EXPERIMENT-6

Name: Adwait Purao
UID: 2021300101
Div: B
Batch: B2

Problem Definition: To perform data preprocessing on the dataset

Theory:
Data preprocessing is the process of cleaning and preparing data for analysis. It involves steps such as:

- Identifying and handling missing values: Missing values can occur in data for a variety of reasons, such as errors in data collection or data entry. There are a number of ways to handle missing values, such as deleting rows with missing values, imputing missing values, or flagging missing values.
- Encoding categorical data: Categorical data is data that can be divided into a number of categories, such as gender, country, or product type. Categorical data must be encoded into numerical values before it can be used in most machine learning algorithms.
- Scaling numerical data: Numerical data should be scaled so that all features have a similar range of values. This helps to prevent any one feature from dominating the model.
- Handling outliers: Outliers are data points that are significantly different from the rest of the data. Outliers can occur due to errors in data collection or data entry, or they may be legitimate data points that represent rare events. Outliers can be handled by removing them from the data, or by using a robust machine learning algorithm that is not sensitive to outliers.

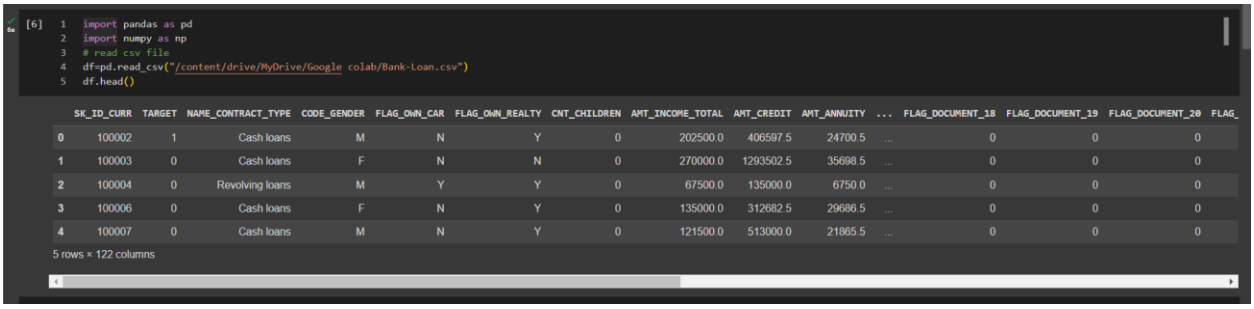
Data preprocessing is an important step in any machine learning project. By carefully cleaning and preparing your data, you can improve the accuracy and performance of your machine learning models.
Here are some additional tips for performing data preprocessing:

- Understand the data: Before you start preprocessing the data, it is important to understand the data and the problem that you are trying to solve. This will help you to identify the most important features and the best way to preprocess the data.
- Use visualization: Visualization can be a helpful tool for understanding the data and identifying problems with the data. For example, you can use histograms to identify outliers and scatter plots to identify relationships between features.

- Be consistent: When preprocessing the data, it is important to be consistent in your approach. This means using the same methods to handle missing values, encode categorical data, and scale numerical data for all of your data.
- Document your work: It is important to document the steps that you take when preprocessing the data. This will help you to reproduce your results and to troubleshoot any problems that may arise.

Output:

Step 1: Imports pandas and numpy libraries, reads a CSV file into a pandas DataFrame object, and displays the DataFrame object.

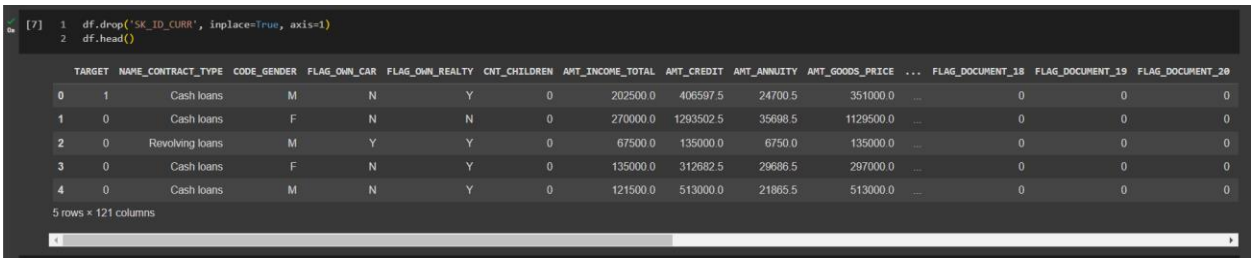


```
[6]  1  import pandas as pd
     2  import numpy as np
     3  # read csv file
     4  df=pd.read_csv("/content/drive/MyDrive/Google colab/Bank-Loan.csv")
     5  df.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20	FLAG_
0	100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	...	0	0	0	
1	100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	...	0	0	0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	...	0	0	0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	...	0	0	0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	...	0	0	0	

5 rows × 122 columns

Step 2: Drop the column SK_ID_CURR from the DataFrame df and then display the resulting DataFrame.

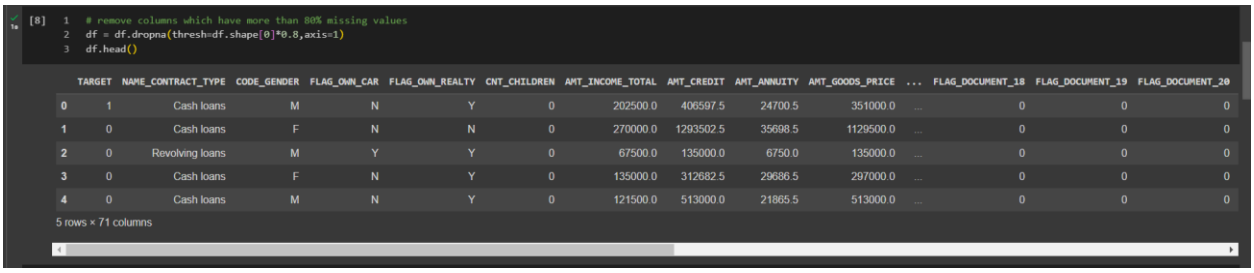


```
[7]  1  df.drop('SK_ID_CURR', inplace=True, axis=1)
     2  df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	...	0	0	0
1	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	...	0	0	0
2	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	...	0	0	0
3	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	...	0	0	0
4	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	...	0	0	0

5 rows × 121 columns

Step 3: Remove columns in the 'df' DataFrame that contain over 80% missing values.



```
[8]  1  # remove columns which have more than 80% missing values
     2  df = df.dropna(thresh=df.shape[0]*0.8,axis=1)
     3  df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	...	0	0	0
1	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	...	0	0	0
2	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	...	0	0	0
3	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	...	0	0	0
4	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	...	0	0	0

5 rows × 71 columns

Step 4: Process the columns with string values in the DataFrame df without using the scikit-learn library and then update the original DataFrame with the processed values.

```
[9]  #process columns with string values without sklearn, keep the original data frame
     df_string = df.select_dtypes(include=['object'])
     df_string = df_string.apply(lambda x: pd.factorize(x)[0])
     df[df_string.columns] = df_string
     df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1	0	0	0	0	0	202500.0	406597.5	24700.5	351000.0	...	0	0	0
1	0	0	1	0	1	0	270000.0	1293502.5	35698.5	1129500.0	...	0	0	0
2	0	1	0	1	0	0	67500.0	135000.0	6750.0	135000.0	...	0	0	0
3	0	0	1	0	0	0	135000.0	312682.5	29686.5	297000.0	...	0	0	0
4	0	0	0	0	0	0	121500.0	513000.0	21865.5	513000.0	...	0	0	0

5 rows × 71 columns

Step 5: Normalize the data in the DataFrame df using min-max scaling.

```
[10] # normalize the data without sklearn
     df = (df - df.min()) / (df.max() - df.min())
     df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1.0	0.0	0.0	0.0	0.0	0.0	0.001512	0.090287	0.090032	0.077441	...	0.0	0.0	0.0
1	0.0	0.0	0.5	0.0	1.0	0.0	0.002089	0.311736	0.132924	0.271605	...	0.0	0.0	0.0
2	0.0	1.0	0.0	1.0	0.0	0.0	0.000358	0.022472	0.020025	0.023569	...	0.0	0.0	0.0
3	0.0	0.0	0.5	0.0	0.0	0.0	0.000935	0.066837	0.109477	0.063973	...	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.000819	0.116854	0.078975	0.117845	...	0.0	0.0	0.0

5 rows × 71 columns

Step 6: Fill in all missing values in the df DataFrame with the mean of the corresponding column and then display the df DataFrame.

```
[11] df.fillna(df.mean(), inplace=True)
     df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1.0	0.0	0.0	0.0	0.0	0.0	0.001512	0.090287	0.090032	0.077441	...	0.0	0.0	0.0
1	0.0	0.0	0.5	0.0	1.0	0.0	0.002089	0.311736	0.132924	0.271605	...	0.0	0.0	0.0
2	0.0	1.0	0.0	1.0	0.0	0.0	0.000358	0.022472	0.020025	0.023569	...	0.0	0.0	0.0
3	0.0	0.0	0.5	0.0	0.0	0.0	0.000935	0.066837	0.109477	0.063973	...	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.000819	0.116854	0.078975	0.117845	...	0.0	0.0	0.0

5 rows × 71 columns

Step 7: Generate descriptive statistics for the DataFrame df including information such as the mean, standard deviation, minimum, and maximum values for each numeric column in the DataFrame.

```
[12] df.describe()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	F
count	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	307511.000000	...	307511.000000	307511.000000	
mean	0.080729	0.095213	0.329185	0.340108	0.306327	0.021950	0.001224	0.138334	0.099423	0.124179	...	0.008130	0.000595	
std	0.272419	0.293509	0.237142	0.473746	0.460968	0.038006	0.002027	0.100497	0.056525	0.092101	...	0.089798	0.024387	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000742	0.056180	0.058143	0.049383	...	0.000000	0.000000	
50%	0.000000	0.000000	0.500000	0.000000	0.000000	0.000000	0.001039	0.116987	0.090821	0.102132	...	0.000000	0.000000	
75%	0.000000	0.000000	0.500000	1.000000	1.000000	0.052632	0.001512	0.190674	0.128624	0.159371	...	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	

8 rows × 71 columns

Step 8: Saving the df DataFrame object to a CSV file named bank-loan-processed.csv and then reading the CSV file back into a new DataFrame object named new_df.

```
[13] 1 df.to_csv("bank-loan-processed.csv", index=False)
     2 new_df = pd.read_csv("bank-loan-processed.csv", sep=",")
     3 new_df.head()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	FLAG_DOCUMENT_20
0	1.0	0.0	0.0	0.0	0.0	0.0	0.001512	0.090287	0.090032	0.077441	...	0.0	0.0	0.0
1	0.0	0.0	0.5	0.0	1.0	0.0	0.002089	0.311736	0.132924	0.271605	...	0.0	0.0	0.0
2	0.0	1.0	0.0	1.0	0.0	0.0	0.000358	0.022472	0.020025	0.023569	...	0.0	0.0	0.0
3	0.0	0.0	0.5	0.0	0.0	0.0	0.000935	0.066837	0.109477	0.063973	...	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.000819	0.116854	0.078975	0.117845	...	0.0	0.0	0.0

5 rows × 71 columns

Step 9: Calculating the pairwise correlation coefficients between the columns of the new_df DataFrame like measuring the strength and direction of linear relationships between the variables. The output is a correlation matrix, where each cell represents the correlation coefficient between the corresponding pair of columns.

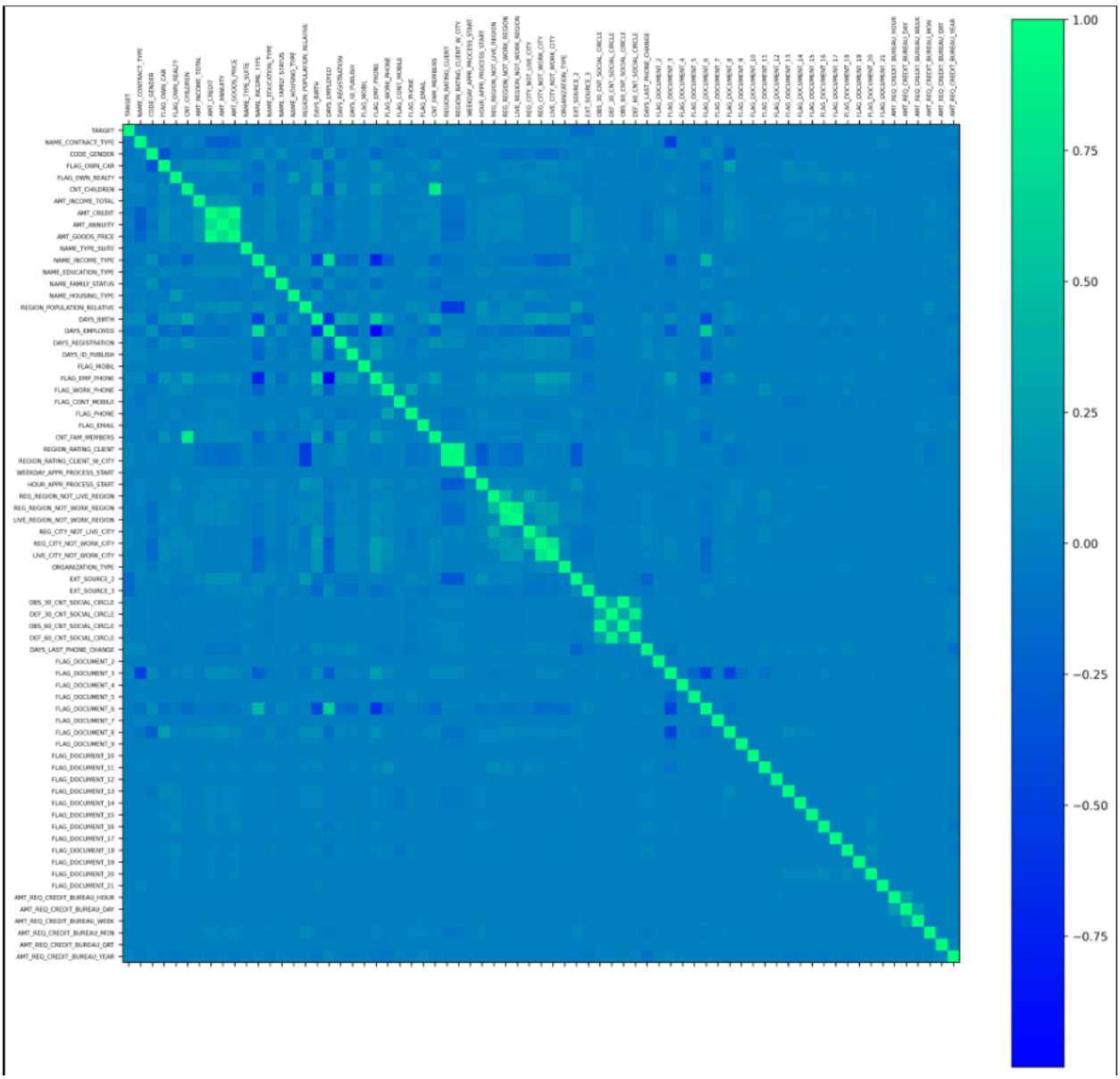
```
[14] 1 new_df.corr()
```

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	...	FLAG_DOCUMENT_18	FL
TARGET	1.000000	-0.030896	-0.054718	-0.021851	0.006148	0.019187	-0.003982	-0.030369	-0.012817	-0.039628	...	-0.007952	
NAME_CONTRACT_TYPE	-0.030896	1.000000	0.008867	0.004022	-0.067177	0.029998	-0.003531	-0.221648	-0.241543	-0.185049	...	-0.007530	
CODE_GENDER	-0.054718	0.008867	1.000000	-0.345815	-0.044396	-0.047367	-0.074724	-0.021614	-0.077002	-0.022512	...	-0.022213	
FLAG_OWN_CAR	-0.021851	0.004022	-0.345815	1.000000	0.002817	0.102023	0.083383	0.116225	0.141586	0.120282	...	-0.000097	
FLAG_OWN_REALTY	0.006148	-0.067177	-0.044396	0.002817	1.000000	0.002366	-0.002934	0.039270	0.005225	0.045521	...	0.087687	
...	...	...	...	...	...	...	...	...	...	...	...	...	
AMT_REQ_CREDIT_BUREAU_DAY	0.002464	-0.004732	-0.001061	0.000535	0.008645	-0.000342	0.002868	0.003964	0.002018	0.004374	...	0.012675	
AMT_REQ_CREDIT_BUREAU_WEEK	0.000718	-0.014144	0.001439	0.000227	-0.006972	-0.002277	0.002326	-0.001192	0.012815	-0.000940	...	-0.004428	
AMT_REQ_CREDIT_BUREAU_MON	-0.011356	-0.013286	-0.008260	0.019149	0.004180	-0.010101	0.024063	0.050934	0.036148	0.052755	...	-0.001493	
AMT_REQ_CREDIT_BUREAU_QRT	-0.001842	-0.020307	0.006920	-0.009291	-0.014414	-0.007324	0.004734	0.014896	0.009348	0.015364	...	-0.004891	
AMT_REQ_CREDIT_BUREAU_YEAR	0.018160	-0.048539	0.016969	-0.033988	-0.062927	-0.038834	0.011388	-0.045318	-0.010452	-0.047669	...	-0.045267	

71 rows × 71 columns

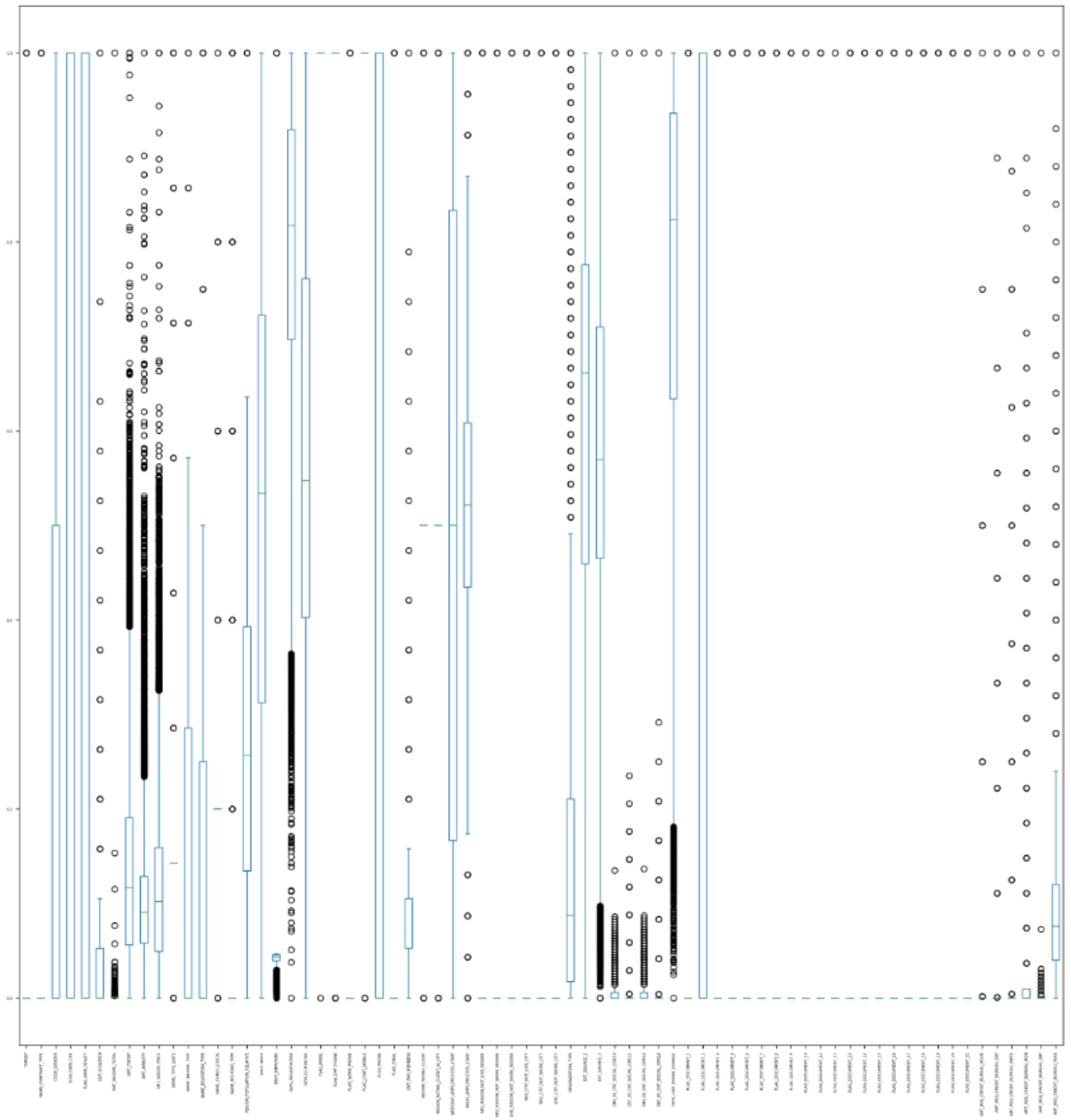
Step 10: Creating a heatmap visualization of the correlation matrix of the new_df DataFrame

```
# heatmap
import matplotlib.pyplot as plt
plt.matshow(new_df.corr(), cmap="winter")
plt.colorbar()
plt.xticks(np.arange(len(new_df.columns)), new_df.columns, rotation=90, fontsize=5)
plt.yticks(np.arange(len(new_df.columns)), new_df.columns, fontsize=5)
plt.rcParams["figure.figsize"] = [20,20]
plt.rcParams["figure.dpi"] = 300
plt.draw()
plt.savefig("heatmap.png", dpi=300, bbox_inches='tight')
plt.show()
```



Step 11: Creating a boxplot of the data in the new_df DataFrame object

```
1  new_df.plot.box(rot=90, fontsize=4)
```



Conclusion:

- From the data preprocessing experiment, I learned that it is important to clean and prepare data before using it in a machine learning model. This helps to improve the accuracy and performance of the model.
- I also learned that there are a number of different steps involved in data preprocessing, such as identifying and handling missing values, encoding categorical data, scaling numerical data, and handling outliers.
- It is important to understand the data and the problem that you are trying to solve before you start preprocessing the data.