# WMMwAD pracownia 3-2023

January 15, 2024

## 1 Pracownia 3

#### 1.0.1 Marcin Koźniewski

**10 grudnia 2023**

[27]:
```
#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).
```

**Ćwiczenia** Ćwiczenie 7. Pobierz plik danych Dry Soy Beans i umieść go w katalogu z zeszytem. Dla tego zbioru danych (wykluczając kolumnę klasy) dokonaj ograniczenia wymiarów:

- o dwa wymiary

[2]:
```
import pandas as pd

#drybeans = pd.read_csv("Dry_Bean_Dataset.csv")
drybeans = pd.read_csv("C:/Users/48511/Documents/Studia_Podyplomowe/jezyk R/
↪wybraneMetodyMatematyczne/zajecia3/Dry_Bean_Dataset.csv")
```

[4]:
```
```

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

[6]:
```
drybeans
```

[6]:
```
        Area  Perimeter  MajorAxisLength  MinorAxisLength  AspectRation  \
0      28395    610.291       208.178117       173.888747      1.197191
1      28734    638.018       200.524796       182.734419      1.097356
2      29380    624.110       212.826130       175.931143      1.209713
3      30008    645.884       210.557999       182.516516      1.153638
4      30140    620.134       201.847882       190.279279      1.060798
…      …        …              …                …               …
13606  42097    759.696       288.721612       185.944705      1.552728
13607  42101    757.499       281.576392       190.713136      1.476439
13608  42139    759.321       281.539928       191.187979      1.472582
13609  42147    763.779       283.382636       190.275731      1.489326
```

```
13610  42159    772.237       295.142741       182.204716      1.619841

       Eccentricity  ConvexArea  EquivDiameter    Extent  Solidity  roundness  \
0          0.549812       28715     190.141097  0.763923  0.988856   0.958027
1          0.411785       29172     191.272750  0.783968  0.984986   0.887034
2          0.562727       29690     193.410904  0.778113  0.989559   0.947849
3          0.498616       30724     195.467062  0.782681  0.976696   0.903936
4          0.333680       30417     195.896503  0.773098  0.990893   0.984877
…               …           …            …         …         …         …
13606      0.765002       42508     231.515799  0.714574  0.990331   0.916603
13607      0.735702       42494     231.526798  0.799943  0.990752   0.922015
13608      0.734065       42569     231.631261  0.729932  0.989899   0.918424
13609      0.741055       42667     231.653248  0.705389  0.987813   0.907906
13610      0.786693       42600     231.686223  0.788962  0.989648   0.888380

       Compactness  ShapeFactor1  ShapeFactor2  ShapeFactor3  ShapeFactor4  \
0         0.913358      0.007332      0.003147      0.834222      0.998724
1         0.953861      0.006979      0.003564      0.909851      0.998430
2         0.908774      0.007244      0.003048      0.825871      0.999066
3         0.928329      0.007017      0.003215      0.861794      0.994199
4         0.970516      0.006697      0.003665      0.941900      0.999166
…              …            …            …            …            …
13606     0.801865      0.006858      0.001749      0.642988      0.998385
13607     0.822252      0.006688      0.001886      0.676099      0.998219
13608     0.822730      0.006681      0.001888      0.676884      0.996767
13609     0.817457      0.006724      0.001852      0.668237      0.995222
13610     0.784997      0.007001      0.001640      0.616221      0.998180

          Class
0         SEKER
1         SEKER
2         SEKER
3         SEKER
4         SEKER
…           …
13606  DERMASON
13607  DERMASON
13608  DERMASON
13609  DERMASON
13610  DERMASON

[13611 rows x 17 columns]
```

[7]:
```
drybeans.iloc[:,16]
```

[7]:
```
0            SEKER
1            SEKER
```

```
2           SEKER
3           SEKER
4           SEKER
          …
13606    DERMASON
13607    DERMASON
13608    DERMASON
13609    DERMASON
13610    DERMASON
Name: Class, Length: 13611, dtype: object
```

[8]:
```
drybeans.iloc[:,16].unique()
```

[8]:
```
array(['SEKER', 'BARBUNYA', 'BOMBAY', 'CALI', 'HOROZ', 'SIRA', 'DERMASON'],
      dtype=object)
```

[9]:
```
drybeans.describe()
```

[9]:
```
                Area    Perimeter  MajorAxisLength  MinorAxisLength  \
count   13611.000000 13611.000000     13611.000000     13611.000000
mean    53048.284549   855.283459       320.141867       202.270714
std     29324.095717   214.289696        85.694186        44.970091
min     20420.000000   524.736000       183.601165       122.512653
25%     36328.000000   703.523500       253.303633       175.848170
50%     44652.000000   794.941000       296.883367       192.431733
75%     61332.000000   977.213000       376.495012       217.031741
max    254616.000000  1985.370000       738.860153       460.198497

       AspectRation Eccentricity     ConvexArea EquivDiameter        Extent  \
count  13611.000000 13611.000000   13611.000000  13611.000000  13611.000000
mean       1.583242     0.750895   53768.200206    253.064220      0.749733
std        0.246678     0.092002   29774.915817     59.177120      0.049086
min        1.024868     0.218951   20684.000000    161.243764      0.555315
25%        1.432307     0.715928   36714.500000    215.068003      0.718634
50%        1.551124     0.764441   45178.000000    238.438026      0.759859
75%        1.707109     0.810466   62294.000000    279.446467      0.786851
max        2.430306     0.911423  263261.000000    569.374358      0.866195

           Solidity    roundness  Compactness ShapeFactor1 ShapeFactor2  \
count  13611.000000 13611.000000 13611.000000 13611.000000 13611.000000
mean       0.987143     0.873282     0.799864     0.006564     0.001716
std        0.004660     0.059520     0.061713     0.001128     0.000596
min        0.919246     0.489618     0.640577     0.002778     0.000564
25%        0.985670     0.832096     0.762469     0.005900     0.001154
50%        0.988283     0.883157     0.801277     0.006645     0.001694
75%        0.990013     0.916869     0.834270     0.007271     0.002170
max        0.994677     0.990685     0.987303     0.010451     0.003665
```

```
       ShapeFactor3  ShapeFactor4
count  13611.000000  13611.000000
mean       0.643590      0.995063
std        0.098996      0.004366
min        0.410339      0.947687
25%        0.581359      0.993703
50%        0.642044      0.996386
75%        0.696006      0.997883
max        0.974767      0.999733
```

[10]:
```
#import seaborn as sb
#sb.pairplot(drybeans.iloc[:,0:15])
```

[7]:
```
from math import sqrt
from sklearn import decomposition
import numpy as np

drybeansnp = drybeans.iloc[:,0:15].to_numpy()
#redukujemy o dwa wymiary
#15 - 2 = 13
pca = decomposition.PCA(n_components=7,svd_solver="full")
pca.fit(drybeansnp)
danePrzeksztalcone = pca.transform(drybeansnp)
```

[12]:
```
danePrzeksztalcone
```

[12]:
```
array([[-3.51499237e+04, -2.20868414e+01,  8.32067957e+01, …,
         2.16130997e+00,  1.21042287e+00, -2.18686126e-02],
       [-3.45863219e+04,  5.85865650e+01,  7.31450790e+01, …,
         9.16348630e+00,  2.54858151e+00, -5.73116091e-02],
       [-3.37640263e+04, -3.89953729e+01,  7.45121433e+01, …,
         2.68196268e+00,  1.10351777e+00, -3.34772697e-02],
       …,
       [-1.56345152e+04, -8.72390530e+01,  1.24999846e+01, …,
        -3.64666220e+00, -5.13075529e-01,  3.25153475e-02],
       [-1.55590550e+04, -2.40310051e+01,  1.37846596e+01, …,
        -3.39085808e+00, -4.73084770e-01,  5.41626471e-02],
       [-1.55983141e+04, -7.82400446e+01, -4.58653078e+00, …,
        -1.19139916e+00, -8.59214579e-01, -3.26622054e-02]])
```

[13]:
```
#print("Pozostawione składowe (w wierszach)")
#print(pca.components_)
print("Ile wariancji wyjaśniały pozostawione składowe (znormalizowane)")
print(pca.explained_variance_ratio_)

print(pca.explained_variance_)
```

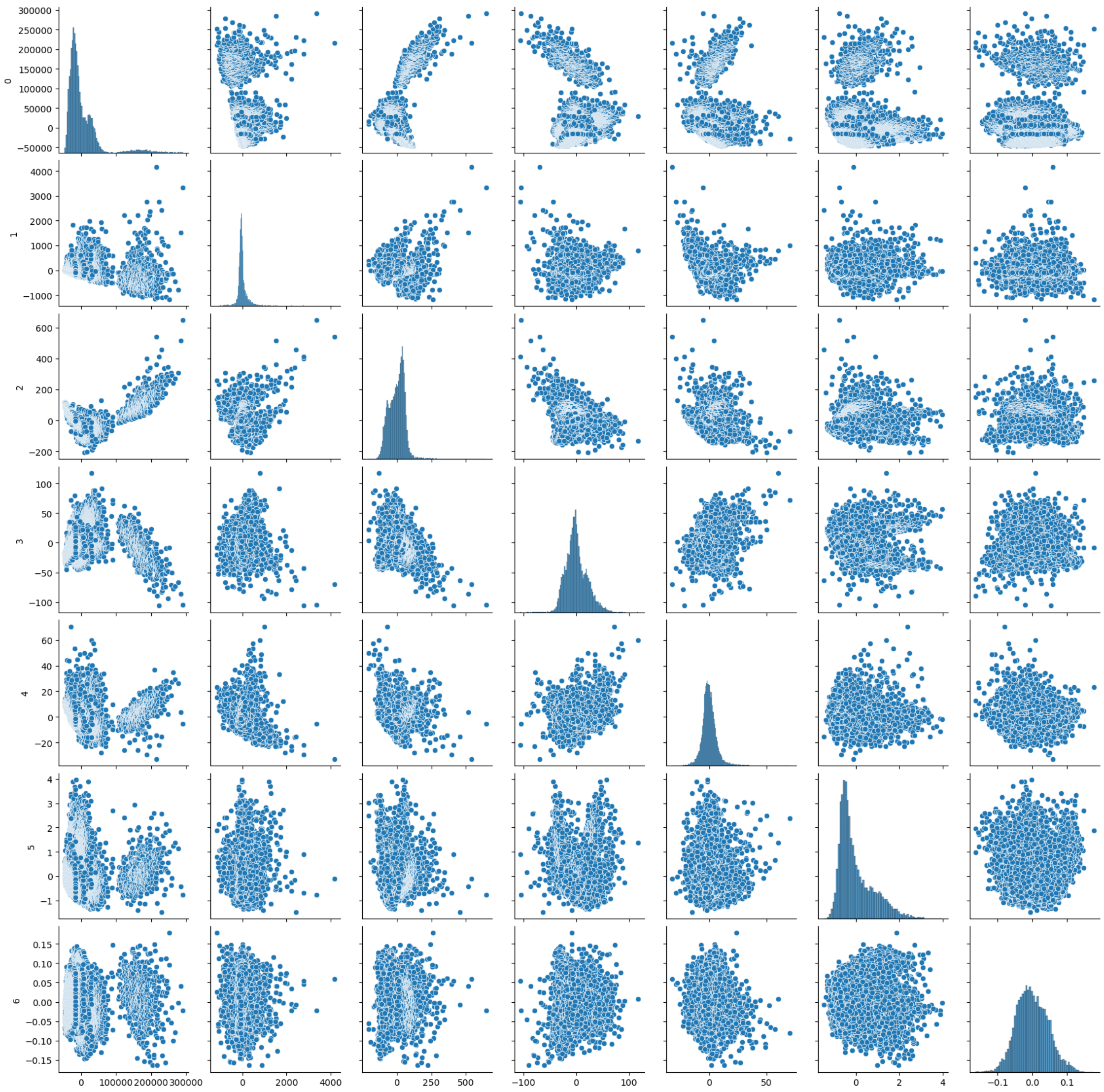
```
print("Ile szumu odrzucono")
print(pca.noise_variance_)
```

```
Ile wariancji wyjaśniały pozostawione składowe (znormalizowane)
[9.99967207e-01 3.06176794e-05 1.92111562e-06 2.29430253e-07
 2.46998543e-08 3.57262984e-10 1.16781839e-12]
[1.74644972e+09 5.34739910e+04 3.35524185e+03 4.00701541e+02
 4.31384683e+01 6.23962299e-01 2.03960298e-03]
Ile szumu odrzucono
0.0001971950590689791
```

Usuwając dwa wymiary usunęliśmy niewiele informacji.

[14]:
```
import seaborn as sb
sb.pairplot(pd.DataFrame( danePrzeksztalcone))
```

[14]:
```
<seaborn.axisgrid.PairGrid at 0x21caa563fa0>
```




## 1.1 Proces przetwarzania danych przy budowaniu modeli

Gdy pracujemy ze zbiorem danych, na którym chcemy zbudować model czy to do klasyfikacji czy regresji, musimy mieć świadomość w jakim celu wykonujemy walidację na odpowiednim zbiorze danych. Chcemy sprawdzić, czy nasz model odpowiednio generalizuje problem i w jakim stopniu jest skuteczny.

Wszystko co wnioskujemy ze zbioru danych (model, sposób przekształcania, itp...) również podlega jednoczesnemu sprawdzeniu jako cały proces pozyskania modelu.

Dlatego w przypadku budowy modelu operację redukcji wymiarów wykonujemy najpierw na zbiorze treningowym i uzyskane przekształcenie stosujemy na zbiorze testowym. Elementy (wiersze) zbioru testowego nie mogą być w takiej sytuacji brane pod uwagę przy poszukiwaniu odpowiedniego przekształcenia zbioru poza weryfikacją finalnego modelu.

Ogólny zarys schematu postępowania: - dokonujemy redukcji wymiarów na zbiorze cech w danych treningowych - budujemy model - działamy uzyskanym przekształceniem na dane testowe - aplikujemy model w celu przetestowania jego działania.

Spróbujmy prześledzić ten proces.

Podział zbioru na zbiór uczący i testujący. Najprostsza metoda.

[3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(drybeans.iloc[:,0:15].
 ↪to_numpy(),
                                                    drybeans.iloc[:,16].
 ↪to_numpy(),
                                                    test_size=0.2)
```

Redukcja wymiarow i uzyskanie przekształcenia

[16]:
```
pca = decomposition.PCA(n_components=7,svd_solver="full")
pca.fit(X_train)
X_train_trans = pca.transform(X_train)
```

Podglądamy wynik redukcji danych

[17]:
```
print("Ile szumu odrzucono")
print(pca.noise_variance_)
```

```
Ile szumu odrzucono
0.00019293422319653484
```

Budujemy model

[18]:
```
from sklearn import svm
clf = svm.SVC(gamma=0.001, C=100.) # dowolny wybrany klasyfikator
clf.fit(X_train_trans, y_train)
```

[18]:
```
SVC(C=100.0, gamma=0.001)
```

Testujemy po przekształceniu zbioru testującego tym samym przekształceniem

[19]:
```
X_test_trans = pca.transform(X_test)
y_test_pred = clf.predict(X_test_trans)
```

Sprawdzamy wynik

[20]:
```
from sklearn.metrics import accuracy_score, confusion_matrix
accuracy_score(y_test, y_test_pred)
```

[20]:
```
0.5673889092912229
```

[21]:
```
confusion_matrix(y_test, y_test_pred)
```

[21]:
```
array([[ 30,   0,  23, 199,   7,   0,  10],
       [  0,   0,   0, 123,   0,   0,   0],
       [  9,   0,  60, 251,   8,   0,   3],
       [  0,   0,   0, 610,   4,  24,  70],
       [  4,   0,   2, 134, 200,   0,  38],
       [  0,   0,   2,  69,   0, 261,  63],
       [  2,   0,   2,  96,  23,  12, 384]], dtype=int64)
```

### 1.1.1 Zadania samodzielne

Proszę sporządzić raport z wykonania poniższych zadań. Może to być prosty plik tekstowy lub notes Jupyter z wykonanymi poleceniami i wynikiem ich wywołania z dodatkiem komentarzy.

Zadanie 1. Dla całego zbioru danych Dry Bean zredukuj wymiar danych o 5, 7, 10 wymiarów wykorzystując PCA. Odnotuj ile zostało usuniętego szumu.

[23]:
```
for item in [5,7,10]:
    pca = decomposition.PCA(n_components=item,svd_solver="full")
    pca.fit(X_train)
    print(pca.noise_variance_)
```

```
0.06321983118795146
0.00019293422319653484
2.748506319277072e-06
```

Zadanie 2. Pobierz zbiór danych CoverType. Spróbuj zbudować dowolny klasyfikator na odpowiednio wydzielonym zbiorze trenującym (patrz opis procesu działania wyżej). Wykonaj redukcję wymiarów przy wykorzystaniu PCA (pomijając ostatnią kolumnę - klasa). naucz model tego samego typu i sprawdź na zbiorze testowym czy model klasyfikuje go lepiej. Uwaga: prowadzący nie zna wyniku :-)

[34]:
```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in
c:\users\48511\anaconda3\lib\site-packages (1.2.1)
Requirement already satisfied: numpy>=1.17.3 in
c:\users\48511\anaconda3\lib\site-packages (from scikit-learn) (1.23.5)
Requirement already satisfied: scipy>=1.3.2 in
c:\users\48511\anaconda3\lib\site-packages (from scikit-learn) (1.10.0)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\48511\anaconda3\lib\site-packages (from scikit-learn) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\48511\anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
```

[9]:
```
for item in range(1,55):
    print(item)
```

```
1
2
3
```

52
53
54

[5]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

ctDF = pd.read_csv("covertype_csv.csv")
X_train, X_test, y_train, y_test= train_test_split(ctDF.iloc[:,0:54].to_numpy(),
                                                    ctDF.iloc[:,54].to_numpy(),
                                                    test_size=0.2)


clf = DecisionTreeClassifier()
```

[8]:
```
for i in range(1,ctDF.shape[1] + 1):
    pca = decomposition.PCA(n_components=i,svd_solver="full")
    x_train_after_sel = pca.fit_transform(X_train)
    x_test_after_sel = pca.transform(X_test)
    clf.fit(x_train_after_sel, y_train)
    y_pred = clf.predict(x_test_after_sel)
    print(f'for {i} selected features the accuracy score is␣
 ↪{accuracy_score(y_test, y_pred)}')
```

```
for 1 selected features the accuracy score is 0.45486777449807664
for 2 selected features the accuracy score is 0.5743311274235605
for 3 selected features the accuracy score is 0.7424507112553033
for 4 selected features the accuracy score is 0.7960982074473121
for 5 selected features the accuracy score is 0.823248969475831
for 6 selected features the accuracy score is 0.8361402029207508
for 7 selected features the accuracy score is 0.8585406572980043
for 8 selected features the accuracy score is 0.8778172680567627
for 9 selected features the accuracy score is 0.8822233505159075
for 10 selected features the accuracy score is 0.8812767312375757
for 11 selected features the accuracy score is 0.8843145185580407
for 12 selected features the accuracy score is 0.8907601352805005
for 13 selected features the accuracy score is 0.8959751469411289
for 14 selected features the accuracy score is 0.8991678355980482
for 15 selected features the accuracy score is 0.903608340576405
for 16 selected features the accuracy score is 0.9024810030722098
for 17 selected features the accuracy score is 0.9018613977263926
for 18 selected features the accuracy score is 0.9033673829419206
for 19 selected features the accuracy score is 0.9046151992633581
for 20 selected features the accuracy score is 0.9056822973589322
for 21 selected features the accuracy score is 0.9043140022202525
for 22 selected features the accuracy score is 0.9044344810374947
for 23 selected features the accuracy score is 0.9041677065136012
for 24 selected features the accuracy score is 0.9029973408603909
```

```
for 25 selected features the accuracy score is 0.9006393982943641
for 26 selected features the accuracy score is 0.9002521449532284
for 27 selected features the accuracy score is 0.8996755677564262
for 28 selected features the accuracy score is 0.8981265543918832
for 29 selected features the accuracy score is 0.8972229632625663
for 30 selected features the accuracy score is 0.9128679982444515
for 31 selected features the accuracy score is 0.9165167852809308
for 32 selected features the accuracy score is 0.918616558952867
for 33 selected features the accuracy score is 0.9163704895742795
for 34 selected features the accuracy score is 0.9174289820400506
for 35 selected features the accuracy score is 0.9160434756417648
for 36 selected features the accuracy score is 0.917144996256551
for 37 selected features the accuracy score is 0.9172310525545813
for 38 selected features the accuracy score is 0.9170159118095058
for 39 selected features the accuracy score is 0.917101968107536
for 40 selected features the accuracy score is 0.9139437019698287
for 41 selected features the accuracy score is 0.9137629837439653
for 42 selected features the accuracy score is 0.9130228995809059
for 43 selected features the accuracy score is 0.9132036178067692
for 44 selected features the accuracy score is 0.9143998003493886
for 45 selected features the accuracy score is 0.9158025180072804
for 46 selected features the accuracy score is 0.916077898160977
for 47 selected features the accuracy score is 0.9129626601722847
for 48 selected features the accuracy score is 0.9128938151338606
for 49 selected features the accuracy score is 0.9143481665705705
for 50 selected features the accuracy score is 0.9130142939511028
for 51 selected features the accuracy score is 0.9140813920466769
for 52 selected features the accuracy score is 0.9140813920466769
for 53 selected features the accuracy score is 0.9139264907102226
for 54 selected features the accuracy score is 0.9137113499651472
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[8], line 3
      1 for i in range(1,ctDF.shape[1] + 1):
      2     pca = decomposition.PCA(n_components=i,svd_solver="full")
----> 3     x_train_after_sel = pca.fit_transform(X_train)
      4     x_test_after_sel = pca.transform(X_test)
      5     clf.fit(x_train_after_sel, y_train)

File ~\anaconda3\lib\site-packages\sklearn\utils\_set_output.py:142, in
 ↪_wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    140 @wraps(f)
    141 def wrapped(self, X, *args, **kwargs):
--> 142     data_to_wrap = f(self, X, *args, **kwargs)
    143     if isinstance(data_to_wrap, tuple):
    144         # only wrap the first output for cross decomposition
```

```
    145         return (
    146             _wrap_data_with_container(method, data_to_wrap[0], X, self)
    147             *data_to_wrap[1:],
    148         )

File ~\anaconda3\lib\site-packages\sklearn\decomposition\_pca.py:462, in PCA.
 ↪fit_transform(self, X, y)
    439 """Fit the model with X and apply the dimensionality reduction on X.
    440 
    441 Parameters
   (…)
    458 C-ordered array, use 'np.ascontiguousarray'.
    459 """
    460 self._validate_params()
--> 462 U, S, Vt = self._fit(X)
    463 U = U[:, : self.n_components_]
    465 if self.whiten:
    466     # X_new = X * V / S * sqrt(n_samples) = U * sqrt(n_samples)

File ~\anaconda3\lib\site-packages\sklearn\decomposition\_pca.py:512, in PCA.
 ↪_fit(self, X)
    510 # Call different fits for either full or truncated SVD
    511 if self._fit_svd_solver == "full":
--> 512     return self._fit_full(X, n_components)
    513 elif self._fit_svd_solver in ["arpack", "randomized"]:
    514     return self._fit_truncated(X, n_components, self._fit_svd_solver)

File ~\anaconda3\lib\site-packages\sklearn\decomposition\_pca.py:526, in PCA.
 ↪_fit_full(self, X, n_components)
    522         raise ValueError(
    523             "n_components='mle' is only supported if n_samples >=␣
 ↪n_features"
    524         )
    525 elif not 0 <= n_components <= min(n_samples, n_features):
--> 526     raise ValueError(
    527         "n_components=%r must be between 0 and "
    528         "min(n_samples, n_features)=%r with "
    529         "svd_solver='full'" % (n_components, min(n_samples, n_features))
    530     )
    532 # Center data
    533 self.mean_ = np.mean(X, axis=0)

ValueError: n_components=55 must be between 0 and min(n_samples, n_features)=54
 ↪with svd_solver='full'
```

```
[ ]:
```