Data Science Techniques and Applications - Principal components analysis

March 14, 2021

Orlando Taddeo, MSc Data Science, Student ID 13180720

1 Academic Declaration

"I have read and understood the sections of plagiarism in the College Policy on assessment offences and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my report to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software".

2 Introduction

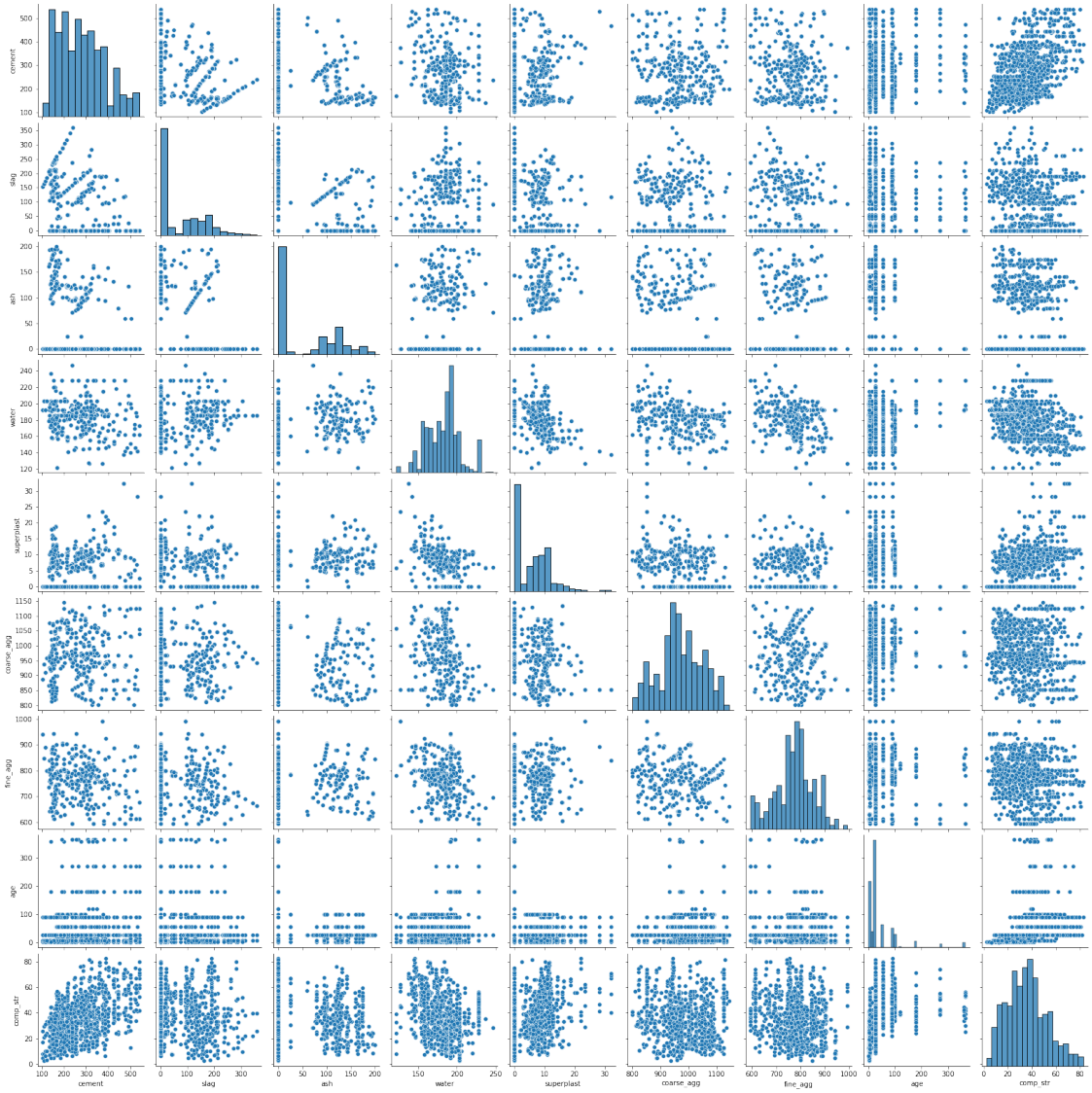
In Coursework 1, we chose the dataset '**Concrete Compressive Strength Data**' [1] for analysis. Whilst we refer to the previous coursework for more detailed information about the dataset, we report below a pairplot that summarises it, for our convenience.

[1]:
```
# Importing libraries

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import os
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Importing data

source_path = '/Users/orlandotaddeo/Desktop/concrete data/Concrete_Data.csv'
source_df = pd.read_csv(source_path)

# DataFrame columns renaming (for simplicity)

new_names = {'Cement (component 1)(kg in a m^3 mixture)': 'cement',
             'Blast Furnace Slag (component 2)(kg in a m^3 mixture)': 'slag',
             'Fly Ash (component 3)(kg in a m^3 mixture)': 'ash',
```

```
            'Water  (component 4)(kg in a m^3 mixture)': 'water',
            'Superplasticizer (component 5)(kg in a m^3 mixture)':␣
↪'superplast',
            'Coarse Aggregate  (component 6)(kg in a m^3 mixture)':␣
↪'coarse_agg',
            'Fine Aggregate (component 7)(kg in a m^3 mixture)': 'fine_agg',
            'Age (day)': 'age',
            'Concrete compressive strength(MPa)': 'comp_str'}

source_df.rename(columns=new_names, inplace=True)
```

[2]:
```
sns.pairplot(source_df)
```

[2]: <seaborn.axisgrid.PairGrid at 0x7fae67069dc0>



3 Further dataset analysis

As we pointed out in Coursework 1, the dataset comprises quantities of all the different components of concrete mixtures. The components are:

1. water
2. cement
3. coarse aggregate
4. fine aggregate
5. fly ash
6. furnace slag
7. superplasticizer

Whilst the first four items in the list are **always** present in the mixture, the others may be not. For example, just one of them can be included, or any combination of them. There is a great abundance of concrete formulas on the market nowadays, and a rigorous classification is a really challenging task. We will attempt, hereby, a rough classification of the different kinds of concrete that is detailed enough for our data analysis purposes. The reasons for introducing this classification and its effects on the considered dataset will be clear shortly. We may group the types of concrete present in the dataset in the following categories:

1. **Classical concrete** This is the simplest concrete found in the built environment. It contains only the first four components reported in the list above: water, cement, coarse aggregate, fine aggregate.

2. **Fly ash concrete** This is a classical concrete with the addition of fly ash, for the reasons explained in the previous coursework. It contains water, cement, coarse aggregate, fine aggregate, fly ash, and no other components.

3. **Furnace slag concrete** This is a classical concrete with the addition of furnace slag, for the reasons explained in the previous coursework. It contains water, cement, coarse aggregate, fine aggregate, furnace slag, and no other components.

4. **Classical with superplaticizer concrete** This group simply identifies classical concrete to which superplasticizer has been added, for the reasons explained in the previous coursework. It contains water, cement, coarse aggregate, fine aggregate, superlasticizer, and no other components.

5. **Slag and ash concrete** This group includes kinds of concrete made by adding both fly ash and furnace slag to the mixture. It contains water, cement, coarse aggregate, fine aggregate, furnace slag, fly ash, and no other components.

6. **Slag and superpasticizer concrete** This kind of concrete is based on the furnace slag one, but modified by adding superplasticizer. It contains water, cement, coarse aggregate, fine aggregate, furnace slag, superplasticizer, and no other components.

7. **Ash and superplasticizer concrete**. This kind of concrete is based on the fly ash one, but modified by adding superplasticizer. It contains water, cement, coarse aggregate, fine aggregate, fly ash, superplasticizer, and no other components.

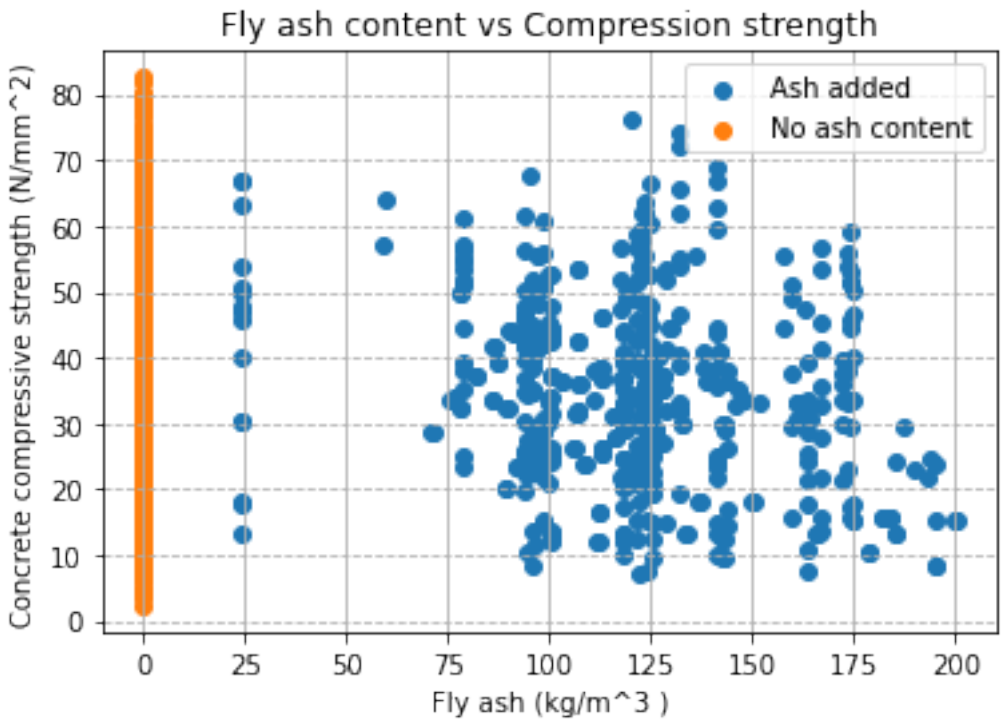
8. **Complete mix concrete** This kind of concrete contains all the components reported in the list above. The proportions may vary greatly, so we gave a very general definition of it, that identifies more its role in the dataset rather than its commercial name. It is worth underlying that, for this kind, *none of the components mentioned above is absent.*

The reasons why we needed to separate kinds of concrete with different compositions are mainly two.

1. Different compositions might affect the mechanical behaviour of the concrete, and therefore its strength. While performing any statistical analysis to predict the strength, we deem important to keep separated factors that can have influence on the results. It makes little sense to mix data coming from different kinds of material; in fact, concrete with presence of ash, slag and/or superplasticizer are building materials with dissimilar binders, and therefore should be treated independently.

2. In predicting the strength, we operate mainly in a regression context. Without loss of generality, let us suppose we are interested in performing a linear regression after carrying out a PCA. If we keep together all the data, and if we plot one predictor against the target, we might get a situation similar to that shown in the following plot.

[3]:
```
nonzero = source_df['ash']!=0
x_no_0 = source_df['ash'][nonzero]
y_no_0 = source_df['comp_str'][nonzero]
zero = source_df['ash']==0
x_0 = source_df['ash'][zero]
y_0 = source_df['comp_str'][zero]
fig, ax = plt.subplots()
plt.title('Fly ash content vs Compression strength')
ax.set_xlabel('Fly ash (kg/m^3 )')
ax.set_ylabel('Concrete compressive strength (N/mm^2)')
plt.grid(b=True, which='major', axis='x', linestyle='-')
plt.grid(b=True, which='major', axis='y', linestyle='--')
ax.scatter(x_no_0, y_no_0)
ax.scatter(x_0, y_0)
plt.legend(['Ash added', 'No ash content'])
plt.figure(figsize=(7, 5))
plt.show()
```




```
<Figure size 504x360 with 0 Axes>
```

As it can be seen from the plot, there are many datapoints that represent concrete without fly ash. If our aim is to model the relationship between the amount of fly ash and concrete strength, the presence of a mass of zero-valued datapoints will significantly affect the final result, without providing much information about such a relationship. Even though they do affect our model, these points are related to a physically different case (i.e., another kind of concrete). The same thing happens for the presence of furnace slag and superplaticizer, as it can be seen from the pairplot above. Therefore, the approach we decided to use is to *split the dataframe records into sub-groups* to isolate the eight types of concrete we identified previously. In addition to this, we consider also the **entire dataframe**, to provide a comparison. The code used to split the dataframe is shown below, together with the main information about the sub-parts obtained. The final dictionary provides the id's given to each dataframe, that will be useful to identify the respective plots.

[4]:
```
classical_df = source_df[
    (source_df['slag']==0) &
    (source_df['ash']==0) &
    (source_df['superplast']==0)].drop(['slag', 'ash', 'superplast'], axis=1)

slag_only_df = source_df[
    (source_df['slag']!=0) &
    (source_df['ash']==0) &
    (source_df['superplast']==0)].drop(['ash', 'superplast'], axis=1)
```

```
ash_only_df = source_df[
    (source_df['slag']==0) &
    (source_df['ash']!=0) &
    (source_df['superplast']==0)].drop(['slag', 'superplast'], axis=1)

slag_ash_df = source_df[
    (source_df['slag']!=0) &
    (source_df['ash']!=0) &
    (source_df['superplast']==0)].drop(['superplast'], axis=1)

plast_only_df =  source_df[
    (source_df['slag']==0) &
    (source_df['ash']==0) &
    (source_df['superplast']!=0)].drop(['slag', 'ash'], axis=1)

slag_plast_df =  source_df[
    (source_df['slag']!=0) &
    (source_df['ash']==0) &
    (source_df['superplast']!=0)].drop(['ash'], axis=1)

ash_plast_df =  source_df[
    (source_df['slag']==0) &
    (source_df['ash']!=0) &
    (source_df['superplast']!=0)].drop(['slag'], axis=1)

complete_df = source_df[
    (source_df['slag']!=0) &
    (source_df['ash']!=0) &
    (source_df['superplast']!=0)]

all_dataframes = {'entire_dataframe': source_df,
                  'complete': complete_df,
                  'classical': classical_df,
                  'slag_only': slag_only_df,
                  'ash_only': ash_only_df,
                  'plast_only': plast_only_df,
                  'slag_ash': slag_ash_df,
                  'slag_plast': slag_plast_df,
                  'ash_plast': ash_plast_df}
```

Now, we print out the number of records of the dataframes obtained by this procedure.

[5]:
```
for key, value in all_dataframes.items():
    print('Observations in ' + key + ' = ' + str(all_dataframes[key].shape[0]))
```

```
Observations in entire_dataframe = 1030
Observations in complete = 225
Observations in classical = 209
```

```
Observations in slag_only = 164
Observations in ash_only = 6
Observations in plast_only = 23
Observations in slag_ash = 0
Observations in slag_plast = 170
Observations in ash_plast = 233
```

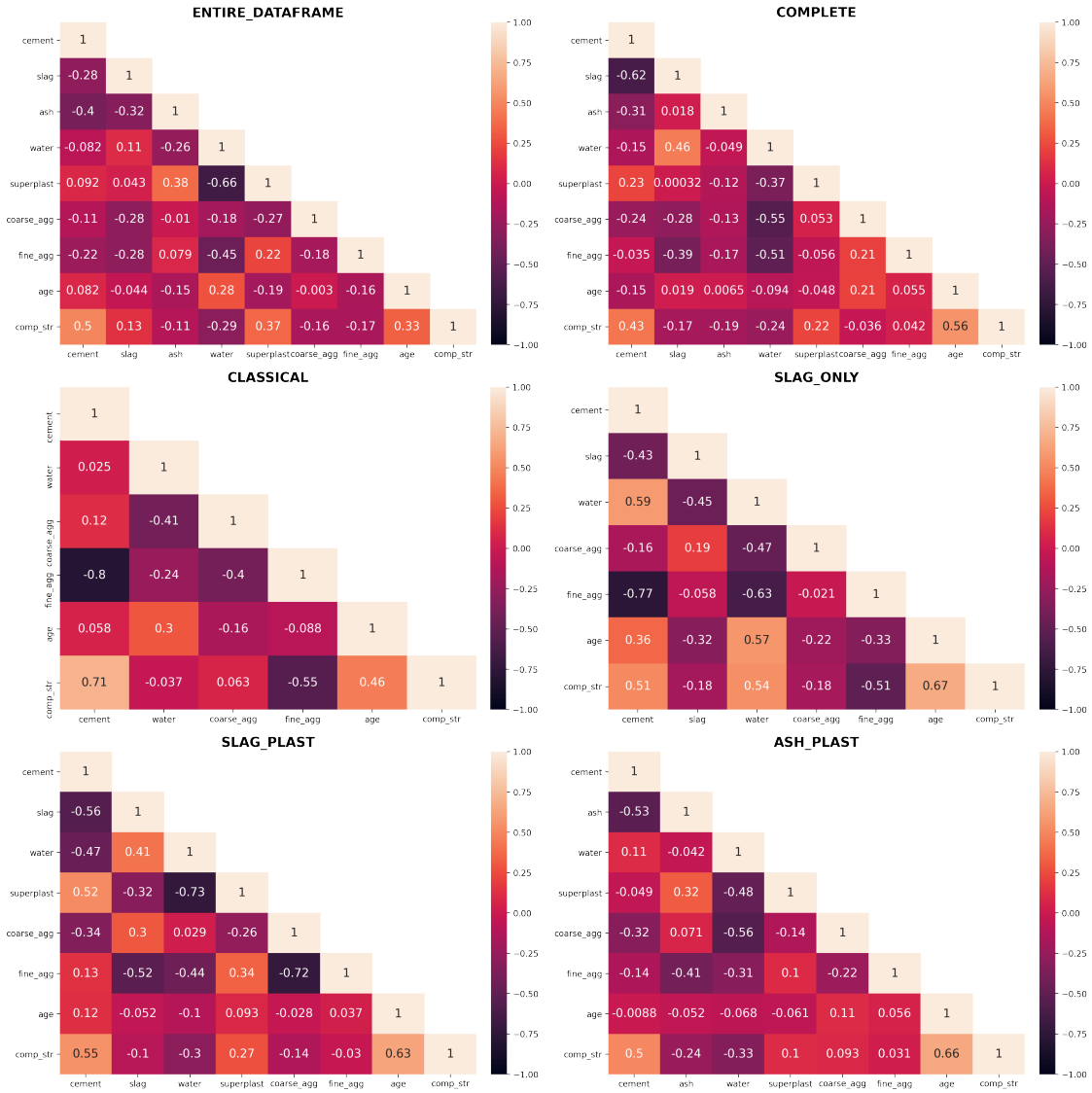
As we can see, three kinds of concrete we identified are *significantly underrepresented* in the dataset: **Fly ash concrete**, **Classical with superplaticizer concrete**, **Slag and ash concrete**. Since we do not have enough information about them to obtain meaningful results, we decide to remove the instances representing these materials. However, since the database is updated regularly, more data may be available in the future. Therefore, we keep for analysis only the types of concrete contained in the following dictionary.

[6]:
```
dataframes = {'entire_dataframe': source_df,
              'complete': complete_df,
              'classical': classical_df,
              'slag_only': slag_only_df,
              'slag_plast': slag_plast_df,
              'ash_plast': ash_plast_df}
```

4 Choice of the most representative features

Starting from the observations on the data reported in Coursework 1, we now want to precisely identify **three features** that we believe have the greatest predictive power among all those available. We start by comparing the correlation coefficients between the predictors and the target variable related to the different dataframes just mentioned.

[7]:
```
fig, axes = plt.subplots(3, 2, figsize = (18, 18), dpi=200,␣
 ↪constrained_layout=True)

i = 0
j = 0
for key, value in dataframes.items():
    corr_df = value.corr()
    mask = np.zeros_like(corr_df)
    mask[np.triu_indices_from(mask, k=1)] = True
    corr_plot = sns.heatmap(corr_df, annot=True, vmin=-1, vmax=1, mask=mask, ␣
 ↪annot_kws={"fontsize":14}, ax=axes[i, j])
    axes[i, j].set_title(key.upper(), fontsize=16, fontweight='bold')
    if j == 1:
        i += 1
        j = 0
    else:
        j += 1
```



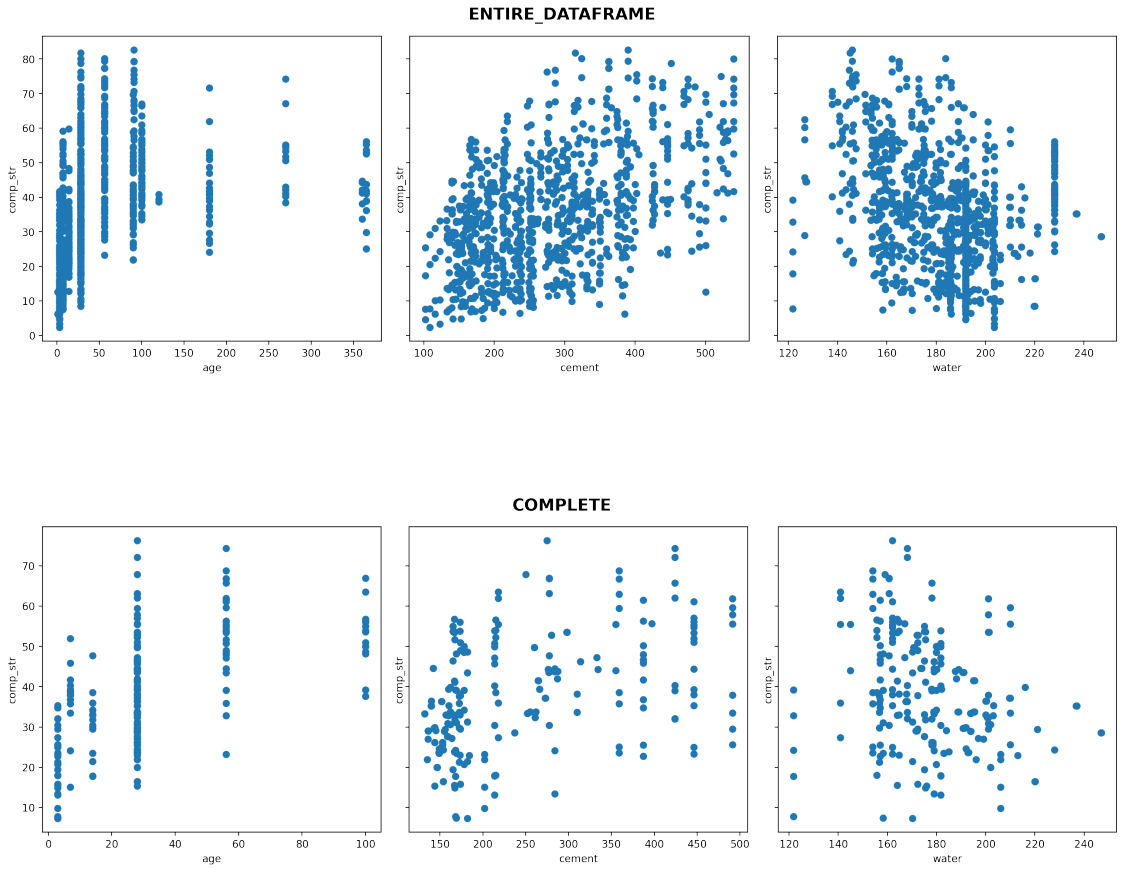
The following observations can be made:

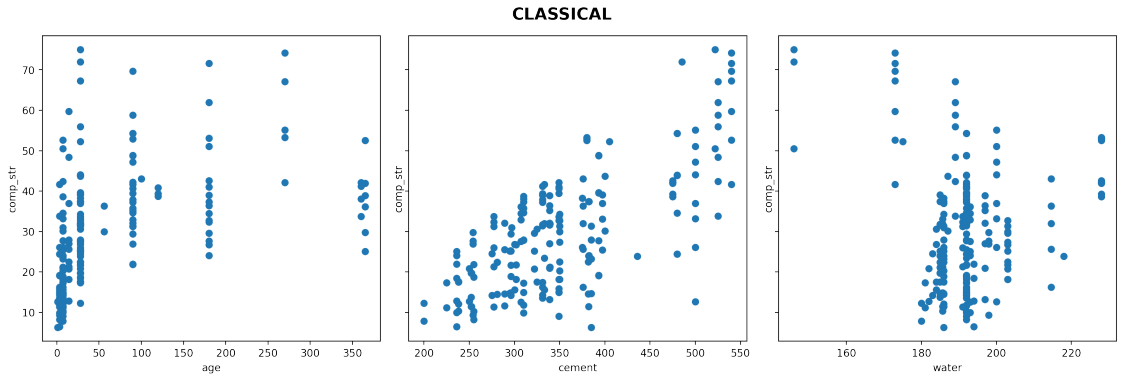
1. In all the cases, the strongest (linear) correlations to compression strength are given by cement, water, and age.
2. In some cases, applying the dataframe splitting, we get more predictors with a stronger correlation to compression strength, compared to the entire dataset.
3. Observing the correlation coefficients and the pairplots (available by running the attached script), we can conjecture that the three features that can be identified as the most important in predicting compression strength are:
    1. **Age**
    2. **Cement content**
    3. **Water content**

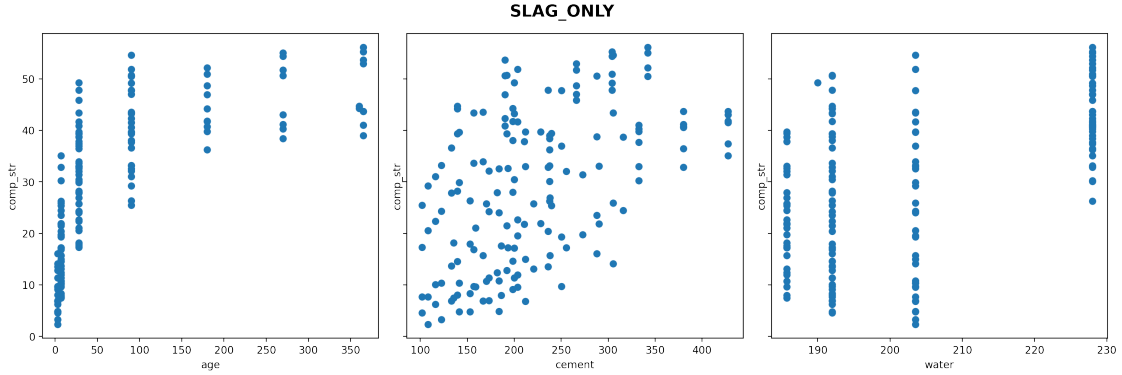
We reported the three features according to what we believe is a possible decreasing order of accuracy in predicting the compression strength values. Our conjecture is based on the values of the correlation coefficients seen above.

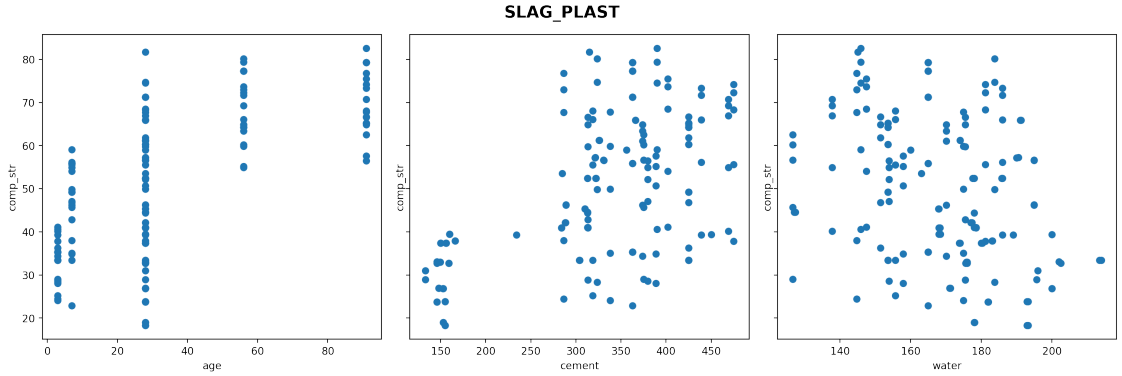
In the following, we plot the target dimension against each of the three selected predictors, for each of the six dataframes considered.

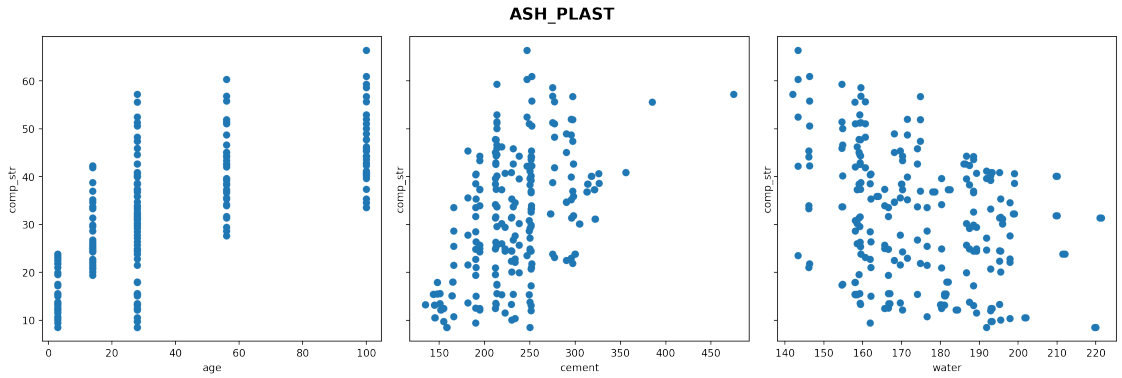
[8]:
```
for key, value in dataframes.items():
    j = 0
    fig, axes = plt.subplots(1, 3, sharey=True, figsize = (15,5), dpi=200,␣
 ↪constrained_layout=True)
    plt.suptitle(key.upper(), fontsize=16, fontweight='bold')
    for predictor in ['age', 'cement', 'water']:
        x = value[predictor]
        y = value['comp_str']
        axes[j].set_xlabel(predictor)
        axes[j].set_ylabel('comp_str')
        axes[j].scatter(x, y)
        j += 1
    plt.show()
```




CLASSICAL
comp_str
age
cement
water


SLAG_ONLY
comp_str
age
cement
water


SLAG_PLAST
comp_str
age
cement
water



5 Principal component analysis

If a dataset has d features and n observations, each observation can be interpreted geometrically as a feature vector in a *d-dimensional* space. In order to reduce the dimensionality of the dataset, one possible technique is the **Principal Component Analysis (PCA)**. Briefly, this algorithm finds the directions, in the feature space, along which the data presents the largest variance. These directions are identified by a set of orthonormal vectors, that happen to be the eigenvectors of the covariance matrix of the dataset. There will be d eigenvectors and d related eigenvalues: each eigenvector is associated with one eigenvalue. It is possible to order the eigenvalues from the biggest to the smallest; in fact, the greater the eigenvalue associated to a certain direction (represented by the respective eigenvector), the greater the variance along that direction. The aim of the PCA is to choose the first r eigenvectors, where $r < d$, to try to have a less heavy dataset but preserving as much variance (and therefore, information) as possible. It is possible to calculate the fraction of the total variance captured along each principal component; all these fractions will sum up to the total variance in the data. We can, for example, decide to consider the first r principal components, such that the fraction of the total variance that they capture is above a certain threshold. Usually, a 95/100 threshold is used [2]. Finally, the feature vectors are projected on the r dimensions chosen, and the components of such vectors along these directions are usually called *scores* of the principal components.

We use the package `scikit-learn` [3] to perform the PCA on the datasets just considered. First, we will use all the features they contain, and then just the three features we identified as the most useful ones. The following plot contains the results related to a PCA carried out using all the features. This is done via a specifically designed function called `pca_calculator`, included in the code block below. In the printout, it is possible to see the following information:

1. `loadings` These are the components of each eigenvector along each of the d axes of the original space. The greater the component along a certain direction, the greater the 'weight' of the respective feature on the considered principal component.

2. `explained_variance` This is the fraction of the total variance captured along each principal component, from the greatest to the smallest.

3. `cumulative_explained_variance` The i-th item of this list is the fraction of the total variance captured by the first i principal components (i.e., summing up the first i explained variances

of the list `explained_variance`).

The scores are not printed in the output, even though they are available when calling the `fit_transform` method on the `PCA` object created. This is done to keep the report short. In the script submitted along with this report, such data can be easily obtained.

It is important to note that, before passing the dataframe to the `PCA` class provided by scikit-learn, the data is subjected to a procedure called **Feature scaling**. It consist in centering each of the features in the dataset such that they have mean zero, and dividing each of their components by the feature's standard deviation. We obtain, therefore, a dataset where each feature has *mean zero* and *unit standard deviation*. In fact, different features can have significantly different numerical values, that can differ from each other even by some order of magnitudes. This will produce much bigger variances for some features compared to others only because of their numerical values, and can make the PCA algorithm identify directions of greater variance incorrectly. The scikit-learn package provides an easy-to-use tool to scale (or *standardize*) the data, via the class `StandardScaler`, used in the function `pca_calculator`.

[9]:
```
def pca_calculator(df_dict, target):
    proj_df_dict = {}
    for key, value in df_dict.items():
        scaler = StandardScaler()
        df_scaled = pd.DataFrame(scaler.fit_transform(value), columns=value.
 ↪columns)
        r = df_scaled.shape[1] - 1
        pc_names = ['PC' + str(i) for i in range(1, r+1)]
        pca = PCA(n_components=r)
        df_proj = pd.DataFrame(pca.fit_transform(df_scaled.drop([target],␣
 ↪axis=1)),columns=pc_names)
        df_proj.insert(len(df_proj.columns), 'scaled_target', df_scaled[target])
        feat_names = list(df_scaled.columns)[:-1]
        loadings = pd.DataFrame(pca.components_.T, index=feat_names,␣
 ↪columns=pc_names)
        cum_expl_var = [round(sum(pca.explained_variance_ratio_[:i+1]), 3)
                        for i in range(len(pca.explained_variance_ratio_))]
        new_key = key + '_proj'
        proj_df_dict[new_key] = {}
        proj_df_dict[new_key]['proj_data_and_target'] = df_proj
        proj_df_dict[new_key]['loadings'] = loadings
        proj_df_dict[new_key]['explained_variance'] = pca.
 ↪explained_variance_ratio_
        proj_df_dict[new_key]['cumulative_explained_variance'] = cum_expl_var
    return proj_df_dict


def print_pca_results(pca_dict):
    for pca_id in pca_dict.keys():
        print('*****' + pca_id.upper() + '*****')
        print()
```

```
        for key, value in pca_dict[pca_id].items():
            if not(key == 'proj_data_and_target'):
                print('**' + key + '**')
                print(value)
                print()

all_pcas = pca_calculator(dataframes, 'comp_str')
print_pca_results(all_pcas)
```

```
*****ENTIRE_DATAFRAME_PROJ*****

**loadings**
                PC1       PC2       PC3       PC4       PC5       PC6  \
cement      0.098401 -0.113737  0.814202 -0.054297  0.148206 -0.203142
slag        0.177262  0.686053 -0.171794 -0.362699 -0.020932  0.304882
ash        -0.394662 -0.142948 -0.408221  0.226751  0.549631 -0.183267
water       0.547004  0.053256 -0.213190  0.296060  0.070222 -0.365970
superplast -0.505945  0.282930  0.234597 -0.037274  0.354618  0.193294
coarse_agg  0.037928 -0.629943 -0.174088 -0.545805 -0.033083  0.314559
fine_agg   -0.401926 -0.019391 -0.004569  0.385282 -0.701237  0.092466
age         0.291479 -0.125981  0.100521  0.527919  0.228010  0.743908

                PC7       PC8
cement      0.221844  0.446163
slag        0.228363  0.437384
ash         0.352463  0.381886
water      -0.524275  0.388741
superplast -0.664643  0.051750
coarse_agg -0.226840  0.349320
fine_agg   -0.039026  0.433370
age         0.069367  0.012881

**explained_variance**
[0.28501242 0.17700935 0.16750988 0.12676986 0.1189385  0.09877443
 0.02223021 0.00375535]

**cumulative_explained_variance**
[0.285, 0.462, 0.63, 0.756, 0.875, 0.974, 0.996, 1.0]

*****COMPLETE_PROJ*****

**loadings**
                PC1       PC2       PC3       PC4       PC5       PC6  \
cement     -0.270504 -0.659293 -0.069131  0.031915  0.314217 -0.030883
slag        0.500007  0.230252  0.394314  0.128926 -0.233935  0.188185
ash         0.151136  0.245498 -0.222916 -0.843608  0.226875  0.075872
water       0.550299 -0.185966 -0.167369  0.275601  0.070301 -0.077201
```

```
superplast -0.202258 -0.187700  0.795432 -0.232956 -0.010046  0.237864
coarse_agg -0.369965  0.446494  0.149670  0.121957 -0.027687 -0.655042
fine_agg   -0.410864  0.182293 -0.318960  0.146541 -0.413288  0.590164
age        -0.067508  0.386839  0.071996  0.322666  0.786418  0.343458

                PC7       PC8
cement     -0.273440  0.558486
slag       -0.460270  0.472105
ash         0.084975  0.301784
water       0.677289  0.298323
superplast  0.424088  0.018672
coarse_agg  0.155906  0.415139
fine_agg    0.207914  0.335294
age         0.007449 -0.005378

**explained_variance**
[0.29635143 0.20207306 0.14014098 0.12957557 0.11138046 0.08738313
 0.02835487 0.0047405 ]

**cumulative_explained_variance**
[0.296, 0.498, 0.639, 0.768, 0.88, 0.967, 0.995, 1.0]

*****CLASSICAL_PROJ*****

**loadings**
                PC1       PC2       PC3       PC4       PC5
cement      0.631789  0.073614 -0.219593  0.513519 -0.532448
water       0.068630  0.665840 -0.248231 -0.621173 -0.323224
coarse_agg  0.329178 -0.541019  0.433102 -0.524692 -0.368865
fine_agg   -0.695149 -0.074789  0.062050  0.177205 -0.689872
age         0.067399  0.502935  0.835902  0.209140  0.006468

**explained_variance**
[0.39047837 0.31780518 0.16115097 0.11974396 0.01082152]

**cumulative_explained_variance**
[0.39, 0.708, 0.869, 0.989, 1.0]

*****SLAG_ONLY_PROJ*****

**loadings**
                PC1       PC2       PC3       PC4       PC5       PC6
cement      0.488482 -0.271341  0.189778 -0.362625 -0.554361 -0.461444
slag       -0.304652 -0.470474 -0.695251  0.218870 -0.103681 -0.379228
water       0.528421  0.097039 -0.169983  0.005638  0.709064 -0.423860
coarse_agg -0.237776 -0.565017  0.648802  0.281661  0.281209 -0.211808
fine_agg   -0.420758  0.597464  0.172663  0.073108 -0.113817 -0.646442
age         0.394676  0.139297  0.030986  0.857840 -0.295178  0.029118
```

explained_variance
[0.4897037 0.20645768 0.14356168 0.11165993 0.04672407 0.00189294]

cumulative_explained_variance
[0.49, 0.696, 0.84, 0.951, 0.998, 1.0]

*****SLAG_PLAST_PROJ*****

loadings

	PC1	PC2	PC3	PC4	PC5	PC6	\
cement	0.406745	0.255191	-0.003551	0.612150	-0.350766	0.375544	
slag	-0.418297	0.060840	0.049280	-0.479828	-0.614961	0.394474	
water	-0.426036	-0.400273	0.160101	0.393635	-0.208947	-0.506878	
superplast	0.432927	0.327080	-0.124087	-0.323867	-0.386465	-0.647880	
coarse_agg	-0.334426	0.615025	-0.141181	-0.038418	0.472042	-0.047473	
fine_agg	0.413840	-0.490941	0.030747	-0.358070	0.278041	0.153443	
age	0.086891	0.211419	0.967291	-0.074232	0.075108	-0.029741	

	PC7
cement	0.361423
slag	0.234257
water	0.420844
superplast	0.127288
coarse_agg	0.513243
fine_agg	0.598083
age	-0.008035

explained_variance
[0.43771693 0.18585499 0.13903031 0.11337901 0.09004982 0.03197161
 0.00199733]

cumulative_explained_variance
[0.438, 0.624, 0.763, 0.876, 0.966, 0.998, 1.0]

*****ASH_PLAST_PROJ*****

loadings

	PC1	PC2	PC3	PC4	PC5	PC6	\
cement	-0.466288	-0.204819	0.040868	0.658728	0.188540	0.383711	
ash	0.435012	0.523004	0.260933	-0.092537	0.172944	0.587587	
water	-0.513985	0.476983	0.044991	-0.286754	0.052221	-0.384614	
superplast	0.358578	-0.186082	0.616803	0.268916	0.271690	-0.515842	
coarse_agg	0.445583	-0.026215	-0.615418	0.240928	-0.220822	-0.192622	
fine_agg	-0.011250	-0.631809	0.169525	-0.542289	-0.159604	0.236529	
age	0.043713	-0.150020	-0.374561	-0.225492	0.885305	-0.020914	

	PC7

```
cement      -0.349643
ash         -0.292255
water       -0.522876
superplast -0.210062
coarse_agg -0.527369
fine_agg    -0.443234
age          0.015414

**explained_variance**
[0.27931973 0.22334226 0.18533879 0.14445887 0.13260952 0.03118159
 0.00374923]

**cumulative_explained_variance**
[0.279, 0.503, 0.688, 0.832, 0.965, 0.996, 1.0]
```

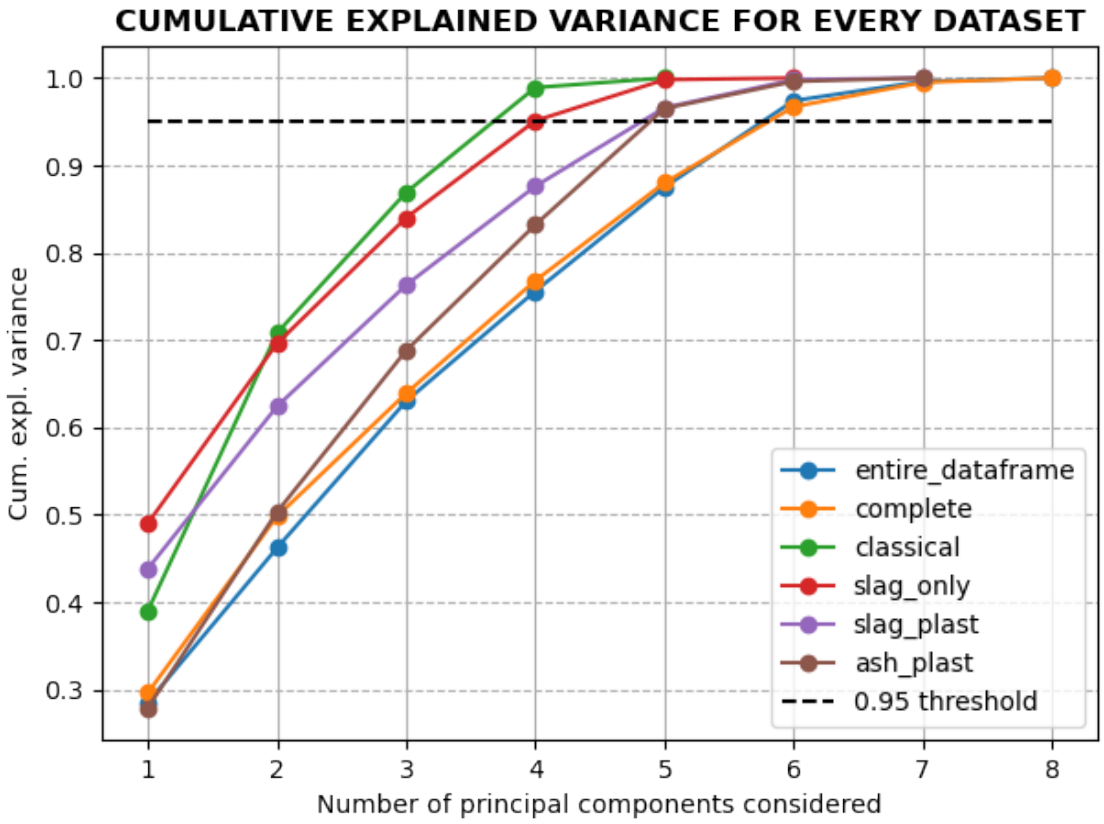
From the results printed above, we can observe that:

1. In almost all cases, there is no feature that has a dominant weight (or loading) on the others for any of the principal components. Since these loadings represent the coordinates of the eigenvectors in the original reference system, this means that the new principal components tend not to be almost parallel to any of the original features.
2. The three features that we previously selected have particularly high loadings, if compared to the others, often exceeding 0.8. This means that they can be good predictors of the target variable, as we expected.
3. The fraction of total variance explained by the principal components does not increase quickly as the number of components increase. This means that, unfortunately, it will be difficult to reduce significantly the number of dimensions while preserving a high portion of the total variance.

What has been stated in the last point can be shown better by an image. In the following plot, we show the progressive increase of the explained variance as the number of principal components considered increases.

[10]:
```
def cum_expl_var_plotter(pca_dict):
    fig, ax = plt.subplots(figsize = (7,5), dpi=100)
    plt.title('CUMULATIVE EXPLAINED VARIANCE FOR EVERY DATASET', fontsize=12,␣
 ↪fontweight='bold')
    ax.set_xlabel('Number of principal components considered')
    ax.set_ylabel('Cum. expl. variance')
    plt.grid(b=True, which='major', axis='x', linestyle='-')
    plt.grid(b=True, which='major', axis='y', linestyle='--')
    legend_items = []
    n_comp_max = 0
    for pca_id in pca_dict.keys():
        legend_items.append(pca_id[:-5])
        n_components = len(pca_dict[pca_id]['cumulative_explained_variance'])
        if n_components > n_comp_max:
```

```
            n_comp_max = n_components
        x = list(range(1, n_components+1))
        y = pca_dict[pca_id]['cumulative_explained_variance']
        ax.plot(x, y, '-o')
    x_thresh = [1, n_comp_max]
    y_thresh = [0.95, 0.95]
    legend_items.append('0.95 threshold')
    ax.plot(x_thresh, y_thresh, linestyle='--', color='black')
    plt.xticks(np.arange(1, n_comp_max+1, 1.0))
    plt.legend(legend_items)
    plt.show()


cum_expl_var_plotter(all_pcas)
```



Examining the plot, we can make some observations:

1. In the two cases in which we consider the original (i.e., entire) and the complete dataframe (i.e., the one comprising only concrete containing all the available 'ingredients'), the explained variance increases more slowly, and the threshold is reached only when six components are included.

2. The variance explained increases faster for the reduced dataframes, with the classical concrete and the slag concrete reaching the threshold with four components. In these two curves, a 'knee' is much more evident, and using the PCA is a bit more effective in reducing the number of features.

3. For all the cases, a dramatic reduction in the number of features is not achievable.

Now, we draw our attention to the analysis of a dataframe comprising only the 3 features we considered to be the most significant: **age**, **cement content**, **water content**. The results of the PCAs carried out are reported in the following output. The functions `pca_calculator`, `print_pca_results` and `cum_expl_var_plotter` defined above are used.

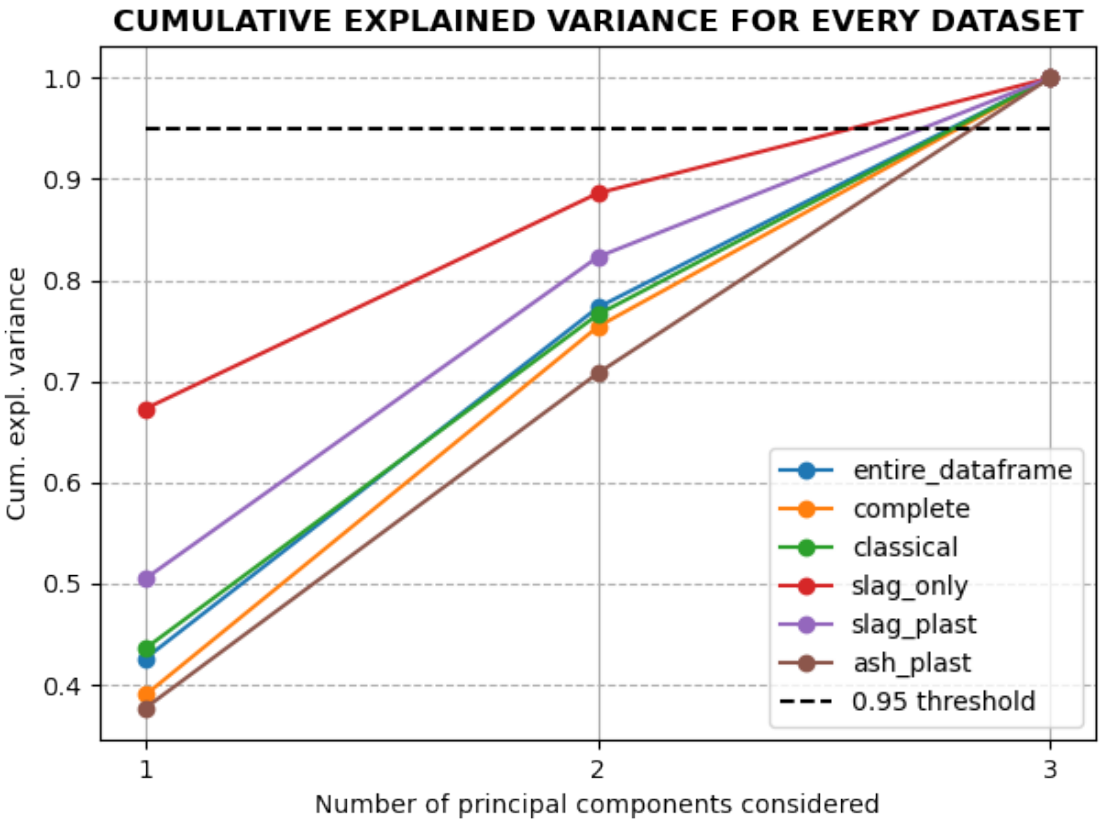
[11]:
```
dataframes_3_feat = {key : dataframes[key].loc[:, ['cement', 'water', 'age',␣
↪'comp_str']]
                     for key, value in dataframes.items()}

all_pcas_3_feat = pca_calculator(dataframes_3_feat, 'comp_str')
print_pca_results(all_pcas_3_feat)
```

```
*****ENTIRE_DATAFRAME_PROJ*****

**loadings**
             PC1       PC2       PC3
cement  0.001002  0.940299 -0.340347
water   0.706959 -0.241378 -0.664790
age     0.707254  0.239945  0.664995

**explained_variance**
[0.42587283 0.34728486 0.22684232]

**cumulative_explained_variance**
[0.426, 0.773, 1.0]

*****COMPLETE_PROJ*****

**loadings**
             PC1       PC2       PC3
cement  0.779694  0.023345  0.625725
water  -0.417585 -0.725245  0.547397
age    -0.466583  0.688095  0.555720

**explained_variance**
[0.38983286 0.36460297 0.24556417]

**cumulative_explained_variance**
[0.39, 0.754, 1.0]

*****CLASSICAL_PROJ*****
```

```
**loadings**
             PC1       PC2       PC3
cement  0.188070  0.978848  0.080532
water   0.689266 -0.189954  0.699164
age     0.699673 -0.075984 -0.710412

**explained_variance**
[0.43576788 0.33022421 0.2340079 ]

**cumulative_explained_variance**
[0.436, 0.766, 1.0]

*****SLAG_ONLY_PROJ*****

**loadings**
             PC1       PC2       PC3
cement  0.555662 -0.687358 -0.467738
water   0.626183 -0.024097  0.779303
age     0.546932  0.725919 -0.417022

**explained_variance**
[0.67324183 0.21276781 0.11399037]

**cumulative_explained_variance**
[0.673, 0.886, 1.0]

*****SLAG_PLAST_PROJ*****

**loadings**
             PC1       PC2       PC3
cement  0.678866 -0.186684  0.710134
water  -0.673708  0.226231  0.703517
age     0.291990  0.956017 -0.027810

**explained_variance**
[0.50515539 0.3174371  0.17740751]

**cumulative_explained_variance**
[0.505, 0.823, 1.0]

*****ASH_PLAST_PROJ*****

**loadings**
             PC1       PC2       PC3
cement  0.598979  0.526561  0.603289
water   0.695826  0.030593 -0.717558
age    -0.396295  0.849587 -0.348071
```

```
**explained_variance**
[0.37747469 0.33069532 0.29182999]

**cumulative_explained_variance**
[0.377, 0.708, 1.0]
```

[12]:
```
cum_expl_var_plotter(all_pcas_3_feat)
```



We can observe from the plot that, even in this case, preserving the variance whilst reducing the number of features is rather difficult. In fact, only in one case using two principal components leads to an acceptable cumulative explained variance of about 0.90. In all the other cases, instead, this value ranges from about 0.70 to 0.82. Losing an average of 30% of the variance to obtain a reduction of just one feature seems a rather high price to pay. We can therefore conclude that the PCA does not simplify the dataset significantly, and in general too many components are necessary to avoid losing too much information due to the dimensionality reduction process. This is clear from the cumulative explained variance plot, and happens for each of the datasets considered. The choice of the three features **age**, **water content** and **cement content** alone does not seem to imply a tangible benefit in the performance of the PCA. In fact, it is not possible to select less than three features while preserving an acceptable amount of information from the dataset. The high variability of the data, that is present in all the features, is perhaps the most important factor in

preventing us from obtaining a significant reduction in dimensions to be considered as a consequence of the PCA. Since there are no few dimensions along which a much higher variance is present, the PCA fails to find few principal components that could help us reduce the dimensionality.

6 Conclusions

In this second part of the Coursework, we further analysed the dataset presented in the preceding work. We highlighted how, in order to use the data in a more sensible manner, we needed to split the dataframe into several sub-dataframes, each of these corresponding to a specific kind of concrete. Building on what had been done in Coursework 1, we identified the three features that we believed were the most significant in terms of predictive ability and information content. Such features were chosen on the basis of the (linear) correlation with the target variable, and on the mechanical nature of concrete itself. Finally, we carried out a PCA both on the dataframes comprising all the available features and on the dataframes containing only the three predictors selected as the most important ones: **age**, **water content**, **cement content**. The results of the PCA, especially the cumulative fraction of total variance explained by an increasing number of principal components, highlighted how difficult it was to efficiently reduce the data dimensionality. In particular, given a reasonable threshold of 0.95, it was impossible to reach the latter without including almost all the principal components. This behaviour was observed in both the cases considered. We conjectured that this was caused by the high variance of all the features, that prevented the PCA algorithm from finding a small number of predictors that could explain a significant proportion of the variance.

References

[1] Kaggle, Concrete Compressive Strength Data, available online at https://www.kaggle.com/vivekgediya/concrete-data. Last accessed 23/02/2021.

[2] Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Machine Learning: Fundamental Concepts and Algorithms, 2nd Edition, Cambridge University Press, March 2020. ISBN: 978-1108473989.

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J., 2011, Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, pp.2825-2830.

7 Appendix

Three files are part of this submission:

1. This report
2. The script used to generate all the plots and the text output cited in this report, in .py format
3. The .csv file containing the original data

The script is interactive, and contains instructions to load the data and produce the output. The output comprises:

1. The pairplots for all the dataframes cited in this report
2. The correlation plots for all the dataframes cited in this report
3. The cumulative variance plots included in this report

4. The textual output of the initial analysis of the whole dataset
5. The textual output containing detailed information about all the PCAs carried out.