my_analytics

December 16, 2021

References: Loan Approval Prediction

Step 1: Set state number

[ ]:
```
# replace 888 with the last three digits of your student id
# and then press CTRL + Enter
my_state_number = 673
```

Step 2: Read the data_set.csv

[ ]:
```
import pandas as pd
data = pd.read_csv("data_set.csv")

data.head()
```

[ ]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             360.0
1             4583             1508.0       128.0             360.0
2             3000                0.0        66.0             360.0
3             2583             2358.0       120.0             360.0
4             6000                0.0       141.0             360.0

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

Step 3: Sample the data randomly the data and save the dataframe as myNewData

[ ]:
```
myNewData = data.sample(frac =.90, replace = False, random_state =
↪my_state_number)
```

Start the Analytics using **myNewData** dataframe as the raw data note: your *myNewData* dataframe may be different from other students' *myNewData* dataframe

1 0.0 Import dependencies

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

2 1.0 Data Exploration

2.1 1.1 Dataset Details

2.1.1 1.1.1 Data Shape

[ ]:
```
# Start your codes
myNewData.shape
```

[ ]:
```
(553, 13)
```

The data has 553 rows (data) and 13 columns (features).

2.1.2 1.1.2 Data Information

[ ]:
```
myNewData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 553 entries, 461 to 66
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            553 non-null    object
 1   Gender             541 non-null    object
 2   Married            551 non-null    object
 3   Dependents         540 non-null    object
 4   Education          553 non-null    object
 5   Self_Employed      523 non-null    object
 6   ApplicantIncome    553 non-null    int64
```

```
 7   CoapplicantIncome  553 non-null    float64
 8   LoanAmount         535 non-null    float64
 9   Loan_Amount_Term   541 non-null    float64
 10  Credit_History     507 non-null    float64
 11  Property_Area      553 non-null    object
 12  Loan_Status        553 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 60.5+ KB
```

All columns are of the `object` datatype except for `ApplicantIncome` that has `int64` type, `CoapplicantIncome`, `LoanAmount`, `Loan_Amount_Term`, `Credit_History` of `float64` type. There are missing data where some columns do not have all 553 rows. The column names are inconsistent where some uses underscores and some using pure `CamelCase`.

2.1.3 1.1.3 Index Dropping

Dropping `Loan_ID` before any data analysis is conducted as it does not offer any real meaning.

[ ]:
```
myNewData = myNewData.drop(columns='Loan_ID')
```

2.1.4 1.1.4 Column Renaming

Renaming of data columns for a more consistent experience. `CamelCase` will be used.

[ ]:
```
myNewData = myNewData.rename(columns={'Self_Employed': 'SelfEmployed',␣
↪'Loan_Amount_Term': 'LoanAmountTerm', 'Credit_History': 'CreditHistory',␣
↪'Property_Area': 'PropertyArea', 'Loan_Status': 'LoanStatus'})
```

2.1.5 1.1.5 Nature of Data

[ ]:
```
myNewData.head()
```

[ ]:
```
    Gender Married Dependents     Education SelfEmployed  ApplicantIncome  \
461   Male     Yes         3+      Graduate           No             7740
597   Male      No        NaN      Graduate           No             2987
455   Male     Yes          2      Graduate           No             3859
6     Male     Yes          0  Not Graduate           No             2333
196   Male      No          0      Graduate           No             8333

     CoapplicantIncome  LoanAmount  LoanAmountTerm  CreditHistory  \
461                0.0       128.0           180.0            1.0
597                0.0        88.0           360.0            0.0
455                0.0        96.0           360.0            1.0
6               1516.0        95.0           360.0            1.0
196             3750.0       187.0           360.0            1.0

    PropertyArea LoanStatus
461        Urban          Y
```

```
597     Semiurban           N
455     Semiurban           Y
6           Urban           Y
196         Rural           Y
```

From the table above, it is known that numerical variables are `ApplicantIncome`, `CoapplicantIncome` and `LoanAmount`. The rest are categorical variables.

[ ]:
```
cat_data = (myNewData.drop(columns=['ApplicantIncome', 'CoapplicantIncome',␣
 ↪'LoanAmount'])).columns.values
num_data = (myNewData[['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']]).
 ↪columns.values
```

2.1.6 1.1.6 Data Description

[ ]:
```
myNewData.describe()
```

[ ]:
```
       ApplicantIncome  CoapplicantIncome  LoanAmount  LoanAmountTerm  \
count       553.000000         553.000000  535.000000      541.000000
mean       5479.320072        1577.437468  147.319626      341.656192
std        6357.052607        2916.184573   87.882458       65.922795
min         150.000000           0.000000    9.000000       12.000000
25%        2833.000000           0.000000  100.000000      360.000000
50%        3813.000000        1086.000000  128.000000      360.000000
75%        5780.000000        2250.000000  167.500000      360.000000
max       81000.000000       41667.000000  700.000000      480.000000

       CreditHistory
count     507.000000
mean        0.852071
std         0.355380
min         0.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         1.000000
```

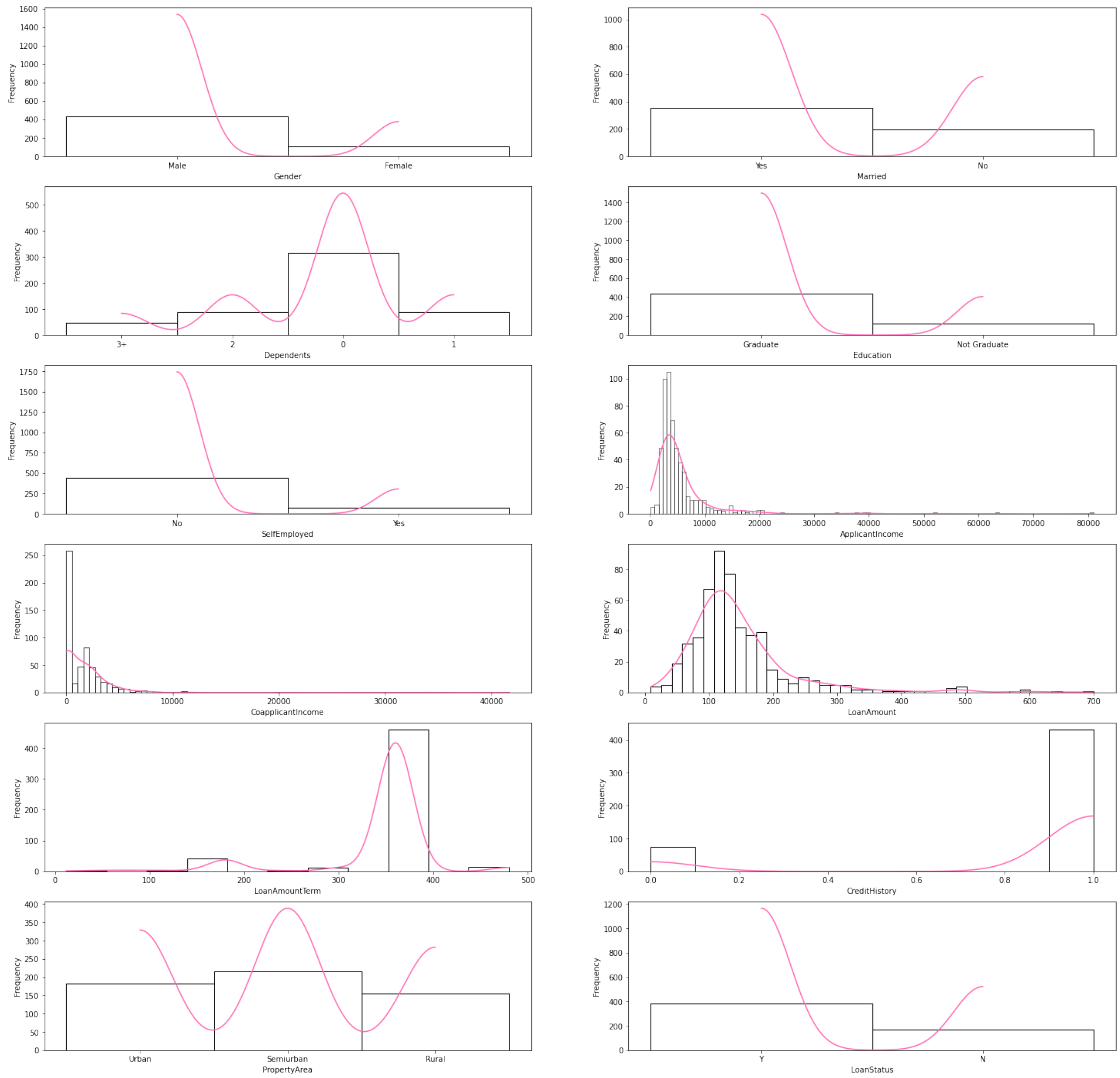
The statistics above describe numerical data columns. It can be known that the data values of these columns are spread across a large range. For example, `ApplicantIncome` has a maximum value of 81000 whereas `CreditHistory` has a maximum value of only 1. These data values need to be scaled for better model accuracy. Algorithms based on gradient descent such as linear regression and neural network perform better with scaled data because the data values will affect the step size of the gradient descent. The gradient descent will converge more quickly towards the minima when using data on a similar scale. Distance-based algorithms such as KNN and SVM are most affected by the range of data values as they calculate the distances between data points to find the similarity. The algorithms will stress more on features with data of a higher value, causing the model to be biased. Tree-based algorithms are quite insensitive to the data scales because the tree splits on a feature without taking other features into consideration. There are two main scaling techinques, that are normalisation and standardisation. Normalisation will transform all values to fit in the range of 0 and 1, also known as min-max scaling. Standardisation turns the mean value into 0 and the other values centred around the mean value will have a unit standard deviation. There is no particular range to this scaling method. Normalisation is used when the data distribution does not follow a Gaussian distribution, especially for KNN and neural networks, but it is very prone to outliers. Standardisation is helpful when the data follows a Gaussian distribution, but it is not necessarily so. Outliers in the data will not be affected by standardisation. The mean and standard deviation will be rescaled in such a way that they are very close to 0 and 1 respectively.

2.2 1.2 Data Exploration

2.2.1 1.2.1 Data Distribution

Exploring the dataset's general distribution pattern.

[ ]:
```
cols = list(myNewData.columns.values)

fig, ax = plt.subplots(6, 2, figsize=(25,25))
# fig.suptitle('Distribution of Dataset')

ax = ax.flatten() # ax is flattened from a 2D array to a 1D array, use ax.T.
↪flatten() to transpose if needed

for i in range(len(cols)):
    sns.histplot(data=myNewData, ax=ax[i], x=cols[i], kde=True,␣
↪color='hotpink', alpha=0)
    ax[i].set(xlabel=cols[i], ylabel='Frequency')
```



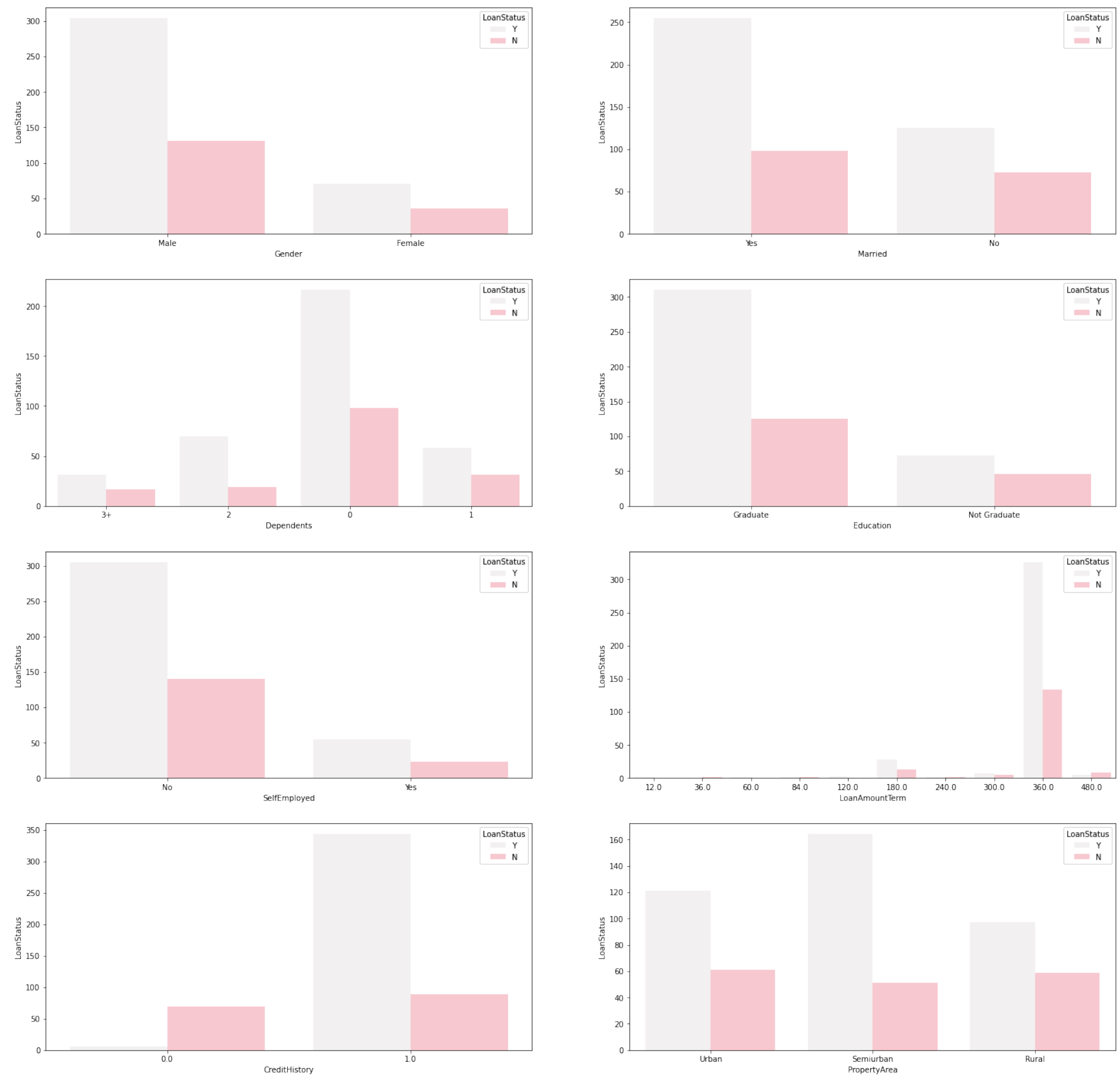
As shown from the graph above, there are five data distributions that do not follow a bell-shaped curve, which are all binary variables. All other data follows a normal distribution pattern. This analysis suggests that data standardisation instead of data normalisation might be applied onto the dataset.

2.2.2 1.2.1 Categorical Data Analysis

Analysing categorical variables. Exploring the relationships between the features and the target variable (`LoanStatus`).

[ ]:
```
fig, ax = plt.subplots(4, 2, figsize=(25,25))

ax = ax.flatten()

for i, v in enumerate(cat_data):
```

```
    if(v == 'LoanStatus'):
        continue
    sns.countplot(data=myNewData, ax=ax[i], x=v, hue='LoanStatus', color='pink', alpha=1)
    ax[i].set(xlabel=v, ylabel='LoanStatus')
```



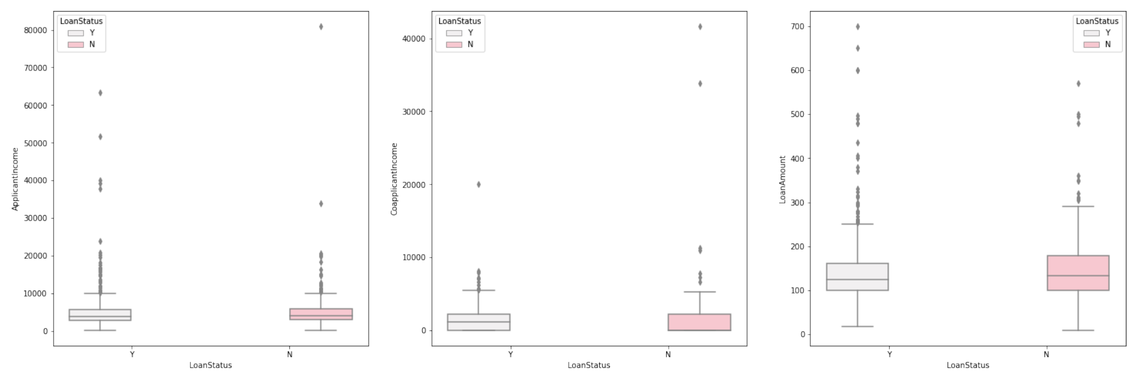
1. Around 3/5 applications have been approved.
2. There are approximately 3 times more male applicants than female.
3. Around 3/5 applicants are married. Married applicants are more likely to be granted loans.
4. Around 3/5 of the applicants have zero dependents. They are more likely to be granted loans.
5. Around 4/5 of the applicants are graduates, who are also more likely to be granted loans.
6. Less than 1/5 of the applicants are self-employed, who are not as likely to be granted loans as their counterparts.
7. Majority of the applicants have applied for a 30-year loan (360 months).

8. Applicants without credit history are unlikely to be granted loans.
9. There are more applicants from the semiurban property area and they are more likely to be granted loans.

2.2.3 1.2.2 Numerical Data Analysis

Analysing numerical columns.

[ ]:
```
fig, ax = plt.subplots(1, 3, figsize=(25,8))

ax = ax.flatten()

for i, v in enumerate(num_data):
    if(v == 'LoanStatus'):
        continue
    sns.boxplot(data=myNewData, ax=ax[i], x='LoanStatus', y=v,␣
 ↪hue='LoanStatus', color='pink')
    ax[i].set(xlabel='LoanStatus', ylabel=v)
```



It seems that these numerical variables do not have a significant relationship to `LoanStatus` as the boxes of `Y` and `N` do not have a significant difference between them.

3 2.0 Data Preprocessing

3.0.1 Null Value Check

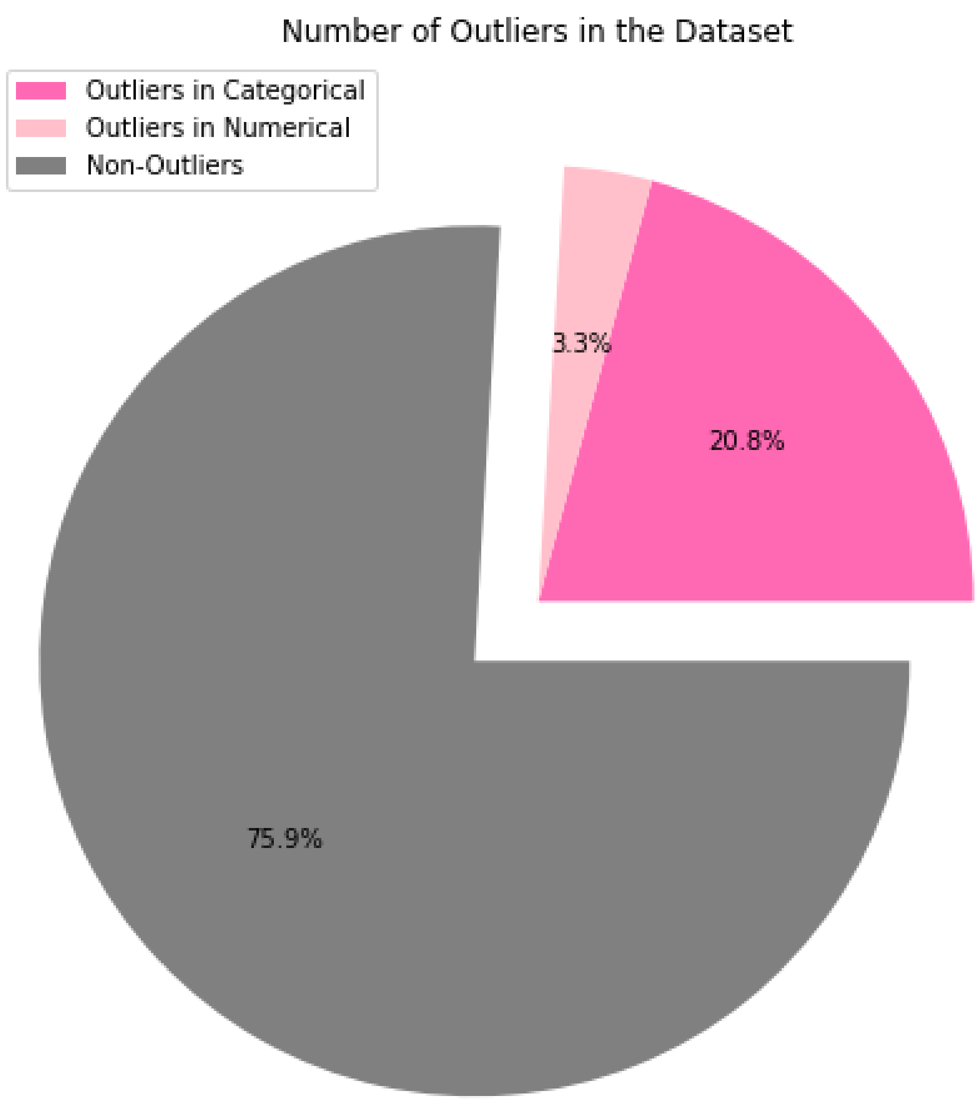
Checking for null values.

3.0.2 Data Encoding

Encoding categorical data values to be numerical for easier further processing. Label encoding is used for binary categorical columns such as `gender`, and these feature columns are renamed in such a way that 0 is false and 1 is true for the column to replicate the effect of one hot encoding. `PropertyArea` and `Dependents` columns will be encoded using one-hot encoding.

```
[ ]: binary = ['Gender', 'Married', 'Education', 'SelfEmployed', 'CreditHistory',␣
      ↪'LoanStatus']

     label_encoder = LabelEncoder()

     for _, v in enumerate(binary):
         myNewData[v] = label_encoder.fit_transform(myNewData[v])
```

```
[ ]: myNewData2 = myNewData.rename(columns={'Gender': 'Male', 'Education':␣
      ↪'Graduate', 'LoanStatus': 'LoanApproved'})
     myNewData2
```



```
[ ]: onehot = ['Dependents', 'PropertyArea']

     myNewData3 = myNewData2

     onehot_encoder = OneHotEncoder(sparse=False, drop=None)

     for _, v in enumerate(onehot):
         tmp_df = pd.DataFrame(onehot_encoder.fit_transform(np.reshape(myNewData2[v].
      ↪values, (-1, 1))))
         myNewData3 = myNewData3.drop(columns=[v])
         myNewData3 = myNewData3.join(tmp_df, rsuffix='_' + v)

     myNewData3
```

```
Outliers in Categorical: 115
Outliers in Numerical: 18
Total Outliers: 133
```

```
[ ]: df_encoded = myNewData.replace({'Male': 0, 'Female': 0, 'No': 0, 'Yes': 1, 'Not␣
      ↪Graduate': 0, 'Graduate': 1, 'Rural': 0, 'Semiurban': 1, 'Urban': 2, 'N': 0,␣
      ↪'Y': 1, '0': 0, '1': 1, '2': 2, '3+': 3})
```

3.0.3 Simple Imputation

There are a total of 115 rows of data which is 20.8% percent of the data that are categorical in nature. These rows will be imputed using the `SimpleImputer` class by using the data with the highest frequency. On the other hand, numerical data will be imputed with the mean of the column.

```
[ ]: for _, v in enumerate(cat_data):
         myNewData[v] = myNewData[v].fillna(myNewData[v].value_counts().index[0]) #␣
      ↪Fill categorical data with the highest frequency

     for _, v in enumerate(num_data):
         myNewData[v] = myNewData[v].fillna(myNewData[v].mean()) # Fill numerical␣
      ↪data with column mean

     myNewData.isnull().sum()
```

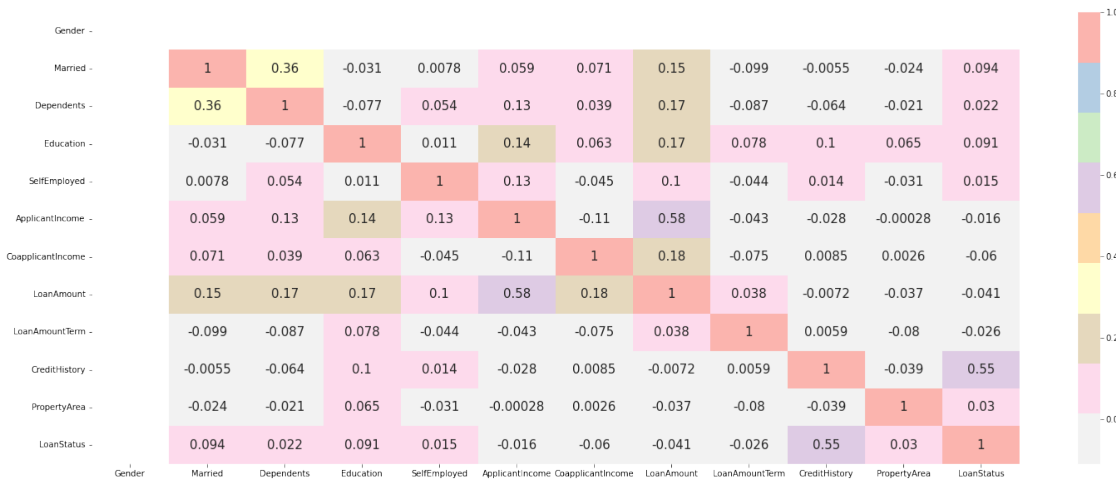
```
[ ]: Gender              0
     Married             0
     Dependents          0
     Education           0
     SelfEmployed        0
     ApplicantIncome     0
     CoapplicantIncome   0
     LoanAmount          0
```

```
LoanAmountTerm       0
CreditHistory        0
PropertyArea         0
LoanStatus           0
dtype: int64
```

Heatmap

```
[ ]: plt.figure(figsize=(25, 10))
     sns.heatmap(df_encoded.corr(), cmap='Pastel1_r', annot=True, annot_kws={'size':␣
      ↪15})
```

```
[ ]: <AxesSubplot:>
```



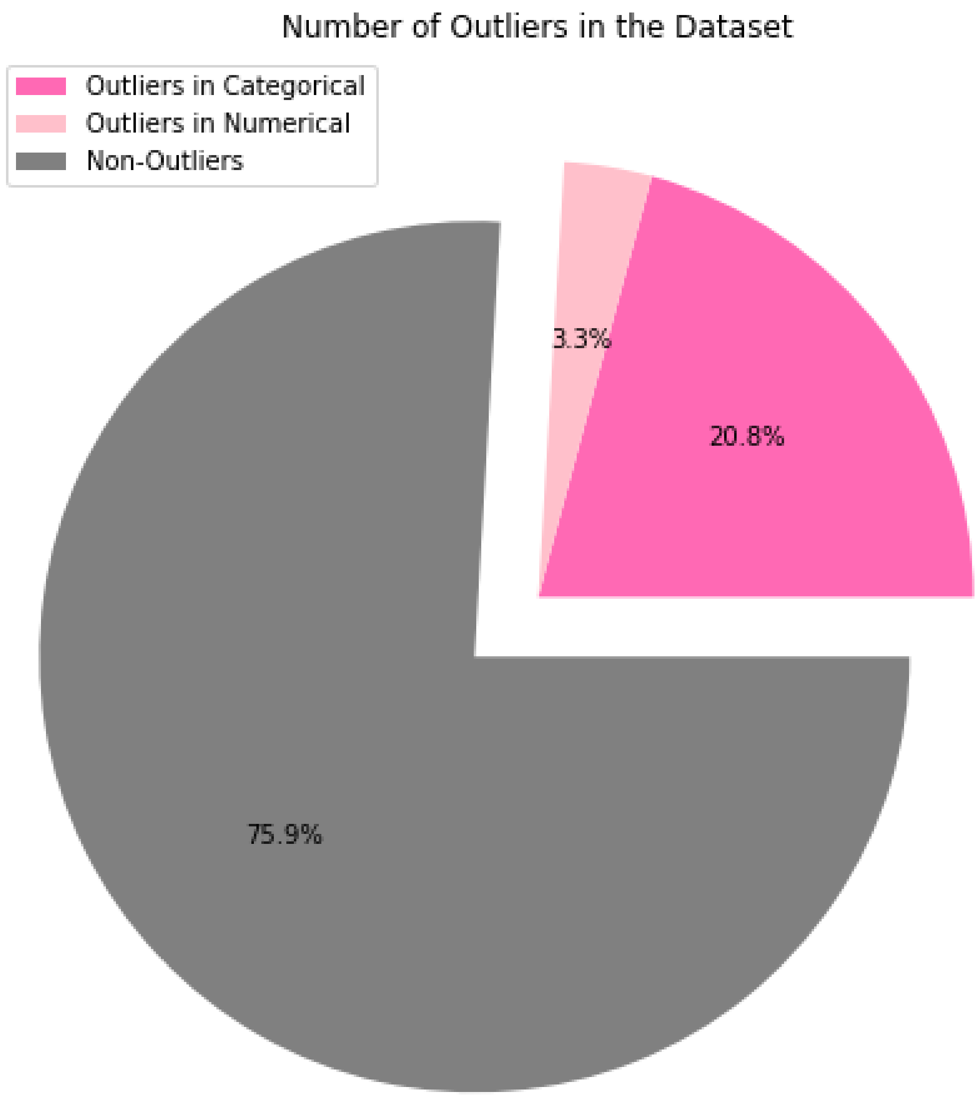

```
[ ]: myNewData.isnull().sum()
```

```
[ ]:      Male  Married Dependents  Graduate  SelfEmployed  ApplicantIncome  \
     461     1        1         3+         0             0             7740
     597     1        0          0         0             0             2987
     455     1        1          2         0             0             3859
     6       1        1          0         1             0             2333
     196     1        0          0         0             0             8333
     ..    …        …          …        …             …                …
     227     1        1          2         0             0             6250
     566     1        0          0         0             0             3333
     229     1        0          0         0             1             6400
     408     1        1          1         0             0             8300
     66      1        0          0         1             0             3200

          CoapplicantIncome  LoanAmount  LoanAmountTerm  CreditHistory  \
     461                0.0       128.0           180.0              1
```

```
597            0.0     88.0        360.0              0
455            0.0     96.0        360.0              1
6           1516.0     95.0        360.0              1
196         3750.0    187.0        360.0              1
..             …         …          …             …
227         1695.0    210.0        360.0              1
566            0.0     70.0        360.0              1
229            0.0    200.0        360.0              1
408            0.0    152.0        300.0              0
66          2254.0    126.0        180.0              0

    PropertyArea  LoanApproved
461        Urban             1
597    Semiurban             0
455    Semiurban             1
6          Urban             1
196        Rural             1
..           …           …
227    Semiurban             1
566        Urban             1
229        Rural             1
408    Semiurban             0
66         Urban             0

[553 rows x 12 columns]
```

There are null values in the columns `Gender`, `Married`, `Dependents`, `SelfEmployed`, `LoanAmount`, `LoanAmountTerm`, `CreditHistory`.

[ ]:
```python
x = ['Outliers in Categorical', 'Outliers in Numerical', 'Non-Outliers']
y = [myNewData[cat_data].isnull().values.sum(), myNewData[num_data].isnull().
 ↪values.sum(), myNewData.shape[0] - myNewData.isnull().values.sum()]

fig = plt.gcf()
fig.set_size_inches(8, 8)
plt.pie(y, colors=['hotpink', 'pink', 'gray'], explode=[0, 0, 0.2], autopct='%1.
 ↪1f%%')
plt.legend(x)
plt.title("Number of Outliers in the Dataset")
plt.show()
```




```
[ ]: print('Outliers in Categorical:', myNewData[cat_data].isnull().values.sum())
     print('Outliers in Numerical:', myNewData[num_data].isnull().values.sum())
     print('Total Outliers:', myNewData.isnull().values.sum())
```

```
Outliers in Categorical: 115
Outliers in Numerical: 18
Total Outliers: 133
```

The total number of rows with missing data is 133, which is almost 1/5 of the dataset. Removing all these data might cause biasness in the model as the remaining dataset will be quite small. Therefore, the treatment approach will be taken.

4 2.0 Data Preprocessing

Dealing with missing values.

[ ]:
```
df_filled = df_encoded.fillna(df_encoded.mean())
df_filled.isnull().any()
```

[ ]:
```
Gender               False
Married              False
Dependents           False
Education            False
SelfEmployed         False
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount           False
LoanAmountTerm       False
CreditHistory        False
PropertyArea         False
LoanStatus           False
dtype: bool
```

The numerical columns are filled with the mean of the columns from the original data before dropping any gender columns. Using mean is due to that it best represents the average. Calculating mean from the original data before dropping rows is because those dropped data are still valid and valuable data that contributes meaningfully to the mean.

[ ]:
```
# df = df_filled.drop()
df = df_filled
```

[ ]:
```
X_df = df.drop('LoanStatus', axis=1)
y_df = df['LoanStatus']
```

Split features and target label.

[ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size = 0.
↪3, random_state=0)
```

Split dataset into training and test sets using the 70:30 ratio for training:testing.

[ ]:
```
scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Normalising the data to improve model accuracy.

5 Model Selection