Understanding A Dataset

Mariana Ávalos Arce
Data Mining, Fall 2021

1 Breast Cancer Wisconsin (Diagnostic) Data Set

1.1 Data Set Information

This is the first data set selected for this report. It is owned by UCI Machine Learning[2], and was found through Kaggle's data set browser. In simple words, this data set is used to train Machine Learning algorithms in order to **predict the diagnosis of breast cancer in future female patients**. The features or attributes presented in the data set were extracted from digitized images of a breast cancer cell in a tumor, describing specifically the cell nuclei presented in the images. The repository cited above explains all the attributes that the data set contains, so that the column identifier is understood. Said explanation is the following:

1. ID number of the patient
2. Diagnosis (M = malignant, B = benign)

   From column 3 - 32, they explain there are ten real-valued features computed for each cell nucleus:
3. cell radius
4. texture, which is a standard deviation from a gray scale image
5. perimeter of the core tumor
6. area of the core tumor
7. smoothness, which is the variance of the different radiuses of the tumor
8. compactness ($perimeter^2/area - 1.0$)
9. concavity, which is the severity of concave portions of the contour
10. concave points, which is the number of concave portions of the contour
11. symmetry
12. fractal dimension, which is computed as "coastline approximation" - 1.0

    All cell-related data is measured in micrometers.

The mean, standard error and "worst" or largest (mean of the three largest values) of these ten features were computed for each cell image, resulting in 30 features. UCI ML gives an example: field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius. That is why these ten values occupy attributes 3 - 32. UCI ML specifies that there are no missing attribute values in this data set.

1.2 Data Set Analysis

head() - Getting to know our data set

The first thing to understand our data set is to get a glimpse of the instances we have:

```
import pandas as pd
import matplotlib as plt
import numpy as np

url = 'db/data.csv'
bcancer = pd.read_csv(url, header=0)
print(bcancer.head())
```

This will output the first five instances of the data set in a table format:

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  ...          17.33           184.60      2019.0            0.1622
1  ...          23.41           158.80      1956.0            0.1238
2  ...          25.53           152.50      1709.0            0.1444
3  ...          26.50            98.87       567.7            0.2098
4  ...          16.67           152.20      1575.0            0.1374

   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

Figure 1: *head*() function call that outputs the first 5 instances of each attribute.

The shape Property

```
print(bcancer.shape)
```

The *shape* property of our data set tells us the dimensions of the set, which is the number of instances (rows) and attributes (columns) that is has, without counting the headers. This is printed as:

```
(569, 33)
```

As we can see, here is the first inconsistency in our data: the data set owners defined it to be 32-columns long, and here we are seeing 33 columns, in addition to the fact that in the *head*() output we see the last column as `Unnamed`, and full of `NaN` values. This unnecessary column will definitely need to be cleaned in the data treatment phase.

The dtypes Property

```
print(bcancer.dtypes)
```

```
id                         int64
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
radius_se                float64
texture_se               float64
perimeter_se             float64
area_se                  float64
smoothness_se            float64
compactness_se           float64
concavity_se             float64
concave points_se        float64
symmetry_se              float64
fractal_dimension_se     float64
radius_worst             float64
texture_worst            float64
perimeter_worst          float64
area_worst               float64
smoothness_worst         float64
compactness_worst        float64
concavity_worst          float64
concave points_worst     float64
symmetry_worst           float64
fractal_dimension_worst  float64
Unnamed: 32              float64
dtype: object
```

Figure 2: *dtypes* property that outputs the data type of all attributes.

The *diagnosis* attribute is of type *object*, when indeed it is a `string` variable, but this is because pandas' dtypes come from Numpy, and since it is a numeric library, anything that is not a number is a type *object*. All the other attributes are of type $float64$, as described in the owner's data set details.

describe() - Data set summary

```
pd.set_option('precision',2)
bcancer.describe()
```

Here we are setting the floating point precision to two decimal places, and then use the function. This function outputs Figure 3.

This table (Figure 3) ignores the attribute `diagnosis` since it is an object type and cannot be applied to the formulas seen above. The function outputs a table with the mean, standard deviation, minimum, maximum and

	id	radius_mean	texture_mean	perimeter_mean	area_mean	\
count	5.69e+02	569.00	569.00	569.00	569.00	
mean	3.04e+07	14.13	19.29	91.97	654.89	
std	1.25e+08	3.52	4.30	24.30	351.91	
min	8.67e+03	6.98	9.71	43.79	143.50	
25%	8.69e+05	11.70	16.17	75.17	420.30	
50%	9.06e+05	13.37	18.84	86.24	551.10	
75%	8.81e+06	15.78	21.80	104.10	782.70	
max	9.11e+08	28.11	39.28	188.50	2501.00	

	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	\
count	569.00	569.00	569.00	569.00	
mean	0.10	0.10	0.09	0.05	
std	0.01	0.05	0.08	0.04	
min	0.05	0.02	0.00	0.00	
25%	0.09	0.06	0.03	0.02	
50%	0.10	0.09	0.06	0.03	
75%	0.11	0.13	0.13	0.07	
max	0.16	0.35	0.43	0.20	

	symmetry_mean	...	texture_worst	perimeter_worst	area_worst	\
count	569.00	...	569.00	569.00	569.00	
mean	0.18	...	25.68	107.26	880.58	
std	0.03	...	6.15	33.60	569.36	
min	0.11	...	12.02	50.41	185.20	
25%	0.16	...	21.08	84.11	515.30	
50%	0.18	...	25.41	97.66	686.50	
75%	0.20	...	29.72	125.40	1084.00	
max	0.30	...	49.54	251.20	4254.00	

	smoothness_worst	compactness_worst	concavity_worst	\
count	569.00	569.00	569.00	
mean	0.13	0.25	0.27	
std	0.02	0.16	0.21	
min	0.07	0.03	0.00	
25%	0.12	0.15	0.11	
50%	0.13	0.21	0.23	
75%	0.15	0.34	0.38	
max	0.22	1.06	1.25	

	concave points_worst	symmetry_worst	fractal_dimension_worst	\
count	569.00	569.00	569.00	
mean	0.11	0.29	0.08	
std	0.07	0.06	0.02	
min	0.00	0.16	0.06	
25%	0.06	0.25	0.07	
50%	0.10	0.28	0.08	
75%	0.16	0.32	0.09	
max	0.29	0.66	0.21	

Figure 3: *describe*() function output with the main descriptive statistics of the attributes.

quartile information of the data set, as a means of summary.

groupby('class').size() - Number of instances of each class

```
for c in bcancer.columns:
    print(bcancer.groupby(c).size())
```

This prints the number of different or unique instances per class or attribute in the data set, in this way we would check easily for inconsistencies in the amounts of instances. Since the output is quite large, I will just show the output when `c = 'diganosis'`:

```
    diagnosis
B     357
M     212
dtype: int64
```

Where we see that we can only have two values: B for benign and M for malign, plus the amount of times each value appears in the diagnosis column. This will be important in the report's conclusions.

hist() - Data visualization

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(18,18))
ax = fig.gca()
bcancer.hist(ax=ax, color='powderblue')
plt.show()
```

Which shows the plots in Figure 4. The histograms plotted in said image are describing the frequency in which the instances' data is repeated inside each column or attribute that is numeric.

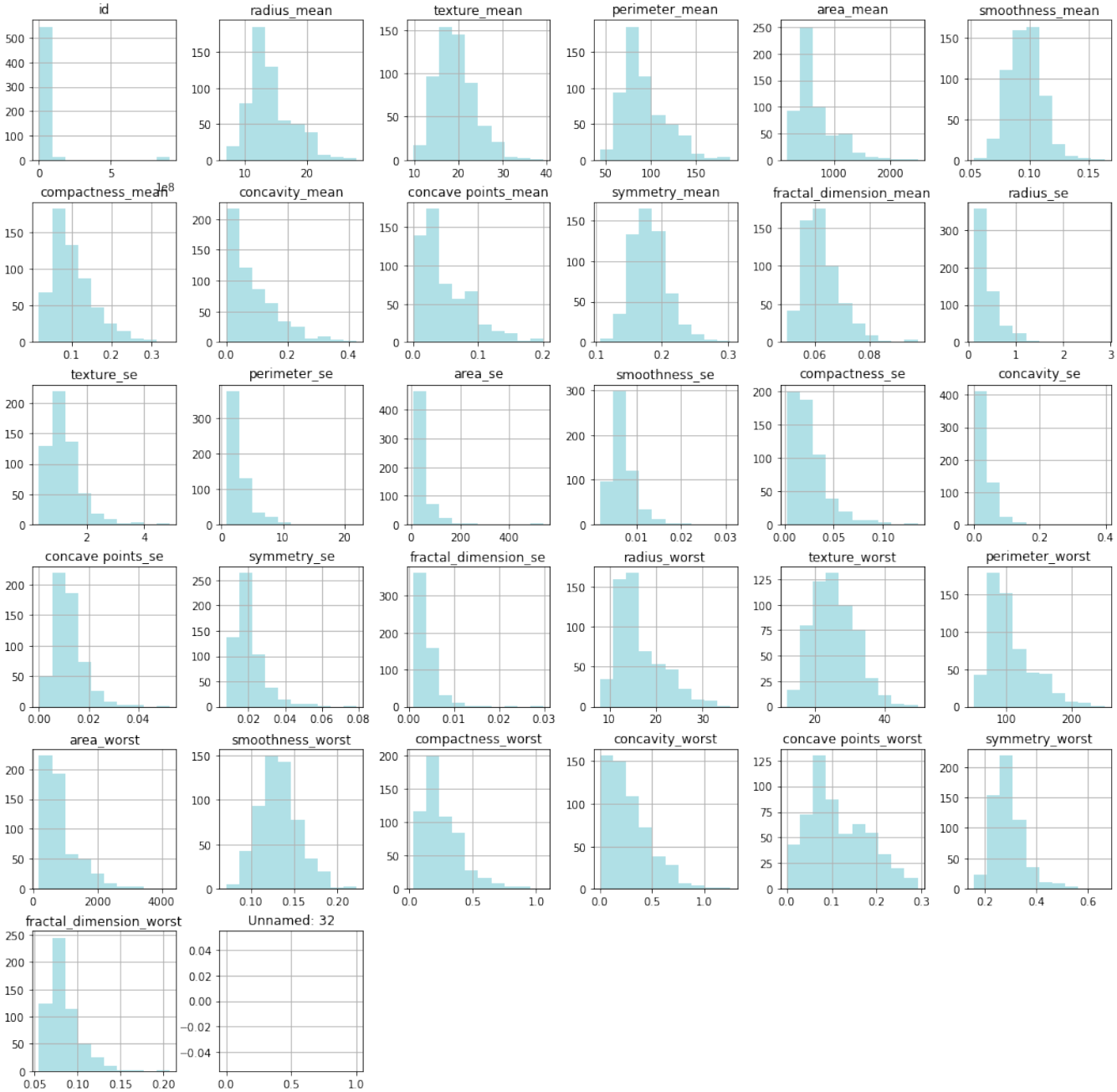


Figure 4: *hist*() function output with the histograms for each attribute frequency.

plot(kind='density') - Data visualization

```
fig = plt.figure(figsize=(18,18))
ax = fig.gca()
bcancer.plot(ax=ax, kind='density', subplots=True, layout=(9,4))
plt.show()
```

Which similarly outputs the plots in Figure 5. I avoided the label of each attribute over its plot because it would cover a lot of the important parts of the plot. The order of attributes follows the same as in Figure 4.

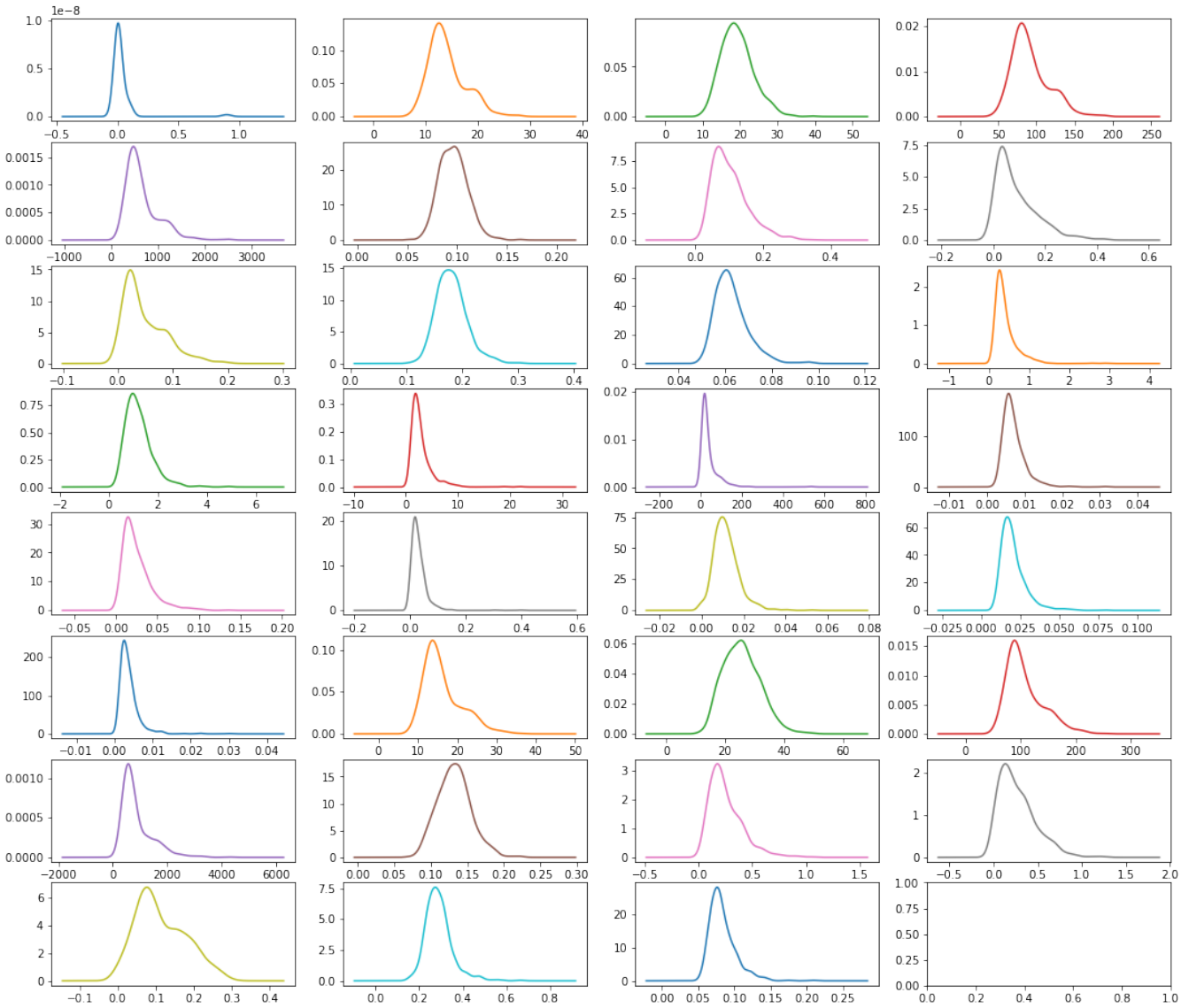


Figure 5: *plot*() function output with the density plots for each attribute.

plot(kind='box') - Data visualization

```
fig = plt.figure(figsize=(20,25))
ax = fig.gca()
bcancer.plot(ax=ax, kind='box', subplots=True, layout=(9,4))
plt.show()
```

In this case, the output is a series of box plots for each of the data set attributes. This plot shows the range of the instances through the box extended lines, the side of the data range to which the 50% of the instances tend

to tilt and the atypical instances as circles outside the box and its lines (Figure 6).

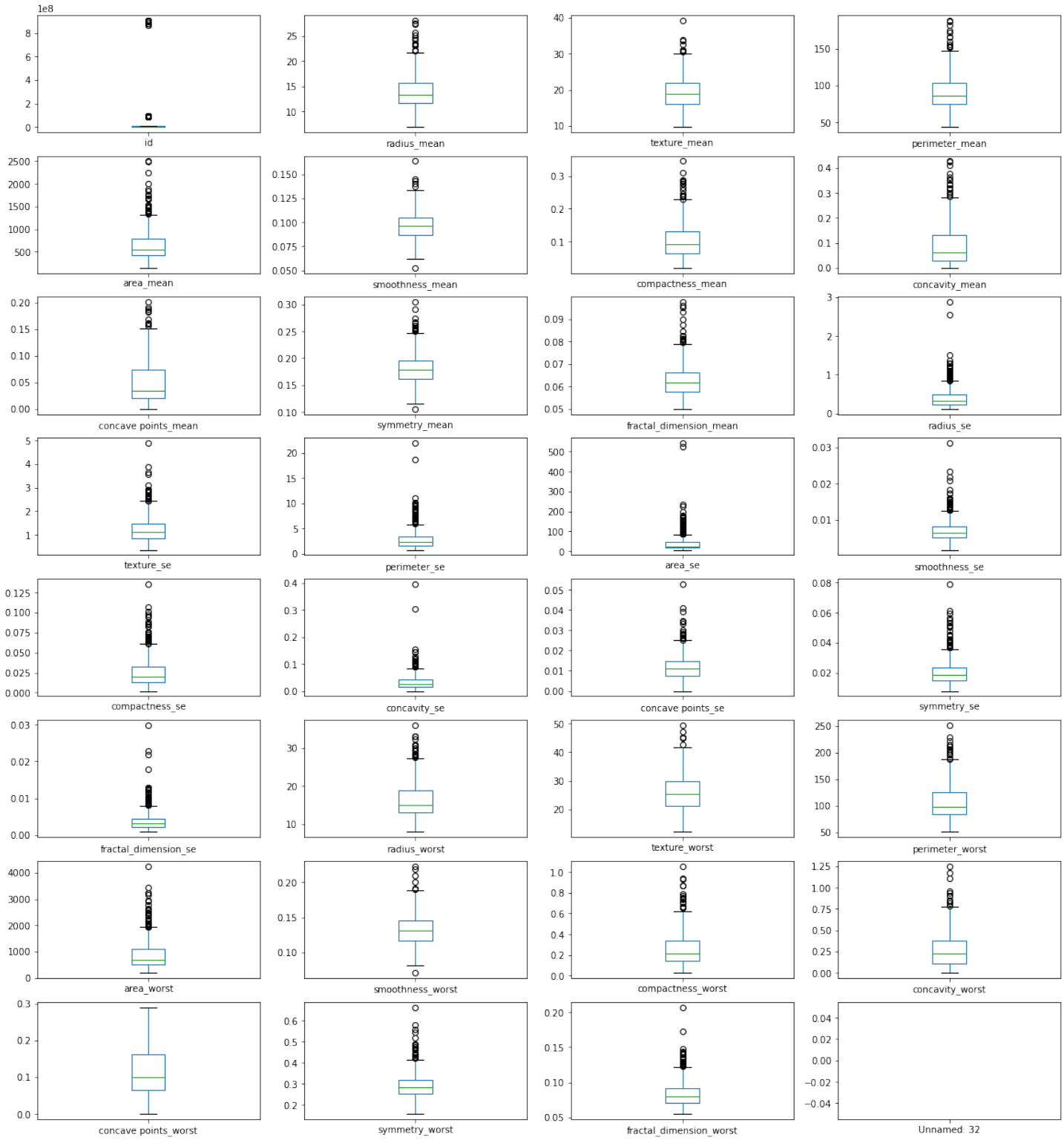


Figure 6: *plot*() function output with the box plots for each attribute.

1.3 Conclusions On Data Set Analysis

This data set has almost all of its fields filled with floating point instances, which would make a perfect match for a **correlation** analysis using the Pearson method. But for this, we must remember the main purpose of the data set: the diagnosis of breast cancer, which located in the *diagnosis* attribute. This attribute is of type *object*,

and so an important conclusion would be that we need to transform the *diagnosis* column to numerical values (0 for benign and 1 for malign, for example). In this way, we can perform a *corr*() function call with the Pearson technique and find **redundant data** quite easily with the correlation between already highly correlated attributes and the *diagnosis* column, in order to decide what to clean from the set. Regarding the central tendency measures, we can say that this data set has a small standard deviation in all attributes except for the ones related to *area* or *perimeter* of the tumor, because those magnitudes include larger instances. In terms of conclusions from the plots, The attribute `texture_mean` seems to be the most gaussian-like regrading the distribution, both because of its histogram and density plots. But in general, this instances tend to have a **positive skew value**, because all the histograms and density plots are tilted towards the right. Also, by looking at the box plots, many instances could be atypical in all attributes, except for the `concave_worst` that has no atypical value plotted.

2 Alzheimer Features Data Set

2.1 Data Set Information

This data set was also found [3] through Kaggle's data set browser, but in this case, the owner of this data set is not specified. This data set is used for training specific Machine Learning algorithms: **unsupervised (clustering) and Classification predictions.** Luckily, the data set contained a very specific explanation for each of the 10 columns or attributes it contains:

1. Group, or Class (Demented or Nondemented)
2. M/F, which is either Male(M)/Female(F)
3. Age
4. EDUC, which corresponds to Years of Education
5. SES, which means Socioeconomic Status and goes from 1-5
6. MMSE, which stands for Mini Mental State Examination
7. CDR, which corresponds to Clinical Dementia Rating (0.0 - 2.0)
8. eTIV, which is Estimated total intracranial volume
9. nWBV, which stands for Normalized Whole Brain Volume
10. ASF, which represents the Atlas Scaling Factor

2.2 Data Set Analysis

head() - Getting to know our data set

```
import pandas as pd
import matplotlib as plt
import numpy as np
url = 'db/alzheimer.csv'
alz = pd.read_csv(url, header=0)
print(alz.head())
```

Much like the first data set, the first attribute describes the type of diagnosis, but in this case, the attribute is called *Group* and its options can be either *Demented* or *Nondemented* (with or without dementia). The next thing we can see through the instances is that this data set contains much more mixed measure units: education (years), socio-economic status (scale), and floating point numbers. The output looks as Figure 7.

	Group	M/F	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
0	Nondemented	M	87	14	2.0	27.0	0.0	1987	0.696	0.883
1	Nondemented	M	88	14	2.0	30.0	0.0	2004	0.681	0.876
2	Demented	M	75	12	NaN	23.0	0.5	1678	0.736	1.046
3	Demented	M	76	12	NaN	28.0	0.5	1738	0.713	1.010
4	Demented	M	80	12	NaN	22.0	0.5	1698	0.701	1.034

Figure 7: *head*() function call that outputs the first 5 instances.

The shape Property

```
print(alz.shape)
```

This property once again tells us the dimensions of our set, which from the output below, we can tell it is a smaller data set than the first in both dimensions. The output (*rows* × *columns*) looks as below.

```
(373, 10)
```

The dtypes Property

```
print(alz.dtypes)
```

This property will tell us what we have been noticing since the beginning: this data set has instances with a little bit more variety than the first one. It shows more integer values for those attributes that use scales. The output is below.

```
Group     object
M/F       object
Age        int64
EDUC       int64
SES      float64
MMSE     float64
CDR      float64
eTIV       int64
nWBV     float64
ASF      float64
dtype: object
```

describe() - Data set summary

```
pd.set_option('precision',2)
print(alz.describe())
```

Here we see the primary summary of the set with some central tendency values and quartile information, and the first thing to notice is that there are some missing values in the column labeled as *SES* or socio-economic status: there are 354 values counted while the other columns have 373 instances. Such output can be seen in Figure 8.

	Age	EDUC	SES	MMSE	CDR	eTIV	nWBV	ASF
count	373.00	373.00	354.00	371.00	373.00	373.00	373.00	373.00
mean	77.01	14.60	2.46	27.34	0.29	1488.13	0.73	1.20
std	7.64	2.88	1.13	3.68	0.37	176.14	0.04	0.14
min	60.00	6.00	1.00	4.00	0.00	1106.00	0.64	0.88
25%	71.00	12.00	2.00	27.00	0.00	1357.00	0.70	1.10
50%	77.00	15.00	2.00	29.00	0.00	1470.00	0.73	1.19
75%	82.00	16.00	3.00	30.00	0.50	1597.00	0.76	1.29
max	98.00	23.00	5.00	30.00	2.00	2004.00	0.84	1.59

Figure 8: *describe*() function call that outputs the statistic summary of the set.

groupby('class').size() - Number of instances of each class

```
alz.groupby('SES').size()
```

I decided to check this *SES* class because ever since the *head*() function, we could see `NaN` values in the class, so this function will give further detail on what is going on, since it will count the appearances of each different values through the class instances. The output is below.

```
SES
1.0     88
2.0    103
3.0     82
4.0     74
5.0      7
dtype: int64
```

hist() - Data visualization

```
fig = plt.figure(figsize=(15,5))
ax = fig.gca()
alz.hist(ax=ax, color='powderblue', layout=(2,4))
plt.show()
```

Whose output is in Figure 9.

plot(kind='density') - Data visualization

```
fig = plt.figure(figsize=(15,5))
ax = fig.gca()
alz.plot(ax=ax, kind='density', subplots=True, layout=(2,4))
plt.show()
```

This function outputs a similar plot to the previous one, but following a line. The output is in Figure 10.

plot(kind='box') - Data visualization

```
fig = plt.figure(figsize=(15,5))
ax = fig.gca()
alz.plot(ax=ax, kind='box', subplots=True, layout=(2,4))
plt.show()
```

This makes a box plot for every class or attribute in out data set (Figure 11).

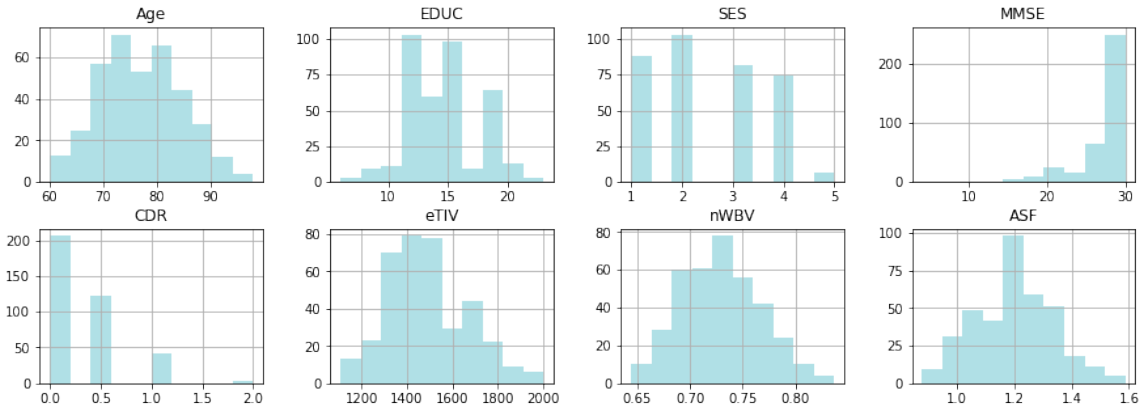


Figure 9: *hist*() function call that outputs bar graphs for all classes.

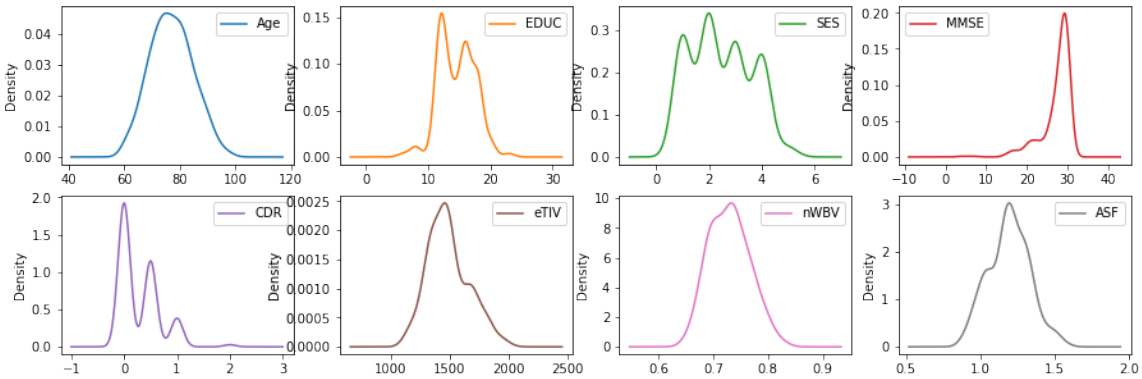


Figure 10: *plot*() function call that outputs density graphs for all classes.

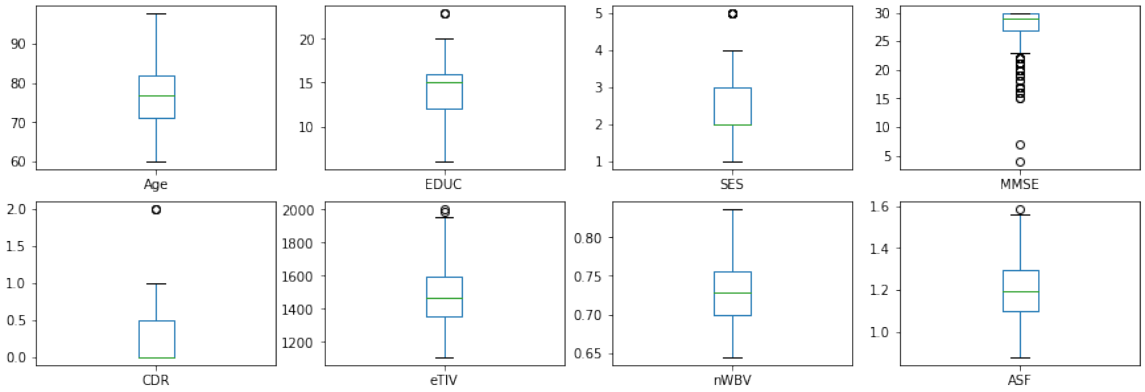


Figure 11: *plot*() function call that outputs a box plot per class.

2.3 Conclusions On Data Set Analysis

This data set has an important feature: it is smaller in dimensions. This helped enormously regarding the visualization functions, because comparing 10 plots is much more accessible than comparing 30, like in the previous data set, which allows better conclusions. Regarding future actions, I would conclude by suggesting also a transformation for the *Group* attribute, in order to make it have binary values (0 and 1) instead of words in order to perform an interesting correlation analysis with the Pearson method together with all the other classes. At last, the plots were quite insightful: the histogram and the density plots show relatively symmetric gauss-like distributions in the continuous attributes, but also the box plots suggest that there are significantly few atypical instances in the data set, and their median seems to be quite in the center of the plot for most of the cases.

3 Heart Failure Clinical Records Data Set

3.1 Data Set Information

This data set is once again found by Kaggle's data set browser. This data set was used and published in the article *Machine learning can predict survival of patients with heart failure from serum creatinine and ejection fraction alone* in 2020 by BMC Medical Informatics and Decision Making [1]. It is further explained that the data set was used to predict the chances of a patient to survive a cardiac arrest or simply to predict the mortality in the case of a heart failure. It also contained a brief explanation as to what the attribute labels mean:

1. Age
2. Anaemia, decrease of red blood cells or hemoglobin (boolean)
3. Creatinine phosphokinase, level of the CPK enzyme in the blood (mcg/L)
4. Diabetes, if the patient has diabetes (boolean)
5. Ejection fraction, percentage of blood leaving the heart at each contraction (percentage)
6. High blood pressure, if the patient has hypertension (boolean)
7. Platelets, which represent platelets in the blood (kiloplatelets/mL)
8. Serum creatinine, level of serum creatinine in the blood (mg/dL)
9. Serum sodium, level of serum sodium in the blood (mEq/L)
10. Sex, woman or man (binary)
11. Smoking, if the patient smokes or not (boolean)
12. Time, follow-up period (days)
13. Death event, if the patient deceased during the follow-up period (boolean)

3.2 Data Set Analysis

head() - Getting to know our data set

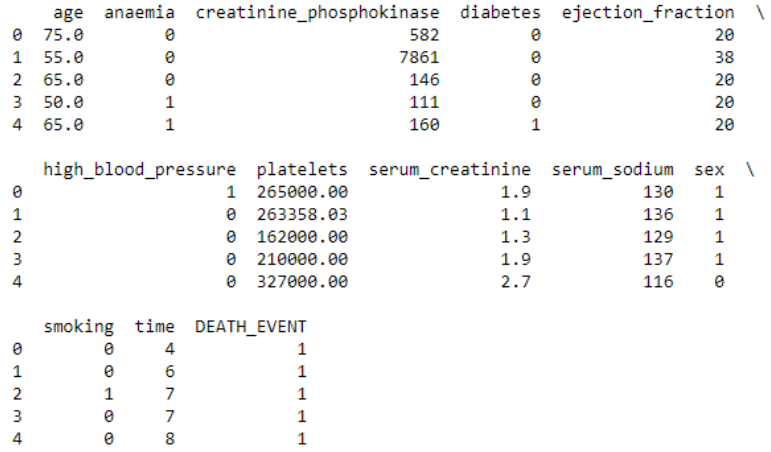
```
import pandas as pd
import matplotlib as plt
import numpy as np
url = 'db/heart_failure_clinical_records_dataset.csv'
heart = pd.read_csv(url, header=0)
print(heart.head())
```

The first thing to perform is the *head*() function to get a glimpse of the instances we have. Surprisingly, there is no ID for the instances. This is quite a good idea in order to save space and make a data set as compact as possible, because an ID can be computed automatically in most of the processing engines that will use the data set, I suppose. The output seems quite expected (Figure 12).

The shape Property

```
print(heart.shape)
```

Fortunately, we have also a considerably smaller data set than the first, which we can see instantly from the output of the *shape* property, which represents the rows and columns, respectively:



```
    age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
0  75.0        0                       582         0                 20
1  55.0        0                      7861         0                 38
2  65.0        0                       146         0                 20
3  50.0        1                       111         0                 20
4  65.0        1                       160         1                 20

   high_blood_pressure  platelets  serum_creatinine  serum_sodium  sex  \
0                    1  265000.00               1.9           130    1
1                    0  263358.03               1.1           136    1
2                    0  162000.00               1.3           129    1
3                    0  210000.00               1.9           137    1
4                    0  327000.00               2.7           116    0

   smoking  time  DEATH_EVENT
0        0     4            1
1        0     6            1
2        1     7            1
3        0     7            1
4        0     8            1
```

Figure 12: *head*() function call that outputs the first 5 instances of all attributes.

```
(299, 13)
```

The dtypes Property

```
print(heart.dtypes)
```

This property will help to confirm what we can see from the *head*() function: there are a lot of binary/boolean (0 or 1) attributes in this data set, but they will end up being treated as integers in the output of *dtypes*. There are more integer values in this data set than floating point ones, and there are no *object* values. The output is below.

```
age                         float64
anaemia                       int64
creatinine_phosphokinase      int64
diabetes                      int64
ejection_fraction             int64
high_blood_pressure           int64
platelets                   float64
serum_creatinine            float64
serum_sodium                  int64
sex                           int64
smoking                       int64
time                          int64
DEATH_EVENT                   int64
dtype: object
```

describe() - Data set summary

```
pd.set_option('precision',2)
print(heart.describe())
```

This function will print the basic and most general statistic calculations for each of the attributes. From the count, we can see that there are no missing values. The output is in Figure 13.

groupby('class').size() - Number of instances of each class

```
print(heart.groupby('DEATH_EVENT').size())
```

```
              age  anaemia  creatinine_phosphokinase  diabetes  ejection_fraction  \
count  299.00   299.00                    299.00    299.00             299.00
mean    60.83     0.43                    581.84      0.42              38.08
std     11.89     0.50                    970.29      0.49              11.83
min     40.00     0.00                     23.00      0.00              14.00
25%     51.00     0.00                    116.50      0.00              30.00
50%     60.00     0.00                    250.00      0.00              38.00
75%     70.00     1.00                    582.00      1.00              45.00
max     95.00     1.00                   7861.00      1.00              80.00

       high_blood_pressure  platelets  serum_creatinine  serum_sodium     sex  \
count               299.00     299.00            299.00        299.00  299.00
mean                  0.35  263358.03              1.39        136.63    0.65
std                   0.48   97804.24              1.03          4.41    0.48
min                   0.00   25100.00              0.50        113.00    0.00
25%                   0.00  212500.00              0.90        134.00    0.00
50%                   0.00  262000.00              1.10        137.00    1.00
75%                   1.00  303500.00              1.40        140.00    1.00
max                   1.00  850000.00              9.40        148.00    1.00

       smoking    time  DEATH_EVENT
count   299.00  299.00       299.00
mean      0.32  130.26         0.32
std       0.47   77.61         0.47
min       0.00    4.00         0.00
25%       0.00   73.00         0.00
50%       0.00  115.00         0.00
75%       1.00  203.00         1.00
max       1.00  285.00         1.00
```

Figure 13: *describe*() function call that outputs basic statistics for all attributes.

Since mortality probability is the key functionality of this data set, I decided to see the count of all dead people due to cardiac arrest (1's) compared to the amount of survivals (0's) by using the *groupby* method. The output is shown below.

```
DEATH_EVENT
0    203
1     96
dtype: int64
```

hist() - Data visualization

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(18,9))
ax = fig.gca()
heart.hist(ax=ax, color='powderblue', layout=(3,5))
plt.show()
```

Where the output is in Figure 14.

plot(kind='density') - Data visualization

```
fig = plt.figure(figsize=(18,9))
ax = fig.gca()
heart.plot(ax=ax, kind='density', subplots=True, layout=(3,5))
plt.show()
```

Whose output is shown in Figure 15.

plot(kind='box') - Data visualization

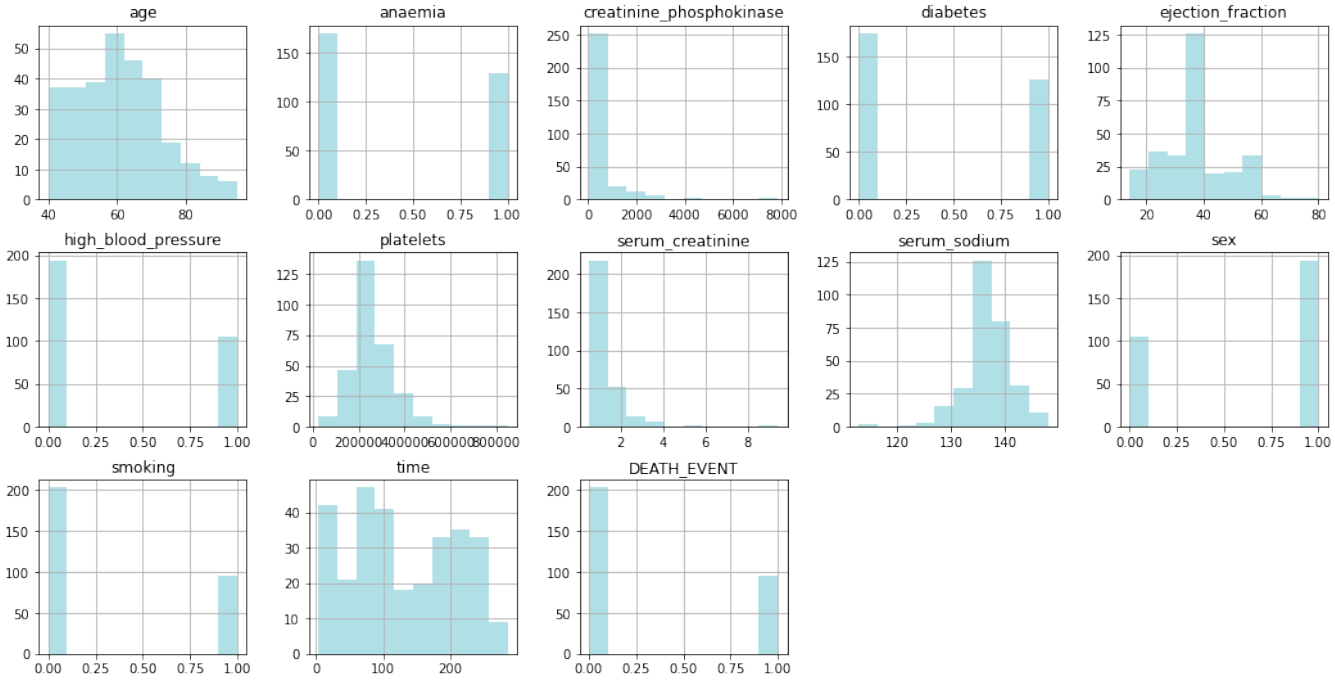


Figure 14: *hist*() function call that outputs a histogram per attribute in the data set.

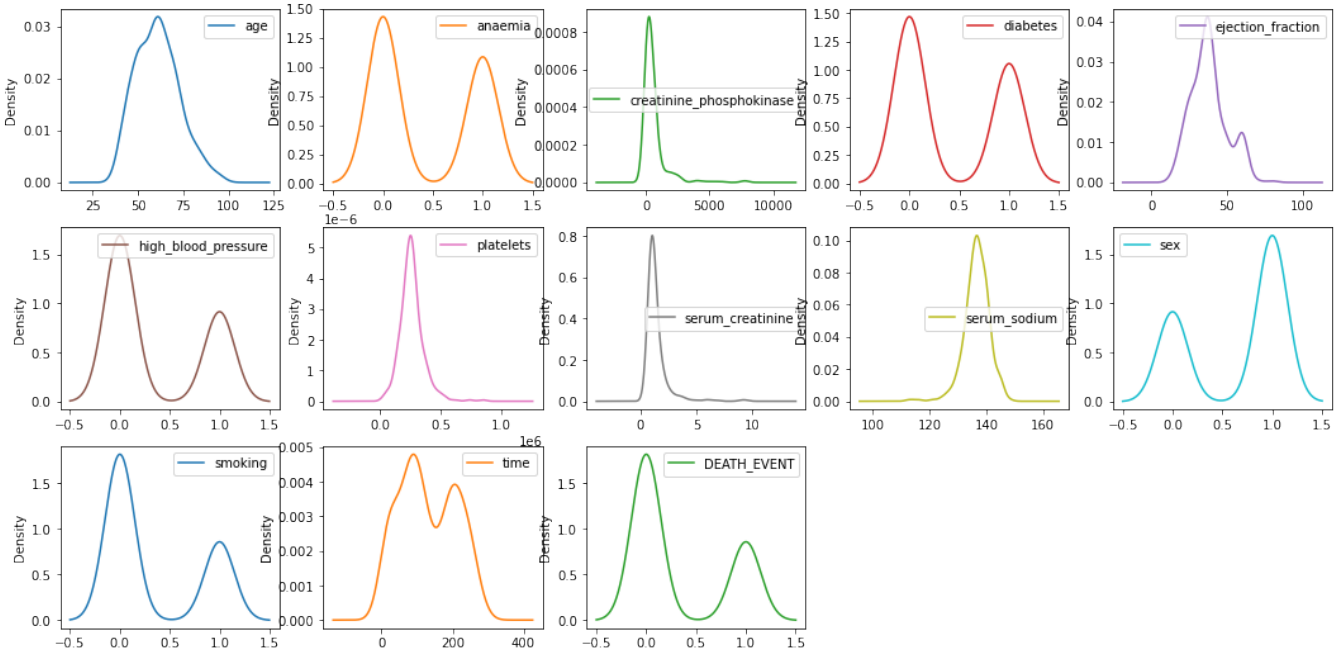


Figure 15: *plot*() function call that outputs a density plot per attribute in the data set.

```
fig = plt.figure(figsize=(18,10))
ax = fig.gca()
heart.plot(ax=ax, kind='box', subplots=True, layout=(3,5))
plt.show()
```

In which the output lies in Figure 16.

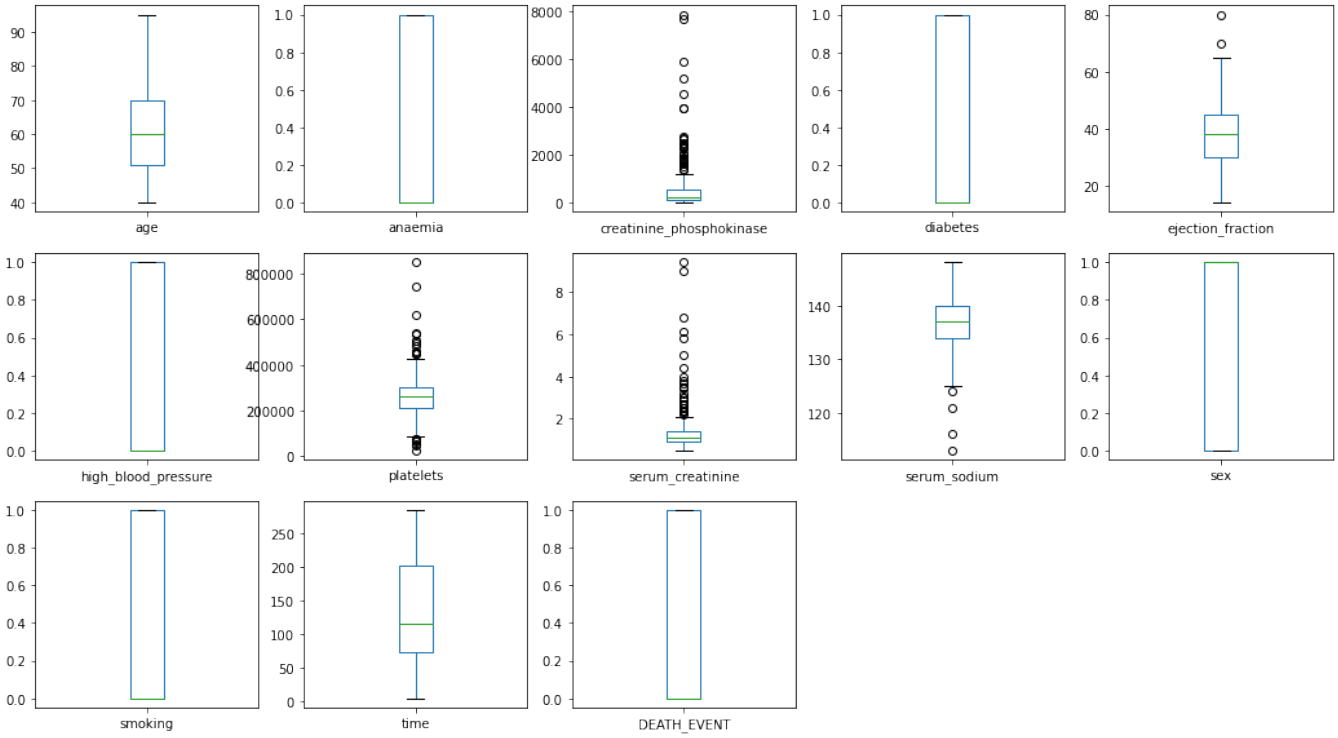


Figure 16: *plot*() function call that outputs a box plot per attribute in the data set.

3.3 Conclusions On Data Set Analysis

This data set has probably the highest usability due to its compactness: all attributes contain numeric data, there seem to be no missing values and the measure units are very clear. The dimensions are small in both rows and columns, which allow a much easier visual analysis of the data as a whole. The data set seems to have binary/boolean values in most of the attributes, which is helpful because it makes the attributes more comparable. The quartile calculations in the *describe* function are more visible in the box plots: the median of all binary attributes is zero and the upper quartile will always be 1, which makes the box plot in these columns look as a simple rectangle. Therefore, box plots may not be the best data visualization tool to show what a binary data set like this has to offer. Also, the distributions shown from the histograms and density functions seem not to explain much when the attribute is binary.

References

[1] Davide Chicco. *Heart Failure Clinical Records Data Set*. URL: `https://www.kaggle.com/andrewmvd/heart-failure-clinical-data`.

[2] UCI Machine Learning. *Breast Cancer Wisconsin (Diagnostic) Data Set*. URL: `https://www.kaggle.com/uciml/breast-cancer-wisconsin-data`.

[3] Kaggle (Unknown Source). *Alzheimer Features*. URL: `https://www.kaggle.com/brsdincer/alzheimer-features`.