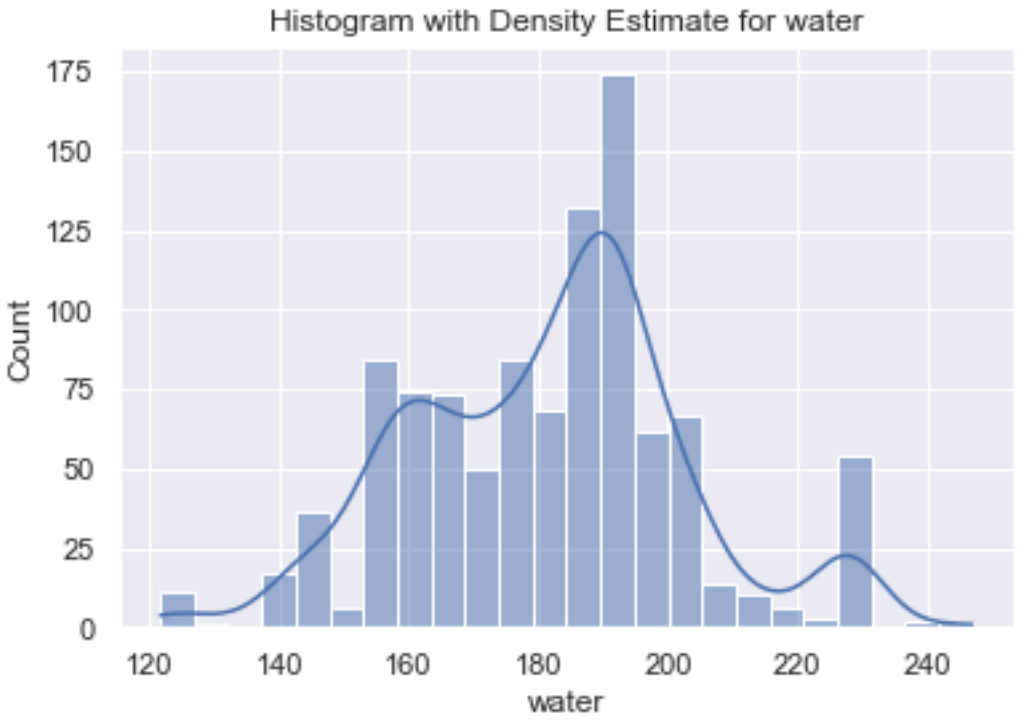


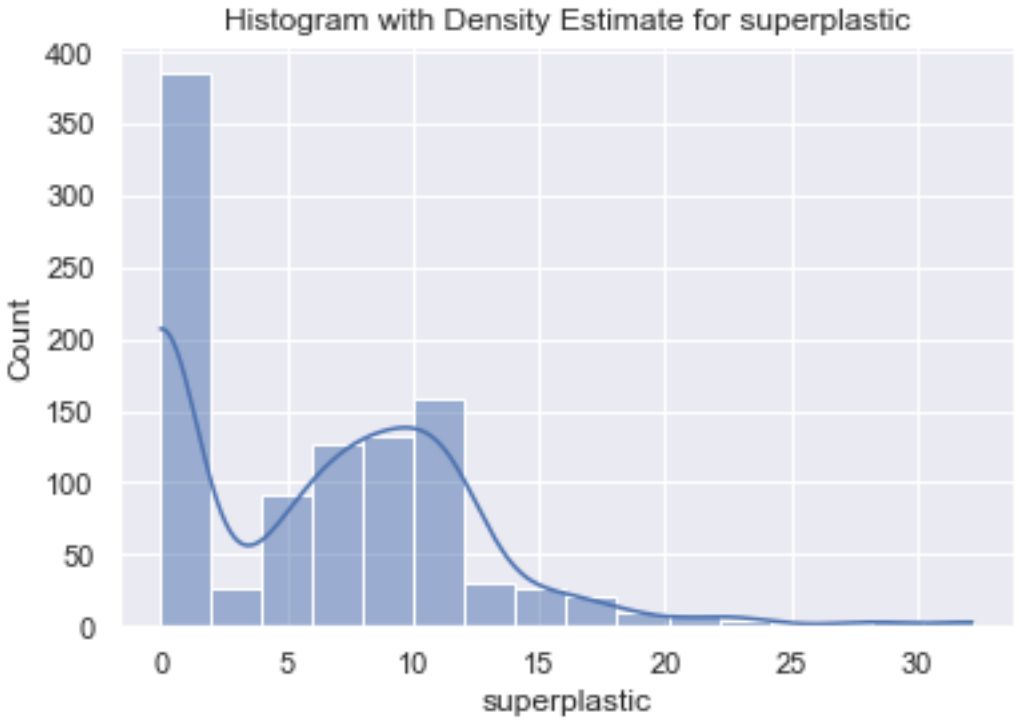
[8]:
```
for col in df.columns:
    sns.histplot(df, x=col, kde=True)
    temp = "Histogram with Density Estimate for " + col
    plt.title(temp)
    plt.show()
```

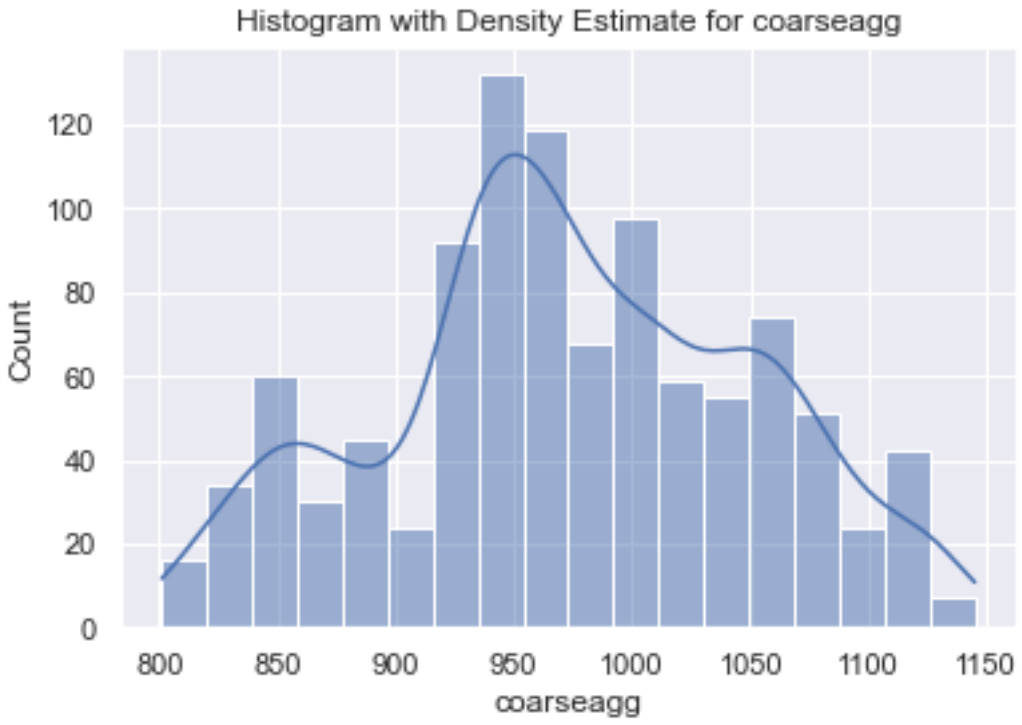


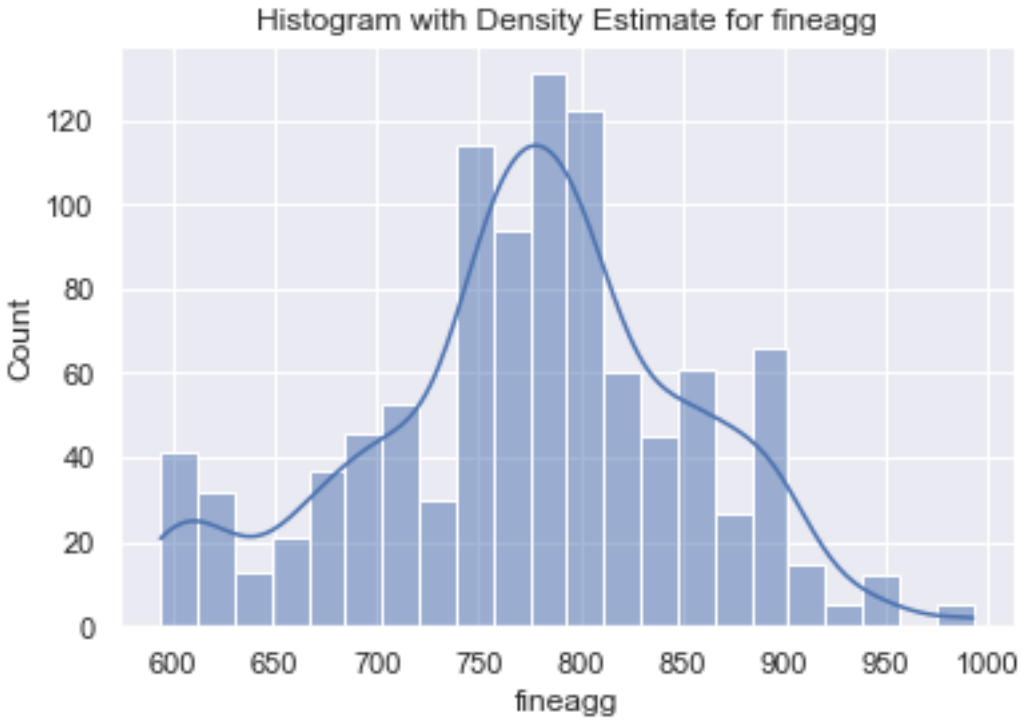

Histogram with Density Estimate for slag
Count
500
400
300
200
100
0
0
50
100
150
200
250
300
350
slag

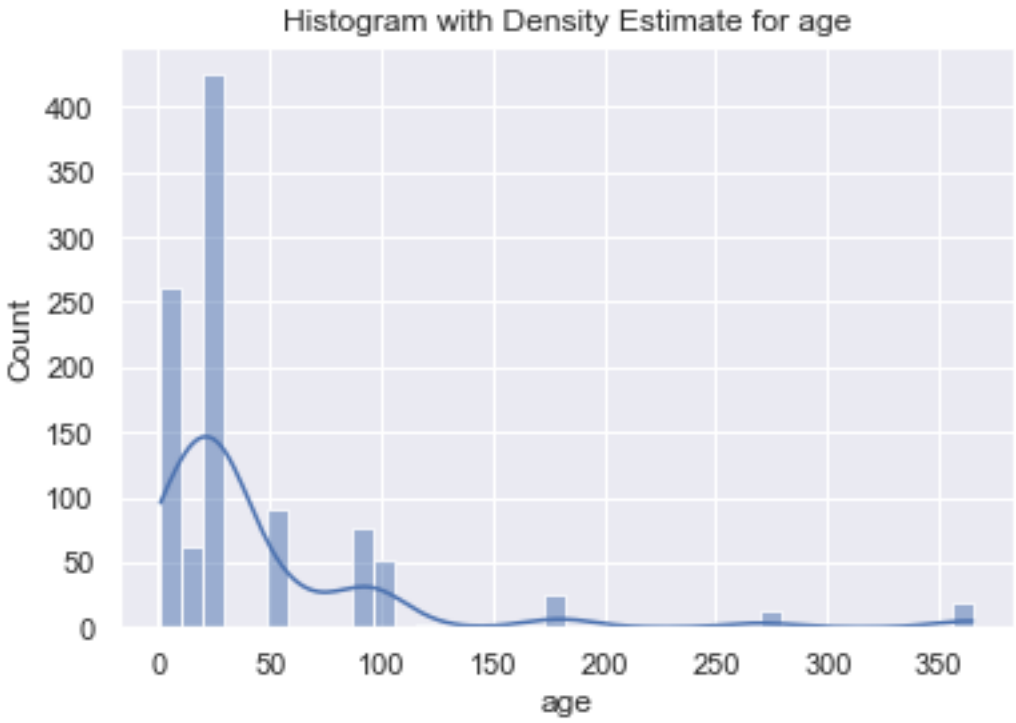

Histogram with Density Estimate for ash
Count
500
400
300
200
100
0
0
25
50
75
100
125
150
175
200
ash

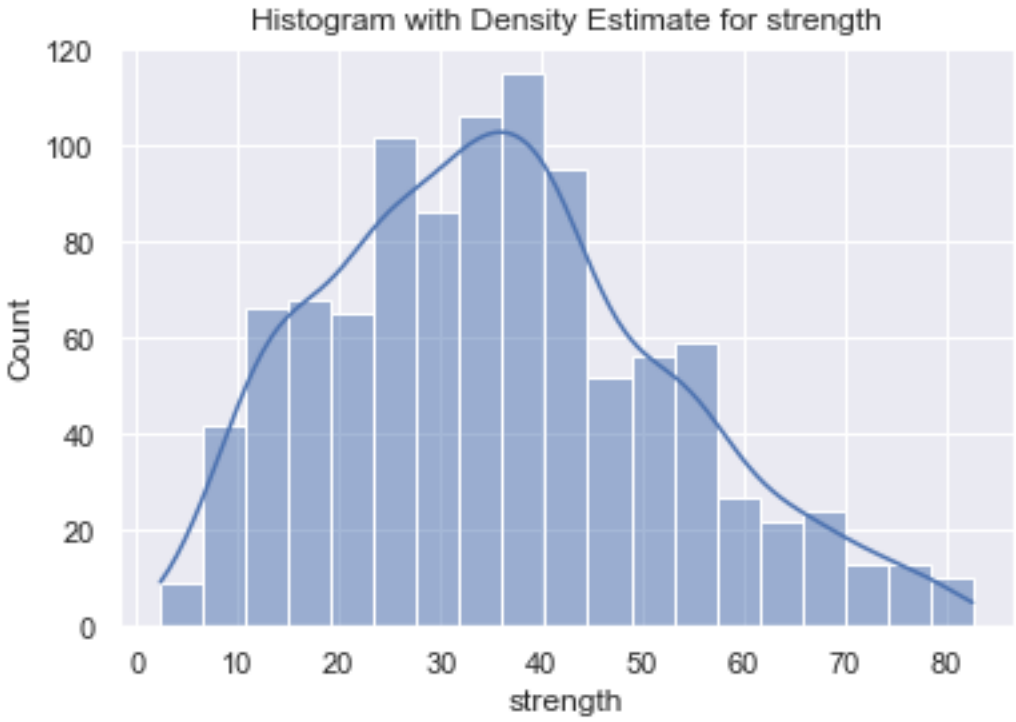

Histogram with Density Estimate for water
Count
175
150
125
100
75
50
25
0
120
140
160
180
200
220
240
water


Histogram with Density Estimate for superplastic
Count
400
350
300
250
200
150
100
50
0
0
5
10
15
20
25
30
superplastic

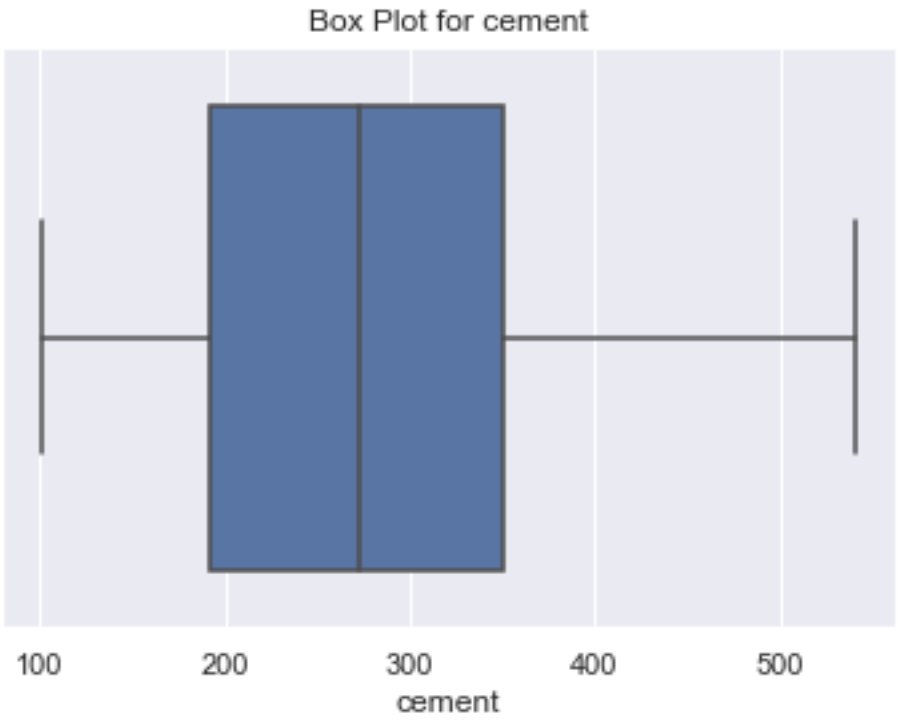

Histogram with Density Estimate for coarseagg
Count
0
20
40
60
80
100
120
800
850
900
950
1000
1050
1100
1150
coarseagg

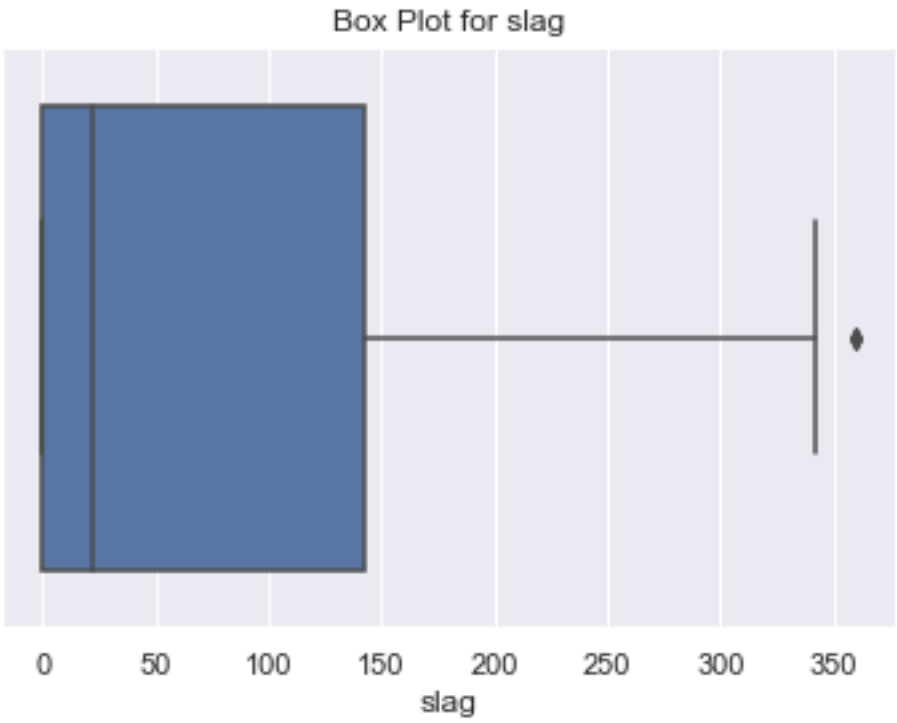

Histogram with Density Estimate for fineagg
Count
0
20
40
60
80
100
120
600
650
700
750
800
850
900
950
1000
fineagg

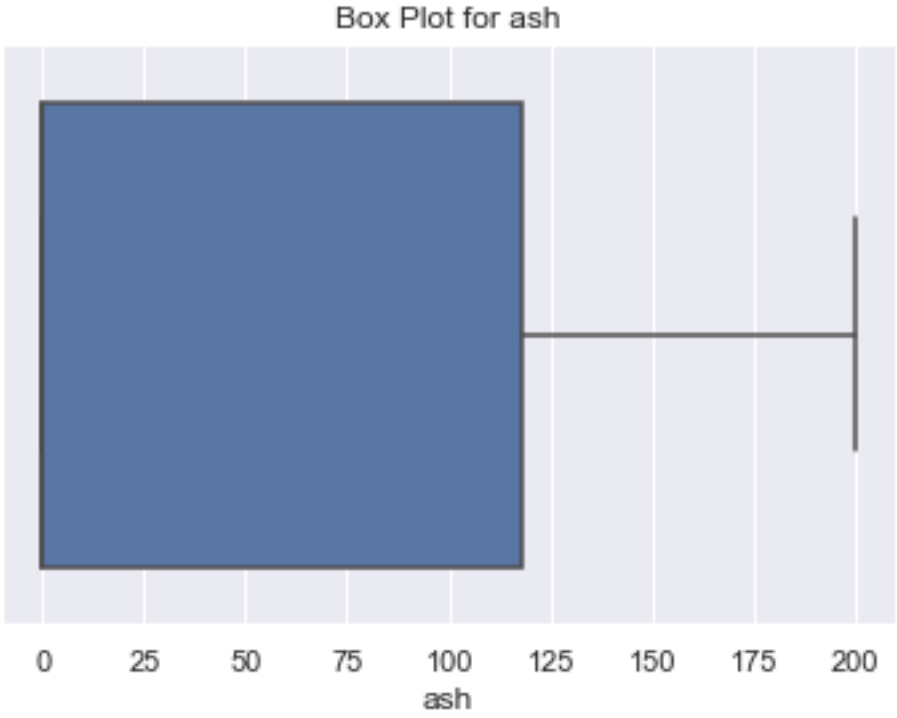

Histogram with Density Estimate for age
Count
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
age

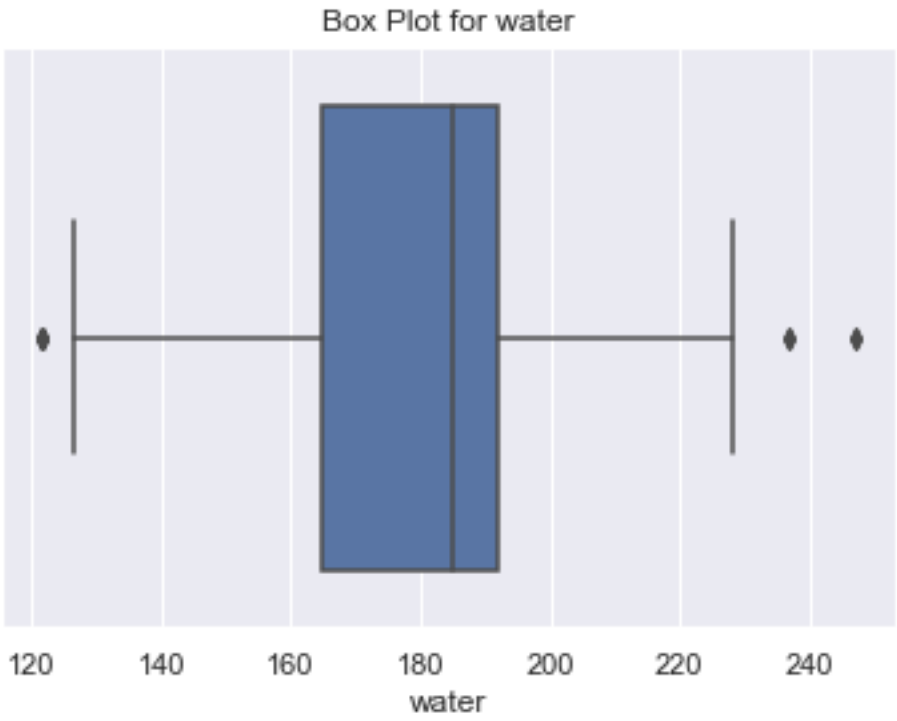

Histogram with Density Estimate for strength
Count
120
100
80
60
40
20
0
0
10
20
30
40
50
60
70
80
strength

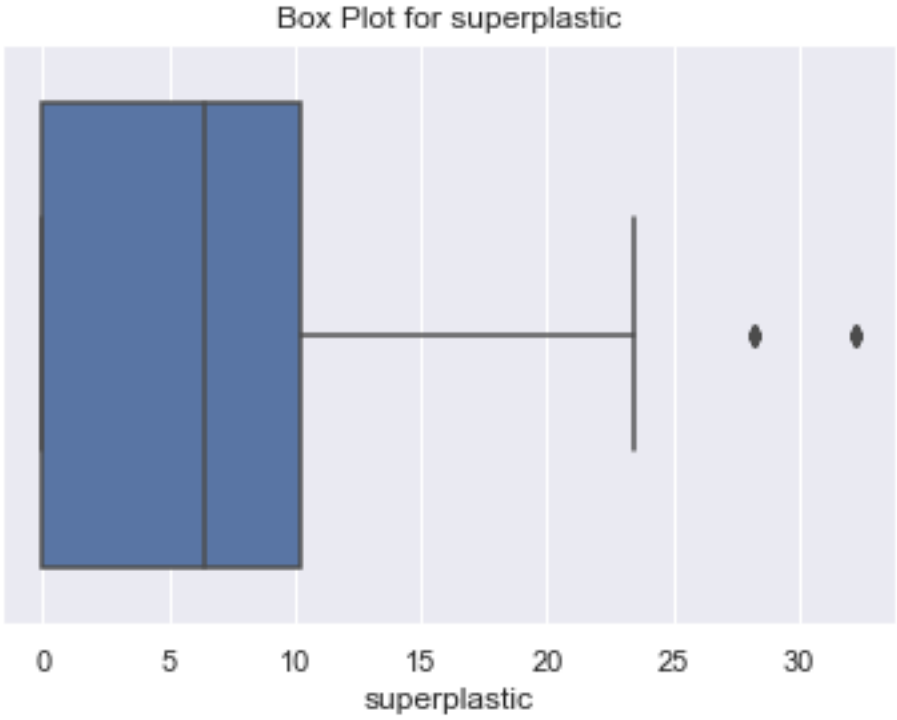
[9]:
```
for col in df.columns:
    sns.boxplot(x=df[col])
    temp = "Box Plot for " + col
    plt.title(temp)
    plt.show()
```

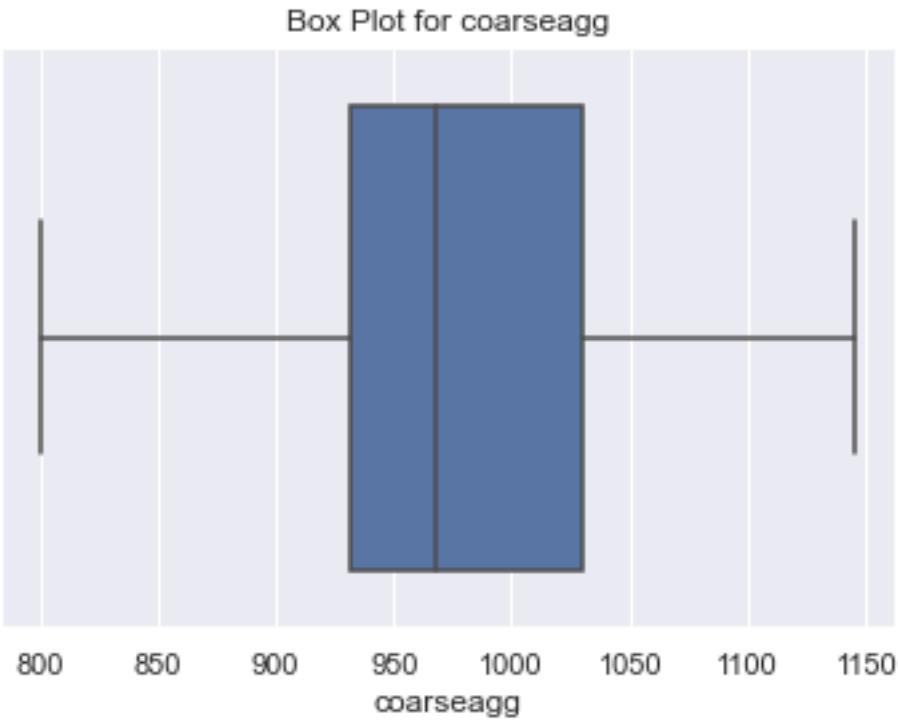


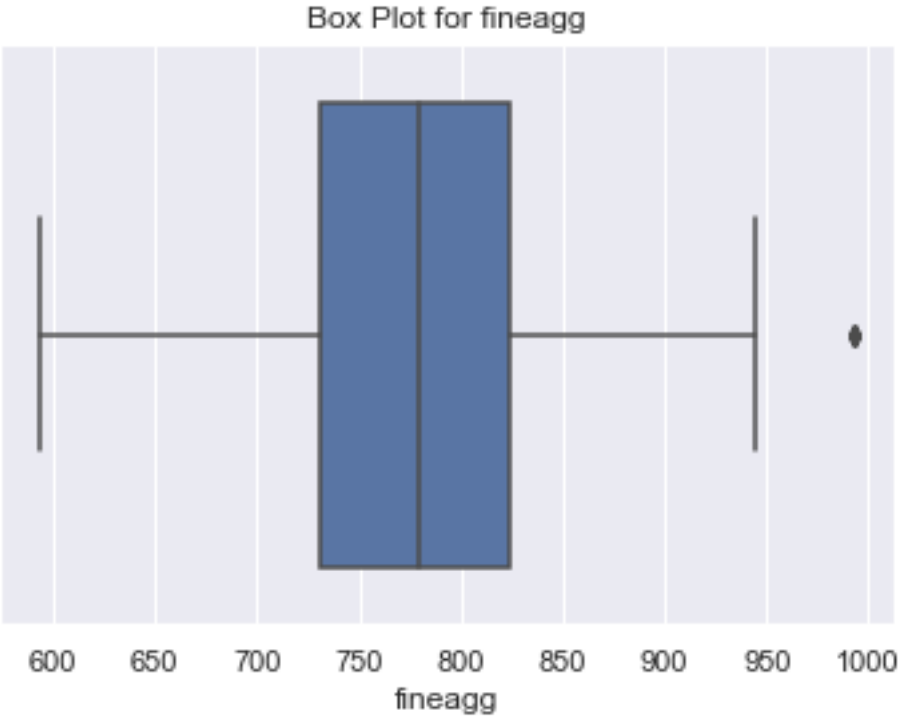

Box Plot for slag
0
50
100
150
200
250
300
350
slag

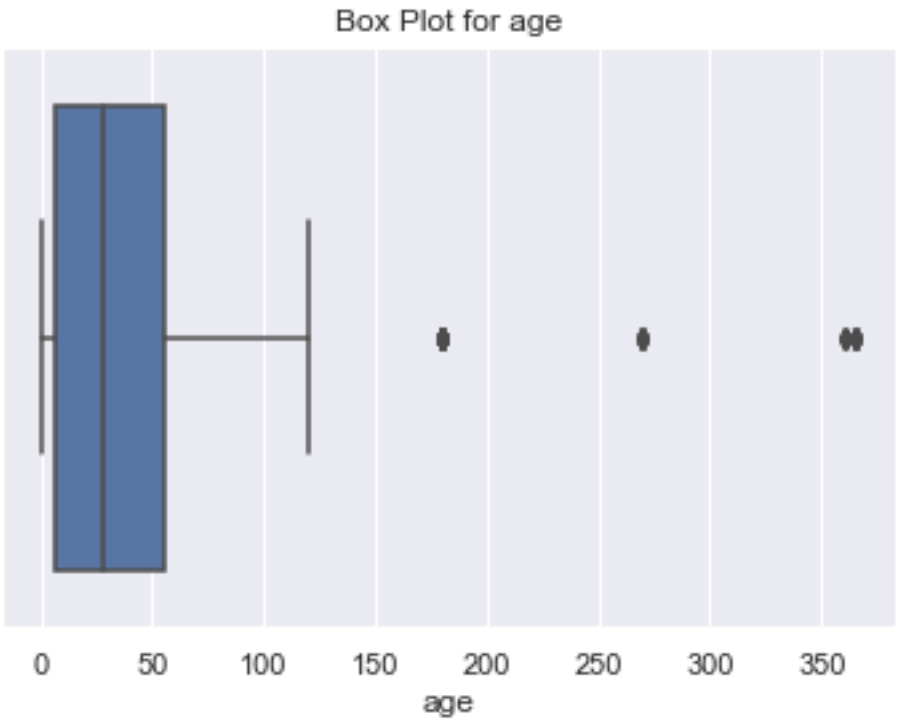

Box Plot for ash
0
25
50
75
100
125
150
175
200
ash

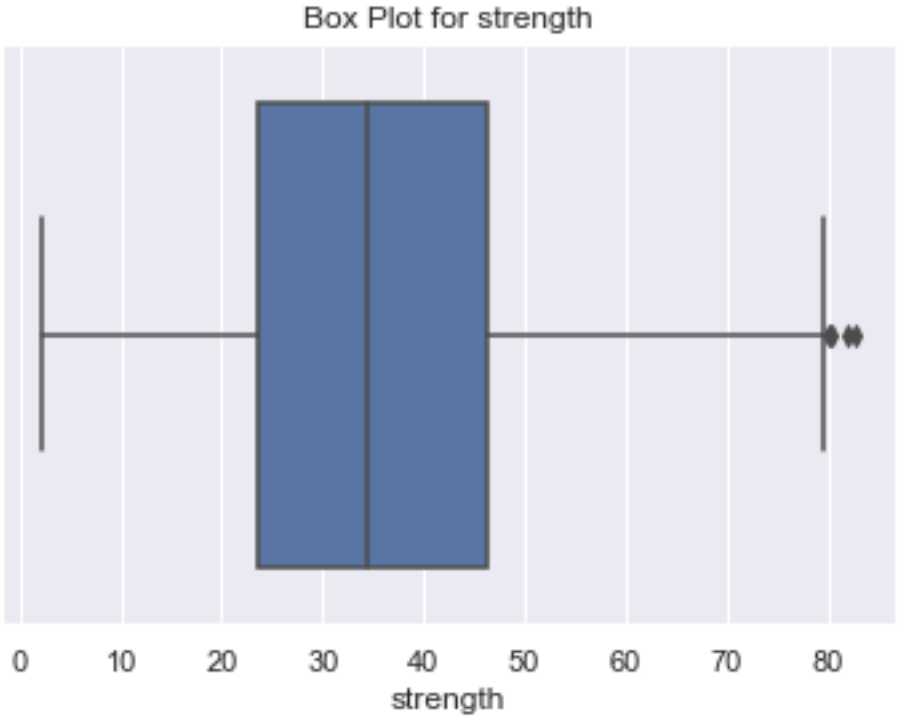

Box Plot for water
120
140
160
180
200
220
240
water


Box Plot for superplastic
0
5
10
15
20
25
30
superplastic

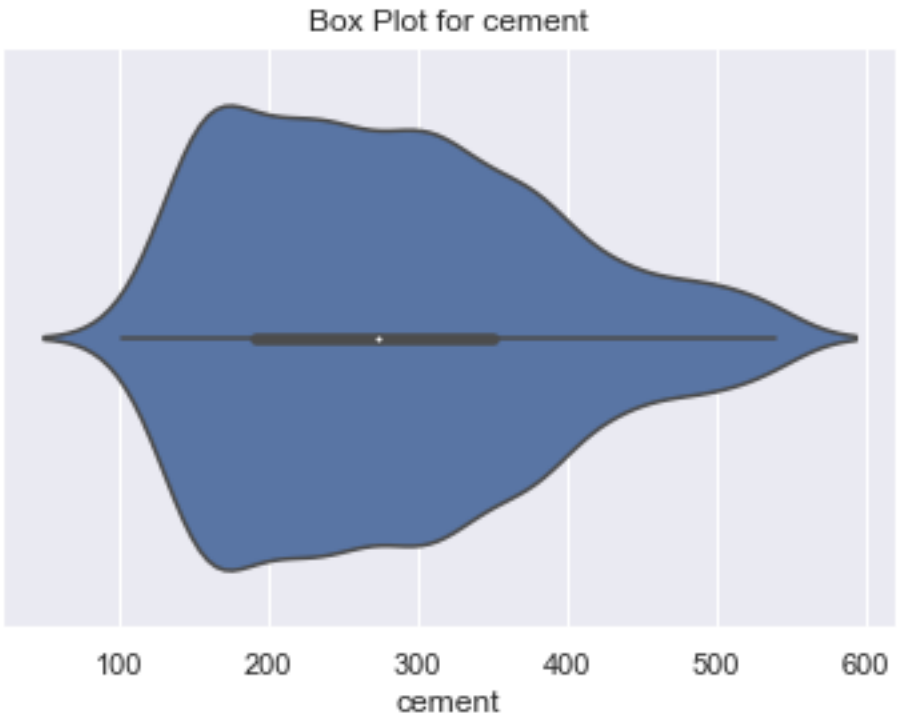

Box Plot for coarseagg
800
850
900
950
1000
1050
1100
1150
coarseagg

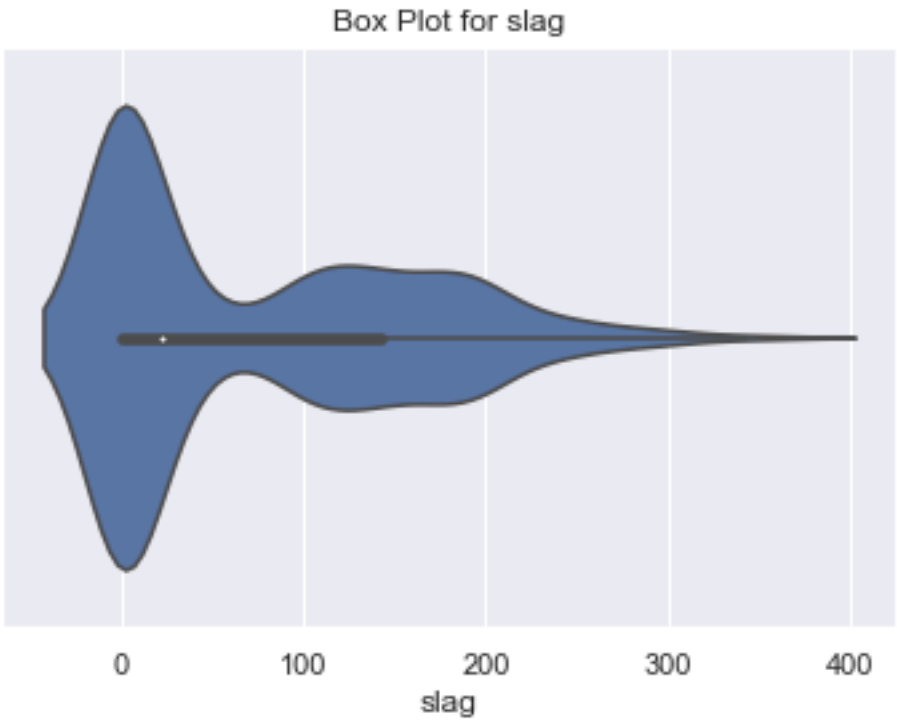

Box Plot for fineagg
600
650
700
750
800
850
900
950
1000
fineagg

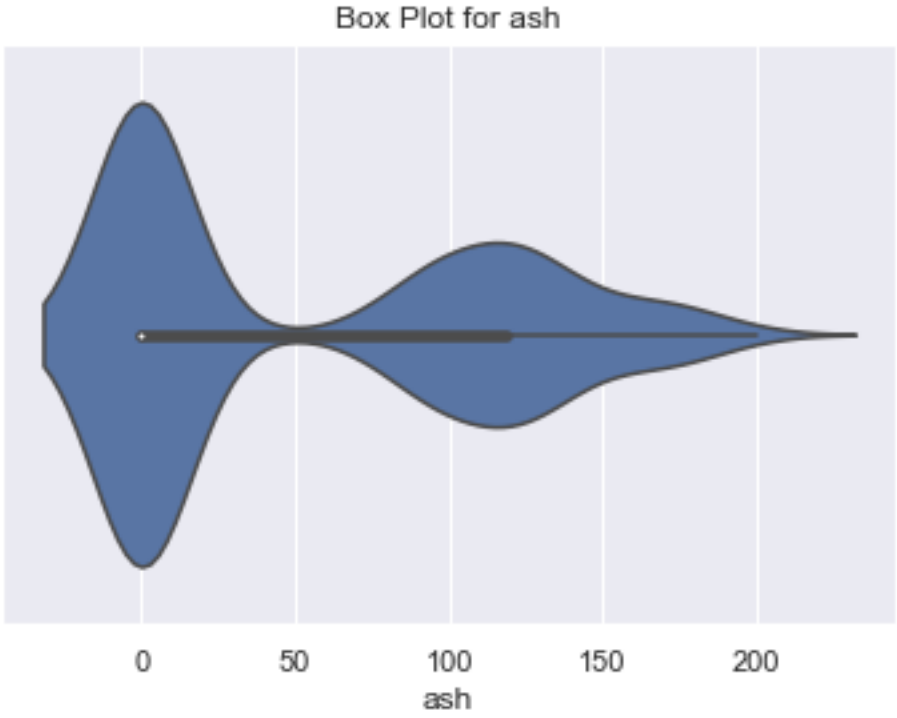

Box Plot for age
0
50
100
150
200
250
300
350
age

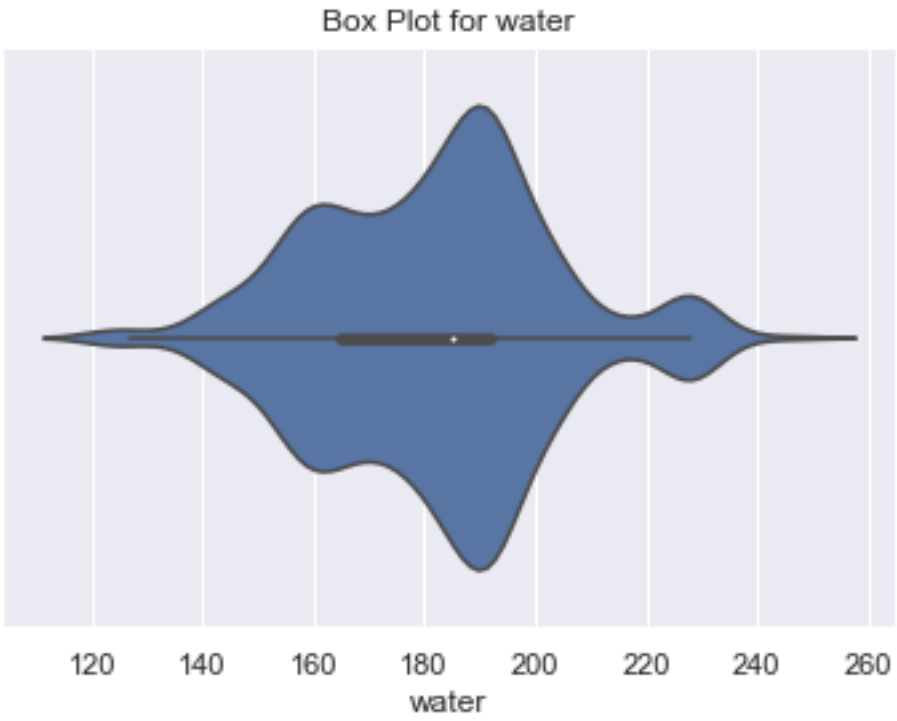

Box Plot for strength
0
10
20
30
40
50
60
70
80
strength

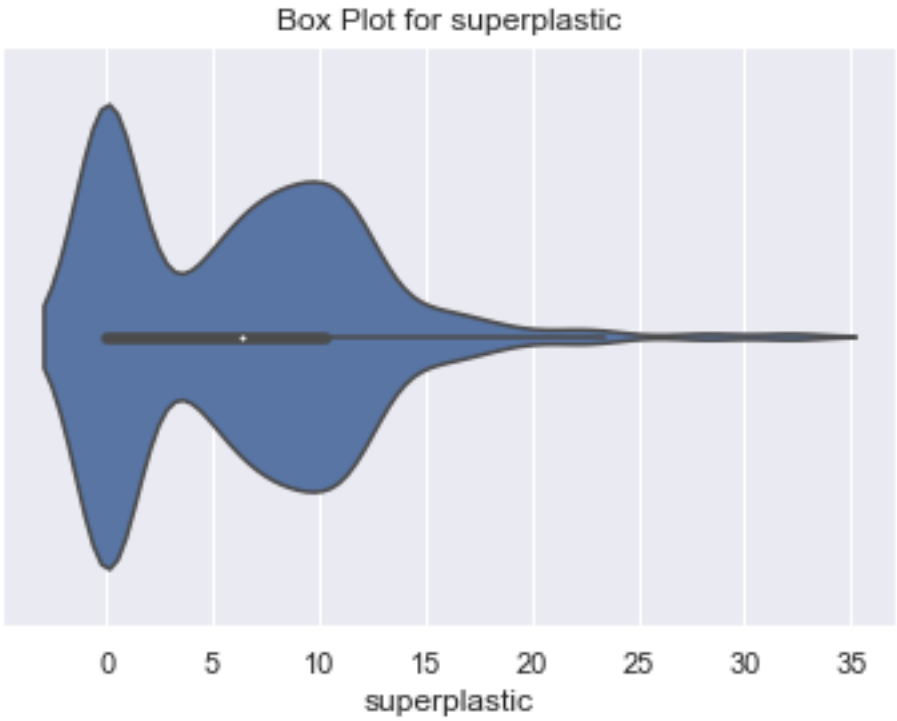
[10]:
```
for col in df.columns:
    sns.violinplot(x=df[col])
    temp = "Box Plot for " + col
    plt.title(temp)
    plt.show()
```

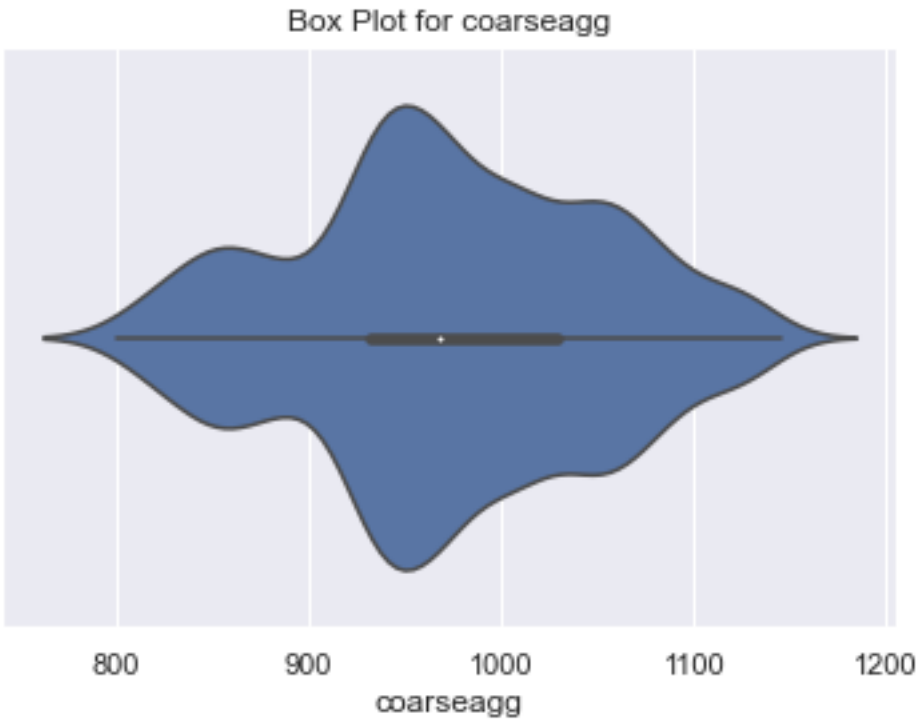


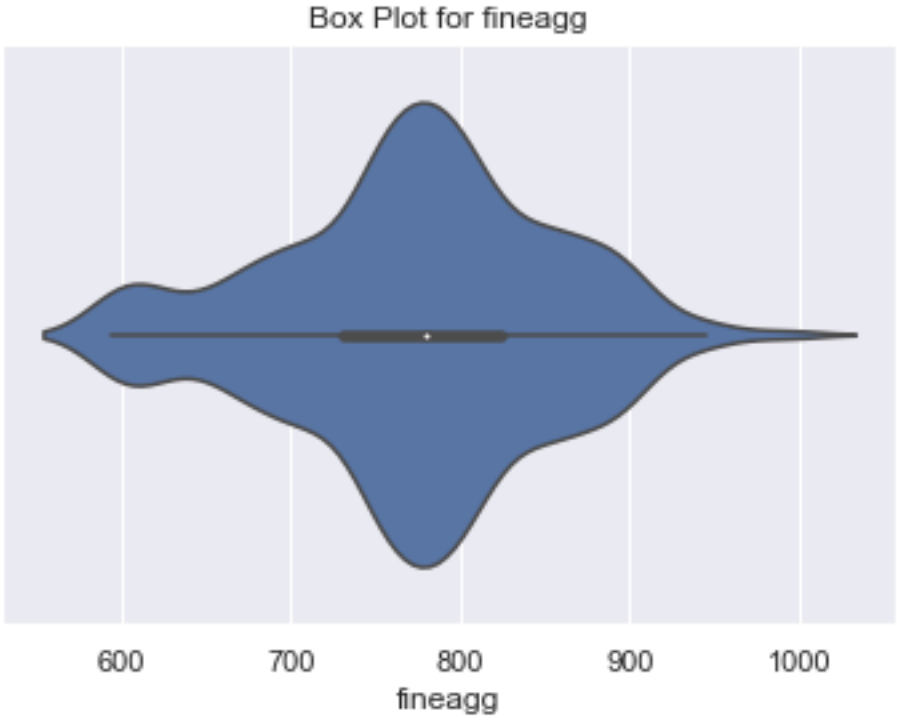

Box Plot for slag
0
100
200
300
400
slag

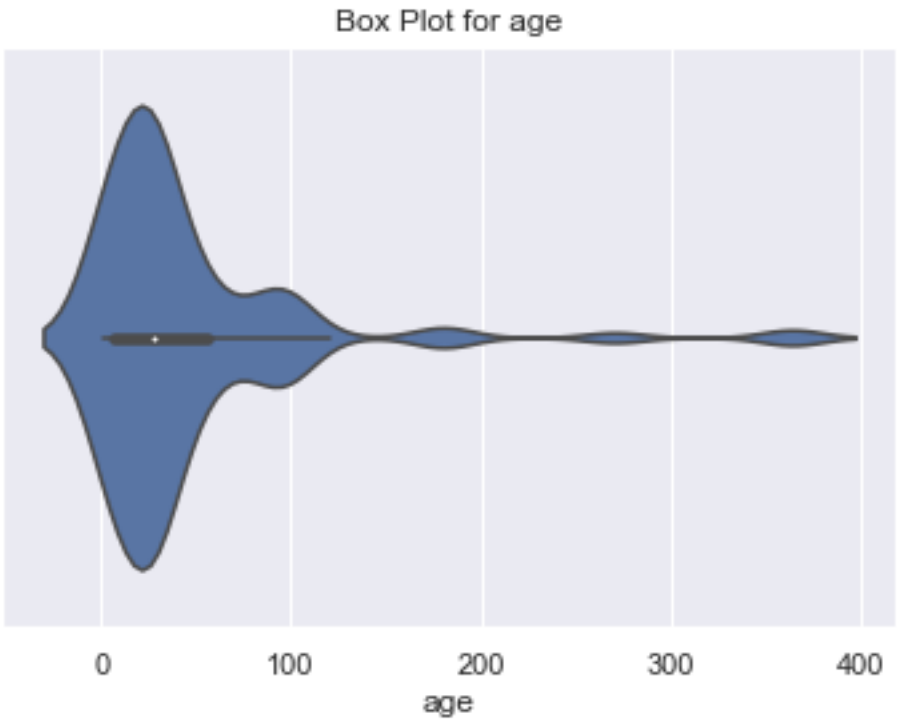

Box Plot for ash
0
50
100
150
200
ash

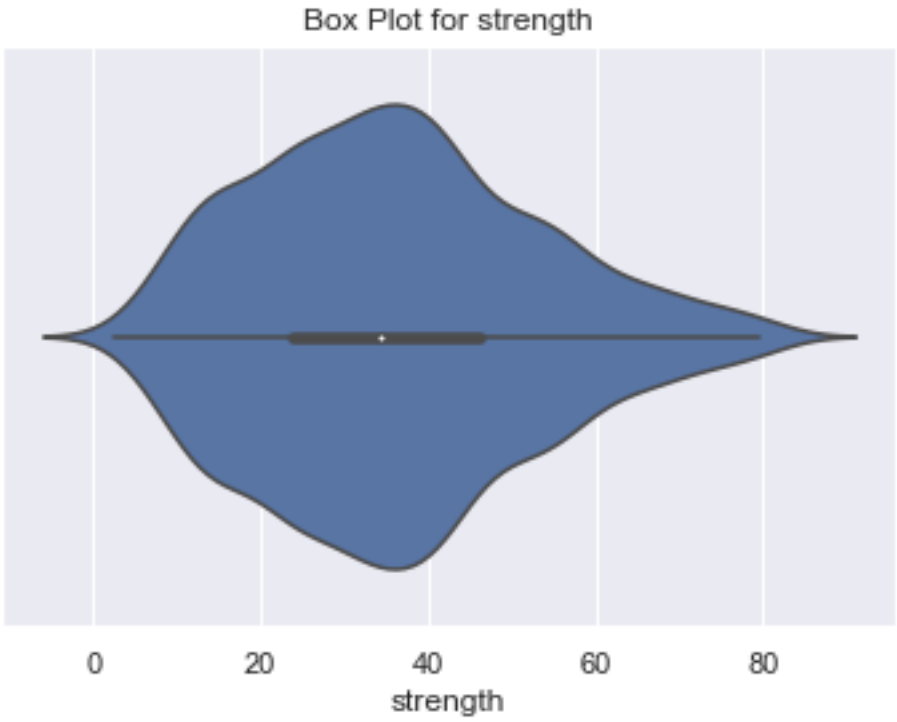

Box Plot for water
120
140
160
180
200
220
240
260
water


Box Plot for superplastic
0
5
10
15
20
25
30
35
superplastic

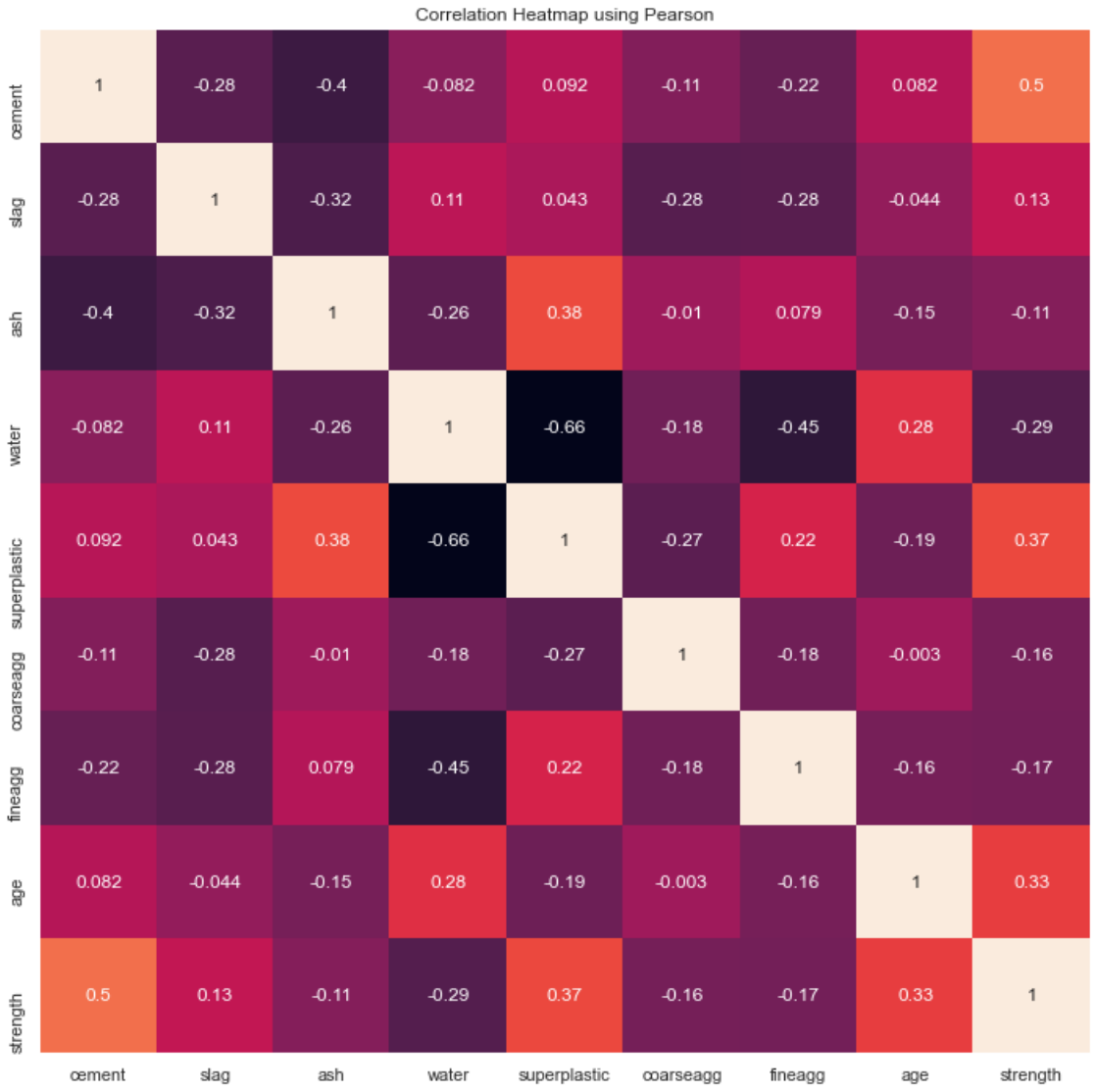

Box Plot for coarseagg
800
900
1000
1100
1200
coarseagg


Box Plot for fineagg
600
700
800
900
1000
fineagg

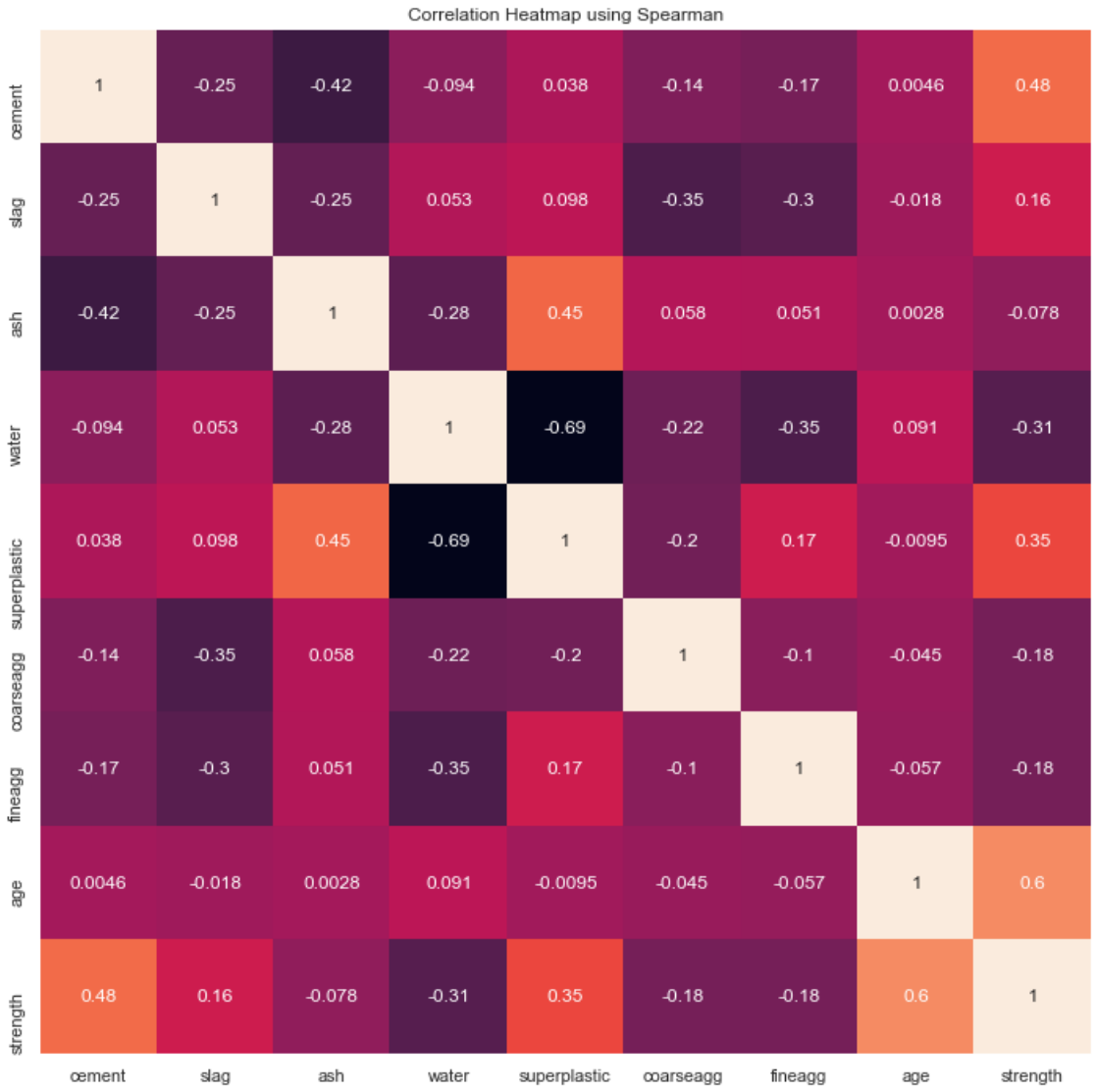

Box Plot for age
0
100
200
300
400
age


Box Plot for strength
0
20
40
60
80
strength

[11]:
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr("pearson"), cbar=False, annot=True)
plt.title("Correlation Heatmap using Pearson")
plt.show()
```



[12]:
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr("spearman"), cbar=False, annot=True)
plt.title("Correlation Heatmap using Spearman")
plt.show()
```



2.0.2 Using various plots, write the insights/observations about it. (6 Marks)

2.0.3 Insights on data shape/tails

the target variable is relatively normally distributed

Age is right skewed

Fine Aggregate is relatively normally distributed

Coarse Aggregate is relatively normally distributed

Superplastic is right skewed with a mode of 0

Water is relatively normallmy distributed

Ash has a right skew with a mode of 0

Slag has a right skew with a mode of 0

Cement has a slight right tail

2.0.4 Insights on data correlation linear/non-linear

Superplastic and water are strongly inversely correlated (-.69)

Quantity of Cement used is correlated with Strength (.5)

Water and Strength are negatively correlated (-.31)

Age of project is correlated with strength (.6)

Ash and Superplastic are relatively correalated (.45)

Ash and Cement are negatively correlated (-.42)

Ash and water are negatively correlated (-.28)

Fine Aggregate and Water are negatively correlated (-.45)

Ash and Slag are negatively correlated (-.32)

2.0.5 Semi-actionable business insights from plots

the business can add more cement to the mix to increase strength (this comes at great cost and after some threshhold will reduce the effectiveness of concrete.)

the business can add less water to the mix to increase strength (a minimum amount of water is needed to mix cement to the correct consistency, any reduction below this level will result in a mix that has poor consistency, is hard to work, and very brittle. again)

the business can just wait until mix sets which will increase strength (on a building site, time is money.)

3 Feature Engineering techniques

3.0.1 Identify opportunities (if any) to extract new features from existing features, drop a feature(if required) Hint: Feature Extraction, for example, consider a dataset with two features length and breadth. From this, we can extract a new feature Area which would be length * breadth. (3 Marks)

[13]:
```
#combining ash and superplastic into a single variable due to multi-colinearity
df['superplastic+ash'] = df['superplastic'] + df['ash']
# Dropped ash and superplastic
df.drop('ash', axis=1, inplace=True)
df.drop('superplastic', axis=1, inplace=True)
```

3.0.2 Consultant business insight:

3.0.3 model trains at R2 over 90% and tests at over 93%. Client stops paying for performance >90%, no other feature engineering needed to acheive business objectives!

3.0.4 Get the data model ready and do a train test split.

[14]:
```
y = df['strength']
x = df.drop('strength', axis=1)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=.3,␣
↪random_state=6)
```

[15]:
```
#verify ratios
print("the ratio of x_train to x is {}".format(len(x_train)/len(x)))
print("the ratio of x_test to x is {}".format(len(x_test)/len(x)))

print("\n\nthe ratio of y_train to y is {}".format(len(y_train)/len(y)))
print("the ratio of y_test to y is {}".format(len(y_test)/len(y)))
```

```
the ratio of x_train to x is 0.7
the ratio of x_test to x is 0.3


the ratio of y_train to y is 0.7
the ratio of y_test to y is 0.3
```

4 Creating the Model and Tuning It:

4.1 Algorithms that you think will be suitable for this project.

4.1.1 Use Kfold Cross-Validation to evaluate model performance. Use appropriate metrics and make a DataFrame to compare models w.r.t their metrics. (at least 3 algorithms, one bagging and one boosting based algorithms have to be there and try to achieve 90% accuracy on testing set). (7 Marks)

[16]:
```
models = ["Ridge Regression", 'DecissionTreeRegressor',␣
↪'RandomForestRegressor', 'BaggingRegressor', 'GradentBoostingRegressor',␣
↪'SVR', 'AdaboostRegressor'] #, 'Neural Network']
x_val_score, testing_score = [],[]
def scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test):
    pipe = Pipeline([('minmaxscaler', RobustScaler()),('regressor',regressor)])␣
↪#not needed at this time, but included as future proofing
    model = GridSearchCV(pipe, param_grid, cv=10)
    model.fit(x_train, y_train)

    print("Best parameters (CV score=%0.3f):" % model.best_score_)
    print(model.best_params_)
```

```
    y_pred = model.predict(x_test)
    print("Testing Score ", r2_score(y_test, y_pred))
    x_val_score.append(round(model.best_score_,3))
    testing_score.append(round(r2_score(y_test, y_pred),3))
```

[17]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'poly__degree': [1, 2, 3, 4, 5],
    'ridge__alpha': [.4,.5,.6,.7,.8,.9],
}


poly = PolynomialFeatures()
ridge = linear_model.Ridge()
pipe = Pipeline([('minmaxscaler',␣
 ↪RobustScaler()),('poly',poly),('ridge',ridge)])
model = GridSearchCV(pipe, param_grid, cv=10)
model.fit(x_train, y_train)

print("Best parameters (CV score=%0.3f):" % model.best_score_)
print(model.best_params_)

y_pred = model.predict(x_test)
print("Testing Score ", r2_score(y_test, y_pred))
x_val_score.append(model.best_score_)
testing_score.append(r2_score(y_test, y_pred))
```

```
Best parameters (CV score=0.800):
{'poly__degree': 4, 'ridge__alpha': 0.5}
Testing Score  0.8728865518829478
```

4.1.2 Decide on the complexity of the model, should it be a simple linear model in terms of parameters or would a quadratic or higher degree. (5 Marks)

The best fit for ridge regression is a polynomial of degree 3 with an alpha value of .4

[18]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'regressor__criterion':['mse', 'friedman_mse', 'mae', 'poisson'],
    'regressor__max_features' : ['None', 'auto', 'sqrt', 'log2'],
    'regressor__max_depth': ['None', 10, 50, 100, 150, 200, 250, 300],
}

regressor = DecisionTreeRegressor()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.807):
```

```
{'regressor__criterion': 'mse', 'regressor__max_depth': 150,
'regressor__max_features': 'auto'}
Testing Score  0.8558920306818429
```

[19]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'regressor__criterion':['mse', 'mae'],
    'regressor__max_features' : ['auto', 'sqrt', 'log2'],
    'regressor__max_depth': ['None', 10, 50, 100, 150, 200, 250, 300],
    'regressor__n_estimators':[150, 200, 250, 300, 350, 400],
}

regressor = RandomForestRegressor()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.892):
{'regressor__criterion': 'mae', 'regressor__max_depth': 50,
'regressor__max_features': 'auto', 'regressor__n_estimators': 200}
Testing Score  0.9171517552763724
```

[20]:
```
# Parameters of pipelines can be set using '__' separated parameter names:

param_grid = {
    'regressor__base_estimator':[DecisionTreeRegressor(), SVR()],
    'regressor__n_estimators':[150, 200, 250, 300, 350, 400],
}

regressor = BaggingRegressor()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.889):
{'regressor__base_estimator': DecisionTreeRegressor(),
'regressor__n_estimators': 350}
Testing Score  0.9146646365167502
```

[32]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'regressor__loss':['ls', 'lad', 'huber', 'quantile'],
    'regressor__n_estimators':[200, 250, 300, 350, 400, 450, 500],
    'regressor__criterion':['mse', 'friedman_mse', 'mae'],
}

regressor = GradientBoostingRegressor()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.913):
{'regressor__criterion': 'mse', 'regressor__loss': 'huber',
'regressor__n_estimators': 500}
```

```
Testing Score  0.9470659928264
```

[22]:
```
from sklearn.svm import SVR
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'regressor__kernel':['linear', 'poly', 'rbf', 'sigmoid', 'precomputed'],
    'regressor__shrinking':[True, False],
    'regressor__degree':[4,5,6,7],
}


regressor = SVR()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.694):
{'regressor__degree': 4, 'regressor__kernel': 'rbf', 'regressor__shrinking':
True}
Testing Score  0.7292327509625709
```

[23]:
```
# Parameters of pipelines can be set using '__' separated parameter names:
param_grid = {
    'regressor__loss':['linear', 'square', 'exponential'],
    'regressor__n_estimators':[1, 5, 10, 20, 50, 100, 150, 200],
    'regressor__learning_rate':[.1,.5,1,2],
}


regressor = AdaBoostRegressor()
scoreModel(param_grid, regressor, x_train, x_test, y_train, y_test)
```

```
Best parameters (CV score=0.771):
{'regressor__learning_rate': 1, 'regressor__loss': 'square',
'regressor__n_estimators': 200}
Testing Score  0.7808594972895909
```

[24]:
```
#x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=.
↪3)
```

[25]:
```
#from kerastuner.tuners import RandomSearch
#from keras.models import Sequential
#from keras.layers import Dense
#from keras.optimizers import Adam
#from keras.wrappers.scikit_learn import KerasRegressor

#def get_params(layers=0, neurons0=0,␣
↪neurons1=0,neurons2=0,neurons3=0,neurons4=0,neurons5=0,neurons6=0,neurons7=0,\
#               neurons8=0, learning_rate0=1e-4, learning_rate1=1e-4):
#    model = Sequential()
```

```
#    model.add(Dense(neurons0, activation='relu', input_dim=7))

#    if layers > 1:
#        model.add(Dense(neurons1,activation = 'relu' ))
#    if layers > 2:
#        model.add(Dense(neurons2,activation = 'relu' ))
#    if layers > 3:
#        model.add(Dense(neurons3,activation = 'relu' ))
#    if layers > 4:
#        model.add(Dense(neurons4,activation = 'relu' ))
#'''    if layers > 5:
#        model.add(Dense(neurons5,activation = 'relu' ))
#    if layers > 6:
#        model.add(Dense(neurons6,activation = 'relu' ))
#    if layers > 7:
#        model.add(Dense(neurons7,activation = 'relu' ))
#    if layers > 8:
#        model.add(Dense(neurons8,activation = 'relu' ))
#'''

#    model.add(Dense(1, activation='linear'))


#    model.compile(
#            optimizer = Adam(learning_rate0),
#            loss = 'mse',
#            metrics = ['accuracy','mse','mae']
#    )
#    model.compile(
#            optimizer = Adam(learning_rate1),
#            loss = 'mse',
#            metrics = ['accuracy','mse','mae']
#    )
#    return model
```

[26]:
```
# model class to use in the scikit random search CV
#model = KerasRegressor(build_fn=get_params, epochs=500, batch_size=1000,␣
 ↪verbose=1)


#Parameters and network structure to search over
# numbers of layers
layers = [1,2,3,4,]

# neurons in each layer
neurons0 = [2,4,6,8,16,32,64,128,256,512,1024]
```

```
neurons1 = [2,4,6,8,16,32,64,128,256,512,1024]
neurons2 = [2,4,6,8,16,32,64,128,256,512,1024]
neurons3 = [2,4,6,8,16,32,64,128,256,512,1024]
neurons4 = [2,4,6,8,16,32,64,128,256,512,1024]
#neurons5 = [2,4,6,8,16,32,64,128,256,512,1024]
#neurons6 = [2,4,6,8,16,32,64,128,256,512,1024]
#neurons7 = [2,4,6,8,16,32,64,128,256,512,1024]
#neurons8 = [2,4,6,8,16,32,64,128,256,512,1024]


learning_rate0 = [1e-3, 1e-4]
learning_rate1 = [1e-5, 1e-6]

# dictionary summary
param_grid = dict(
                layers=layers,
                neurons0=neurons0,
                neurons1=neurons1,
                neurons2=neurons2,
                neurons3=neurons3,
                neurons4=neurons4,
                #neurons5=neurons5,
                #neurons6=neurons6,
                #neurons7=neurons7,
                #neurons8=neurons8,
                learning_rate0=learning_rate0,
                learning_rate1=learning_rate1,
)
#grid = GridSearchCV(model, cv=3, param_grid=param_grid,
#                           verbose=20, scoring='r2', return_train_score=True,␣
 ↪n_jobs=8)
```

[27]:
```
#grid.fit(x_train, y_train, validation_data=(x_val, y_val))
```

[28]:
```
#print("Best parameters (CV score=%0.3f):" % grid.best_score_)
#print(grid.best_params_)

#y_pred = grid.predict(x_test)
#print("Testing Score ", r2_score(y_test, y_pred))
#x_val_score.append(round(grid.best_score_,3))
#testing_score.append(round(r2_score(y_test, y_pred),3))
```

4.1.3 Make a DataFrame to compare models after hyperparameter tuning and their metrics as above. (8 Marks)

[29]:
```
df_results = pd.DataFrame(columns = ['Testing Score', 'CV Score'], index =␣
↪models)
df_results['CV Score'] = x_val_score
df_results['Testing Score'] = testing_score
df_results.sort_values(ascending=False, by = "Testing Score", inplace=True)
df_results.head(10)
```

[29]:

	Testing Score	CV Score
GradentBoostingRegressor	0.947000	0.912000
RandomForestRegressor	0.917000	0.892000
BaggingRegressor	0.915000	0.889000
Ridge Regression	0.872887	0.800461
DecissionTreeRegressor	0.856000	0.807000
AdaboostRegressor	0.781000	0.771000
SVR	0.729000	0.694000

4.1.4 Write the significance of the R2 score from the business perspective. Explain the effect of an increase/decrease in the R2 score on the business model. (4 Marks)

The R2 score in this model represents the models ability to predict concrete strength given solely the raw materials included. The higher the R2 score the better the business will be able to reliably manufacuter concrete for each of it's specific business needs By this point, the definition of R2 score should be understood, but it's included here for excessive completeness: R-squared (R2) 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. R2 =1− Unexplained Variation/Total Variation.

taken from: https://www.investopedia.com/terms/r/r-squared.asp

5 Model Performance and Deployment to Production

[30]:
```
output = pd.DataFrame()
model = GradientBoostingRegressor(criterion= 'friedman_mse', loss = 'huber',␣
↪n_estimators = 300)
model = make_pipeline(StandardScaler(), model)
model.fit(x_train, y_train)

feature_weights = model.steps[1][1].feature_importances_
output['feature_rating'] =  feature_weights[0:]
output['features'] = x_train.columns
print(output.sort_values(by=['feature_rating'], ascending = False,␣
↪ignore_index=True))
```

```
   feature_rating          features
```

```
0        0.370464                age
1        0.318445             cement
2        0.130202              water
3        0.070601               slag
4        0.053930           fineagg
5        0.041297  superplastic+ash
6        0.015060         coarseagg
```

5.0.1 Analysis:

Age and Quantity of cement used have high impact on the strength of the concrete. Superplastic, Slag, Fine Aggregate, Coarse Aggregate, and Ash play minor roles in the strengh of the concrete. (See feature ratings above)

From a business perspective, Concrete mix design is the process of developing the correct proportions of cement, water, range of aggregate from sand to larger aggregate usually up to 1.5 in. to achieve the desired strength for cost, performance and durability. taken from: http://blog.bartellglobal.com/what-makes-concrete-hard-medium-or-soft-and-how-can-i-tell

also referenced: https://cor-tuf.com/everything-you-need-to-know-about-concrete-strength/

Since there is no one size fits all solution to concrete strength, the business can use the model to compute the desired strength for their specific application. methodology explicitly demonstrated below:

[31]:
```
# setup datafram with the columns and values to be assessed as shown below
ex_a = {'cement' : [168.9],'slag' : [42.2], 'water' : [158.3], 'coarseagg' :␣
 ↪[1080.8],'fineagg' : [796.2],'age': [14],'superplastic+ash' : [135.1]}
ex = pd.DataFrame(ex_a, )
#use the previously trained model to predict the strength of the concrete
output = model.predict(ex)
print("The expected concrete strength is {}".format(round(output[0],3)))
```

```
The expected concrete strength is 24.456
```

5.0.2 a web tool has been created and deployed that prompts the user to input values for cement, slag, water, superplastic, coarseagg, and fineagg. The tool then outputs the expected strength

6 model deployed to production: http://bigdogai.com/concrete/

6.1 CI/CD plan.

6.1.1 The data for this model are fixed so no CI/CD will be performed.

if there were new data, the new data would be added to the dataset, the model would be retrained, and uploaded to an AWS S3 bucket, via background processes. The updated model would then be instantly available to users.

6.1.2 The marks for the score will be according to the following: testing score 90 -95 (5 marks), 85 - 90 (4 marks), 80 - 85(3 marks), <80 (2 marks)

Three models scoring > 90% on testing data with no data leakage!!! AMAZING GENERALIZATION!!!