Data Preprocessing Report

Author: Rodrigo Santos de Carvalho - Published 2021-11-25 09:32 - (0 Reads)

Steps in data preprocessing

Machine learning algorithms depend heavily on the quality of the given data. That's why it's required that the data is suitable for a machine learning model, and since it's common that the data is not always correct (improper data or input errors), preprocessing, the act of preparing the raw data to be used in a machine learning algorithms, is necessary. This report covers the steps in data preprocessing, as well as some examples.

Course Example

The example I studied was the preparation of a credit database, which would be used in a machine learning algorithm that will determine the credit risk of someone based on their data.

Data Exploration

Since the database was already given by the course, the first step was importing the libraries that would be used to prepare the data. Those libraries are pandas, numpy, seaborn, matplotlib and plotly.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

The second step was exploring the data, identifying and handling the missing values.

code:

```
base_credit = pd.read_csv('/content/credit_data.csv')
print(base_credit)
```

output:

	clientd	income	age	loan	defaut
0	1	66155.92510	59.017015	8106.532131	0
1	2	34415.15397	48.117153	6564.745018	0
2	3	57317.17006	63.108049	8020.953296	0
3	4	42709.53420	45.751972	6103.642260	0
4	5	66952.68885	18.584336	8770.099235	1
...	...	...	...	...	...
1995	1996	59221.04487	48.518179	1926.729397	0
1996	1997	69516.12757	23.162104	3503.176156	0
1997	1998	44311.44926	28.017167	5522.786693	1
1998	1999	43756.05660	63.971796	1622.722598	0
1999	2000	69436.57955	56.152617	7378.833599	0

With that command, the entire database was shown. Due to the large amount of data, in order to analyze it properly, the describe() method was used. That method gives us useful and important information about the database.

code:

```
base_credit.describe()
```

output:

	clientid	income	age	loan	default
count	2000.000000	2000.000000	1997.000000	2000.000000	2000.000000
mean	1000.500000	45331.600018	40.807559	4444.369695	0.141500
std	577.494589	14326.327119	13.624469	3045.410024	0.348624
min	1.000000	20014.489470	-52.423280	1.377630	0.000000
25%	500.750000	32796.459720	28.990415	1939.708847	0.000000
50%	1000.500000	45789.117310	41.317159	3974.719418	0.000000
75%	1500.250000	57791.281670	52.587040	6432.410625	0.000000
max	2000.000000	69995.685580	63.971796	13766.051240	1.000000

Data Visualization

After exploring the database, it's time to visualize it. That may help in finding missing and inconsistent values. The following command returns the number of those who pay their loans and those who do not:

code:

```
np.unique(base_credit['default'], return_counts=True)
```

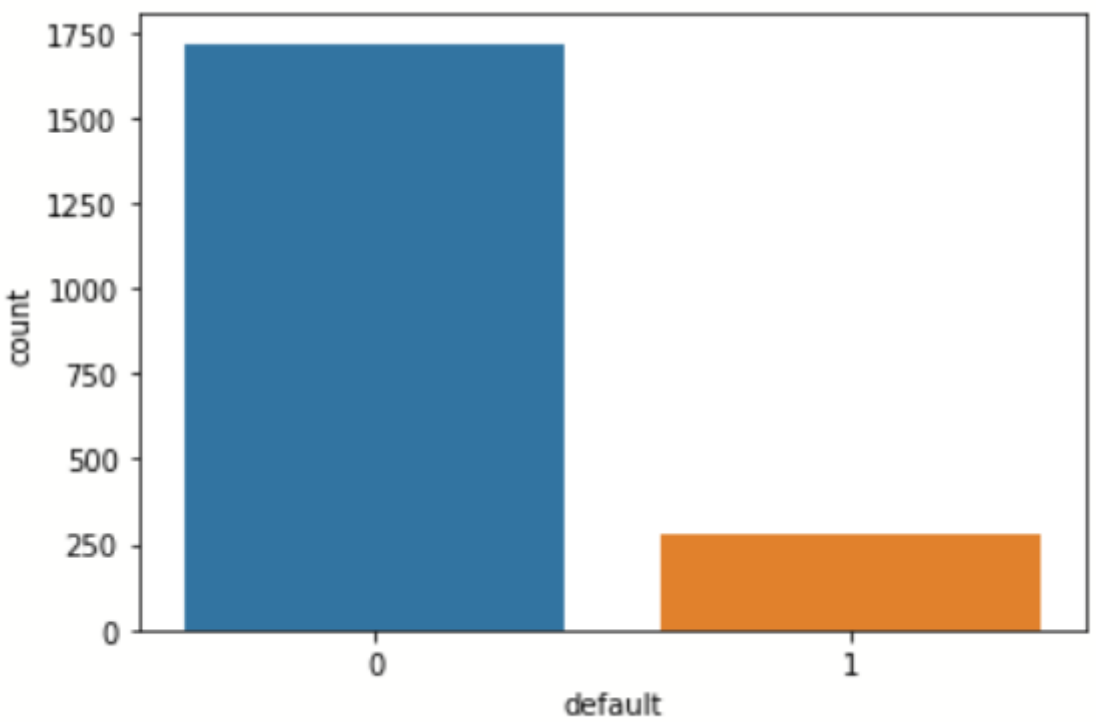
```
(array([0, 1]), array([1717,  283]))
```

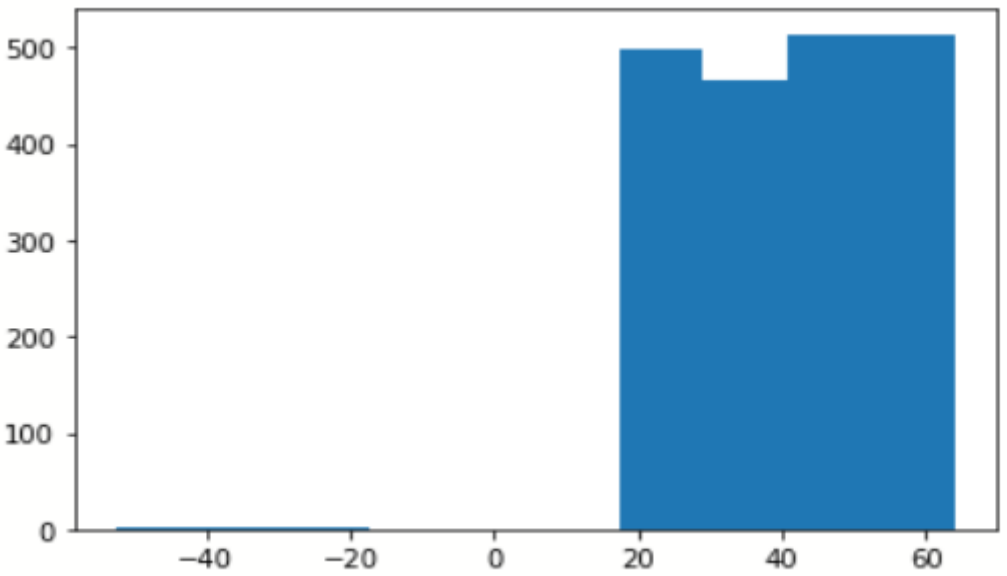
Therefore, 1717 people pay their loans and 283 don't. And to help with the analysis of the data, some graphs were generated with seaborn, matplotlib and plotly.

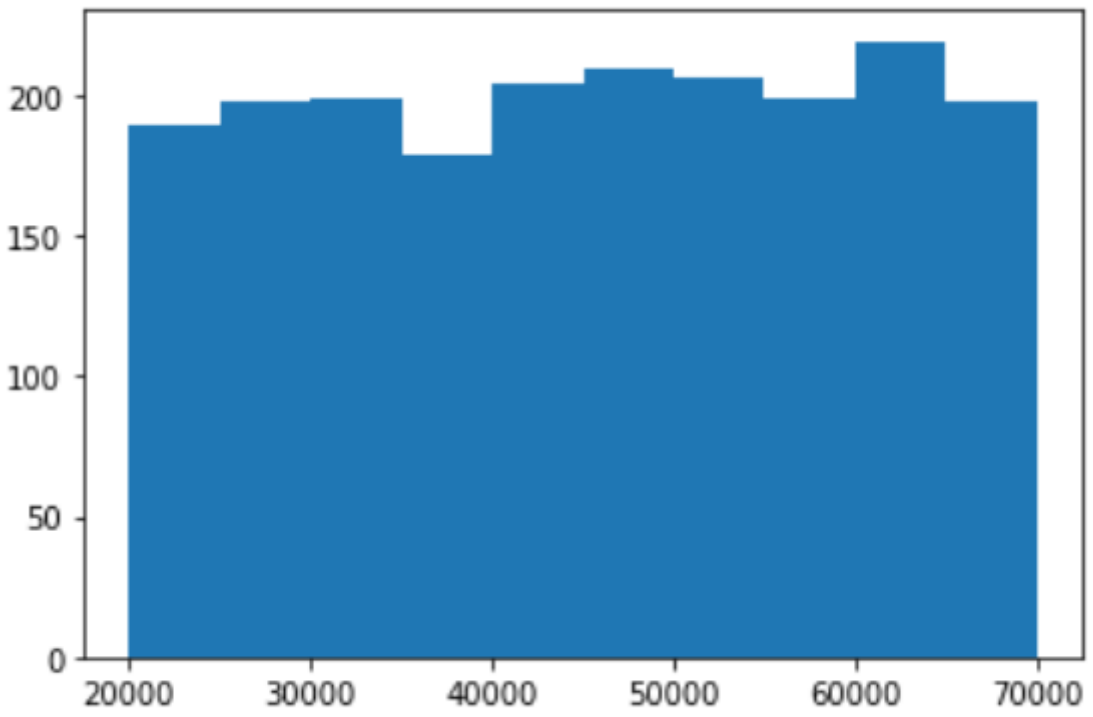
code:

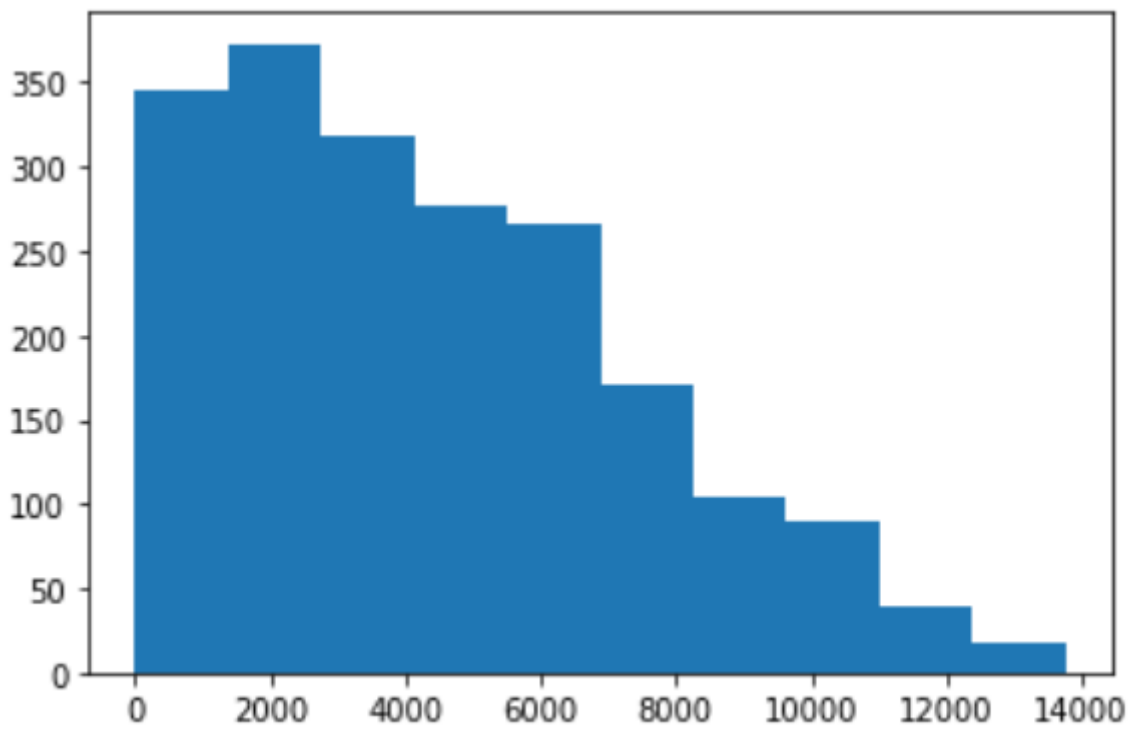
```
sns.countplot(x = base_credit['default']);
plt.hist(x = base_credit['age']);
plt.hist(x = base_credit['income']);
plt.hist(x = base_credit['loan']);
grafico = px.scatter_matrix(base_credit, dimensions=['age', 'income', 'loan'], color = 'default')
grafico.show()
```

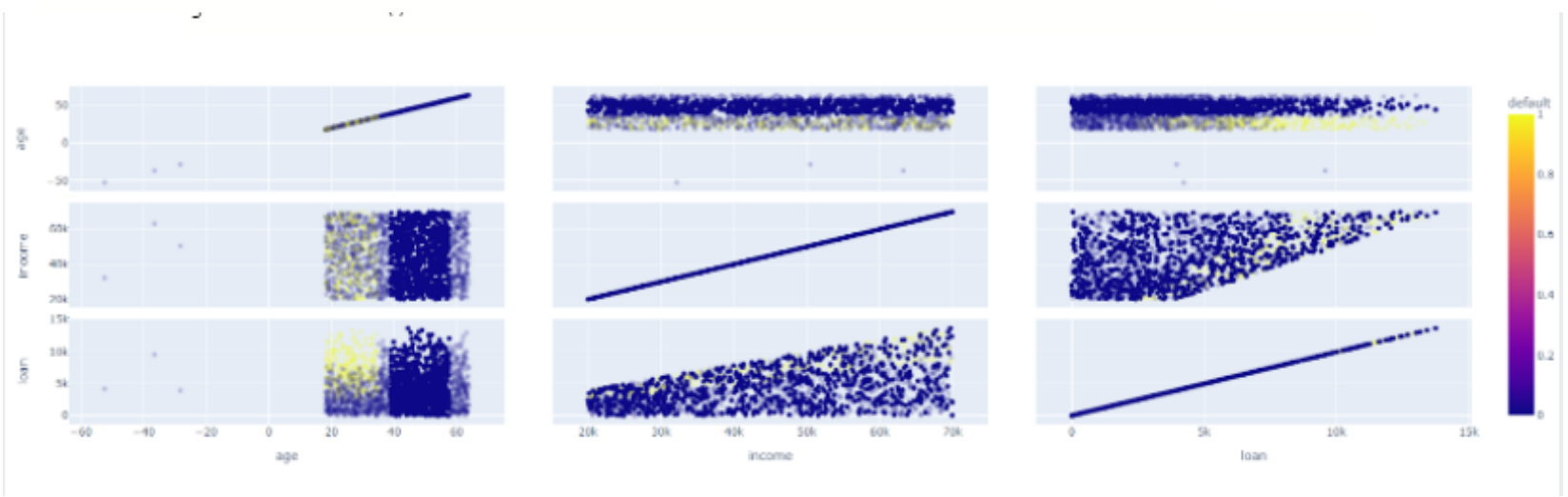
output:











Those graphs show that there are inconsistent values in the database, as a result of the presence of negative values for age in the database. The inconsistent data must be located and treated.

To locate the inconsistent data, the following command is executed:

code:

```
print(base_credit[base_credit['age'] < 0])
```

output:

	clientid	income	age	loan	default
15	16	50501.726689	-28.218361	3977.287432	0
21	22	32197.620701	-52.423280	4244.057136	0
26	27	63287.038908	-36.496976	9595.286289	0

The clients 15, 21 and 26 must have their ages altered. Their ages can be deleted through the drop() function.

code:

```
base_credit3 = base_credit.drop(base_credit[base_credit['age'] < 0].index)
```

After deleting the inconsistent data, the database should be verified again, to make sure that there aren't any inconsistent data left.

code:

```
print(base_credit3.loc[base_credit3['age'] < 0])
```

output:

	clientid	income	age	loan	default

Or, instead of deleting it and assigning a new value, a new value can be assigned to the negative ages without deleting the previous values. The negative values will be replaced by the mean value of the ages in the database (not considering the negative values).

code:

```
base_credit['age'][base_credit['age'] > 0].mean()
```

output:
40.92770044906149

The mean value is approximately 40.92, and it'll replace all the negative age values.

code:

```
base_credit.loc[base_credit['age'] < 0, 'age'] = 40.92
```

Missing Data

Data may not only be inconsistent, but also missing, and that must be treated as well. The process of treating it is similar to how the negative values were treated. In the first place, the missing data must be located (if there are any missing data).
The function isnull() goes through the database and shows which values are missing.

code:
{CODE()
print(base_credit.isnull())
{CODE}

output:

	clientid	income	age	loan	default
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
1995	False	False	False	False	False
1996	False	False	False	False	False
1997	False	False	False	False	False
1998	False	False	False	False	False
1999	False	False	False	False	False

The isnull().sum() command returns the quantity of missing values.

code:

```
base_credit.isnull().sum()
```

output:

```
clientid    0
income      0
age         3
loan        0
default     0
dtype: int64
```

Three missing values were found, hence a value has to be assigned in their place.

code:

```
base_credit.loc[pd.isnull(base_credit['age'])]
```

output:

	clientid	income	age	loan	default
28	29	59417.805406	NaN	2082.625938	0
30	31	48528.852796	NaN	6155.784670	0
31	32	23526.302555	NaN	2862.010139	0

The clients 29, 31 and 32 have their ages missing. They'll also be replaced by the mean value for all the ages in the database (40.92).

code:

```
base_credit['age'].fillna(base_credit['age'].mean(), inplace = True)
```

And to be sure that there are no more missing values, the clients that had their ages missing can be checked again.

code:

```
print(base_credit.loc[base_credit['clientid'].isin([29, 31, 32])])
```

output:

	clientid	income	age	loan	default
28	29	59417.805406	NaN	2082.625938	0
30	31	48528.852796	NaN	6155.784670	0
31	32	23526.302555	NaN	2862.010139	0

Predictors and Classes

The data must also be divided in two groups, predictors (X) and class (Y). The former is key in the quality of the prediction made by the machine learning algorithm, whereas the latter isn't. For instance, income, age, loan are predictors, and the default is a class.

code:

```
X_credit = base_credit.iloc[:, 1:4].values
print(X_credit)
```

output:

```
array([[6.61559251e+04, 5.90170151e+01, 8.10653213e+03],
       [3.44151540e+04, 4.81171531e+01, 6.56474502e+03],
       [5.73171701e+04, 6.31080495e+01, 8.02095330e+03],
       ...,
       [4.43114493e+04, 2.80171669e+01, 5.52278669e+03],
       [4.37560566e+04, 6.39717958e+01, 1.62272260e+03],
       [6.94365796e+04, 5.61526170e+01, 7.37883360e+03]])
```

code:

```
Y_credit = base_credit.iloc[:, 4].values
print(Y_credit)
```

output:

```
array([0, 0, 0, ..., 1, 0, 0])
```

Feature Scaling

The range of the data can vary widely, which, in some machine learning algorithms, imply that a feature with a higher value range starts dominating over features with a lower value range. Hence, the features must be scaled, so that they can be analyzed by the algorithm on equal footing.

There are two ways to do that scaling: standardization and normalization (Min-Max scaling):

Standardization:

$$z = \frac{x - \mu}{\sigma}$$

with mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

and standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

Min-Max scaling:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Standardization is advised when there are outliers, whereas normalization is advised when there aren't. Therefore, since there were outliers (negative values for age) in the credit database, standardization will be used.

code:

```
from sklearn.preprocessing import StandardScaler
scaler_credit = StandardScaler()
X_credit = scaler_credit.fit_transform(X_credit)
print(X_credit)
```

output:

```
array([[ 1.45393393,  1.36538093,  1.20281942],
       [-0.76217555,  0.5426602 ,  0.69642695],
       [ 0.83682073,  1.67417189,  1.17471147],
       ...,
       [-0.07122592, -0.97448519,  0.35420081],
       [-0.11000289,  1.73936739, -0.92675625],
       [ 1.682986  ,  1.14917639,  0.96381038]])
```