Deepika Singh

Cleaning up Data from Outliers

Deepika Singh

Oct 22, 2019 • 21 Min read • 24,206 Views

Oct 22, 2019 • 21 Min read • 24,206 Views

Data Python

Introduction

Introduction

The difference between a good and an average machine learning model is often its ability to clean data. One of the biggest challenges in data cleaning is the identification and treatment of outliers. In simple terms, outliers are observations that are significantly different from other data points. Even the best machine learning algorithms will underperform if outliers are not cleaned from the data because outliers can adversely affect the training process of a machine learning algorithm, resulting in a loss of

In this guide, you will learn about techniques for outlier identification and treatment in Python.

Data

In this guide, we will be using a fictitious dataset of loan applications containing 600 observations and 6 variables:

1. Income - Annual income of the applicant (in US dollars)

2. Loan_amount - Loan amount (in US dollars) for which the application was submitted

3. Term_months - Tenure of the loan (in months)

4. Credit_score - Whether the applicant's credit score was good ("1") or not ("0")

5. Age - The applicant's age in years

6. Approval_status - Whether the loan application was approved ("1") or not ("0")

Let's start by loading the required libraries and the data.

```
import matplotlib.pyplot as plt

# Reading the data
df = pd.read_csv("data_out.csv")
print(df.shape)
print(df.info())
```

Output:

```
(600, 6)
<class 'pandas.core.frame.DataF
RangeIndex: 600 entries, 0 to 5
Data columns (total 6 columns):
Income              600 non-null
Loan_amount         600 non-null
Term_months         600 non-null
Credit_score        600 non-null
approval_status     600 non-null
Age                 600 non-null
dtypes: int64(6)
memory usage: 28.2 KB
None
```

The above output shows that there are 600 observations of 6 variables. All the variables have 600 records, indicating that there is no missing value in the data.

Outlier Identification

There can be many reasons for the presence of outliers in the data. Sometimes the outliers may be genuine, important to understand the reasons for the outliers before cleaning them.

We will start the process of finding outliers by running the summary statistics on the variables. This is done using the ***describe()*** function below, which provides a statistical summary of all the quantitative variables.

python

```
df.describe()
```

Output:

```
ount    | Term_months    | Credit_score   | appr
-----   |------------    |-------------   |-----
000     | 600.00000      | 600.000000     | 600.
667     | 367.10000      | 0.788333       | 0.68
98      | 63.40892       | 0.408831       | 0.46
00      | 36.00000       | 0.000000       | 0.00
000     | 384.00000      | 1.000000       | 0.00
000     | 384.00000      | 1.000000       | 1.00
000     | 384.00000      | 1.000000       | 1.00
000     | 504.00000      | 1.000000       | 1.00
```

Looking at the 'Age' variable, it is easy to detect outliers resulting from incorrect data. The minimum and maximum ages are 0, and 200, respectively. These are incorrect, and we will treat them later in the guide. These outliers were easy to detect, butthat will not always be the case. In other cases, techniques are discussed in the following sections.

Identifying Outliers with Interquartile Range (IQR)

The interquartile range (IQR) is a measure of statistical dispersion and is calculated as the difference between the 75th and 25th percentiles. It is represented by the formula ***IQR = Q3 − Q1***. The lines of code below calculate and print the interquartile range for each of the variables in the dataset.

python
```python
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

Output:

```
Income              3809.0
Loan_amount           69.5
Term_months            0.0
Credit_score           0.0
approval_status        1.0
Age                   28.0
dtype: float64
```

The above output prints the IQR scores, which can be used to detect outliers. The code below generates an output with the 'True' and 'False' values. Points where the values are 'True' represent python

```python
print(df < (Q1 - 1.5 * IQR)) |(df >
```

Output:

```
            Income  Loan_amount  Term_
      0     False         False
      1     False         False
      2     False         False
      3     False         False
      4     False         False
      5     False         False
      6     False         False
      7     False         False
      8     False         False
      9     False         False
      10    False         False
      11    False         False
      12    False         False
      13    False         False
      14    False         False
      15    False         False
      16    False         False
      17    False         False
      18    False         False
      19    False         False
      20    False         False
      21    False         False
      22    False         False
      23    False         False
      24    False         False
      25    False         False
      26    False         False
      27    False         False
      28    False         False
      29    False         False
      ..      ...           ...
      570   False         False
      571   False         False
      572   False         False
      573   False         False
      574   False         False
      575   False         False
```

```
           579   False         False
           580   False         False
           581   False         False
           582   False         False
           583   False         False
           584   False         False
           585   False         False
           586   False         False
           587   False         False
           588   False         False
           589   False         False
           590   False         False
           591   False         False
           592   False         False
           593   False         False
           594   False         False
           595   False         False
           596   False         False
           597   False         False
           598   False         False
           599   False         False

           [600 rows x 6 columns]
```

Identifying Outliers with Skewness

Several machine learning algorithms make the assumption that the data follow a normal (or Gaussian) distribution. This is easy to check with the *skewness value*, which explains the extent to which the data is normally distributed. Ideally, the skewness value should be between -1 and +1, and any major deviation from this range indicates the presence of extreme values.

The first line of code below prints the

while the second line prints the summary statistics.

```python
print(df['Income'].skew())
df['Income'].describe()
```

Output:

```
6.499

count       600.000000
mean       7210.720000
std        8224.445086
min         200.000000
25%        3832.500000
50%        5075.000000
75%        7641.500000
max      108000.000000
Name: Income, dtype: float64
```

The skewness value of 6.5 shows that the variable 'Income' has a right-skewed distribution, indicating the presence of extreme higher values. The maximum 'Income' value of USD 108,000 proves this point.

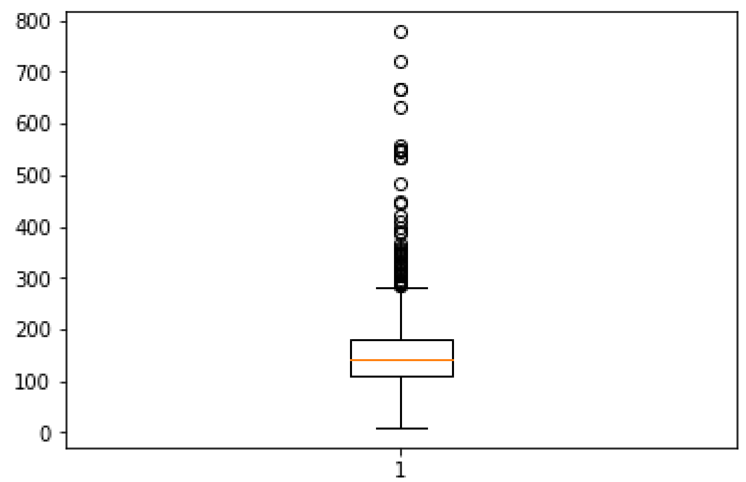
Identifying Outliers with Visualization

In the previous section, we used quantitative methods for outlier identification. This can also be achieved with visualization. Some of the common plots used for outlier detection are

The box plot is a standardized way of displaying the distribution of data based on the five-number summary (minimum, first quartile (Q1), median, third quartile (Q3), and maximum). It is often used to identify data distribution and detect outliers. The line of code below plots the box plot of the numeric variable 'Loan_amount'.

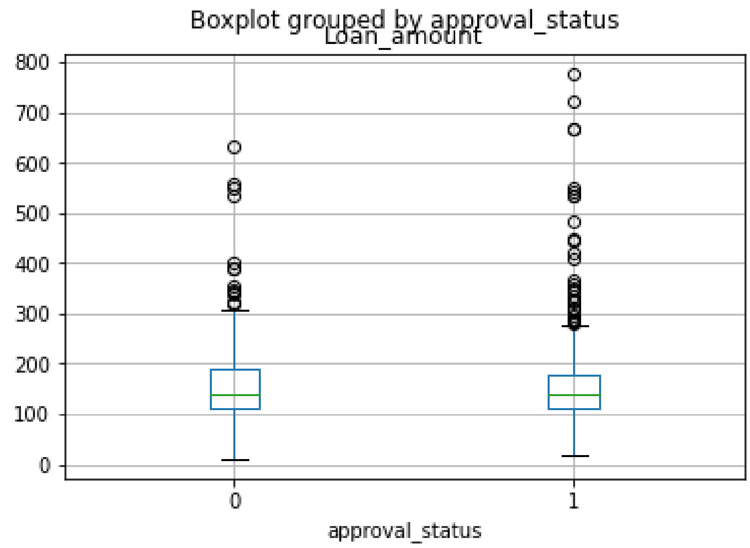
python
```python
plt.boxplot(df["Loan_amount"])
plt.show()
```

Output:



In the above output, the circles indicate the outliers, and there are many. It is also possible to identify outliers using more than one variable. We can modify the above code to visualize outliers in the 'Loan_amount' variable by the approval status.

Output:



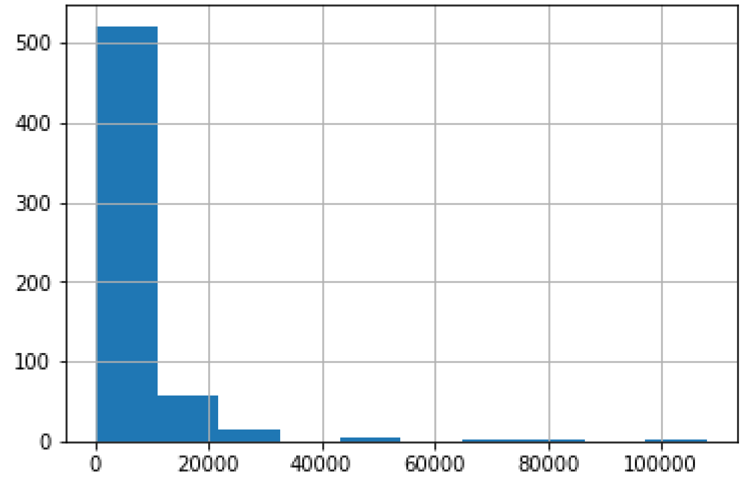
The output shows that the number of outliers is higher for approved loan applicants (denoted by the label '1') than for rejected applicants (denoted by the label '0').

2. Histogram

A histogram is used to visualize the distribution of a numerical variable. An outlier will appear outside the overall pattern of distribution. The line of code below plots a histogram of the 'Income' variable, using the ***hist()*** function.

python
```
df.Income.hist()
```

Output:



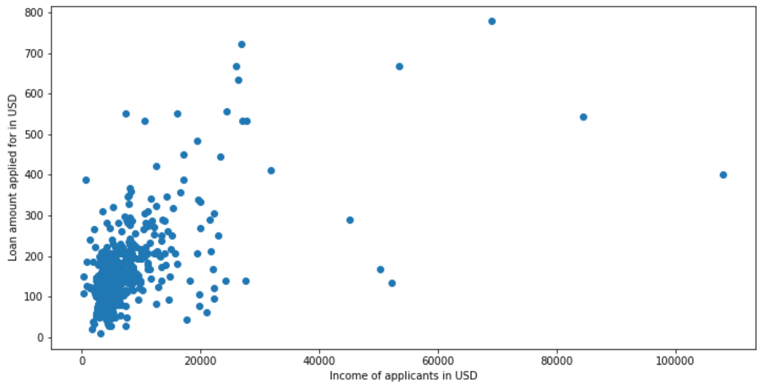
The above chart shows that the distribution is right-skewed, and there are extreme higher values at the right of the histogram. This step can be repeated for other variables as well.

3. Scatterplot

A scatterplot visualizes the relationship between two quantitative variables. The data are displayed as a collection of points, and any points that fall outside the general clustering of the two variables may indicate outliers. The lines of code below generate a scatterplot between the variables 'Income' and 'Loan_amount'.

python

```
fig, ax = plt.subplots(figsize=(12,6
ax.scatter(df['Income'], df['Loan_am
ax.set_xlabel('Income of applicants 
ax.set_ylabel('Loan amount applied f
plt.show()
```



The above chart indicates that most of the data points are clustered in the lower half of the plot. The points located to the extreme right of the x-axis or the y-axis indicate outliers.

Outlier Treatment

In the previous sections, we learned about techniques for outlier detection. However, this is only half of the task. Once we have identified the outliers, we need to treat them. There are several techniques for this, and we will discuss the most widely used ones below.

Quantile-based Flooring and Capping

In this technique, we will do the flooring (e.g., the 10th percentile) for the lower values and capping (e.g., the 90th percentile) for the higher values. The lines of code below print the 10th and 90th percentiles of the variable be used for quantile-based flooring and capping.

python

```
print(df['Income'].quantile(0.10))
print(df['Income'].quantile(0.90))
```

Output:

```
2960.1
12681.0
```

Now we will remove the outliers, as shown in the lines of code below. Finally, we calculate the skewness value again, which comes out much better now.

python

```
df["Income"] = np.where(df["Income"]
df["Income"] = np.where(df["Income"]
print(df['Income'].skew())
```

Output:

```
1.04
```

Trimming

In this method, we completely remove data points that are outliers. Consider the 'Age' variable, which had a minimum value of 0 and a maximum value of 200.

line drops these index rows from the data, while the third line of code prints summary statistics for the variable.

After trimming, the number of observations is reduced from 600 to 594, and the minimum and maximum values are much more acceptable.

python
```
index = df[(df['Age'] >= 100)|(df['A
df.drop(index, inplace=True)
df['Age'].describe()
```

Output:

```
count    594.000000
mean      50.606061
std       16.266324
min       22.000000
25%       36.000000
50%       50.500000
75%       64.000000
max       80.000000
Name: Age, dtype: float64
```

IQR Score

This technique uses the IQR scores calculated earlier to remove outliers. The rule of thumb is that anything not in the range of ***(Q1 - 1.5* IQR) and (Q3 + 1.5 *IQR)*** is an outlier, and can be removed. The first line of code below removes outliers based on the IQR range and shape of this data, which comes out to be 375 observations of 6 variables. This shows that for our data, a lot of records get deleted if we use the IQR method.

python
```python
df_out = df[~((df < (Q1 - 1.5 * IQR)
print(df_out.shape)
```

Output:

```
(375, 6)
```

Log Transformation

Transformation of the skewed variables may also help correct the distribution of the variables. These could be logarithmic, square root, or square transformations. The most common is the logarithmic transformation, which is done on the 'Loan_amount' variable in the first line of code below. The second and third lines of code print the skewness value before and after the transformation.

python
```python
df["Log_Loanamt"] = df["Loan_amount"
print(df['Loan_amount'].skew())
print(df['Log_Loanamt'].skew())
```

Output:

```
2.8146019248106815
-0.17792641310111373
```

The above output shows that the skewness value came down from 2.8 to -0.18, confirming that the distribution has been treated for extreme values.

Replacing Outliers with Median Values

In this technique, we replace the extreme values with median values. It is advised to not use mean values as they are affected by outliers. The first line of code below prints the 50th percentile value, or the median, which comes out to be 140. The second line prints the 95th percentile value, which comes out to be around 326. The third line of code below replaces all those values in the 'Loan_amount' variable, which are greater than the 95th percentile, with the median value. Finally, the fourth line prints summary statistics after all these techniques have been employed for outlier treatment.

python

```
1 print(df['Loan_amount'].quantile(0.5
2 print(df['Loan_amount'].quantile(0.9
3 df['Loan_amount'] = np.where(df['Loa
4 df.describe()
```

Output:

```
        140.0
        325.7500000000001


|           | Income        | Loan_
|-------    |-------------- |------
| count     | 594.000000    | 594.0
| mean      | 6112.375421   | 144.2
| std       | 3044.257269   | 53.03
| min       | 2960.000000   | 10.00
| 25%       | 3831.500000   | 111.0
| 50%       | 5050.000000   | 140.0
| 75%       | 7629.000000   | 171.0
| max       | 12681.000000  | 324.0
```

Conclusion

In this guide, you have learned methods of identifying outliersusing bothquantitative and visualization techniques. You have also learnedtechniques for treating the identified outliers. Your usage of these techniques will depend on the data, the problem statement, and the machine learning algorithm selected for building the model.

To learn more about data preparation and building machine learning models using Python, please refer to the

2. Linear, Lasso, and Ridge Regression with scikit-learn
3. Non-Linear Regression Trees with scikit-learn
4. Machine Learning with Neural Networks Using scikit-learn
5. Validating Machine Learning Models with scikit-learn
6. Ensemble Modeling with scikit-learn
7. Preparing Data for Modeling with scikit-learn
8. Interpreting Data Using Descriptive Statistics with Python

To learn more about building deep learning models using **Keras**, please refer to the following guides:

1. Regression with Keras
2. Classification with Keras

WEEKEND)

SOLUTIONS

Pluralsight Skills (/product/skills)