Data Glacier

Your Deep Learning Partner

Week 9 deliverables

Bank Marketing (Campaign) - Group Project

Group Name - Bloodhounds,
Batch code - LISUM09,
Specialization: Data science.
Group member details:

- Name - Margarita Prokhorovich,
- email - marusya15071240@gmail.com,
- Country – Thailand,
- Submission date – 2 July, 2022

Problem description

Statement

- ABC Bank wants to sell it's term deposit product to customers and before launching the product they want to develop a model which help them in understanding whether a particular customer will buy their product or not (based on customer's past interaction with bank or other Financial Institution).
- Bank wants to use ML model to shortlist customer whose chances of buying the product is more so that their marketing channel (tele marketing, SMS/email marketing etc) can focus only to those customers whose chances of buying the product is more. This will save resource and their time (which is directly involved in the cost (resource billing))[1].

Data set problem statement

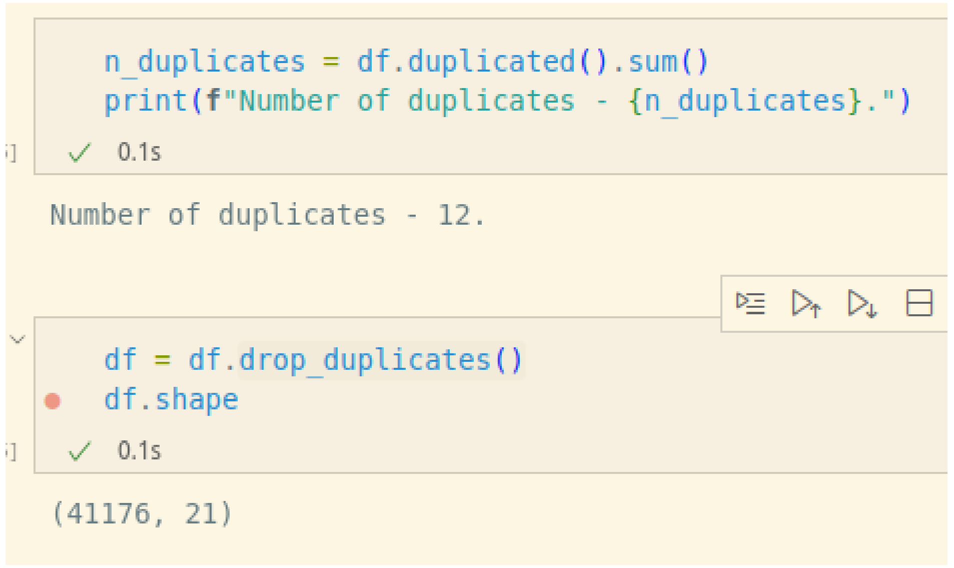
- A big part of customers are convinced of the effectiveness of an individual approach to service. In an Accenture Financial Services global study of nearly 33,000 banking customers spanning 18 markets, 49% of respondents indicated that customer service drives loyalty. By knowing the customer and engaging with them accordingly, financial institutions can optimize interactions that result in increased customer satisfaction and wallet share, and a subsequent decrease in customer churn[2].
- One of the challenges the banks encounter, is following:
  How does a bank figure out what its customers specifically think about its services? Are their issues getting resolved? How satisfied are they with the experience? Why can't banks analyze customer care sessions to find real-time information about the customers and their pressing issues? Customer care records are very pointed and specific about the challenges the customer faces. In these cases building a model and detect patterns can help to improve customers' retaining and loyalty and reduce the churn[3].

1. Problem Statement. Data Science:: Bank Marketing (Campaign) -- Group Project. Data Glacier, URL
2. Top 10 Banking Industry Challenges — And How You Can Overcome Them. Hitachi Solutions, URL
3. Why Retaining Customers For Banks Is As Important As Winning New Ones. Forbes, URL

Data cleansing and transformation

These data transformation steps were presented in a previous week report. For these issues in the data I use only one technique because

- drop duplicates is the most convenient way to handle duplicates;
- I don't see any options to work with numeric pdays feature and decide to move to categorical one.



```
n_duplicates = df.duplicated().sum()
print(f"Number of duplicates - {n_duplicates}.")
```
✓ 0.1s

```
Number of duplicates - 12.
```

```
df = df.drop_duplicates()
df.shape
```
✓ 0.1s

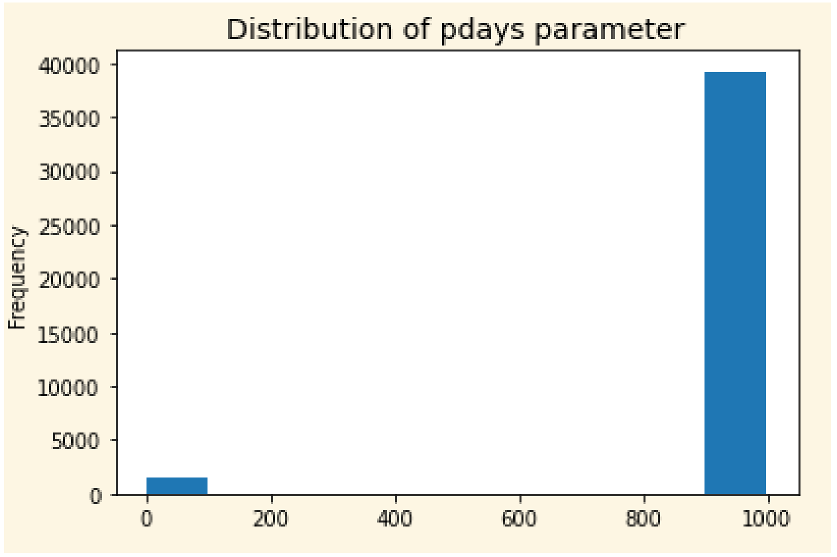
```
(41176, 21)
```

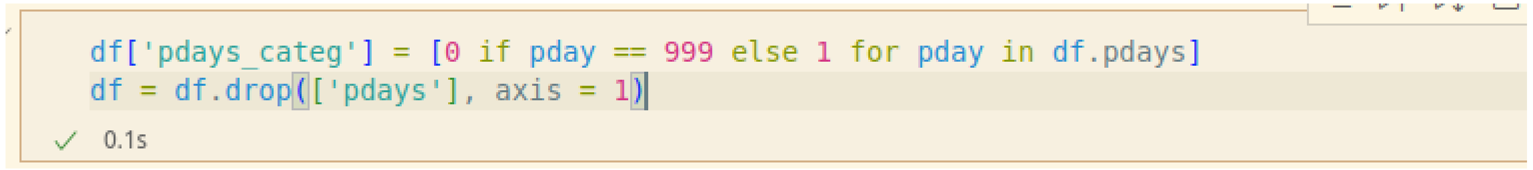
Duplicate values

- Since we have duplicates in our data set, we need to delete them.
- We have 12 duplicates and remove them using drop_duplicates() method. This method deletes complete duplicates from the data set.

Pdays parameter values

Pdays parameter has a lot of 999 values. 999 means client was not previously contacted. Since the variable is numeric, it can affect the interpretation of the model. Replacing 999s with 0 is also not effective since interpretation can be wrong - 0 days passed by after the client was last contacted from a previous campaign. Therefore it was decided to move from numeric variable to a binary one. It's also reasonable because except for 999s, another values lie in not large range.





```
df['pdays_categ'] = [0 if pday == 999 else 1 for pday in df.pdays]
df = df.drop(['pdays'], axis = 1)
```
✓ 0.1s

Data cleansing and transformation

‘Unknown’ values

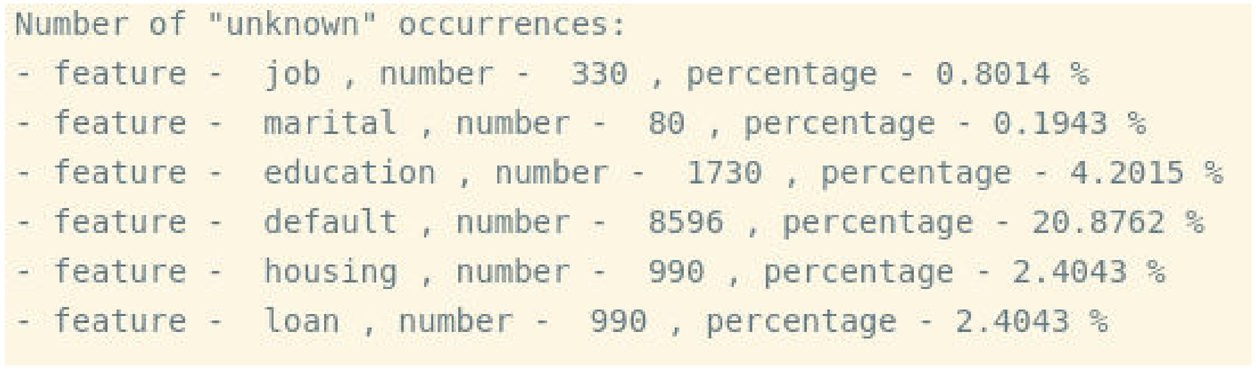
Options for handling

- Data set has no null values but some categorical features have values marked ‘unknown’. Following options can be considered:
- Delete all the rows with ‘unknown’ values or delete them only in certain columns
- Populate ‘unknown’ values with a major category.

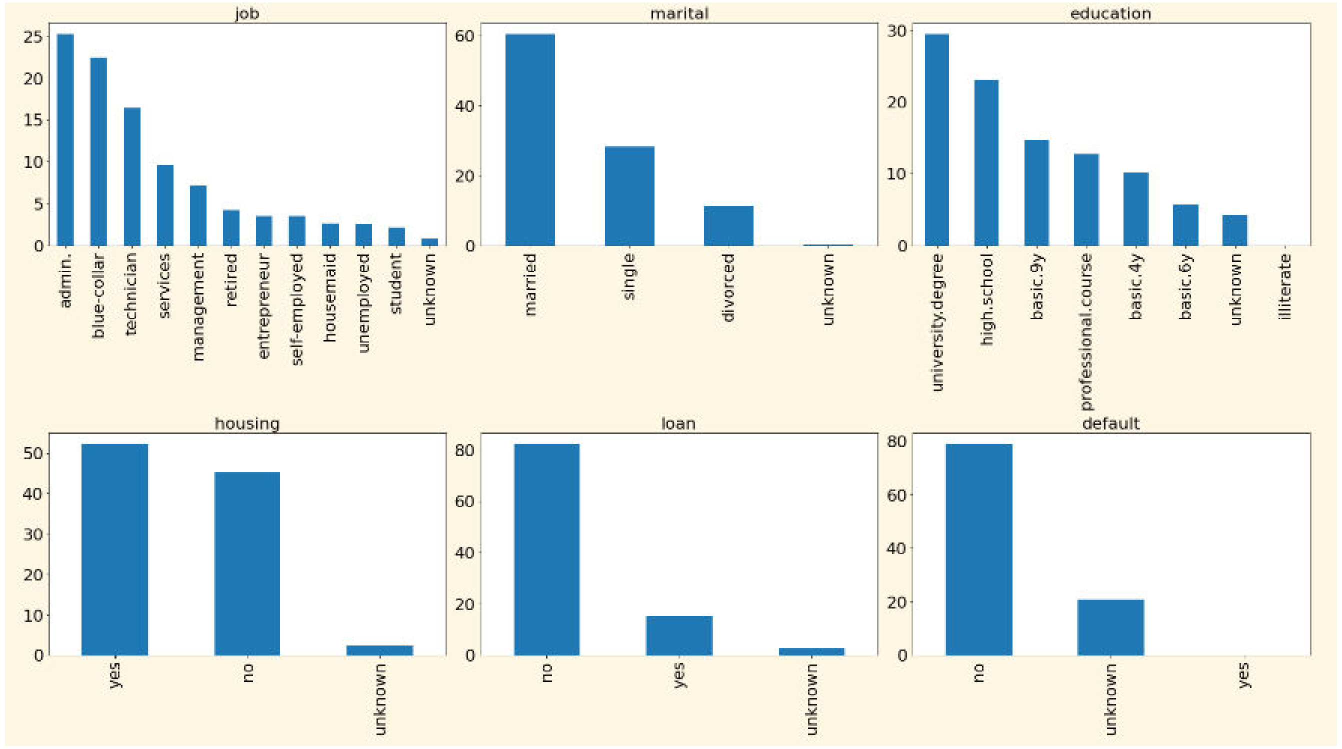
The first approach – fill unknown values with values of a major class

The suggested approach is to populate the variables with the most common category. The exception is the default variable, since the proportion of unknowns is quite large, with a high probability we will consider ‘unknown’ as a separate category.

The plots on the right show distribution of categorical features before the processing.



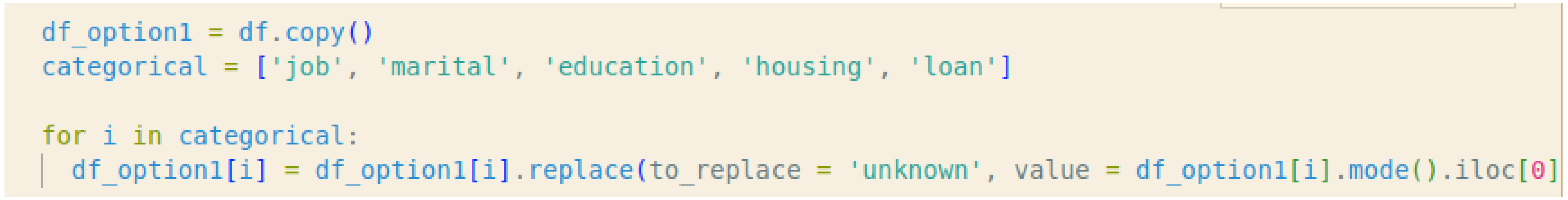
```
Number of "unknown" occurrences:
- feature -  job , number -  330 , percentage - 0.8014 %
- feature -  marital , number -  80 , percentage - 0.1943 %
- feature -  education , number -  1730 , percentage - 4.2015 %
- feature -  default , number -  8596 , percentage - 20.8762 %
- feature -  housing , number -  990 , percentage - 2.4043 %
- feature -  loan , number -  990 , percentage - 2.4043 %
```



Data cleansing and transformation

‘Unknown’ values

The first approach – fill unknown values with values of a major class

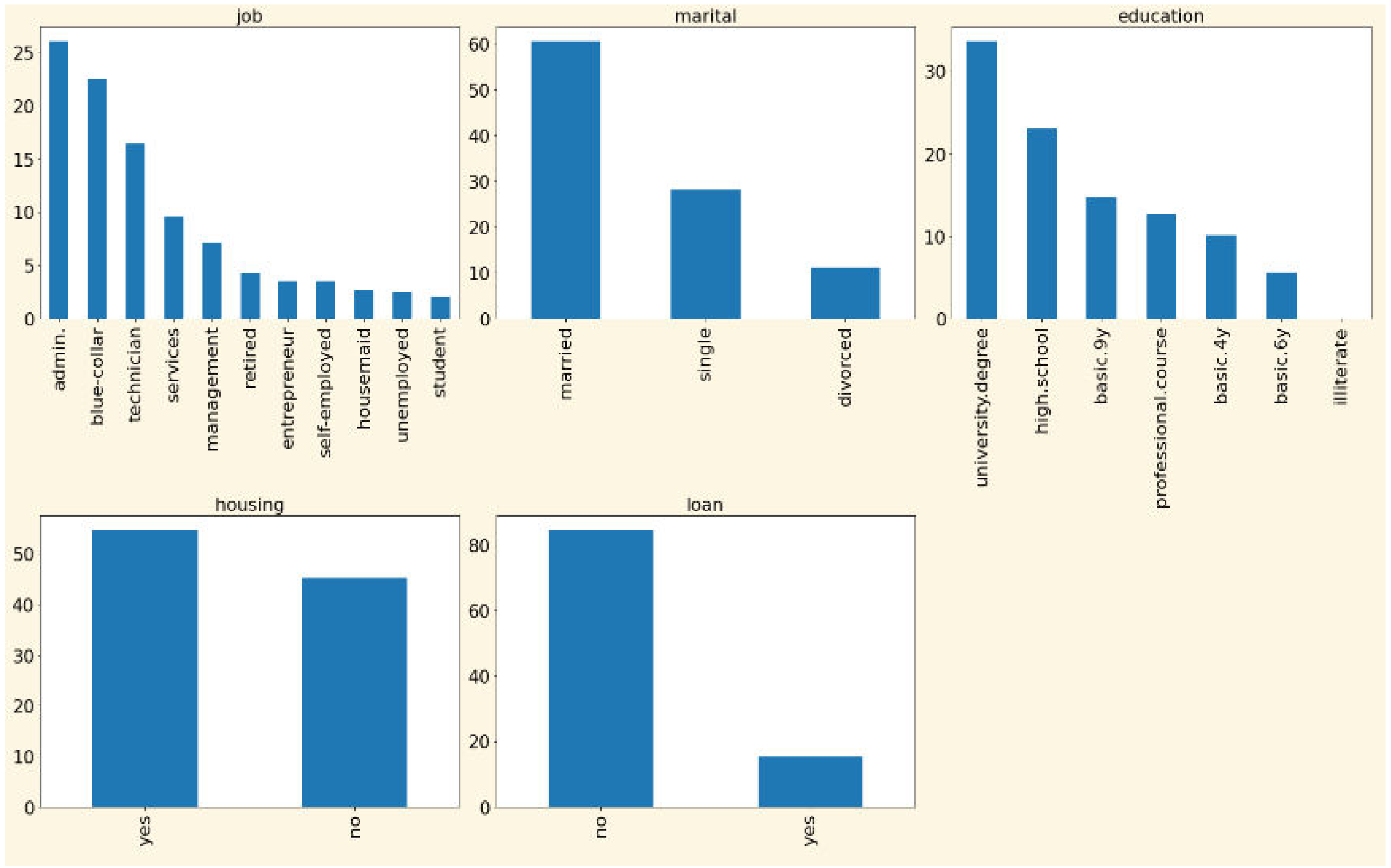


```
df_option1 = df.copy()
categorical = ['job', 'marital', 'education', 'housing', 'loan']

for i in categorical:
    df_option1[i] = df_option1[i].replace(to_replace = 'unknown', value = df_option1[i].mode().iloc[0]
```

We could see on the previous chart that in all the features values, except default, unknown values take quite a small percentage and adding it to the most frequent category won't affect imbalance between different categories badly. Since default feature has too many unknown values and this data is quite sensitive, it could be better to keep 'unknown' as a separate category.

After filling unknown variables with the most frequent category values we can see that change in values distribution isn't dramatic. However, this approach makes not very gentle assumption that all unknown values belong to a 'mode' category. It especially affects binary features with high imbalance, such as housing and loan.



Data cleansing and transformation

‘Unknown’ values

The second approach – building logistic regression to predict unknown values in each category

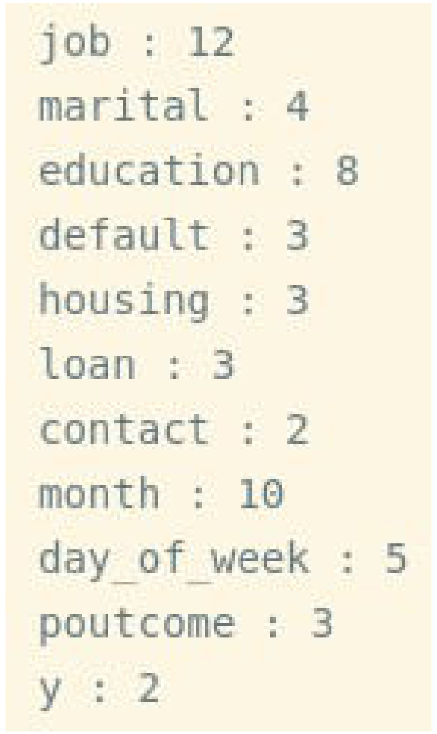
We could solve a classification problem for each column with unknown values and build a model to predict such values. Since we have six features containing unknown values, it's needed to develop six different models.

We will use all the rest variables in prediction cause if we use some individual set of features (which affects each output feature the most), we can lose a part of information, we need to use the same set of features for prediction to save the whole picture. We'll treat ‘unknown’ values in other categorical input features as a separate class.

First, we need to encode out input and output categorical features. I'll use get_dummies method and Label Encoder for this purpose respectivelly.

Further steps are the following:

- Divide the data into two parts. One part will have the present values of the column including the original output column, the other part will have the rows with the missing values.
- Divide the 1st part (present values) into cross-validation set for model selection.
- Train the models and test their metrics against the cross-validated data.
- Finally, with the model, predict the unknown values[4].



```
job : 12
marital : 4
education : 8
default : 3
housing : 3
loan : 3
contact : 2
month : 10
day_of_week : 5
poutcome : 3
y : 2
```

Number of classes in each category (including ‘unknown’)

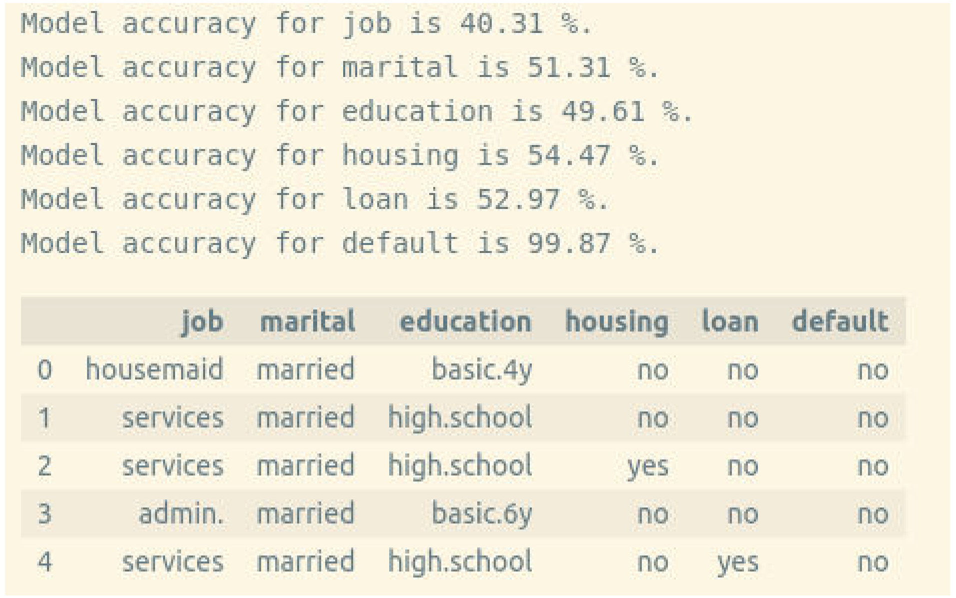
4. CHIRAG GOYAL. How to Handle Missing Values of Categorical Variables? Analytics Vidhya, URL

Data cleansing and transformation

‘Unknown’ values

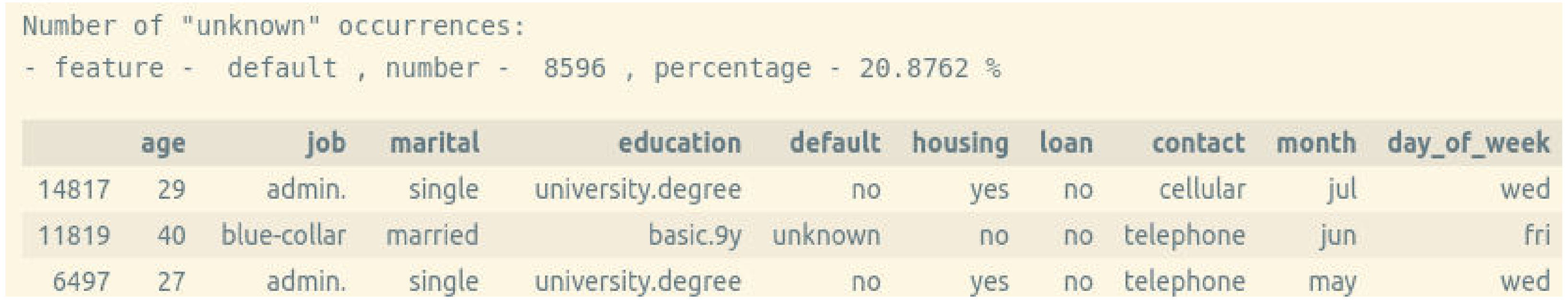
The second approach – building logistic regression to predict unknown values in each category

We can see that models' accuracy isn't high. It could be associated with big number of classes in some features, low relationship between input and output variables in general. Maybe the models could perform better with limited set of input features but, since all features are placed within the current data set, we find it more appropriate to include the same set of variables for all models. The only one model with high accuracy is a model for default feature. Since this feature values distribution is highly imbalanced, even adding class weight parameter to the model didn't solve the problem. So, this approach isn't acceptable for this feature and we again treat 'unknown' as a separate class here.



```
Model accuracy for job is 40.31 %.
Model accuracy for marital is 51.31 %.
Model accuracy for education is 49.61 %.
Model accuracy for housing is 54.47 %.
Model accuracy for loan is 52.97 %.
Model accuracy for default is 99.87 %.
```

	job	marital	education	housing	loan	default
0	housemaid	married	basic.4y	no	no	no
1	services	married	high.school	no	no	no
2	services	married	high.school	yes	no	no
3	admin.	married	basic.6y	no	no	no
4	services	married	high.school	no	yes	no



```
Number of "unknown" occurrences:
- feature -  default , number -  8596 , percentage - 20.8762 %
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week
14817	29	admin.	single	university.degree	no	yes	no	cellular	jul	wed
11819	40	blue-collar	married	basic.9y	unknown	no	no	telephone	jun	fri
6497	27	admin.	single	university.degree	no	yes	no	telephone	may	wed

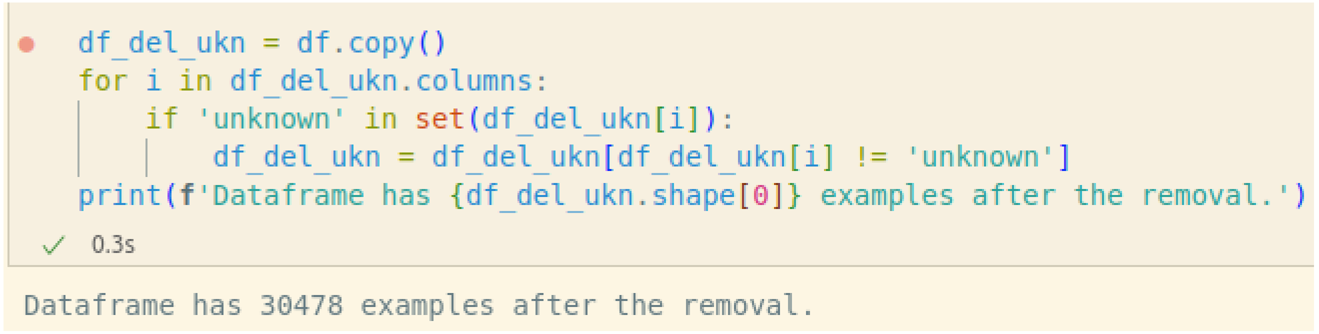
By and large, after filling the missing values ratio between classes within each feature is kept. Probably, in comparison with the first method this one fills missing values in a smoother way.

Data cleansing and transformation

‘Unknown’ values

The third approach – delete all the rows with unknown variables

Although it's not the best choice, we can perform removal of all the rows with unknown variables.



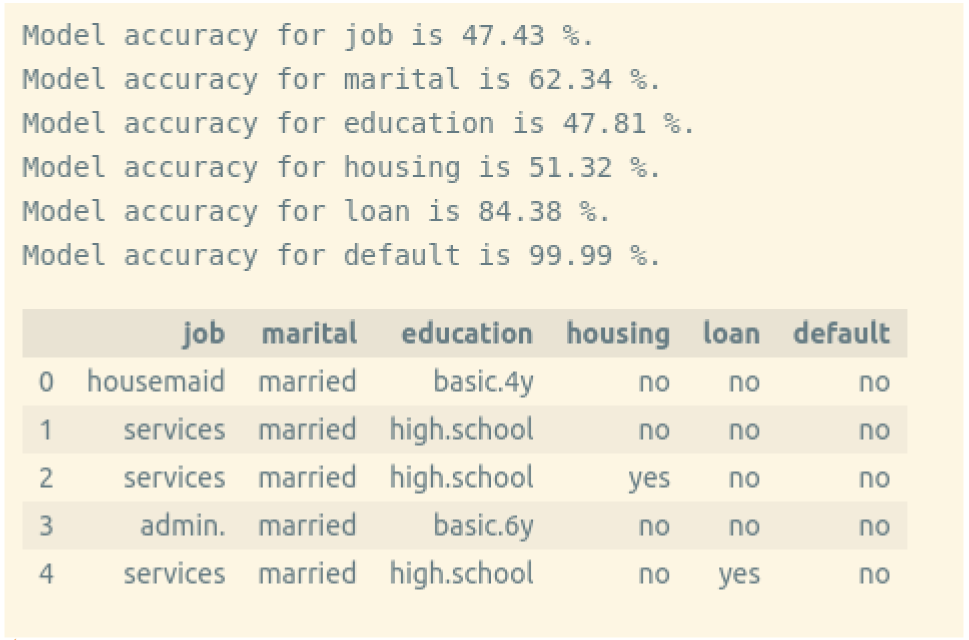
```
df_del_ukn = df.copy()
for i in df_del_ukn.columns:
    if 'unknown' in set(df_del_ukn[i]):
        df_del_ukn = df_del_ukn[df_del_ukn[i] != 'unknown']
print(f'Dataframe has {df_del_ukn.shape[0]} examples after the removal.')
```

✓ 0.3s

```
Dataframe has 30478 examples after the removal.
```

The fourth approach – use unsupervised learning (KNN)

In this approach, we use unsupervised machine learning, concretely K-Nearest-Neighbors algorithm. The idea is that we use a feature with unknown values as a target variable, other features as input variables and try to find the category in which each unknown value falls. We will use 10 nearest neighbors. The algorithm of preprocessing the data is similar to one we used in logistic regression, we just change the classifier.



```
Model accuracy for job is 47.43 %.
Model accuracy for marital is 62.34 %.
Model accuracy for education is 47.81 %.
Model accuracy for housing is 51.32 %.
Model accuracy for loan is 84.38 %.
Model accuracy for default is 99.99 %.
```

	job	marital	education	housing	loan	default
0	housemaid	married	basic.4y	no	no	no
1	services	married	high.school	no	no	no
2	services	married	high.school	yes	no	no
3	admin.	married	basic.6y	no	no	no
4	services	married	high.school	no	yes	no

We can see that the models perform even better than logistic regression models. However, for default feature we are not going to use this method again and keep 'unknown' class.

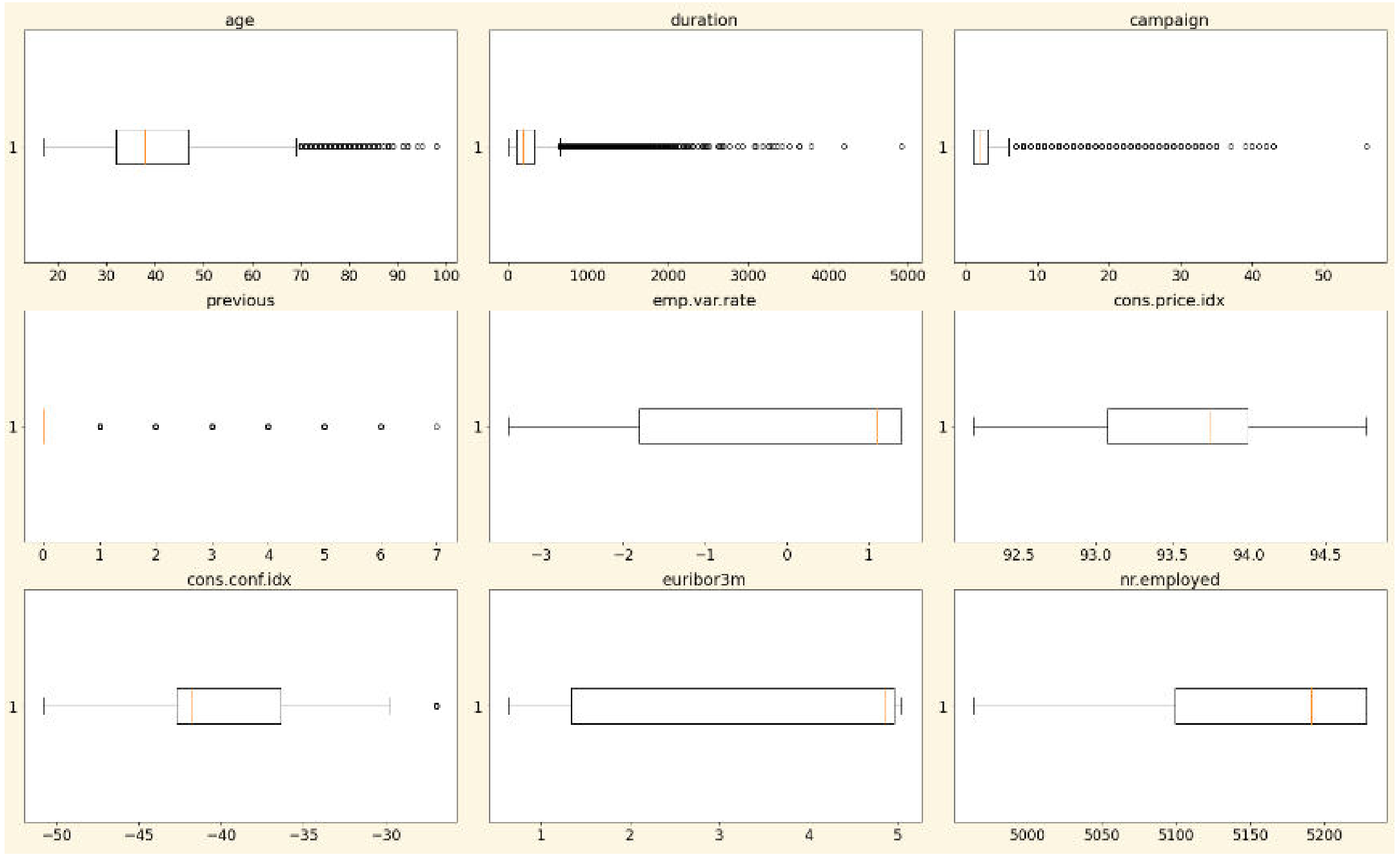
Data cleansing and transformation

Outliers

Since we detected outliers in some variables, we can either keep or remove them.

- Usually outliers cannot be removed without analyzing.
- We could keep the outliers in age variable cause removing the oldest clients will affect the customer base understanding.
- We could keep outliers in duration, campaign and previous cause they are just technical parameters and these outliers aren't related to specific customer groups.
- Also we could keep an outlier in consumer confidence index also for a reason that removing a customer with a higher confidence index will not display diversity of the customers.

Outliers graphs



However, there are several outlier removal approaches worth considering.

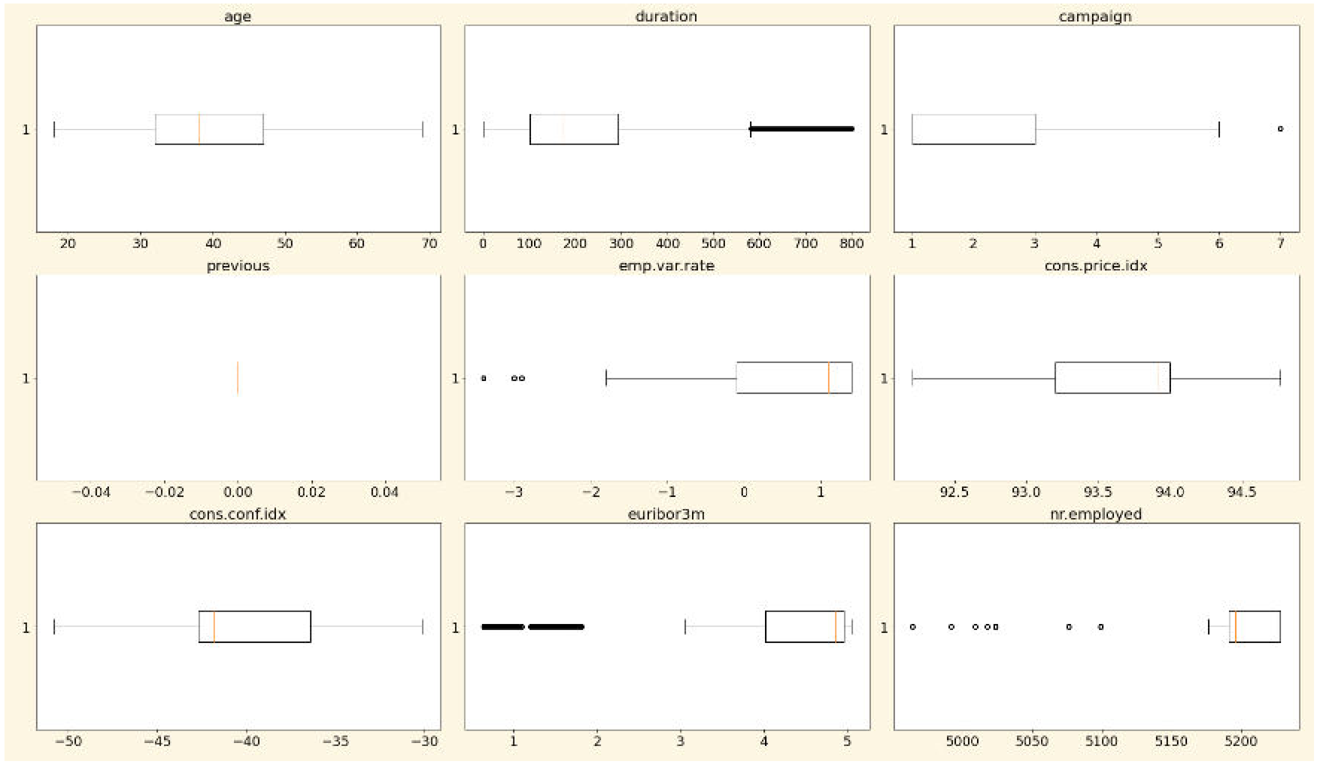
Data cleansing and transformation

Outliers

The first approach – use boxplot data

First option for detecting outliers is to visualize them. For this purpose we can use boxplot (i.e. whisker plot). We need to find indexes of rows with outliers (according to the graphs) and remove them. We look at the graphs and set limits for each column. After that we sequentially filter dataframe columns.

```
Need to remove 469 outliers from age feature.
Need to remove 1767 outliers from duration feature.
Need to remove 1777 outliers from campaign feature.
Need to remove 5625 outliers from previous feature.
Need to remove 0 outliers from emp.var.rate feature.
Need to remove 0 outliers from cons.price.idx feature.
Need to remove 714 outliers from cons.conf.idx feature.
Need to remove 0 outliers from euribor3m feature.
Need to remove 0 outliers from nr.employed feature.
Total number of duplicates to remove - 9440.
Number of rows after outliers removal - 31748.
Number of rows after outliers removal - 31737.
```



We can see that even after removal boxplots show that new arrays also have values considered as outliers. So, it's quite hard to determine, when we should stop removing outliers. If we keep removing outliers, we can lose some essential part of information.

Data cleansing and transformation

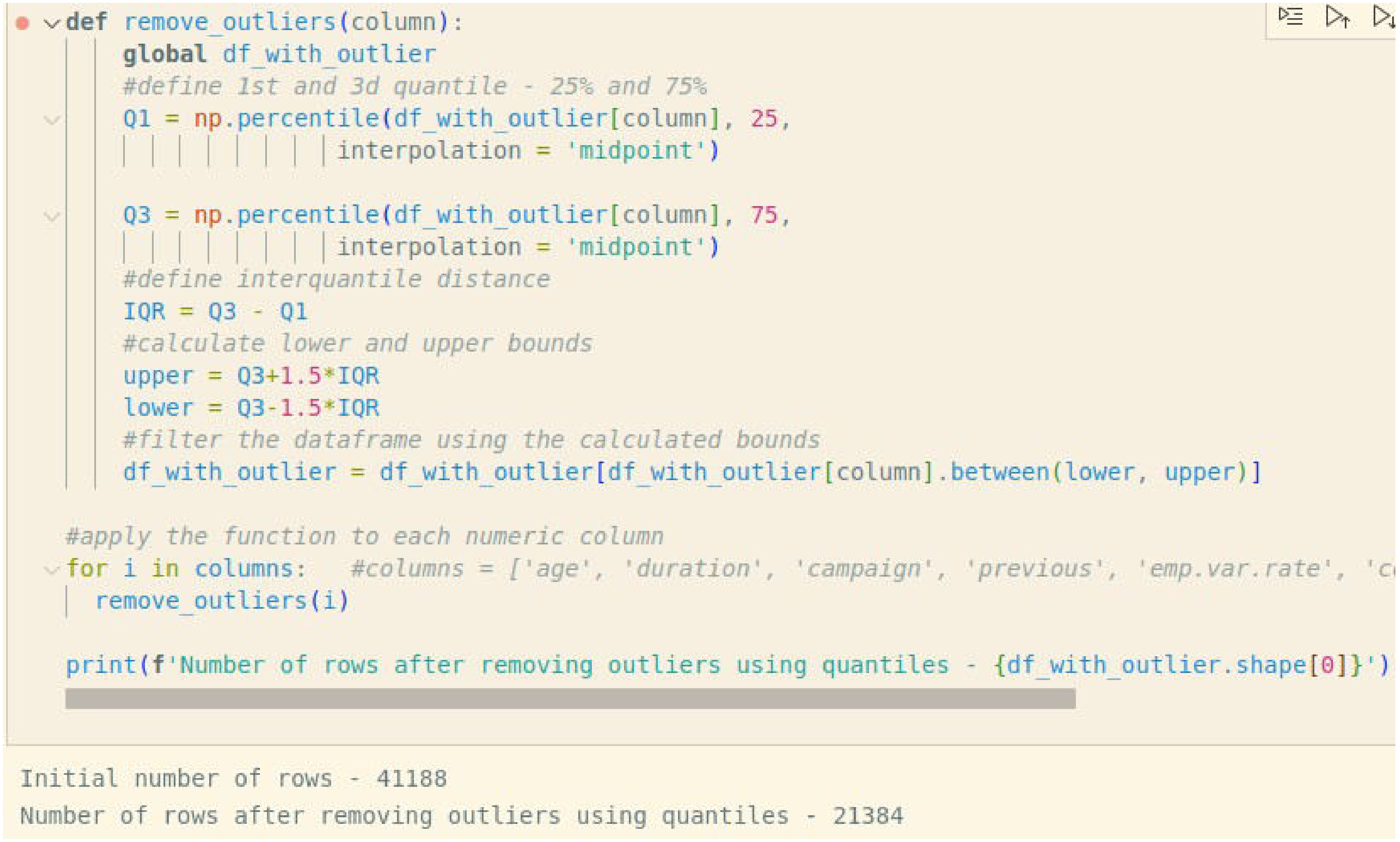
Outliers

The second approach – use quantiles

This method uses IQR (Inter Quartile Range) to find and remove outliers. The idea is to calculate upper and lower bounds and remove the values beyond them.

Although the method is considered very reliable, we can see that if we apply outliers removing to each numeric feature, we lost essential amount of data. Even if we use 2 IQR instead of 1.5 IQR, number of remaining examples almost doesn't change.

Function defined for using quantile approach



```
def remove_outliers(column):
    global df_with_outlier
    #define 1st and 3d quantile - 25% and 75%
    Q1 = np.percentile(df_with_outlier[column], 25,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df_with_outlier[column], 75,
                       interpolation = 'midpoint')
    #define interquantile distance
    IQR = Q3 - Q1
    #calculate lower and upper bounds
    upper = Q3+1.5*IQR
    lower = Q3-1.5*IQR
    #filter the dataframe using the calculated bounds
    df_with_outlier = df_with_outlier[df_with_outlier[column].between(lower, upper)]

#apply the function to each numeric column
for i in columns:   #columns = ['age', 'duration', 'campaign', 'previous', 'emp.var.rate', 'c
    remove_outliers(i)

print(f'Number of rows after removing outliers using quantiles - {df_with_outlier.shape[0]}')
```

```
Initial number of rows - 41188
Number of rows after removing outliers using quantiles - 21384
```

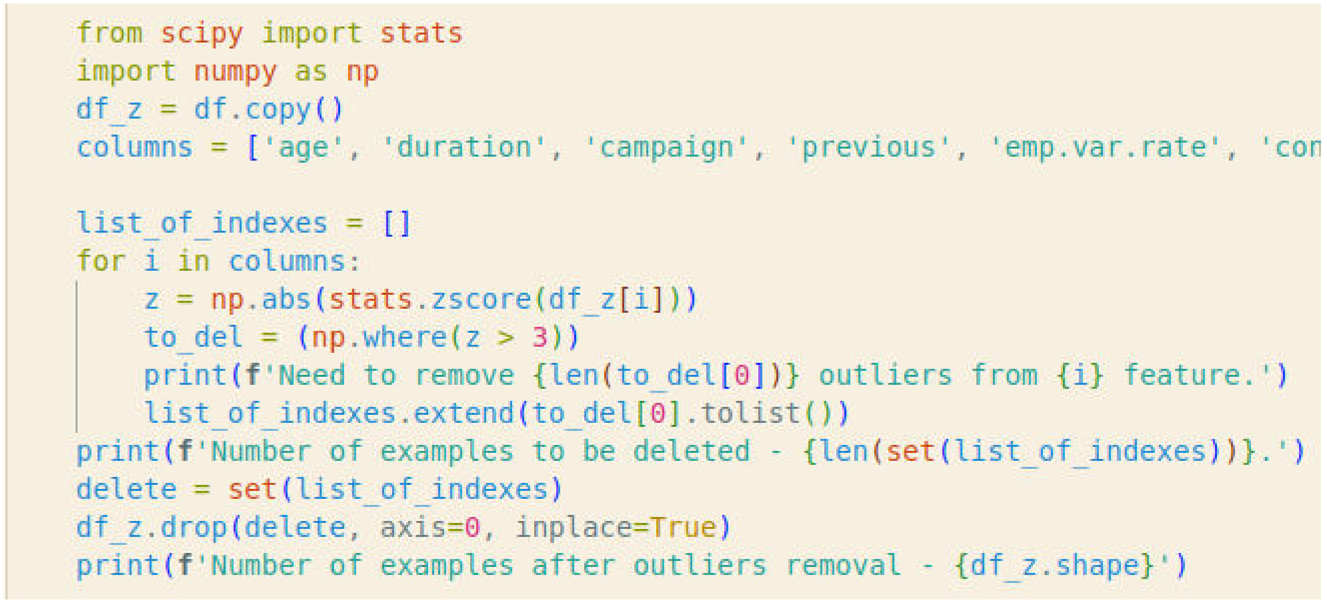
Data cleansing and transformation

Outliers

The third approach – use z-score

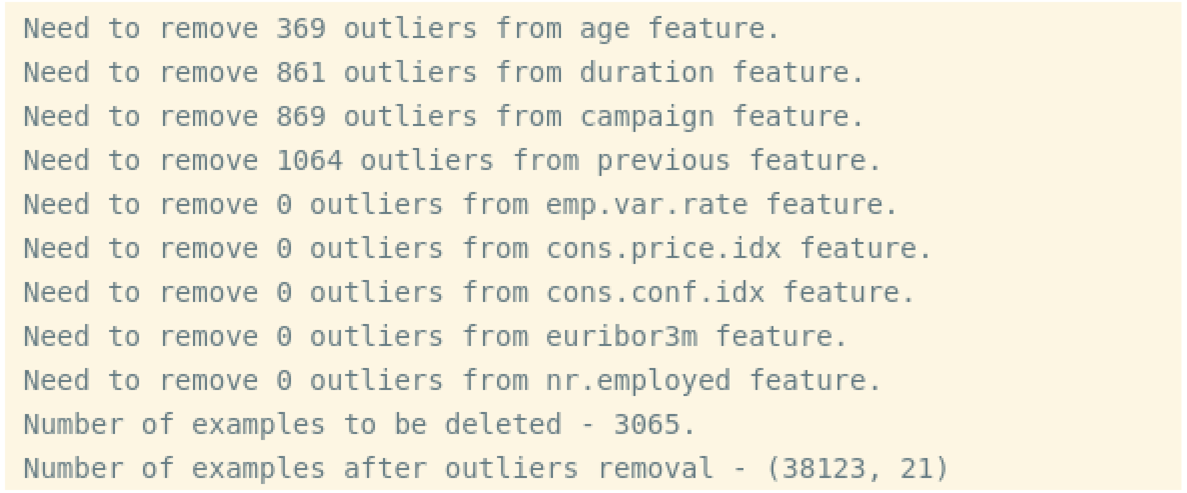
Z- Score is also called a standard score. This value/score helps to understand that how far is the data point from the mean. And after setting up a threshold value one can utilize z score values of data points to define the outliers.

- Zscore = (data_point -mean) / std. deviation
- To define an outlier threshold value is chosen which is generally 3.0. As 99.7% of the data points lie between +/- 3 standard deviation (using Gaussian Distribution approach).
- We can see that compared to previous methods, z-score removes quite a small part of data[5].



```
from scipy import stats
import numpy as np
df_z = df.copy()
columns = ['age', 'duration', 'campaign', 'previous', 'emp.var.rate', 'con

list_of_indexes = []
for i in columns:
    z = np.abs(stats.zscore(df_z[i]))
    to_del = (np.where(z > 3))
    print(f'Need to remove {len(to_del[0])} outliers from {i} feature.')
    list_of_indexes.extend(to_del[0].tolist())
print(f'Number of examples to be deleted - {len(set(list_of_indexes))}.')
delete = set(list_of_indexes)
df_z.drop(delete, axis=0, inplace=True)
print(f'Number of examples after outliers removal - {df_z.shape}')
```



```
Need to remove 369 outliers from age feature.
Need to remove 861 outliers from duration feature.
Need to remove 869 outliers from campaign feature.
Need to remove 1064 outliers from previous feature.
Need to remove 0 outliers from emp.var.rate feature.
Need to remove 0 outliers from cons.price.idx feature.
Need to remove 0 outliers from cons.conf.idx feature.
Need to remove 0 outliers from euribor3m feature.
Need to remove 0 outliers from nr.employed feature.
Number of examples to be deleted - 3065.
Number of examples after outliers removal - (38123, 21)
```

5. rajeshsharma7. Detect and Remove the Outliers using Python. Geeks for Geeks, URL

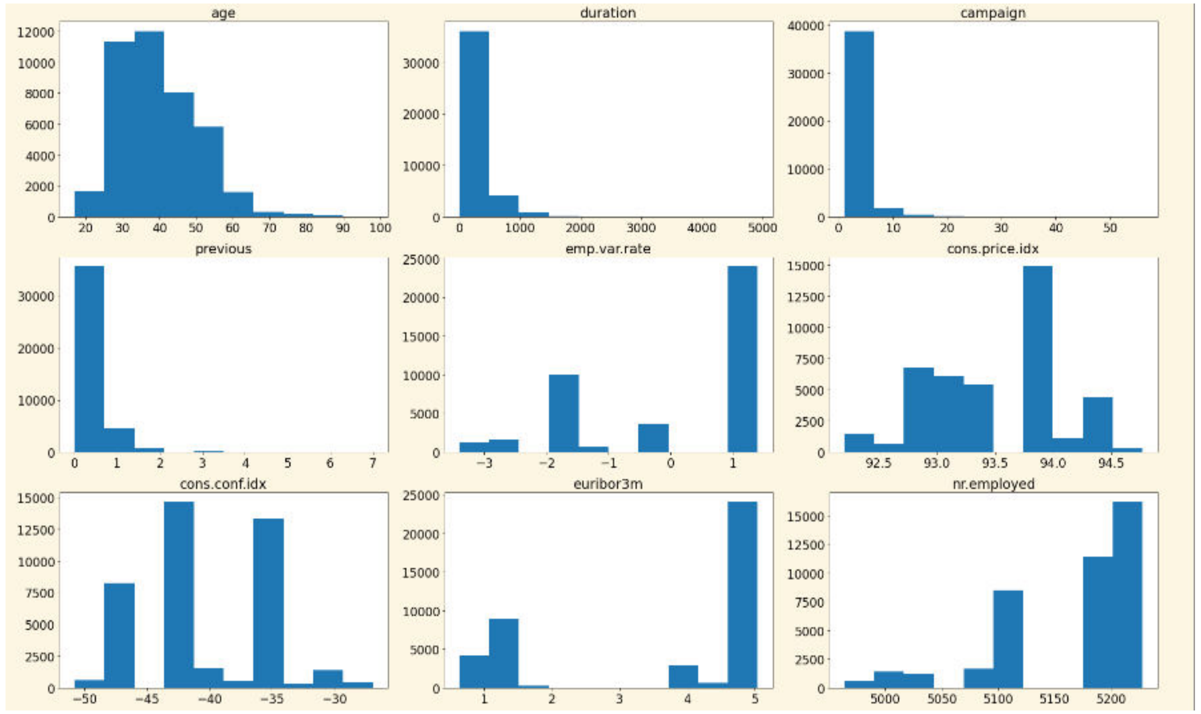
Data cleansing and transformation

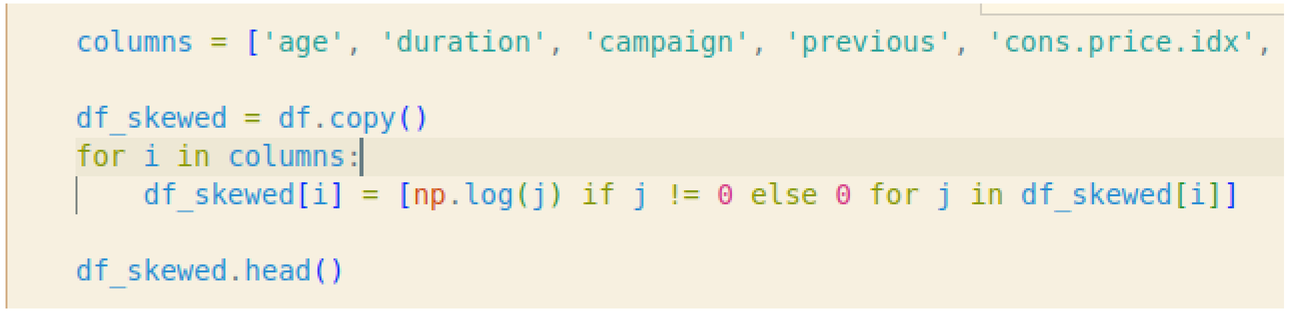
Skewed distribution

As for features distributions, each distribution is quite far from a bell shape the normal distribution has. We could transform some of them to normal by applying log function. However, it becomes harder to interpret the results, so we decided to keep the variables distributions in initial condition. Anyway, we can try and see how transformed distributions can look. We can apply this method only to features, where all the values are positive. Also, there is a limitation for features with zero values cause logarithm from zero is not defined.

- After applying this approach we cannot see that forms of distributions have become closer to a bell shape in most cases. Therefore, we'd rather not use this approach and keep the distributions in an initial condition.

Variables distributions





```
columns = ['age', 'duration', 'campaign', 'previous', 'cons.price.idx',

df_skewed = df.copy()
for i in columns:
    df_skewed[i] = [np.log(j) if j != 0 else 0 for j in df_skewed[i]]

df_skewed.head()
```

Bank Marketing (Campaign) - Group Project

Data Glacier

Your Deep Learning Partner

Thank You